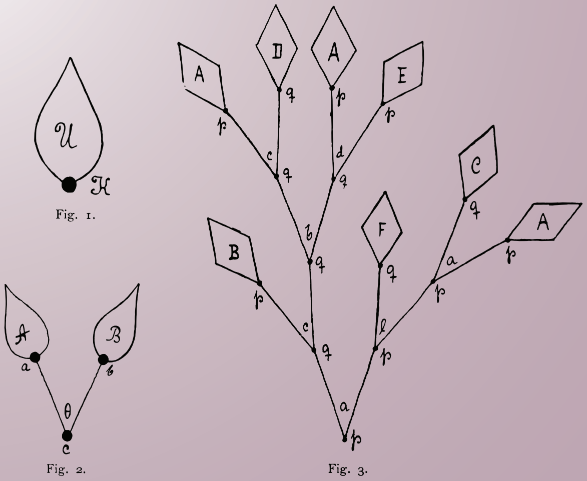
T·L·G
Texts in Logic and Games
Volume 2

Logic and Automata

History and Perspectives

EDITED BY
JÖRG FLUM, ERICH GRÄDEL AND THOMAS WILKE



AMSTERDAM UNIVERSITY PRESS

LOGIC AND AUTOMATA

T·L·G
Texts in Logic and Games
Volume 2

Logic and Automata

History and Perspectives

EDITED BY
JÖRG FLUM
ERICH GRÄDEL
THOMAS WILKE

Texts in Logic and Games
Volume 2

AMSTERDAM UNIVERSITY PRESS

Cover design: Maedium, Utrecht

ISBN 978 90 5356 576 6
NUR 918

Table of Contents

Preface

Mathematical logic and automata theory are two scientific disciplines with a close relationship that is not only fundamental for many theoretical results but also forms the basis of a coherent methodology for the verification and synthesis of computing systems. Although both automata theory and mathematical logic look back to a much longer history, they have come together in the 1960s through the fundamental work of Büchi, Elgot, Rabin and others who showed the expressive equivalence of automata with logical systems such as monadic second-order logic on finite and infinite words and trees. This allowed the handling of specifications (where global system properties are stated) and implementations (which involve the definition of the local steps in order to satisfy the global goals laid out in the specification) in a single framework. Moreover this framework offered algorithmic procedures for essential questions such as the consistency of the specifications or the correctness of implementations. Through the methodology of model-checking the connection between automata theory and logic has indeed become the basis of efficient verification methods with industrial scale applications.

Wolfgang Thomas is one of the leading scientists in logic and automata theory. He has shaped this scientific area, not only through many deep and beautiful results, but also through his ability to bring together different research threads, to provide a convincing synthesis of them, and to point out new and promising directions. For a whole generation of scientists in the field, including most of the collaborators of this volume, his tutorials and surveys on automata theory, language theory and logic, his activities as a teacher, and his lucid contributions at conferences and in informal discussions, have been extremely influential. We now take the occasion of the 60th birthday of Wolfgang Thomas to present a *tour d'horizon* on automata theory and logic. The twenty papers assembled in this volume, written by experts of the respective area upon invitation by the editors, cover many different facets of logic and automata theory. They emphasize the connections of automata theory and logic to other disciplines such as complexity theory, games, algorithms, and semigroup theory and stress their importance for modern applications in computer science such as the synthesis and verification of reactive systems. The volume puts modern scientific developments into a historical perspective, and shows how they are rooted in more than forty years of automata theory and mathematical logic. Perhaps even more importantly, the authors present and discuss current perspectives of automata and logic based methodologies in different areas of computer science.

The cover picture of this volume is taken from an old paper by the Norwegian mathematician Axel Thue (1863–1922)[1] which is historically quite remarkable. While Thue's work on word rewriting and combinatorics of words has been widely acknowledged, and notions such as Thue systems or Thue-Morse sequences are familiar to most computer scientists, it had gone unnoticed for a long time that Thue also, in the above mentioned paper, introduced the concept of trees into logic, and was apparently the first to discuss problems such as tree rewriting and the word problem for tree identities, and to use notions such as the Church-Rosser property, confluence, and termination. Only recently, Magnus Steinby and Wolfgang Thomas brought Thue's 1910 paper again to the attention of the scientific community and pointed out its historical importance.[2]

Freiburg, Aachen & Kiel J.F. E.G. T.W.

[1] Axel Thue, Die Lösung eines Spezialfalles eines generellen logischen Problems, Kra. Vidensk. Selsk. Skrifter. I. Mat.-Nat. Kl., Christiana 1910, Nr. 8.

[2] M. Steinby and W. Thomas. Trees and term rewriting in 1910: On a paper by Axel Thue. Bulletin of the European Association for Theoretical Computer Science, 72:256-269, 2000.

On the topological complexity of tree languages

André Arnold [1]
Jacques Duparc[2]
Filip Murlak[3]
Damian Niwiński[3]

[1] Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux 1
351 cours de la Libération
33405 Talence cedex, France
`andre.arnold@club-internet.fr`

[2] École des Hautes Études Commerciales
Université de Lausanne
1015 Dorigny, Switzerland
`Jacques.Duparc@unil.ch`

[3] Institute of Informatics
Uniwersytet Warszawski
Banacha 2
02-097 Warszawa, Poland
{`fmurlak,niwinski`}`@mimuw.edu.pl`

Abstract

The article surveys recent results in the study of topological complexity of recognizable tree languages. Emphasis is put on the relation between topological hierarchies, like the Borel hierarchy or the Wadge hierarchy, and the hierarchies resulting from the structure of automata, as the Rabin-Mostowski index hierarchy. The topological complexity of recognizable tree languages is seen as an evidence of their structural complexity, which also induces the computational complexity of the verification problems related to automata, as the non-emptiness problem. Indeed, the topological aspect can be seen as a rudiment of the infinite computation complexity theory.

1 Introduction

Since the discovery of irrational numbers, the issue of impossibility has been one of the driving forces in mathematics. Computer science brings forward a related problem, that of difficulty. The mathematical expression of difficulty is complexity, the concept which affects virtually all subjects in computing science, taking on various contents in various contexts.

In this paper we focus on infinite computations, and more specifically on finite-state recognition of infinite trees. It is clearly not a topic of classical complexity theory which confines itself to computable functions and relations over integers or words, and measures their complexity by the—supposedly finite—time and space used in computation. However, infinite computations are meaningful in computer science, as an abstraction of some real phenomena as, e.g., interaction between an open system and its environment. The finite and infinite computations could be reconciliated in the framework of descriptive complexity, which measures difficulty by the amount of logic necessary to describe a given property of objects, were they finite or infinite. However the automata theory has also developed its own complexity measures which refer explicitly to the dynamics of infinite computations.

From yet another perspective, infinite words (or trees) are roughly the real numbers, equipped with their usual metric. Classification of functions and relations over reals was an issue in mathematics long before the birth of computer science. The history goes back to Émil Borel and the circle of semi-intuitionists around 1900, who attempted to restrict the mathematical universe to mentally constructible (*définissables*) objects, rejecting set-theoretic pathologies as unnecessary. This program was subsequently challenged by a discovery made by Mikhail Suslin in 1917: the projection of a Borel relation may not be Borel anymore (see [12], but also [1] for a brief introduction to definability theory). It is an intriguing fact that this phenomenon is also of interest in automata theory. For example, the set of trees recognized by a finite automaton may be non-Borel, even though the criterion for a path being successful is so. One consequence is that the Büchi acceptance condition is insufficient for tree automata.

Classical theory of definability developed two basic topological hierarchies: Borel and projective, along with their recursion-theoretic counterparts: arithmetical and analytical. These hierarchies classify the relations over both finite (integers) and infinite (reals, or ω^ω) objects. Although the classical hierarchies are relevant to both finite and infinite computations, it is not in the same way.

Classical complexity theory borrows its basic concepts from recursion theory (reduction, completeness), and applies them by analogy, but the scopes of the two theories are, strictly speaking, different. Indeed, complexity theory studies only a fragment of computable sets and functions, while recursion theory goes far beyond computable world. Finite-state recognizability (regularity) forms the very basic level in complexity hierarchies (although it is of some interest for circuit complexity).

In contrast, finite state automata running over infinite words or trees exhibit remarkable expressive power in terms of the classical hierarchies. Not surprisingly, such automata can recognize uncomputable sets if *computable* means *finite time*. Actually, the word automata reach the second level of the Borel hierarchy, while the tree automata can recognize Borel sets on any finite level, and also — as we have already remarked — some non-Borel sets. So, in spite of a strong restriction to finite memory, automata can reach the very level of complexity studied by the classical definability theory. Putting it the other way around, the classical hierarchies reveal their finite state hardcore.

In this paper we overview the interplay between automata on infinite trees and the classical definability hierarchies, along with a subtle refinement of the Borel hierarchy, known as the hierarchy of Wadge. The emerging picture is not always as expected. Although, in general, topological complexity underlines the automata-theoretic one, the yardsticks are not always compatible, and at one level automata actually refine the Wadge hierarchy. A remarkable application exploits the properties of complete metric spaces: in the proof of the hierarchy theorem for alternating automata, the diagonal argument follows directly from the Banach fixed-point theorem.

2 Climbing up the hierarchies

It is sufficiently representative to consider binary trees. A full binary tree over a finite alphabet Σ is a mapping $t : \{1,2\}^* \to \Sigma$. As a motivating example consider two properties of trees over $\{a, b\}$.

- L is the set of trees such that, on each path, there are infinitely many b's (in symbols: $(\forall \pi \in \{1,2\}^\omega)(\forall m)(\exists n \geq m)\ t(\pi \upharpoonright n) = b$).
- M is the set of trees such that, on each path, there are only finitely many a's (in symbols: $(\forall \pi \in \{1,2\}^\omega)(\exists m)(\forall n \geq m)\ t(\pi \upharpoonright n) = b$).

(In the above, $\pi \upharpoonright n$ denotes the prefix of π of length n.) At first sight the two properties look similar, although the quantifier alternations are slightly different. The analysis below will exhibit a huge difference in complexity: one of the sets is definable by a Π^0_2 formula of arithmetics, while the other is not arithmetical, and even not Borel.

We have just mentioned two views of classical mathematics, where the complexity of sets of trees can be expressed: topology and arithmetics. For the former, the set T_Σ of trees over Σ is equipped with a metric

$$\mathrm{d}(t_1, t_2) = \begin{cases} 0 & \text{if } t_1 = t_2 \\ 2^{-n} \text{ with } n = \min\{|w| : t_1(w) \neq t_2(w)\} & \text{otherwise} \end{cases}$$

For the latter, trees can be encoded as functions over natural numbers ω. The two approaches are reconciliated by viewing trees as elements of the Cantor discontinuum $\{0,1\}^\omega$. Indeed, by fixing a bijection $\iota : \omega \to \{1,2\}^*$, and an injection $\rho : \Sigma \to \{0,1\}^\ell$ (for sufficiently large ℓ), we continuously embed

$$t \mapsto \rho \circ t \circ \iota$$

T_Σ into $(\{0,1\}^\omega)^\ell$, which in turn is homeomorphic to $\{0,1\}^\omega$. It is easy to see that we have a homeomorphism $T_\Sigma \approx \{0,1\}^\omega$, whenever $2 \leq |\Sigma|$.

On the other hand, as far as computability is concerned, the functions in ω^ω can be encoded as elements of $\{0,1\}^\omega$. Assuming that ι above is computable, we can apply the recursion-theoretic classification to trees.

We now recall classical definitions. Following [10], we present topological hierarchies as the relativized versions of recursion-theoretic ones. Thus we somehow inverse the historical order, as the projective hierarchy (over reals) was the first one studied by Borel, Lusin, Kuratowski, Tarski, and others (see [1]). However, from computer science perspective, it is natural to start with Turing machine. Let $k, \ell, m, n, \ldots$ range over natural numbers, and $\alpha, \beta, \gamma, \ldots$ over infinite words in $\{0,1\}^\omega$; boldface versions stand for vectors thereof. We consider relations of the form $R \subseteq \omega^k \times (\{0,1\}^\omega)^\ell$, where (k,ℓ) is the *type* of R. The concept of *(partially) recursive relation* directly generalizes the familiar one (see, e.g., [10, 23]). In terms of Turing machines, a tuple $\langle \mathbf{m}, \alpha \rangle$ forms an entry for a machine, with α spread over infinite tapes. Note that if a Turing machine gives an answer in finite time, the assertion $R(\mathbf{m}, \alpha)$ depends only on a finite fragment of α. Consequently the complement $\overline{R}$ of a recursive relation R is also recursive.

The first-order projection of an arbitrary relation R of type $(k+1, \ell)$ is given by

$$\exists^0 R \quad = \quad \{\langle \mathbf{m}, \alpha \rangle : (\exists n)\, R(\mathbf{m}, n, \alpha)\}$$

and the second-order projection of a relation R of type $(k, \ell+1)$ is given by

$$\exists^1 R \quad = \quad \{\langle \mathbf{m}, \alpha \rangle : (\exists \beta)\, R(\mathbf{m}, \alpha, \beta)\}$$

The *arithmetical hierarchy* can be presented by

$$\begin{array}{rcl} \Sigma^0_0 & = & \text{the class of recursive relations} \\ \Pi^0_n & = & \{\overline{R} : R \in \Sigma^0_n\} \\ \Sigma^0_{n+1} & = & \{\exists^0 R : R \in \Pi^0_n\} \\ \Delta^0_n & = & \Sigma^0_n \cap \Pi^0_n \end{array}$$

The relations in the class $\bigcup_{n<\omega} \Sigma^0_n = \bigcup_{n<\omega} \Pi^0_n$ are called *arithmetical*. Note that $\overline{R}$ is arithmetical if so is R.

The *analytical hierarchy* can be presented by

$$\begin{array}{rcl} \Sigma^1_0 & = & \text{the class of arithmetical relations} \\ \Pi^1_n & = & \{\overline{R} : R \in \Sigma^1_n\} \\ \Sigma^1_{n+1} & = & \{\exists^1 R : R \in \Pi^1_n\} \\ \Delta^1_n & = & \Sigma^1_n \cap \Pi^1_n . \end{array}$$

The two hierarchies have their relativized counterparts usually distinguished by the boldface notation. For a relation R of type $(k, \ell+1)$ and $\beta \in \{0,1\}^\omega$, let

$$R[\beta] \quad = \quad \{\langle \mathbf{m}, \alpha \rangle : R(\mathbf{m}, \alpha, \beta)\}$$

Then, for $i = 0, 1$, we define

$$\begin{array}{rcl} \boldsymbol{\Sigma}^i_n & = & \{R[\beta] : R \in \Sigma^i_n,\ \beta \in \{0,1\}^\omega\} \\ \boldsymbol{\Pi}^i_n & = & \{R[\beta] : R \in \Pi^i_n,\ \beta \in \{0,1\}^\omega\} \\ \boldsymbol{\Delta}^i_n & = & \boldsymbol{\Sigma}^0_n \cap \boldsymbol{\Pi}^i_n \end{array}$$

The crucial observation is that the $\boldsymbol{\Sigma}^0_1$ relations (of type $(0, \ell)$) coincide with *open* relations on $\{0,1\}^\omega$ with the Cantor topology. To see this, note that an open set in $\{0,1\}^\omega$ can be presented by $\bigcup_{v \in B} v\{0,1\}^\omega$, for some $B \subseteq \{0,1\}^*$, and hence we can present it by $(\exists n)\, R(n, \alpha, \beta)$, where the parameter β lists the elements of B, and the recursive relation verifies, given $n = \langle k, m \rangle$ that the kth prefix of α coincides with the mth element of B. (The other direction is straightforward.) Next it is easy to see that relations in $\boldsymbol{\Sigma}^0_{n+1}$ coincide with the countable unions of relations in $\boldsymbol{\Pi}^0_n$ (of suitable type). Therefore the classes $\boldsymbol{\Sigma}^0_n$, $\boldsymbol{\Pi}^0_n$ form the initial segment of the *Borel hierarchy* over $\{0,1\}^\omega$.

Similarly, the classes $\boldsymbol{\Sigma}^1_n$, $\boldsymbol{\Pi}^1_n$, form the so-called *projective* hierarchy over $\{0,1\}^\omega$.

Like in computation/complexity theory, the problems can be compared *via* reductions. We say that a continuous mapping of topological spaces, $\varphi : \mathcal{T}_1 \to \mathcal{T}_2$, *reduces* a set $A \subseteq \mathcal{T}_1$ to a set $B \subseteq \mathcal{T}_2$, if $A = \varphi^{-1}(B)$; in this case we say that A is *Wadge reducible* to B, in symbols $A \leq_{\mathrm{W}} B$. A set B is *complete* in a class $\mathcal{C} \subseteq \wp(\mathcal{T})$ if $B \in \mathcal{C}$ and $(\forall A \in \mathcal{C})\ A \leq_{\mathrm{W}} B$.

A remarkable point is that complete sets may have very simple structure.

Example 2.1. The singleton $\{0^\omega\}$ is in Π^0_1, and it is complete for $\boldsymbol{\Pi}^0_1$. The membership in Π^0_1 is seen by presentation of the complement by $(\exists n)\, \alpha(n) \neq 0$. Now let L be any closed subset of ω^ω. Define $\hat{f} : \omega^* \to \omega^*$ by

$$\hat{f}(xy) = 0^{|x|} y$$

where x is the longest prefix of xy being also a prefix of u, for some $u \in L$. Then it is easy to see that the mapping $f : \omega^\omega \to \omega^\omega$ given by

$$f(u)(n) = \hat{f}(u \restriction n+1)(n)$$

is a desired reduction (where $u = u_0 u_1 \ldots$ and $u \restriction n+1 = u_0 u_1 \ldots u_n$).

It can be seen that, in fact, any singleton $\{\alpha\}$ is complete in $\mathbf{\Pi}^0_1$, although in general it need not be in Π^0_1.

The reader may be puzzled by triviality of this example compared to the construction of complete sets of natural numbers in Π^0_1 or in Σ^0_1. Intuitively, the second-order objects (trees or words) are "less sensitive" to first-order quantification.
In a similar vein, one can show

Example 2.2. The set $\{0,1\}^*0^\omega$ is in Σ^0_2, and it is complete in $\mathbf{\Sigma}^0_2$.

We now revisit our motivating example from beginning of this section.

Example 2.3. It is not hard to see that the set L is in class Π^0_2. Although the original definition has used a second-order quantifier (for all paths), a simpler definition can be given by exploiting arithmetic (like encoding finite sets of nodes by single numbers):

> $t \in L \iff$ for all $v \in \{1,2\}^*$, there is a finite maximal antichain B below v with $(\forall w \in B)\, t(w) = b$.

On the other hand, the set M, which is by definition in Π^1_1, is also complete in $\mathbf{\Pi}^1_1$ w.r.t. continuous reductions, hence not Borel. The completeness can be seen by reduction of the set W of the suitably encoded wellfounded (non-labeled) trees $T \subseteq \omega^*$ (see, e.g., [19]), which is well-known to be $\mathbf{\Pi}^1_1$-complete [11].

3 The power of game languages

The properties of Example 2.3 have a powerful generalization, which is best understood by viewing sequences in $\{a,b\}^\omega$ as outcomes of some infinite two-player game, where one of the players wants to see b infinitely often, while the other does not. To make this game more general/symmetric, we assume that each player has her or his favorite set of letters, and to make the result definite, we assume a priority order on letters. This gives rise to parity games (introduced by Emerson and Jutla [8], and independently by A.W. Mostowski [13]), the concept highly relevant to the μ-calculus-based model checking and to automata theory (see [26]). We briefly recall it now.

A *parity game* is a perfect information game of possibly infinite duration played by two players, say Eve and Adam. We present it as a tuple

$\langle V_\exists, V_\forall, \mathrm{Move}, p_0, \mathrm{rank}\rangle$, where $V_\exists$ and $V_\forall$ are (disjoint) sets of positions of Eve and Adam, respectively, $\mathrm{Move} \subseteq V \times V$ is the relation of possible moves, with $V = V_\exists \cup V_\forall$, $p_0 \in V$ is a designated initial position, and $\mathrm{rank} : V \to \omega$ is the ranking function.

The players start a play in the position p_0 and then move the token according to relation Move (always to a successor of the current position), thus forming a path in the graph (V, Move). The move is selected by Eve or Adam, depending on who is the owner of the current position. If a player cannot move, she/he looses. Otherwise, the result of the play is an infinite path in the graph, $v_0, v_1, v_2, \ldots$ Eve wins the play if $\limsup_{n\to\infty} \mathrm{rank}(v_n)$, is even, otherwise Adam wins. A crucial property of parity games is the *positional determinacy*: any position is winning for one of the players, and moreover a winning strategy of player θ can be chosen *positional*, i.e., represented by a (partial) function $\sigma : V_\theta \to V$. We simply say that Eve *wins* the game if she has a winning strategy, the similar for Adam. (See [9] for more detailed introduction to parity games.)

Here we are interested in several groups of tree languages related to the parity games.

For $\iota \in \{0,1\}$ and $\iota \leq \kappa < \omega$, let

$$\begin{aligned}
\Sigma_{(\iota,\kappa)} &= \{\iota, \iota+1, \ldots, \kappa\} \\
M_{(\iota,\kappa)} &= \{u \in \Sigma^\omega_{(\iota,\kappa)} : \limsup_{n\to\infty} u_n \text{ is even } \} \\
T_{(\iota,\kappa)} &= \{t \in T_{\Sigma_{(\iota,\kappa)}} : (\forall \pi \in \{1,2\}^\omega) t \restriction \pi \in M_{(\iota,\kappa)}\},
\end{aligned}$$

where $t \restriction \pi$ stands for the restriction of t to the path π. That is, $T_{(\iota,\kappa)}$ is the set of trees over $\Sigma_{(\iota,\kappa)}$ such that, on each path, the highest label occurring infinitely often is even. The sets L and M of Example 2.3 can be readily identified with $T_{(1,2)}$ and $T_{(0,1)}$, respectively.

We now present an important game variation of sets $T_{(\iota,\kappa)}$; these will be tree languages over alphabet $\{\exists,\forall\} \times \Sigma_{(\iota,\kappa)}$.

With each tree t in $T_{\{\exists,\forall\}\times\Sigma_{(\iota,\kappa)}}$, we associate a parity game $G(t)$, as described in the previous section, with

- $V_\exists = \{v \in \{1,2\}^* : t(v) \downarrow_1 = \exists\}$,
- $V_\forall = \{v \in \{1,2\}^* : t(v) \downarrow_1 = \forall\}$,
- $\mathrm{Move} = \{(w, wi) : w \in \{1,2\}^*,\ i \in \{1,2\}\}$,
- $p_0 = \varepsilon$ (the root of the tree),
- $\mathrm{rank}(v) = t(v) \downarrow_2$, for $v \in \{1,2\}^*$.

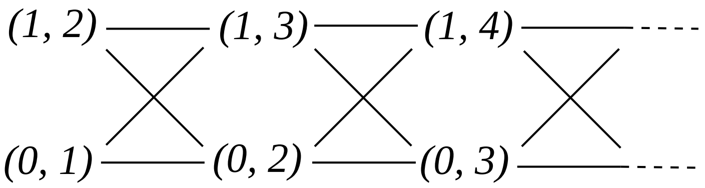


FIGURE 1. The Mostowski–Rabin index hierarchy.

The set $W_{(\iota,\kappa)}$ consists of those trees for which Eve wins the game $G(t)$. Note that this means that Eve can force the resulting path π to satisfy $(t \restriction \pi) \downarrow_2 \in M_{(\iota,\kappa)}$.

Finally, we introduce the *weak version* of all the concepts above, which is obtained by replacing everywhere lim sup by sup. We denote by $L^\flat$ the weak version of L. So, in particular $M^\flat_{(\iota,\kappa)} = \{u \in \Sigma^\omega_{(\iota,\kappa)} : \sup_{n\to\infty} u_n \text{ is even }\}$. Similarly, the *weak parity games* differ from the games defined above in that Eve wins a play if the *highest* rank occurring in the play is even.

It is useful to have a partial ordering on pairs (ι,κ), with $\iota \in \{0,1\}$, which we call *Mostowski-Rabin indices*. We let $(\iota,\kappa) \sqsubseteq (\iota',\kappa')$ if either $\iota' \le \iota$ and $\kappa \le \kappa'$ (i.e., $\{\iota,\ldots,\kappa\} \subseteq \{\iota',\ldots,\kappa'\}$) or $\iota = 0$, $\iota' = 1$, and $\kappa + 2 \le \kappa'$ (i.e., $\{\iota+2,\ldots,\kappa+2\} \subseteq \{\iota',\ldots,\kappa'\}$). We consider the indices $(1,\kappa)$ and $(0,\kappa-1)$ as *dual*, and let $\overline{(\iota,\kappa)}$ denote the index dual to (ι,κ). Note that $\overline{\overline{(\iota,\kappa)}} = (\iota,\kappa)$. The ordering is represented on Figure 1.

Clearly, in each of the above-defined families, the ordering on Mostowski-Rabin indices induces inclusion of corresponding sets.

Now the crucial observation is the following. If $\mathcal{T}$ is a complete metric space then no *contracting* reduction can reduce a set $A \subseteq \mathcal{T}$ to its complement $\overline{A}$. Indeed, otherwise, by the Banach Fixed-Point Theorem, we would have

$$a \in A \iff f(a) \in \overline{A} \iff a \in \overline{A} \quad \text{(contradiction)},$$

for the fixed point $a = f(a)$.
It immediately implies the following.

Lemma 3.1. No contracting mapping reduces $W_{\overline{(\iota,\kappa)}}$ to $W_{(\iota,\kappa)}$, or $W^\flat_{\overline{(\iota,\kappa)}}$ to $W^\flat_{(\iota,\kappa)}$.

Proof. Although $W_{\overline{(\iota,\kappa)}}$ and $W_{(\iota,\kappa)}$ are over different alphabets, we have an isometry of $T_{\Sigma_{(\iota,\kappa)}}$ and $T_{\Sigma_{\overline{(\iota,\kappa)}}}$, induced by the re-labeling of symbols which exchanges quantifiers and alters the ranks by ± 1. This isometry reduces $\overline{W_{(\iota,\kappa)}}$ to $W_{\overline{(\iota,\kappa)}}$, so the claim follows from the observation above. The argument for weak version is similar. Q.E.D.

It turns out that we can strengthen the above lemma by removing the hypothesis of contractivity. This is because, in general, any continuous reduction of $W_{(\iota,\kappa)}$ to some L can be improved to a contracting one, by composing it with a "stretching" reduction of $W_{(\iota,\kappa)}$ to itself. The details can be found in [3]. Thus we obtain the following.

Theorem 3.2. The game languages form a hierarchy w.r.t. the Wadge reducibility, i.e.,

$$\begin{aligned}(\iota,\kappa) \sqsubseteq (\iota',\kappa') \quad &\text{iff} \quad W_{(\iota,\kappa)} \leq_{\mathrm{W}} W_{(\iota',\kappa')} \\ &\text{iff} \quad W^{\flat}_{(\iota,\kappa)} \leq_{\mathrm{W}} W^{\flat}_{(\iota',\kappa')}\end{aligned}$$

This result has several applications involving automata. Let us first recall definition of an alternating parity automaton.

An *alternating parity tree automaton* can be presented as a tuple $\mathcal{A} = \langle \Sigma, Q_{\exists}, Q_{\forall}, q_0, \delta, \mathrm{rank}\rangle$, where the set of states Q is partitioned into existential states $Q_{\exists}$ and universal states $Q_{\forall}$, $\delta \subseteq Q \times \Sigma \times \{1,2,\varepsilon\} \times Q$ is a transition relation, and $\mathrm{rank} : Q \to \omega$ a *rank* function. An input tree t is accepted by $\mathcal{A}$ iff Eve has a winning strategy in the parity game $\langle Q_{\exists} \times \{1,2\}^*, Q_{\forall} \times \{1,2\}^*, (q_0,\varepsilon), \mathrm{Move}, \mathrm{rank}\rangle$, where $\mathrm{Move} = \{((p,v),(q,vd)) \colon v \in \mathrm{dom}(t), (p,t(v),d,q) \in \delta\}$ and $\mathrm{rank}(q,v) = \mathrm{rank}(q)$.

We can assume without loss of generality that $\min \mathrm{rank}(Q)$ is 0 or 1. The pair $(\min \mathrm{rank}(Q), \max \mathrm{rank}(Q))$ is the *Mostowski-Rabin index* of the automaton.

A *weak alternating parity tree automaton* is defined similarly, by restriction to weak parity games. Strictly speaking, a weak automaton is not a parity automaton, but it can be easily turned into one. It is enough to multiply the set of states by $\mathrm{rank}(Q)$ so that the second component keeps record of the highest rank seen so far (it can only increase). It is well known that the languages recognized by weak alternating automata are exactly those recognizable by both $(0,1)$ and $(1,2)$ automata (it follows essentially from [22]).

It is straightforward to see that each $W_{(\iota,\kappa)}$ is recognized by a parity automaton of index (ι,κ), and each $W^{\flat}_{(\iota,\kappa)}$ is recognized by a weak parity automaton of index (ι,κ).

The next important observation is the following lemma:

Lemma 3.3. If a set of trees T is recognized by a (weak) alternating automaton of index (ι,κ) then $T \leq_{\mathrm{W}} W_{(\iota,\kappa)}$ (resp. $T \leq_{\mathrm{W}} W^{\flat}_{(\iota,\kappa)}$).

The exact construction is somewhat tedious, but the idea of the reduction is simple: for a tree t, we construct a full game tree and then forget anythings but ranks. The details are presented in [2, 5], where the reduction is even made contracting, but in view of Theorem 3.2, it is not necessary.

Combining Theorem 3.2 with Lemma 3.3, we obtain

Theorem 3.4. The tree languages $W_{(\iota,\kappa)}$ form a strict hierarchy for the Mostowski-Rabin indices of alternating parity automata.

The tree languages $W^\flat_{(\iota,\kappa)}$ form a strict hierarchy for Mostowski-Rabin indices of weak alternating parity automata.

The first claim was established by Bradfield [6]; the proof *via* the Banach Theorem was given later by Arnold [2] (see also [5]).

The strictness of the hierarchy of weak automata was first established by Mostowski [14], who shown that it is equivalent to a hierarchy based on weak monadic formulas, and then used the strictness of the latter hierarchy, previously proved by W. Thomas [25].

As Skurczyński showed [24] (by other examples) that there are $\mathbf{\Pi}^0_n$ and $\mathbf{\Sigma}^0_n$-complete tree languages recognized by weak alternating automata of index $(0,n)$ and $(1,n+1)$ accordingly, Lemma 3.3 also implies that the sets $W^\flat_{(\iota,\kappa)}$ are hard on the corresponding finite levels of the Borel hierarchy. Recently, Duparc and Murlak [7] showed that these sets are actually complete in these classes.

Theorem 3.5 (Duparc-Murlak, [7])**.** If a tree language T is recognized by a weak alternating automaton of index $(0,n)$ (resp. $(1,n+1)$) it holds that $T \in \mathbf{\Pi}^0_n$ (resp. $T \in \mathbf{\Sigma}^0_n$).

Let us complete this recent theorem by what we have known since long time about strong alternating automata.

Theorem 3.6. If a tree language T is recognized by an alternating automaton of index $(0,1)$ (resp. $(1,2)$) it holds that $T \in \mathbf{\Pi}^1_1$ (resp. $T \in \mathbf{\Sigma}^1_1$). For any recognizable tree language T, $T \in \mathbf{\Delta}^1_2$.

The first claim was (essentially) established by Rabin [22] in terms of the formulas of S2S and for *nondeterministic* automata of index $(1,2)$, now called *Büchi automata.* It was later shown [4] that for Büchi automata alternation does not matter. Note that this implies in particular that the set M of Example 2.3 cannot be recognized by a Büchi automaton [22]. The second claim follows from definition and Rabin's Complementation Lemma [21].

4 How fine is the Wadge hierarchy?

In the previous section we saw that with regular tree languages one can go much higher in the Borel hierarchy than with regular ω-languages. Now we should like to concentrate on the *fineness* of the hierarchy. Let us start with a simple example.

For $n \in \omega$, let L_n denote the set of trees over the alphabet $[0,n] = \{0,1,\ldots,n\}$, whose leftmost path satisfies the weak parity condition, i. e., the highest label on this path is even. For example: $L_0 = T_{[0,0]}$ consists of the only tree over the alphabet $\{0\}$, and L_1, a closed subset of $T_{[0,1]}$, consists of trees with 0's on the leftmost path and 0's or 1's elsewhere. It is an easy exercise to show that L_n are regular.

Even everyday intuition of complexity tells us that L_{k+1} is more complex then L_k. This can be formalized by means of continuous reductions introduced in the previous section. Consider an identity function $\mathrm{id} : T_{[0,\ell]} \to T_{[0,k]}$, with $\ell < k$. Clearly, this function reduces L_ℓ to L_k: $t \in L_\ell$ iff $\mathrm{id}(t) \in L_k$. Hence the languages L_n form a hierarchy: $L_0 \leq_{\mathrm{W}} L_1 \leq_{\mathrm{W}} L_2 \leq_{\mathrm{W}} \dots$.

OK, but this already happened with the weak game languages from the previous section, so what is the difference? Well, observe that all these languages can be presented as a finite Boolean combination of closed sets, e.g.

$$L_3 = \{t\colon \forall_i\ t(0^i) \in [0,2]\} \setminus \{t\colon \forall_i\ t(0^i) \in [0,1]\} \cup \{t\colon \forall_i\ t(0^i) \in [0,0]\}\,.$$

Consequently, our entire hierarchy lies within Δ^0_2!

'All right,' the reader might say, 'but how do I know that, say, L_7 cannot be reduced to L_6? How do I know that this "hierarchy" is *strict*?' It is, but showing that directly would be rather tiresome. Instead, we shall use a handy characterization provided by Wadge games.

Originally, these games were defined for ω-words (see [20]). Here, we shall use a tree version. For any pair of tree languages $L \subseteq T_\Sigma, M \subseteq T_\Gamma$ the *Wadge game* $G_{\mathrm{W}}(L, M)$ is played by Spoiler and Duplicator. Each player builds a tree, $t_S \in T_\Sigma$ and $t_D \in T_\Gamma$ respectively. In every round, first Spoiler adds at least one level to t_S and then Duplicator can either add some levels to t_D or skip a round. Duplicator must not skip infinitely long, so that t_D is really an infinite tree. Duplicator wins the game if $t_S \in L \iff t_D \in M$.

Lemma 4.1 (Wadge)**.** Duplicator has a winning strategy in $G_{\mathrm{W}}(L, M)$ if and only if $L \leq_{\mathrm{W}} M$.

Proof. Essentially, a winning strategy for Duplicator can be transformed into a continuous reduction, and vice versa.

Suppose Duplicator has a winning strategy ρ. For any tree t constructed by Spoiler, there exist a unique tree t_ρ which will be constructed by Duplicator if he is using the strategy ρ. The map $t \mapsto t_\rho$ is continuous by the rules of the Wadge game, and $t \in L \iff t_\rho \in M$ since ρ is winning.

Conversely, suppose there exist a reduction $t \mapsto \varphi(t)$. It follows that there exist a sequence n_k (without loss of generality, increasing) such that the level k of $\varphi(t)$ depends only on the levels $1, \dots, n_k$ of t. Then the strategy for Duplicator is the following: if the number of the round is n_k, fill in the k-th level of t_D according to $\varphi(t_S)$; otherwise skip. Q.E.D.

Let us now see that the languages $L_1, L_2, \ldots$ form a strict hierarchy, i.e., $L_\ell \not\leq_{\mathrm{W}} L_k$ for $\ell > k$. Consider the following strategy for Spoiler in $G_{\mathrm{W}}(L_\ell, L_k)$. Outside of the leftmost path play 1 all the time - it does not matter anyhow. On the leftmost path always play $m+1$, where m is the last number played by Duplicator on the leftmost path of his tree (or 0 if he has kept skipping so far). This strategy only uses numbers $[1, k+1] \subseteq [1, \ell]$, so it is legal. Obviously, the highest number we use on the leftmost path is of different parity then the highest number used by Duplicator, so $t_S \in L_\ell \iff t_D \notin L_k$. Hence, the strategy is winning for Spoiler, and by the lemma above $L_\ell \not\leq_{\mathrm{W}} L_k$.

Observe that in the above argument we have shown that Duplicator does not have a winning strategy by providing a winning strategy for Spoiler. In general it does not always hold that one of the players must have a winning strategy in $G_{\mathrm{W}}(L, M)$. Luckily, by Martin's famous determinacy theorem, it holds for Borel sets.

Theorem 4.2. If L, M are Borel languages, than one of the players has a winning strategy in $G_{\mathrm{W}}(L, M)$.

In fact the power of Wadge games relies on the above result: it lets us replace a non-existence proof with an existence proof. Without determinacy, Wadge games only give a rather trivial correspondence between reductions and strategies.

The Wadge ordering $\leq_{\mathrm{W}}$ induces a natural equivalence relation, $L \equiv_{\mathrm{W}} M$ iff $L \leq_{\mathrm{W}} M$ and $L \geq_{\mathrm{W}} M$. The order induced on the $\equiv_{\mathrm{W}}$ equivalence classes of Borel languages is called the *Wadge hierarchy.* The determinacy theorem actually gives a very precise information on the shape of the Wadge hierarchy.

Theorem 4.3 (Wadge Lemma)**.** For Borel languages L, M it holds that

$$L \leq_{\mathrm{W}} M \quad \text{or} \quad \overline{L} \geq_{\mathrm{W}} M\,.$$

The proof of this result simply transforms Spoiler's winning strategy in $G_{\mathrm{W}}(L, M)$, which must exist by determinacy, into Duplicator's winning strategy in $G_{\mathrm{W}}(M, \overline{L})$ (see [11] or [20]). In other words the theorem says that the width of the Wadge hierarchy is at most two, and if L and M are incomparable, then $L \equiv_{\mathrm{W}} \overline{M}$. It means that the Wadge ordering is almost linear. The second fundamental result states that it is also a well-ordering.

Theorem 4.4 (Wadge-Martin-Monk)**.** The Wadge hierarchy is wellfounded.

Altogether, the position of a language in the Wadge hierarchy is determined, up to complementation, by its height.

If $L \equiv_{\mathrm{W}} \overline{L}$ then L is called *selfdual.* Otherwise L is not comparable with $\overline{L}$ and is called *non-selfdual.* Steel and Van Weesp proved that the selfdual and non-selfdual levels alternate (see [11]). If the alphabet is finite, which is our case, on limit steps we have non-selfduals. Furthermore, the selfduals on successor levels can be obtained as disjoint unions of their predecessors. All this makes it reasonable to ignore selfduals when counting the height. Hence, we choose the following definition of the Wadge degree:

- $\mathbf{d}_{\mathrm{W}}(\varnothing) = \mathbf{d}_{\mathrm{W}}(\overline{\varnothing}) = 1$,
- $\mathbf{d}_{\mathrm{W}}(L) = \sup\{\mathbf{d}_{\mathrm{W}}(M)+1 \colon M \text{ is non-selfdual, } M <_{\mathrm{W}} L\}$ for $L >_{\mathrm{W}} \varnothing$.

We have now all the tools necessary to formalize the question asked in the title of the present section. For a family of languages $\mathcal{F}$ define the *height* of the Wadge hierarchy restricted to $\mathcal{F}$ as the order type of the set $\{\mathbf{d}_{\mathrm{W}}(L) \colon L \in \mathcal{F}\}$ with respect to the usual order on ordinals. What we are interested in is the height of the hierarchy of regular languages.

We have shown already that the height of the hierarchy of $\{L_0, L_1, \ldots\}$ is ω. This of course gives a lower bound for the height of the hierarchy of all regular languages. We shall now see how this result can be improved. We consider a subclass of regular languages, the languages recognized by weak alternating automata. Any lower bound for weak languages will obviously hold for regular languages as well.

It will be convenient to work with languages of binary trees which are not necessarily full, i.e., partial functions from $\{0,1\}^*$ to Σ with prefix closed domain. We call such trees *conciliatory*. Observe that the definition of weak automata works for conciliatory trees as well. We shall write $L_{\mathrm{C}}(\mathcal{A})$ to denote the set of conciliatory trees accepted by $\mathcal{A}$. For conciliatory languages L, M one can define a suitable version of Wadge games $G_{\mathrm{C}}(L, M)$. Since it is not a problem if the players construct a conciliatory tree during the play, they are now both allowed to skip, even infinitely long. Analogously one defines the conciliatory hierarchy induced by the order $\leq_{\mathrm{C}}$, and the conciliatory degree $\mathbf{d}_{\mathrm{C}}$.

The conciliatory hierarchy embeds naturally into the non-selfdual part of the Wadge hierarchy. The embedding is given by the mapping $L \mapsto L_S$, where L is a language of conciliatory trees over Σ, and L_s is a language of full trees over $\Sigma \cup \{s\}$ which belong to L when we ignore the nodes labeled with s (together with the subtrees rooted in their right children) in a top down manner. Proving that $L \leq_{\mathrm{C}} M \iff L_s \leq_{\mathrm{W}} M_s$ for all conciliatory languages L and M only requires translating strategies form one game to the other. It can be done easily, since arbitrary skipping in $G_{\mathrm{C}}(L, M)$ gives the same power as the s labels in $G_{\mathrm{W}}(L_s, M_s)$. Within the family of languages of finite Borel rank, the embedding is actually an isomorphism, and $\mathbf{d}_{\mathrm{C}}(L) = \mathbf{d}_{\mathrm{W}}(L_s)$ [7].

Observe that if L is recognized by a weak alternating automaton, so is L_s. Indeed, by adding to δ a transition $p \xrightarrow{0,s} p$ for each state p one

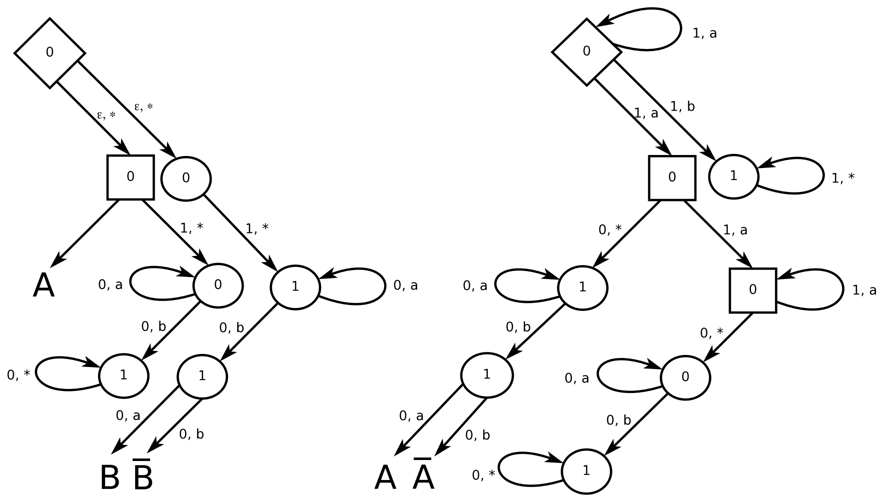


FIGURE 2. The automata $\mathcal{B} + \mathcal{A}$ and $\mathcal{A} \cdot \omega$.

transforms an automaton $\mathcal{A}$ into $\mathcal{A}_s$ such that $L(\mathcal{A}_s) = (L_{\mathrm{C}}(\mathcal{A}))_s$. Hence, the conciliatory subhierarchy of weakly recognizable languages embeds into the Wadge hierarchy of weakly recognizable languages, and it is enough to show a lower bound for conciliatory languages.

So far, when constructing hierarchies, we have been defining the whole family of languages right off. This time we shall use a different method. We shall define operations transforming simple languages into more sophisticated ones. These operations will induce, almost accurately, classical ordinal operations on the degrees of languages: sum, multiplication by ω, and exponentiation with the base ω_1. We shall work with automata on trees over a fixed alphabet $\{a, b\}$.

The sum $\mathcal{B} + \mathcal{A}$ and multiplication $\mathcal{A} \cdot \omega$ are realized by combining automata recognizing simpler languages with a carefully designed gadget. The constructions are shown on Figure 2. The diamond states are existential and the box states are universal. The circle states can be treated as existential, but in fact they give no choice to either player. The transitions leading to $\mathcal{A}$, $\overline{\mathcal{A}}$, $\mathcal{B}$ and $\overline{\mathcal{B}}$ should be understood as transitions to the initial states of the according automata. The priority functions of these automata might need shifting up, so that they were not using the value 0.

The automaton $\exp \mathcal{A}$ is a bit more tricky. This time, we have to change the whole structure of the automaton. Instead of adding one gadget, we replace each state of $\mathcal{A}$ by a different gadget. The gadget for a state p is shown on Figure 3. By replacing p with the gadget we mean that all the

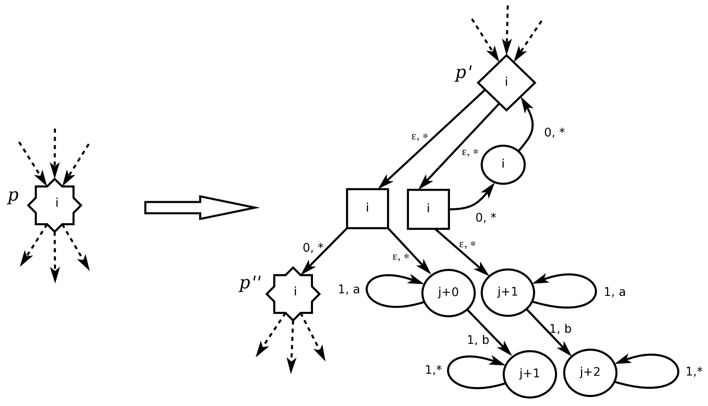


FIGURE 3. The gadget to replace p in the construction of $\exp \mathcal{A}$.

transitions ending in p should now end in p' and all the transitions starting in p should start in p''. Note that the state p'' is the place where the original transition is chosen, so p'' should be existential iff p is existential. The number j is the least even number greater or equal to $i = \operatorname{rank} p$.

Abusing slightly the notation we may formulate the properties of the three constructions as follows.

Theorem 4.5 (Duparc-Murlak, [7]). For all weak alternating automata $\mathcal{A}$, $\mathcal{B}$ it holds that $\mathbf{d}_{\mathrm{C}}(\mathcal{B}+\mathcal{A}) = \mathbf{d}_{\mathrm{C}}(\mathcal{B}) + \mathbf{d}_{\mathrm{C}}(\mathcal{A})$, $\mathbf{d}_{\mathrm{C}}(\mathcal{A}\cdot\omega) = \mathbf{d}_{\mathrm{C}}(\mathcal{A})\cdot\omega$, and $\mathbf{d}_{\mathrm{C}}(\exp\mathcal{A}) = \omega_1^{\mathbf{d}_{\mathrm{C}}(\mathcal{A})+\varepsilon}$, where

$$\varepsilon = \begin{cases} -1 & \text{if } \mathbf{d}_{\mathrm{C}}(\mathcal{A}) < \omega \\ 0 & \text{if } \mathbf{d}_{\mathrm{C}}(\mathcal{A}) = \beta + n \text{ and } \operatorname{cof}\beta = \omega_1 \\ +1 & \text{if } \mathbf{d}_{\mathrm{C}}(\mathcal{A}) = \beta + n \text{ and } \operatorname{cof}\beta = \omega. \end{cases}$$

As a corollary we obtain the promised bound.

Theorem 4.6 (Duparc-Murlak, [7]). The Wadge hierarchy of weakly recognizable tree languages has the height of at least ε_0, the least fixed point of the exponentiation with the base ω.

Proof. It is enough to show the bound for conciliatory languages. By iterating finitely many times sum and multiplication by ω we obtain multiplication by ordinals of the form $\omega^n k_n + \ldots + \omega k_1 + k_0$, i.e., all ordinals less then ω^ω. In other words, we can find a weakly recognizable language of any conciliatory degree from the closure of $\{1\}$ by ordinal sum, multiplication by ordinals $< \omega^\omega$ and pseudo-exponentiation with the base ω_1. It is easy to see that the order type of this set is not changed if we replace pseudo-exponentiation with ordinary exponentiation $\alpha \mapsto \omega_1^\alpha$. This in turn is isomorphic with the closure of $\{1\}$ by ordinal sum, multiplication by ordinals $< \omega^\omega$, and exponentiation with the base ω^ω. This last set is obviously ε_0, the least fixpoint of the exponentiation with the base ω. Q.E.D.

Recently, the second author of this survey has found a modification of the pseudo-exponentiation construction which results in ordinary exponentiation $\alpha \mapsto \omega_1^\alpha$. This result makes it very tempting to conjecture that these are in fact all Wadge degrees realised by weak automata, and if one replaces ω_1 by ω^ω, one gets the degree of the language in the Wadge hierarchy restricted to weakly recognizable languages.

Supposing that the conjecture is true, the next step is an effective description of each degree. Or, in other words, an algorithm to calculate the position of a given language in the hierarchy. Obtaining such a description for all regular languages is the ultimate goal of the field we are surveying. So far this goal is seems far away. The solution might actually rely on analytical determinacy. On the other hand, it may also be the case that determinacy for regular languages is implied by ZFC. The knowledge in this subject is scarce.

To end up with some good news, the problem has been solved for an important and natural subclass of regular languages, the languages recognized by deterministic automata (see below for definition).

Theorem 4.7 (Murlak, [17])**.** The hierarchy of deterministically recognizable languages has the height of $\omega^{\omega\cdot 3} + 3$. Furthermore, there exist an algorithm calculating the exact position of a given language in this hierarchy.

5 Topology versus computation

In this concluding section we should like to confront the classical definability hierarchies with the automata-theoretic hierarchies based on the Mostowski–Rabin index. To this end, let us first recall the concepts of non-deterministic and deterministic tree automata. They are special cases of alternating automata, but it is convenient to use traditional definitions. A *non-deterministic* parity tree automaton over trees in T_Σ can be presented as $A = \langle \Sigma, Q, q_0, \delta, \mathrm{rank}\rangle$, where $\delta \subseteq Q \times \Sigma \times Q \times Q$. A transition $(q, \sigma, p_1, p_2) \in \delta$ is usually written $q \xrightarrow{\sigma} p_1, p_2$.

A *run* of A on a tree $t \in T_\Sigma$ is itself a tree in T_Q such that $\rho(\varepsilon) = q_0$, and, for each $w \in \mathit{dom}\,(\rho)$, $\rho(w) \xrightarrow{t(w)} \rho(w1), \rho(w2)$ is a transition in δ. A *path* in ρ is *accepting* if the highest rank occurring infinitely often along it is even. A *run is accepting* if so are all its paths. Again, the Mostowski-Rabin index of an automaton is the pair $(\min \mathrm{rank}(Q), \max \mathrm{rank}(Q))$, where we assume that the first component is 0 or 1.

An automaton is *deterministic* if δ is a partial function from $Q \times \Sigma$ to $Q \times Q$. It can be observed that languages $W_{(\iota,\kappa)}$ defined in Section 3 can be recognized by non-deterministic automata of index (ι, κ), respectively, and that languages $T_{(\iota,\kappa)}$ defined there can be recognized by deterministic automata of corresponding indices.

In general, the index may decrease if we replace an automaton by an equivalent one of higher type. For example, it is not hard to see that the complements of languages $T_{(\iota,\kappa)}$ can all be recognized by non-deterministic automata of index $(1, 2)$ (Büchi automata), hence these languages themselves are of alternating index $(0, 1)$. But it was showed in [18] that these languages form a hierarchy for the Mostowski-Rabin index of non-deterministic automata. It can be further observed that all $T_{(\iota,\kappa)}$ with $(0, 1) \sqsubseteq (\iota, \kappa)$ are $\mathbf{\Pi}^1_1$-complete, hence by the general theory [11], they are all equivalent w.r.t. the Wadge reducibility. (In fact, it is not difficult to find the reductions to $T_{(0,1)}$ directly.) So in this case the automata-theoretic hierarchy is more fine than the Wadge hierarchy, which is a bit surprising in view of the fineness of the latter hierarchy, as seen in the previous section.

Let us now compare the index hierarchy and the Wadge hierarchy. For infinite words, this comparison reveals a beautiful correspondence, discovered by Klaus Wagner.

Theorem 5.1 (Wagner, [27]).

1. Regular ω-languages have exactly the Wadge degrees of the form $\omega_1^k n_k + \ldots + \omega_1^1 n_1 + n_0$ for $k < \omega$ and $n_0, \ldots, n_k < \omega$.

2. The languages recognized by deterministic automata using $k+1$ ranks (index $[0, k]$ or $[1, k+1]$) correspond to degrees $\leq \omega_1^k$.

Hence, for regular ω-languages, the Wadge hierarchy is a refinement of the index hierarchy. For trees the situation is more complex because we have four nontrivial hierarchies (alternating, weak-alternating, nondeterministic, and deterministic).

The correspondence for weak alternating automata is not yet fully understood. By Theorem 3.5, the raise of topological complexity (in terms of Borel hierarchy) forces the raise of the index complexity. However, the converse is an open problem. *A priori* it is possible that an infinite sequence of tree languages witnessing the weak index hierarchy can be found inside a single Borel class, although it would be rather surprising.

What we do know is that a similar pathology cannot happen for deterministically recognizable tree languages. Indeed, for this class the two hierarchies are largely compatible, however their scope is not large: a deterministic language can either be recognized by a weak automaton of index (at most) $(0,3)$, and hence, by Theorem 3.5 is in the Borel class $\mathbf{\Pi}^0_3$, or it is $\mathbf{\Pi}^1_1$-complete [19]. Moreover, the membership in Borel and in weak-index classes is decidable for deterministic languages [19, 16].

On the other hand, the kind of pathology described above actually does happen if we regard the *deterministic* index hierarchy, i.e., for a deterministically recognizable language we look for the lowest index of a deterministic automaton recognizing it (the case rarely considered in literature). Observe that the hierarchy of regular ω-languages embeds into the hierarchy of deterministic tree languages by a mapping $L \mapsto \{t\colon$ the leftmost branch of t is in $L\}$. Recall that all the regular ω-languages are Boolean combinations of Σ^0_2 languages, denoted Boole(Σ^0_2). It follows that there are deterministic tree languages from each level of the deterministic index hierarchy which are inside Boole(Σ^0_2). At the same time one only needs index $(0,1)$ to get a $\mathbf{\Pi}^1_1$-complete set. In other words, for some $\mathbf{\Pi}^1_1$-complete languages $(0,1)$ is enough, but there are Σ^0_2 languages which need an arbitrarily high index! This means that the deterministic index hierarchy does not embed into the Wadge hierarchy. Apparently, it measures an entirely different kind of complexity.

One might suspect that alternating index would be a more suitable measure in this context. Alternation saves us from increasing the index with complementation. Indeed, the complementation of an alternating automaton is done simply by swapping $Q_\exists$ and $Q_\forall$, and shifting the ranks by one. (To make complementation easy was an original motivation behind alternating automata [15].) If a language has index (ι,κ), its complement will only need $\overline{(\iota,\kappa)}$, and vice versa. As it was stated in Section 3, the strong game languages showing the strictness of the alternating hierarchy form also a strict hierarchy within the Wadge hierarchy. In fact, since each recognizable tree language can be continuously reduced to one of them, they give a scaffold for further investigation of the hierarchy. Such a scaffold will be much needed since the non-Borel part of the Wadge hierarchy is a much dreaded and rarely visited place.

References

[1] J. W. Addison. Tarski's theory of definability: common themes in descriptive set theory, recursive function theory, classical pure logic, and finite-universe logic. *Ann. Pure Appl. Logic*, 126(1-3):77–92, 2004.

[2] A. Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. *ITA*, 33(4/5):329–340, 1999.

[3] A. Arnold and D. Niwiński. Continuous separation of game languages. To appear.

[4] A. Arnold and D. Niwinski. Fixed point characterization of büchi automata on infinite trees. *Elektronische Informationsverarbeitung und Kybernetik*, 26(8/9):451–459, 1990.

[5] A. Arnold and D. Niwiński. *Rudiments of μ-calculus*, volume 146 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., Amsterdam, 2001.

[6] J. C. Bradfield. Simplifying the modal mu-calculus alternation hierarchy. In M. Morvan, C. Meinel, and D. Krob, editors, *STACS*, volume 1373 of *Lecture Notes in Computer Science*, pages 39–49. Springer, 1998.

[7] J. Duparc and F. Murlak. On the topological complexity of weakly recognizable tree languages. In E. Csuhaj-Varjú and Z. Ésik, editors, *FCT*, volume 4639 of *Lecture Notes in Computer Science*, pages 261–273. Springer, 2007.

[8] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377. IEEE, 1991.

[9] E. Grädel, W. Thomas, and Thomas. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science.* Springer, 2002.

[10] P. G. Hinman. *Recursion-theoretic hierarchies.* Springer-Verlag, Berlin, 1978. Perspectives in Mathematical Logic.

[11] A. S. Kechris. *Classical descriptive set theory*, volume 156 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, 1995.

[12] Y. N. Moschovakis. *Descriptive set theory*, volume 100 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., Amsterdam, 1980.

[13] A. W. Mostowski. Games with forbidden positions. Preprint 78, Uniwersytet Gdańsk, Instytyt Matematyki, 1991.

[14] A. W. Mostowski. Hierarchies of weak automata and weak monadic formulas. *Theor. Comput. Sci.*, 83(2):323–335, 1991.

[15] D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theor. Comput. Sci.*, 54:267–276, 1987.

[16] F. Murlak. On deciding topological classes of deterministic tree languages. In C.-H. L. Ong, editor, *CSL*, volume 3634 of *Lecture Notes in Computer Science*, pages 428–441. Springer, 2005.

[17] F. Murlak. The wadge hierarchy of deterministic tree languages. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 408–419. Springer, 2006.

[18] D. Niwiński. On fixed-point clones (extended abstract). In L. Kott, editor, *ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 464–473. Springer, 1986.

[19] D. Niwinski and I. Walukiewicz. A gap property of deterministic tree languages. *Theor. Comput. Sci.*, 1(303):215–231, 2003.

[20] D. Perrin and J.-E. Pin. *Infinite Words. Automata, Semigroups, Logic and Games*, volume 141 of *Pure and Applied Mathematics.* Elsevier, Amsterdam, 2004.

[21] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[22] M. O. Rabin. Weakly definable relations and special automata. In *Mathematical Logic and Foundations of Set Theory (Proc. Internat. Colloq., Jerusalem, 1968)*, pages 1–23. North-Holland, Amsterdam, 1970.

[23] H. Rogers, Jr. *Theory of recursive functions and effective computability.* McGraw-Hill Book Co., New York, 1967.

[24] J. Skurczynski. The borel hierarchy is infinite in the class of regular sets of trees. *Theor. Comput. Sci.*, 112(2):413–418, 1993.

[25] W. Thomas. A hierarchy of sets of infinite trees. In A. B. Cremers and H.-P. Kriegel, editors, *Theoretical Computer Science*, volume 145 of *Lecture Notes in Computer Science*, pages 335–342. Springer, 1983.

[26] W. Thomas. Languages, automata, and logic. In *Handbook of formal languages, Vol. 3*, pages 389–455. Springer, Berlin, 1997.

[27] K. Wagner. On omega-regular sets. *Information and Control*, 43(2):123–177, 1979.

Nondeterministic controllers of nondeterministic processes

André Arnold
Igor Walukiewicz

Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux 1
351 cours de la Libération
33405 Talence cedex, France
andre.arnold@club-internet.fr, igw@labri.fr

Abstract

The controller synthesis problem as formalized by Ramadge and Wonham consists of finding a finite controller that when synchronized with a given plant results in a system satisfying a required property. In this setting, both a plant and a controller are deterministic finite automata, while synchronization is modelled by a synchronous product. Originally, the framework was developed only for safety and some deadlock properties. More recently, Arnold et. al. have extended the setting to all mu-calculus expressible properties and proposed a reduction of the synthesis problem to the satisfiability problem of the mu-calculus. They have also presented some results on decidability of distributed synthesis problem where one requires to find several controllers that control the plant at the same time. The additional difficulty in this case is that each controller is aware of a different part of the whole system. In this paper, an extension of the setting to nondeterministic processes is studied.

1 Introduction

At the end of the eighties, Ramadge and Wonham introduced the theory of control of discrete event systems (see the survey [13] and the books [6] and [3]). In this theory a process (also called a *plant*) is a deterministic non-complete finite state automaton over an alphabet A of events, which defines all possible sequential behaviours of the process. The goal is to find for a given plant another process, called *controller*, such that a synchronous product of the plant and the controller satisfies desired properties. The usual properties considered are for instance, that some dangerous states are never reachable, or that one can always go back to the initial state of the plant. In *decentralized* control one looks for a fixed number of controllers that control the plant simultaneously.

In the setting described above one assumes that both a plant and controllers are deterministic automata. This paper examines what changes when assumption on determinism is dropped. It is shown that nondeterminism in a plant can be handled at no cost, while nondeterminism in controllers may lead to undecidability in the case of decentralized control.

The synthesis problem would be interesting neither form theoretical nor from practical point of view if there were no additional restrictions on controllers. In the most standard form a restriction is determined by two subsets A_{unc} and A_{uobs} of the alphabet A with the associated requirement that:

(C) For any state q of the controller, and for any uncontrollable event a, there is a transition from q labelled by a.

(O) For any state q of the controller, and for any unobservable event a, if there is a transition from q labelled by a then this transition is a loop over q.

In other words, a controller must react to any uncontrollable event and cannot detect the occurrence of an unobservable event.

In [1] an extension of this setting was proposed that handles specifications expressed in the mu-calculus, or rather in its extension called modal loop mu-calculus. This allowed a more general formulation of the synthesis problem:

> (**CC**) Given a plant P and two formulas α and β, does there exist a controller R satisfying β such that $P \times R$ satisfies α?

With formulas α and β one can express properties (C) and (O) but also much more, as for example that an action becomes unobservable after a failure has occurred, or that always one of two actions is controllable but never both at the same time.

The problem (**CC**) can be solved thanks to the division operation [1]. For a process P and a formula α there is a formula α/P such that: $R \vDash \alpha/P$ iff $P \times R \vDash \alpha$. This way a process R is a solution to (**CC**) if and only if $R \vDash (\alpha/P) \wedge \beta$. As $(\alpha/P) \wedge \beta$ is a formula of the modal loop mu-calculus the synthesis problem reduces to the constructive satisfiability problem: construct a model for a formula whenever a model exists. The latter is decidable and a witnessing model, which is a controller, can be constructed.

Ramadge and Wonham have considered also the problem of synthesis of decentralized controllers: a plant can be supervised by several independent controllers (instead of only one). But each controller has its own set of controllable and observable events. Hence, the decentralized control problem is to find $R_1, \ldots, R_n$ such that the supervised system $P \times R_1 \times \cdots \times R_n$ satisfies the specification S and for each i, R_i satisfies (C_i) and (O_i). More generally, in our setting, the decentralized control problem is:

> (**DC**) Given a plant P and modal-loop mu-calculus formulas α, β_1, $\dots$,β_n, do there exist controllers R_i satisfying β_i (for $i = 1, \dots, n$) such that $P \times R_1 \times \cdots \times R_n$ satisfies α?

In [1] it is shown how to solve a decentralized control problem when at most one of the formulas α_i restrains visibility of a controller. If one allows to put visibility restrictions on at least two controllers then the existence of a solution to the problem is undecidable.

Till now, all the constructions assumed that processes are deterministic automata. This may be a limiting assumption if, for example, a plant is a model of a continuous system. In this case a continuous domain of values must be sampled to a discrete one. Hence, the same measurement can correspond to different values that may have different effect on the behaviour of the plant. For similar reasons, the result of actions of controllers may be also not completely determined.

In this paper, we show that in the case of centralized synthesis the approach via division operation still works. We do this by generalizing the division operation described above to a division by a nondeterministic process. This shows that nondeterminism in a plant can be handled at no cost. Next, we study decidability of (**DC**) problem. Thanks to the division, allowing nondeterministic plant does not make the problem more complex. By contrast, if we allow at least two controllers to be nondeterministic, then the problem becomes undecidable even for formulas in the standard mu-calculus. We study also the case when at most one of the controllers can be nondeterministic, obtaining a full characterisation of decidability of (**DC**) problem.

The paper is organized as follows. In the next section we introduce processes and automata on processes. This will be a rather rich version of alternating automata that has not only loop testing, $\circlearrowleft_a$, but also indistinguishability testing $\downdownarrows_{a,b}$. Intuitively, the first constraint will be used to say that a controller cannot observe a, and the second that it cannot make a difference between a and b. These kinds of automata were introduced in [1], and $\downdownarrows_{a,b}$ test was added in [2]. In Section 3 we give basic properties of these automata, like closure under boolean operations and decidability of emptiness. Section 4 presents an operation of division of an automaton by a process. This operation is used in the following section to solve centralized synthesis problem and to eliminate the plant from formalization of decentralized synthesis problem. Main results of the paper are given in this section. The proofs of decidability and undecidability results announced here are given in Sections 6 and 7 respectively.

2 Processes and automata

2.1 Processes

Let A be a finite alphabet of *actions*. A *process* is a finite graph with a distinguished node and with edges labelled by actions:

$$P = \langle S, A, s^0 \in S, e \subseteq S \times A \times S \rangle$$

We shall usually refer to nodes as *states*. We shall write $s \xrightarrow{a} s'$ instead of $(s, a, s') \in e$, and will say that there is a transition labelled a form a state s to a state s'. A process is *deterministic* if e is a partial function $e : S \times A \to S$. We shall write $\mathrm{out}_P(s, a)$ for the set of states reachable from s by a transition labelled a: $\mathrm{out}_P(s, a) = \{s' : s \xrightarrow{a} s'\}$.

A *product* of two processes over the same alphabet is standard

$$P \times R = \langle S_P \times S_R, A, (s^0_P, s^0_R), e_{P\times R} \rangle$$

where $((s_P, s_R), a, (s'_P, s'_R)) \in e_{P\times R}$ if $(s_P, a, s'_P) \in e_P$ and $(s_R, a, s'_R) \in e_R$.

2.2 Games

As our specification language we shall use a rich variant of alternating automata that we shall introduce in the next subsection. It will be very convenient to describe its semantics in terms of games, so we recall necessary definitions here.

A *game* G is a tuple $\langle V_{\mathrm{E}}, V_{\mathrm{A}}, T \subseteq (V_{\mathrm{E}} \cup V_{\mathrm{A}})^2, \mathrm{Acc} \subseteq V^\omega \rangle$ where Acc is a set defining the *winning condition*, and $\langle V_{\mathrm{E}} \cup V_{\mathrm{A}}, T \rangle$ is a graph with the vertices partitioned into those of Eve and those of Adam. We say that a vertex v' is a *successor* of a vertex v if $T(v, v')$ holds.

A *play* between Eve and Adam from some vertex $v \in V = V_{\mathrm{E}} \cup V_{\mathrm{A}}$ proceeds as follows: if $v \in V_{\mathrm{E}}$ then Eve makes a choice of a successor, otherwise Adam chooses a successor; from this successor the same rule applies and the play goes on forever unless one of the parties cannot make a move. The player who cannot make a move looses. The result of an infinite play is an infinite path $v_0 v_1 v_2 \ldots$ This *path is winning* for Eve if it belongs to Acc. Otherwise Adam is the winner.

A *strategy* σ for Eve is a function assigning to every sequence of vertices $\vec{v}$ ending in a vertex v from V_{E} a vertex $\sigma(\vec{v})$ which is a successor of v. A *play respecting* σ is a sequence $v_0 v_1 \ldots$ such that $v_{i+1} = \sigma(v_i)$ for all i with $v_i \in V_{\mathrm{E}}$. The *strategy* σ *is winning for Eve* from a vertex v iff all the plays starting in v and respecting σ are winning. A *vertex is winning* if there exists a strategy winning from it. The strategies for Adam are defined similarly. A strategy is *positional* if it does not depend on the sequence of vertices that were played till now, but only on the present vertex. So such a strategy can be represented as a function $\sigma : V_{\mathrm{E}} \to V$ and identified with a choice of edges in the graph of the game.

In this paper the winning conditions $\mathrm{Acc} \subseteq V^\omega$ will be *regular conditions*. That is conditions defined in monadic second-order logic on sequences. An important special case is a *parity condition*. It is a condition determined by a function $\Omega : V \to \{0, \ldots, d\}$ in the following way:

$$\mathrm{Acc} = \{v_0 v_1 \ldots \in V^\omega : \limsup_{i\to\infty} \Omega(v_i) \ \text{ is even}\}$$

Hence, in this case, each position is assigned a natural number and we require that the largest among those appearing infinitely often is even. This condition was discovered by Mostowski [9] and is the most useful form of regular conditions. It guarantees existence of positional strategies [4, 10, 8]. It is closed by negation (the negation of a parity condition is a parity condition). It is universal in the sense that every game with a regular condition can be reduced to a game with a parity condition [9].

The main results about games that we need are summarized in the following theorem and uses results from [7, 4, 10].

Theorem 2.1. Every game with regular winning conditions is determined, i.e., every vertex is winning for one of the players. It is algorithmically decidable who is a winner from a given vertex in a finite game. In a parity game a player has a positional strategy winning from each of his winning vertices.

2.3 Automata

We shall need automata that work on graphs. These automata should cope with multiple outgoing edges with the same label. Moreover, we should like to equip them with tests of some simple structural graph properties. They will be able to check that an edge on a given letter is a self-loop or that the edges on two different letters lead to the same states. To incorporate all these tests it will be simpler to define automata which use a kind of modal formulas over the set of states in a process. Thus we start with defining these formulas.

Let A be an alphabet and let Q be a finite set. The set of *modal formulas* over A and Q, denoted $\mathcal{F}(A,Q)$, is the smallest set closed under the following rules:

- `tt`, `ff`, q, $\circlearrowleft_a$, $\overline{\circlearrowleft}_a$, $\downdownarrows_{a.b}$ $\overline{\downdownarrows}_{a,b}$ are formulas, for any $q \in Q$ and $a, b \in A$.
- $\alpha \vee \beta$ and $\alpha \wedge \beta$ are formulas for all $\alpha, \beta \in \mathcal{F}(A,Q)$.
- $\langle a \rangle \alpha$ and $[a]\alpha$ are formulas for all $\alpha \in \mathcal{F}(A,Q)$ and $a \in A$.

An *automaton* over a set of actions A is a tuple:

$$\mathcal{A} = \langle Q, A, q^0 \in Q, \delta : Q \to \mathcal{F}(A,Q), \mathrm{Acc} \subseteq Q^\omega \rangle$$

where Q is a finite set of states, A is a finite alphabet, and Acc is an accepting condition that is a regular set of infinite sequences of states.

The acceptance of a process P by an automaton $\mathcal{A}$ is defined in terms of strategies in a game $G(P, \mathcal{A})$ that we describe now. Let $\mathcal{F}^{\mathcal{A}}$ be the smallest set of formulas closed under taking subformulas, and containing all formulas in the range of δ together with `tt` and `ff`. We have

$$G(P, \mathcal{A}) = \langle V_{\mathrm{E}}, V_{\mathrm{A}}, T, \mathrm{Acc}_G \rangle$$

where

- $V_{\mathrm{E}} = S \times \mathcal{F}^{\mathcal{A}}_{\mathrm{E}}$, and $\mathcal{F}^{\mathcal{A}}_{\mathrm{E}}$ is the set of formulas form $\mathcal{F}^{\mathcal{A}}$ of one of the forms: $\mathtt{ff}, \circlearrowleft_a, \downdownarrows_{a,b}, q, \langle a \rangle \alpha,\ \alpha \vee \beta$.
- $V_{\mathrm{A}} = S \times \mathcal{F}^{\mathcal{A}} - V_{\mathrm{E}}$.
- From $(s, \mathtt{tt})$ and $(s, \mathtt{ff})$ no move is possible.
- From $(s, \circlearrowleft_a)$ there is a move to $(s, \mathtt{tt})$ if $\mathrm{out}_P(s, a) = \{s\}$ and to $(s, \mathtt{ff})$ otherwise.
- From $(s, \downdownarrows_{a,b})$ there is a move to $(s, \mathtt{tt})$ if $\mathrm{out}_P(s, a) = \mathrm{out}_P(s, b)$ and to $(s, \mathtt{ff})$ otherwise.
- Similarly for $(s, \overline{\circlearrowleft}_a)$ and $(s, \overline{\downdownarrows}_{a,b})$ but with roles of $(s, \mathtt{tt})$ and $(s, \mathtt{ff})$ interchanged.
- From $(s, \alpha \wedge \beta)$ as well as from $(s, \alpha \vee \beta)$ there are moves to (s, α) and to (s, β).
- From $(s, \langle a \rangle \alpha)$ and from $(s, [a]\alpha)$ there are moves to (t, α) for every $t \in \mathrm{out}(s, a)$.
- Finally, from (s, q) there is a move to $(s, \delta(q))$.
- The winning condition Acc_G contains sequences such that when looking only at the elements of Q appearing in the sequence we obtain an element of Acc.

We say that P *satisfies* $\mathcal{A}$, in symbols $P \vDash \mathcal{A}$, if Eve has a winning strategy in $G(P, \mathcal{A})$ from (s^0, q^0), which is the pair consisting from the initial states of P and $\mathcal{A}$, respectively. As our automata are very close to formulas we prefer to talk about satisfiability instead of acceptance. We shall still use some automata terminology though. For example, the *language recognized by an automaton* $\mathcal{A}$ will be the class of processes that satisfy $\mathcal{A}$.

Our automata are a variant of alternating automata. In particular the formulas used in the transition function are “closed” under disjunction and conjunction. Using standard constructions on alternating automata we get.

Proposition 2.2. The class of languages recognized by automata is closed under sum, intersection and complement.

This proposition allows to write $\mathcal{A} \wedge \mathcal{C}$ to denote an automaton which recognizes $L(\mathcal{A}) \cap L(\mathcal{C})$.

Definition 2.3. An automaton is called *simple* if formulas in its transition function use none of $\circlearrowleft_a, \overline{\circlearrowleft}_a, \downdownarrows_{a,b}, \overline{\downdownarrows}_{a,b}$.

A simple automaton is nothing else but a μ-calculus formula in a different presentation. Using the results on the μ-calculus we have:

Theorem 2.4 (Emerson-Jutla, [4])**.** It is decidable if for a given simple automaton $\mathcal{A}$ there is a process P such that $P \vDash \mathcal{A}$. Similarly, if we require P to be deterministic. In both cases a process P can be constructed if the answer is positive.

Theorem 2.5 (Niwiński-Janin-Walukiewicz, [11, 5])**.** Over deterministic systems which are trees, the expressive power of simple automata is equivalent to that of monadic second-order logic. Over all processes: a property is expressible by a simple automaton iff it is expressible in monadic second-order logic and bisimulation invariant.

3 Satisfiability

The basic question one can ask about our automata model is whether for a given automaton $\mathcal{A}$ there is a process that satisfies it. From the previous section we know that there is an algorithm answering this question in the case of simple automata. We shall now reduce the general case to that of simple automata. For this, we shall encode information about loops and parallel tests in additional transitions. This way for a process P we shall define a process Code(P). It will then turn out that behaviour of an automaton over P can be simulated by a behaviour of a simple automaton over Code(P).

A *type* of a state s of a process P is:

$$\mathrm{type}(s) = \{\circlearrowleft_a : \mathrm{out}(s,a) = \{s\}\} \cup \{\downdownarrows_{a,b} : \mathrm{out}(s,a) = \mathrm{out}(s,b)\}$$

Let Types(A) be the set of all possible types over an alphabet A.

Note that if $\tau \in \mathrm{Types}(A)$ and $\downdownarrows_{a,b} \in \tau$ then $\circlearrowleft_a \in \tau$ implies $\circlearrowleft_b \in \tau$, and also $\downdownarrows_{a,c} \in \tau$ implies $\downdownarrows_{b,c} \in \tau$.

Fix an alphabet A and some arbitrary ordering $<_A$ on it. For a process P over an alphabet A we define its code Code(P) over an alphabet $A \cup \mathrm{Types}(A)$. For each state s of P we do the following:

- Add a transition on action $\tau = \mathrm{type}(s)$, the target of this transition is some arbitrary fixed state (say, the initial state).

- Remove transitions on a if $\circlearrowleft_a \in \text{type}(s)$ or $\downdownarrows_{a,b} \in \text{type}(s)$ for some $b <_A a$.

Let $\mathcal{C}$ be a simple automaton expressing the conditions:

- For every state there is transition on exactly one letter from Types(A).
- For every $a \in A$, there is no transition on a if $\circlearrowleft_a \in \tau$ or $\downdownarrows_{a,b} \in \tau$ for some $b <_A a$.

Lemma 3.1. The process Code(P) satisfies $\mathcal{C}$ and has no loops $s \xrightarrow{a} s$ Moreover, Code(P) is deterministic if P is. If R is a process satisfying $\mathcal{C}$ without loops $s \xrightarrow{a} s$ then there is a unique process P such that Code(P) is isomorphic to R.

The next step is to transform an automaton over an alphabet A into an "equivalent" automaton over an alphabet $A \cup \text{Types}(A)$. Take an automaton

$$\mathcal{A} = \langle Q, A, q^0, \delta : Q \to \mathcal{F}(A, Q), \text{Acc} \subseteq Q^\omega \rangle$$

We first define transformation, Code(α), on formulas from $\mathcal{F}(A, Q)$:

- $\text{Code}(q) = q$.
- $\text{Code}(\circlearrowleft_a) = \bigvee\{\langle\tau\rangle\mathtt{tt} : \tau \in \text{Types}(A), \circlearrowleft_a \in \tau\}$. Similarly for $\downdownarrows_{a,b}$.
- $\text{Code}(\overline{\circlearrowleft}_a) = \bigvee\{\langle\tau\rangle\mathtt{tt} : \tau \in \text{Types}(A), \circlearrowleft_a \notin \tau\}$. Similarly for $\overline{\downdownarrows}_{a,b}$.
- $\text{Code}(\alpha \vee \beta) = \text{Code}(\alpha) \vee \text{Code}(\beta)$, and similarly for the conjunction.
- $\text{Code}(\langle a\rangle\alpha) = \bigvee\{\langle\tau\rangle\mathtt{tt} \wedge \text{Code}(\langle a\rangle\alpha, \tau) : \tau \in \text{Types}(A)\}$ where

$$\text{Code}(\langle a\rangle\alpha, \tau) = \begin{cases} \text{Code}(\alpha) & \text{if } \circlearrowleft_a \in \tau \\ \langle a\rangle\text{Code}(\alpha) \vee \bigvee\{\langle b\rangle\text{Code}(\alpha) : \downdownarrows_{a,b} \in \tau\} & \text{otherwise} \end{cases}$$

- $\text{Code}([a]\alpha) = \bigvee\{\langle\tau\rangle\mathtt{tt} \wedge \text{Code}([a]\alpha, \tau) : \tau \in \text{Types}(A)\}$; where the definition of $\text{Code}([a]\alpha, \tau)$ is as above but replacing $\langle a\rangle$ by $[a]$, $\langle b\rangle$ by $[b]$, and disjunctions by conjunctions.

Then automaton Code($\mathcal{A}$) is the same as $\mathcal{A}$ except for the transition function δ_{Code}. We put $\delta_{\text{Code}}(q) = \text{Code}(\delta(q))$. The following lemma follows directly from definitions.

Lemma 3.2. For every process P and automaton $\mathcal{A}$ over an alphabet A:

$$P \vDash \mathcal{A} \quad \text{iff} \quad \text{Code}(P) \vDash \text{Code}(\mathcal{A})$$

Observe that Code($\mathcal{A}$) is simple, i.e., does not use neither $\circlearrowleft$ nor $\downdownarrows$ constraints. Using Code($\mathcal{A}$) we can transfer results from simple automata to the general case.

Theorem 3.3. It is decidable if for a given automaton $\mathcal{A}$ there exist a process P such that $P \vDash \mathcal{A}$. Similarly, if we ask for P being deterministic. In both cases, if the answer is positive then a process satisfying $\mathcal{A}$ can be constructed.

Proof. Consider Code($\mathcal{A}$). As Code($\mathcal{A}$) $\wedge\, \mathcal{C}$ is a simple automaton, by Theorem 2.4 we can test if there exists a process $R \vDash$ Code($\mathcal{A}$) $\wedge\, \mathcal{C}$. Unfolding the loops of R we can construct a process R' without loops such that $R' \vDash$ Code($\mathcal{A}$) $\wedge\, \mathcal{C}$. Lemma 3.1 gives us a process P such that Code(P) is isomorphic to R', hence Code(P) $\vDash$ Code($\mathcal{A}$). By Lemma 3.2 we have $P \vDash \mathcal{A}$. This construction works also when we require P to be deterministic.

Conversely, if P is a (deterministic) process that satisfies $\mathcal{A}$ then the (deterministic) process Code(P) satisfies Code($\mathcal{A}$), by Lemma 3.2, and $\mathcal{C}$, by Lemma 3.1. Q.E.D.

4 Quotient for extended automata

In this section we present an operation that will permit us to reduce synthesis problems to the satisfiability problems. Consider an extended automaton $\mathcal{A} = \langle Q, A, q_0, \delta, \mathrm{Acc}\rangle$ and a process $P = \langle S, A, s^0, e\rangle$ over a common alphabet A. Our goal is to construct an automaton $\mathcal{A}/P$ such that for every process R:

$$R \vDash \mathcal{A}/P \quad \text{if and only if} \quad P \times R \vDash \mathcal{A}$$

We first define a division α/s for α a formula from $\mathcal{F}(A,Q)$, and s a state of P. The result is a formula from $\mathcal{F}(A, Q \times S)$:

$$\begin{aligned} q/s &= (q,s) & & \\ (\alpha \vee \beta)/s &= \alpha/s \vee \beta/s & (\alpha \wedge \beta)/s &= \alpha/s \wedge \beta/s \\ (\langle a\rangle\alpha)/s &= \langle a\rangle \bigvee\{\alpha/s' : s \xrightarrow{a} s'\} & ([a]\alpha)/s &= [a] \bigwedge\{\alpha/s' : s \xrightarrow{a} s'\} \end{aligned}$$

Now

$$\mathcal{A}/P = \langle Q \times S, A, (q^0, s^0), \delta_/ : Q \times S \to \mathcal{F}(Q \times S), \Omega\rangle$$

where $\delta_/(q,s) = \delta(q)/s$; recall that $\delta(q) \in \mathcal{F}(Q)$, so $\delta(q)/s \in F(Q \times S)$.

Theorem 4.1. Let A be an alphabet. For every extended automaton $\mathcal{A}$ and every process P, both over A, there is an automaton $\mathcal{A}/P$ such that for every process R over A:

$$R \vDash \mathcal{A}/P \quad \text{if and only if} \quad P \times R \vDash \mathcal{A}$$

Proof. Fix a process R. We examine the games $G_\times = G(P \times R, \mathcal{A})$ and $G_/ = G(R, \mathcal{A}/P)$. We want to show how a move of one of the players in $G_\times$ from a position of the form $((s,r),\alpha)$ can be mimicked by, possibly several, moves of the same player in $G_/$ from $(r, \alpha/s)$. For example, suppose that a position has the form $((s,r), \alpha \vee \beta)$ and that Eve chooses $((s,r),\alpha)$. From a position $(r, (\alpha \vee \beta)/s))$ this move can be mimicked by going to $(r, \alpha/s)$. Slightly more complicated is the case of $((s,r), \langle a \rangle \alpha)$. In this case Eve can choose $((s',r'),\alpha)$ for $s \xrightarrow{a} s'$ and $r \xrightarrow{a} r'$. From $(r, (\langle a \rangle \alpha)/s)$ this move can be mimicked by first choosing $(r', \bigvee\{\alpha/s'' : s \xrightarrow{a} s''\})$ and then $(r', \alpha/s')$; this is possible as $(\langle a \rangle \alpha)/s = \langle a \rangle \bigvee\{\alpha/s'' : s \xrightarrow{a} s''\}$. The cases of $\alpha \wedge \beta$ and $[a]\alpha$ are dual.

These observations show that any play in $G_\times$ can be mimicked in $G_/$, so the same player has a winning strategy from $((s^0,r^0),q^0)$ in $G_\times$ and from $(r^0,(q^0,s^0))$ in $G_/$.

Q.E.D.

5 Solving controller synthesis problems

Equipped with the operation of division we can reduce the control problem to the satisfiability problem.

5.1 Centralized control

As we have argued in the introduction, the centralized controller synthesis problem can be formulated as:

> For a given process P and two automata $\mathcal{A}$, $\mathcal{B}$ over an alphabet A, find R such that:
> $$P \times R \vDash \mathcal{A} \quad \text{and} \quad R \vDash \mathcal{B}$$

We denote by $\mathrm{Sol}(P, \mathcal{A}, \mathcal{B})$ the set of solutions to the problem. The following is a direct corollary of Theorem 4.1

Corollary 5.1. For every process R:

$$R \in \mathrm{Sol}(P, \mathcal{A}, \mathcal{B}) \text{ if and only if } R \vDash (\mathcal{A}/P) \wedge \mathcal{B}.$$

This means that solving a synthesis problem amounts to checking emptiness of the automaton $(\mathcal{A}/P) \wedge \mathcal{B}$. Theorem 2.1 then states that this problem is decidable both for the general, nondeterministic, case as well as for the case of deterministic processes.

5.2 Decentralized control

The decentralized controller synthesis problem is:

> For a given process P and automata $\mathcal{A}$, $\mathcal{B}_1, \dots, \mathcal{B}_n$ over an alphabet A, find $R_1, \dots, R_n$ such that:
> $$P \times R_1 \times \cdots \times R_n \vDash \mathcal{A} \quad \text{and} \quad R_i \vDash \mathcal{B}_i \text{ for all } i = 1, \dots, n$$

Thanks to Theorem 4.1 we can take $\mathcal{A}/P$ and remove P from the left hand side. This shows that we can as well consider the following simpler formulation of the problem where P is not mentioned.

> For a given automata $\mathcal{A}$, $\mathcal{B}_1, \dots, \mathcal{B}_n$ over an alphabet A, find $R_1, \dots, R_n$ such that:
>
> $$R_1 \times \cdots \times R_n \vDash \mathcal{A} \quad \text{and} \quad R_i \vDash \mathcal{B}_i \text{ for all } i = 1, \dots, n$$

This last problem was studied in [1, 2] in the case when $R_1, \dots, R_n$ are required to be deterministic. In particular the problem was shown decidable when all but one $\mathcal{B}_i$ are simple. We shall see later that the same problem is undecidable in the nondeterministic case. To better understand the decidable/undecidable borderline we propose a classification of decentralized synthesis problems with respect to restrictions on $R_1, \dots, R_n$.

A pair $(pt, st) \subseteq \{det, nondet\} \times \{simple, full\}$ describes requirements on processes and specifications. A choice of a number of components and a type for each component determines a distributed control problem:

> DS$(n, (pt_1, st_1), \dots, (pt_n, st_n))$:
>
> Given automata $\mathcal{A}$, $\mathcal{B}_1, \dots, \mathcal{B}_n$ such that $\mathcal{B}_i$ is simple if $st_i = simple$, find $R_1, \dots, R_n$ such that R_i is deterministic if $pt_i = det$, satisfying $P \times R_1 \times \cdots \times R_n \vDash \mathcal{A}$ and $R_i \vDash \mathcal{B}_i$ for all $i = 1, \dots, n$.

The following theorem gives a complete classification of these problems with respect to decidability.

Theorem 5.2. The problem DS$(\mathcal{A}, (pt_1, st_1), \dots, (pt_n, st_n))$ is decidable iff

- There is at most one i such that $st_i = full$.
- There is at most one j such that $pt_j = nondet$ and moreover $j \neq i$.

The proof of this theorem will be given in the two following sections. In terms of the decentralized control problem formulated at the beginning of the section, we get that the problem is decidable if at most one of $\mathcal{B}_i$ is not simple and at most one R_j is allowed to be nondeterministic (moreover $j \neq i$). Probably the most important difference with respect to the deterministic case considered in [1] is that now P can be nondeterministic.

6 The decidable sub-case of decentralized control

We should like to show the right to left implication of Theorem 5.2. The solution in the case when all $R_1, \dots, R_n$ are required to be deterministic uses the following extension of the quotient operation:

Theorem 6.1 (Arnold-Vincent-Walukiewicz, [1])**.** For every automaton $\mathcal{A}$ and every simple automaton $\mathcal{B}$ there is an automaton $\mathcal{A}/\mathcal{B}$ such that for every deterministic process P:

$$P \vDash \mathcal{A}/\mathcal{B} \quad \text{iff} \quad \exists R.\ R \text{ deterministic, } R \vDash \mathcal{B}, \text{ and } P \times R \vDash \mathcal{A}$$

Here we show existence of a variant of this division operation. The difference is that the existentially quantified process R need not be deterministic.

Theorem 6.2. For every automaton $\mathcal{A}$ and every simple automaton $\mathcal{B}$, there is an automaton $\mathcal{A}/_{\mathrm{ndet}}\mathcal{B}$ such that for every deterministic process P:

$$P \vDash \mathcal{A}/_{\mathrm{ndet}}\mathcal{B} \quad \text{iff} \quad \exists R.\ R \vDash \mathcal{B} \text{ and } P \times R \vDash \mathcal{A}$$

Before giving the proof of this theorem let us show how it can be used to prove right to left direction of Theorem 5.2.

Let us assume that $st_n = \mathit{full}$ and $pt_1 = \mathit{nondet}$. First, we find a deterministic process $R_n \vDash (\mathcal{A}/_{\mathrm{ndet}}\mathcal{B}_1/\mathcal{B}_2 \dots /\mathcal{B}_{n-1}) \wedge \mathcal{B}_n$. If none exists then the problem has no solutions. Otherwise, by Theorem 3.3, we can construct required R_n. Equipped with it we can find a deterministic process $R_{n-1} \vDash (\mathcal{A}/R_n/_{\mathrm{ndet}}\mathcal{B}_1/\mathcal{B}_2 \dots /\mathcal{B}_{n-2}) \wedge \mathcal{B}_{n-1}$. This construction can be repeated, giving $R_{n-1} \dots$, until we construct a deterministic $R_2 \vDash (\mathcal{A}/R_n/R_{n-1}/\dots/R_3/_{\mathrm{ndet}}\mathcal{B}_1) \wedge \mathcal{B}_2$. Once $R_2, \dots, R_n$ are fixed, we can look for, this time nondeterministic, process $R_1 \vDash (\mathcal{A}/R_n/R_{n-1}/\dots/R_2) \wedge \mathcal{B}_1$. By the above two theorems on division operations, $R_1, \dots, R_n$ is a solution to the problem. The theorems also guarantee that all solutions to the problem can be obtained this way.

The rest of this section presents the proof of Theorem 6.2. We want to transform the property of a deterministic process P:

$$\exists R. \quad R \vDash \mathcal{B} \quad \text{and} \quad P \times R \vDash \mathcal{A} \tag{1.1}$$

to an equivalent formulation that is expressible by an automaton. This will be our automaton $\mathcal{A}/\mathcal{B}$.

The first step is to introduce well-typed processes and restrict our problem only to this kind of processes. Given a process P over an alphabet A, a *well-typed* process, $\mathrm{wt}(P)$, is a process over the alphabet $A \cup \mathcal{P}(A)$ that is obtained form P by adding a new state $\top$, and precisely one action from each state as follows: to a state s of P we add a transition to $\top$ on $\mathrm{out}^P(s) \in \mathcal{P}(A)$, where $\mathrm{out}^P(s)$ is the set of actions possible form s, $\mathrm{out}^P(s) = \{b : \mathrm{out}^P(s,b) \neq \varnothing\}$. It should be clear that there is an automaton checking if a processes is of the form $\mathrm{wt}(P)$. It is also easy, given an automaton $\mathcal{C}$, to construct an automaton $\mathcal{C}'$ such that for all processes P over an alphabet A

$$\mathrm{wt}(P) \vDash \mathcal{C} \quad \text{iff} \quad P \vDash \mathcal{C}'$$

This means that in the following we can consider only processes of the form $\mathrm{wt}(P)$. We call these processes *well-typed.*

The restriction to well-typed processes is important for the first simplification step. We want to find an automaton $\mathcal{D}$ such that (1.1) is equivalent to

$$\exists R. \quad P \times R \vDash \mathcal{D} \tag{1.2}$$

For this we construct an automaton $\mathcal{B}'$ and show that (1.1) is equivalent to $\exists R'. \, P \times R' \vDash \mathcal{B}' \wedge P \times R' \vDash \mathcal{A}$. Having this, we can just take $\mathcal{B}' \wedge \mathcal{A}$ for $\mathcal{D}$. We call a process over $A \cup \mathcal{P}(A)$ *typed* if every state has precisely one transition on a letter from $\mathcal{P}(A)$. Compared with well-typed processes, we do not put any restriction what a γ is. We also define a *safe extension* of a typed process R to be a process obtained form R by adding some states and transitions provided that if (s, b, t) is an added transition and s is a state from R then t must be an added state and b must not appear in the label of the unique action from $\mathcal{P}(A)$ possible from s. With these definitions we can say what the automation $\mathcal{B}'$ is. We want $\mathcal{B}'$ to accept a process if it is typed, and moreover it has a safe extension that is accepted by $\mathcal{B}$. It remains to argue that $\mathcal{B}'$ has the desired property. For one direction suppose that we have R' with $P \times R' \vDash \mathcal{B}'$ and $P \times R' \vDash \mathcal{A}'$. If $P \times R' \vDash \mathcal{B}'$ then, by the definition of $\mathcal{B}'$, there is a safe extension R of $P \times R'$ that satisfies $\mathcal{B}$. By the definition of the safe extension, and the fact that P is well-typed we have that $P \times R' = P \times R$. So $P \times R \vDash \mathcal{A}$. Now consider the opposite direction. Take R which is assumed to exists and add to R a state $\top$ as well as transitions to $\top$ from each state of R on every letter from $\mathcal{P}(A)$. As $\mathcal{B}$ does not talk about the actions from $\mathcal{P}(A)$ then $R' \vDash \mathcal{B}$. We have $P \times R' \vDash \mathcal{B}'$ because $P \times R'$ is typed and R' is a safe extension of $P \times R'$. We also have $P \times R' \vDash \mathcal{A}$ as $\mathcal{A}$ does not talk about actions from $\mathcal{P}(A)$.

The above argument reduces our task to the problem of expressing by an automaton the property (1.2) of well-typed P. First, we shall consider a simpler property where the branching of the process R we quantify over is bounded by k, i.e. for every $s \in R$ and a, $|\mathrm{out}(s, a)| \leq k$.

$$\exists R. \quad \mathrm{branching}(R) \leq k \quad \text{and} \quad P \times R \vDash \mathcal{D} \tag{1.3}$$

This formulation will allow us to use the division operation for the deterministic case, i.e, Theorem 6.1. Consider processes over an alphabet $A_{[k]} = A \times \{1, \dots, k\}$. A deterministic process P' over an alphabet $A_{[k]}$ represents a nondeterministic process $\mathrm{red}(P')$ over an alphabet A where each action (a, i), for $i = 1, \dots, k$, is mapped to a. Every nondeterministic process of branching bounded by k can be represented in such a way (in general not uniquely). From automaton $\mathcal{D}$ it is easy to construct an automaton $\mathcal{D}_{[k]}$ which accepts a process P' over $A_{[k]}$ iff $\mathrm{red}(P')$ is accepted by $\mathcal{D}$. Consider

$\mathcal{D}_{[k]}/\mathtt{tt}$ where $\mathtt{tt}$ is an automaton accepting all the processes over $\mathcal{A}_{[k]}$. By Theorem 6.1 we have

$$P' \vDash \mathcal{D}_{[k]}/\mathtt{tt} \quad \text{iff} \quad \exists R'.\ P' \times R' \vDash \mathcal{D}_{[k]};$$

Here, all the processes are over $A_{[k]}$. For a deterministic process P over A we can define $P_{[k]}$ to be a deterministic process over $A_{[k]}$ where there is an edge (b, i), for $i = 1, \ldots, k$, between two nodes iff in P there is an edge b between them. For an automaton $\mathcal{D}'$ over $A_{[k]}$ is easy to construct an automaton $\mathrm{red}(\mathcal{D}')$ such that for all deterministic processes P over A

$$P \vDash \mathrm{red}(\mathcal{D}') \quad \text{iff} \quad P_{[k]} \vDash \mathcal{D}'$$

With this we get

$$P \vDash \mathrm{red}(\mathcal{D}_{[k]}/\mathtt{tt}) \quad \text{iff} \quad P_{[k]} \vDash \mathcal{D}_{[k]}/\mathtt{tt} \quad \text{iff} \quad \exists R'.\ P_{[k]} \times R' \vDash \mathcal{D}_{[k]}$$

where R' and $P_{[k]}$ are over the alphabet $A_{[k]}$. By definition, the last formula is equivalent to $\exists R'.\mathrm{red}(P_{[k]} \times R') \vDash \mathcal{D}$. As P is deterministic $\mathrm{red}(\ P_{[k]} \times R') = P \times \mathrm{red}(R')$. It is easy to see that (1.3) is equivalent to $\exists R'.\ P \times \mathrm{red}(R') \vDash \mathcal{D}$ and in consequence to $P \vDash \mathrm{red}(\mathcal{D}_{[k]}/\mathtt{tt})$. So, for $\mathcal{A}/_{\mathrm{ndet}}\mathcal{B}$ we could take $\mathrm{red}(\mathcal{D}_{[k]}/\mathtt{tt})$ if only we could find a bound on k.

We are left to show that we can bound the branching in our problem (1.2), so that we can fix k. The following proposition gives the desired bound.

Lemma 6.3. Let P be a deterministic process and let $\mathcal{A}$ be an automaton with parity acceptance conditions. If there is (possibly nondeterministic) process R such that:

$$P \times R \vDash \mathcal{A}$$

then there is R' with the same property which has the branching degree $|A||\mathcal{A}|$

Proof. Take R such that $P \times R \vDash \mathcal{A}$. Then Eve has a positional winning strategy (cf. Theorem 2.1) in the game $G(P \times R, \mathcal{A})$. This strategy is a function $\sigma : (P\times R)\times\mathcal{F}^{\mathcal{A}}_{\mathrm{E}} \to (P\times R)\times\mathcal{F}^{\mathcal{A}}$ which to pairs of the form $(s, \alpha\vee\beta)$ assigns either (s, α) or (s, β); and to pairs of the form $(s, \langle b\rangle\alpha)$ assigns a pair (s', α) for some $s' \in \mathrm{out}_{P\times R}(s, b)$. This function has the property that all the plays respecting suggestions of this function are winning for Eve.

Take some state s of $P \times R$. Let $\mathrm{us}(s, b)$, the set of useful successors, be the set of all successors t of s such that $(t, \alpha) = \sigma(s, \langle b\rangle\alpha)$ for some formula $\langle b\rangle\alpha$. Because the number of formulas of this kind is bounded by the size of $\mathcal{A}$, so is the size of $\mathrm{us}(s, b)$.

The intention is that we should like to prune $P \times R$ so that on action b from s only edges to $\mathrm{us}(s, b)$ remain. This may not be correct as the following example shows. Suppose that $\mathrm{us}(s,b) = \mathrm{us}(s,c)$, while $\mathrm{out}_{P\times R}(s,b) \neq \mathrm{out}_{P\times R}(s,c)$. Now, the result of $\downdownarrows_{b,c}$ test will be different in $P \times R$ and in the pruned structure. Hence, it may happen that $\mathcal{A}$ does not accept the pruned structure.

In order to avoid the problem mentioned in the above paragraph we extend $\mathrm{us}(s,b)$ to $\mathrm{us}'(s,b)$. For every state s and action b, let $\mathrm{us}'(s,b)$ be a set satisfying the following.

- $\mathrm{us}(s,b) \subseteq \mathrm{us}'(s,b)$.
- if $s \models \circlearrowleft_b$ then $s \in \mathrm{us}'(s,b)$.
- if $s \models \overline{\circlearrowleft}_b$ then either $\mathrm{us}'(s,b') = \varnothing$, or $s' \in \mathrm{us}'(s,b')$ for some $s' \neq s$ and $s' \in \mathrm{out}_{P\times R}(s,b)$.
- if $s \models \downdownarrows_{b,c}$ then $\mathrm{us}'(s,b) = \mathrm{us}'(s,c)$.
- if $s \models \overline{\downdownarrows}_{b,c}$ and $\mathrm{out}_{P\times R}(s,b) \not\subseteq \mathrm{out}_{P\times R}(s,c)$ then $s' \in \mathrm{us}'(s,b)$ for some arbitrary chosen $s' \in \mathrm{out}_{P\times R}(s,b) - \mathrm{out}_{P\times R}(s,c)$.

It is easy to see that $\mathrm{us}'(s,b)$ can be chosen in such a way that it is at most $|A|$-times bigger than $\mathrm{us}(s,b)$.

Now take $P \times R$ and delete all edges (s,b,t) such that $t \not\in \mathrm{us}'(s,b)$. Let us call the resulting process R'. In R', strategy σ is still a winning strategy because we have only limited choices for Adam. Hence, Eve wins in $G(R',\mathcal{A})$, and in consequence $R' \models \mathcal{A}$. We have that $P \times R' \models \mathcal{A}$, as $P \times R' = R'$, since P is deterministic. By construction, the branching of R' is bounded by the maximal possible size of $\mathrm{us}'(s,b)$ which is $|A||\mathcal{A}|$.

Q.E.D.

Remark 6.4. If the restriction of determinism of P is dropped than the division $\mathcal{A}/_{\mathrm{ndet}}\mathcal{B}$ does not exist even when $\mathcal{A}$ and $\mathcal{B}$ are simple. For example, take $\mathcal{A}$ which says that all maximal paths are of the form a^*b, and if a state has an successor on action a then it does not have one on action b. Consider $\mathcal{A}/_{\mathrm{ndet}}\mathcal{A}$. Condition $P \models \mathcal{A}/_{\mathrm{ndet}}\mathcal{A}$ means that there is R such that $P \times R \models \mathcal{A}$ and $R \models \mathcal{A}$. If P had two paths a^ib and a^jb of different length then in $P \times R$ we would have a path that does not finish with b. This implies that $P \models \mathcal{A}/_{\mathrm{ndet}}\mathcal{A}$ iff there is k such that all the paths in P have the form a^kb. So the set of processes satisfying $\mathcal{A}/_{\mathrm{ndet}}\mathcal{A}$ is not regular. Observe that in this argument it did not matter whether we restrict to R being deterministic or not.

Remark 6.5. Even when restricting to deterministic processes, automaton $\mathcal{A}/\mathcal{B}$ may not exist if $\mathcal{B}$ is not simple. In [1] it is shown that decentralized control problem is undecidable for $n = 2$ if both $\mathcal{B}_1$ and $\mathcal{B}_2$ are automata with $\circlearrowleft_a$ constraints. In [2] undecidability is shown when both automata used $\downdownarrows_{a,b}$ constraints, or when one uses $\circlearrowleft_a$ constrains and the other $\downdownarrows_{a,b}$ constraints.

7 Undecidable cases of decentralized control

In this subsection we show left to right direction of Theorem 5.2. It will be enough to study the version of the control problem for two processes:

> (ABC) Given automata $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ over the same action alphabet A, do there exist, possibly nondeterministic, processes P, R such that
>
> $$P \vDash \mathcal{A},\ R \vDash \mathcal{B} \text{ and } P \times R \vDash \mathcal{C}.$$

First, we shall show that the problem is undecidable even when $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are simple automata. This will give the proof of Theorem 5.2 for the case when there are at least two processes that can be nondeterministic. Next, we shall consider the case when at most one of the processes can be nondeterministic. We shall show that the above problem is undecidable when only R can be nondeterministic, and when $\mathcal{B}$ can use either $\circlearrowleft$ constraints or $\downdownarrows$ constrains. This not only will imply the remaining part of Theorem 5.2, but will also show that restricting our automata uniquely to $\circlearrowleft$ constraints or to $\downdownarrows$ constraints does not change the decidability classification.

Before showing these results we should like to introduce a syntactic extension of our setting which will make the presentation easier. We shall suppose that we have propositional letters labelling states of processes. So each process comes not only with an alphabet of actions but also with an alphabet Λ of propositions:

$$P = \langle A, \Lambda, S, s^0, e \subseteq S \times A \times S, \lambda : S \to \Lambda \rangle$$

Automata are also extended to reflect this, so the transition function can test what is a label of the current state:

$$\mathcal{A} = \langle Q, A, \Lambda, q^0, \delta : Q \times \Lambda \to F(A, Q), \mathrm{Acc} \subseteq Q^\omega \rangle$$

There are many possible definitions of a product of two processes with state labels. Here we choose the one that will suit our needs. Given two processes over the same action alphabet, but possibly different proposition alphabets:

$$P = \langle A, \Lambda_P, S_P, s_P^0, e_p, \lambda_P \rangle \qquad R = \langle A, \Lambda_R, S_R, s_R^0, e_R, \lambda_R \rangle$$

we define their product as:

$$P \otimes R = \langle A, \Lambda_P \times \Lambda_R, S_P \times S_R, (s_P^0, s_R^0), e_\otimes, \lambda_\otimes \rangle$$

where $\lambda_\otimes(s_P, s_R) = (\lambda_P(s_P), \lambda_R(s_R))$ and, as before, $((s_p, s_R), a, (s'_P, s'_R)) \in e_\otimes$ iff $(s_P, a, s'_P) \in e_P$ and $(s_R, a, s'_R) \in e_R$.

It is quite straightforward to see how to simulate propositional letters by actions. One can add propositional letters to the action alphabet and require that from each state there is a transition on exactly one propositional letter; the target of this transitions is of no importance.

The problem with this coding is that the standard product does not reflect our $\otimes$-product. In order to recover the $\otimes$-product, we first make the alphabets Λ_P and Λ_R disjoint. Let $\widehat{P}$, $\widehat{R}$ denote respective plants with encoding of propositions as described above. We add to every state of $\widehat{P}$ an action on every letter from Λ_R and to every state of $\widehat{R}$ an action on every letter of Λ_P. This way we have that $\widehat{P} \times \widehat{R}$ is the encoding of $P \otimes R$: from every state of $\widehat{P} \times \widehat{R}$ we have a successor on exactly one letter from Λ_P and on one letter from Λ_R.

After these remarks it should be clear that instead of the problem (ABC) we can consider the problem (ABC$_\otimes$) where the processes are allowed to have propositions and $\otimes$ is used in place of ordinary product.

(ABC$_\otimes$) Given automata $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ over the same action alphabet A, and over proposition alphabets Λ_a, Λ_b and $\Lambda_a \times \Lambda_b$ respectively, do there exist processes P, R such that

$$P \vDash \mathcal{A},\ R \vDash \mathcal{B} \text{ and } P \otimes R \vDash \mathcal{C}.$$

Thus, the following proposition implies the undecidability of the problem (ABC).

Proposition 7.1. The problem (ABC$_\otimes$) is undecidable.

Proof. We shall present a reduction of the halting problem. Let us fix a deterministic Turing machine together with an alphabet Γ needed to encode its configurations. We write $c \vdash c'$ to say that a configuration c' is a successor of a configuration c. Without a loss of generality we assume that the machine loops from the accepting configuration. We shall use just one action letter, so we shall not mention it in the description below. The alphabet of propositions will contain Γ and special symbols: l and $\#$. The nodes labelled by l will be called *l-nodes*; similarly for $\#$-nodes, and γ-nodes for $\gamma \in \Gamma$. We shall say that a node is a *Γ-node*, if it is a γ-node for some γ. We shall also talk about an l-successor of a node, this a successor that is an l-node. Finally, when we shall say that there is a path $\gamma_1 \dots \gamma_n$ in a process, this would mean that there is a sequence of nodes, that is a path, and such that the propositional letters associated to nodes form the sequence $\gamma_1 \cdots \gamma_n$.

We want to construct $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ so that the problem (ABC$_\otimes$) has a solution iff the machine accepts when started from the empty tape. Consider the following three conditions that will be used for specifications $\mathcal{A}$ and $\mathcal{B}$:

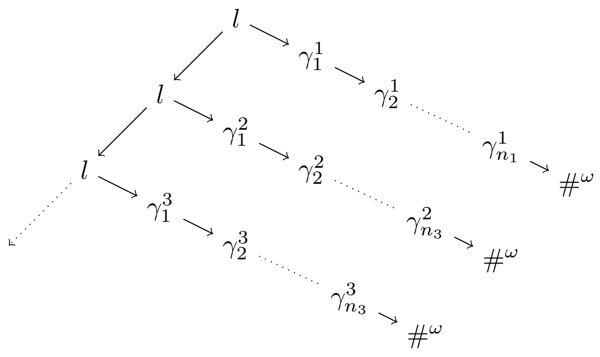


FIGURE 1. Intended shape of a process satisfying **AB1**, **AB2**, **AB3**.

AB1 Every l-node has an l-successor and a Γ-successor. Every Γ-node has either only Γ-nodes or only $\#$-nodes as successors.

AB2 From every Γ-node, every path reaches a $\#$-node.

AB3 Every $\#$-node has only $\#$-nodes as successors.

The intended shape of a process satisfying these conditions is presented in Figure 1. These conditions do not imply that the shape is exactly as presented in the figure. For example, they do not guarantee that there is only one infinite path labelled with l.

The constraints on the product of two processes are listed below. They are formulated in terms of the product alphabet.

C1 Every (l,l)-node has an (l,l)-successor and an (γ,γ)-successor for some $\gamma \in \Gamma$. Moreover all its successors are labelled either by (l,l), (l,γ), (γ,l) or (γ,γ).

C2 Every maximal path starting with $(l,l)^i(\gamma,\gamma)$ has a form $(l,l)^i\Delta^+(\#,\#)^\omega$ where $\Delta = \{(\gamma,\gamma) : \gamma \in \Gamma\}$.

C3 Every maximal path that starts with $(l,l)^i(\gamma_1,l)(\gamma_2,\gamma_1')$ for some γ_1,γ_2, $\gamma_1' \in \Gamma$ has the form: $(l,l)^i(\gamma_1,l)(\gamma_2,\gamma_1')\dots(\gamma_k,\gamma_{k-1}')(\#,\gamma_k)(\#,\#)^\omega$. Moreover $\gamma_1\dots\gamma_k \vdash \gamma_1'\dots\gamma_k'$, or the two are identical if $\gamma_1\dots\gamma_k$ is an accepting configuration.

C4 For every path labelled $(l,l)(\gamma_1,\gamma_2)\dots(\gamma_k,\gamma_k)(\#,\#)^\omega$, the sequence $\gamma_1\dots\gamma_k$ represents the initial configuration for the Turing machine. An accepting state of the machine appears in the tree.

Let $\mathcal{C}$ be the automaton expressing the conjunction of the conditions C1-C4. We claim that with this choice of automata the problem $(\mathrm{ABC}_{\otimes})$ has a solution iff the Turing machine halts on the initial configuration.

We first consider an easy direction. Suppose that the Turing machine halts on the initial configuration. Then we construct P and R as in the Figure 1, where for every i the sequence of Γ letters after l^i is the i-th configuration of the machine (we assume that all configurations are represented by words of the same length). This way P and R satisfy conditions AB1-3. It is straightforward to verify that $P \otimes R$ satisfies the conditions C1-C4.

For the other direction, suppose that P and R are as required. We shall show that the machine has an accepting computation from the initial configuration.

First, we show that the conditions we have imposed limit very much possible nondeterminism in P and R. Take any n and a path labelled $l^n\gamma_1 \dots \gamma_{k_n}\#^\omega$ in P as well as a path labelled $l^n\gamma'_1 \dots \gamma'_{m_n}\#^\omega$ in R. These paths exist by conditions AB1-AB3. In $P \times R$ these two paths give a path that starts with $(l,l)^n(\gamma_1,\gamma'_1)$. The condition AB1 implies that $\gamma_1 = \gamma'_1$. Consequently, the condition AB2 implies that $k_n = m_n$ and $\gamma_i = \gamma'_i$ for all $i = 1,\dots,k_n$. This allows us to define $u_n = \gamma_1 \dots \gamma_{k_n}$. To summarize, in P all paths of the form $l^n\Gamma^+\#^\omega$ have the same labels: $l^n u_n \#^\omega$. Similarly for paths in R.

It remains to show that u_n is the n-th configuration of the computation of the Turing machine. By condition A3, we know that u_1 is the initial configuration. Consider now a path in $P \otimes R$ labelled with

$$(l,l)^n(\gamma_1,l)(\gamma_2,\gamma'_1)\dots(\gamma_k,\gamma'_{k-1})(\#,\gamma'_k)(\#,\#)^\omega$$

This path exists as it is a product of a path in P starting with $l^n\gamma_1$ and a path in R starting with $l^{n+1}\gamma'_1$. We have that $u_n = \gamma_1 \dots \gamma_k$ and $u_{n+1} = \gamma'_1 \dots \gamma'_k$. By the condition A2 we get $u_n \vdash u_{n+1}$. Q.E.D.

This finishes the case of Theorem 5.2 when at least two processes can be nondeterministic. It remains to consider the case when only one of the processes, say R can be nondeterministic, and when specification $\mathcal{B}$ of R is not simple. We shall show that in this case the problem is undecidable even if $\mathcal{B}$ uses uniquely $\downdownarrows$ constraints, or uniquely $\circlearrowleft$ constraints. Recall that the problem is decidable if $\mathcal{B}$ is simple, i.e. uses neither $\downdownarrows$ nor $\circlearrowleft$.

Proposition 7.2. The problem $(\mathrm{ABC}_{\otimes})$ is undecidable if P is required to be deterministic but R may be nondeterministic and moreover a specification for R may use constraints $\downdownarrows$.

The reduction is very similar to the previous one. We just need to replace nondeterminism with appropriate use of $\downdownarrows$. This time our processes will be

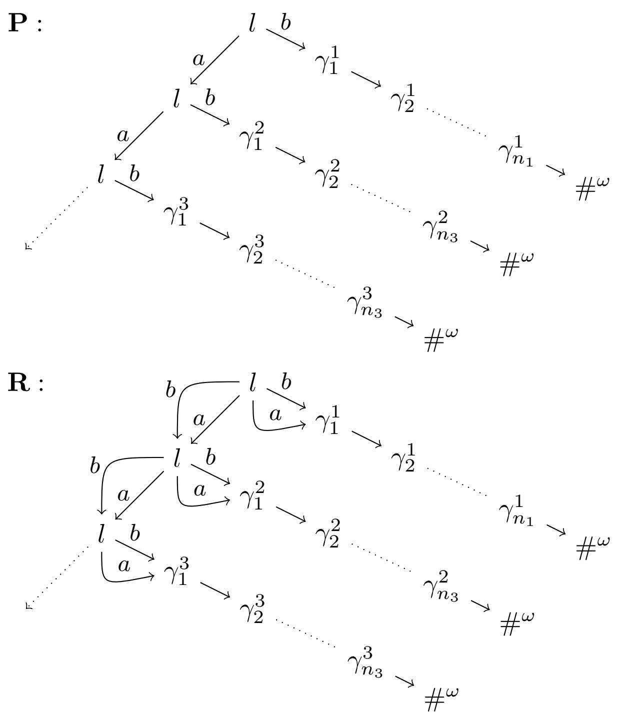


FIGURE 2. $\downarrow\downarrow$ constraints. Intended shapes of P and R.

over the alphabet of two actions $\{a, b\}$. The intended shapes of processes P and R are shown in the Figure 2.

The shape of P is almost the same as in the previous construction, but as P needs to be deterministic, some a transitions have to be changed to b transitions. Process R has almost the same structure as P but it is nondeterministic, and each a transition has a b transition in parallel.

Looking at $P \otimes R$ we get almost exactly the same structure as in the case of nondeterministic processes. The fact that process P is deterministic and that the two transitions from an l-node of P have different actions is compensated by the fact that a and b transitions have the same targets in R.

The formalization of the constraints and the proof of the proposition is almost the same as in case of Proposition 7.1. The following proposition treats the remaining case of $\circlearrowleft$ constraints.

Proposition 7.3. The problem (ABC$_\otimes$) is undecidable when P is required to be deterministic but R may be nondeterministic and moreover a specification for R may use looping constraints $\circlearrowleft$.

Proof. Consider an instance of the Post correspondence problem:

$$\{(u_1, v_1), \dots, (u_k, v_k)\};$$

where all u_i, v_i are words over an alphabet Σ. Let $D = \{1, \dots, k\}$ stand for the alphabet of indices. As an alphabet of actions we take $A = \Sigma \cup D \cup \{\alpha_1, \alpha_2, \beta, \#\}$, with an assumption that the last four actions do not appear in $\Sigma \cup D$.

The specification $\mathcal{A}$ for P will require that

A1 Every state, except the root, has only one successor. The root has successors on α_1 and α_2.

A2 There is a maximal path of the form $\alpha_1\beta i_1 u_{i_1} \dots i_m u_{i_m}\#$ for some $i_1, \dots, i_m \in D$.

A3 There is a maximal path of the form $\alpha_2\beta j_1 v_{j_1} \dots j_m v_{j_m}\#$ for some $j_1, \dots, j_m \in D$.

Observe that together with requirement that P is deterministic, the first condition implies that P has exactly two maximal paths. The shape of P is presented in Figure 3.

The specification $\mathcal{B}$ for R will require that:

B1 The root has loops on actions α_1 and α_2 and some transitions on β.

B2 There is a path from the root of the form $\beta\Sigma^*\#$. Every node on this path except the root has loops on all actions from D and has a successor on at most one action from $\Sigma \cup \{\#\}$.

B3 There is a path from the root of the form $\beta D^*\#$. This time every node except the root has loops on actions from Σ and a successor on at most one action from $D \cup \{\#\}$.

The intended shape of a process satisfying $\mathcal{B}$ is presented in Figure 3. Observe that we cannot force this process to be deterministic.

The specification $\mathcal{C}$ for $P \times R$ will require that all the paths are finite and that the last action on every path is $\#$.

We claim that with this choice of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, the problem (ABC) has a solution iff the instance of the Post correspondence problem has a solution.

For the right-to-left direction, take a solution $i_1, \dots, i_m$ to the correspondence problem. We construct P that has two paths: $\alpha_1\beta i_1 u_{i_1} \dots i_m u_{i_m}\#$ and $\alpha_2\beta i_1 v_{i_1} \dots i_m v_{i_m}\#$. As R we take a process as depicted in Figure 3

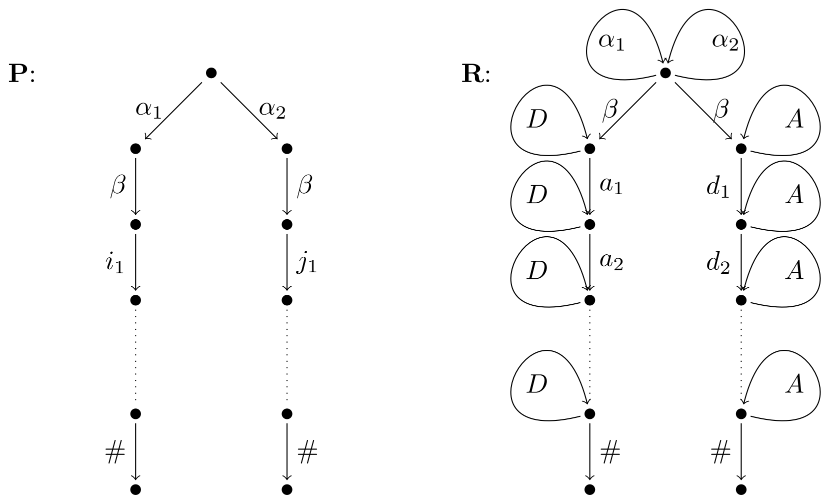


FIGURE 3. ↻-constraints. Intended shapes of P and R.

where the path satisfying condition B2 has the form $\beta u_{i_1} \dots u_{i_m}\#$, and the path satisfying B3 is $\beta i_1 \dots i_m \#$. It is easy to see that $P \times R$ satisfies $\mathcal{A}$.

For the direction from left to right suppose that P and R are a solution to the problem. Consider a path of R labelled $\beta\Sigma^*\#$ satisfying B2 and the path $\alpha_1\beta i_1 u_{i_1} \dots i_m u_{i_m}\#$ of P as required by the condition A2. Recall that there are loops on α_1 and α_2 in the root of R. This means that the two paths synchronize, at least at the beginning. The only way that the synchronization can continue until $\#$ is that $u_{i_1} \dots u_{i_m}$ is exactly the labelling of the path in R. We can use the same argument for the path $\alpha_2\beta j_1 v_{j_1} \dots j_m v_{j_n}\#$ and in consequence we get $u_{i_1} \dots u_{i_m} = v_{j_1} \dots v_{j_n}$. If we now repeat this argument once again but with a path of R labelled with $\beta D^*\#$ as required by condition B3 then we shall get that $i_1 \dots i_m = j_1 \dots j_n$. This finishes the proof. Q.E.D.

We can now summarize how the three propositions of this subsection can be used to show left to right implication of Theorem 5.2. If two of the processes R_i are allowed to be nondeterministic then the undecidability follows from Proposition 7.1. The case when there are two automata that are not simple but all processes are deterministic was proven in [1] for ↻ constraints and in [2] for ⇊ constraints, and a mix of ↻ and ⇊ constraints. If a specification can use either ↻ or ⇊ constraints and the corresponding process can be nondeterministic then undecidability follows from Propositions 7.2 and 7.3, respectively.

8 Conclusions

In this paper we have studied the controller synthesis problem for nondeterministic plants and controllers. We have seen that going from deterministic to nondeterministic plants does not change the complexity of the problem. Allowing nondeterministic controllers is more delicate. It can be done in centralized case, but in the decentralized case at most one controller can be nondeterministic, moreover it should be able to observe all actions of the plant.

Let us briefly comment on the complexity of the constructions presented here. The operation of division by a process gives an exponential blow-up. It is unavoidable for the same reason as in a translation from alternating to nondeterministic automaton. The complexity of the construction for division by automaton is also exponential.

Given the results above one can ask whether they also apply to the setting of architectures of Pnueli and Rosner [12]. It is quite simple to encode this latter setting into our setting using unobservable actions. Thus all decidability results in our setting transfer to architecture setting. As for undecidability results, one can show by methods very similar to those used in this paper that even two element pipeline becomes undecidable when specifications for controllers are allowed to be nondeterministic.

References

[1] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theor. Comput. Sci.*, 1(303):7–34, 2003.

[2] X. Briand. *Contrôle avec événements indiscernables et inobservables.* PhD thesis, Bordeaux University, 2006.

[3] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems.* The Kluwer International Series on Discrete Event Dynamic Systems, 11. Kluwer Academic Publishers, Boston, MA, 1999.

[4] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377. IEEE, 1991.

[5] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic.

In U. Montanari and V. Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996.

[6] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems.* Kluwer Academic Publishers, Norwell, USA, 1995.

[7] D. A. Martin. Borel determinacy. *Ann. of Math. (2)*, 102(2):363–371, 1975.

[8] R. McNaughton. Infinite games played on finite graphs. *Ann. Pure Appl. Logic*, 65(2):149–184, 1993.

[9] A. W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation theory (Zaborów, 1984)*, volume 208 of *Lecture Notes in Comput. Sci.*, pages 157–168, Berlin, 1985. Springer.

[10] A. W. Mostowski. Games with forbidden positions. Technical Report 78, University of Gdansk, 1991.

[11] D. Niwiński. Fixed points vs. infinite generation. In *LICS*, pages 402–409. IEEE Computer Society, 1988.

[12] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

[13] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77(1):81–98, Jan. 1989.

Reachability in continuous-time Markov reward decision processes*

Christel Baier [1]
Boudewijn R. Haverkort[2]
Holger Hermanns[3]
Joost-Pieter Katoen[4]

[1] Faculty of Computer Science
Technical University of Dresden
01062 Dresden, Germany
baier@tcs.inf.tu-dresden.de

[2] Department of Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede, The Netherlands
brh@cs.utwente.nl

[3] Department of Computer Science
Saarland University
66123 Saarbrücken, Germany
hermanns@cs.uni-sb.de

[4] Department of Computer Science
RWTH Aachen University
52056 Aachen, Germany
katoen@cs.rwth-aachen.de

Abstract

Continuous-time Markov decision processes (CTMDPs) are widely used for the control of queueing systems, epidemic and manufacturing processes. Various results on optimal schedulers for discounted and average reward optimality criteria in CTMDPs are known, but the typical game-theoretic winning objectives have received scant attention so far. This paper studies various sorts of reachability objectives for CTMDPs. The central result is that for any CTMDP, reward reachability objectives are dual to timed ones.

1 Introduction

Having their roots in economics, Markov decision processes (MDPs, for short) in computer science are used in application areas such as randomised

* We thank Gethin Norman (Oxford) for his comments on an earlier version of this paper. This work has been partially funded by the bilateral NWO-DFG Project Validation of Stochastic Systems 2 (VOSS2).

distributed algorithms and security protocols. The discrete probabilities are used to model random phenomena in such algorithms, like flipping a coin or choosing an identity from a fixed range according to a uniform distribution, whereas the nondeterminism in MDPs is used to specify unknown or underspecified behaviour, e.g., concurrency (interleaving) or the unknown malicious behavior of an attacker.

MDPs—also considered as turn-based $1\frac{1}{2}$-player stochastic games—consist of decision epochs, states, actions, and transition probabilities. On entering a state, an action, α, say, is nondeterministically selected and the next state is determined randomly by a probability distribution that depends on α. Actions may incur a reward, interpreted as gain, or dually, as cost. Schedulers or strategies prescribe which actions to choose in a state. One of the simplest schedulers, the so-called memoryless ones, base their decision solely on the current state and not on the further history. A plethora of results for MDPs are known that mainly focus on finding an optimal scheduler for a certain objective, see e.g. [8]. For, e.g., reachability objectives—find a scheduler, possibly the simplest one, that maximises the probability to reach a set of states— memoryless schedulers suffice and can be determined in polynomial time. For step-bounded reachability objectives, finite memory schedulers are sufficient. These schedulers perform the selection process on the basis of a finite piece of information, typically encoded as a finite-state automaton that runs in parallel to the MDP at hand.

This paper considers turn-based $1\frac{1}{2}$-player stochastically timed games, also known as *continuous-time* Markov decision processes (CTMDPs) [8]. They behave as MDPs but in addition their timing behaviour is random. The probability to stay at most t time units in a state is determined by a negative exponential distribution of which the rate depends on α. A reward is obtained which is linearly dependent on the time t spent in state s, as well as on a factor $\varrho(s, \alpha)$, the state- and action-dependent reward rate. In contrast to MDPs, CTMDPs have received far less attention; a reason for this might be the increased complexity when moving to continuous time. This paper studies reachability objectives for CTMDPs, in particular time-bounded reachability—what is the optimal policy to reach a set of states within a certain deadline—reward-bounded reachability, and their combination. We survey the results in this field, and show that reward-bounded and time-bounded reachability are interchangeable.

The presented reachability objectives are for instance relevant for job-shop scheduling problems where individual jobs have a random exponential duration, see e.g., [5]. The problem of finding a schedule for a fixed number of such (preemptable) jobs on a given set of identical machines such that the probability to meet a given deadline is maximised, is, in fact, an instance of timed reachability on CTMDPs. Optimal memoryless strategies exist for minimising the sum of the job completion times, but, as is shown, this is not the case for maximising the probability to reach the deadline. The same applies for maximising the probability to complete all jobs within a fixed cost.

This paper is further structured as follows. Section 2 rehearses the necessary background in the area of Markov decision processes, schedulers, stochastic processes, and reachability objectives. Section 3 then recalls the logic CSRL and discusses its semantics for continuous-time Markov reward decision processes. Section 4 then discusses a number of new results on the duality of the roles of time and reward in such processes. Section 5 concludes the paper.

2 Preliminaries

2.1 Markov decision processes

Let AP be a fixed set of atomic propositions.

Definition 2.1 (CTMDP)**.** A *continuous-time Markov decision process* (CTMDP) $\mathcal{M}$ is a tuple $(S, \mathrm{Act}, \mathbf{R}, L)$ with S, a countable set of *states*, Act, a set of *actions*, $\mathbf{R} : S \times \mathrm{Act} \times S \to \mathbb{R}_{\geqslant 0}$, the rate function such that for each $s \in S$ there exists a pair $(\alpha, s') \in \mathrm{Act} \times S$ with $\mathbf{R}(s, \alpha, s') > 0$, and labeling function $L : S \to 2^{\mathrm{AP}}$.

The set of actions that are enabled in state s is denoted $\mathrm{Act}(s) = \{\, \alpha \in \mathrm{Act} \mid \exists s'.\, \mathbf{R}(s, \alpha, s') > 0 \,\}$. The above condition thus requires each state to have at least one outgoing transition. Note that this condition can easily be fulfilled by adding self-loops.

The operational behavior of a CTMDP is as follows. On entering state s, an action α, say, in $\mathrm{Act}(s)$ is nondeterministically selected. Let $\mathbf{R}(s, \alpha, B)$ denote the total rate from state s to some state in B, i.e.,

$$\mathbf{R}(s, \alpha, B) \;=\; \sum_{s' \in B} \mathbf{R}(s, \alpha, s').$$

Given that action α has been chosen, the probability that the transition $s \xrightarrow{\alpha} s'$ can be triggered within the next t time units is $1 - e^{-\mathbf{R}(s,\alpha,s')\cdot t}$. The delay of transition $s \xrightarrow{\alpha} s'$ is thus governed by a negative exponential distribution with rate $\mathbf{R}(s, \alpha, s')$. If multiple outgoing transitions exist for the chosen action, they compete according to their exponentially distributed delays. For $B \subseteq S$, let $\underline{E}(s, \alpha) = \mathbf{R}(s, \alpha, S)$ denote the exit rate of state s under action α. If $\underline{E}(s, \alpha) > 0$, the probability to move from s to s' via action α within t time units, i.e., the probability that $s \xrightarrow{\alpha} s'$ wins the competition among all outgoing α-transitions of s is:

$$\frac{\mathbf{R}(s,\alpha,s')}{\underline{E}(s,\alpha)} \cdot \left(1 - e^{-\underline{E}(s,\alpha)\cdot t}\right),$$

where the first factor describes the discrete probability to take transition $s \xrightarrow{\alpha} s'$ and the second factor reflects the sojourn time in state s given that s is left via action α. Note that the sojourn time is distributed negative exponentially with rate equal to the sum of the rates of the outgoing α-transitions of state s. This is conform the minimum property of exponential distributions.

A CTMC (a *continuous-time Markov chain*) is a CTMDP in which for each state s, $\mathrm{Act}(s)$ is a singleton. In this case, the selection of actions is purely deterministic, and $\mathbf{R}$ can be projected on an $(S \times S)$ matrix, known as the transition rate matrix.

Definition 2.2 (MDP)**.** A (discrete-time) *Markov decision process* (MDP) $\mathcal{M}$ is a tuple $(S, \mathrm{Act}, \mathbf{P}, L)$ with S, Act, and L as before and $\mathbf{P} : S \times \mathrm{Act} \times S \to [0,1]$, a probability function such that for each pair (s,α):

$$\sum_{s' \in S} \mathbf{P}(s,\alpha,s') \in \{\,0,1\,\}.$$

A DTMC (a *discrete-time Markov chain*) is an MDP in which for each state s, $\mathrm{Act}(s)$ is a singleton. In this case, $\mathbf{P}$ can be projected on an $(S \times S)$ matrix, known as the transition probability matrix of a DTMC.

Definition 2.3 (Embedded MDP of a CTMDP)**.** For CTMDP $\mathcal{M} = (S, \mathrm{Act}, \mathbf{R}, L)$, the discrete probability of selecting transition $s \xrightarrow{\alpha} s'$ is determined by the *embedded MDP*, denoted $\mathrm{emb}(\mathcal{M}) = (S, \mathrm{Act}, \mathbf{P}, L)$, with:

$$\mathbf{P}(s,\alpha,s') \;=\; \begin{cases} \dfrac{\mathbf{R}(s,\alpha,s')}{\underline{E}(s,\alpha)}, & \text{if } \underline{E}(s,\alpha) > 0, \\ 0, & \text{otherwise.} \end{cases}$$

$\mathbf{P}(s,\alpha,s')$ is the time-abstract probability for the α-transition from s to s' on selecting action α. For $B \subseteq S$ let $\mathbf{P}(s,\alpha,B) = \sum_{s' \in B} \mathbf{P}(s,\alpha,s')$.

Definition 2.4 (Path in a CTMDP)**.** An infinite path in a CTMDP $\mathcal{M} = (S, \mathrm{Act}, \mathbf{R}, L)$ is a sequence $s_0, \alpha_0, t_0, s_1, \alpha_1, t_1, s_2, \alpha_2, t_2, \ldots$ in $(S \times \mathrm{Act} \times \mathbb{R}_{>0})^{\omega}$, written as:

$$s_0 \xrightarrow{\alpha_0, t_0} s_1 \xrightarrow{\alpha_1, t_1} s_2 \xrightarrow{\alpha_2, t_2} \cdots .$$

Any finite prefix of σ that ends in a state is a finite path in $\mathcal{M}$. Let $\mathrm{Paths}(\mathcal{M})$ denote the set of infinite paths in $\mathcal{M}$.

Let $\sigma = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} s_2 \xrightarrow{\alpha_2,t_2} \cdots \in \mathrm{Paths}(\mathcal{M})$. The time-abstract path of σ is $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots$, the corresponding action-abstract path is: $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \ldots$, and the time- and action-abstract path is the state sequence $s_0, s_1, s_2, \ldots$. Let first(σ) denote the first state of σ. For finite path σ, last(σ) denotes the last state of σ, and $\sigma \to s$ denotes the finite time- and action-abstract path σ followed by state s. For $i \in \mathbb{N}$, let $\sigma[i] = s_i$ denote the $(i{+}1)$-st state of σ. $\sigma@t$ denotes the state occupied at time instant $t \in \mathbb{R}_{\geqslant 0}$, i.e., $\sigma@t = \sigma[k]$ where k is the smallest index such that $\sum_{i=0}^{k} t_i > t$.

Definition 2.5 (CMRDP)**.** A *continuous-time Markov reward decision process* (CMRDP) is a pair $(\mathcal{M}, \varrho)$ with $\mathcal{M}$ a CTMDP with state space S and $\varrho : S \times \mathrm{Act} \to \mathbb{R}_{\geqslant 0}$ a reward function.

CMRDPs are often called CTMDPs in the literature [8]. The state reward function ϱ assigns to each state $s \in S$ and action $\alpha \in \mathrm{Act}$ a reward rate $\varrho(s, \alpha)$. Under the condition that action α has been selected in state s, a reward $\varrho(s, \alpha)\cdot t$ is acquired after residing t time units in state s. Recall that t is governed by an exponential distribution with rate $\underline{E}(s, \alpha)$, i.e., t randomly depends on action α. A path through a CMRDP is a path through its underlying CTMDP. For timed path $\sigma = s_0 \xrightarrow{\alpha_0,t_0} s_1 \xrightarrow{\alpha_1,t_1} s_2 \xrightarrow{\alpha_2,t_2} \cdots$ and $t = \sum_{i=0}^{k-1} t_i + t'$ with $t' \leqslant t_k$ let:

$$y(\sigma, t) = \sum_{i=0}^{k-1} t_i \cdot \varrho(s_i, \alpha_i) + t' \cdot \varrho(s_k, \alpha_k)$$

the accumulated reward along σ up to time t. An MRM (*Markov reward model*) is a CTMC equipped with a reward function. As an MRM is action-deterministic, ϱ may be viewed as a function of the type $S \to \mathbb{R}_{\geqslant 0}$.

2.2 Schedulers

CMRDPs incorporate nondeterministic decisions, not present in CTMCs. Nondeterminism in a CTMDP is resolved by a *scheduler*. In the literature, schedulers are sometimes also referred to as adversaries, policies, or strategies. For deciding which of the next nondeterministic actions to take, a scheduler may "have access" to the current state only or to the path from the initial to the current state (either with or without timing information). Schedulers may select the next action either (i) *deterministically*, i.e., depending on the available information, the next action is chosen in a deterministic way, or (ii) in a *randomized* fashion, i.e., depending on the available information the next action is chosen probabilistically. Accordingly, the following classes of schedulers D are distinguished [8], where Distr(Act) denotes the collection of all probability distributions on Act:

- stationary Markovian deterministic (SMD), $D : S \to \mathrm{Act}$ such that $D(s) \in \mathrm{Act}(s)$
- stationary Markovian randomized (SMR), $D : S \to \mathrm{Distr}(\mathrm{Act})$ such that $D(s)(\alpha) > 0$ implies $\alpha \in \mathrm{Act}(s)$
- Markovian deterministic (MD, also called *step-dependent* schedulers), $D : S \times \mathbb{N} \to \mathrm{Act}$ such that $D(s, n) \in \mathrm{Act}(s)$
- Markovian randomized (MR), $D : S \times \mathbb{N} \to \mathrm{Distr}(\mathrm{Act})$ such that $D(s, n)(\alpha) > 0$ implies $\alpha \in \mathrm{Act}(s)$
- (time-abstract) history-dependent, deterministic (HD), $D\colon (S \times \mathrm{Act})^* \times S \to \mathrm{Act}$ such that

$$D(\underbrace{s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}}}_{\text{time-abstract history}}, s_n) \in \mathrm{Act}(s_n)$$

- (time-abstract) history-dependent, randomized (HR), $D : (S \times \mathrm{Act})^* \times S \to \mathrm{Distr}(\mathrm{Act})$ such that $D(s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}}, s_n)(\alpha) > 0$ implies $\alpha \in \mathrm{Act}(s_n)$.

All these schedulers are time-abstract and cannot base their decisions on the sojourn times. Timed (measurable) schedulers [9, 7] are not considered in this paper. Finally, let X denote the class of all X-schedulers over a fixed CTMDP $\mathcal{M}$.[1]

Note that for any HD-scheduler, the actions can be dropped from the history, i.e., HD-schedulers may be considered as functions $D : S^+ \to \mathrm{Act}$, as for any sequence $s_0, s_1, \ldots, s_n$ the relevant actions α_i are given by $\alpha_i = D(s_0, s_1, \ldots, s_i)$, and, hence, the scheduled action sequence can be constructed from prefixes of the path at hand. Any state-action sequence $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{n-1}} s_n$ where $\alpha_i \neq D(s_0, s_1, \ldots, s_i)$ for some i, does not describe a path fragment that can be obtained from D.

The scheduler-types form a hierarchy, e.g., any SMD-scheduler can be viewed as an MD-scheduler (by ignoring parameter n) which, in turn, can be viewed as an HD-scheduler (by ignoring everything from the history except its length). A similar hierarchy exists between SMR, MR, and HR schedulers. Moreover, deterministic schedulers can be regarded as trivial versions of their corresponding randomized counterparts that assign probability one to the actions selected.

[1] Strictly speaking, we should write $X(\mathcal{M})$ but $\mathcal{M}$ is omitted as it should be clear from the context.

2.3 Induced stochastic process

Given a scheduler D (of arbitrary type listed above) and a starting state, D induces a stochastic process on a CTMDP $\mathcal{M}$. For deterministic schedulers (HD, MD, and SMD), the induced process is a CTMC, referred to as $\mathcal{M}_D$ in the sequel. For MD- and HD-schedulers, though, the state space of $\mathcal{M}_D$ will in general be infinitely large (but countable).

Definition 2.6 (Induced CTMC of a CTMDP)**.** Let $\mathcal{M} = (S, \mathrm{Act}, \mathbf{R}, L)$ be a CTMDP and $D : S^+ \to \mathrm{Act}$ an HD-scheduler on $\mathcal{M}$. The CTMC $\mathcal{M}_D = (S^+, \mathbf{R}_D, L')$ with:

$$\mathbf{R}_D(\sigma, \sigma') \quad = \quad \begin{cases} \mathbf{R}(\mathrm{last}(\sigma), D(\sigma), s), & \text{if } \sigma' = \sigma \to s, \\ 0, & \text{otherwise,} \end{cases}$$

and $L'(\sigma) = L(\mathrm{last}(\sigma))$.

The embedded DTMC $\mathrm{emb}(\mathcal{M}_D)$ is a tuple $(S^+, \mathbf{P}_D, L)$ where:

$$\mathbf{P}_D(\sigma, \sigma') \quad = \quad \begin{cases} \dfrac{\mathbf{R}_D(\sigma, \sigma')}{E_D(\sigma)}, & \text{if } E_D(\sigma) > 0, \\ 0, & \text{otherwise.} \end{cases}$$

Here, $E_D(\sigma) = \mathbf{R}_D(\sigma, S^+)$, i.e., the exit rate of σ in $\mathcal{M}_D$. States in CTMC $\mathcal{M}_D$ can be seen as state sequences $s_0 \to s_1 \to \ldots \to s_{n-1} \to s_n$ corresponding to time- and action-abstract path fragments in the CTMDP $\mathcal{M}$. State s_n stands for the current state in the CTMDP whereas states s_0 through s_{n-1} describe the history. Intuitively, the stochastic process induced by an HD-scheduler D on the CTMDP $\mathcal{M}$ results from unfolding $\mathcal{M}$ into an (infinite) tree while resolving the nondeterministic choices according to D. For SMD-schedulers, the induced CTMC is guaranteed to be finite. More precisely, for SMD-scheduler D, $\mathcal{M}_D$ can be viewed as a CTMC with the original state space S, as all sequences that end in s, say, are lumping equivalent [6].

In contrast to a CTMDP (or MDP), a CTMC (or DTMC) is a fully determined stochastic process. For a given initial state s_0 in CTMC $\mathcal{M}$, a unique probability measure $\Pr_{s_0}$ on $\mathrm{Paths}(s_0)$ exists, where $\mathrm{Paths}(s_0)$ denotes the set of timed paths that start in s_0. Timed paths through a CTMC are defined as for CTMDPs, but by nature are action-abstract. The inductive construction of the probability measure below follows [2], the fact that we allow countable-state Markov chains does not alter the construction. Let $\mathbf{P}$ be the probability matrix of the embedded DTMC of $\mathcal{M}$ and let $\mathrm{Cyl}(s_0 \xrightarrow{I_0} \cdots \xrightarrow{I_{k-1}} s_k)$ denote the cylinder set consisting of all timed paths σ that start in state s_0 such that s_i $(i \leqslant k)$ is the $(i{+}1)$th state on σ and the time spent in s_i lies in the non-empty interval I_i $(i < k)$ in $\mathbb{R}_{\geqslant 0}$.

The cylinder sets induce the probability measure $\Pr_{s_0}$ on the timed paths through $\mathcal{M}$, defined by induction on k by $\Pr_{s_0}(\mathrm{Cyl}(s_0)) = 1$, and, for $k > 0$:

$$\Pr_{s_0}(\mathrm{Cyl}(s_0 \xrightarrow{I_0} \cdots \xrightarrow{I_{k-1}} s_k \xrightarrow{I'} s')) \;=\;$$
$$\Pr_{s_0}(\mathrm{Cyl}(s_0 \xrightarrow{I_0} \cdots \xrightarrow{I_{k-1}} s_k))\cdot\, \mathbf{P}(s_k, s') \cdot \left(e^{-E(s_k)\cdot a} - e^{-E(s_k)\cdot b}\right),$$

where $a = \inf I'$ and $b = \sup I'$.

2.4 Reachability objectives

For CTMDP $\mathcal{M}$ with state space S and $B \subseteq S$, we consider the maximum (or, dually, minimum) probability to reach B under a given class of schedulers. Let $\Diamond B$ denote the event to eventually reach some state in B, $\Diamond^{\leqslant t} B$ the same event with the extra condition that B is reached within t time units, and $\Diamond_{\leqslant r} B$ the event that B is eventually reached within accumulated reward r. The event $\Diamond^{\leqslant t}_{\leqslant r} B$ asserts that B is reached within t time units and accumulated reward at most r. Note that the accumulated reward gained depends on the sojourn times in states, hence the bounds t and r are not independent. It is not difficult to assess that these events are measurable for the time-abstract schedulers considered here. A detailed proof of the measurability of $\Diamond^{\leqslant t} B$ for measurable timed schedulers (a richer class of schedulers) can be found in [7]. The probability for such an event φ to hold in state s of $\mathcal{M}$ is denoted $\Pr(s \models \varphi)$, i.e.,

$$\Pr(s \models \varphi) \;=\; \Pr_s\{\, \sigma \in \mathrm{Paths}(\mathcal{M}) \mid \sigma \models \varphi \,\}.$$

The maximal probability to reach a state in B under a HR-scheduler is given by:

$$\Pr^{\max}_{\mathrm{HR}}(s \models \Diamond B) \;=\; \sup_{D \in \mathrm{HR}} \Pr(s \models \Diamond B).$$

In a similar way, $\Pr^{\min}_{\mathrm{HR}}(s \models \Diamond B) = \inf_{D \in \mathrm{HR}} \Pr(s \models \Diamond B)$.

The following result follows immediately from the fact that for event $\Diamond B$ it suffices to consider the embedded MDP of a given CTMDP, and the fact that memoryless schedulers for finite MDPs exist that maximize the reachability probability for B. Such memoryless schedulers are obtained in polynomial time by solving a linear optimization problem. A similar result holds for minimal probabilities and for events of the form $\Diamond^{\leqslant n} B$, i.e., the event that B is reached within $n \in \mathbb{N}$ steps (i.e., transitions). Note that the event $\Diamond^{\leqslant t} B$ requires a state in B to be reached within t time units (using an arbitrary number of transitions), while $\Diamond^{\leqslant n} B$ requires B to be reached in n discrete steps, regardless of the time spent to reach B.

Lemma 2.7 (Optimal SMD schedulers for reachability)**.** Let $\mathcal{M}$ be a finite CTMDP with state space S and $B \subseteq S$. There exists an SMD scheduler D

such that for any $s \in S$:

$$\Pr^D(s \models \Diamond B) \;=\; \Pr^{\max}_{\mathrm{HR}}(s \models \Diamond B).$$

2.5 Time- and cost-bounded reachability

Consider the following class of CTMDPs:

Definition 2.8 (Uniform CTMDP)**.** A CTMDP $(S, \mathrm{Act}, \mathbf{R}, L)$ is *uniform* if for some $E > 0$ it holds $E(s, \alpha) = E$ for any state $s \in S$ and $\alpha \in \mathrm{Act}(s)$.

Stated in words, in a uniform CTMDP the exit rates for all states and all enabled actions are equal. It follows from [3]:

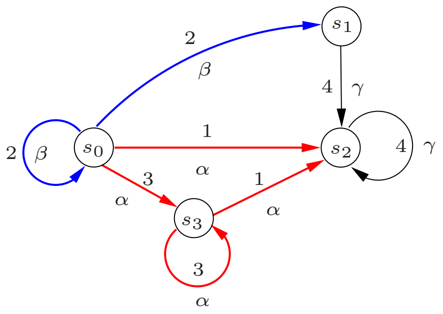
Theorem 2.9 (Optimal MD schedulers for timed reachability)**.** Let $\mathcal{M}$ be a finite *uniform* CTMDP with state space S, $t \in \mathbb{R}_{\geqslant 0}$ and $B \subseteq S$. There exists an MD scheduler D such that for any $s \in S$:

$$\Pr^D(s \models \Diamond^{\leqslant t} B) \;=\; \Pr^{\max}_{\mathrm{HR}}(s \models \Diamond^{\leqslant t} B).$$

An ε-approximation of such scheduler, i.e., a scheduler that obtains $\Pr^D(s \models \Diamond^{\leqslant t} B)$ up to an accuracy of ε, can be obtained in polynomial time by a greedy backward reachability algorithm as presented in [3]. A similar result can be obtained for minimal time-bounded reachability probabilities by selecting a transition with smallest, rather than largest, probability in the greedy algorithm.

The following example shows that memoryless schedulers for maximal time-bounded reachability probabilities may not exist.

Example 2.10 (Optimal SMD schedulers may not exist)**.** Consider the following uniform CTMDP:



Action labels and rates are indicated at each edge. Let $B = \{ s_2 \}$, and consider the SMD-schedulers, D_α, selecting action α in state s_0, and D_β, selecting action β. Comparing them with $D_{\beta\alpha}$, i.e., the scheduler that after selecting β once switches to selecting α in state s_0, we find that for a certain range of time bounds t, $D_{\beta\alpha}$ outperforms both D_β and D_α. Intuitively, the probability of stuttering in state s_0 (by choosing β initially) may influence the remaining time to reach B to an extent that it becomes profitable to continue choosing α. For $t = 0.5$, for instance, $\Pr_{D_{\beta\alpha}}(s_0, \Diamond^{\leqslant 0.5} B) = 0.4152$, whereas for D_α and D_β these probabilities are 0.3935 and 0.3996, respectively.

The following result is of importance later and is based on a result in [3]. Informally, it states that maximal (and minimal) probabilities for timed reachabilities in CTMDPs under deterministic and randomised HD schedulers coincide. As this result holds for arbitrary CTMDPs, there is no need to restrict to uniform ones here.

Theorem 2.11 (Maximal probabilities are invariant under randomization)**.** For CMRDP $\mathcal{M}$ with state space S, $s \in S$ and $B \subseteq S$, it holds for any $r, t \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$:

$$\begin{array}{rcl}
\sup_{D\in\mathrm{HD}} \Pr^D(s \models \Diamond^{\leqslant t} B) & = & \sup_{D\in\mathrm{HR}} \Pr^D(s \models \Diamond^{\leqslant t} B) \\
\sup_{D\in\mathrm{HD}} \Pr^D(s \models \Diamond_{\leqslant r} B) & = & \sup_{D\in\mathrm{HR}} \Pr^D(s \models \Diamond_{\leqslant r} B) \\
\sup_{D\in\mathrm{HD}} \Pr^D(s \models \Diamond^{\leqslant t}_{\leqslant r} B) & = & \sup_{D\in\mathrm{HR}} \Pr^D(s \models \Diamond^{\leqslant t}_{\leqslant r} B).
\end{array}$$

Analogous results hold for minimal probabilities for the events $\Diamond^{\leqslant t} B$, $\Diamond_{\leqslant r} B$, and $\Diamond^{\leqslant t}_{\leqslant r} B$.

Proof. For any HD-scheduler D for the CTMDP $\mathcal{M}$ it holds:

$$\Pr^D(s \models \Diamond^{\leqslant t} B) = \lim_{n\to\infty} \Pr^D(s \models \Diamond^{\leqslant t, \leqslant n} B)$$

where the superscript $\leqslant n$ denotes that B has to be reached within at most n transitions. Similarly, we have:

$$\Pr^D(s \models \Diamond_{\leqslant r} B) = \lim_{n\to\infty} \Pr^D(s \models \Diamond^{\leqslant n}_{\leqslant r} B).$$

By induction on n, it can be shown (cf. [3, Theorem 7]) that there is a finite family $(D_i)_{i\in J_n}$ (with J_n an index set) of HD-schedulers such that the measure $\Pr_{D'}$ induced by an HR-scheduler D' for the cylinder sets induced by path fragments consisting of n transitions is a convex combination of the measures $\Pr_{D_i}$, $i \in J_n$. Q.E.D.

The results for the events $\Diamond B$ and $\Diamond^{\leqslant t} B$ in finite CTMDP $\mathcal{M}$ can be generalized towards constrained reachability properties $C \,\mathsf{U}\, B$ and $C \,\mathsf{U}^{\leqslant t} B$, respectively, where $C \subseteq S$. This works as follows. First, all states in $S \setminus (C \cup B)$ and in B are made absorbing, i.e., their enabled actions are replaced by a single action, α_s, say, with $\mathbf{R}(s, \alpha_s, s) > 0$. The remaining states are unaffected. Paths that visit some state in $S \setminus (C \cup B)$ contribute probability zero to the event $C \,\mathsf{U}\, B$ while the continuation of paths that have reached B is of no importance to the probability of this event. For the resulting CTMDP $\mathcal{M}'$ it follows:

$$\begin{aligned}
\Pr^{\max}_{\mathcal{M},X}(s \models C \,\mathsf{U}^{\leqslant n}\, B) &= \Pr^{\max}_{\mathcal{M}',X}(s \models \Diamond^{\leqslant n}\, B),\\
\Pr^{\max}_{\mathcal{M},X}(s \models C \,\mathsf{U}\, B) &= \Pr^{\max}_{\mathcal{M}',X}(s \models \Diamond B),\\
\Pr^{\max}_{\mathcal{M},X}(s \models C \,\mathsf{U}^{\leqslant t}\, B) &= \Pr^{\max}_{\mathcal{M}',X}(s \models \Diamond^{\leqslant t}\, B),
\end{aligned}$$

where the subscript of Pr indicates the CTMDP of interest. Similar results hold for $\Pr^{\min}$.

For the event $C \,\mathsf{U}_{\leqslant r}\, B$ in CMRDP $\mathcal{M}$, the states in $S \setminus C \cup B$ are made absorbing (as before) and the reward of states in B is set to zero. The latter ensures that the accumulation of reward halts as soon as B is reached. Then it follows:

$$\Pr^{\max}_{\mathcal{M},X}(s \models C \,\mathsf{U}_{\leqslant r}\, B) = \Pr^{\max}_{\mathcal{M}^*,X}(s \models \Diamond_{\leqslant r}\, B)$$

and similar for $\Pr^{\min}$, where $\mathcal{M}^*$ is the resulting CMRDP after the transformations indicated above.

3 Continuous stochastic reward logic

CSRL is a branching-time temporal logic, based on the Computation Tree Logic (CTL). A CSRL formula asserts conditions on a state of a CMRDP. Besides the standard propositional logic operators, CSRL incorporates the probabilistic operator $\mathbb{P}_J(\varphi)$ where φ is a path-formula and J is an interval of $[0, 1]$. The path-formula φ imposes a condition on the set of paths, whereas J indicates a lower bound and/or upper bound on the probability. The intuitive meaning of the formula $\mathbb{P}_J(\varphi)$ in state s is: the probability for the set of paths satisfying φ and starting in s meets the bounds given by J. The probabilistic operator can be considered as the quantitative counterpart to the CTL-path quantifiers $\exists$ and $\forall$.

The path formulae φ are defined as for CTL, except that a bounded until operator is additionally incorporated. The intuitive meaning of the path formula $\Phi \,\mathsf{U}^I_K\, \Psi$ for intervals $I, K \subseteq \mathbb{R}_{\geqslant 0}$ is that a Ψ-state should be reached within $t \in I$ time units via a Φ-path with total cost $r \in K$.

Definition 3.1 (Syntax of CSRL)**.** CSRL *state-formulae* over the set AP of atomic propositions are formed according to the following grammar:

$$\Phi ::= \mathrm{true} \;\Big|\; a \;\Big|\; \Phi_1 \wedge \Phi_2 \;\Big|\; \neg\Phi \;\Big|\; \mathbb{P}_J(\varphi),$$

where $a \in \mathrm{AP}$, φ is a path-formula and $J \subseteq [0, 1]$ is an interval with rational bounds. CSRL *path-formulae* are formed according to:

$$\varphi ::= \bigcirc^I_K\, \Phi \;\Big|\; \Phi_1 \,\mathsf{U}^I_K\, \Phi_2,$$

where Φ, Φ_1 and Φ_2 are state-formulae, and $I, K \subseteq \mathbb{R}_{\geqslant 0} \cup \{ \infty \}$.

Other boolean connectives such as $\vee$ and $\rightarrow$ are derived in the obvious way. The reachability event considered before is obtained as $\Diamond^I_K \Phi = \text{true} \, \mathsf{U}^I_K \Phi$. The always-operator $\Box$ can be obtained by the duality of always/eventually and lower/upper probability bounds, e.g.,

$$\mathbb{P}_{\geqslant p}(\Box^I_K \Phi) = \mathbb{P}_{\leqslant 1-p}(\Diamond^I_K \neg\Phi) \text{ and } \mathbb{P}_{[p,q]}(\Box^I_K \Phi) = \mathbb{P}_{[1-q,1-p]}(\Diamond^I_K \neg\Phi).$$

Special cases occur for the trivial time-bound $I = [0, \infty)$ and the trivial reward-bound $K = [0, \infty)$:

$$\bigcirc \Phi = \bigcirc^{[0,\infty)}_{[0,\infty)} \Phi \text{ and } \Phi \, \mathsf{U} \, \Psi = \Phi \, \mathsf{U}^{[0,\infty)}_{[0,\infty)} \Psi.$$

The semantics of CSRL is defined over the class of HR-schedulers.

Definition 3.2 (Semantics of CSRL)**.** Let $a \in \text{AP}$, $\mathcal{M} = (S, \text{Act}, \mathbf{R}, L)$ a CMRDP, $s \in S$, Φ, Ψ CSRL state-formulae, and φ a CSRL path-formula. The satisfaction relation $\models$ for state-formulae is defined by:

$$\begin{array}{lll}
s \models a & \text{iff} & a \in L(s) \\
s \models \neg\Phi & \text{iff} & s \not\models \Phi \\
s \models \Phi \wedge \Psi & \text{iff} & s \models \Phi \text{ and } s \models \Psi \\
s \models \mathbb{P}_J(\varphi) & \text{iff} & \text{for any scheduler } D \in \text{HR} : \text{Pr}^D(s \models \varphi) \in J.
\end{array}$$

For path $\sigma = s_0 \xrightarrow{\alpha_0, t_0} s_1 \xrightarrow{\alpha_1, t_1} s_2 \xrightarrow{\alpha_2, t_2} \cdots$ in $\mathcal{M}$:

$$\begin{array}{lll}
\sigma \models \bigcirc^I_K \Phi & \text{iff} & \sigma[1] \models \Phi, t_0 \in I \text{ and } y(\sigma, t_0) \in K \\
\sigma \models \Phi \, \mathsf{U}^I_K \Psi & \text{iff} & \exists t \in I \, (\sigma@t \models \Psi \wedge (\forall \, t' < t. \, \sigma@t' \models \Phi) \wedge y(\sigma, t) \in K) .
\end{array}$$

The semantics for the propositional fragment of CSRL is standard. The probability operator $\mathbb{P}_J(\cdot)$ imposes probability bounds for all (time-abstract) schedulers. Accordingly, $s \models \mathbb{P}_{\leqslant p}(\varphi)$ if and only if $\text{Pr}^{\max}_{\text{HR}}(s \models \varphi) \leqslant p$, and similarly, $s \models \mathbb{P}_{\geqslant p}(\varphi)$ if and only if $\text{Pr}^{\min}_{\text{HR}}(s \models \varphi) \geqslant p$. The well-definedness of the semantics of $\mathbb{P}_J(\varphi)$ follows from the fact that for any CSRL path-formula φ, the set $\{ \sigma \in \text{Paths}(s) \mid \sigma \models \varphi \}$ is measurable. This follows from a standard measure space construction over the infinite paths in the stochastic process induced by an HD-scheduler over the CMRDP $\mathcal{M}$. In fact, the measurability of these events can also be guaranteed for measurable timed schedulers, cf. [7].

Recall that $\sigma@t$ denotes the current state along σ at time instant t, and $y(\sigma, t)$ denotes the accumulated reward along the prefix of σ up to time t. The intuition behind $y(\sigma, t)$ depends on the formula under consideration and the interpretation of the rewards in the CMRDP $\mathcal{M}$ at hand. For instance, for $\varphi = \Diamond \mathsf{good}$ and path σ that satisfies φ, the accumulated reward $y(\sigma, t)$ can be interpreted as the cost to reach a *good* state within t time units. For $\varphi = \Diamond \mathsf{bad}$, it may, e.g., be interpreted as the energy used before reaching a *bad* state within t time units.

4 Duality of time and reward

The main aim of this section is to show the duality of rewards and the elapse of time in a CMRDP. The proof strategy is as follows. We first consider the action-deterministic case, i.e., MRMs, and show that—in spirit of the observations in the late 1970s by Beaudry [4]— the progress of time can be regarded as the earning of reward and vice versa in the case of non-zero rewards. The key to the proof of this result is a least fixed-point characterization of $\Pr(C \,\mathsf{U}_K^I B)$ in MRMs. This result is then lifted to CMRDPs under HD-schedulers. By Theorem 2.11, the duality result also applies to HR-schedulers.

Consider first CMRDPs for which $\mathrm{Act}(s)$ is a singleton for each state s, i.e., MRMs. For time-bounded until-formula φ and MRM $\mathcal{M}$, $\Pr^{\mathcal{M}}(s \models \varphi)$ is characterized by a fixed-point equation. This is similar to CTL where appropriate fixed-point characterizations constitute the key towards model checking until-formulas. It suffices to consider time bounds specified by closed intervals since:

$$\Pr(s, \Phi \,\mathsf{U}_K^I \Psi) = \Pr(s, \Phi \,\mathsf{U}_{\mathrm{cl}(K)}^{\mathrm{cl}(I)} \Psi),$$

where $\mathrm{cl}(I)$ denotes the closure of interval I. A similar result holds for the next-step operator. The result follows from the fact that the probability measure of a basic cylinder set does not change when some of the intervals are replaced by their closure. In the sequel, we assume that intervals I and K are compact.

In the sequel, let $I \ominus x$ denote $\{\, t - x \mid t \in I \wedge t \geqslant x \,\}$ and $\mathbf{T}(s, s', x)$ denotes the density of moving from state s to s' in x time units, i.e.,

$$\mathbf{T}(s, s', x) = \mathbf{P}(s, s') \cdot \underline{E}(s) \cdot e^{-\underline{E}(s) \cdot x} = \mathbf{R}(s, s') \cdot e^{-\underline{E}(s) \cdot x}.$$

Here, $\underline{E}(s) \cdot e^{-\underline{E}(s) \cdot x}$ is the probability density function of the residence time in state s at instant x. Let Int denote the set of all (nonempty) intervals in $\mathbb{R}_{\geqslant 0}$. Let $L = \{\, x \in I \mid \varrho(s) \cdot x \in K \,\}$ for closed intervals I and K. As we consider MRMs, note that ϱ can be viewed as function $S \to \mathbb{R}_{\geqslant 0}$. (Strictly speaking, L is a function depending on s. As s is clear from the context, we omit it and write L instead of $L(s)$.) Stated in words, L is the subset of I such that the accumulated reward (in s) lies in K.

Theorem 4.1. Let $s \in S$, interval $I, K \subseteq \mathbb{R}_{\geqslant 0}$ and Φ, Ψ be CSRL state-formulas. The function $(s, I, K) \mapsto \Pr(s, \Phi \, \mathsf{U}_K^I \Psi)$ is the least fixed point of the (monotonic) higher-order operator

$$\Omega : (S \times \mathrm{Int}^2 \to [0,1]) \to (S \times \mathrm{Int}^2 \to [0,1]),$$

where

$$\Omega(F)(s,I,K) := \begin{cases} 1, & \text{if } s \models \neg\Phi \wedge \Psi \text{ and} \\ & \inf I = \inf K = 0, \\ S_{\Phi\wedge\neg\Psi}(F,s,I,K) & \text{if } s \models \Phi \wedge \neg\Psi, \\ S_{\Phi\wedge\Psi}(F,s,I,K) & \text{if } s \models \Phi \wedge \Psi, \\ 0, & \text{otherwise,} \end{cases}$$

with

$$S_{\Phi\wedge\neg\Psi}(F,s,I,K) := \int_0^{\sup L} \sum_{s' \in S} \mathbf{T}(s,s',x) \cdot F(s', I \ominus x, K \ominus \varrho(s) \cdot x) \; dx$$

and

$$S_{\Phi\wedge\Psi}(F,s,I,K) := e^{-\underline{E}(s)\cdot \inf L} + \int_0^{\inf L} \sum_{s' \in S} \mathbf{T}(s,s',x) \cdot F(s', I \ominus x, K \ominus \varrho(s) \cdot x) \; dx.$$

Proof. Along the same lines as the proof of [2, Theorem 1]. Q.E.D.

The above characterisation is justified as follows. If s satisfies Φ and $\neg\Psi$ (second case), the probability of reaching a Ψ-state from s at time $t \in I$ by earning a reward $r \in K$ equals the probability of reaching some direct successor s' of s within x time units ($x \leqslant \sup I$ and $\varrho(s) \cdot x \leqslant \sup K$, that is, $x \leqslant \sup L$), multiplied by the probability of reaching a Ψ-state from s' in the remaining time $t - x$ while earning a reward of at most $r - \varrho(s) \cdot x$. If s satisfies $\Phi \wedge \Psi$ (third case), the path-formula φ is satisfied if no outgoing transition of s is taken for at least $\inf L$ time units[2] (first summand).

Alternatively, state s should be left before $\inf L$ in which case the probability is defined in a similar way as for the case $s \models \Phi \wedge \neg\Psi$ (second summand). Note that $\inf L = 0$ is possible (if e.g., $\inf K = \inf I = 0$). In this case, $s \models \Phi \wedge \Psi$ yields that any path starting in s satisfies $\varphi = \Phi \, \mathsf{U}_K^I \, \Psi$ and $\Pr(s, \varphi) = 1$.

[2] By convention, $\inf \varnothing = \infty$.

Definition 4.2 (Dual CMRDP)**.** The *dual* of a CMRDP $\mathcal{M} = (S, \mathrm{Act}, \mathbf{R}, L, \varrho)$ with $\varrho(s,\alpha) > 0$ for all $s \in S$ and $\alpha \in \mathrm{Act}$ is the CMRDP $\mathcal{M}^* = (S, \mathrm{Act}, \mathbf{R}^*, L, \varrho^*)$ where for $s, s' \in S$ and $\alpha \in \mathit{Act}$:

$$\mathbf{R}^*(s,\alpha,s') \;=\; \frac{\mathbf{R}(s,\alpha,s')}{\varrho(s,\alpha)} \quad \text{and} \quad \varrho^*(s,\alpha) \;=\; \frac{1}{\varrho(s,\alpha)}.$$

Intuitively, the transformation of $\mathcal{M}$ into $\mathcal{M}^*$ stretches the residence time in state s under action α with a factor that is proportional to the reciprocal of reward $\varrho(s,\alpha)$ if $0 < \varrho(s,\alpha) < 1$. The reward function is changed similarly. Thus, for pairs (s,α) with $\varrho(s,\alpha) < 1$ the sojourn time in s is extended, whereas if $\varrho(s,\alpha) > 1$ they are accelerated. For fixed action α, the residence of t time units in state s in $\mathcal{M}^*$ may be interpreted as the earning of t reward in s in $\mathcal{M}$, or reversely, earning a reward r in state s in $\mathcal{M}$ corresponds to a residence of r time units in s in $\mathcal{M}^*$.

The exit rates in $\mathcal{M}^*$ are given by $\underline{E}^*(s,\alpha) = \underline{E}(s,\alpha)/\varrho(s,\alpha)$. It follows that $(\mathcal{M}^*)^* = \mathcal{M}$ and that $\mathcal{M}$ and $\mathcal{M}^*$ have the same time-abstract transition probabilities as $\underline{E}^*(s,\alpha) = 0$ iff $\underline{E}(s,\alpha) = 0$ and for $\underline{E}^*(s,\alpha) > 0$:

$$\mathbf{P}^*(s,\alpha,s') = \frac{\mathbf{R}^*(s,\alpha,s')}{\underline{E}^*(s,\alpha)} = \frac{\mathbf{R}(s,\alpha,s')/\varrho(s,\alpha)}{\underline{E}(s,\alpha)/\varrho(s,\alpha)} = \frac{\mathbf{R}(s,\alpha,s')}{\underline{E}(s,\alpha)} = \mathbf{P}(s,\alpha,s').$$

Note that a time-abstract scheduler on CMRDP $\mathcal{M}$ is also a time-abstract scheduler on $\mathcal{M}^*$ and vice versa, as such schedulers can only base their decisions on time-abstract histories, and the set of time-abstract histories for $\mathcal{M}$ and $\mathcal{M}^*$ coincide. Finally, observe that uniformity is *not* maintained by *: $\mathcal{M}^*$ is in general not uniform for uniform $\mathcal{M}$.

Definition 4.3 (Dual formula)**.** For state formula Φ, Φ^* is the state formula obtained from Φ by swapping the time- and reward-bound in each subformula of the form $\bigcirc^I_K$ or U^I_K.

For state-formula Φ, let $\mathit{Sat}(\Phi) = \{\, s \in S \mid s \models \Phi \,\}$.

Theorem 4.4 (Duality for MRMs)**.** For MRM $\mathcal{M} = (S, \mathbf{R}, L, \varrho)$ with $\varrho(s) > 0$ for all $s \in S$ and CSRL state-formula Φ:

$$\mathit{Sat}^{\mathcal{M}}(\Phi) \;=\; \mathit{Sat}^{\mathcal{M}^*}(\Phi^*).$$

Proof. By induction on the structure of Φ. Let MRM $\mathcal{M} = (S, \mathbf{R}, L, \varrho)$ with $\varrho(s) > 0$ for all $s \in S$. We show that for each $s \in S$ and sets of states $B, C \subseteq S$:

$$\Pr{}^{\mathcal{M}}(s \models C \,\mathsf{U}^I_K\, B) \;=\; \Pr{}^{\mathcal{M}^*}(s \models C \,\mathsf{U}^K_I\, B).$$

The proof for a similar result for the next-step operator is obtained in an analogous, though simpler way. For the sake of simplicity, let $I = [0,t]$ and

$K = [0, r]$ with $r, t \in \mathbb{R}_{\geqslant 0}$. The general case can be obtained in a similar way. Let $s \in C \setminus B$. From Theorem 4.1 it follows:

$$\Pr^{\mathcal{M}^*}(s \models C \mathbin{\mathsf{U}}_I^K B) \;=\; \int_{L^*} \sum_{s' \in S} \mathbf{T}^*(s, s', x) \cdot \Pr^{\mathcal{M}^*}(s', C \mathbin{\mathsf{U}}_{I \ominus \varrho^*(s) \cdot x}^{K \ominus x} B)\; dx$$

for $L^* = \{\, x \in [0, t] | \varrho^*(s) \cdot x \in [0, r] \,\}$, i.e., $L^* = [0, \min(t, \frac{r}{\varrho^*(s)})]$. By the definition of $\mathcal{M}^*$ and $\mathbf{T}^*(s, s', x) = \mathbf{R}^*(s, s') \cdot e^{-\underline{E}^*(s) \cdot x}$, the right-hand side equals:

$$\int_{L^*} \sum_{s' \in S} \frac{\mathbf{R}(s, s')}{\varrho(s)} \cdot e^{-\frac{\underline{E}(s)}{\varrho(s)} \cdot x} \cdot \Pr^{\mathcal{M}^*}(s', C \mathbin{\mathsf{U}}_{I \ominus \frac{x}{\varrho(s)}}^{K \ominus x} B)\; dx.$$

By substitution $y = \frac{x}{\varrho(s)}$ this integral reduces to:

$$\int_{L} \sum_{s' \in S} \mathbf{R}(s, s') \cdot e^{-\underline{E}(s) \cdot y} \cdot \Pr^{\mathcal{M}^*}(s', C \mathbin{\mathsf{U}}_{I \ominus y}^{K \ominus \varrho(s) \cdot y} B)\; dy,$$

where $L = [0, \min(\frac{t}{\varrho(s)}, r)]$. Thus, the values $\Pr^{\mathcal{M}^*}(s, C \mathbin{\mathsf{U}}_I^K B)$ yield a solution to the equation system in Theorem 4.1 for $\Pr^{\mathcal{M}}(s, C \mathbin{\mathsf{U}}_K^I B)$. In fact, these values yield the *least* solution. The formal argument for this latter observation uses the fact that $\mathcal{M}$ and $\mathcal{M}^*$ have the same underlying digraph, and hence, $\Pr^{\mathcal{M}}(s, C \mathbin{\mathsf{U}}_K^I B) = 0$ iff $\Pr^{\mathcal{M}^*}(s, C \mathbin{\mathsf{U}}_I^K B) = 0$ iff there is no path starting in s where $C \mathbin{\mathsf{U}} B$ holds. In fact, the equation system restricted to $\{\, s \in S \mid \Pr^{\mathcal{M}}(s, C \mathbin{\mathsf{U}}_K^I B) > 0 \,\}$ has a unique solution. The values $\Pr^{\mathcal{M}^*}(s, C \mathbin{\mathsf{U}}_I^K B)$ and $\Pr^{\mathcal{M}}(s, C \mathbin{\mathsf{U}}_K^I B)$ are least solutions of the same equation system, and are thus equal. Hence, we obtain:

$$\int_{L} \sum_{s' \in S} \mathbf{T}(s, s', y) \cdot \Pr^{\mathcal{M}}(s', C \mathbin{\mathsf{U}}_{K \ominus \varrho(s) \cdot y}^{I \ominus y} B)\; dy$$

which equals $\Pr^{\mathcal{M}}(s \models C \mathbin{\mathsf{U}}_K^I B)$ for $s \in C \setminus B$. Q.E.D.

If $\mathcal{M}$ contains states equipped with a zero reward, the duality result does not hold, as the reverse of earning a zero reward in $\mathcal{M}$ when considering Φ should correspond to a residence of 0 time units in $\mathcal{M}^*$ for Φ^*, which—as the advance of time in a state cannot be halted— is in general not possible. However, the result of Theorem 4.4 applies to some restricted, though still practical, cases, viz. if (i) for each sub-formula of Φ of the form $\bigcirc_K^I \Phi'$ we have $K = [0, \infty)$, and (ii) for each sub-formula of the form $\Phi \mathbin{\mathsf{U}}_K^I \Psi$ either $K = [0, \infty)$ or $\mathit{Sat}^{\mathcal{M}}(\Phi) \subseteq \{\, s \in S \mid \varrho(s) > 0 \,\}$. The intuition is that either the reward constraint (i.e., time constraint) is trivial in Φ (in Φ^*), or that zero-rewarded states are not involved in checking the reward constraint. In such cases, let $\mathcal{M}^*$ be defined by $\mathbf{R}^*(s, s') = \mathbf{R}(s, s')$ and $\varrho^*(s) = 0$ in case $\varrho(s) = 0$ and defined as before otherwise.

Corollary 4.5 (Optimal MD schedulers for cost reachability)**.** Let $\mathcal{M}$ be a finite *uniform* CMRDP with state space S, $r \in \mathbb{R}_{\geqslant 0}$ and $B \subseteq S$. There exists an MD scheduler D such that for any $s \in S$:

$$\Pr^D(s \models \Diamond_{\leqslant r} B) \;=\; \Pr^{\max}_{\mathrm{HR}}(s \models \Diamond_{\leqslant r} B).$$

Proof. Let $\mathcal{M}$ be a uniform CMRDP. By Theorem 2.9 it follows:

$$\sup_{D \in \mathrm{HD}} \Pr^D_{\mathcal{M}}(s \models \Diamond^{\leqslant t} B) = \sup_{D \in \mathrm{MD}} \Pr^D_{\mathcal{M}}(s \models \Diamond^{\leqslant t} B).$$

Observe that there is a one-to-one relationship between schedulers of $\mathcal{M}$ and of its dual $\mathcal{M}^*$ as $\mathcal{M}$ and $\mathcal{M}^*$ have the same time-abstract scheduler for any class X as defined before. Moreover, for HD-scheduler D, the dual of MRM $\mathcal{M}_D$ is identical to the induced MRM of the dual of $\mathcal{M}$, i.e., $(\mathcal{M}_D)^* = \mathcal{M}^*_D$. Thus:

$$\sup_{D \in \mathrm{HD}} \Pr^D_{\mathcal{M}}(s \models \Diamond^{\leqslant t} B) = \sup_{D^* \in \mathrm{HD}} \Pr^{D^*}_{\mathcal{M}^*}(s \models \Diamond^{\leqslant t} B).$$

Applying Theorem 4.4 to $\mathcal{M}^*$ yields:

$$\sup_{D \in \mathrm{HD}} \Pr^D_{\mathcal{M}}(s \models \Diamond^{\leqslant t} B) = \sup_{D^* \in \mathrm{HD}} \Pr^{D^*}_{\mathcal{M}^*}(s \models \Diamond_{\leqslant r} B),$$

and by an analogous argument for MD-schedulers:

$$\sup_{D \in \mathrm{MD}} \Pr^D_{\mathcal{M}}(s \models \Diamond^{\leqslant t} B) = \sup_{D^* \in \mathrm{MD}} \Pr^{D^*}_{\mathcal{M}^*}(s \models \Diamond_{\leqslant r} B).$$

Thus:

$$\sup_{D \in \mathrm{HD}} \Pr^D_{\mathcal{M}^*}(s \models \Diamond_{\leqslant r} B) = \sup_{D \in \mathrm{MD}} \Pr^D_{\mathcal{M}^*}(s \models \Diamond_{\leqslant r} B).$$

In addition, Theorem 2.11 asserts:

$$\sup_{D \in \mathrm{HD}} \Pr^D_{\mathcal{M}}(s \models \Diamond_{\leqslant r} B) = \sup_{D \in \mathrm{HR}} \Pr^D_{\mathcal{M}}(s \models \Diamond_{\leqslant r} B)$$

and hence $\sup_{D^* \in \mathrm{MD}} \Pr^{D^*}_{\mathcal{M}^*}(s \models \Diamond_{\leqslant r} B)$ coincides with the suprema for the probability to reach B within reward bound r under all HD-, HR- and MD-schedulers. As MR-schedulers are between HR- and MD-schedulers, the stated result follows. Q.E.D.

Unfortunately, this result does not imply that the algorithm in [3] applied on $\mathcal{M}^*$ yields the optimal result for the event $\Diamond_{\leqslant r} B$, as $\mathcal{M}^*$ is not guaranteed to be uniform whereas the algorithm ensures optimality only for uniform CTMDPs.

We conclude this note by a duality result for arbitrary CMRDPs.

Corollary 4.6 (Duality for CMRDPs)**.** For a CMRDP $\mathcal{M} = (S, \mathrm{Act}, \mathbf{R}, L, \varrho)$ with $\varrho(s, \alpha) > 0$ for all $s \in S$ and $\alpha \in \mathrm{Act}$, and CSRL state-formula Φ:

$$Sat^{\mathcal{M}}(\Phi) \quad = \quad Sat^{\mathcal{M}^*}(\Phi^*).$$

Proof. By induction on the structure of Φ. Let CMRDP $\mathcal{M} = (S, \mathrm{Act}, \mathbf{R}, L, \varrho)$ with $\varrho(s, \alpha) > 0$ for all $s \in S$ and $\alpha \in \mathrm{Act}$. Consider $\Phi = \mathbb{P}_{\leqslant p}(C \, \mathsf{U}_K^I \, B)$. The proof for bounds of the form $\geqslant p$, and for the next-step operator are similar. From the semantics of CSRL it follows:

$$s \models_{\mathcal{M}} \mathbb{P}_{\leqslant p}(C \, \mathsf{U}_K^I \, B) \text{ iff } \sup_{D \in \mathrm{HR}} \mathrm{Pr}_{\mathcal{M}}^D(s \models C \, \mathsf{U}_K^I \, B) \leqslant p.$$

In a similar way as stated in the third item of Theorem 2.11 it follows:

$$\sup_{D \in \mathrm{HR}} \mathrm{Pr}_{\mathcal{M}}^D(s \models C \, \mathsf{U}_K^I \, B) = \sup_{D \in \mathrm{HD}} \mathrm{Pr}_{\mathcal{M}}^D(s \models C \, \mathsf{U}_K^I \, B).$$

$\mathcal{M}$ and $\mathcal{M}^*$ have the same time-abstract HD-schedulers and $(\mathcal{M}_D)^* = \mathcal{M}_D^*$. Theorem 4.4 yields:

$$\sup_{D \in \mathrm{HD}} \mathrm{Pr}_{\mathcal{M}}^D(s \models C \, \mathsf{U}_K^I \, B) = \sup_{D^* \in \mathrm{HD}} \mathrm{Pr}_{\mathcal{M}^*}^{D^*}(s \models C \, \mathsf{U}_I^K \, B).$$

As HD- and HR-schedulers are indistinguishable for events of the form $C \, \mathsf{U}_K^I \, B$ (the proof of this fact is analogous to that of Theorem 2.11), it follows:

$$\sup_{D^* \in \mathrm{HD}} \mathrm{Pr}_{\mathcal{M}^*}^{D^*}(s \models C \, \mathsf{U}_I^K \, B) = \sup_{D^* \in \mathrm{HR}} \mathrm{Pr}_{\mathcal{M}^*}^{D^*}(s \models C \, \mathsf{U}_I^K \, B).$$

Thus:

$$s \models_{\mathcal{M}} \mathbb{P}_{\leqslant p}(C \, \mathsf{U}_K^I \, B) \text{ iff } s \models_{\mathcal{M}^*} \mathbb{P}_{\leqslant p}(C \, \mathsf{U}_I^K \, B).$$

Q.E.D.

5 Epilogue

In this paper we have brought together results on the use of the logic CSRL and time and reward duality for MRMs [1], with recent results on reachability in CTMDPs [3]. This leads to a duality result for CMRDPs, as well as to the existence of optimal MD schedulers for cost reachability in uniform CMRDPs.

References

[1] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 780–792. Springer, 2000.

[2] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[3] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theor. Comput. Sci.*, 345(1):2–26, 2005.

[4] M. D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Trans. Computers*, 27(6):540–547, 1978.

[5] J. L. Bruno, P. J. Downey, and G. N. Frederickson. Sequencing tasks with exponential service times to minimize the expected flow time or makespan. *J. ACM*, 28(1):100–113, 1981.

[6] P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *J. Appl. Probab.*, 31(1):59–75, 1994.

[7] M. R. Neuhäußer and J.-P. Katoen. Bisimulation and logical preservation for continuous-time Markov decision processes. In L. Caires and V. T. Vasconcelos, editors, *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 412–427. Springer, 2007.

[8] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming.* Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. John Wiley & Sons Inc., New York, 1994. , A Wiley-Interscience Publication.

[9] N. Wolovick and S. Johr. A characterization of meaningful schedulers for continuous-time Markov decision processes. In E. Asarin and P. Bouyer, editors, *FORMATS*, volume 4202 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2006.

Logical theories and compatible operations

Achim Blumensath[1]
Thomas Colcombet[2]
Christof Löding[3]

[1] Fachbereich Mathematik
TU Darmstadt
Schloßgartenstraße 7
64289 Darmstadt, Germany
blumensath@mathematik.tu-darmstadt.de

[2] Laboratoire d'Informatique Algorithmique: Fondements et Applications
Université Paris Diderot, Paris 7
Case 7014
75205 Paris Cedex 13, France
thomas.colcombet@liafa.jussieu.fr

[3] Lehrstuhl Informatik 7
RWTH Aachen
Ahornstraße 55
52074 Aachen, Germany
loeding@cs.rwth-aachen.de

Abstract

We survey operations on (possibly infinite) relational structures that are *compatible* with logical theories in the sense that, if we apply the operation to given structures then we can compute the theory of the resulting structure from the theories of the arguments (the logics under consideration for the result and the arguments might differ).

Besides general compatibility results for these operations we also present several results on restricted classes of structures, and their use for obtaining classes of infinite structures with decidable theories.

1 Introduction

The aim of this article is to give a survey of operations that can be performed on relational structures while preserving decidability of theories. We mainly consider first-order logic (FO), monadic second-order logic (MSO), and guarded second-order logic (GSO, also called MS_2 by Courcelle). For example, we might be interested in an operation f that takes a single structure $\mathfrak{a}$ and produces a new structure $f(\mathfrak{a})$ such that the FO-theory of $f(\mathfrak{a})$ can be effectively computed from the MSO-theory of $\mathfrak{a}$ (we call such operations (MSO, FO)-*compatible*), i.e., for each FO-formula φ over $f(\mathfrak{a})$ we can construct an MSO-formula φ^* such that

$$f(\mathfrak{a}) \models \varphi \quad \text{iff} \quad \mathfrak{a} \models \varphi^*.$$

The main application of such operations is to transfer decidability results for logical theories. This technique can be applied for single structures, as well as uniformly over classes of structures. The first approach is often used for infinite structures, but it becomes trivial if the structure is finite since each finite structure has a decidable MSO-theory (even a decidable full second-order theory). The second approach is also useful for classes of finite structures as not every such class has a decidable theory.

In order to process structures by algorithmic means, a finite encoding of the structure is required. Such encodings are trivial when structures are finite (though one may be interested into finding compact presentations), but the choice of encoding becomes a real issue when dealing with infinite structures. The approach using operations compatible with logical theories is as follows. Starting from a (countable) set B of structures all of which have a decidable theory for a certain logic L, we can construct new structures with a decidable theory (possibly for a different logic L') by applying operations from a fixed (countable) set O of operations of the above form. This gives rise to the class C of all structures that can be obtained from the basic structures in B by application of the operations in O. Every element of C can be represented by a term over $O \cup B$. Evaluating an L'-formula over a structure in C then amounts to constructing and evaluating L-formulae over structures from B.

Given such a definition of a class of structures, an interesting problem is to understand what structures can be encoded in this way and to give alternative characterisations of them. Before we give examples of such classes, let us briefly summarise the main operations we are interested in.

Interpretations. An interpretation uses logical formulae with free variables to describe relations of a new structure inside a given one. Each formula with n free variables defines the relation of arity n that contains all tuples satisfying the formula. Usually, the free variables are first-order variables and the universe of the new structure is a subset of the universe of the original structure. Depending on the type of the formulae one speaks of FO- and MSO-interpretations, and it is not difficult to see that these types of interpretations preserve the respective logic. We shall frequently combine other operations with interpretations that perform some pre-processing and post-processing of structures.

Products. The simplest form is the *direct* or *Cartesian product* of two or more structures. A generalised version allows us to additionally define new relations on the product by evaluating formulae on the factors and relating the results on the different factors by another formula. Feferman and Vaught [32] proved that the first-order theory of such a product is determined by the first-order theories of its factors (see also [39] for an overview).

Sums. To transfer the results of Feferman and Vaught for products to monadic second-order logic, Shelah considered sums (or unions) of structures instead of products [45].

Iteration. The iteration of a structure consists of copies of the original structure that are arranged in a tree-like fashion. A theorem of Muchnik that has been proven in [49, 50] states that the MSO-theory of an iteration can be reduced to the MSO-theory of the original structure.

Incidence Structure. The universe of the incidence structure contains, in addition to the elements of the original structure, all tuples that appear in some relation. This construction can be used to reduce the GSO-theory of a structure to the MSO-theory of its incidence structure [33].

Power set. The power set of a structure consists of all of its subsets. The relations are transferred to the singleton sets and the signature additionally contains the subset relation. There is also a weak variant of the power-set operation that takes only the finite subsets of a structure. These constructions allow us to translate FO-formulae over the power-set structure to MSO-formulae over the original structure, and to weak MSO-formulae in case of finite sets [21].

Of course, these operations can also be combined to obtain more complex ones. For example, applying a product with a finite structure followed by an MSO-interpretation yields a *parameterless* MSO-*transduction* (see e.g., [24]). Or applying the power-set operation followed by an FO-interpretation gives an operation called a *set interpretation* (or finite set interpretation in the case of the weak power set) [21].

Besides the general results on the compatibility of these operations, we are interested in their behaviour on special classes of structures. In particular we consider the following families.

Tree-interpretable structures are structures that can be obtained by the application of an interpretation to a tree. Here, the interpretation can be chosen to be first-order, weak monadic-second order, or monadic second-order without affecting the definition (if the tree is changed accordingly). This class coincides with the class of structures of finite partition width [7]. The corresponding class of graphs consists of those with finite clique width [28]. Seese [43] conjectures that all structures with decidable MSO-theory are tree-interpretable.

Structures of finite tree width resemble trees. They can be characterised as the structures with a tree-interpretable incidence graph. A theorem of Seese [43] states that all structures with decidable GSO-theory are have finite tree width.

Uniformly sparse structures are the structures where the relations contain "few" tuples. Over these structures the expressive powers of GSO and MSO coincide [27]. A tree-interpretable structure is uniformly sparse if and only if it has finite tree width.

Structures FO-interpretable in the weak power set of a tree have a FO-theory which is reducible to the WMSO-theory of the tree. Special techniques are developed to study those structures. In particular, we present reductions to questions about WMSO-interpretability in trees.

Finally, we employ compatible operations to define classes of infinite structures with decidable theories. We use the following classes of structures to illustrate this method.

Prefix-recognisable structures. The original definition of this class is based on term rewriting systems [17]. In our framework, these are all structures that can be obtained from the infinite binary tree by an MSO-interpretation, or equivalently by an FO-interpretation [20]. As the infinite binary tree has a decidable MSO-theory [41], the same holds for all prefix-recognisable structures. A fourth definition can be given in terms of the configuration graphs of pushdown automata [40]. A graph is prefix-recognisable if and only if it can be obtained from such a configuration graph by factoring out ε-transitions. The class of HR-*equational structures* is a proper subclass of the prefix-recognisable structures [22]. By definition, each prefix-recognisable structure is tree-interpretable and it is HR-equational if and only if it has finite tree width or, equivalently, if it is uniformly sparse.

The Caucal hierarchy. This hierarchy is defined by combining MSO-interpretations with the iteration operation. Starting from the set of all finite structures one alternatingly applies these two operations [18]. The first level of this strict hierarchy corresponds to the class of prefix-recognisable structures. As both operations are compatible with MSO, one obtains a large class of infinite graphs with decidable MSO-theories. Each structure in the Caucal hierarchy is tree-interpretable.

Automatic structures. According to the original definition, the universe of an automatic structure is a regular set of words and the relations are defined by finite automata that read tuples of words in a synchronous way [36]. In the same way one can define tree-automatic structures using tree automata instead of word automata (and an appropriate definition of automata reading tuples of trees).

In our approach, automatic structures are obtained via an FO-interpretation from the weak power set of the structure $\langle \omega, < \rangle$ (the natural numbers with order). In the same way, tree-automatic structures can be obtained from the infinite binary tree [21]. By the choice of the operations it follows that each (tree-)automatic structure has a decidable FO-theory.

Tree-automatic hierarchy. Combining the previous ideas, one can consider the hierarchy of structures obtained by applying the weak power-set operation followed by an FO-interpretation to all trees in the Caucal hierarchy. It can be shown that this yields a strict hierarchy of structures with a decidable FO-theory.

The article is structured as follows. In the next section we introduce basic terminology and definitions. Section 3 is devoted to the presentation of the operations and basic results concerning their compatibility. Further results that can be obtained on restricted classes of structures are presented in Section 4. The use of compatible operations for defining classes of structures with decidable theories is illustrated in Section 5.

2 Preliminaries

Let us fix notation. We define $[n] := \{0, \dots, n-1\}$, and $\mathcal{P}(X)$ denotes the power set of X. Tuples $\bar{a} = \langle a_0, \dots, a_{n-1} \rangle \in A^n$ will be identified with functions $[n] \to A$. We shall only consider relational structures $\mathfrak{a} = \langle A, R_0, \dots, R_{n-1} \rangle$ with finitely many relations $R_0, \dots, R_{n-1}$ and where the universe A is at most countable.

An important special case of structures are trees. Let D be a set. We denote by D^* the set of all finite sequences of elements of D. The empty sequence is $\langle\rangle$. The *prefix ordering* is the relation $\preceq \subseteq D^* \times D^*$ defined by

$$x \preceq y \quad :\text{iff} \quad y = xz \text{ for some } z \in D^*.$$

An *unlabelled tree* is a structure $\mathfrak{t}$ isomorphic to $\langle T, \preceq \rangle$ where $T \subseteq D^*$ is prefix closed, for some set D. A *tree* is a structure of the form $\langle T, \preceq, \bar{P} \rangle$ where $\langle T, \preceq \rangle$ is an unlabelled tree and the P_i are unary predicates.

A tree is *deterministic* if it is of the form $\langle T, \preceq, (\mathrm{child}_d)_{d \in D}, \bar{P} \rangle$ where D is finite and

$$\mathrm{child}_d := \{ud \mid u \in D^*\}.$$

The *complete binary tree* is $\mathfrak{t}_2 := \big\langle \{0,1\}^*, \mathrm{child}_0, \mathrm{child}_1, \preceq \big\rangle$.

We shall consider several logics. Besides *first-order logic* FO we shall use *monadic second-order logic* MSO which extends FO by set variables and set quantifiers, *weak monadic second-order logic* WMSO which extends FO by variables for finite sets and the corresponding quantifiers, and *guarded second-order logic* GSO. The syntax of GSO is the same as that of full second-order logic where we allow variables for relations of arbitrary arity and quantification over such variables. The semantics of such a second-order quantifier is as follows (for a more detailed definition see [33]). We call a tuple $\bar{a}$ *guarded* if there exists a relation R_i and a tuple $\bar{c} \in R_i$ such that every component a_i of $\bar{a}$ appears in $\bar{c}$. A relation is guarded if it only contains guarded tuples. We define a formula of the form $\exists S\varphi(S)$ to be true if there exists a guarded relation S satisfying φ. Similarly, $\forall S\varphi(S)$ holds if every guarded relation S satisfies φ. For instance, given a graph $\mathfrak{g} = \langle V, E\rangle$ we can use guarded quantifiers to quantify over sets of edges.

Definition 2.1. Let L and L' be two logics. A (total) unary operation f on structures is (L, L')-*compatible* if, for every sentence $\varphi \in L'$, we can effectively compute a sentence $\varphi^f \in L$ such that

$$f(\mathfrak{a}) \models \varphi \quad \text{iff} \quad \mathfrak{a} \models \varphi^f, \quad \text{for every structure } \mathfrak{a}.$$

We call f (L, L')-*bicompatible* if, furthermore, for every sentence $\varphi \in L$, we can effectively compute a sentence $\varphi' \in L'$ such that

$$\mathfrak{a} \models \varphi \quad \text{iff} \quad f(\mathfrak{a}) \models \varphi', \quad \text{for every structure } \mathfrak{a}.$$

For the case that $L = L'$ we simply speak of L-*compatible* and L-*bicompatible* operations.

The interest in compatible operations is mainly based on the fact that they preserve the decidability of theories.

Lemma 2.2. Let f be a (L, L')-compatible operation. If the L-theory of $\mathfrak{a}$ is decidable then so is the L'-theory of $f(\mathfrak{a})$.

Another natural property of this definition is the ability to compose compatible operations.

Lemma 2.3. If f is an (L, L')-compatible operation and g an (L', L'')-compatible one then $g \circ f$ is (L, L'')-compatible. If f and g are bicompatible then so is $g \circ f$.

3 Operations

In this section we survey various operations on structures and their effect on logical theories (see also [39, 47, 34]). We attempt to provide a generic and self-contained panorama. We do not intend to present all results in their strongest and most precise form. For instance, many compatibility statements can be strengthened to compatibility for (i) the bounded quantifier fragments of the corresponding logics; (ii) their extensions by cardinality and counting quantifiers; or (iii) operations depending on parameters. The statements we present could also be refined by studying their complexity in terms of the size of formulae. This goes beyond the scope of this survey.

3.1 Generic operations

We start with *interpretations,* which are among the most versatile operations we shall introduce. In fact, all other operations we present are quite limited on their own. Only when combined with an interpretation they reveal their full strength.

Definition 3.1. Let L be a logic and Σ and Γ signatures. An *L-interpretation* from Σ to Γ is a list

$$\mathcal{I} = \big\langle \delta(x), (\varphi_R(\bar{x}))_{R\in\Gamma} \big\rangle$$

of L-formulae over the signature Σ where δ has one free (first-order) variable and the number of free variables of φ_R coincides with the arity of R.

Such an interpretation induces an operation mapping a Σ-structure $\mathfrak{a}$ to the Γ-structure

$$\mathcal{I}(\mathfrak{a}) := \langle D, R_0, \dots, R_{r-1} \rangle$$

where

$$D := \big\{ a \in A \bigm| \mathfrak{a} \models \delta(a) \big\} \quad \text{and} \quad R_i := \big\{ \bar{a} \in A^n \bigm| \mathfrak{a} \models \varphi_{R_i}(\bar{a}) \big\}.$$

The *coordinate map* of $\mathcal{I}$ is the function mapping those elements of A that encode an element of $\mathcal{I}(\mathfrak{a})$ to that element. It is also denoted by $\mathcal{I}$.

An L-interpretation with $\delta(x) = \texttt{true}$ is called an *L-expansion.* An *L-marking* is an L-expansion that only adds unary predicates without changing the existing relations of a structure.

Proposition 3.2. Let $\mathcal{I}$ be an L-interpretation where L is one of FO, WMSO, or MSO. For every L-formula $\varphi(\bar{x})$, there exists an L-formula $\varphi^{\mathcal{I}}(\bar{x})$ such that

$$\mathcal{I}(\mathfrak{a}) \models \varphi(\mathcal{I}(\bar{a})) \quad \text{iff} \quad \mathfrak{a} \models \varphi^{\mathcal{I}}(\bar{a}),$$

for all structures $\mathfrak{a}$ and all elements $a_i \in A$ with $\mathfrak{a} \models \delta(a_i)$.

The formula $\varphi^{\mathcal{I}}$ is easily constructed from φ by performing the following operations: (i) replacing every atom $R\bar{x}$ by its definition φ_R; (ii) relativising all first-order quantifiers to elements satisfying δ, and all set quantifiers to sets of such elements.

Corollary 3.3. L-interpretations are L-compatible if L is one of FO, WMSO, or MSO.

A nice property of interpretations is that they are closed under composition.

Proposition 3.4. Let L be one of FO, WMSO, or MSO. For all L-interpretations $\mathcal{I}$ and $\mathcal{J}$, there exists an L-interpretation $\mathcal{K}$ such that $\mathcal{K} = \mathcal{I} \circ \mathcal{J}$.

The second generic operation we are considering is the quotient operation.

Definition 3.5. Let $\mathfrak{a} = \langle A, \bar{R} \rangle$ be a structure and $\sim$ a binary relation. If $\sim$ is a congruence relation of $\mathfrak{a}$ then we can form the *quotient* of $\mathfrak{a}$ by $\sim$ which is the structure

$$\mathfrak{a}/{\sim} := \langle A/{\sim}, \bar{S} \rangle$$

where, if we denote the $\sim$-class of a by $[a]$, we have

$$\begin{aligned} A/{\sim} &:= \big\{ [a] \bigm| a \in A \big\}, \\ S_i &:= \big\{ \langle [a_0], \dots, [a_{n-1}] \rangle \bigm| \langle a_0, \dots, a_{n-1} \rangle \in R_i \big\}. \end{aligned}$$

By convention, if $\sim$ is not a congruence, we set $\mathfrak{a}/{\sim}$ to be $\mathfrak{a}$.

We shall only consider quotients by relations $\sim$ that are already present in the structure. This is no loss of generality since we can use a suitable interpretation to add any definable equivalence relation. For a relation symbol R and a structure $\mathfrak{a}$ we denote by $R^{\mathfrak{a}}$ the relation of $\mathfrak{a}$ corresponding to R.

Proposition 3.6. Let L be one of FO, WMSO or MSO, and $\sim$ a binary relation symbol. The quotient operation $\mathfrak{a} \mapsto \mathfrak{a}/{\sim^{\mathfrak{a}}}$ is L-compatible.

Remark 3.7.

(a) The convention in the case that $\sim$ is not a congruence causes no problems for the logics we are considering since in each of them we can express the fact that a given binary relation is a congruence.

(b) In order to factorise by a definable congruence relation that is not present in the structure we can precede the quotient operation by a suitable interpretation that expands the structure by the congruence.

(c) It is also possible to define quotients with respect to equivalence relations that are no congruences. This case is also subsumed by our definition since, given an equivalence relation $\sim$, we can use an FO-interpretation $\mathcal{I}$ to modify the relations of a structure $\mathfrak{a}$ in such a way that $\sim$ becomes a congruence and the quotient $\mathcal{I}(\mathfrak{a})/{\sim}$ equals $\mathfrak{a}/{\sim}$.

Another property of the quotient operation is that it commutes with interpretations in the sense of the following proposition.

Proposition 3.8. Let L be one of FO, WMSO or MSO. For every L-interpretation $\mathcal{I}$ and each binary relation symbol $\sim$, there exists an L-interpretation $\mathcal{J}$ such that

$$\mathcal{I}(\mathfrak{a}/{\sim^{\mathfrak{a}}}) = \mathcal{J}(\mathfrak{a})/{\sim^{\mathcal{J}(\mathfrak{a})}}, \quad \text{for every structure } \mathfrak{a}.$$

In combination with Proposition 3.4, it follows that every sequence of L-interpretations and quotients can equivalently be written as a single L-interpretation followed by a quotient. This is the reason why one often defines a more general notion of an interpretation that combines the simple interpretations above with a quotient operation by a definable congruence. It follows that these generalised interpretations are also closed under composition.

3.2 Monadic second-order logic

We now turn to operations compatible specifically with monadic second-order logic. The simplest one is the disjoint union. We also present a much more general kind of union called a *generalised sum.* Finally we present Muchnik's iteration construction.

Definition 3.9. The *disjoint union* of two structures $\mathfrak{a} = \langle A, \bar{R}\rangle$ and $\mathfrak{b} = \langle B, \bar{S}\rangle$ is the structure

$$\mathfrak{a} \uplus \mathfrak{b} := \langle A \mathbin{\dot\cup} B, \bar{T}\rangle \quad \text{where} \quad T_i := R_i \mathbin{\dot\cup} S_i.$$

The theory of the sum can be reduced to the theory of the two arguments using the following proposition.

Proposition 3.10. Let L be one of FO, MSO, WMSO or GSO. For every L-formula φ there exist L-formulae $\psi_0, \dots, \psi_n$ and $\vartheta_0, \dots, \vartheta_n$ such that

$$\mathfrak{a} \uplus \mathfrak{b} \models \varphi \quad \text{iff} \quad \text{there is some } i \leq n \text{ such that } \mathfrak{a} \models \varphi_i \text{ and } \mathfrak{b} \models \vartheta_i.$$

Unions behave well with respect to MSO, but the same does not hold for products. A notable exception are products with a fixed finite structure. In the following definition we introduce the simpler product with a finite set, which, up to FO-interpretations, is equivalent to using a finite structure.

Definition 3.11. Let $\mathfrak{a} = \langle A, \bar{R}\rangle$ be a structure and $k < \omega$ a number. The *product* of $\mathfrak{a}$ with k is the structure

$$k \times \mathfrak{a} := \big\langle [k] \times A, \bar{R}', \bar{P}, I\big\rangle,$$

where

$$\begin{aligned}
R'_j &:= \{(\langle i, a_0\rangle, \dots, \langle i, a_{n-1}\rangle) \mid \bar{a} \in R_j \text{ and } i < k\},\\
P_i &:= \{i\} \times A,\\
I &:= \{(\langle i, a\rangle, \langle j, a\rangle) \mid a \in A,\ i, j < k\}.
\end{aligned}$$

Proposition 3.12. For every MSO-formula $\varphi(X^0, \dots, X^{n-1})$ and all $k < \omega$, there exists an MSO-formula $\varphi_k(\bar{X}^0, \dots, \bar{X}^{n-1})$ such that

$$k \times \mathfrak{a} \models \varphi(P^0, \dots, P^{n-1}) \quad \text{iff} \quad \mathfrak{a} \models \varphi_k(\bar{Q}^0, \dots, \bar{Q}^{n-1}),$$

where $Q^\ell_i := \{a \in A \mid \langle i, a\rangle \in P^\ell\}$. The same holds for WMSO.

This result can be proven as a consequence of Theorem 3.16 below.

Corollary 3.13. For $k < \omega$, the product operation $\mathfrak{a} \mapsto k \times \mathfrak{a}$ is MSO-compatible. It is MSO-bicompatible if $k \neq 0$. The same holds for WMSO and GSO.

Finite products are sometimes combined with MSO-interpretations resulting in what is called a *parameterless* MSO-*transduction* [24]. Such a transduction maps a structure $\mathfrak{a}$ to the structure $\mathcal{I}(k \times \mathfrak{a})$, where k is a natural number, and $\mathcal{I}$ is an MSO-interpretation. It follows that parameterless MSO-transductions are MSO-compatible. Furthermore, they are closed under composition since, for every MSO-interpretation $\mathcal{J}$, there exists an MSO-interpretation $\mathcal{K}$ with

$$k \times \mathcal{J}(l \times \mathfrak{a}) \cong \mathcal{K}(kl \times \mathfrak{a}).$$

The operation of disjoint union can be generalised to a union of infinitely many structures. Furthermore, we can endow the index set with a structure of its own. This operations also generalises the product with a finite structure.

Definition 3.14. Let $\mathfrak{i} = \langle I, \bar{S} \rangle$ be a structure and $(\mathfrak{a}^{(i)})_{i \in I}$ a sequence of structures $\mathfrak{a}^{(i)} = \langle A^{(i)}, \bar{R}^{(i)} \rangle$ indexed by elements i of $\mathfrak{i}$.

The *generalised sum* of $(\mathfrak{a}^{(i)})_{i \in I}$ is the structure

$$\sum_{i \in \mathfrak{i}} \mathfrak{a}^{(i)} := \langle U, \sim, \bar{R}', \bar{S}' \rangle$$

with universe $U := \{ \langle i, a \rangle \mid i \in I,\ a \in A^{(i)} \}$ and relations

$$\begin{aligned} \langle i, a \rangle \sim \langle j, b \rangle \quad &: \text{iff} \quad i = j, \\ R'_\ell &:= \big\{ (\langle i, a_0 \rangle, \dots, \langle i, a_{n-1} \rangle) \bigm| i \in I \text{ and } \bar{a} \in R^{(i)}_\ell \big\}, \\ S'_\ell &:= \big\{ (\langle i_0, a_0 \rangle, \dots, \langle i_{n-1}, a_{n-1} \rangle) \bigm| \bar{\imath} \in S_\ell \big\}. \end{aligned}$$

To illustrate the definition let us show how a generalised sum can be used to define the standard ordered sum of linear orderings.

Example 3.15. Let $\mathfrak{i} = \langle I, \sqsubset \rangle$ and $\mathfrak{a}^{(i)} = \langle A^{(i)}, <^{(i)} \rangle$, for $i \in I$, be linear orders. Then

$$\sum_{i \in \mathfrak{i}} \mathfrak{a}^{(i)} = \langle U, \sim, <, \sqsubset \rangle$$

where $U = \{ \langle i, a \rangle \mid a \in A^{(i)} \}$ and we have

$$\begin{aligned} \langle i, a \rangle < \langle j, b \rangle \quad &\text{iff} \quad i = j \text{ and } a <^{(i)} b, \\ \langle i, a \rangle \sqsubset \langle j, b \rangle \quad &\text{iff} \quad i \sqsubset j. \end{aligned}$$

If we introduce the new (definable) relation

$$\langle i,a\rangle \prec \langle j,b\rangle \quad :\text{iff} \quad \langle i,a\rangle \sqsubset \langle j,b\rangle \text{ or } \langle i,a\rangle < \langle j,b\rangle$$

then the structure $\langle U, \prec\rangle$ is isomorphic to the ordered sum of the orders $\mathfrak{a}^{(i)}$.

The generalisation of Proposition 3.10 takes the following form.

Theorem 3.16. For every MSO-sentence φ, we can construct a finite sequence of MSO-formulae $\chi_0, \dots, \chi_{s-1}$ and an MSO-formula ψ such that

$$\sum_{i\in\mathfrak{i}} \mathfrak{a}^{(i)} \models \varphi \quad \text{iff} \quad \big\langle \mathfrak{i}, [\![\chi_0]\!], \dots, [\![\chi_{s-1}]\!]\big\rangle \models \psi,$$

where $[\![\chi]\!] := \{i \in I \mid \mathfrak{a}^{(i)} \models \chi\}$.

Remark 3.17. This theorem is a special case of a result of Shelah [45] following the ideas developped by Feferman and Vaught [32], see [47, 34] for a readable exposition. As mentioned above it implies Proposition 3.10 (for MSO) as well as Proposition 3.12.

We finally survey the iteration operation originally introduced by Muchnik. Given a structure $\mathfrak{a}$ this operation produces a structure consisting of infinitely many copies of $\mathfrak{a}$ arranged in a tree-like fashion.

Definition 3.18. The *iteration* of a structure $\mathfrak{a} = \langle A, \bar{R}\rangle$ is the structure $\mathfrak{a}^* := \langle A^*, \preceq, \mathrm{cl}, \bar{R}^*\rangle$ where $\preceq$ is the prefix ordering and

$$\begin{aligned} \mathrm{cl} &:= \{waa \mid w \in A^*,\ a \in A\}, \\ R_i^* &:= \{(wa_0, \dots, wa_r) \mid w \in A^*,\ \bar{a} \in R_i\}. \end{aligned}$$

Theorem 3.19 (Muchnik)**.** The iteration operation is MSO-bicompatible and WMSO-bicompatible.

Remark 3.20. The Theorem of Muchnik was announced without proof in [44]. The first published proof, based on automata-theoretic techniques, is due to Walukiewicz [49, 50]. An exposition can be found in [2] and a generalisation to various other logics is given in [9].

Example 3.21.

(a) Let $\mathfrak{a} := \langle [2], P_0, P_1\rangle$ be a structure with two elements and unary predicates $P_0 := \{0\}$ and $P_1 := \{1\}$ to distinguish them. Its iteration $\mathfrak{a}^* = \langle [2]^*, \preceq, \mathrm{cl}, P_0^*, P_1^*\rangle$ resembles the complete binary tree $\mathfrak{t}_2$. Applying a simple (quantifier free) FO-interpretation $\mathcal{I}$ we obtain $\mathfrak{t}_2 = \mathcal{I}(\mathfrak{a}^*)$.

(b) Let $\mathfrak{g}$ be a graph. The *unravelling* of $\mathfrak{g}$ is the graph $\mathcal{U}(\mathfrak{g}) := \langle U, F\rangle$ where U is the set of all paths through $\mathfrak{g}$ and F consists of all pairs $\langle u, v\rangle$ such that the path v is obtained from u by appending a single edge of $\mathfrak{g}$.

The unravelling of $\mathfrak{g}$ can be obtained from $\mathfrak{g}$ via an iteration followed by an interpretation. Note that $\mathfrak{g}^*$ consists of all sequences of vertices of $\mathfrak{g}$. All that is needed to get $\mathcal{U}(\mathfrak{g})$ is to define the subset of those sequences that are paths through $\mathfrak{g}$. This can be done by the formula

$$\delta(w) := \forall u \forall v \big[\mathrm{suc}(u,v) \wedge v \preceq w \rightarrow \exists u' (\mathrm{suc}(u,u') \wedge \mathrm{cl}(u') \wedge E^* u' v)\big].$$

In view of the examples above we directly obtain the following corollaries.

Corollary 3.22 (Rabin, [41])**.** The MSO-theory of the infinite binary tree $\mathfrak{t}_2$ is decidable.

Corollary 3.23 (Courcelle-Walukiewicz, [31])**.** The unravelling operation $\mathcal{U}$ is MSO-compatible and WMSO-compatible.

Finally, let us mention that iterations commute with interpretations in the following sense.

Lemma 3.24 (Blumensath, [5])**.** For every MSO-interpretation $\mathcal{I}$, there exists an MSO-interpretation $\mathcal{J}$ such that

$$\mathcal{I}(\mathfrak{a})^* = \mathcal{J}(\mathfrak{a}^*), \quad \text{for all structures } \mathfrak{a}.$$

3.3 First-order logic

In this section we concentrate on first-order logic. We start by introducing the power-set operation which relates MSO-theories to FO-theories. This operation provides a systematic way to relate results about FO-compatibility to those about MSO-compatibility above.

Definition 3.25. Let $\mathfrak{a} = \langle A, \bar{R} \rangle$ be a structure. The *power set* of $\mathfrak{a}$ is the structure

$$\mathcal{P}(\mathfrak{a}) := \langle \mathcal{P}(A), \bar{R}', \subseteq \rangle,$$

where $R'_i := \big\{ \langle \{a_0\}, \dots, \{a_{n-1}\} \rangle \;\big|\; \bar{a} \in R_i \big\}$.

The *weak power set* $\mathcal{P}_{\mathrm{w}}(\mathfrak{a})$ of $\mathfrak{a}$ is the substructure of $\mathcal{P}(\mathfrak{a})$ induced by the set of all *finite* subsets of A.

Since elements of $\mathcal{P}(\mathfrak{a})$ are sets of elements of $\mathfrak{a}$, FO-formulae over $\mathcal{P}(\mathfrak{a})$ directly correspond to MSO-formulae over $\mathfrak{a}$ (and similarly for WMSO).

Proposition 3.26.

(a) For every FO-formula $\varphi(\bar{x})$, we can construct an MSO-formula $\varphi'(\bar{X})$ such that

$$\mathcal{P}(\mathfrak{a}) \models \varphi(\bar{P}) \quad \text{iff} \quad \mathfrak{a} \models \varphi'(\bar{P}),$$

for every structure $\mathfrak{a}$ and all subsets $P_i \subseteq A$.

(b) For every MSO-formula $\varphi(\bar{X})$, we can construct an FO-formula $\varphi'(\bar{x})$ such that

$$\mathfrak{a} \models \varphi(\bar{P}) \quad \text{iff} \quad \mathcal{P}(\mathfrak{a}) \models \varphi'(\bar{P}),$$

for every structure $\mathfrak{a}$ and all subsets $P_i \subseteq A$.

(c) Analogous statements hold for WMSO-formulae and the weak power-set operation $\mathcal{P}_{\mathrm{w}}$.

Corollary 3.27. The power-set operation $\mathcal{P}$ is (MSO, FO)-bicompatible and the weak power-set operation $\mathcal{P}_{\mathrm{w}}$ is (WMSO, FO)-bicompatible.

Lemma 3.28. For every MSO-interpretation $\mathcal{I}$, there exists an FO-interpretation $\mathcal{J}$ such that

$$\mathcal{P} \circ \mathcal{I} = \mathcal{J} \circ \mathcal{P}.$$

A similar statement holds with WMSO instead of MSO and $\mathcal{P}_{\mathrm{w}}$ instead of $\mathcal{P}$.

Remark 3.29. In [19, 21] (*finite*) *set interpretations* are introduced which are halfway between first-order and monadic second-order interpretations. A (finite) set interpretation is of the form

$$\mathcal{I} = \big\langle \delta(X), (\varphi_R(\bar{X}))_{R\in\Gamma} \big\rangle$$

where δ, φ_R are (weak) monadic second-order formulae with *set variables* as free variables. Correspondingly the elements of the structure $\mathcal{I}(\mathfrak{a})$ are encoded by (finite) subsets of the original structure. With the operations of the present article we can express such a set interpretation as, respectively,

$$\mathcal{J} \circ \mathcal{P} \quad \text{or} \quad \mathcal{J} \circ \mathcal{P}_{\mathrm{w}}$$

where $\mathcal{J}$ is an FO-interpretation. From Corollary 3.27 it follows that

- set interpretations are (MSO, FO)-compatible and
- finite set interpretations are (WMSO, FO)-compatible.

From Lemma 3.28 and Proposition 3.4 it follows that, if $\mathcal{I}$ is an FO-interpretation, $\mathcal{J}$ a set interpretation, and $\mathcal{K}$ an MSO-interpretation then their composition $\mathcal{I} \circ \mathcal{J} \circ \mathcal{K}$ is also a set interpretation. The same holds for finite set interpretations provided $\mathcal{K}$ is a WMSO-interpretation.

We have mentioned that products are not compatible with monadic second-order logic. But they are compatible with first-order logic. In fact, historically they were among the first operations shown to be compatible with some logic.

Definition 3.30. The (*direct, or Cartesian*) *product* of two structures $\mathfrak{a} = \langle A, \bar{R}\rangle$ and $\mathfrak{b} = \langle B, \bar{S}\rangle$ is the structure

$$\mathfrak{a} \times \mathfrak{b} := \langle A \times B, \bar{T}\rangle,$$

where $T_i := \big\{ (\langle a_0, b_0\rangle, \dots, \langle a_{n-1}, b_{n-1}\rangle) \bigm| \bar{a} \in R_i \text{ and } \bar{b} \in S_i \big\}$.

Proposition 3.31. For every FO-formula φ, we can construct FO-formulae $\psi_0, \dots, \psi_n$ and $\vartheta_0, \dots, \vartheta_n$ such that

$$\mathfrak{a} \times \mathfrak{b} \models \varphi \quad \text{iff} \quad \text{there is some } i \leq n \text{ such that } \mathfrak{a} \models \psi_i \text{ and } \mathfrak{b} \models \vartheta_i.$$

Product and disjoint union are related via the power-set construction.

Proposition 3.32. There exist FO-interpretations $\mathcal{I}, \mathcal{J}$ and $\mathcal{K}$ such that

$$\mathcal{P}(\mathfrak{a} \uplus \mathfrak{b}) \cong \mathcal{I}(\mathcal{J}(\mathcal{P}(\mathfrak{a})) \times \mathcal{K}(\mathcal{P}(\mathfrak{b}))), \quad \text{for all structures } \mathfrak{a} \text{ and } \mathfrak{b}.$$

A similar statement holds with $\mathcal{P}_{\mathrm{w}}$ instead of $\mathcal{P}$.

Remark 3.33.

(a) The interpretations $\mathcal{J}$ and $\mathcal{K}$ are only needed to avoid problems with empty relations. If a relation is empty in one of the factors then the corresponding relation of the product is also empty and cannot be reconstructed. The quantifier-free interpretations are used to create dummy relations to avoid this phenomenon.

(b) Using this result together with the (MSO, FO)-bicompatibility of $\mathcal{P}$ we can deduce the MSO variant of Proposition 3.10 from Proposition 3.31. A similar argument yields the WMSO version.

Similar to finite products that are MSO-compatible we can define a finite exponentiation which is FO-compatible.

Definition 3.34. Let $\mathfrak{a} = \langle A, \bar{R}\rangle$ be a structure and $k < \omega$ a number. The *exponent* of $\mathfrak{a}$ to the k is the structure

$$\mathfrak{a}^k := \langle A^k, \bar{R}', \bar{E}\rangle$$

with relations

$$\begin{aligned} R'_{\ell i} &:= \{ (\bar{a}^0, \dots, \bar{a}^{n-1}) \mid (a_i^0, \dots, a_i^{n-1}) \in R_\ell \}, \\ E_{ij} &:= \{ (\bar{a}, \bar{b}) \mid a_i = b_j \}. \end{aligned}$$

The good behaviour of the finite exponent operation is illustrated by the next proposition.

Proposition 3.35. For each $k < \omega$ and every FO-formula $\varphi(x^0, \dots, x^{n-1})$, there exists an FO-formula $\varphi_k(\bar{x}^0, \dots, \bar{x}^{n-1})$ such that

$$\mathfrak{a}^k \models \varphi(\bar{a}^0, \dots, \bar{a}^{n-1}) \quad \text{iff} \quad \mathfrak{a} \models \varphi_k(\bar{a}^0, \dots, \bar{a}^{n-1}),$$

for every structure $\mathfrak{a}$ and all $\bar{a}^i \in A^k$.

Corollary 3.36. Let $k < \omega$. The exponent operation $\mathfrak{a} \mapsto \mathfrak{a}^k$ is FO-compatible. It is FO-bicompatible for $k \geq 1$.

The relation between finite exponentiation and finite products is given in the next proposition. (This allows us to deduce Proposition 3.12 from Proposition 3.35).

Proposition 3.37. For every $k < \omega$, there exists an FO-interpretation $\mathcal{I}$ such that

$$\mathcal{P}(k \times \mathfrak{a}) \cong \mathcal{I}(\mathcal{P}(\mathfrak{a})^k), \quad \text{for every structure } \mathfrak{a}.$$

The same holds for the weak power-set operation.

In the same way as the combination of MSO-interpretations and finite products leads to the notion of a parameterless MSO-transduction, one can perform a finite exponentiation before an FO-interpretation. The resulting operation is called a *k-dimensional* FO*-interpretation.* The composition of a k-dimensional FO-interpretation with an l-dimensional one yields a kl-dimensional FO-interpretation. In the same spirit as above, multi-dimensional interpretations are correlated to parameterless MSO-transductions via the power-set operation.

As for unions we can generalise products to infinitely many factors. In the original definition of a generalised product by Feferman and Vaught [32] FO-formulae are used to determine the relations in the product structure. We shall adopt a simpler yet richer definition where the product structure is completely determined by the index structure and the factors.

Definition 3.38. Let $\mathfrak{a}^{(i)} = \langle A^{(i)}, \bar{R}^{(i)} \rangle$, $i \in I$, be structures, and let

$$\mathfrak{i} = \langle \mathcal{P}(I), \subseteq, \bar{S} \rangle$$

be the expansion of the power-set algebra $\mathcal{P}(I)$ by arbitrary relations $\bar{S}$. We define the *generalised product* of the $\mathfrak{a}^{(i)}$ over $\mathfrak{i}$ to be the structure

$$\prod_{i \in \mathfrak{i}} \mathfrak{a}^{(i)} := \langle U, \subseteq, \bar{S}, \bar{R}', E_= \rangle$$

with universe

$$U := \mathcal{P}(I) \uplus \prod_{i \in I} A^{(i)}$$

where the relations $\subseteq$ and $\bar{S}$ are those of $\mathfrak{i}$ and we have

$$\begin{aligned} R'_k &:= \bigl\{(X, \bar{a}) \bigm| X = [\![R\bar{a}]\!]\bigr\}, \\ E_= &:= \bigl\{(X, a, b) \bigm| X = [\![a = b]\!]\bigr\}, \end{aligned}$$

and $[\![\chi(\bar{a})]\!] := \{i \in I \mid \mathfrak{a}^{(i)} \models \chi(\bar{a}^{(i)})\}$.

Before stating that the generalised products are compatible with first-order logic let us give two examples.

Example 3.39. Let $\mathfrak{g}_0 = \langle V_0, E_0\rangle$ and $\mathfrak{g}_1 = \langle V_1, E_1\rangle$ be two directed graphs. There are two standard ways to form their product: we can take the *direct* or *synchronous* product with edge relation $E_\mathrm{s} := E_0 \times E_1$, and we can take the *asynchronous* product with edge relation $E_\mathrm{a} := (E_0 \times \mathrm{id}) \cup (\mathrm{id} \times E_1)$. Both kinds of products can be obtained from the generalised product via a first-order interpretation.
For the direct product, we define the edge relation by the formula

$$\varphi_{E_\mathrm{s}}(x, y) := \exists z[\mathrm{All}(z) \wedge Ezxy]$$

where the formula

$$\mathrm{All}(x) := x \subseteq x \wedge \forall y(x \subseteq y \to x = y)$$

states that $x = I$ is the maximal element of $\mathcal{P}(I)$. (Note that the condition $x \subseteq x$ is needed to ensure that $x \in \mathcal{P}(I)$.)
Similarly, we define the edge relation of the asynchronous product by

$$\begin{aligned} \varphi_{E_\mathrm{a}}(x, y) := \exists u \exists v \bigl[&E_= vxy \wedge Euxy \wedge \mathrm{Sing}(u) \\ &\wedge \forall z(z \subseteq z \to (u \nsubseteq z \leftrightarrow z \subseteq v))\bigr] \end{aligned}$$

where the formula

$$\mathrm{Sing}(z) := z \subseteq z \wedge \forall u \forall v[v \subseteq u \subseteq z \to (v = u \vee u = z)]$$

states that z is a singleton set in $\mathcal{P}(I)$.

Theorem 3.40 (Feferman-Vaught, [32])**.** For every FO-sentence φ, there exist an FO-sentence φ' and a finite sequence of FO-sentences $\chi_0, \dots, \chi_m$ such that

$$\prod_{i \in \mathfrak{i}} \mathfrak{a}^{(i)} \models \varphi \quad \text{iff} \quad \bigl\langle \mathfrak{i}, [\![\chi_0]\!], \dots, [\![\chi_m]\!]\bigr\rangle \models \varphi',$$

where $[\![\chi]\!] := \{i \in I \mid \mathfrak{a}^{(i)} \models \chi\}$.

Remark 3.41.

(a) If the structure $\mathfrak{i}$ is of the form $\mathcal{P}(\mathfrak{j})$, for some index structure $\mathfrak{j}$, then instead of a FO-formula φ' over $\mathfrak{i}$ we can also construct an MSO-formula over $\mathfrak{j}$, by Proposition 3.26. Hence, in this case we can reduce the FO-theory of the product $\prod_i \mathfrak{a}^i$ to the MSO-theory of the index structure $\mathfrak{j}$.

(b) Note that Theorem 3.16 follows from Theorem 3.40 and Proposition 3.26 since there exist FO-interpretations $\mathcal{I}, \mathcal{J}$ such that

$$\mathcal{P}\Big(\sum_{i\in\mathfrak{i}} \mathfrak{a}_i\Big) = \mathcal{I}\Big(\prod_{i\in\mathcal{P}(\mathfrak{i})} \mathcal{J}(\mathcal{P}(\mathfrak{a}_i))\Big).$$

(c) As an application of the generalised product we give an alternative proof to a result of Kuske and Lohrey [38] which states that, if we modify the iteration operation by omitting the clone relation cl then the resulting operation is (FO, Chain)-compatible. Here, Chain denotes the restriction of MSO where set variables only range over *chains,* i.e., sets that are totally ordered with respect to the prefix order $\preceq$. Let us denote by $\mathfrak{a}^\sharp$ the iteration of $\mathfrak{a}$ without cl and let $\mathcal{P}_{\text{ch}}(\mathfrak{a})$ be the substructure of $\mathcal{P}(\mathfrak{a})$ induced by all chains of $\mathfrak{a}$ (we assume that $\mathfrak{a}$ contains a partial order $\preceq$). A closer inspection reveals that, up to isomorphism, the structure $\mathcal{P}_{\text{ch}}(\mathfrak{a}^\sharp)$ can be obtained by a (2-dimensional) FO-interpretation from the generalised product of several copies of $\mathfrak{a}$ indexed by the structure $\mathcal{P}\langle\omega,<\rangle$. By Theorem 3.40 and the decidability of the MSO-theory of $\langle\omega,<\rangle$ [13], it follows that the operation $\mathfrak{a} \mapsto \mathfrak{a}^\sharp$ is (FO, Chain)-compatible.

3.4 Guarded second-order logic

We conclude this section by considering an operation that connects guarded second-order logic with monadic second-order logic.

Definition 3.42. The *incidence structure* of a structure $\mathfrak{a} = \langle A, R_0, \dots, R_r\rangle$ is

$$\text{In}(\mathfrak{a}) := \langle A \uplus G, \bar{R}', I_0, \dots, I_{n-1}\rangle$$

where $G := R_0 \uplus \dots \uplus R_r$ is the set of all tuples appearing in a relation of $\mathfrak{a}$, we have *unary* predicates

$$R'_i := \{\bar{a} \in G \mid \bar{a} \in R_i\},$$

and binary incidence relations $I_i \subseteq A \times G$ with

$$I_i := \{(a_i, \bar{a}) \in A \times G \mid \bar{a} \in G\}.$$

Example 3.43. The incidence structure of a graph $\mathfrak{g} = \langle V, E\rangle$ is the structure

$$\text{In}(\mathfrak{g}) = \langle V \uplus E, E', I_0, I_1\rangle$$

where the universe consists of all vertices and edges, the unary predicate E' identifies the edges, and the incidence relations I_0 and I_1 map each edge to its first and second vertex, respectively.

The GSO-theory of a structure is equivalent to the MSO-theory of its incidence structure.

Proposition 3.44. The operation In is (GSO, MSO)-bicompatible.

Remark 3.45. For the proof, note that we can encode every guarded n-tuple $\bar{a}$ by a triple $\langle R, \bar{c}, \sigma\rangle$ consisting of an m-ary relation R, a tuple $\bar{c} \in R$, and the function $\sigma : [n] \to [m]$ such that $a_i = c_{\sigma(i)}$. Consequently, we can encode a guarded relation $S \subseteq A^n$ by a (finite) family of subsets $P_{R,\sigma} \subseteq G$ where

$$P_{R,\sigma} := \{\bar{c} \in G \mid \langle R, \bar{c}, \sigma\rangle \text{ encodes an element of } S\}.$$

4 Structural properties

So far, we have presented a number of purely logical properties of operations. In this section, we survey other equivalences which hold under some additional hypothesis on the structures in question. First we study properties specific to trees. Then we present results for uniformly sparse structures. Finally we consider structures interpretable in the weak power set of a tree.

4.1 Tree-interpretable structures

When studying logical theories of trees various tools become available that fail for arbitrary structures. The most prominent example are *automata-theoretic methods.* For instance, one can translate every MSO-formula into an equivalent tree automaton (see [12, 46, 41]). Closer to the topic of the present paper are *composition arguments* which are based on Theorem 3.16 and its variants. Those techniques provide the necessary arguments for the tree-specific statements of the present section.

Definition 4.1. A structure is *tree-interpretable* if it is isomorphic to $\mathcal{I}(\mathfrak{t})$ for some MSO-interpretation $\mathcal{I}$ and tree $\mathfrak{t}$.

The notion of tree-interpretability is linked to two complexity measures: the *clique width* [28] (for graphs) and the *partition width* [5, 7] (for arbitrary structures). It turns out that a graph/structure is tree-interpretable if and only if its clique width/partition width is finite.

In the definition of tree-interpretable structures, we can require the tree $\mathfrak{t}$ to be deterministic without any effect. We can also replace MSO by WMSO without changing the definition. Our first result implies that the definition still remains equivalent if we use FO instead of MSO.

Theorem 4.2 (Colcombet, [20])**.** For every MSO-interpretation $\mathcal{I}$, there exists an FO-interpretation $\mathcal{J}$ and an MSO-marking $\mathcal{M}$ such that

$$\mathcal{I}(\mathfrak{t}) = (\mathcal{J} \circ \mathcal{M})(\mathfrak{t}), \quad \text{for every tree } \mathfrak{t}.$$

The same holds when MSO is replaced by WMSO.

Indeed, since the class of trees is closed under MSO-markings every tree-interpretable structure can be obtained by an FO-interpretation from a tree. Note that it is mandatory for this result that trees are defined in terms of the prefix order $\preceq$ instead of using just the immediate successor relation.

One motivation for the study of tree-interpretable structures is the fact that this class seems to capture the dividing line between simple and complicated MSO-theories. On the one hand, trees have simple MSO-theories and, therefore, so have all structures that can be interpreted in a tree. Conversely, it is conjectured that the MSO-theory of every structure that is not tree-interpretable is complicated.

Conjecture 4.3 (Seese, [43])**.** Every structure with a decidable MSO-theory is tree-interpretable.

Currently the best result in this direction was recently obtained by Courcelle and Oum [30]. It states that every graph that is not tree-interpretable has an undecidable $\mathrm{C_2MSO}$-theory where $\mathrm{C_2MSO}$ is the extension of MSO by predicates for counting modulo 2. Unfortunately their proof appears surprisingly difficult to generalise to arbitrary structures.

One evidence for Seese's conjecture is the fact that the class of tree-interpretable structures is closed under all known MSO-compatible operations.

Proposition 4.4. The class of tree-interpretable structures is closed under (i) disjoint unions, (ii) generalised sums, (iii) finite products, (iv) quotients, (v) MSO-interpretations, and (vi) iterations.

There is no difficulty in proving this proposition. In particular, it is easy to establish that the quotient of a tree-interpretable structure is also tree-interpretable. Indeed it is sufficient to guess a system C of representatives of the equivalence classes. Once we have expanded the tree by this new unary predicate C we can use a simple MSO-interpretation to obtain the quotient. However, if one wants the representatives C to be unique and MSO-definable this becomes impossible. This follows from the following result of Gurevich and Shelah [35] (see [15] for a simple proof): There is no MSO-formula $\varphi(x, X)$ such that, for every deterministic tree $\mathfrak{t}$ and all nonempty sets $P \subseteq \mathfrak{t}$, there is a unique element $a \in P$ such that $\mathfrak{t} \models \varphi(a, P)$.

The following theorem circumvents this difficulty. It is more precise than simply claiming the closure under quotients in that it states that we can choose the same deterministic tree. The result is given for FO, but it can also be derived for MSO and WMSO by a direct application of Theorem 4.2.

Theorem 4.5. Let $\mathcal{I}$ be an FO-interpretation and $\sim$ a binary relation symbol. There exists an FO-interpretation $\mathcal{J}$ such that

$$\mathcal{I}(\mathfrak{t})/{\sim^{\mathcal{I}(\mathfrak{t})}} \cong \mathcal{J}(\mathfrak{t}), \quad \text{for every deterministic tree } \mathfrak{t}.$$

Remark 4.6. For the proof of this result it is sufficient to assign to each $\sim$-class a unique element of the tree in an FO-definable way. First, one maps each class to its infimum (for the prefix order $\preceq$). With this definition several classes might be mapped to the same element. Using a technique similar to the one from [21] it is possible to distribute those elements in a FO-definable way and thereby to transform the original mapping into an injective one.

Another phenomenon is that the iteration and unravelling operations turn out to be equivalent in the context of MSO-interpretations over trees.

Theorem 4.7 (Carayol-Wöhrle, [16])**.** There exist MSO-interpretations $\mathcal{I}, \mathcal{J}$ such that

$$\mathfrak{t}^* \cong \mathcal{I}(\mathcal{U}(\mathcal{J}(\mathfrak{t}))), \quad \text{for every deterministic tree } \mathfrak{t}.$$

The first interpretation $\mathcal{J}$ adds backward edges and loops to every vertex of $\mathfrak{t}$. From the unravelling of this structure we can reconstruct the iteration of $\mathfrak{t}$ by an MSO-interpretation.

4.2 Tree width, uniform sparse structures, and complete bipartite subgraphs

In this section we introduce the *tree width* of a structure, a complexity measure similar to the clique width or partition width, which were related to the notion of tree-interpretability. Intuitively the tree width of a structure measures how much it resembles a tree (see [10] for a survey).

Definition 4.8. Let $\mathfrak{a} = \langle A, \bar{R} \rangle$ be a structure.

(a) A *tree decomposition* of $\mathfrak{a}$ is a family $(U_v)_{v \in T}$ of subsets $U_v \subseteq A$ indexed by an undirected tree T with the following properties:

1. $\bigcup_v U_v = A$.
2. For all $\bar{a} \in R_i$ in some relation of $\mathfrak{a}$, there is some $v \in T$ with $\bar{a} \subseteq U_v$.
3. For every element $a \in A$, the set $\{v \in T \mid a \in U_v\}$ is connected.

(b) The *width* of such a tree decomposition $(U_v)_{v \in T}$ is

$$\sup \{ |U_v| \mid v \in T \}.$$

(For aesthetic reasons the width is traditionally defined as supremum of $|U_v| - 1$. We have dropped the -1 since it makes many statements more complicated and omitting it does not influence the results.)

(c) The *tree width* $\operatorname{twd} \mathfrak{a}$ is the minimal width of a tree decomposition of $\mathfrak{a}$.

It turns out that, with respect to tree width, GSO plays a similar role as MSO does with respect to tree-interpretability. The incidence structure allows to go back and forth in this analogy.

Theorem 4.9. A structure $\mathfrak{a}$ has finite tree width iff $\operatorname{In}(\mathfrak{a})$ is tree-interpretable.

The corresponding result for classes of finite structures is due to Courcelle and Engelfriet [29]. The same ideas can be used to prove Theorem 4.9. Note that this theorem in particular implies that every structure with finite tree width is tree-interpretable. However the converse does not hold. For instance, the infinite clique is tree-interpretable but its tree width is infinite. The equivalent of Seese's Conjecture 4.3 for tree width has been proved by Seese.

Theorem 4.10 (Seese, [43])**.** Every structure with a decidable GSO-theory has finite tree width.

The proof is based on the Excluded Grid Theorem of Robertson and Seymour [42] and on the fact that the class of all finite grids has an undecidable MSO-theory (see also [25, 5]).

In the remaining of this section, we present two other complexity measures for countable structures: sparsity and the existence of big complete bipartite subgraphs in the Gaifman graph. A structure is uniformly sparse if, in every substructure, the number of guarded tuples is linearly bounded by the size of the substructure.

Definition 4.11. Let $k < \omega$. A structure $\mathfrak{a} = \langle A, R_0, \dots, R_{n-1} \rangle$ is called *uniformly k-sparse* if, for all finite sets $X \subseteq A$ and every $i < n$, we have

$$|R_i|_X| \leq k \cdot |X|.$$

A structure is *uniformly sparse* if it is uniformly k-sparse for some $k < \omega$.

The requirement of uniform sparsity is less restrictive than that of having a finite tree width: every structure of finite tree width is uniformly sparse, but the converse does not hold in general. Consider for instance the infinite grid $\mathbb{Z} \times \mathbb{Z}$ with an edge between (i, k) and (j, l) if $|i - j| + |k - l| = 1$. This graph is uniformly sparse, but has infinite tree width.

The work of Courcelle [27] shows that the property of being uniformly sparse is the correct notion for studying the relationship between GSO and MSO. While, in general, GSO is strictly more expressive than MSO, it collapses to MSO on uniformly sparse structures.

Theorem 4.12. Let $k < \omega$. For every GSO-sentence φ, we can construct an MSO-sentence φ' such that

$$\mathfrak{a} \models \varphi \quad \text{iff} \quad \mathfrak{a} \models \varphi', \quad \text{for all countable uniformly } k\text{-sparse structures } \mathfrak{a}.$$

The proof of this result relies on the possibility, once k is fixed, to interpret $\mathrm{In}(\mathfrak{a})$ in $n \times \mathfrak{a}$ for a suitably chosen n, provided one has correctly labelled $n \times \mathfrak{a}$ by a certain number of monadic *parameters.* Then Theorem 4.12 follows by Proposition 3.44. This technique is formalised by the following lemma.

Lemma 4.13. For all $k < \omega$, there exist $n < \omega$, an MSO-interpretation $\mathcal{I}$, and an MSO-formula φ such that, for every countable uniformly k-sparse structure $\mathfrak{a}$,

- there exist unary predicates $\bar{P}$ such that $\mathfrak{a} \models \varphi(\bar{P})$ and
- $\mathrm{In}(\mathfrak{a}) = \mathcal{I}(n \times \langle \mathfrak{a}, \bar{P} \rangle)$, for all $\bar{P}$ with $\mathfrak{a} \models \varphi(\bar{P})$.

The last notion we present is based on the Gaifman graph of a structure.

Definition 4.14. Let $\mathfrak{a} = \langle A, R_0, \dots, R_{n-1} \rangle$ be a structure. The *Gaifman graph* of $\mathfrak{a}$ is the undirected graph

$$\mathrm{Gaif}(\mathfrak{a}) := \langle A, E \rangle$$

with edge relation

$$E := \{(a, b) \mid a \neq b \text{ and } (a, b) \text{ is guarded}\}.$$

The Gaifman graph gives an approximation of the relations in a structure. All the notions of this section can be defined in terms of the Gaifman graph as stated by the following proposition.

Proposition 4.15. A structure has finite tree width iff its Gaifman graph has finite tree width. A structure is uniformly sparse iff its Gaifman graph is uniformly sparse.

A *complete bipartite graph* is an undirected graph $\langle V, E\rangle$ where V is partitioned into two sets $A \uplus B$ such that

$$E = (A \times B) \cup (B \times A).$$

If $|A| = |B| = n$ then we say that the graph is of size n. If a graph has complete bipartite subgraphs of arbitrary size this implies that for those subgraphs the number of edges is quadratic in the number of vertices. As a consequence such a graph cannot be uniformly sparse. Hence, for every uniformly sparse graph, there is a bound on the size of its complete bipartite subgraphs. Over structures this means that for every uniformly sparse structure there exists a bound on the size of the complete bipartite subgraphs of its Gaifman graph. However the converse does not hold in general. It is possible to define non-uniformly sparse graphs which do not possess any complete bipartite subgraphs of size larger than some constant. For instance, the graph with vertices $\mathbb{Z}$ with an edge between m and n iff $|m - n|$ is a power of 2.

The three notions of (i) admitting a bound on the size of complete bipartite subgraphs; (ii) being uniformly sparse; and (iii) having bounded tree width; are related but do not coincide. The following theorem states the equivalence of these three notions over tree-interpretable structures. It was first proved for finite graphs in [26]. The generalisation to infinite structures proceeds along the same lines (see [5]).

Theorem 4.16. For every structure $\mathfrak{a}$, the following statements are equivalent:

1. $\mathfrak{a}$ has finite tree width.
2. $\mathfrak{a}$ is tree-interpretable and uniformly sparse.
3. $\mathfrak{a}$ is tree-interpretable and the size of the complete bipartite subgraphs of its Gaifman graph is bounded.

4.3 The weak power set of trees

We have seen that the power-set construction allows us to relate MSO and FO, in the same way MSO and GSO are related by the incidence structure construction. Hence, one may wonder whether results similar to Theorem 4.10 for GSO or Conjecture 4.3 for MSO hold in this setting. The answer is negative.

Proposition 4.17 (Colcombet-Löding, [21])**.** There are structures of decidable FO-theory which are not of the form $\mathcal{I}(\mathcal{P}_{\mathrm{w}}(\mathfrak{t}))$, for a tree $\mathfrak{t}$ and an FO-interpretation $\mathcal{I}$.

An example of this phenomenon is the random graph (a graph in which every finite graph can be embedded) which has a decidable FO-theory but is not of the above form. This propositon is established as an application of the following theorem which eliminates the weak power-set operation in the equation $(\mathcal{I} \circ \mathcal{P}_{\mathrm{w}})(\mathfrak{t}) = \mathcal{P}_{\mathrm{w}}(\mathfrak{a})$, provided that $\mathfrak{t}$ is a deterministic tree.

Theorem 4.18 (Colcombet-Löding, [21]). For every FO-interpretation $\mathcal{I}$, there exists a WMSO-interpretation $\mathcal{J}$ such that

$$(\mathcal{I} \circ \mathcal{P}_{\mathrm{w}})(\mathfrak{t}) \cong \mathcal{P}_{\mathrm{w}}(\mathfrak{a}) \quad \text{implies} \quad \mathcal{J}(\mathfrak{t}) \cong \mathfrak{a},$$

for every deterministic tree $\mathfrak{t}$ and every structure $\mathfrak{a}$.

Note that some kind of converse to this theorem can easily be deduced from Lemma 3.28. Indeed, for every WMSO-interpretation $\mathcal{J}$, there exists an FO-interpretation $\mathcal{I}$ such that

$$\mathcal{I} \circ \mathcal{P}_{\mathrm{w}} = \mathcal{P}_{\mathrm{w}} \circ \mathcal{J}.$$

Consequently, $\mathcal{J}(\mathfrak{t}) \cong \mathfrak{a}$ implies $(\mathcal{I} \circ \mathcal{P}_{\mathrm{w}})(\mathfrak{t}) = (\mathcal{P}_{\mathrm{w}} \circ \mathcal{J})(\mathfrak{t}) \cong \mathcal{P}_{\mathrm{w}}(\mathfrak{a})$.

Finally, let us state a variant of Theorem 4.5 for the weak power set of a tree.

Theorem 4.19 (Colcombet-Löding, [21]). For every FO-interpretation $\mathcal{I}$ and every binary relation symbol $\sim$, there is an FO-interpretation $\mathcal{J}$ such that:

$$(\mathcal{I} \circ \mathcal{P}_{\mathrm{w}})(\mathfrak{t})/{\sim^{(\mathcal{I} \circ \mathcal{P}_{\mathrm{w}})(\mathfrak{t})}} \cong (\mathcal{J} \circ \mathcal{P}_{\mathrm{w}})(\mathfrak{t}), \quad \text{for every deterministic tree } \mathfrak{t}.$$

When the power set operation is used instead of the weak power-set, we conjecture that this theorem becomes false, whereas Theorem 4.18 remains true: New phenomena arise when infinite sets are allowed.

5 Classes

Suppose that we are interested in, say, the monadic second-order theory of some structure $\mathfrak{a}$. One way to show the decidability of this theory is to start from a structure $\mathfrak{b}$ for which we already know that its monadic second-order theory is decidable, and then to construct $\mathfrak{a}$ from $\mathfrak{b}$ using MSO-compatible operations. We have seen an example of this approach in Corollary 3.22 where the infinite binary tree $\mathfrak{t}_2$ is constructed from a finite structure using an iteration and an MSO-interpretation.

In this last section we follow this idea and consider not only single structures but classes of structures that can be obtained in the way described above. For example, by applying the iteration operation to a finite structure followed by an MSO-interpretation we can not only construct $\mathfrak{t}_2$ but a whole class of structures with a decidable monadic second-order theory. This class and its generalisations are the subject of the first part of this section. In Section 5.2 we consider classes of structures with a decidable first-order theory that can be obtained with the help of FO-interpretations and the (weak) power-set operation. We conclude our survey in Section 5.3 by presenting HR-equational structures and their GSO-theory.

5.1 Prefix-recognisable structures and the Caucal hierarchy

We have conjectured above that all structures with a decidable MSO-theory are tree-interpretable. In this section we take the opposite direction and define large classes of tree-interpretable structures with a decidable MSO-theory. We start with the class of prefix-recognisable structures. Originally, this class was defined as a class of graphs in [17]. These graphs are defined over a universe consisting of a regular set of finite words and their edge relation is given as a finite union of relations of the form

$$(U \times V)W := \{(uw, vw) \mid u \in U,\ v \in V,\ w \in W\},$$

for regular languages U, V, W. Such relations are a combination of a recognisable relation $U \times V$ for regular U and V, followed by the identity relation, explaining the term 'prefix-recognisable'.

This definition can been extended to arbitrary structures instead of graphs (see [6, 14]) but the description of prefix-recognisable relations gets more complicated. Using the approach of compatible operations we obtain an alternative and simpler definition of the same class of structures.

Definition 5.1. A structure $\mathfrak{a}$ is *prefix-recognisable* if and only if $\mathfrak{a} \cong \mathcal{I}(\mathfrak{t}_2)$, for some MSO-interpretation $\mathcal{I}$.

This definition directly implies that each prefix-recognisable structure is tree-interpretable and has a decidable monadic second-order theory because $\mathfrak{t}_2$ has. Further elementary properties are summarised in the following proposition.

Proposition 5.2. The class of prefix-recognisable structures is closed under (i) MSO-interpretations, (ii) parameterless MSO-transductions, (iii) disjoint unions, (iv) finite products, (v) quotients, and (vi) generalised sums of the form $\sum_{i \in \mathfrak{i}} \mathfrak{a}$ in which both $\mathfrak{a}$ and $\mathfrak{i}$ are prefix-recognisable and all summands $\mathfrak{a}$ are isomorphic.

In fact, according to Theorem 4.2[1], we can even replace MSO-interpretations by FO-interpretations.

Theorem 5.3 (Colcombet, [20])**.** A structure $\mathfrak{a}$ is prefix-recognisable if and only if $\mathfrak{a} \cong \mathcal{I}(\mathfrak{t}_2)$, for some FO-interpretation $\mathcal{I}$.

[1] In combination with the fact that every regular tree is FO-interpretable in $\mathfrak{t}_2$.

For prefix-recognisable graphs several alternative characterisations have been given, for example they are the configuration graphs of pushdown automata after factoring out the ε-transitions, and also those graphs obtained as the least solutions of finite systems of equations whose operations consists of (i) disjoint unions and (ii) positive quantifier-free interpretations (this approach is due to Barthelmann [1], see [4] for an overview).

In the definition of prefix-recognisable structures we have used the infinite binary tree $\mathfrak{t}_2$ as a generator and applied MSO-interpretations to it. In Section 3 we have seen how $\mathfrak{t}_2$ can be obtained from a finite structure with the help of the iteration operation. In fact, we do not get more structures when we allow the application of an MSO-interpretation to the iteration of an arbitrary finite structure.

Proposition 5.4. The prefix-recognisable structures are exactly those of the form $\mathcal{I}(\mathfrak{a}^*)$ for an MSO-interpretation $\mathcal{I}$ and a finite structure $\mathfrak{a}$.

As both operations used in Proposition 5.4 are MSO-compatible there is no reason to stop after just one application of each of them. This idea is used in [18] for graphs using the unravelling operation instead of the iteration and an inverse rational mapping (a weakening of an MSO-interpretation) instead of an MSO-interpretation. According to [16] the following definition is equivalent to the original one.

Definition 5.5. The *Caucal hierarchy* $\mathcal{C}_0 \subset \mathcal{C}_1 \subset \dots$ is defined as follows. The first level $\mathcal{C}_0$ consists of all finite structures. Each higher level $\mathcal{C}_{n+1}$ consists of all structures of the form $\mathcal{I}(\mathfrak{a}^*)$ where $\mathcal{I}$ is an MSO-interpretation and $\mathfrak{a} \in \mathcal{C}_n$.

The compatibility of the employed operations directly yields the decidability of the MSO-theory for all structures in this class.

Theorem 5.6. All structures in the Caucal hierarchy have a decidable MSO-theory.

In the same spirit as Theorem 5.3 one can show that MSO-interpretations can be replaced by FO-interpretations. Furthermore, the iteration can also be replaced by the unravelling operation applied to the graphs on each level.

Theorem 5.7 (Colcombet, [20])**.** A structure belongs to $\mathcal{C}_{n+1}$ if and only if it is of the form $(\mathcal{I} \circ \mathcal{U})(\mathfrak{g})$ where $\mathcal{I}$ is an FO-interpretation, $\mathcal{U}$ the unravelling operation, and $\mathfrak{g} \in \mathcal{C}_n$ is a graph.

At present, the Caucal hierarchy is the largest known natural class of structures with a decidable MSO-theory (other structures with decidable MSO-theory can be constructed by ad hoc arguments; see, e.g., Proposition 5 of [16]). The first level of this hierarchy, i.e., the class of prefix-recognisable structures, is already well investigated. In [16] the graphs of level n are shown to be the same as the configuration graphs of higher-order pushdown automata of level n (automata using nested stacks of nesting depth n). Using this equivalence and a result on the languages accepted by higher-order pushdown automata, one obtains the strictness of the Caucal hierarchy. In [16] it is also shown that not all structures of decidable MSO-theory are captured: There is a tree with decidable MSO-theory that is not contained in any level of the hierarchy. It remains an open task to gain a better understanding of the structures in higher levels of the hierarchy.

5.2 Automatic structures and extensions

Let us turn to structures with a decidable FO-theory. A prominent class of such structures is the class of automatic (and tree-automatic) structures, a notion originally introduced by Hodgson [36].

A relation $R \subseteq (\Sigma^*)^r$ on words is *automatic* if there is a finite automaton accepting exactly the tuples $(w_0, \ldots, w_{r-1}) \in R$, where the automaton reads all the words in parallel with the shorter words padded with a blank symbol (for formal definitions see, e.g., [37, 3, 8]). A structure is called *automatic* (or has an automatic presentation) if it is isomorphic to a structure whose universe is a regular set of words and whose relations are automatic in the sense described above.

In the same way we can also use automata on finite (ranked) trees to recognise relations. The superposition of a tuple of trees is defined by aligning their roots and then, for each node aligning the sequence of successors from left to right, filling up missing positions with a blank symbol (again, a formal definition can be found in [3, 8]). Accordingly, a structure is called *tree-automatic* if it is isomorphic to a structure whose domain consists of a regular set of finite trees and whose relations are recognised by finite automata reading the superpositions of tuples of trees. An alternative definition for tree-automatic structures can be given via least solutions of system of equations [19] in the same spirit as [1] for prefix-recognisable structures. In addition to the operations for prefix-recognisable structures one allows the Cartesian product in the equations.

By inductively translating formulae to automata we can use the strong closure properties of finite automata to show that each FO-definable relation over an automatic structure is again automatic. As the emptiness problem for finite automata is decidable this yields a decision procedure for the model-checking of FO-formulae over (tree-)automatic structures.

We are interested in generating structures with a decidable FO-theory using FO-compatible operations. We focus here on the use of FO-interpretations. The first possibility is to start from structures with a decidable FO-theory and then apply FO-interpretations to it. Alternatively we can start from structures with a decidable MSO-theory and then apply the (weak) power-set operation followed by an FO-interpretation.

To obtain the class of automatic structures in this way let us first note that each automatic structure can be represented using a binary alphabet, say $[2] = \{0, 1\}$. A word over this alphabet can either be seen as the binary encoding of a number, or as a set of natural numbers, namely the set of all positions in the word that are labelled by 1.

When encoding [2]-words by natural numbers we need relations that allow us to extract single bits of a number to be able to simulate the behaviour of finite automata in first-order logic. This can be done using the addition operation $+$ and the relation $|_2$ defined as follows (see, e.g., [11, 3]):

$$k \mid_2 m \quad :\text{iff} \quad k \text{ is a power of 2 dividing } m.$$

Similarly, if [2]-words are viewed as sets of natural numbers we have to be able to access the elements of the set. This is possible in the weak power-set of the structure $\langle \omega, < \rangle$. By Corollary 3.27, FO over $\mathcal{P}_{\mathrm{w}}\langle \omega, < \rangle$ corresponds to WMSO over $\langle \omega, < \rangle$, which is known to have the same expressive power as finite automata (see, e.g., [48]).

These ideas lead to the following characterisations of automatic structures.

Proposition 5.8. Let $\mathfrak{a}$ be a structure. The following statements are equivalent:

1. $\mathfrak{a}$ is automatic.
2. $\mathfrak{a} \cong \mathcal{I}\langle \mathbb{N}, +, |_2 \rangle$, for some FO-interpretation $\mathcal{I}$.
3. $\mathfrak{a} \cong (\mathcal{I} \circ \mathcal{P}_{\mathrm{w}})\langle \omega, < \rangle$, for some FO-interpretation $\mathcal{I}$.

To obtain tree-automatic structures we first note that it is enough to consider unlabelled finite binary trees. Such a tree can be encoded in the infinite binary tree $\mathfrak{t}_2$ by the set of its nodes. It is not difficult to see that first-order logic over the weak power-set structure of $\mathfrak{t}_2$ has the same expressive power as finite automata over trees.

Proposition 5.9. A structure $\mathfrak{a}$ is tree-automatic if and only if $\mathfrak{a} = (\mathcal{I} \circ \mathcal{P}_{\mathrm{w}})(\mathfrak{t}_2)$, for some FO-interpretation $\mathcal{I}$.

This approach via compatible operations can easily be generalised by using other generators than $\langle \omega, < \rangle$ and $\mathfrak{t}_2$. In the previous section we have obtained a hierarchy of structures with a decidable MSO-theory. The infinite binary tree $\mathfrak{t}_2$ is on the first level of this hierarchy. Using Proposition 5.9 as a definition for tree-automatic structures, we obtain a natural hierarchy of higher-order tree-automatic structures.

Definition 5.10. A *higher-order tree-automatic structure of level n* is a structure of the form $(\mathcal{I} \circ \mathcal{P}_{\mathrm{w}})(\mathfrak{t})$ for some tree $\mathfrak{t}$ from $\mathcal{C}_n$, the nth level of the Caucal hierarchy.

Using Theorem 5.6 and the properties of the operations involved we obtain the following result.

Theorem 5.11. Every higher-order tree-automatic structure has a decidable first-order theory.

Although the Caucal hierarchy is known to be strict this does not directly imply that the hierarchy of higher-order tree-automatic structures is also strict. But by Theorem 4.18 it follows that, if the hierarchy would collapse then all the trees in the Caucal hierarchy could be generated from a single tree $\mathfrak{t}$ in this hierarchy by means of WMSO-interpretations. This would contradict the strictness of the Caucal hierarchy because, according to [16][2], each level is closed under WMSO-interpretations.

Theorem 5.12 (Colcombet-Löding, [21])**.** The hierarchy of higher-order tree-automatic structures is strict.

As mentioned in the previous section very little is known about structures on the higher levels of the Caucal hierarchy. As higher-order tree-automatic structures are defined by means of the Caucal hierarchy we even know less about these structures. In [21] it is illustrated how to apply Theorem 4.18 to show that structures are not higher-order tree-automatic.

5.3 HR-equational structures

In [22] equations using operations on structures are used to define infinite structures. The operations work on structures that are coloured by a finite set of colours. We introduce constant symbols for each finite structure (over a fixed signature). From these we build new structures using:

- the disjoint union operation $\uplus$;
- unary operations ρ_{ab} recolouring all elements of colour a to colour b;
- unary operations θ_a that merge all elements of colour a into a single element.

For example, the equation

$$x = \rho_{20}\big(\theta_2\big((\overset{1}{\bullet}\!\to\!\overset{2}{\bullet}) \uplus \rho_{12}(x)\big)\big)$$

[2] In [16] the closure of each level under MSO-interpretations is shown. But in the same paper it is shown that each level can be generated by MSO-interpretations from a deterministic tree of this level, and on deterministic trees the finiteness of a set can be expressed in MSO. Hence the levels are also closed under WMSO-interpretations.

has as least solution the graph

$$\begin{tikzcd}[column sep=small] \overset{1}{\bullet} \arrow[r] & \overset{0}{\bullet} \arrow[r] & \overset{0}{\bullet} \arrow[r] & \overset{0}{\bullet} \arrow[r] & \cdots \end{tikzcd}$$

The class of structures obtained as solutions of finite systems of equations over these operations has various names in the literature (equational, regular, hyperedge replacement). We use here the term HR-*equational.*

We obtain a connection between HR-equational structures and trees by unravelling the system of equations defining a given structure $\mathfrak{a}$ into an infinite tree $\mathfrak{t}$. The inner nodes of the tree are labelled with the operations and the leaves with the finite structures that are used as building blocks for the resulting infinite structure. As an unravelling of a finite system of equations the tree $\mathfrak{t}$ is regular and it contains all the information on how to build the structure $\mathfrak{a}$.

It should not be surprising that it is possible to construct the structure $\mathfrak{a}$ from $\mathfrak{t}$ via a parameterless MSO-transduction. But we can do even better because all the information on the relations of $\mathfrak{a}$ is contained in the leaves of the defining tree. This allows us to construct not only $\mathfrak{a}$ but also $\mathrm{In}(\mathfrak{a})$ by a parameterless MSO-transduction. It turns out that this property characterises HR-equational structures. As for prefix-recognisable structures we therefore choose this property as the definition.

Definition 5.13. A structure $\mathfrak{a}$ is HR-*equational* if and only if $\mathrm{In}(\mathfrak{a})$ is prefix-recognisable.

By Proposition 3.44 we can reduce the GSO-theory of an HR-equational structure to the MSO-theory of a prefix-recognisable one.

Proposition 5.14. Every HR-equational structure has a decidable GSO-theory.

Courcelle [23] has proved that the isomorphism problem for HR-equational structures is decidable. We can generalise this result as follows. In [6] it is shown that prefix-recognisable structures can be axiomatised in GSO, i.e., for each prefix recognisable structure $\mathfrak{a}$, one can construct a GSO-sentence $\psi_{\mathfrak{a}}$ such that

$$\mathfrak{b} \models \psi_{\mathfrak{a}} \quad \text{iff} \quad \mathfrak{b} \cong \mathfrak{a}, \quad \text{for every structure } \mathfrak{b}.$$

If we take $\mathfrak{b}$ from a class of structures for which we can decide whether $\mathfrak{b} \models \psi_{\mathfrak{a}}$ holds then this allows us to solve the isomorphism problem for $\mathfrak{a}$ and $\mathfrak{b}$. To this end let $\mathfrak{b}$ be a uniformly sparse structure from the Caucal hierarchy. (Note that every HR-equational structure is uniformly sparse.) According to Theorem 4.12 we can construct an MSO-sentence $\psi'_{\mathfrak{a}}$ that is equivalent to $\psi_{\mathfrak{a}}$ on $\mathfrak{b}$. And since the MSO-theory of each structure in the Caucal hierarchy is decidable we can now verify if $\mathfrak{b} \models \psi'_{\mathfrak{a}}$, which is the case if, and only if, $\mathfrak{a} \cong \mathfrak{b}$.

Theorem 5.15. Given an HR-equational structure $\mathfrak{a}$ and a uniformly sparse structure $\mathfrak{b}$ from the Caucal hierarchy, we can decide whether $\mathfrak{a} \cong \mathfrak{b}$.

The above description is slightly simplified. The GSO-sentence $\psi_{\mathfrak{a}}$ constructed in [6] uses cardinality quantifiers $\exists^{\kappa}$ meaning "there are at least κ many", for a cardinal κ. To make Theorem 5.15 work in this extended setting, we first note that Theorem 4.12 also works if the logics are extended with cardinality quantifiers. Second, we have to verify that $\mathfrak{b} \models \psi'_{\mathfrak{a}}$ can also be checked if $\psi'_{\mathfrak{a}}$ contains cardinality quantifiers. Because $\mathfrak{b}$ is countable, we only need to consider the quantifier "there are infinitely many". This quantifier can be eliminated since each structure of the Caucal hierarchy can be obtained by an MSO-interpretation from a deterministic tree of the same level and on such trees the property of a set being infinite can be expressed in MSO.

Bibliography

[1] K. Barthelmann. When can an equational simple graph be generated by hyperedge replacement? In *MFCS*, volume 1450 of *LNCS*, pages 543–552, 1998.

[2] D. Berwanger and A. Blumensath. The monadic theory of tree-like structures. In E. Grädel, W. Thomas, and T. Wilke, editors, *Automata, Logic, and Infinite Games*, LNCS 2500, pages 285–301. Springer, 2002.

[3] A. Blumensath. Automatic structures. Diploma thesis, RWTH Aachen, 1999.

[4] A. Blumensath. Prefix-recognisable graphs and monadic second-order logic. Technical Report AIB-06-2001, RWTH Aachen, May 2001.

[5] A. Blumensath. *Structures of Bounded Partition Width.* Ph. D. Thesis, RWTH Aachen, Aachen, 2003.

[6] A. Blumensath. Axiomatising tree-interpretable structures. *Theory of Computing Systems*, 37:3–27, 2004.

[7] A. Blumensath. A Model Theoretic Characterisation of Clique-Width. *Annals of Pure and Applied Logic*, 142:321–350, 2006.

[8] A. Blumensath and E. Grädel. Automatic structures. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science LICS 2000*, pages 51–62, 2000.

[9] A. Blumensath and S. Kreutzer. An extension to muchnik's theorem. *Journal of Logic and Computation*, 15:59–74, 2005.

[10] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.

[11] V. Bruyère, G. Hansel, C. Michaux, and R. Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc.*, 1:191–238, 1994.

[12] J. R. Büchi. Weak second order logic and finite automata. *Z. Math. Logik Grundlag. Math.*, 6:66–92, 1960.

[13] J. R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. 1960 Int. Congr. for Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

[14] A. Carayol and T. Colcombet. On equivalent representations of infinite structures. In *ICALP 2003*, volume 2719 of *LNCS*, pages 599–610. Springer, 2003.

[15] A. Carayol and C. Löding. MSO on the infinite binary tree: Choice and order. In *CSL'07*, volume 4646 of *LNCS*, pages 161–176. Springer, 2007.

[16] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS'03*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.

[17] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *ICALP'96*, volume 1099 of *LNCS*, pages 194–205. Springer, 1996.

[18] D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, volume 2420 of *LNCS*, pages 165–176. Springer, 2002.

[19] T. Colcombet. *Représentations et propriétés de structures infinies.* Phd thesis, Université de Rennes, Rennes, 2004.

[20] T. Colcombet. A combinatorial theorem for trees. In *ICALP 2007*, volume 4596 of *LNCS*, pages 901–912. Springer, 2007.

[21] T. Colcombet and C. Löding. Transforming structures by set interpretations. *Logical Methods in Computer Science*, 3(2), 2007.

[22] B. Courcelle. The monadic second order logic of graphs II: Infinite graphs of bounded width. *Mathematical System Theory*, 21:187–222, 1989.

[23] B. Courcelle. The monadic second-order logic of graphs IV: Definability properties of equational graphs. *Annals of Pure and Applied Logic*, 49:193–255, 1990.

[24] B. Courcelle. Monadic second-order definable graph transductions: A survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994.

[25] B. Courcelle. The monadic second-order logic of graphs VIII: Orientations. *Annals of Pure and Applied Logic*, 72:103–143, 1995.

[26] B. Courcelle. Structural properties of context-free sets of graphs generated by vertex replacement. *Inf. Comput.*, 116(2):275–293, 1995.

[27] B. Courcelle. The monadic second-order logic of graphs XIV: Uniformly sparse graphs and edge set quantifications. *Theoretical Computer Science*, 299(1-36), 2003.

[28] B. Courcelle. Clique-width of countable graphs: a compactness property. *Discrete Mathematics*, 276(1-3):127–148, 2004.

[29] B. Courcelle and J. Engelfriet. A Logical Characterization of the Sets of Hypergraphs Defined by Hyperedge Replacement Grammars. *Math. System Theory*, 28:515–552, 1995.

[30] B. Courcelle and S.-I. Oum. Vertex-minors, monadic second-order logic and a conjecture by Seese. *Journal of Combinatorial Theory, Series B*, 97:91–126, 2007.

[31] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92:35–62, 1998.

[32] S. Feferman and R. Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicæ*, 47:57–103, 1959.

[33] E. Grädel, C. Hirsch, and M. Otto. Back and forth between guarded and modal logics. *ACM Transactions on Computational Logics*, 3(3):418–463, 2002.

[34] Y. Gurevich. Monadic second-order theories. In J. Barwise and S. Feferman, editors, *Model-Theoretic Logics*, pages 479–506. Springer, 1985.

[35] Y. Gurevich and S. Shelah. Rabin's uniformization problem. *J. Symb. Log.*, 48(4):1105–1119, 1983.

[36] B. R. Hodgson. Décidabilité par automate fini. *Ann. Sci. Math. Québec*, 7(3):39–57, 1983.

[37] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *Workshop LCC '94*, volume 960 of *LNCS*, pages 367–392. Springer, 1995.

[38] D. Kuske and M. Lohrey. Monadic chain logic over iterations and applications to pushdown systems. In *LICS*, pages 91–100, 2006.

[39] J. A. Makowsky. Algorithmic aspects of the feferman-vaught theorem. *Annals of Pure and Applied Logic*, 126:159–213, 2004.

[40] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[41] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. soc.*, 141:1–35, 1969.

[42] N. Robertson and P. D. Seymour. Graph minors. v. excluding a planar graph. *Journal of Combinatorial Theory B*, 41:92–114, 1986.

[43] D. Seese. The structure of models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic*, 53:169–195, 1991.

[44] A. L. Semenov. Decidability of monadic theories. In *MFCS'84*, volume 176 of *LNCS*, pages 162–175, 1984.

[45] S. Shelah. The monadic theory of order. *Annals Math*, 102:379–419, 1975.

[46] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[47] W. Thomas. Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In *Structures in Logic and Computer Science, A Selection of Essays in Honor of A. Ehrenfeucht*, volume 1261 of *LNCS*, pages 118–143. Springer, 1997.

[48] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.

[49] I. Walukiewicz. Monadic second order logic on tree-like structures. In *STACS'96*, volume 1046 of *LNCS*, pages 401–413, 1996.

[50] I. Walukiewicz. Monadic second order logic on tree-like structures. *TCS*, 275(1–2):230–249, 2002.

Forest algebras*

Mikołaj Bojańczyk[1]
Igor Walukiewicz[2]

[1] Institute of Informatics
Uniwersytet Warszawski
Banacha 2
02-097 Warszawa, Poland
bojan@mimuw.edu.pl

[2] Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux 1
351, cours de la Libération
33405 Talence cedex, France
igw@labri.fr

Abstract

There are at least as many interesting classes of regular tree languages as there are of regular word languages. However, much less is known about the former ones. In particular, very few decidable characterizations of tree language classes are known. For words, most known characterizations are obtained using algebra. With this in mind, the present paper proposes an algebraic framework for classifying regular languages of finite unranked labeled trees.

If in a transformation semigroup we assume that the set being acted upon has a semigroup structure, then the transformation semigroup can be used to recognize languages of unranked trees. This observation allows us to examine the relationship connecting tree languages with standard algebraic concepts such as aperiodicity idempotency, or commutativity. The new algebraic setting is used to give several examples of decidable algebraic characterizations.

1 Introduction

There is a well-known decision problem in formal language theory:

> Decide if a given a regular language of finite binary trees can be defined by a formula of first-order logic with three relations: ancestor, left and right successor.

* We would like to thank Olivier Carton, Jean-Éric Pin, Thomas Schwentick, Luc Segoufin and Pascal Weil for their helpful comments. Special thanks are due to Howard Straubing, for correcting several errors in a previous version, and suggesting some improved definitions.

If the language is a word language (there is only one successor relation in this case) the problem is known to be decidable thanks to fundamental results of Schützenberger [14] and McNaughton and Papert [11]. The problem is also decidable for words when only the successor relation is available [18, 1]. However, no algorithm is known for the case of tree languages, see [9, 13, 3, 2] for some results in this direction.

There is a large body of work on problems of the type: decide if a given regular word language can be defined using such and such a logic [6, 12, 15, 19, 20, 22]. Most of the results have been obtained using algebraic techniques of semigroup theory. Recently, there has even been some progress for tree languages [21, 8, 5, 2]. There is, however, a feeling that we still do not have the right algebraic tools to deal with tree languages. In this paper we propose an algebraic framework, called forest algebras, and study the notion of recognizability in this framework. We want it to be as close to the word case as possible to benefit from the rich theory of semigroups. We show how standard notions, such as aperiodicity, idempotency, or commutativity, can be used in our framework to characterize classes of tree languages.

Forest algebras are defined for forests (ordered sequences) of unranked trees, where a node may have more than two (ordered) successors. This more general (more general than, say, binary trees) setting is justified by cleaner definitions, where semigroup theory can be used more easily.

We begin our discussion of forest algebras with the free forest algebra. Just as the free monoid is the set of words, the free forest algebra is going to be the set of forests. For finite words, there is one natural monoid structure: concatenation of words with the empty word as a neutral element. For forests there is also a concatenation operation that puts one forest after the other (see Figure 1). This operation though, has very limited power as the depth of the resulting forest is the maximum of the depths of the arguments. One needs also some kind of vertical composition that makes forests grow. This requires a notion of a context, which is a forest with a single hole in some leaf. Contexts can be composed by putting one of them in the hole of the other (see Figure 2). Moreover, by putting a forest in the hole of a context we obtain again a forest. Summarizing, for unranked, ordered, finite forests there are two natural monoids:

- *Horizontal monoid.* Forests with concatenation, and the empty tree as a neutral element.

- *Vertical monoid.* Contexts with context composition, and the context with a hole in the root as a neutral element.

The two monoids are linked by an action of contexts on forests: if p is a context and t is a forest then pt is a forest obtained by putting t in the hole of p in the same way as the contexts are composed.

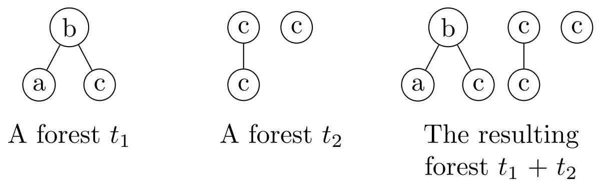


FIGURE 1. Forest concatenation

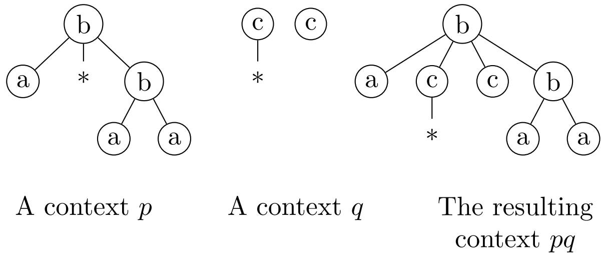


FIGURE 2. Context composition

In the case of words, a language of finite words induces a congruence, the Myhill-Nerode equivalence relation, which has finite index whenever the language is regular. The same concepts apply to forest algebras, except that we get two congruences: one for the vertical semigroup and one for the horizontal semigroup. A regular language of finite forests can be thus seen as one where both congruences are of finite index.

An important property of a forest algebra is that it is a special case of a transformation semigroup. Recall that a *transformation semigroup* is a semigroup along with an action over a set. In the forest algebra, the acting semigroup is the set of contexts, while that set acted upon is the set of forests (which itself is equipped with a semigroup structure).

There is a well-developed theory of transformation semigroups that is useful in classifying regular word languages. We hope that this theory might extend to the case of trees and this paper presents first steps in this direction. To illustrate how forest algebra can be used in classifying regular languages, we show how two language classes—forest languages determined by the labels occurring in a forest, and forest languages definable by a Σ_1 formula—can be described in terms of forest algebra. We also present a more involved example: languages definable in the temporal logic EF.

2 Preliminaries

The set of trees and forests over a finite alphabet A is defined as follows:

- an empty tree, denoted 0, is a tree (and therefore also a forest);
- if s, t are forests, then $s + t$ is a forest; moreover $+$ is associative;
- If s is a forest, then as is a tree (and also a forest) for every $a \in A$.

The empty tree is a neutral element for the operation $+$ of forest concatenation. This operation is in general non-commutative. A tree is a forest of the form as, where s is a forest. We denote trees, as well as forests, by s, t and u. Most of the time we shall be working with forests and we shall say explicitly when a variable denotes a tree.

It will be convenient to interpret a forest as a partial function $t : \mathbb{N}^+ \to A$ with a finite domain (the roots of this forest are the nodes from $\mathbb{N}$). Elements of this finite domain are called *nodes* of t. (The domain is closed under nonempty prefixes, and if $y < y'$ are natural numbers with $x \cdot y'$ in the domain, then also $x \cdot y$ belongs to the domain.) This function assigns to each node its *label.* If x, y are two nodes of t, we write $x \leq y$ ($x < y$) if x is a (proper) prefix of y (i.e x is closer to the root than y). If x is a maximal node satisfying $x < y$, then we call x the *parent* of y and we call y a *successor* of x. (Each node has at most one parent, but may have many successors.) Two nodes are *siblings* if they have the same parent. A *leaf* is a node without successors. The *subtree of* t *rooted in the node* x, denoted $t|_x$, assigns the label $t(x \cdot y)$ to a node $0 \cdot y$. The *successor forest* of a node is the forest of subtrees rooted in that node's successors.

An A-*context* is an $(A \cup \{*\})$-forest, where $*$ is a special symbol not in A. Moreover, $*$ occurs in exactly one leaf, which is called the *hole.* We use letters p, q to denote contexts. When p is a context and t is a forest, pt is the forest obtained from p by replacing the hole with t (see Figure 2). Similarly we define the composition of two contexts p, q – this is the context $p \cdot q$ that satisfies $(p \cdot q)t = p(qt)$ for every forest t. The neutral element of context composition is a context, denoted 1, consisting only of a single node labeled $*$.

3 Forest algebras

In this section we formally define a forest algebra. We give some examples and explore some basic properties.

A *forest algebra* $(H, V, \mathrm{act}, \mathrm{in}_\mathrm{L}, \mathrm{in}_\mathrm{R})$ consists of two monoids H, V, along with an action $\mathrm{act} : H \times V \to H$ of V on H and two operations $\mathrm{in}_\mathrm{L}, \mathrm{in}_\mathrm{R} : H \to V$. We denote the monoid operation in H by $+$ and the monoid operation in V by $\cdot$. The neutral elements of the two monoids will be denoted respectively: 0 and 1. Instead of writing $\mathrm{act}(h, v)$, we write vh (notice a reversal of arguments). A forest algebra must satisfy the following axioms:

action $(v \cdot w)h = v(wh)$;

insertion $\mathrm{in}_\mathrm{L}(g)h = g + h$ and $\mathrm{in}_\mathrm{R}(g)h = h + g$;

faithfulness for every two distinct $v, w \in V$ there is $h \in H$ with $vh \neq wh$;

We call V the *vertical monoid* and H the *horizontal monoid*. Thanks to the action axiom it is unambiguous to write vwh. Most of the time we shall omit the act, in_L, in_R from $(H, V, \mathrm{act}, \mathrm{in}_\mathrm{L}, \mathrm{in}_\mathrm{R})$ and write (H, V), just as we identify a monoid with its carrier set. We shall also sometimes write $h + 1$ instead of $\mathrm{in}_\mathrm{L}h$, and $1 + h$ instead of $\mathrm{in}_\mathrm{R}h$.

Example 3.1. Let H be any monoid. Let V be the set H^H of all transformations of H into H, with composition as the operation. To obtain a forest algebra from (H, V) it suffices to add the action and in_L, in_R. We can take the action of V on H to be just function application. The operations in_L and in_R are then determined by the insertion axiom. Faithfulness can be easily verified.

Note 3.2. As mentioned earlier, we have chosen to write the action on the left, while the standard practice in the algebraic study of languages of words is to write it on the right. That is, we write $\mathrm{act}(h, v)$ as vh, while in most papers on monoids and word languages one would see hv. We feel that this choice is justified by the difference in the way words and trees are denoted. In the case of words, writing the action on the right is justified by the way words are written (with the first letter on the left) as well as the way finite automata read the input (from left to right). For example, if one wants to calculate the action of a word abb on a state q of an automaton, one writes $qf_af_bf_b$; where f_a, f_b are the actions associated with the corresponding letters. Using standard functional notation this would give $f_b(f_b(f_a(q)))$. Hence, writing action on the right saves tiresome reversal of the word. For trees the situation is different. Usually, one describes trees with terms. So $a(t_1 + t_2)$ denotes a tree with the root a and two subtrees t_1 and t_2. If we were writing actions on the right, the value of this tree would be denoted by $(h_1 + h_2)v_a$, where h_i is the value of t_i and v_a is the value of a. In consequence, writing the action to the right corresponds to writing terms in reverse Polish notation. Writing the action on the right would thus force us either: to do the conversion into reverse Polish notation each time we go from trees to algebra, or to write trees in reverse Polish notation. The authors think that both options are more troublesome than the choice of the writing action on the left.

Note 3.3. Despite additive notation for monoid $(H, +)$, we do not require $+$ to be commutative. Having H commutative would be equivalent to saying that the order of siblings in a tree is not relevant. Although in all the examples given in this paper + will be commutative, one can easily find examples when it will not be the case. A prominent one is first-order logic with order on siblings.

Note 3.4. The axioms of forest algebra imply the existence of strong links between horizontal and vertical monoids. The first observation is that every element of h of H is of the form $v0$ for some $v \in V$. Indeed, it is enough to take $\mathrm{in}_\mathrm{L} h$ for v. Moreover, the mappings $\mathrm{in}_\mathrm{L}, \mathrm{in}_\mathrm{R} : H \to V$ are monoid morphisms as $\mathrm{in}_\mathrm{L}(h_1 + h_2) = \mathrm{in}_\mathrm{L}(h_1)\mathrm{in}_\mathrm{L}(h_2)$ and $\mathrm{in}_\mathrm{L}(0) = 1$.

A *morphism* between two forest algebras (H, V) and (G, W) is a pair of monoid morphisms $(\alpha : H \to G, \beta : V \to W)$ with additional requirements ensuring that the operations are preserved:

$$\alpha(vh) = \beta(v)\alpha(h)$$
$$\beta(\mathrm{in}_\mathrm{L}(h)) = \mathrm{in}_\mathrm{L}(\alpha(h)) \quad \text{and} \quad \beta(\mathrm{in}_\mathrm{R}(h)) = \mathrm{in}_\mathrm{R}(\alpha(h))$$

Note 3.5. The morphism α is determined by β via

$$\alpha(h) = \alpha(h + 0) = \alpha(\mathrm{in}_\mathrm{L}(h)0) = \beta(\mathrm{in}_\mathrm{L}(h))\alpha(0) \ ,$$

where $\alpha(0)$ must be the neutral element in G by the assumption on α being a monoid morphism. So it is enough to give a morphism β and verify if together with the uniquely determined α they preserve the operations.

Given an alphabet A, we define the *free forest algebra* over A, which is denoted by A^Δ, as follows:

- The horizontal monoid is the set of forests over A.
- The vertical monoid is the set of contexts over A.
- The action is the substitution of forests in contexts.
- The in_L function takes a forest and transforms it into a context with a hole to the right of all the roots in the forest. Similarly for in_R but the hole is to the left of the roots.

Observe that in_L and in_R are uniquely determined by insertion axioms, once the action is defined. The following lemma shows that free forest algebra is free in the sense of universal algebra.

Lemma 3.6. The free forest algebra A^Δ is a forest algebra. Moreover, for every forest algebra (H, V), every function $f : A \to V$ can be uniquely extended to a morphism $(\alpha, \beta) : A^\Delta \to (H, V)$ such that $\beta(a(*)) = f(a)$ for every $a \in A$.

Proof. That A^{Δ} is a forest algebra can be easily verified. We define a homomorphism by induction on the size of a tree/context:

$$\begin{aligned} \alpha(0) &= 0 & \beta(*) &= 1 \\ \alpha(at) &= f(a)(\alpha(t)) & \beta(a(p)) &= f(a)\beta(p) \\ \alpha(t_1+t_2) &= \alpha(t_1)+\alpha(t_2) & \beta(t_1+p+t_2) &= \mathrm{in}_{\mathrm{L}}(\alpha(t_1))\mathrm{in}_{\mathrm{R}}(\alpha(t_2))\beta(p) \end{aligned}$$

Directly from the definition it follows that α, β is a unique possible extension of f to a homomorphism. It can be checked that the two mappings are well defined. It is clear that α preserves $+$ operation. One shows that $\beta(pq) = \beta(p)\beta(q)$ by induction on the size of p. The preservation of the action property: $\alpha(pt) = \beta(p)\alpha(t)$ is also proved by induction on p. Finally, $\beta(\mathrm{in}_{\mathrm{L}}(t)) = \beta(t + *) = \alpha(t_1) + \beta(1) = \mathrm{in}_{\mathrm{L}}(\alpha(t))\beta(1) = \mathrm{in}_{\mathrm{L}}(\alpha(t))$. Q.E.D.

We now proceed to define languages recognized by forest algebras.

Definition 3.7. A set L of A-forests is said to be *recognized* by a surjective morphism $(\alpha, \beta) : A^{\Delta} \to (H, V)$ if L is the inverse image $\alpha^{-1}(G)$ of some $G \subseteq H$. The morphism (α, β) is said to recognize L, the set G is called the *accepting set*, and L is said to be recognized by (H, V).

Generally, we are interested in the case when (H, V) is finite; in this case we say that L is *recognizable*.

Example 3.8. Consider the set L of forests with an even number of nodes. We present here a finite forest algebra (H, V) recognizing L. Both H and V are $\{0, 1\}$ with addition modulo 2. The action is also addition; this defines the insertion functions uniquely. The recognizing morphism maps a context onto 0 if it has an even number of nodes. The accepting set is $\{0\}$.

Example 3.9. A language L of A-forests is called *label-testable* if the membership $t \in L$ depends only on the sets of labels that occur in t. The appropriate forest algebra is defined as follows. Both H and V are the same monoid: the set $P(A)$ with union as the operation. This determines the action, which must also be also union. We can take as a recognizing morphism a function that maps a context to the set of its labels.

Note 3.10. Another way to look at a forest algebra is from the point of view of universal algebra. In this setting, a forest algebra is a two-sorted algebra (with the sorts being H and V) along with two constants (neutral elements for H and V) and five operations: (i) monoid operations in H and V, (ii) the action vh of V on H and (iii) the two insertion operations in_{L} and

in_{R}. Forest algebras cannot be defined equationally due to the faithfulness requirement.

The universal algebra viewpoint gives us definitions of such concepts as subalgebra, cartesian product, quotient, morphism. The requirement of faithfulness is not preserved by homomorphic images and quotients. This implies that every time we take a quotient we need to check if the result is a faithful algebra.

3.1 Syntactic algebra for forest languages

Our aim now is to establish the concept of a syntactic forest algebra of a forest language. This is going to be a forest algebra that recognizes the language, and one that is optimal among those that do.

Definition 3.11. We associate with a forest language L two equivalence relations on the free forest algebra A^{Δ}:

- Two A-forests s, t are *L-equivalent* if for every context p, either both or none of the forests ps, pt belong to L.
- Two A-contexts p, q are *L-equivalent* if for every forest t, the forests pt and qt are L-equivalent.

Lemma 3.12. Both L-equivalence relations are congruences with respect to the operations of the forest algebra A^{Δ}.

Proof. We first show that L-equivalence for forests is a congruence with respect to concatenation of forests. We shall consider only concatenation to the right. We show that if s and s' are L-equivalent, then so are the forests $s+t$ and $s'+t$, for every forest t. Unraveling the definition of L-equivalence, we must show that for every context p we have: $p(s+t) \in L$ iff $p(s'+t) \in L$. Taking $q = p \cdot \mathrm{in}_{\mathrm{R}}(t)$ we get $qs = p(\mathrm{in}_{\mathrm{R}}(t)s) = p(s+t)$. In consequence:

$$p(s+t) \in L \quad \text{iff} \quad q(s) \in L \quad \text{iff} \quad q(s') \in L \quad \text{iff} \quad p(s'+t) \in L \ ,$$

where the middle equivalence follows from L-equivalence of s and s'. The proof for the concatenation to the left is analogous.

We now proceed to show that L-equivalence for contexts is a congruence with respect to context composition. We need to show that if two contexts p and p' are L-equivalent, then so are the contexts pq and $p'q$ for any context q (and similarly for the concatenation to the left). We need to show that for every forest t and every context q',

$$q'pqt \in L \quad \text{iff} \quad q'p'qt \in L \ .$$

The above equivalence follows immediately from the L-equivalence of p and p': it suffices to consider qt as a tree that is plugged into the contexts p and p'.

In a similar way, one shows that L-equivalence is a congruence with respect to the action pt and the insertions $\mathrm{in}_{\mathrm{L}}(t)$, $\mathrm{in}_{\mathrm{R}}(t)$. Q.E.D.

Definition 3.13. The *syntactic forest algebra* for L is the quotient of A^Δ with respect to L-equivalence, where the horizontal semigroup H^L consists of equivalence classes of forests over A, while the vertical semigroup V^L consists of equivalence classes of contexts over A. The *syntactic morphism* (α^L, β^L) assigns to every element of A^Δ its equivalence class in (H^L, V^L).

The above lemma guarantees that the quotient is well defined. In this quotient, faithfulness holds thanks to the definition of L-equivalence over contexts. The action and insertion axioms are also satisfied (as it is a quotient of a forest algebra). Hence, it is a forest algebra. We claim that this forest algebra satisfies the properties required from the syntactic forest algebra of L.

Proposition 3.14. A language L of A-forests is recognized by a the syntactic morphism (α^L, β^L). Moreover, any morphism $(\alpha, \beta) : A^\Delta \to (H, V)$ that recognizes L can be extended by a morphism $(\alpha', \beta') : (H, V) \to (H^L, V^L)$ so that $\beta' \circ \beta = \beta^L$.

Proof. The first part follows immediately by taking as an accepting set the set of L-equivalence classes of all the elements of L. The second statement follows from the observation that if two A-forests or contexts have the same image under (α, β) then they are L-equivalent. Q.E.D.

Note that in general the syntactic forest algebra may be infinite. However, Proposition 3.14 shows that if a forest language is recognized by some finite forest algebra, then its syntactic forest algebra must also be finite. In this case the syntactic forest algebra can be also easily computed. The procedure is the same as for syntactic monoids. Given a finite forest algebra (H, V) and a subset $G \subseteq H$ one marks iteratively all the pairs of elements that are not equivalent with respect to G. First, one marks all pairs consisting of an element of G and of an element of $H \setminus G$. Then one marks a pair $(h_1, h_2) \in H \times H$ if there is a $v \in V$ such that (vh_1, vh_2) is already marked. One marks also a pair of vertical elements (v_1, v_2) if there is a horizontal element h with (v_1h, v_2h) already marked. This process continues until no new pairs can be marked. The syntactic forest algebra is the quotient of the given algebra by the relation consisting of all the pairs that are not marked. In Section 3.3 we shall show that recognizability is equivalent with being accepted by the standard form of automata. In particular the proof of Proposition 3.19 gives a way of constructing a forest algebra from automaton. Together with the above discussion this gives a method of constructing a syntactic forest algebra for the language accepted by a given tree automaton.

3.2 Forest algebras and tree languages

Forest algebras give a natural definition of recognizable forest languages (Definition 3.7). However, tree languages are studied more often than forest languages. In this section we describe how a forest algebra can be used to recognize a language of unranked trees.

Definition 3.15. Given a tree language L over A and a letter $a \in A$, the *a-quotient*, denoted $a^{-1}L$, is the set of forests t that satisfy $at \in L$. A language L of A-trees is *tree-recognized* by a morphism $(\alpha, \beta) : A^{\Delta} \to (H, V)$ if $a^{-1}L$ is recognized by (α, β) for all $a \in A$.

Note that the above definition does not say anything about trees with only one (root-leaf) node; but these are finitely many and irrelevant most of the time. In particular, regular languages are closed under adding or removing a finite number of trees.

Example 3.16. A tree language of the form: "the root label is $a \in A$" is tree-recognized by any forest algebra. This because all the quotients $b^{-1}L$ for $b \in A$ are either empty (when $b \neq a$) or contain all forests (when $b = a$).

The above definition of recognizability induces a definition of syntactic forest algebra for a tree language L. Consider the intersection of all $(a^{-1}L)$-equivalences for $a \in A$. This is a congruence on A^{Δ} as it is an intersection of congruences. It is easy to check that the result is a faithful algebra.

Note 3.17. There is an alternative definition of tree-recognizability. In the alternative definition, we say that a tree language L is tree-recognized by a forest algebra (H, V) if there is a forest language K recognized by (H, V) such that L is the intersection of K with the set of trees. Under this alternative definition, there is no correct notion of syntactic algebra. For instance, the tree language "trees whose root label is a" can be tree-recognized by two forest algebras that have no common quotient tree-recognizing this language. Indeed, these may be forest algebras for two different forest languages that agree on trees.

Note 3.18. Yet another alternative definition of tree-recognizability says that L is tree-recognized iff it is recognized. In this case, the forest algebra must keep track of what is a single tree, and what is a forest. As a result, it becomes impossible to characterize some languages by properties of their syntactic algebras. For instance, consider the tree language "all trees". The horizontal monoid of this language has three elements: $H = \{0, 1, 2+\}$ which keep track of the number of trees in the forest. The vertical monoid has the transformations

$$V = \{h \mapsto h, h \mapsto h + 1, h \mapsto h + 2, h \mapsto 1, h \mapsto 2+\} .$$

The first three are contexts with the hole in the root, and the last two have the hole in a non root node. It is already inconvenient that the simplest possible tree language needs a non-trivial algebra. Furthermore, this same algebra recognizes the language "trees over $\{a, b\}$ where a does not appear in the root". The recognizing morphism maps the label a to $h \mapsto h + 2$ and the label b to $h \mapsto 1$. One can suggest several logics that can describe the first language but not the second, for all these logics characterizations in terms of syntactic algebras will be impossible.

3.3 Automata over forests

We would like to show that our definition of recognizability is equivalent with the standard notion of regular languages, i.e., languages accepted by automata. There are numerous presentations of automata on finite unranked trees and forests; here we shall use one that matches our algebraic definitions.

A *forest automaton* over an alphabet A is a tuple

$$\mathcal{A} = \langle (Q, 0, +),\ A,\ \delta : (A \times Q \to Q),\ F \subseteq Q \rangle$$

where $(Q, 0, +)$ is a finite monoid; intuitively a set of states with an operation of composition on states.

The automaton assigns to every forest t a value $t^{\mathcal{A}} \in Q$, which is defined by induction as follows:

- if t is an empty forest, then $t^{\mathcal{A}}$ is 0;
- if $t = as$, then $t^{\mathcal{A}}$ is defined to be $\delta(a, s^{\mathcal{A}})$, in particular if t is a leaf then $t^{\mathcal{A}} = \delta(a, 0)$;
- if $t = t_1 + \cdots + t_n$ then $t^{\mathcal{A}}$ is defined to be $t_1^{\mathcal{A}} + \cdots + t_n^{\mathcal{A}}$; observe that the $+$ operation in the last expression is done in $(Q, 0, +)$.

A forest t is *accepted by* $\mathcal{A}$ if $t^{\mathcal{A}} \in F$.

Proposition 3.19. A forest language is recognized by a finite forest algebra if and only if it is the language of forests accepted by some forest automaton.

Proof. Take a tree language L recognized by a morphism $(\alpha, \beta) : A^{\Delta} \to (H, V)$. That is $L = \alpha^{-1}(F)$ for some $F \subseteq H$. For the "only if" part, we need to show how it can be recognized by an automaton. Let $\mathcal{A} = \langle H, A, \delta : A \times H \to H, F \rangle$ where H is the horizontal monoid, $F \subseteq H$ is as above, and δ is defined by

$$\delta(a, h) = \beta(a)h \qquad \text{for } a \in A\ .$$

By induction on the size of the forest one can show that $t^{\mathcal{A}} = \alpha(t)$. Thus $\mathcal{A}$ recognizes the language of forests L.

For the other direction, suppose that we are given an automaton $\mathcal{A} = \langle (Q, 0, +), A, \delta, F \rangle$. We consider a forest algebra (H, V) where H is $(Q, 0, +)$ and V is the function space $H \to H$ with function composition as the operation and the identity as the neutral element. The action is function application and the insertions are uniquely determined. It is easy to see that (H, V) is a forest algebra. Consider now the unique homomorphism $(\alpha, \beta) : A^{\Delta} \to (H, V)$ with

$$\beta(a) = \delta(a) \qquad \text{for } a \in A \ ;$$

observe that each $\delta(a)$ is a function from H to H. This homomorphism might not be surjective as required by Definition 3.7, in this case we only keep the part of the algebra used by the homomorphism. By induction on the height of the forest one can show that $t^{\mathcal{A}} = \alpha(t)$. Q.E.D.

Actually, the above notion of automaton can be refined to a notion of (H, V) automaton for any forest algebra (H, V). Such an automaton has the form:

$$\mathcal{A} = \langle H,\ A,\ \delta : A \to V,\ F \subseteq H \rangle$$

thus the only change is that now states are from H and $\delta(b)$ is an element of from V while before it was a function from $Q \to Q$. We can do this because using the action act of the forest algebra, each $v \in V$ defines a function $\mathrm{act}(v) : H \to H$.

By the same reasoning as before, every language accepted by a (H, V) automaton is recognized by the algebra (H, V). Conversely, every language recognized by (H, V) is accepted by some (H, V) automaton. This equivalence shows essential differences between algebras and automata. Algebras do not depend on alphabets, while alphabets are explicitly declared in the description of an automaton. More importantly, the structure of the vertical semigroup is not visible in an automaton: in an automaton we see only generators of the vertical semigroup.

It may be worth to compare the above automata model with unranked tree automata (UTA's) [17, 10]. The only difference of any importance between these models is that UTA's have transition function of the form $\delta : \Sigma \times Q \to \mathrm{Reg}(Q)$, i.e., to each pair of state and letter, a UTA assigns a regular language over the alphabet Q. A tree whose root is labeled a can be assigned a state q if the sequence of states assigned to its children is in the regular language $\delta(q, a)$. In our case regular languages are represented by monoids. More precisely, we use one monoid structure on states to simultaneously recognize all regular word languages that appear in transitions. The two automata models have the same expressive power, and effective translations can be easily presented. Note that since we use monoids, there may be an exponential growth when translating from a UTA to a forest automaton.

3.4 Other possible variants of forest algebra

For words, one can use either monoids or semigroups to recognize word languages. In the first case, the appropriate languages are of the form $L \subseteq A^*$, while the second case disallows the empty word, and only languages $L \subseteq A^+$ are considered.

For forests, the number of choices is much greater. Not only do we have two sorts (forests and contexts) instead of just one (words), but these sorts are also more complex. This requires at least two choices:

- Is the empty forest a forest? Here, we say yes.
- Is the empty context a context? Here, we say yes.

We can also put some other restrictions on a position of the hole in the context, for example that it cannot have siblings, or that it cannot be in the root. Each combination of answers to the above questions gives rise to an appropriate definition of a forest algebra, as long as the correct axioms are formulated.

We do not lay any claim to the superiority of our choices. The others are just as viable, but this does not mean that they are all equivalent. The difference becomes visible when one tries to characterize algebras by equations. For example, the equation $vh = vg$ in our setting implies $h = g$ because this equation should be valid for all assignments of elements to variables, and in particular we can assign the identity context to v. But then, $h = g$ says that the horizontal monoid is trivial. If we did not allow contexts with the hole in a root, this equation would describe forest languages where membership of a forest depends only on the labels of its roots.

One may also ask what would happen if we had dropped the vertical structure. We could work with pairs of the form (H, Z) where Z is just a set and not a semigroup, but still we could have an action of Z on H. Such pairs correspond to automata where the alphabet is not fixed. For such objects we do not need to require insertion axioms as these axioms talk about the structure of the vertical semigroup which is not present here. All the theory could be developed in this setting but once again equations would have different meaning in this setting. In particular we would not have any way to refer explicitly to vertical composition. We refrain from doing this because we think that the structure of the vertical semigroup is important.

4 Simple applications

In this section we present two straightforward characterizations of forest languages. Both are effective, meaning that the conditions on the forest algebra can be effectively tested. The first characterization—of label testable languages—illustrates how a property of the context monoid can have important implications for the forest monoid. The second characterization—of languages definable by a Σ_1 formula—shows that we can also consider language classes that are not closed under boolean operations.

In the following we shall very often express properties of algebras by equations. An equation is a pair of terms in the signature of forest algebras over two types of variables: horizontal variables (h, g, …), and vertical variables (v, w,…). These terms should be well typed in an obvious sense and should have the same type: both should be either of the forest type, or of the context type. An algebra *satisfies* an equation if for any valuation assigning elements of the horizontal monoid to horizontal variables, and elements of the vertical monoid to vertical variables, the two terms have the same value. In this way an equation expresses a propery of algebras.

We say a forest language is *label testable* if the membership in the language depends only on the set of labels that appear in the forest.

Theorem 4.1. A language is label testable if and only if its syntactic algebra satisfies the equations:

$$vv = v \qquad vw = wv \; .$$

Proof. The only if part is fairly obvious, we only concentrate on the if part. Let then L be a language recognized by a morphism $(\alpha, \beta) : A^{\Delta} \to (H, V)$, with the target forest algebra satisfying the equations in the statement of the theorem. We will show that for every forest t the value $\alpha(t)$ depends only on the labels appearing in t.

We start by showing that the two equations from the statement of the theorem imply another three. The first is the idempotency of the horizontal monoid:

$$h + h = h \; .$$

This equation must hold in any forest algebra satisfying our assumption because of the following reasoning which uses the idempotency of the vertical monoid:

$$h + h = (h + 1)(h + 1)0 = (h + 1)0 = h \; .$$

(In the above, $h + 1$ denotes the context $\mathrm{in_L}(h)$.) The second is the commutativity of the horizontal monoid:

$$h + g = g + h \; .$$

The argument uses commutativity of the vertical monoid:

$$h + g = (h + 1)(g + 1)0 = (g + 1)(h + 1)0 = g + h \; .$$

Finally, we have an equation that allows us to flatten the trees:

$$v(h) = h + v0 \ .$$

The proof uses once again the commutativity of the vertical monoid:

$$v(h) = v(h+1)0 = (h+1)v0 = h + v0 \ .$$

The normal form of a forest will be a forest $a_1 0 + \cdots + a_n 0$, where each tree contains only one node, labeled a_i. Furthermore, the labels $a_1, \ldots, a_n$ are exactly the labels used in t, sorted without repetition under some arbitrary order on the set A. Using the three equations above one can show that every forest has the same value under α as its normal form. Starting from the normal form one can first use idempotency to "produce" as many copies of each label as the number of its appearances in the tree. Then using the last equation and the commutativity one can reconstruct the tree starting from leaves and proceeding to the root. Q.E.D.

Note 4.2. If we omit the equation $vv = v$, we get languages that can be defined by a boolean combination of clauses of the forms: "label a occurs at least k times", or "the number of occurrences of label a is $k \mod n$".

We now present the second characterization. A Σ_1 *formula* is a formula of first-order logic, where only existential quantifiers appear in the quantifier prenex normal form. The logic we have in mind uses the signature allowing label tests (a node x has label a) and the descendant order (a node x is a descendant of a node y). The following result shows which forest languages can be defined in Σ_1:

Theorem 4.3. Let L be a forest language, and let (α, β) be its syntactic morphism. A language L is definable in Σ_1 if and only if $vh \in \alpha(L)$ implies $vwh \in \alpha(L)$, for every v, w, h.

Proof. The only if implication is an immediate consequence of the fact that languages defined in Σ_1 are closed under adding nodes. We will now show the if implication. Below, we shall say that a forest s is a piece of a forest t if s can be obtained from t by removing nodes (i.e. the transitive closure of the relation which reduces a forest pqs to a forest ps).

Let L be a language recognized by a morphism $(\alpha, \beta) : A^\Delta \to (H, V)$, with α satisfying the property in the statement of the theorem. For each $h \in H$, let T_h be the set of forests that are assigned h by α, but have no proper piece with this property. Using a pumping argument, one can show that each set T_h is finite. We claim that a forest belongs to L if and only if it contains a piece $t \in T_h$, with $h \in \alpha(L)$. The theorem follows from this claim, since the latter property can be expressed in Σ_1.

The only if part of the claim is obvious: if a forest t belongs to L, then by definition it contains a piece from $T_{\alpha(t)}$, since $\alpha(t)$ belongs to $\alpha(L)$. For the if part of the claim, we need to use the property of α stated in the theorem: if t contains a piece s with $\alpha(s) \in \alpha(L)$, then by iterative application of the implication $vh \in \alpha(L) \Rightarrow vwh \in \alpha(L)$, we can show that $\alpha(t)$ also belongs to $\alpha(L)$, and hence t belongs to L. Q.E.D.

5 Characterization of EF

In this section we show how forest algebras can be used to give a decidable characterization of a known temporal logic for trees. The logic in question, called EF, is a fragment of CTL where EF is the only temporal operator allowed. Decidability of this fragment for the case of binary trees is known [5], and several alternative proofs have already appeared [23, 7]. Here, we should like to show how our setting—which talks about forests—can be used to show decidability of a logic over trees.

5.1 The logic EF

EF is a temporal logic that expresses properties of trees. The name EF is due to the unique temporal operator in the logic, EF, which stands for Exists (some path) Further down (on this path). Formulas of EF are defined as follows:

- If a is a letter, then a is a formula true in trees whose root label is a.
- EF formulas are closed under boolean connectives.
- If φ is an EF formula, then $\mathsf{EF}\varphi$ is an EF formula true in trees having a *proper* subtree satisfying φ.

We write $t \vDash \varphi$ to denote that a formula φ is true in a tree t. Restricting to proper subtrees in the definition of EF gives us more power, since the non-proper operator can be defined as $\varphi \vee \mathsf{EF}\varphi$.

We need to deal with a mismatch due to the fact that EF is defined over trees and our algebraic setting works with forests. For this, we need to define how forest languages can be defined in EF.

Definition 5.1. A *tree language L is definable in* EF iff there is an EF formula α with $L = \{t : t \vDash \alpha\}$. A *forest language L is definable in* EF if for some $a \in A$ the tree language $\{at : t \in L\}$ is definable in EF.

Notice that the choice of a in the above definition does not matter. The following observation shows that we can use forest definability to decide tree definability.

Lemma 5.2. A tree language L is EF definable iff for every $a \in A$ the forest language $a^{-1}L$ is EF definable (as a language of forests).

Proof. Suppose L is a tree language defined by a formula φ. This formula is boolean combination for formulas starting with EF and formulas of the form b for some $b \in A$. It is easy to see that φ can rewritten as a conjunction of implications $\bigwedge_{b\in A} b \Rightarrow \varphi_b$, where φ_b, for all $b \in A$, is a boolean combination of formulas starting with EF. Then φ_a defines the forest language $a^{-1}L$.

For the other direction suppose that for each $a \in A$ the forest language $a^{-1}L$ is EF-definable. So there is a formula φ_a and a letter $b \in A$ such that $bt \vDash \varphi_a$ iff $t \in a^{-1}L$. We can, if necessary, modify φ_a into φ'_a with the property that $bt \vDash \varphi_a$ if and only if $at \vDash \varphi'_a$. The tree language L is then defined by $\bigwedge_{a\in A} a \Rightarrow \varphi'_a$. Q.E.D.

As the main result of this section, we present two equations and show that a forest language is definable by an EF formula if and only if its syntactic forest algebra satisfies these equations. In particular, it is decidable if a regular tree language can be defined in EF.

Theorem 5.3. A forest language is definable in EF if and only if its syntactic forest algebra satisfies the following equations, called the EF equations:

$$g + h = h + g \tag{1.1}$$

$$vh = h + vh\ . \tag{1.2}$$

Equation (1.1) states that the horizontal monoid is commutative. In other words, membership of a forest in the language does not depend on order of siblings. Equation (1.2) is specific to EF and talks about interaction between two monoids. This equation also shows an advantage of our setting: the equation can be that simple because we need not to worry about the degree of vertices, and we can compare not only trees but also forests. The proof of the theorem is split across the following two subsections.

Note 5.4. One can also consider the logic EF^*, where the EF modality is replaced by its non-strict version EF^*. A formula $\mathsf{EF}^*\varphi$ is equivalent to $\varphi \vee \mathsf{EF}\varphi$. As mentioned before, this logic is strictly weaker than EF. For example, one cannot express in EF^* that a tree consists only of one leaf. Recently, a decidable characterization of EF^* was given in [23, 7]. The logic $\mathsf{EF}^*\varphi$ is not as well-behaved in our algebraic setting as EF. The problem is that one cannot tell if a forest language is definable in EF^* just by looking at its syntactic forest algebra. For an example, consider the language defined by the formula $\mathsf{EF}^*(b \wedge \mathsf{EF}^*c)$, over the alphabet $\{a, b, c\}$. The syntactic forest algebra for this language can also recognize the language of flat forests (where every tree consists only of the root). But the latter language is not EF^* definable.

5.2 Correctness

We show that the syntactic algebra of a forest language definable in EF must satisfy the EF equations. The basic idea is to prove that any language definable in EF is recognized by a forest algebra satisfying the EF equations. We shall then be able to conclude that the syntactic algebra must also satisfy these equations, as it is a morphic image of any algebra recognizing the language.

Assume then that a forest language L over an alphabet A is defined by a formula φ. The EF-*closure* of φ, denoted $\mathrm{cl}_{\mathsf{EF}}(\varphi)$, is the set of all subformulas of φ of the form $\mathsf{EF}\psi$ for some ψ.

Given a forest t and $a \in A$ we define a *forest type* of t (with respect to our fixed φ):

$$FT_\varphi(t) = \{\psi \in \mathrm{cl}_{\mathsf{EF}}(\varphi) : at \vDash \psi\} \ .$$

It is clear that this definition does not depend on the choice of a, so we do not include it in the notation.

We now define an equivalence relation on forests by saying that two forest are φ-equivalent if their FT_φ values are the same. We denote this relation by $\sim_\varphi$. The relation can be extended to contexts by saying that two contexts p, q are $\sim_\varphi$ equivalent if for every nonempty forest t, the forests pt and qt are $\sim_\varphi$ equivalent.

Lemma 5.5. The relation $\sim_\varphi$ is a congruence of the free forest algebra A^Δ.

Proof. It is clear that $\sim_\varphi$ is an equivalence relation on forests and contexts. We need to show that it is a congruence. The first preparatory step is to show by induction on the size of a context p that for any two forests $t_1 \sim_\varphi t_2$ we have $pt_1 \sim_\varphi pt_2$.

Using this we can now show that $\sim_\varphi$ preserves the action. Suppose that $p_1 \sim_\varphi p_2$ and $t_1 \sim_\varphi t_2$. Then $p_1t_1 \sim_\varphi p_1t_2 \sim_\varphi p_2t_2$; where the second equivalence follows directly from the definition of $\sim_\varphi$ for contexts.

Next, we deal with monoid operations in H and V. From the definition it easily follows that if $s_1 \sim_\varphi t_1$ and $s_2 \sim_\varphi t_2$ then $s_1 + s_2 \sim_\varphi t_1 + t_2$. For the contexts take $p_1 \sim_\varphi p_2$ and $q_1 \sim_\varphi q_2$. For an arbitrary tree t we have: $q_1p_1t \sim_\varphi q_1p_2t \sim_\varphi q_2p_2t$. The first equivalence follows from the property proved in above, as $p_1t \sim_\varphi p_2t$.

Finally, we deal with the insertion operations. Take $s_1 \sim_\varphi s_2$ and an arbitrary tree t. We have $(\mathrm{in}_\mathrm{L}(s_1))t = s_1 + t \sim_\varphi s_2 + t = (\mathrm{in}_\mathrm{L}(s_2))t$. Q.E.D.

Lemma 5.6. The quotient $A^\Delta/\sim_\varphi$ is a forest algebra, and it recognizes L. Equations (1.1) and (1.2) are satisfied in the quotient.

Proof. For $A^\Delta/\sim_\varphi$ to be a forest algebra we must check if it is faithful. To check faithfulness take p, q which are not in $\sim_\varphi$ relation. Then there is a tree t such that $pt \not\sim_\varphi qt$ which gives: $[p][t] = [pt] \not\sim_\varphi [qt] = [q][t]$.

The language L is recognized by a canonical homomorphism assigning to each context its equivalence class, and the accepting set consisting of equivalence classes of trees from L. To show that it is correct we need to show that if two trees are equivalent then either both or none of them satisfies φ. This follows from the observation that φ is equivalent to a formula of the form $a \Rightarrow \varphi'$ where φ' is a boolean combination of some formulas form $\mathrm{cl}_{\mathsf{EF}}(\varphi)$.

A straightforward inspection shows that the equations are satisfied. For example, the fact that the trees vh and $h + vh$ have the same FT_φ value follows directly from the definition of the value. Q.E.D.

As the syntactic algebra for L is a morphic image of any other algebra recognizing L (cf. Proposition 3.14), all equations satisfied in $A^\Delta / \sim_\varphi$ must hold also in the syntactic algebra.

Corollary 5.7. The syntactic algebra of an EF definable forest language satisfies the equations (1.1) and (1.2).

5.3 Completeness

In this section we show that if a forest algebra satisfies the two EF equations, then every forest language recognized by this algebra can be defined in EF. This gives the desired result, since the syntactic algebra of L recognizes L.

From now on we fix a forest algebra (H, V) that recognizes a forest language L via a morphism

$$(\alpha, \beta) : A^\Delta \to (H, V) \ .$$

We assume that the forest algebra (H, V) satisfies the two EF equations (1.1) and (1.2). We shall show that L can be defined using an EF formula.

We first show that the EF equations imply two other properties:

$$h = h + h \tag{1.3}$$

$$w(vw)^\omega = (vw)^\omega \ . \tag{1.4}$$

These state idempotency of the horizontal monoid, and L-triviality of the vertical monoid, respectively. We need to explain the ω notation, though. In each finite semigroup (and hence in each monoid) S, there is a power $n \in \mathbb{N}$ such that all elements $s \in S$ satisfy $s^n = s^n s^n$. We refer to this power as ω, and use it in equations. In particular, every finite semigroup satisfies the equation: $s^\omega = s^\omega s^\omega$. The reader is advised to substitute "a very large power" for the term ω when reading the equations.

The idempotency of the horizontal monoid follows directly from the equation $vh = h + vh$, by taking v to be the neutral element of the vertical monoid. Observe that we always have $1h = h$, as $h = u0$ for some u and then $1(u0) = (1u)0 = u0$. The proof for the other equation is more complicated.

Lemma 5.8. For each $v, w \in V$, we have $w(vw)^\omega = (vw)^\omega$

Proof. First we show that the EF equations imply aperiodicity for the context monoid:

$$v^\omega = vv^\omega \ .$$

Indeed, by applying the first equation repeatedly to $v^\omega v^\omega$, we obtain:

$$v^\omega = v^\omega v^\omega = v^\omega + vv^\omega + vvv^\omega + \cdots + v^\omega v^\omega$$

Likewise for $vv^\omega v^\omega$:

$$vv^\omega = vv^\omega v^\omega = vv^\omega + vvv^\omega + vvvv^\omega + \cdots + v^\omega v^\omega + vv^\omega v^\omega$$

If we cancel out $vv^\omega v^\omega = vv^\omega$, and use idempotency and commutativity of H, we obtain the desired equality $v^\omega = vv^\omega$.

We now proceed to show the statement of the lemma.

$$w(vw)^\omega = (vw)^\omega + w(vw)^\omega = vw(vw)^\omega + w(vw)^\omega = vw(vw)^\omega = (vw)^\omega \ .$$

In the first and third equation we use $vh = h + vh$, while in the second and fourth we use aperiodicity. Q.E.D.

The main idea of the proof is to do an induction with respect to what forests can be found inside other forests. Given $g, h \in H$, we write $g \le h$ there is some context $u \in V$ such that $h = ug$. We write $g \sim h$ if $\le$ holds both ways. Here are three simple properties of these relations. The first is a direct consequence of the second EF equation. The other two require a short calculation.

Lemma 5.9. If $g \le h$ then $g + h = h$.

Lemma 5.10. If $g \sim h$ then $g = h$. In particular, $\le$ is a partial order.

Proof. Assume that $g \ne h$. If $g \sim h$ then there are contexts v, w such that $h = wg$ and $g = vh$. Iterating this process ω-times we obtain

$$h = wvh = (wv)^\omega h$$

But then, by applying Lemma 5.8, we get

$$h = (wv)^\omega h = v(wv)^\omega h = g \ .$$

Q.E.D.

Lemma 5.11. If $g_1 \le h_1$ and $g_2 \le h_2$ then $g_1 + g_2 \le h_1 + h_2$.

Proof. By assumption $h_1 = v_1 g_1$ and $h_2 = v_2 g_2$. Then, by using commutativity of H and equation (1.2), we get

$$h_1 + h_2 = v_1 g_1 + v_2 g_2 = v_1 g_1 + g_1 + v_2 g_2 + g_2 \geq g_1 + g_2 \ .$$

The last inequality is a consequence of the property $g + h \geq g$ which follows from the definition of the order as $g + h = (1 + h)g$. Q.E.D.

The next proposition is the main induction in the completeness proof:

Proposition 5.12. For every $h \in H$, there is an EF formula φ_h such that for every forest t and letter a we have

$$at \vDash \varphi_h \quad \text{iff} \quad \alpha(t) = h \ .$$

Proof. The proof is by induction on the depth of h in the order $\leq$, i.e. on the number of f satisfying $f < h$ (as usual, $<$ denotes the strict version of $\leq$).

Consider first the base case, when h is minimal for $\leq$; which by the way implies that $h = 0$ is the identity of the horizontal monoid. How can a forest t satisfy $\alpha(t) = h$? All leaves need to have labels $a \in A$ satisfying $\alpha(a) = h$; this can be easily tested in EF. Second, all internal nodes need to have labels $a \in A$ satisfying $\alpha(a)h = h$; this can also be tested in EF. These conditions are clearly necessary, but thanks to idempotency $h + h = h$, they are also sufficient. It remains to say how these conditions can be expressed in EF. The formula $\exists tt$ says that a node has a proper subtree, i.e., that a node is an internal node. So, the formula $\bigwedge_{b \in B} \neg\exists b \wedge \exists tt$ expresses the fact that no internal node has the label from a set B. Similarly one can say that no leaf has a label form B.

We now proceed with the induction step. We take some $h \in H$ and assume that the proposition is true for all $f < h$. We claim that a forest t satisfies $\alpha(t) = h$ iff the following three conditions hold:

- The forest t contains a *witness*. There are two types of witness. The first type, is a forest of a form $s_1 + s_2$ with $\alpha(s_1) + \alpha(s_2) = h$ but $\alpha(s_1), \alpha(s_2) < h$. The second type is a tree of the form as, with $\alpha(s) < h$ and $\beta(a)\alpha(s) = h$.
- For all subtrees as of t with s containing a witness, $\beta(a)(h) = h$ holds.
- For all subtrees as of t with $\alpha(s) < h$ we have $\beta(a)\alpha(s) \leq h$; moreover, for all subtrees s_1 and s_2 of t, with $\alpha(s_1) < h$ and $\alpha(s_2) < h$ we have $\alpha(s_1 + s_2) \leq h$.

These conditions can be easily written in EF using formulas φ_f for all $f < h$. So it remains to show that they are equivalent to $\alpha(t) = h$.

Suppose that the three conditions hold. By the first condition $\alpha(t) \geq h$. If $\alpha(t)$ were strictly greater than h then there would be a minimal size subtree s of t with $\alpha(s) \not\leq h$. It cannot be of the form $s_1 + s_2$ because, by Lemma 5.11, if $\alpha(s_1), \alpha(s_2) \leq h$ then $\alpha(s_1) + \alpha(s_2) \leq h$. So this minimal tree should be of the form as. It cannot be the case that $\alpha(s) = h$ because of the second property. If $\alpha(s) < h$ then the third property guarantees $\beta(a)\alpha(s) \leq h$, a contradiction.

Suppose now that $\alpha(t) = h$. It is clear that a minimal subtree of t which has the value h is a witness tree satisfying the first property. The second property is obvious. Regarding the third property, it is also clear that for every subtree of the form as if $\alpha(s) < h$ then $\beta(a)\alpha(s) \leq h$. It remains to check that for every two subtrees s_1, s_2 with $\alpha(s_1), \alpha(s_2) < h$ we have $\alpha(s_1) + \alpha(s_2) \leq h$. Take two such subtrees and a minimal tree containing both of them. If it is, say s_2, then $\alpha(s_1) < \alpha(s_2)$ and $\alpha(s_1) + \alpha(s_2) = \alpha(s_2) < h$. Otherwise, s_1 and s_2 are disjoint, and the minimal subtree has the form $b(t_1 + t_2 + t_3)$ with t_1 containing s_1, and t_2 containing s_2 (due to commutativity, the order of siblings does not matter). Now we have $\alpha(s_1) \leq \alpha(t_1)$ and $\alpha(s_2) \leq \alpha(t_2)$ which gives $\alpha(s_1 + s_2) \leq \alpha(t_1 + t_2) \leq \alpha(t) = h$ by Lemma 5.11. Q.E.D.

6 Conclusions and future work

This work is motivated by decidability problems for tree logics. As mentioned in the introduction, surprisingly little is known about this subject. We hope that this paper represents an advance, if only by making more explicit the algebraic questions that are behind these problems. Below we discuss some possibilities for future work.

Wherever there is an algebraic structure for recognizing languages, there is an Eilenberg theorem. This theorem gives a bijective mapping between classes of languages with good closure properties (language varieties) and classes of monoids with good closure properties (monoid varieties). It would be interesting to see how this extends to trees, i.e. study varieties forest algebras. Indeed, we have used equations to characterize EF, in particular the appropriate class of forest algebras will satisfy all closure properties usually required of a variety. The next step is to a develop variety theory, and check what classes of forest algebras can be defined using equations. Under a certain definition, it can be shown that first-order definable languages form a variety, so does CTL*, and chain logic. There are also logics that do not correspond to varieties; we have given EF^* as an example. This situation is well known in the word case: for some logics one needs to work in monoids, for others in semigroups. In the case of trees the choice is bigger. For example, a characterization of EF^* requires to forbid contexts consisting of just a hole. Another example is a characterization of first-order logic with two variables [4] where the empty tree is excluded.

A related topic concerns $\mathcal{C}$-varieties [16]. This is a notion from semigroup theory, which — among others — does away with the tedious distinction between semigroup and monoid varieties. It would be interesting to unify the variants mentioned above in a notion of $\mathcal{C}$-variety of forest algebras.

There are of course classes of tree languages — perhaps even more so in trees than words — that are not closed under boolean operations: take for instance languages defined by deterministic top down automata, or Σ_1 definable languages presented here. In the case of words, ordered semigroups extend the algebraic approach to such classes. It would be interesting to develop a similar concept of ordered forest algebras.

The logics considered in this paper cannot refer to the order on siblings in a tree. It would be worthwhile to find correct equations for logics with the order relation on siblings. It is also not clear how to cope with trees of bounded branching. One can also ask what is the right concept of forest algebras for languages of infinite trees.

References

[1] D. Beauquier and J.-E. Pin. Factors of words. In G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, editors, *ICALP*, volume 372 of *Lecture Notes in Computer Science*, pages 63–79. Springer, 1989.

[2] M. Benedikt and L. Segoufin. Regular tree languages definable in FO. In V. Diekert and B. Durand, editors, *STACS*, volume 3404 of *Lecture Notes in Computer Science*, pages 327–339. Springer, 2005.

[3] M. Bojańczyk. *Decidable Properties of Tree Languages.* PhD thesis, Warsaw University, 2004.

[4] M. Bojańczyk. Two-way unary temporal logic over trees. In *LICS*, pages 121–130. IEEE Computer Society, 2007.

[5] M. Bojańczyk and I. Walukiewicz. Characterizing EF and EX tree logics. *Theor. Comput. Sci.*, 358(2-3):255–272, 2006.

[6] J. Cohen, D. Perrin, and J.-E. Pin. On the expressive power of temporal logic. *J. Comput. Syst. Sci.*, 46(3):271–294, 1993.

[7] Z. Ésik and I. Szabolcs. Some varieties of finite tree automata related to restricted temporal logics. *Fund. Inform.*, 2007. To appear.

[8] Z. Ésik and P. Weil. On logically defined recognizable tree languages. In P. K. Pandya and J. Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 2003.

[9] U. Heuter. First-order properties of trees, star-free expressions, and aperiodicity. In R. Cori and M. Wirsing, editors, *STACS*, volume 294 of *Lecture Notes in Computer Science*, pages 136–148. Springer, 1988.

[10] L. Libkin. Logics for unranked trees: An overview. *Logical Methods in Computer Science*, 2(3), 2006.

[11] R. McNaughton and S. Papert. *Counter-free automata.* The M.I.T. Press, Cambridge, Mass.-London, 1971. M.I.T. Research Monograph, No. 65.

[12] J.-E. Pin. Logic, semigroups and automata on words. *Ann. Math. Artif. Intell.*, 16:343–384, 1996.

[13] A. Potthoff. First-order logic on finite trees. In P. D. Mosses, M. Nielsen, and M. I. Schwartzbach, editors, *TAPSOFT*, volume 915 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 1995.

[14] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[15] H. Straubing. *Finite automata, formal logic, and circuit complexity.* Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1994.

[16] H. Straubing. On logical descriptions of regular languages. In S. Rajsbaum, editor, *LATIN*, volume 2286 of *Lecture Notes in Computer Science*, pages 528–538. Springer, 2002.

[17] J. W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *J. Comput. Syst. Sci.*, 1(4):317–322, 1967.

[18] D. Thérien and A. Weiss. Graph congruences and wreath products. *J. Pure Appl. Algebra*, 36(2):205–215, 1985.

[19] D. Thérien and Th. Wilke. Temporal logic and semidirect products: An effective characterization of the until hierarchy. In *FOCS*, pages 256–263, 1996.

[20] D. Thérien and Th. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC*, pages 234–240, 1998.

[21] T. Wilke. Algebras for classifying regular tree languages and an application to frontier testability. In A. Lingas, R. G. Karlsson, and S. Carlsson, editors, *ICALP*, volume 700 of *Lecture Notes in Computer Science*, pages 347–358. Springer, 1993.

[22] Th. Wilke. Classifying discrete temporal properties. In C. Meinel and S. Tison, editors, *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1999.

[23] Z. Wu. A note on the characterization of `TL[EF]`. *Inform. Process. Lett.*, 102(2-3):48–54, 2007.

Automata and semigroups recognizing infinite words

Olivier Carton[1]
Dominique Perrin[2]
Jean-Éric Pin[1]

[1] Laboratoire d'Informatique Algorithmique: Fondements et Applications
Université Paris Diderot, Paris 7
Case 7014
75205 Paris Cedex 13, France
Olivier.Carton@liafa.jussieu.fr, Jean-Eric.Pin@liafa.jussieu.fr

[2] Institut Gaspard Monge
Université de Paris-Est
5, boulevard Descartes
77454 Champs-sur-Marne, France
perrin@univ-mlv.fr

Abstract

This paper is a survey on the algebraic approach to the theory of automata accepting infinite words. We discuss the various acceptance modes (Büchi automata, Muller automata, transition automata, weak recognition by a finite semigroup, ω-semigroups) and prove their equivalence. We also give two algebraic proofs of McNaughton's theorem on the equivalence between Büchi and Muller automata. Finally, we present some recent work on prophetic automata and discuss its extension to transfinite words.

1 Introduction

Among the many research contributions of Wolfgang Thomas, those regarding automata on infinite words and more generally, on infinite objects, have been highly inspiring to the authors. In particular, we should like to emphasize the historical importance of his early papers [33, 34, 35], his illuminating surveys [36, 37] and the Lecture Notes volume on games and automata [15].

Besides being a source of inspiration, Wolfgang always had nice words for our own research on the *algebraic approach* to automata theory. This survey, which presents this theory for infinite words, owes much to his encouragement.

Büchi has extended the classical theory of languages to infinite words instead of finite ones. Most notions and results known for finite words extend to infinite words, often at the price of more difficult proofs. For example, proving that rational languages are closed under Boolean operations becomes, in the infinite case, a delicate result, the proof of which makes use of Ramsey theorem. In the same way, the determinization of automata, an easy algorithm on finite words, turns to a difficult theorem in the infinite case.

Not surprisingly, the same kind of obstacle occurred in the algebraic approach to automata theory. It was soon recognized that finite automata are closely linked with finite semigroups, thus giving an algebraic counterpart of the definition of recognizability by finite automata. In this setting, every rational language X of A^+ is recognized by a morphism from A^+ onto a finite semigroup. There is also a minimal semigroup recognizing X, called the syntactic semigroup of X. The success of the algebraic approach for studying regular languages was already firmly established by the end of the seventies, but it took another ten years to find the appropriate framework for infinite words. Semigroups are replaced by ω-semigroups, which are, roughly speaking, semigroups equipped with an infinite product. In this new setting, the definitions of recognizable sets of infinite words and of syntactic congruence become natural and most results valid for finite words can be adapted to infinite words. Carrying on the work of Arnold [1], Pécuchet [21, 20] and the second author [22, 23] , Wilke [38, 39] has pushed the analogy with the theory for finite words sufficiently far to obtain a counterpart of Eilenberg's variety theorem for finite or infinite words. This theory was further extended by using ordered ω-semigroups [26, 24]. Notwithstanding the importance of the variety theory, we do not cover it in this article but rather choose to present some applications of the algebraic approach to automata theory. The first nontrivial application is the construction of a Muller automaton, given a finite semigroup weakly recognizing a language. The second one is a purely algebraic proof of the theorem of McNaughton stating that any recognizable subset of infinite words is a Boolean combination of deterministic recognizable sets. The third one deals with prophetic automata, a subclass of Büchi automata in which any infinite word is the label of exactly one final path. The main result states that these automata are equivalent to Büchi automata. We show, however, that this result does not extend to words indexed by ordinals.

Our paper has the character of a survey. For the reader's convenience it reproduces some of the material published in the book *Semigroups and automata on infinite words* [25], which owes a debt of gratitude to Wolfgang Thomas. Proofs are often only sketched in the present paper, but complete proofs can be found in [25]. Other surveys on automata and infinite words include [23, 24, 36, 37, 32].

Our article is divided into seven sections. Automata on infinite words are introduced in Section 2. Algebraic recognition modes are discussed in

Section 3. The syntactic congruence is defined in Section 4. In Section 5, we show that all recognition modes defined so far are equivalent. Sections 6 and 7 illustrate the power of the algebraic approach. In Section 6, we give an algebraic proof of McNaughton's theorem. Section 7 is devoted to prophetic automata.

2 Automata

Let A be an alphabet. We denote by A^+, A^* and A^ω, respectively, the sets of nonempty finite words, finite words and infinite words on the alphabet A. We also denote by A^∞ the set $A^* \cup A^\omega$ of finite or infinite words on A. By definition, an *ω-rational* subset of A^ω is a finite union of sets of the form XY^ω where X and Y are rational subsets of A^*.

An *automaton* is given by a finite alphabet A, a finite set of states Q and a subset E of $Q \times A \times Q$, called the set of *edges* or *transitions*. Two transitions (p, a, q) and (p', a', q') are called *consecutive* if $q = p'$. An *infinite path* in the automaton $\mathcal{A}$ is an infinite sequence p of consecutive transitions

$$p: \quad q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \ \cdots$$

The state q_0 is the *origin* of the infinite path and the infinite word $a_0a_1\cdots$ is its *label*. We say that the path p *passes infinitely often* through a state q (or that p *visits* q infinitely often, or yet that q is *infinitely repeated* in p) if there are infinitely many integers n such that $q_n = q$. The set of *infinitely repeated* states in p is denoted by $\mathrm{Inf}(p)$.

An automaton $\mathcal{A} = (Q, A, E)$ is said to have *deterministic transitions*, if, for every state $q \in Q$ and every letter $a \in A$, there is at most one state q' such that (q, a, q') is a transition. It is *deterministic* if it has deterministic transitions and if I is a singleton. Dually, $\mathcal{A}$ has *complete transitions* if, for every state $q \in Q$ and every letter $a \in A$, there is at least one state q' such that (q, a, q') is a transition.

Acceptance modes are usually defined by specifying a set of *successful* finite or infinite paths. This gives rise to different types of automata. We shall only recall here the definition of two classes: the Büchi automata and the Muller automata.

2.1 Büchi automata

In the model introduced by Büchi, one is given a set of *initial states* I and a set of *final states* F. Here are the precise definitions.

Let $\mathcal{A} = (Q, A, E, I, F)$ be a Büchi automaton. We say that an infinite path in $\mathcal{A}$ is *initial* if its origin is in I and *final* if it visits F infinitely often. It is *successful* if it is initial and final. The set of infinite words *recognized* by $\mathcal{A}$ is the set, denoted by $L^\omega(\mathcal{A})$, of labels of infinite successful paths in $\mathcal{A}$. It is also the set of labels of infinite initial paths p in $\mathcal{A}$ and such that $\mathrm{Inf}(p) \cap F \neq \varnothing$.

By definition, a set of infinite words is *recognizable* if it is recognized by some finite Büchi automaton. Büchi has shown that Kleene's theorem on regular languages extends to infinite words.

Theorem 2.1. A set of infinite words is recognizable if and only if it is ω-rational.

The notion of trim automaton can also be adapted to the case of infinite words. A state q is called *accessible* if there is a (possibly empty) finite initial path in $\mathcal{A}$ ending in q. A state q is called *coaccessible* if there exists an infinite final path starting at q. Finally, $\mathcal{A}$ is *trim* if all its states are both accessible and coaccessible.

It is easy to see that every Büchi automaton is equivalent to a trim Büchi automaton. For this reason, we shall assume that all the automata considered in this paper are trim.

So far, extending automata theory to infinite words did not raise any insuperable problems. However, it starts getting harder when it comes to determinism.

The description of the subsets of A^ω recognized by deterministic Büchi automata involves a new operator. For a subset L of A^*, let

$$\overrightarrow{L} = \{u \in A^\omega \mid u \text{ has infinitely many prefixes in } L\}.$$

Example 2.2.

(a) If $L = a^*b$, then $\overrightarrow{L} = \varnothing$.

(b) If $L = (ab)^+$, then $\overrightarrow{L} = (ab)^\omega$.

(c) If $L = (a^*b)^+ = (a+b)^*b$, that is if L is the set of words ending with b, then $\overrightarrow{L} = (a^*b)^\omega$, which is the set of infinite words containing infinitely many occurrences of b.

The following example shows that not every set of words can be written in the form $\overrightarrow{L}$.

Example 2.3. The set $X = (a+b)^*a^\omega$ of words with a *finite* number of occurrences of b is not of the form $\overrightarrow{L}$. Otherwise, the word ba^ω would have a prefix $u_1 = ba^{n_1}$ in L, the word $ba^{n_1}ba^\omega$ would have a prefix $u_2 = ba^{n_1}ba^{n_2}$ in L, etc. and the infinite word $u = ba^{n_1}ba^{n_2}ba^{n_3}\cdots$ would have an infinity of prefixes in L and hence would be in $\overrightarrow{L}$. This is impossible, since u contains infinitely many b's.

A set of infinite words which can be recognized by a deterministic Büchi automaton is called *deterministic*.

Theorem 2.4. A subset X of A^ω is deterministic if and only if there exists a recognizable set L of A^+ such that $X = \overrightarrow{L}$.

2.2 Muller automata

Contrary to the case of finite words, deterministic Büchi automata fail to recognize all recognizable sets of infinite words. This is the motivation for introducing Muller automata which are also deterministic, but have a more powerful acceptance mode. In this model, an infinite path p is final if the set $\mathrm{Inf}(p)$ belongs to a prescribed set $\mathcal{T}$ of sets of states. The definition of initial and successful paths are unchanged.

A *Muller automaton* is a 5-tuple $\mathcal{A} = (Q, A, E, i, \mathcal{T})$ where (Q, A, E) is a deterministic automaton, i is the initial state and $\mathcal{T}$ is a set of subsets of Q, called the *table of states* of the automaton. The set of infinite words recognized by $\mathcal{A}$ is the set, denoted by $L^\omega(\mathcal{A})$, of labels of infinite successful paths in $\mathcal{A}$.

A fundamental result, due to R. McNaughton [18], states that any Büchi automaton is equivalent to a Muller automaton.

Theorem 2.5. Any recognizable set of infinite words can be recognized by a Muller automaton.

This implies in particular that recognizable sets of infinite words are closed under complementation, a result proved for the first time by Büchi in a direct way.

2.3 Transition automata

It is sometimes convenient to use a variant of automata in which a set of final transitions is specified, instead of the usual set of final states. This idea can be applied to all variants of automata.

Formally, a *Büchi transition automaton* is a 5-tuple $\mathcal{A} = (Q, A, E, I, F)$ where (Q, A, E) is an automaton, $I \subseteq Q$ is the set of initial states and $F \subseteq E$ is the set of final transitions. If p is an infinite path, we denote by $\mathrm{Inf}_T(p)$ the set of transitions through which p goes infinitely often. A path p is final if it goes through F infinitely often, that is, if $\mathrm{Inf}_T(p) \cap F \neq \varnothing$.

Similarly, a *transition Muller automaton* is a 5-tuple $\mathcal{A} = (Q, A, E, I, \mathcal{T})$ where (Q, A, E) is a finite deterministic automaton, i is the initial state and $\mathcal{T}$ is a set of subsets of E, called the *table of transitions* of the automaton. A path is final if $\mathrm{Inf}_T(p) \in \mathcal{T}$, that is, if the set of transitions occurring infinitely often in p is an element of the table.

Proposition 2.6.

(1) Büchi automata and transition Büchi automata are equivalent.

(2) Muller automata and transition Muller automata are equivalent.

3 Algebraic recognition modes

In this section, we give an historical survey on the various algebraic notions of recognizability that have been considered. The two earlier ones, weak and strong recognition, are now superseded by the notions of ω-semigroupsand Wilke algebras.

Recall that a *semigroup* is a set equipped with an associative operation which does not necessarily admit an identity. If S is a semigroup, S^1 denotes the monoid equal to S if S is a monoid, and to $S \cup \{1\}$ if S is not a monoid. In the latter case, the operation of S is completed by the rules $1s = s1 = s$ for each $s \in S^1$. An element e of S is *idempotent* if $e^2 = e$.

The preorder $\leqslant_{\mathcal{R}}$ is defined on S by setting $s \leqslant_{\mathcal{R}} s'$ if there exists $t \in S^1$ such that $s = s't$. We also write $s \mathrel{\mathcal{R}} s'$ if $s \leqslant_{\mathcal{R}} s'$ and $s' \leqslant_{\mathcal{R}} s$ and $s <_{\mathcal{R}} s'$ if $s \leqslant_{\mathcal{R}} s'$ and $s' \not\leqslant_{\mathcal{R}} s$. The equivalence classes of the relation $\mathcal{R}$ are called the *$\mathcal{R}$-classes* of S.

3.1 Weak recognition

The early attempts aimed at understanding the behaviour of a semigroup morphism from A^+ onto a finite semigroup. The key result is a consequence of Ramsey's theorem in combinatorics, which involves the notion of a linked pair: a *linked pair* of a finite semigroup S is a pair (s, e) of elements of S satisfying $se = s$ and $e^2 = e$.

Theorem 3.1. Let $\varphi : A^+ \to S$ be a morphism from A^+ into a finite semigroup S. For each infinite word $u \in A^\omega$, there exist a linked pair (s, e) of S and a factorization $u = u_0 u_1 \cdots$ of u as a product of words of A^+ such that $\varphi(u_0) = s$ and $\varphi(u_n) = e$ for all $n > 0$.

Theorem 3.1 is frequently used in a slightly different form:

Proposition 3.2. Let $\varphi : A^+ \to S$ be a morphism from A^+ into a finite semigroup S. Let u be an infinite word of A^ω, and let $u = u_0 u_1 \ldots$ be a factorisation of u in words of A^+. Then there exist a linked pair (s, e) of S and a strictly increasing sequence of integers $(k_n)_{n \geqslant 0}$ such that $\varphi(u_0 u_1 \cdots u_{k_0 - 1}) = s$ and $\varphi(u_{k_n} u_{k_n + 1} \cdots u_{k_{n+1} - 1}) = e$ for every $n \geqslant 0$.

Theorem 3.1 lead to the first attempt to extend the notion of recognizable sets. Let us call *φ-simple* a set of infinite words of the form $\varphi^{-1}(s) \left(\varphi^{-1}(e)\right)^\omega$, where (s, e) is a linked pair of S. Then we say that a subset of A^ω is *weakly recognized* by φ if it is a finite union of φ-simple subsets. The following result justifies the term "recognized".

Proposition 3.3. A set of infinite words is recognizable if and only if it is weakly recognized by some morphism onto a finite semigroup.

However, the notion of weak recognition has several drawbacks: there is no natural notion of syntactic semigroup, dealing with complementation is uneasy and more generally, the algebraic tools that were present in the case of finite words are missing.

3.2 Strong recognition

This notion emerged as an attempt to obtain an algebraic proof of the closure of recognizable sets of infinite words under complement.

Let $\varphi : A^+ \to S$ be a morphism from A^+ into a finite semigroup S. Then φ *strongly recognizes* (or *saturates*) a subset X of A^ω if all the φ-simple sets have a trivial intersection with X, that is, for each linked pair (s, e) of S,

$$\varphi^{-1}(s)\left(\varphi^{-1}(e)\right)^\omega \cap X = \varnothing \text{ or } \varphi^{-1}(s)\left(\varphi^{-1}(e)\right)^\omega \subseteq X$$

Theorem 3.1 shows that A^ω is a finite union of φ-simple sets. It follows that if a morphism strongly recognizes a set of infinite words, then it also weakly recognizes it. Furthermore, Proposition 3.3 can be improved.

Proposition 3.4. A set of infinite words is recognizable if and only if it is strongly recognized by some morphism onto a finite semigroup.

The proof relies on a construction which is interesting on its own right. Given a semigroup S, we define a new semigroup

$$T = \{\left(\begin{smallmatrix} s & P \\ 0 & s \end{smallmatrix}\right) \mid \ s \in S, \ P \text{ is a subset of } S \times S\}$$

with multiplication defined by

$$\left(\begin{smallmatrix} s & P \\ 0 & s \end{smallmatrix}\right)\left(\begin{smallmatrix} t & Q \\ 0 & t \end{smallmatrix}\right) = \left(\begin{smallmatrix} st & sQ\cup Pt \\ 0 & st \end{smallmatrix}\right)$$

where $sQ = \{(sq_1, q_2) \mid (q_1, q_2) \in Q\}$ and $Pt = \{(p_1, p_2t) \mid (p_1, p_2) \in P\}$. Let now φ be a morphism from A^+ onto S. Then one can show that the map $\psi : A^+ \to T$ defined by

$$\psi(u) = \begin{pmatrix} \varphi(u) & \tau(u) \\ 0 & \varphi(u) \end{pmatrix} \quad \text{with} \quad \tau(u) = \{(\varphi(u_1), \varphi(u_2)) \mid u = u_1u_2\}$$

is a semigroup morphism and that any set of infinite words weakly recognized by φ is strongly recognized by ψ.

Proposition 3.4 leads to a simple proof of Büchi's complementation theorem.

Corollary 3.5. Recognizable sets of infinite words are closed under complement.

Proof. Indeed, if a morphism strongly recognizes a set of infinite words, it also recognizes its complement. Q.E.D.

3.3 ω-semigroups and Wilke algebras

Although strong recognition constituted an improvement over weak recognition, there were still obstacles to extend to infinite words Eilenberg's variety theorem, which gives a correspondence between recognizable sets and finite semigroups. The solution was found by Wilke [38] and reformulated in slightly different terms by the two last authors in [24]. The idea is to use an algebraic structure, called an ω-semigroup, which is a sort of semigroup in which infinite products are defined. This structure was actually implicit in the original construction of Büchi to recognize the complement [7].

3.3.1 ω-semigroups

An *ω-semigroup* is a two-sorted algebra $S = (S_+, S_\omega)$ equipped with the following operations:

(a) A binary operation defined on S_+ and denoted multiplicatively,

(b) A mapping $S_+ \times S_\omega \to S_\omega$, called *mixed product*, that associates with each pair $(s, t) \in S_+ \times S_\omega$ an element of S_ω denoted st,

(c) A surjective mapping $\pi : S_+^\omega \to S_\omega$, called *infinite product*

These three operations satisfy the following properties:

(1) S_+, equipped with the binary operation, is a semigroup,

(2) for every $s, t \in S_+$ and for every $u \in S_\omega$, $s(tu) = (st)u$,

(3) for every increasing sequence $(k_n)_{n>0}$ and for every sequence $(s_n)_{n\geqslant 0}$ of elements of S_+,

$$\pi(s_0 s_1 \ \cdots \ s_{k_1-1}, s_{k_1} s_{k_1+1} \ \cdots \ s_{k_2-1}, \ \ldots) = \pi(s_0, s_1, s_2, \ldots)$$

(4) for every $s \in S_+$ and for every sequence $(s_n)_{n\geqslant 0}$ of elements of S_+,

$$s\pi(s_0, s_1, s_2, \ldots) = \pi(s, s_0, s_1, s_2, \ldots)$$

These conditions can be thought of as an extension of associativity. In particular, conditions (3) and (4) show that one can replace $\pi(s_0, s_1, s_2, \ldots)$ by $s_0 s_1 s_2 \cdots$ without ambiguity. We shall use this simplified notation in the sequel.

Example 3.6.

(1) We denote by A^∞ the ω-semigroup (A^+, A^ω) equipped with the usual concatenation product. One can show that A^∞ is the free ω-semigroup generated by A.

(2) The *trivial* ω-semigroup is the ω-semigroup $1 = (\{1\}, \{a\})$, obtained by equipping the trivial semigroup $\{1\}$ with an infinite product: the unique way is to declare that every infinite product is equal to a.

(3) Consider the ω-semigroup $S = (\{0, 1\}, \{a\})$ defined as follows: every infinite product is equal to a and every finite product $s_0 s_1 \ldots s_n$ is

equal to 0 except if all the s_i's are equal to 1. In particular, the elements 0 and 1 are idempotents and thus, for all $n > 0$, $1^n \neq 0^n$. Nevertheless $1^\omega = 0^\omega = a$.

These examples, especially the third one, make apparent an algorithmic problem. Even if the sets S_+ and S_∞ are finite, the infinite product is still an operation of infinite arity and it is not clear how to define it as a finite object. The problem was solved by Wilke [38], who proved that finite ω-semigroups are totally determined by only three operations of finite arity. This leads to the notion of Wilke algebras, that we now define.

3.3.2 Wilke Algebras

A *Wilke algebra* is a two-sorted algebra $S = (S_+, S_\omega)$, equipped with the following operations:

(1) an associative product on S_+,

(2) a mixed product, which maps each pair $(s, t) \in S_+ \times S_\omega$ onto an element of S_ω denoted by st, such that, for every $s, t \in S_+$ and for every $u \in S_\omega$, $s(tu) = (st)u$,

(3) a map from S_+ in S_ω, denoted by $s \to s^\omega$ satisfying, for each $s, t \in S_+$,

$$\begin{aligned} s(ts)^\omega &= (st)^\omega \\ (s^n)^\omega &= s^\omega \quad \text{for each } n > 0 \end{aligned}$$

and such that every element of S_ω can be written as st^ω with $s, t \in S_+$.

Wilke's theorem states the equivalence between finite Wilke algebra and finite ω-semigroup. A consequence is that for a finite ω-semigroup, any infinite product is equal to an element of the form st^ω, with $s, t \in S_+$.

Theorem 3.7. Every finite Wilke algebra $S = (S_+, S_\omega)$ can be equipped, in a unique way, with a structure of ω-semigroup that inherits the given mixed product and such that, for each $s \in S_+$, the infinite product $sss\cdots$ is equal to s^ω.

We still need to define morphisms for these algebras. We shall just give the definition for ω-semigroups, but the definition for Wilke algebras would be similar.

3.3.3 Morphisms of ω-semigroups

As ω-semigroups are two-sorted algebras, morphisms are defined as pairs of morphisms. Given two ω-semigroups $S = (S_+, S_\omega)$ and $T = (T_+, T_\omega)$, a *morphism of ω-semigroups* S is a pair $\varphi = (\varphi_+, \varphi_\omega)$ consisting of a semigroup morphism $\varphi_+ : S_+ \to T_+$ and of a mapping $\varphi_\omega : S_\omega \to T_\omega$ preserving the infinite product: for every sequence $(s_n)_{n\in\mathbb{N}}$ of elements of S_+,

$$\varphi_\omega(s_0 s_1 s_2 \cdots) = \varphi_+(s_0)\varphi_+(s_1)\varphi_+(s_2)\cdots$$

It is an easy exercise to verify that these conditions imply that φ also preserves the mixed product, that is, for all $s \in S_+$, and for each $t \in S_\omega$,

$$\varphi_+(s)\varphi_\omega(t) = \varphi_\omega(st)$$

Algebraic concepts like isomorphism, ω-subsemigroup, congruence, quotient, division are easily adapted from semigroups to ω-semigroups. We are now ready for our algebraic version of recognizability.

3.3.4 Recognition by morphism of ω-semigroups.

In the context of ω-semigroups, it is more natural to define recognizable subsets of A^∞, although we shall mainly use this definition for subsets of A^ω. This global point of view has been confirmed to be the right one in the study of words indexed by ordinals or by linear orders [3, 4, 5, 6, 28]. Thus a subset X of A^∞ is split into two components $X_+ = X \cap A^+$ and $X_\omega = X \cap A^\omega$.

Let $S = (S_+, S_\omega)$ be a finite ω-semigroup, and let $\varphi : A^\infty \to S$ be a morphism. We say that φ *recognizes* a subset X of A^∞ if there exist a pair $P = (P_+, P_\omega)$ with $P_+ \subseteq S_+$ and $P_\omega \subseteq S_\omega$ such that $X_+ = \varphi_+^{-1}(P_+)$ and $X_\omega = \varphi_\omega^{-1}(P_\omega)$. In the sequel, we shall often omit the subscripts and simply write $X = \varphi^{-1}(P)$. It is time again to justify our terminology by a theorem, whose proof will be given in Section 5.

Theorem 3.8. A set of infinite words is recognizable if and only if it is recognized by some morphism onto a finite ω-semigroup.

Example 3.9. Let $A = \{a, b\}$, and consider the ω-semigroup

$$S = (\{1, 0\}, \{1^\omega, 0^\omega\})$$

equipped with the operations $11 = 1$, $10 = 01 = 00 = 0$, $11^\omega = 1^\omega$, $10^\omega = 00^\omega = 01^\omega = 0^\omega$. Let $\varphi : A^\infty \to S$ be the morphism of ω-semigroups defined by $\varphi(a) = 1$ and $\varphi(b) = 0$. We have

$\varphi^{-1}(1) = a^+$ (finite words containing no occurrence of b),

$\varphi^{-1}(0) = A^*bA^*$ (finite words containing at least one occurrence of b),

$\varphi^{-1}(1^\omega) = a^\omega$ (infinite words containing no occurrence of b),

$\varphi^{-1}(0^\omega) = A^\omega \setminus a^\omega$ (infinite words containing at least one occurrence of b),

The morphism φ recognizes each of these sets, as well as any union of these sets.

Example 3.10. Let us take the same ω-semigroup S and consider the morphism of ω-semigroups $\varphi : A^\infty \to S$ defined by $\varphi(a) = s$ for each $a \in A$. We have $\varphi^{-1}(s) = A^+$, $\varphi^{-1}(t) = \varnothing$ and $\varphi^{-1}(u) = A^\omega$. Thus the morphism φ recognizes the empty set and the sets A^+, A^ω and A^∞.

Example 3.11. Let $A = \{a, b\}$, and consider the ω-semigroup

$$S = (\{a, b\}, \{a^\omega, b^\omega\})$$

equipped with the following operations:

$$\begin{array}{cccc} aa = a & ab = a & aa^\omega = a^\omega & ab^\omega = a^\omega \\ ba = b & bb = b & ba^\omega = b^\omega & bb^\omega = b^\omega \end{array}$$

The morphism of ω-semigroups $\varphi : A^\infty \to S$ defined by $\varphi(a) = a$ and $\varphi(b) = b$ recognizes aA^ω since we have $\varphi^{-1}(a^\omega) = aA^\omega$.

Boolean operations can be easily translated in terms of morphisms. Let us start with a result which allows us to treat separately, the subsets of A_+ and those of A_ω.

Proposition 3.12. Let φ be a morphism of ω-semigroups recognizing a subset X of A^∞. Then the subsets X_+, X_ω, $X_+ \cup A^\omega$ and $A^+ \cup X_\omega$ are also recognized by φ.

We now consider the complement.

Proposition 3.13. Let φ be a morphism of ω-semigroups recognizing a subset X of A^∞ (resp. A_+, A_ω). Then φ also recognizes the complement of X in A^∞ (resp. A_+, A_ω).

For union and intersection, we have the following results.

Proposition 3.14. Let $(\varphi_i)_{i \in F} : A^\infty \to S_i$ be a family of surjective morphisms recognizing a subset X_i of A^∞. Then the subsets $\bigcup_{i \in F} X_i$ and $\bigcap_{i \in F} X_i$ are recognized by an ω-subsemigroup of the product $\prod_{i \in F} S_i$.

In the same spirit, the following properties hold:

Proposition 3.15. Let $\alpha : A^\infty \to B^\infty$ be a morphism of ω-semigroups and let φ be a morphism of ω-semigroups recognizing a subset X of B^∞. Then the morphism $\varphi \circ \alpha$ recognizes the set $\alpha^{-1}(X)$.

4 Syntactic congruence

The definition of the syntactic congruence of a recognizable subset of infinite words is due to Arnold [1]. It was then adapted to the context of ω-semigroups. Therefore, this definition can be given for recognizable subsets of A^∞, but we restrict ourself to the case of subsets of A^ω.

The *syntactic congruence* of a recognizable subset of A^ω is defined on A^+ by $u \sim_X v$ if and only if, for each $x, y \in A^*$ and for each $z \in A^+$,

$$\begin{aligned} xuyz^\omega \in X &\Longleftrightarrow xvyz^\omega \in X \\ x(uy)^\omega \in X &\Longleftrightarrow x(vy)^\omega \in X \end{aligned} \tag{4.1}$$

and on A^ω by $u \sim_X v$ if and only if, for each $x \in A^*$,

$$xu \in X \iff xv \in X \tag{4.2}$$

The syntactic ω-semigroup of X is the quotient of A^∞ by the syntactic congruence of X.

Example 4.1. Let $A = \{a, b\}$ and $X = \{a^\omega\}$. The syntactic congruence of X divides A^+ into two classes: a^+ and A^*bA^* and A^ω into two classes also: A^*bA^ω and a^ω. The syntactic ω-semigroup of X is the four element ω-semigroup of Example 3.9.

Example 4.2. Let $A = \{a, b\}$ and let $X = aA^\omega$. The syntactic ω-semigroup of X is the ω-semigroup of Example 3.11.

Example 4.3. When X is not recognizable, the equivalence relation $\sim$ defined on A^+ by (4.1) and on A^ω by (4.2) is not in general a congruence. For instance, let $A = \{a, b\}$ and $X = \{ba^1ba^2ba^3b\cdots\}$. We have, for each $n > 0$, $b \sim_X ba^n$, but nevertheless $ba^1ba^2ba^3b\cdots$ is not equivalent to b^ω since $ba^1ba^2ba^3b\cdots \in X$ but $b^\omega \notin X$.

Example 4.4. Let $X = (a\{b, c\}^* \cup \{b\})^\omega$. We shall compute in Example 5.3 an ω-semigroup S recognizing this set. One can show that its syntactic ω-semigroup is $S(X) = (\{a, b, c, ca\}, \{a^\omega, c^\omega, (ca)^\omega\})$, presented by the relations

$$\begin{array}{ccccc}
a^2 = a & ab = a & ac = a & ba = a & b^2 = b \\
bc = c & cb = c & c^2 = c & b^\omega = a^\omega & ba^\omega = a^\omega \\
ac^\omega = c^\omega & ca^\omega = (ca)^\omega & a(ca)^\omega = a^\omega & b(ca)^\omega = (ca)^\omega & c(ca)^\omega = (ca)^\omega
\end{array}$$

The syntactic ω-semigroup is the least ω-semigroup recognizing a recognizable set. More precisely, we have the following statement:

Proposition 4.5. Let X be a recognizable subset of A^∞. An ω-semigroup S recognizes X if and only if the syntactic ω-semigroup of X is a quotient of S.

Note in particular that, if $u \sim_X v$ for two words u, v of A^+, then, for all $x \in A^*$ and $z \in A^\omega$

$$xuz \in X \iff xvz \in X \tag{4.3}$$

Indeed, if $\varphi : A^\infty \to S$ denotes the syntactic morphism of X, the condition $u \sim_X v$ implies $\varphi(u) = \varphi(v)$. It follows that $\varphi(xuz) = \varphi(xvz)$, which gives (4.3).

5 Conversions from one acceptance mode into one another

In this section, we explain how to convert the various acceptance modes one into one another. We have already seen how to pass from weak to strong recognition by a finite semigroup. We shall now describe, in order, the conversions form weak recognition to Büchi automata, from Büchi automata to ω-semigroups, from strong recognition to ω-semigroups and finally from weak recognition to Muller automata.

5.1 From weak recognition to Büchi automata

Let $\varphi : A^+ \to S$ be a morphism from A^+ onto a finite semigroup S. First observe that, given Büchi automata $\mathcal{A}_1, \dots, \mathcal{A}_n$, their disjoint union recognizes the set $L^\omega(\mathcal{A}_1) \cup \dots \cup L^\omega(\mathcal{A}_n)$. Therefore, we may suppose that X is a φ-simple set of infinite words, say $X = \varphi^{-1}(s)(\varphi^{-1}(e))^\omega$ for some linked pair (s, e) of S. We construct a nondeterministic Büchi automaton $\mathcal{A}$ that accepts X as follows. The set Q of states of $\mathcal{A}$ is the set $S^I = S \cup \{f\}$ where f is a new neutral element added to S even if S has already one. The product of S is thus extended to S^I by setting $tf = ft = t$ for any $t \in S^I$. The initial state of $\mathcal{A}$ is s and the unique final state is f. The set of transitions is

$$E = \{\varphi(a)t \xrightarrow{a} t \mid a \in A \text{ and } t \in Q\} \\ \cup \{f \xrightarrow{a} t \mid a \in A, t \in Q \text{ and } \varphi(a)t = e\}.$$

Let $t \in S$. It is easily proved that a word w satisfies $\varphi(w) = t$ if and only if it labels a path from t to f visiting f only at the end. It follows that w labels a path from f to f if and only if $\varphi(w) = e$ and thus $\mathcal{A}$ accepts X.

The previous construction has one main drawback. The transition semigroup of the automaton $\mathcal{A}$ may not belong to the variety of finite semigroups generated by S, as shown by the following example.

Example 5.1. Let S be the semigroup $\{0, 1\}$ endowed with the usual multiplication. Let A be the alphabet $\{a, b\}$ and $\varphi : A^+ \to S$ be the morphism defined by $\varphi(a) = 0$ and $\varphi(b) = 1$. Let (s, e) be the pair $(0, 0)$. The set $\varphi^{-1}(s)(\varphi^{-1}(e))^\omega$ is thus equal to $(b^*a)^\omega$. The automaton $\mathcal{A}$ obtained with the previous construction is pictured in Figure 1. The semigroup S is commutative but the transition semigroup of $\mathcal{A}$ is not. Indeed, there is a path from 1 to 0 labeled by ba but there is no path from 1 to 0 labeled by ab.

In order to solve this problem, Pécuchet [21] proposed the following construction, which is quite similar to the previous one but has better properties. The set of states of the automaton is still the set $S^I = S \cup \{f\}$. The

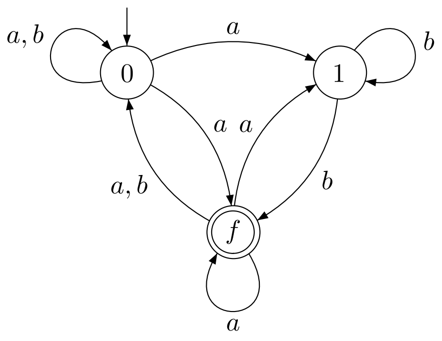


FIGURE 1. The automaton $\mathcal{A}$.

initial state is s and the unique final state is f. The set E of transitions is modified as follows:

$$E = \{t' \xrightarrow{a} t \mid a \in A, t, t' \in Q \text{ and } (t' = \varphi(a)t \text{ or } t'e = \varphi(a)t)\}$$

The automaton $\mathcal{B}$ obtained with this construction is pictured in Figure 2. It can be proved that for any states t and t', there is a path from t' to t

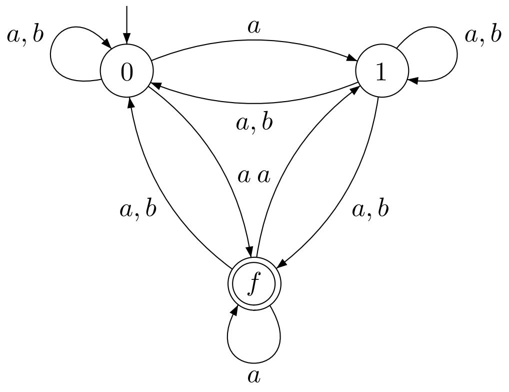


FIGURE 2. The automaton $\mathcal{B}$.

labeled by w if and only if $t' = \varphi(w)t$ or $t'e = \varphi(w)t$. It follows that if two words w and w' satisfy $\varphi(w) = \varphi(w')$, there is path from t' to t labeled by w if and only if there is path from t' to t labeled by w'. This means that the transition semigroup of the automaton $\mathcal{B}$ divides the semigroup S and hence belongs to the variety of finite semigroups generated by S.

5.2 From Büchi automata to ω-semigroups

Let $\mathcal{A} = (Q, A, E, I, F)$ be a Büchi automaton recognizing a subset X of A^ω. The idea is the following. Given a finite word u and two states p and q, we define a *multiplicity* expressing the following possibilities for the set P of paths from p to q labeled by u:

(1) P is empty,

(2) P is nonempty, but contains no path visiting a final state,

(3) P contains a path visiting a final state.

Our construction makes use of the semiring $\mathbb{K} = \{-\infty, 0, 1\}$ in which addition is the maximum for the ordering $-\infty < 0 < 1$ and multiplication, which extends the Boolean addition, is given in Table 1. Conditions (1), (2) and (3) will be encoded by $-\infty$, 0 and 1, respectively. Formally, we

	$-\infty$	0	1
$-\infty$	$-\infty$	$-\infty$	$-\infty$
0	$-\infty$	0	1
1	$-\infty$	1	1

Table 1. The multiplication table.

associate with each finite word u a $(Q \times Q)$-matrix $\mu(u)$ with entries in $\mathbb{K}$ defined by

$$\mu(u)_{p,q} = \begin{cases} -\infty & \text{in case (1),} \\ 0 & \text{in case (2),} \\ 1 & \text{in case (3)} \end{cases}$$

It is easy to see that μ is a morphism from A^+ into the multiplicative semigroup of $Q \times Q$-matrices with entries in $\mathbb{K}$. Let $S_+ = \mu(A^+)$.

The next step is to complete our structure of Wilke algebra by defining an appropriate set S_ω, an ω-power and a mixed product. The solution consists in coding infinite paths by column matrices of $\mathbb{K}^Q$, in such a way that each coefficient $\mu(u)_p$ codes the existence of an infinite path of label u starting at p.

The usual product of matrices induces a mixed product $\mathbb{K}^{Q\times Q} \times \mathbb{K}^Q \to \mathbb{K}^Q$. In order to define the operation ω on square matrices, we need the following definition. Given a matrix s of S_+, we call *infinite s-path starting at p* a sequence $p = p_0, p_1, \dots$ of states such that, for all i, $s_{p_i,p_{i+1}} \neq -\infty$. An s-path is said to be *successful* if $s_{p_i,p_{i+1}} = 1$ for an infinite number of indices i. We define the column matrix s^ω as follows. For every $p \in Q$,

$$s^\omega_p = \begin{cases} 1 & \text{if there exists a successful } s\text{-path of origin } p, \\ -\infty & \text{otherwise} \end{cases}$$

Note that the coefficients of this matrix can be effectively computed. Indeed, computing s_p^ω amounts to checking the existence of circuits containing a given edge in a finite graph.

Finally, S_ω is the set of all column matrices of the form st^ω, with $s, t \in S_+$. One can verify that $S = (S_+, S_\omega)$, equipped with these operations, is a Wilke algebra. Further, the morphism μ can be extended in a unique way as a morphism of ω-semigroups from A^∞ into S which recognizes the set $L^\omega(\mathcal{A})$.

Example 5.2. Let $\mathcal{A}$ be the Büchi automaton represented in Figure 3. The

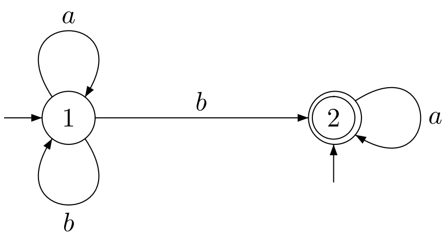


FIGURE 3. A Büchi automaton.

morphism $\mu : A^\infty \to S(\mathcal{A})$ is defined by the formula

$$\mu(a) = \begin{pmatrix} 0 & -\infty \\ -\infty & 1 \end{pmatrix} \quad \text{and} \quad \mu(b) = \begin{pmatrix} 0 & 1 \\ -\infty & -\infty \end{pmatrix}$$

The ω-semigroup generated by these matrices contains five elements:

$$a = \begin{pmatrix} 0 & -\infty \\ -\infty & 1 \end{pmatrix} \quad b = \begin{pmatrix} 0 & 1 \\ -\infty & -\infty \end{pmatrix} \quad a^\omega = \begin{pmatrix} -\infty \\ 1 \end{pmatrix} \quad b^\omega = \begin{pmatrix} -\infty \\ -\infty \end{pmatrix} \quad ba^\omega = \begin{pmatrix} 1 \\ -\infty \end{pmatrix}$$

and is presented by the relations:

$$a^2 = a \quad ab = b \quad ba = b \quad b^2 = b \quad aa^\omega = a^\omega \quad ab^\omega = b^\omega \quad bb^\omega = b^\omega$$

Example 5.3. Let $X = (a\{b,c\}^* \cup \{b\})^\omega$. A Büchi automaton recognizing X is represented in Figure 4. For this automaton, the previous computation

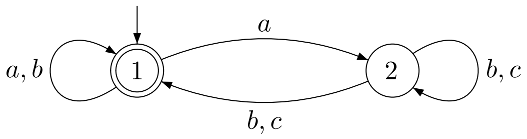


FIGURE 4. An automaton.

provides an ω-semigroup with nine elements

$$S = (\{a, b, c, ba, ca\}, \{a^\omega, b^\omega, c^\omega, (ca)^\omega\}),$$

where

$$a = \begin{pmatrix} 1 & 1 \\ -\infty & -\infty \end{pmatrix} \qquad b = \begin{pmatrix} 1 & -\infty \\ 1 & 0 \end{pmatrix} \qquad c = \begin{pmatrix} -\infty & -\infty \\ 1 & 0 \end{pmatrix} \qquad ba = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$
$$ca = \begin{pmatrix} -\infty & -\infty \\ 1 & 1 \end{pmatrix}$$
$$a^\omega = \begin{pmatrix} 1 \\ -\infty \end{pmatrix} \qquad b^\omega = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad c^\omega = \begin{pmatrix} -\infty \\ -\infty \end{pmatrix} \qquad (ca)^\omega = \begin{pmatrix} -\infty \\ 1 \end{pmatrix}$$

It is presented by the following relations:

$$\begin{array}{ccccc} a^2 = a & ab = a & ac = a & b^2 = b & bc = c \\ cb = c & c^2 = c & (ba)^\omega = b^\omega & aa^\omega = a^\omega & ab^\omega = a^\omega \\ ac^\omega = c^\omega & a^\omega = a(ca)^\omega & ba^\omega = b^\omega & bb^\omega = b^\omega & bc^\omega = c^\omega \\ b(ca)^\omega = (ca)^\omega & ca^\omega = (ca)^\omega & (ca)^\omega = cb^\omega & cc^\omega = c^\omega & c(ca)^\omega = (ca)^\omega \end{array}$$

Note that the syntactic ω-semigroup $S(X)$ of X is not equal to S. To compute $S(X)$, one should first compute the image of X in S, which is $P = \{a^\omega, b^\omega\}$. Next, one should compute the syntactic congruence $\sim_P$ of P in S, which is defined on S_+ by $u \sim_P v$ if and only if, for every $x, y, z \in S_+$

$$\begin{aligned} xuyz^\omega \in P &\iff xvyz^\omega \in P \\ x(uy)^\omega \in P &\iff x(vy)^\omega \in P \end{aligned} \tag{5.4}$$

and on S^ω by $u \sim_P v$ if and only if, for each $x \in S_+$,

$$xu \in P \iff xv \in P \tag{5.5}$$

Here we get $a \sim_P ba$ and $a^\omega \sim_P b^\omega$ and hence we recovered the semigroup

$$S(X) = (\{a, b, c, ca\}, \{a^\omega, c^\omega, (ca)^\omega\})$$

presented in Example 4.4.

5.3 From strong recognition to ω-semigroups

It is easy to associate a Wilke algebra $\bar{S} = (S, S_\omega)$ to a finite semigroup S.

Let π be the exponent of S, that is, the smallest integer n such that s^n is idempotent for every $s \in S$. Two linked pairs (s, e) and (s', e') of S are said to be *conjugate* if there exist $x, y \in S^1$ such that $e = xy$, $e' = yx$ and $s' = sx$. These equalities also imply $s = s'y$ (since $s'y = sxy = se = s$), showing the symmetry of the definition. One can verify that the conjugacy relation is an equivalence relation on the set of linked pairs of S. We shall denote by $[s, e]$ the conjugacy class of a linked pair (s, e).

We take for S_ω the set of conjugacy classes of the linked pairs of S. One can prove that the set $\bar{S}$ is equipped with a structure of Wilke algebra by setting, for each $[s, e] \in S_\omega$ and $t \in S$,

$$t[s, e] = [ts, e] \quad \text{and} \quad t^\omega = [t^\pi, t^\pi]$$

The definition is consistent since if (s', e') is conjugate to (s, e), then (ts', e') is conjugate to (ts, e). It is now easy to convert strong recognition to recognition by an ω-semigroup.

Proposition 5.4. If a set of infinite words is strongly recognized by a finite semigroup S, then it is recognized by the ω-semigroup $\bar{S}$.

5.4 From weak recognition to Muller automata

The construction given by Le Saec, Pin and Weil [29, 16] permits to convert a semigroup that weakly recognizes a set of infinite words into a transition Muller automaton. It relies, however, on two difficult results of finite semigroup theory. Recall that a semigroup is *idempotent* if all its elements are idempotent and *$\mathcal{R}$-trivial* if the condition $s \mathrel{\mathcal{R}} t$ implies $s = t$.

The first one is a cover theorem also proved in [29, 16]. Recall that the right stabilizer of an element s of a semigroup S is the set of all $t \in S$ such that $st = s$. These stabilizers are themselves semigroups, and reflect rather well the structure of S: if S is a group, every stabilizer is trivial, but if S is has a zero, the stabilizer of the zero is equal to S. Here we consider an intermediate case: the stabilizers are idempotent and $\mathcal{R}$-trivial, which amounts to saying that, for each $s, t, u \in S$, the condition $s = st = su$ implies $t^2 = t$ and $tut = tu$. We can now state the cover theorem precisely.

Theorem 5.5. Each finite semigroup is the quotient of a finite semigroup in which the right stabilizers satisfy the identities $x = x^2$ and $xyx = xy$.

The second result we need is a property of path congruences due to I. Simon. A proof of this property can be found in [14]. Given an automaton $\mathcal{A}$, a path congruence is an equivalence relation on the set of finite paths of $\mathcal{A}$ satisfying the following conditions:

(1) any two equivalent paths are coterminal (that is, they have the same origin and the same end),

(2) if p and q are equivalent paths, and if r, p and s are consecutive paths, then rps is equivalent to rqs.

Proposition 5.6 (I. Simon)**.** Let $\sim$ be a path congruence such that, for every pair of loops p, q around the same state, $p^2 \sim p$ and $pq \sim qp$. Then two coterminal paths visiting the same sets of transitions are equivalent.

We are now ready to present our algorithm. Let X be a recognizable subset of A^ω and let $\varphi : A^+ \to S$ be a morphism weakly recognizing X. By Theorem 5.5, we may assume that the stabilizers of S satisfy the identities $x^2 = x$ and $xyx = xy$. Let S^1 be the monoid equal to S if S is a monoid and to $S \cup \{1\}$ if S is not a monoid.

One naturally associates a deterministic automaton $(S^1, A, \cdot)$ to φ by setting, for every $s \in S^1$ and every $a \in A$

$$s \cdot a = s\varphi(a).$$

Let s be a fixed state of S^1. Then every word u is the label of exactly one path with origin s, called the path with origin s defined by u.

Let $\mathcal{A} = (S^1, A, \cdot, 1, \mathcal{T})$ be the transition Muller automaton with 1 as initial state and such that

$$\mathcal{T} = \{\mathrm{Inf}_T(u) \mid u \in X\}.$$

We claim that $\mathcal{A}$ recognizes X. First, if $u \in X$, then $\mathrm{Inf}_T(u) \in \mathcal{T}$ by definition, and thus u is recognized by $\mathcal{A}$. Conversely, let u be an infinite word recognized by $\mathcal{A}$. Then

$$\mathrm{Inf}_T(u) = \mathrm{Inf}_T(v) = T \quad \text{for some } v \in X.$$

Thus, both paths u and v visit only finitely many times transitions out of T. Therefore, after a certain point, every transition of u (resp. v) belongs to T, and every transition of T is visited infinitely often. Consequently, one can find two factorizations $u = u_0u_1u_2\cdots$ and $v = v_0v_1v_2\cdots$ and a state $s \in S$ such that

(1) u_0 and v_0 define paths from 1 to s,

(2) for every $n > 0$, u_n and v_n define loops around s that visit at least once every transition in T and visit no other transition.

The situation is summarized in Figure 5 below. Furthermore, Proposition

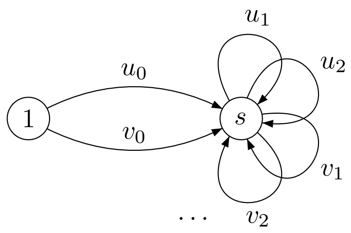


FIGURE 5.

3.2 shows that, by grouping the u_i's (resp. v_i's) together, we may assume that

$$\varphi(u_1) = \varphi(u_2) = \varphi(u_3) = \ldots \quad \text{and} \quad \varphi(v_1) = \varphi(v_2) = \varphi(v_3) = \ldots$$

It follows in particular

$$u_0 v_1^\omega \in X \tag{5.6}$$

since $\varphi(u_0) = \varphi(v_0) = s$, $\varphi(v_1) = \varphi(v_2) = \ldots$ and $v_0v_1v_2 \cdots \in X$. Furthermore,

$$u \in X \quad \text{if and only if} \quad u_0u_1^\omega \in X \tag{5.7}$$

To simplify notation, we shall denote by the same letter a path and its label. We define a path equivalence $\sim$ as follows. Two paths p and q are equivalent if p and q are coterminal, and if, for every nonempty path x from 1 to the origin of p, and for every path r from the end of p to its origin, $x(pr)^\omega \in X$ if and only if $x(qr)^\omega \in X$.

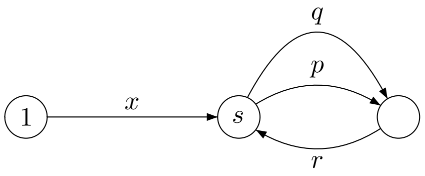


Figure 6.

Lemma 5.7. The equivalence $\sim$ is a path congruence such that, for every pair of loops p, q around the same state, $p^2 \sim p$ and $pq \sim qp$.

Proof. We first verify that $\sim$ is a congruence. Suppose that $p \sim q$ and let u and v be paths such that u, p and v are consecutive. Since $p \sim q$, p and q are coterminal, and thus upv and uqv are also coterminal. Furthermore, if x is a nonempty path from 1 to the origin of upv, and if r is a path from the end of upv to its origin such that $x(upvr)^\omega \in X$, then $(xu)(p(vru))^\omega \in X$, whence $(xu)(q(vru))^\omega \in X$ since $p \sim q$, and thus $x(uqvr)^\omega \in X$. Symmetrically, $x(uqvr)^\omega \in X$ implies $x(upvr)^\omega \in X$, showing that $upv \sim uqv$.

Next we show that if p is a loop around $s \in S$, then $p^2 \sim p$. Let x be a nonempty path from 1 to the origin of p, and let r be a path from the end of p to its origin. Then, since p is a loop, $\varphi(x)\varphi(p) = \varphi(x)$. Now since the stabilisers of S are idempotent semigroups, $\varphi(p) = \varphi(p^2)$ and thus $x(pr)^\omega \in X$ if and only if $x(p^2r)^\omega \in X$ since φ recognizes X.

Finally, we show that if p and q are loops around the same state s, then $pq \sim qp$. Let, as before, x be a nonempty path from 1 to the origin of p, and let r be a path from the end of p to its origin. Then r is a loop around s. We first observe that

$$x(pq)^\omega \in X \Longleftrightarrow x(qp)^\omega \in X \tag{5.8}$$

Indeed $x(pq)^\omega = xp(qp)^\omega$, and since p is a loop, $\varphi(x)\varphi(p) = \varphi(x)$. Thus $xp(qp)^\omega \in X$ if and only if $x(qp)^\omega \in X$, then proving (5.8). Now, we have the following sequence of equivalences

$$\begin{aligned} x(pqr)^\omega \in X &\Longleftrightarrow x(pqrq)^\omega \in X \Longleftrightarrow x(rqpq)^\omega \in X \\ &\Longleftrightarrow x(rqp)^\omega \in X \Longleftrightarrow x(qpr)^\omega \in X, \end{aligned}$$

where the second and fourth equivalences follow from (5.8) and the first and third from the identity $xyx = xy$ satisfied by the right stabilizer of $\varphi(x)$. Q.E.D.

We can now conclude the proof of Theorem 2.5. By assumption, the two loops around s defined by u_1 and v_1 visit exactly the same sets of transitions (namely T). Thus, by Lemma 5.7 and by Proposition 5.6, these two paths are equivalent. In particular, since $u_0 v_1^\omega \in X$ by (5.6), we have $u_0 u_1^\omega \in X$, and thus $u \in X$ by (5.7). Therefore $\mathcal{A}$ recognizes X.

6 An algebraic proof of McNaughton's theorem

McNaughton's celebrated theorem states that any recognizable subset of infinite words is a Boolean combination of deterministic recognizable sets. This Boolean combination can be explicitly computed using ω-semigroups. This proof relies on a few useful formulas of independent interest on deterministic sets. Note that McNaughton's theorem can be formulated as the equivalence of Büchi and Muller automata. Thus the construction described in Section 5.4 gives an alternative proof of McNaughton's theorem. Yet another proof is due to Safra [30]. It provides a direct construction leading to a reduced computational complexity.

Let S be a finite ω-semigroup and let $\varphi : A^\infty \to S$ be a surjective morphism recognizing a subset X of A^ω. Set, for each $s \in S_+$, $X_s = \varphi^{-1}(s)$. Finally, we denote by P the image of X in S and by $F(P)$ the set of linked pairs (s, e) of S_+ such that $se^\omega \in P$.

For each $s \in S_+$, the set $P_s = X_s \setminus X_s A^+$ is prefix-free, since a word of P_s cannot be, by definition, prefix of another word of P_s. Put

$$E_s = \{f \in S_+ \mid f^2 \text{ and } sf = s\} = \{f \in S_+ \mid (s, f) \text{ is a linked pair}\},$$

and denote by $\leqslant$ the relation on E_s defined by

$$g \leqslant e \text{ if and only if } eg = g.$$

It is the restriction to the set E_s of the preorder $\leqslant_{\mathcal{R}}$, since, if $g = ex$ then $eg = eex = ex = g$. We shall use the notation $e < g$ if $e \leqslant g$ and if $g \nleqslant e$. To simplify notation, we shall suppose implicitly that for every expression of the form $X_s X_f^\omega$ or $X_s P_f$, the pair (s, f) is a linked pair of S_+.

Proposition 6.1. For each linked pair (s, e) of S_+, the following formula holds

$$X_s X_e^\omega \subset \overrightarrow{X_s P_e} \subset \bigcup_{f \leqslant e} X_s X_f^\omega. \tag{6.9}$$

Corollary 6.2.

(1) For every idempotent e of S_+, the following formula holds

$$X_e^\omega = \overrightarrow{X_e P_e}. \tag{6.10}$$

(2) For every linked pair (s, e) of S_+, we have

$$\bigcup_{f \leqslant e} X_s X_f^\omega = \bigcup_{f \leqslant e} \overrightarrow{X_s P_f}. \tag{6.11}$$

Proof. Formula (6.10) is obtained by applying (6.9) with $s = e$. Formula (6.11) follows by taking the union of both sides of (6.9) for $f \leqslant e$. Q.E.D.

The previous statement shows that a set of the form X_e^ω, with e idempotent, is always deterministic. This may lead the reader to the conjecture that every subset of the form X^ω, where X is a recognizable subset of A^+, is deterministic. However, this conjecture is ruined by the next example.

Example 6.3. Let $X = (a\{b, c\}^* \cup \{b\})^\omega$. The syntactic ω-semigroup of Y has been computed in Example 4.4. In this ω-semigroup, b is the identity, and all the elements are idempotent. The set X can be split into simple elements as follows:

$$\begin{aligned} X &= \varphi^{-1}(a)\varphi^{-1}(b)^\omega \cup \varphi^{-1}(a)^\omega \\ &= b^* a\{a, b, c\}^* b^\omega \cup (b^* a\{a, b, c\}^*)^\omega. \end{aligned}$$

It is possible to deduce from the previous formulas an explicit Boolean combination of deterministic sets.

Theorem 6.4. The following formula holds

$$X = \bigcup_{(s,e) \in F(P)} \ \bigcup_{f \mathcal{R} e} X_s X_f^\omega \tag{6.12}$$

and, for each $(s, e) \in F(P)$,

$$\bigcup_{f \mathcal{R} e} X_s X_f^\omega = (\overrightarrow{U}_{s,e} \setminus \overrightarrow{V}_{s,e}) \tag{6.13}$$

where $U_{s,e}$ and $V_{s,e}$ are the subsets of A^+ defined by:

$$U_{s,e} = \bigcup_{f \leqslant e} X_s P_f \quad \text{and} \quad V_{s,e} = \bigcup_{f < e} X_s P_f$$

In particular, X is a Boolean combination of deterministic sets.

For a proof, see [25, p. 120]. One can also obtain a characterization of the deterministic subsets.

Theorem 6.5. The set X is deterministic if and only if, for each linked pairs (s, e) and (s, f) of S_+ such that $f \leqslant e$, the condition $se^\omega \in P$ implies $sf^\omega \in P$. In this case

$$X = \bigcup_{(s,e) \in F(P)} \overrightarrow{X_s P_e} \tag{6.14}$$

For a proof, see [25, Theorem 9.4, p. 121].

Example 6.6. We return to Example 6.3. The image of X in its syntactic ω-semigroup is the set $P = \{a^\omega\}$. Now, the pairs (a, b) and (a, c) are linked pairs of S_+ since $ab = ac = a$ and we have $c \leqslant b$ since $bc = c$. But $ab^\omega = a^\omega \in P$, and $ac^\omega = c^\omega \notin P$. Therefore X is not deterministic.

The proof of McNaughton's theorem described above is due to Schützenberger [31]. It is related to the proof given by Rabin [27] and improved by Choueka [12]. See [25, p. 72] for more details.

7 Prophetic automata

In this section, we introduce a new type of automata, called prophetic, because in some sense, all the information concerning the future is encoded in the initial state. We first need to make precise a few notions on Büchi automata.

7.1 More on Büchi automata

There are two competing versions for the notions of determinism and co-determinism for a trim automaton. In the first version, the notions are purely local and are defined by a property of the transitions set. They give rise to the notions of automaton with deterministic or co-deterministic transitions introduced in Section 2. The second version is global: a trim automaton is *deterministic* if it has exactly one initial state and if every word is the label of at most one initial path. Similarly, a trim automaton is *co-deterministic* if every word is the label of at most one final path.

The local and global notions of determinism are equivalent. The local and global notions of co-determinism are also equivalent for finite words. However, for infinite words, the global version is strictly stronger than the local one.

Lemma 7.1. A trim Büchi automata is deterministic if and only if it has exactly one initial state and if its transitions are deterministic. Further, if a trim Büchi automata is co-deterministic, then its transitions are co-deterministic.

The notions of complete and co-complete Büchi automata are also global notions. A trim Büchi automata is *complete* if every word is the label of at least one initial path. It is *co-complete* if every word is the label of at least one final path. Unambiguity is another global notion. A Büchi

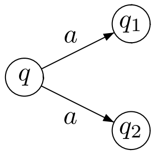
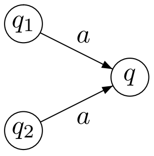
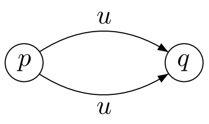
Det. transitions	**Co-det. transitions**	**Unambiguous**
Forbidden configuration:  where a is a letter.	Forbidden configuration:  where a is a letter.	Forbidden configuration:  where u is a word.
Deterministic	**Co-deterministic**	**Unambiguous**
Two initial paths with the same label are equal + exactly one initial state	Two final paths with the same label are equal	Two successful paths with the same label are equal
Complete	**Co-complete**	
Every word is the label of some initial path	Every word is the label of some final path	

Table 2. Summary of the definitions.

automaton $\mathcal{A}$ is said to be *ω-unambiguous* if every infinite word in is the label of at most one successful path. It is clear that any deterministic or co-deterministic Büchi automaton is ω-unambiguous, but the converse is not true. The various terms are summarized in Table 2.

7.2 Prophetic automata

By definition, a *prophetic automaton* is a co-deterministic, co-complete Büchi automaton. Equivalently, a Büchi automaton is prophetic if every word is the label of exactly one final path. Therefore, a word is accepted if the unique final path it defines is also initial. The main result of this section shows that prophetic and Büchi automata are equivalent.

Theorem 7.2. Any recognizable set of infinite words can be recognized by a prophetic automaton.

It was already proved independently in [19] and [2] that any recognizable set of infinite words is recognized by a codeterministic automaton, but the construction given in [2] does not provide unambiguous automata.

Prophetic automata recognize infinite words, but the construction can be adapted to biinfinite words. Two unambiguous automata on infinite words can be merged to make an unambiguous automaton on biinfinite words. This leads to an extension of McNaughton's theorem to the case of biinfinite words. See [25, Section 9.5] for more details.

Theorem 7.2 was originally formulated by Michel in the eighties but remained unpublished for a long time. Another proof was found by the first author and the two proofs were finally published in [10, 11]. Our presentation follows the proof which is based on ω-semigroups. We start with a simple characterization.

Proposition 7.3. Let $\mathcal{A} = (Q, A, E, I, F)$ be a Büchi (transition Büchi) automaton and let, for each $q \in Q$, $L_q = L^\omega(Q, A, E, q, F)$.

(1) $\mathcal{A}$ is co-deterministic if and only if the L_q's are pairwise disjoint.

(2) $\mathcal{A}$ is co-complete if and only if $\cup_{q\in Q} L_q = A^\omega$.

Proof. (1) If $\mathcal{A}$ is co-deterministic, the L_q's are clearly pairwise disjoint. Suppose that the L_q's are pairwise disjoint and let $p_0 \xrightarrow{a_0} p_1 \xrightarrow{a_1} p_2 \;\cdots$ and $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \;\cdots$ be two infinite paths with the same label $u = a_0a_1\cdots$. Then, for each $i \geqslant 0$, $a_ia_{i+1}\cdots \in L(p_i) \cap L(q_i)$, and thus $p_i = q_i$. Thus $\mathcal{A}$ is co-deterministic.

(2) follows immediately from the definition of co-complete automata.

Q.E.D.

Example 7.4. A prophetic automaton is presented in Figure 7. The corresponding partition of A^ω is the following:

$L_0 = A^*ba^\omega$	(at least one, but finitely many b)
$L_1 = a^\omega$	(no b)
$L_2 = a(A^*b)^\omega$	(first letter a, infinitely many b)
$L_3 = b(A^*b)^\omega$	(first letter b, infinitely many b)

Example 7.5. Another example, recognizing the set $A^*(ab)^\omega$, is presented in Figure 8.

Complementation becomes easy with prophetic automata.

Proposition 7.6. Let $\mathcal{A} = (Q, A, E, I, F)$ be a prophetic automaton recognizing a subset X of A^ω. Then the Büchi automaton $(Q, A, E, Q \setminus I, F)$ recognizes the complement of X.

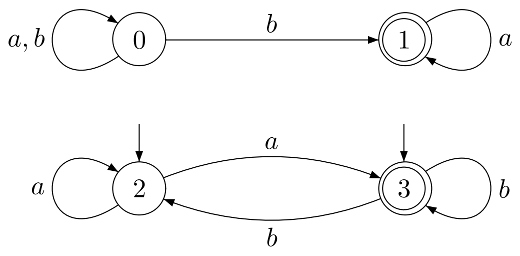


FIGURE 7. A prophetic automaton.

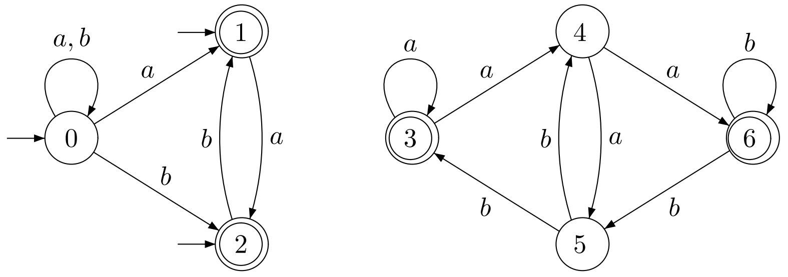


FIGURE 8. A prophetic automaton recognizing $A^*(ab)^\omega$.

It is easier to prove Theorem 7.2 for a variant of prophetic automata that we now define. A *prophetic transition automaton* is a co-deterministic, co-complete, transition automaton. Proposition 2.6 states that Büchi automata and transition Büchi automata are equivalent. It is not difficult to adapt this result to prophetic automata [25, Proposition I.8.1].

Proposition 7.7. Prophetic and transition prophetic automata are equivalent.

Thus Theorem 7.2 can be reformulated as follows.

Theorem 7.8. Any recognizable set of infinite words can be recognized by a prophetic transition automaton.

Proof. Let X be a recognizable subset of A^ω, let $\varphi : A^\infty \to S$ be the syntactic morphism of X and let $P = \varphi(X)$. Our construction strongly relies on the properties of $\geqslant_{\mathcal{R}}$-chains of the semigroup S_+ and requires a few preliminaries.

We shall denote by R the set of all nonempty $>_{\mathcal{R}}$-chains of S_+:

$$R = \{(s_0, s_1, \dots, s_n) \mid n \geqslant 0,\ s_0, \dots, s_n \in S \text{ and } s_0 >_{\mathcal{R}} s_1 >_{\mathcal{R}} \cdots >_{\mathcal{R}} s_n\}$$

In order to convert a $\geqslant_{\mathcal{R}}$-chain into a strict $>_{\mathcal{R}}$-chain, we introduce the reduction ρ, defined inductively as follows

$$\rho(s) = (s)$$
$$\rho(s_1, \ldots, s_n) = \begin{cases} \rho(s_1, \ldots, s_{n-1}) & \text{if } s_n \mathrel{\mathcal{R}} s_{n-1} \\ (\rho(s_1, \ldots, s_{n-1}), s_n) & \text{if } s_{n-1} >_{\mathcal{R}} s_n \end{cases}$$

In particular, for each finite word $u = a_0 a_1 \cdots a_n$ (where the a_i's are letters), let $\hat{\varphi}(u)$ be the $>_{\mathcal{R}}$-chain $\rho(s_0, s_1, \ldots, s_n)$, where $s_i = \varphi(a_0 a_1 \cdots a_i)$ for $0 \leqslant i \leqslant n$. The definition of $\hat{\varphi}$ can be extended to infinite words. Indeed, if $u = a_0 a_1 \cdots$ is an infinite word,

$$s_0 \geqslant_{\mathcal{R}} s_1 \geqslant_{\mathcal{R}} s_2 \cdots$$

and since S_+ is finite, there exists an integer n, such that, for all $i, j \geqslant n$, $s_i \mathrel{\mathcal{R}} s_j$. Then we set $\hat{\varphi}(u) = \hat{\varphi}(a_0 \ldots a_n)$.

Define a map from $A \times S_+^1$ into S_+^1 by setting, for each $a \in A$ and $s \in S_+^1$,

$$a \cdot s = \varphi(a) s$$

We extend this map to a map from $A \times R$ into R by setting, for each $a \in A$ and $(s_1, \ldots, s_n) \in R$,

$$a \cdot (s_1, \ldots, s_n) = \rho(a \cdot 1, a \cdot s_1, \ldots, a \cdot s_n)$$

To extend this map to A^+, it suffices to apply the following induction rule, where $u \in A^+$ and $a \in A$

$$(ua) \cdot (s_1, \ldots, s_n) = u \cdot (a \cdot (s_1, \ldots, s_n))$$

This defines an action of the semigroup A^+ on the set R in the sense that, for all $u, v \in A^*$ and $r \in R$,

$$(uv) \cdot r = u(v \cdot r)$$

The connections between this action, φ and $\hat{\varphi}$ are summarized in the next lemma.

Lemma 7.9. The following formulas hold:

(1) For each $u \in A^+$ and $v \in A^\omega$, $u \cdot \varphi(v) = \varphi(uv)$

(2) For each $u, v \in A^+$, $u \cdot \hat{\varphi}(v) = \hat{\varphi}(uv)$

Proof. (1) follows directly from the definition of the action and it suffices to establish (2) when u reduces to a single letter a. Let $v = a_0 a_1 \ldots a_n$, where the a_i's are letters and let, for $0 \leqslant i \leqslant n$, $s_i = \varphi(a_0 a_1 \ldots a_i)$. Then, by definition, $\hat{\varphi}(v) = \rho(s_0, \ldots, s_n)$ and since, the relation $\geqslant_{\mathcal{R}}$ is stable on the left,

$$a\cdot \hat{\varphi}(v) = \rho(a\cdot 1, a\cdot s_0, a\cdot s_1, \ldots, a\cdot s_n) = \hat{\varphi}(av)$$

which gives (2). Q.E.D. (Theorem 7.8)

We now define a transition Büchi automaton $\mathcal{A} = (Q, A, E, I, F)$ by setting

$$\begin{aligned} Q &= \Big\{ \big((s_1, \ldots, s_n), se^\omega\big) \mid (s_1, \ldots, s_n) \in R, \\ &\qquad\qquad (s, e) \text{ is a linked pair of } S_+ \text{ and } s_n \mathrel{\mathcal{R}} s \Big\} \\ I &= \Big\{ \big((s_1, \ldots, s_n), se^\omega\big) \in Q \mid se^\omega \in P \Big\} \\ E &= \Big\{ \Big(\big(a\cdot (s_1, \ldots, s_n), a\cdot se^\omega\big), a, \big((s_1, \ldots, s_n), se^\omega\big) \Big) \\ &\qquad\qquad \mid a \in A \text{ and } \big((s_1, \ldots, s_n), se^\omega\big) \in Q \Big\} \end{aligned}$$

A transition $\Big(\big(a\cdot (s_1, \ldots, s_n), a\cdot se^\omega\big), a, \big((s_1, \ldots, s_n), se^\omega\big) \Big)$ is said to be *cutting* if the last two elements of the $\geqslant_{\mathcal{R}}$-chain $(a\cdot 1, a\cdot s_1, \ldots, a\cdot s_n)$ are $\mathcal{R}$-equivalent.
We choose for F the set of cutting transitions of the form

$$\Big(\big(a\cdot (s_1, \ldots, s_n), a\cdot e^\omega\big), a, \big((s_1, \ldots, s_n), e^\omega\big) \Big)$$

where e is an idempotent of S_+ such that $s_n \mathrel{\mathcal{R}} e$.

Note that $\mathcal{A}$ has co-deterministic transitions. A typical transition is shown in Figure 9. The first part of the proof consists in proving that every

$$\begin{tikzcd} \big(a\cdot (s_1, \ldots, s_n), a\cdot se^\omega\big) \arrow[r, "a"] & \big((s_1, \ldots, s_n), se^\omega\big) \end{tikzcd}$$

FIGURE 9. A transition of $\mathcal{A}$.

infinite word is the label of a final path. Let $u = a_0a_1 \cdots$ be an infinite word, and let, for each $i \geqslant 0$, $x_i = a_ia_{i+1} \cdots$ and $q_i = \big(\hat{\varphi}(x_i), \varphi(x_i)\big)$. Each q_i is a state of Q, and Lemma 7.9 shows that

$$p = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \;\cdots$$

is a path of $\mathcal{A}$.

Lemma 7.10. The path p is final.

Proof. Let $(u_i)_{i\geqslant 0}$ be a factorization of u associated with the linked pair (s,e). Then for each $i>0$, $\varphi(u_iu_{i+1}\cdots)=e^\omega$. Fix some $i>0$ and let $n_i=|u_0u_1\cdots u_i|$. Then $q_{n_i}=\big((s_1,\ldots,s_n),e^\omega\big)$ with $(s_1,\ldots,s_n)=\hat\varphi(u_{i+1}u_{i+2}\cdots)$. In particular, $s_n\ \mathcal{R}\ e$ and hence $es_n=s_n$. Suppose first that $n\geqslant 2$. Then $\varphi(u_i)s_{n-1}=es_{n-1}\leqslant_{\mathcal{R}} e$ and $\varphi(u_i)s_n=es_n=s_n\ \mathcal{R}\ e$. Therefore the relation $\varphi(u_i)s_{n-1}>_{\mathcal{R}}\varphi(u_i)s_n$ does not hold. If $n=1$, the same argument works by replacing s_{n-1} by 1. It follows that in the path of label u_i from $q_{n_{i-1}}$ to q_{n_i}, at least one of the transitions is cutting. Thus p contains infinitely many cutting transitions and one can select one, say (q,a,q'), that occurs infinitely often. This gives a factorization of the form

$$p=q_0\xrightarrow{x_0}q\xrightarrow{a}q'\xrightarrow{x_1}q\xrightarrow{a}q'\xrightarrow{x_2}\cdots$$

Up to taking a superfactorization, we can assume, by Proposition 3.2, that for some idempotent f, $\varphi(x_ia)=f$ for every $i>0$. It follows that the second component of q' is $\varphi(x_iax_{i+1}a\cdots)=f^\omega$ and thus the transition (q,a,q') is final, which proves the lemma. Q.E.D. (Theorem 7.8)

Furthermore, p is successful if and only if $\varphi(u)\in P$, or, equivalently, if $u\in X$. Thus $\mathcal{A}$ recognizes X and is co-complete. It just remains to prove that $\mathcal{A}$ is co-deterministic, which, by Proposition 7.3, will be a consequence of the following lemma.

Lemma 7.11. Any final path of label u starts at state $\big(\hat\varphi(u),\varphi(u)\big)$.

Proof. Let p be a final path of label u. Then some final transition, say (q,a,q'), occurs infinitely often in p. Highlighting this transition yields a factorization of p

$$q_0\xrightarrow{v_0}q\xrightarrow{ⓐ}q'\xrightarrow{v_1}q\xrightarrow{ⓐ}q'\xrightarrow{v_2}\cdots$$

Let $q'=\big((s_1,\ldots,s_n),e^\omega\big)$, and consider a factor of the path p labelled by a word of the form $v=v_iav_{i+1}a\cdots v_ja$, with $i>0$ and $j-i\geqslant n$. By the choice of v, $q'=v\cdot q'$, and the first component of q' is obtained by reducing the $\geqslant_{\mathcal{R}}$-chain

$$\big(\varphi(v[0,0]),\varphi(v[0,1]),\ldots,\varphi(v),\varphi(v)s_1,\ldots,\varphi(v)s_n\big)$$

Now, since the cutting transition (q,a,q') occurs $n+1$ times in this factor, the last $n+1$ elements of this chain are $\mathcal{R}$-equivalent. It follows that the first component of q' is simply equal to $\hat\varphi(v)$.

Consider now a superfactorization $u=w_0w_1w_2\cdots$ obtained by grouping the factors v_ia

$$u=\underbrace{(v_0a\cdots v_{i_0-1}a)}_{w_0}\underbrace{(v_{i_0}a\cdots v_{i_1-1}a)}_{w_1}\underbrace{(v_{i_1}a\cdots v_{i_2-1}a)}_{w_2}$$

in such a way that, for some idempotent f, $\varphi(w_1) = \varphi(w_2) = \cdots = f$. We may also assume that $i_0 > 0$ and $i_1 - i_0 \geqslant n + 1$. Thus $q' = w_1 \cdot q' = w_1 w_2 \cdot q' = \cdots$, and

$$(s_1, \cdots, s_n) = \hat{\varphi}(w_1) = \hat{\varphi}(w_1 w_2) = \cdots = \hat{\varphi}(w_1 w_2 \cdots)$$

It follows in particular $s_n \mathrel{\mathcal{R}} \varphi(w_1) = f$. Furthermore, $s_n \mathrel{\mathcal{R}} e$ since (q, a, q') is a final transition and thus $e \mathrel{\mathcal{R}} f$. Therefore $e^\omega = f^\omega = \varphi(w_1 w_2 \cdots)$. Thus $q' = \big(\hat{\varphi}(w_1 w_2 \cdots), \varphi(w_1 w_2 \cdots)\big)$ and it follows from Lemma 7.9 that $q_0 = w_0 \cdot q' = \big(\hat{\varphi}(u), \varphi(u)\big)$. Q.E.D. (Lemma 7.11)

Q.E.D. (Theorem 7.8)

The construction given in the proof of Theorem 7.2 is illustrated in the following examples.

Example 7.12. Let $A = \{a, b\}$ and let $X = aA^\omega$. The syntactic ω-semigroup S of X, already computed in Example 4.2 is $S = (S_+, S_\infty)$ where $S_+ = \{a, b\}$, $S_\omega = \{a^\omega, b^\omega\}$, submitted to the following relations

$$\begin{array}{cccc} aa = a & ab = a & aa^\omega = a^\omega & ab^\omega = a^\omega \\ ba = b & bb = b & ba^\omega = b^\omega & bb^\omega = b^\omega \end{array}$$

The syntactic morphism φ of X is defined by $\varphi(a) = a$ and $\varphi(b) = b$. The transition Büchi automaton associated with φ is shown in Figure 10. The final transitions are circled.

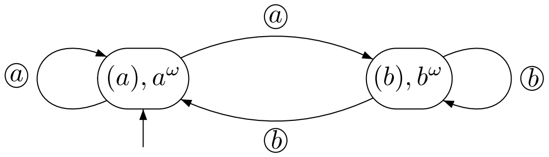


FIGURE 10. The transition Büchi automaton associated with φ.

Example 7.13. Let $A = \{a, b\}$ and let $X = (A^*a)^\omega$. The syntactic ω-semigroup S of X is $S = (S_+, S_\infty)$ where $S_+ = \{0, 1\}$, $S_\omega = \{0^\omega, 1^\omega\}$, submitted to the following relations

$$\begin{array}{cccc} 1 \cdot 1 = 1 & 1 \cdot 0 = 0 & 10^\omega = 0^\omega & 11^\omega = 1^\omega \\ 0 \cdot 1 = 0 & 0 \cdot 0 = 0 & 00^\omega = 0^\omega & 01^\omega = 1^\omega \end{array}$$

The syntactic morphism φ of X is defined by $\varphi(a) = 0$ and $\varphi(b) = 1$. The transition Büchi automaton associated with φ is shown in Figure 11.

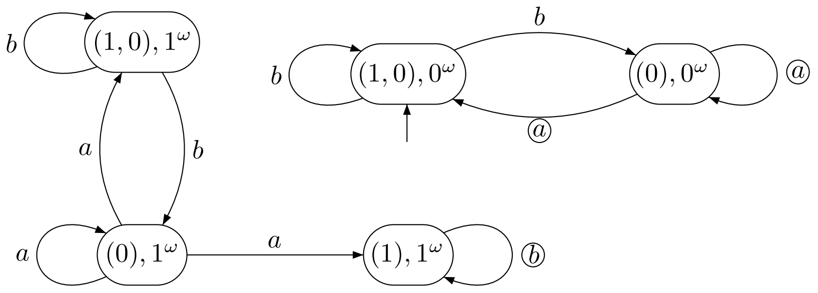


FIGURE 11. The transition Büchi automaton associated with φ.

7.3 Transfinite words

A natural extension to finite and infinite words is to consider words indexed by an ordinal, also called *transfinite word*. Automata on ordinals were introduced by Büchi [8, 9]. This leads to the notion of recognizable set of transfinite words. Subsequent work [3, 4, 5, 12, 13, 40] has shown that a number of results on infinite words can be extended to transfinite words (and even to words on linear orders [6, 28]).

An extension of the notion of ω-semigroup to countable ordinals was given in [3, 4, 5]. A further extension to countable linear orders is given in [6].

It is not difficult to extend the notion of prophetic automata to transfinite words. We show however that prophetic automata do not accept all recognizable sets of transfinite words.

First recall that an automaton on transfinite words is given by a finite set Q of states, sets I and F of initial and final states and a set E of transitions. Each transition is either a triple (p, a, q) where p and q are states and a is a letter or a pair (q, P) where where q is a state and P a subset of states. The former ones are called *successor transitions* and the latter ones *limit transitions*.

Let α be an ordinal. A *path* labeled by a word $x = (a_\beta)_{\beta<\alpha}$ of length α is a sequence $c = (q_\beta)_{\beta\leqslant\alpha}$ of states of length $\alpha + 1$ with the following properties.

(1) for each $\beta < \alpha$, the triple $(q_\beta, a_\beta, q_{\beta+1})$ is a successor transition of $\mathcal{A}$.

(2) for each limit ordinal $\beta \leqslant \alpha$, the pair $(\lim_\beta(c), c_\beta)$ is a limit transition of $\mathcal{A}$, where $\lim_\beta(c)$ is the set of states q such that, for each ordinal $\gamma < \beta$, there is an ordinal η such that $\gamma < \eta < \beta$ and $q = q_\eta$.

Note that since Q is finite, the set $\lim_\beta(c)$ is nonempty for each limit ordinal

$\beta \leqslant \alpha$. A path $c = (q_\beta)_{\beta \leqslant \alpha}$ is initial if its first state q_0 is initial and it is final if its last state q_α is final. It is *accepting* if it is both initial and final. A word x is accepted if it is the label of an accepting path.

The notion of prophetic automaton can be readily adapted to transfinite words: an automaton is *prophetic* if any transfinite word is the label of exactly one final path. However, the next result shows that not every automaton is equivalent to a prophetic one.

Proposition 7.14. The set A^{ω^2} of words of length ω^2 cannot be accepted by a prophetic automaton.

Proof. Suppose there is a prophetic automaton $\mathcal{A}$ accepting the set A^{ω^2}. Since the word a^{ω^2} is accepted by $\mathcal{A}$, there is a unique successful path $c = (q_\beta)_{\beta \leqslant \omega^2}$ labeled by a^{ω^2}. In particular, q_0 is an initial state and q_{ω^2} is a final state. We claim that the word a^ω is also accepted by $\mathcal{A}$.

We first prove that $q_\beta = q_0$ for any $\beta < \omega^2$. The path $(q_\beta)_{1 \leqslant \beta \leqslant \omega^2}$ is also a final path labeled by a^{ω^2}. It must therefore be equal to c. This shows that $q_n = q_0$ for any $n < \omega$. Similarly, the path $(q_\beta)_{\omega \leqslant \beta \leqslant \omega^2}$ is a final path labeled by a^{ω^2} and hence $q_\beta = q$ for any $\beta < \omega^2$. Since the set $\lim_{\omega^2}(c)$ is equal to $\{q_0\}$, the pair $(\{q_0\}, q_{\omega_2})$ must be a limit transition of $\mathcal{A}$. Thus the path $c' = (q'_\beta)_{\beta < \omega}$ defined by $q'_\beta = q_0$ if $\beta < \omega$ and $q'_\omega = q_{\omega^2}$ is a successful path labeled by a^ω. Q.E.D.

References

[1] A. Arnold. A syntactic congruence for rational ω-languages. *Theor. Comput. Sci.*, 39:333–335, 1985.

[2] D. Beauquier and D. Perrin. Codeterministic automata on infinite words. *Inf. Process. Lett.*, 20(2):95–98, 1985.

[3] N. Bedon. Finite automata and ordinals. *Theor. Comput. Sci.*, 156(1&2):119–144, 1996.

[4] N. Bedon. Automata, semigroups and recognizability of words on ordinals. *Internat. J. Algebra Comput.*, 8(1):1–21, 1998.

[5] N. Bedon and O. Carton. An Eilenberg theorem for words on countable ordinals. In Lucchesi and Moura [17], pages 53–64.

[6] V. Bruyère and O. Carton. Automata on linear orderings. In J. Sgall, A. Pultr, and P. Kolman, editors, *MFCS*, volume 2136 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2001.

[7] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr.)*, pages 1–11. Stanford Univ. Press, Stanford, Calif., 1962.

[8] J. R. Büchi. Transfinite automata recursions and weak second order theory of ordinals. In *Logic, Methodology and Philos. Sci. (Proc. 1964 Internat. Congr.)*, pages 3–23. North-Holland, Amsterdam, 1965.

[9] J. R. Büchi. The monadic second order theory of ω_1. In *The monadic second order theory of all countable ordinals (Decidable theories, II)*, pages 1–127. Lecture Notes in Math., Vol. 328, Berlin, 1973. Springer.

[10] O. Carton and M. Michel. Unambiguous Büchi automata. In G. H. Gonnet, D. Panario, and A. Viola, editors, *LATIN*, volume 1776 of *Lecture Notes in Computer Science*, pages 407–416. Springer, 2000.

[11] O. Carton and M. Michel. Unambiguous Büchi automata. *Theor. Comput. Sci.*, 1-3(297):37–81, 2003.

[12] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *J. Comput. Syst. Sci.*, 8(2):117–141, 1974.

[13] Y. Choueka. Finite automata, definable sets, and regular expressions over ω^n-tapes. *J. Comput. Syst. Sci.*, 17(1):81–97, 1978.

[14] S. Eilenberg. *Automata, Languages, and Machines. Vol. B.* Academic Press [Harcourt Brace Jovanovich Publishers], New York, 1976. With two chapters (“Depth decomposition theorem” and “Complexity of semigroups and morphisms”) by Bret Tilson, Pure and Applied Mathematics, Vol. 59.

[15] E. Grädel, W. Thomas, and Th. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[16] B. Le Saëc, J.-E. Pin, and P. Weil. Semigroups with idempotent stabilizers and applications to automata theory. *Internat. J. Algebra Comput.*, 1(3):291–314, 1991.

[17] C. L. Lucchesi and A. V. Moura, editors. *LATIN ’98: Theoretical Informatics, Third Latin American Symposium, Campinas, Brazil, April, 20-24, 1998, Proceedings*, volume 1380 of *Lecture Notes in Computer Science*. Springer, 1998.

[18] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966.

[19] A. W. Mostowski. Determinancy of sinking automata on infinite trees and inequalities between various Rabin's pair indices. *Inf. Process. Lett.*, 15(4):159–163, 1982.

[20] J.-P. Pécuchet. Etude syntaxique des parties reconnaissables de mots infinis. In L. Kott, editor, *ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 294–303. Springer, 1986.

[21] J.-P. Pécuchet. Variétés de semigroupes et mots infinis. In B. Monien and G. Vidal-Naquet, editors, *STACS*, volume 210 of *Lecture Notes in Computer Science*, pages 180–191. Springer, 1986.

[22] D. Perrin. Variétés de semigroupes et mots infinis. *C. R. Acad. Sci. Paris Sér. I Math.*, 295(10):595–598, 1982.

[23] D. Perrin. An introduction to finite automata on infinite words. In M. Nivat and D. Perrin, editors, *Automata on Infinite Words*, volume 192 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 1984.

[24] D. Perrin and J.-E. Pin. Semigroups and automata on infinite words. In *Semigroups, formal languages and groups (York, 1993)*, volume 466 of *NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci.*, pages 49–72. Kluwer Acad. Publ., Dordrecht, 1995.

[25] D. Perrin and J.-E. Pin. *Infinite Words. Automata, Semigroups, Logic and Games*, volume 141 of *Pure and Applied Mathematics.* Elsevier, Amsterdam, 2004.

[26] J.-E. Pin. Positive varieties and infinite words. In Lucchesi and Moura [17], pages 76–87.

[27] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[28] C. Rispal and O. Carton. Complementation of rational sets on countable scattered linear orderings. *Int. J. Found. Comput. Sci.*, 16(4):767–786, 2005.

[29] B. L. Saëc, J.-E. Pin, and P. Weil. A purely algebraic proof of McNaughton's theorem on infinite words. In S. Biswas and K. V. Nori, editors, *FSTTCS*, volume 560 of *Lecture Notes in Computer Science*, pages 141–151. Springer, 1991.

[30] S. Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327. IEEE, 1988.

[31] M. Schützenberger. À propos des relations rationnelles fonctionnelles. In *Automata, Languages and Programming (Proc. Sympos., Rocquencourt, 1972)*, pages 103–114, Amsterdam, 1973. North Holland.

[32] L. Staiger. ω-languages. In *Handbook of formal languages, Vol. 3*, pages 339–387. Springer, Berlin, 1997.

[33] W. Thomas. Star-free regular sets of ω-sequences. *Information and Control*, 42(2):148–156, 1979.

[34] W. Thomas. A combinatorial approach to the theory of ω-automata. *Information and Control*, 48(3):261–283, 1981.

[35] W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25(3):360–376, 1982.

[36] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 133–192. Elsevier, Amsterdam, 1990.

[37] W. Thomas. Languages, automata, and logic. In *Handbook of formal languages, Vol. 3*, pages 389–455. Springer, Berlin, 1997.

[38] Th. Wilke. An Eilenberg theorem for infinity-languages. In J. L. Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 588–599. Springer, 1991.

[39] Th. Wilke. An algebraic theory for regular languages of finite and infinite words. *Internat. J. Algebra Comput.*, 3(4):447–489, 1993.

[40] J. Wojciechowski. The ordinals less than ω^ω are definable by finite automata. In *Algebra, combinatorics and logic in computer science, Vol. I, II (Győr, 1983)*, volume 42 of *Colloq. Math. Soc. János Bolyai*, pages 871–887. North-Holland, Amsterdam, 1986.

Deterministic graph grammars[*]

Didier Caucal

Institut Gaspard Monge / C.N.R.S.
Université de Paris-Est
5, boulevard Descartes
77454 Champs-sur-Marne, France
caucal@univ-mlv.fr

Abstract

This paper is a first attempt at a general survey of deterministic graph grammars and the class of graphs they generate. We focus on providing some of the basic tools to reason about deterministic graph grammars, and on a structural study of their generated graphs.

1 Introduction

Context-free grammars are one of the most classical and fundamental notions in computer science textbooks, in both theoretical and applied settings. As characterizations of the well-known class of context-free languages, they are a very prominent tool in the field of language theory. Since context-free grammars are powerful enough to express most programming languages, they also play an important role in compilation, where they form the basis of many efficient parsing algorithms.

A similar notion can be adapted to the more general setting of grammars generating graphs instead of words. In this case, grammar rules no longer express the replacement of a non-terminal letter by a string of terminal and non-terminal letters, but that of a non-terminal arc (or more generally hyperarc) by a finite graph (or hypergraph) possibly containing new non-terminals, thus generating larger and larger graphs. It is still relevant to call such grammars context-free, since the replacement of a given non-terminal is independent of the context in which it is performed, *i. e.* the remainder of the graph it is applied to, which is left unchanged.

Also, whenever two non-terminals can be replaced, the corresponding derivation steps are independent. Consequently, starting from a given graph, it is possible to describe any sequence of productions (a derivation) as a derivation tree. This intuitively explains why many notions suitable for the study of context-free word grammars extend to context-free (also called hyperedge-replacement) graph grammars (see for instance [9]).

[*] Let me thank Arnaud Carayol and Antoine Meyer for their help in drafting this paper. Many thanks to Wolfgang Thomas for his support, and happy birthday.

In this paper, we are concerned with the specific setting where the considered sets of grammar rules are deterministic, meaning that there is only one production rule for every non-terminal hyperarc. Consequently, from a given axiom, a grammar does not generate a set of graphs (which could be called a “context-free” graph language), but a unique graph up to isomorphism called a regular graph. This is an important restriction, which entails another crucial conceptual difference with word grammars. Note that grammars generating a unique finite graph are trivial: they are equivalent to grammars containing a unique rule, or even no rule if any finite graph is allowed as an axiom. As a result and contrary to the case of words, we are not interested in graphs generated after a finite derivation sequence, but in graphs generated “at the limit” *i. e.* after an infinite number of steps (see Figure 2.8 and Figure 2.9).

These deterministic graph grammars correspond to the finite systems of equations over graph operators originally defined by Courcelle [7], and whose least solutions, called equational graphs, are the regular graphs. This kind of graphs was first considered by Muller and Schupp [8]: they showed that the connected components of the transition graphs of pushdown automata are the connected graphs of finite degree whose decomposition by distance from a vertex yields finitely many non-isomorphic connected components. These graphs are exactly the connected regular graphs of finite degree [4] (see also Section 5).

This work is a first attempt at a general survey of deterministic graph grammars and the class of graphs they generate. We focus on providing some of the basic tools to reason about deterministic graph grammars, and on a structural study of their generated graphs.

First, Section 2 presents the necessary definitions as well as some examples of grammars and their generated graphs. We also define a canonical representant of the set of isomorphic graphs generated by a given grammar.

Second, as is the case for word grammars, we need to provide a collection of normal forms before being able to conveniently write more involved proofs. This is a slightly tedious but necessary task, which is addressed in Section 3, where in particular the notions of reduced, proper and connected grammars are defined. We provide a way to cleanly separate input and output vertices in grammar rules. We also show that considering multi-hypergraphs does not improve expressiveness. All these results are obtained via fixed-point computations. This allows us, as a first application, to derive some structural properties of regular graphs, namely that they only have a finite number of non-isomorphic connected components, and that the sets of possible vertex degrees in such graphs are finite.

A problematic feature of regular graphs is that any given such graph can be generated by infinitely many different graph grammars. In Section

4, we investigate systematic ways to generate regular graphs, for instance according to the length of their vertex names for pushdown graphs, or more generally, by increasing distance from the vertices having a given colour. This yields a notion of a canonical graph grammar associated to any regular graph (which will prove useful in the following section). It also allows us to establish the closure of the class of regular graphs under various vertex colouring operations.

Section 5 builds up on all the notions and results presented in the previous sections to establish a characterization of regular graphs of bounded degree, either in a general way by the suffix transition graphs of labelled word rewriting systems, or in a restrictive way by the transition graphs of pushdown automata in a weak form.

Finally in Section 6, we present a simple and strong connection between deterministic graph grammars and context-free grammars over words, and hence also context-free word languages: even though regular graphs may in general have an infinite degree, the set of path labels between two regular sets of vertices in a regular graph remains a context-free language. In this respect, deterministic graph grammars provide a natural and powerful tool to reason about context-free languages, and indeed several classical results in the theory of context-free languages can be reassessed in this framework. To summarize, deterministic graph grammars are not only finite representations of infinite graphs whose structure is regular (*i. e.* which have a finite decomposition by distance), they are also to context-free languages what finite automata are to regular languages.

2 Regular graphs

In this section, we introduce the notion of deterministic graph grammar (Section 2.2) together with the family of graphs they generate: the regular graphs (Section 2.3). We conclude by presenting several examples of regular graphs. But first, we introduce basic notations on graphs and hypergraphs (Section 2.1).

2.1 Graphs

Let $\mathbb{N}$ be the set of natural numbers and $\mathbb{N}^+ = \mathbb{N} - \{0\}$. A set in bijection with $\mathbb{N}$ is called *countable*. For a set E, we write $|E|$ for its cardinal, 2^E for its powerset and for every $n \geq 0$, $E^n = \{(e_1, \ldots, e_n) \mid e_1, \ldots, e_n \in E\}$ is the set of n-tuples of elements of E. Thus $E^* = \bigcup_{n \geq 0} E^n$ is the free monoid generated by E for the *concatenation*: $(e_1, \ldots, e_m) \cdot (e'_1, \ldots, e'_n) = (e_1, \ldots, e_m, e'_1, \ldots, e'_n)$, and whose neutral element is the 0-tuple $()$. A finite set E of symbols is an *alphabet* of *letters*, and E^* is the set of *words* over E. Any word $u \in E^n$ is of *length* $|u| = n$ and is also represented by a mapping from $[n] = \{1, \ldots, n\}$ into E, or by the juxtaposition of its letters:

$u = u(1)\dots u(|u|)$. The neutral element is the word of length 0 called the *empty word* and denoted by ε.

A *multi-subset* M of E is a mapping from E into $\mathbb{N}$ where for any $e \in E$, the integer $M(e)$ is its *multiplicity* (the number of occurrences of e in M). A multi-subset M of E is also represented by the functional subset $\{(e, M(e)) \mid e \in E \wedge M(e) \neq 0\}$ of $E \times \mathbb{N}^+$: if $(e,m),(e,n) \in M$ then $m = n$. The *cardinal* of M is $|M| = \sum_{e \in E} M(e)$, and M is said to be finite if its *support* $\widehat{M} := \{e \in E \mid M(e) \neq 0\}$ is finite. By extension we write $e \in M$ for $e \in \widehat{M}$. A finite multi-subset M can also be described by a subset of E where each $e \in E$ appears $M(e)$ times. For instance the multi-subset defined by $a \mapsto 3$, $b \mapsto 1$, $x \mapsto 0$ otherwise, is represented by $\{(a,3),(b,1)\}$ or directly by $\{a,a,a,b\}$. For instance $\{2,2,2,5\}$ is the multi-subset of the decomposition of the number 40 into its prime factors. A subset $P \subseteq E$ corresponds to the multi-subset $\{(e,1) \mid e \in P\}$ and vice-versa.

Given multi-subsets M and N, we define the multi-subset

sum	$M + N$	by $(M+N)(e) := M(e) + N(e)$,
difference	$M - N$	by $(M-N)(e) := \max\{M(e) - N(e), 0\}$,
union	$M \cup N$	by $(M \cup N)(e) := \max\{M(e), N(e)\}$,
intersection	$M \cap N$	by $(M \cap N)(e) := \min\{M(e), N(e)\}$,

and *restriction* $M_{|P}$ to $P \subseteq E$ by

$$M_{|P}(e) := \begin{cases} M(e) & \text{if } e \in P, \\ 0 & \text{otherwise;} \end{cases}$$

We shall also write $M_{|-P}$ for $M_{|E-P}$. The *inclusion* $M \subseteq N$ means that $M(e) \leq N(e)$ for every $e \in E$.

Let F be a set of symbols called *labels*, ranked by a mapping $\varrho\colon F \to \mathbb{N}$ associating to each label f its *arity* $\varrho(f)$, and such that

$$F_n := \{f \in F \mid \varrho(f) = n\} \text{ is countable for every } n \geq 0.$$

We consider simple, oriented and labelled hypergraphs: a *hypergraph* G is a subset of $\bigcup_{n \geq 0} F_n V^n$, where V is an arbitrary set, such that its *vertex* set,

$$V_G := \{v \in V \mid FV^*vV^* \cap G \neq \varnothing\},$$

is finite or countable, and its *label* set,

$$F_G := \{f \in F \mid fV^* \cap G \neq \varnothing\},$$

is finite.

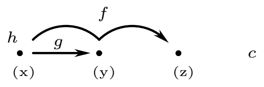


FIGURE 2.1. The hypergraph $\{fxyz, gxy, hx, c\}$.

Any $fv_1 \dots v_{\varrho(f)} \in G$ is a *hyperarc* labelled by f and of successive vertices $v_1, \dots, v_{\varrho(f)}$; it is depicted for

$\varrho(f) \geq 2$	as an arrow labelled f and successively linking $v_1, \dots, v_{\varrho(f)}$;
$\varrho(f) = 1$	as a label f on vertex v_1 and f is called a *colour* of v_1;
$\varrho(f) = 0$	as an isolated label f called a *constant*.

This is illustrated in Figure 2.1.

A vertex v is depicted by a dot named (v) where parentheses are used to differentiate a vertex name from a vertex label (a colour). Note that a hyperarc X is a word whose first letter $X(1)$ is its label, and for $1 < i \leq |X|$, the ith letter $X(i)$ is its $(i-1)$st vertex; to avoid such a shift, we also write a hyperarc as the word fY where f is its label and Y is its vertex word. Observe that a hypergraph is finite if and only if it has a finite vertex set.

The transformation of a hypergraph G by a function h from V_G into a set V is the following hypergraph:

$$h(G) := \{fh(v_1) \dots h(v_{\varrho(f)}) \mid fv_1 \dots v_{\varrho(f)} \in G\}$$

An *isomorphism* h from a hypergraph G to a hypergraph H is a bijection from V_G to V_H such that $h(G) = H$, and we write $G \overset{h}{\sim} H$ or $G \sim H$ if we do not specify the bijection.

The *restriction* of a hypergraph G to a subset $P \subseteq V_G$ is the sub-hypergraph of G induced by P:

$$G_{|P} := G \cap FP^*.$$

So $G_{|P} = \mathrm{Id}_P(G)$ where $\mathrm{Id}_P := \{(v, v) \mid v \in P\}$ is the *identity* on P.

For a hypergraph G, the *edge relation* $\underset{G}{\longleftrightarrow}$ is the binary relation on the vertex set V_G defined by

$$X(i) \underset{G}{\longleftrightarrow} X(j) \text{ for any } X \in G \text{ and } i \neq j \in \{2, \dots, |X|\}.$$

We denote by $\underset{G}{\longleftrightarrow}^n$ with $n \geq 0$ the n-fold composition of $\underset{G}{\longleftrightarrow}$, with $\underset{G}{\longleftrightarrow}^0 := \mathrm{Id}_{V_G}$ the identity on V_G, and by $\underset{G}{\longleftrightarrow}^* := \bigcup_{n \geq 0} \underset{G}{\longleftrightarrow}^n$ the reflexive

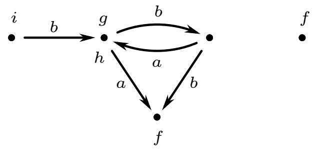


FIGURE 2.2. A finite graph.

and transitive closure of $\underset{G}{\longleftrightarrow}$. As usual s and t are *connected vertices* in G if $s \underset{G}{\longleftrightarrow}^* t$, and G is a *connected hypergraph* if the vertices of G are connected. The *degree* of a vertex s of a hypergraph G is

$$\mathrm{d}_G(s) := |\{(X,i) \mid X \in G - F_1V_G \wedge 2 \leq i \leq |X| \wedge X(i) = s\}|.$$

Note that the colouring does not affect the degree. We say that a hypergraph G is of *finite degree* (or *locally finite*) if $\mathrm{d}_G(s) < \omega$ for any vertex $s \in V_G$, and G is of *bounded degree* (or *globally finite*) if $\max\{\mathrm{d}_G(s) \mid s \in V_G\} < \omega$. For a subset $E \subseteq F$ of labels, we write

$$V_{G,E} := \{v \in V \mid EV^*vV^* \cap G \neq \varnothing\} = V_{G \cap EV_G^*}$$

the set of vertices of G linked by a hyperarc labelled in E.

A *graph* G is a hypergraph without constants and without labels of arity strictly greater than 2: $F_G \subset F_1 \cup F_2$. Hence a graph G is a set of *arcs* av_1v_2 identified with the labelled transition $v_1 \xrightarrow[G]{a} v_2$ or directly $v_1 \xrightarrow{a} v_2$ if G is understood, plus a set of coloured vertices fv. For instance, the finite graph:

$$\{r \xrightarrow{b} p, p \xrightarrow{a} s, p \xrightarrow{b} q, q \xrightarrow{a} p, q \xrightarrow{b} s, ir, gp, hp, fs, ft\}$$

has vertices p,q,r,s,t, colours f,g,h,i and arc labels a,b, and is represented in Figure 2.2; we omit the names of the vertices to give a representation up to isomorphism.

A tuple $(v_0,a_1,v_1,\ldots,a_n,v_n)$ for $n \geq 0$ and $v_0 \xrightarrow[G]{a_1} v_1 \ldots v_{n-1} \xrightarrow[G]{a_n} v_n$ is a *path* from v_0 to v_n labelled by $u = a_1 \ldots a_n$; we write $v_0 \overset{u}{\underset{G}{\Longrightarrow}} v_n$ or directly $v_0 \overset{u}{\Longrightarrow} v_n$ if G is understood. For $E \subseteq F_2^*$, we write $v \overset{E}{\underset{G}{\Longrightarrow}} v'$ if $v \overset{u}{\underset{G}{\Longrightarrow}} v'$ for some $u \in E$.

Given a graph G and vertex sets $P, Q \subseteq V_G$, we write $L(G,P,Q)$ the language of path labels from a vertex in P to a vertex in Q:

$$L(G,P,Q) := \{u \in F_2^* \mid \exists p \in P\ \exists q \in Q (p \overset{u}{\underset{G}{\Longrightarrow}} q)\}.$$

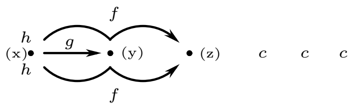


FIGURE 2.3. The multi-hypergraph $\{fxyz, fxyz, gxy, hx, hx, c, c, c\}$.

Given colours $i, f \in F_1$, we define $L(G, i, f) := L(G, V_{G,i}, V_{G,f})$ as the path labels from the set $V_{G,i}$ of vertices coloured by i to the set $V_{G,f}$ of vertices coloured by f. For instance taking the previous graph, its path labels from i to f is $b(ba)^*(a + bb)$.

Hence a finite graph G with two colours i and f is a *finite automaton* recognizing the language $L(G, i, f)$. For any (finite) alphabet $T \subset F_2$, the family

$$\mathrm{Rat}(T^*) := \{L(G, i, f) \mid |G| < \omega \wedge F_G \cap F_2 \subseteq T \wedge i, f \in F_1\}$$

of languages over T recognized by the finite automata coincides with the family of *regular languages* over T. A graph G without vertex label, *i. e.* such that $F_G \subset F_2$, is called an *uncoloured graph.*

The family of hypergraphs ordered by inclusion $\subseteq$ forms a complete partial order: its least element is the empty graph $\varnothing$ and any sequence $(G_n)_{n\geq 0}$ (not necessarily increasing) with a finite label set $\bigcup_{n\geq 0} F_{G_n}$ has a least upper bound $\bigcup_{n\geq 0} G_n$.

If we fix a finite or countable set V of vertices and a finite set $E \subset F$ of labels, the family $\mathcal{G}(V, E)$ of subsets of $\bigcup_{n\geq 0} E_n V^n$ with $E_n = E \cap F_n$ for any $n \geq 0$, is the set of hypergraphs G with $V_G \subseteq V$ and $F_G \subseteq E$. Such a set $\mathcal{G}(V, E)$ is a complete lattice: $\varnothing$ is the least element, $\bigcup_{n\geq 0} E_n V^n$ is the greatest element, and every subset $\mathcal{H} \subseteq \mathcal{G}(V, E)$ has a supremum $\bigcup \mathcal{H}$ and an infimum $\bigcap \mathcal{H}$.

A *multi-hypergraph* G is a multi-subset of $\bigcup_{n\geq 0} F_n V^n$ where V is an arbitrary set; each hyperarc $X \in G$ is depicted $G(X)$ times. The vertex set V_G and the label set F_G of a multi-hypergraph G are the sets defined on its support $\widehat{G}$, *i. e.* $V_G := V_{\widehat{G}}$ and $F_G := F_{\widehat{G}}$. The transformation of any multi-graph G by any function h from V_G into a set is extended in a natural way:

$$h(G)(X) := \sum_{h(Y)=X} G(Y) \text{ for any hyperarc } X$$

assuming that the sum is always finite. Given $f \in F_1$ and $v \in V$, the sequence $\{(fv, n)\}_{n\geq 1}$ is increasing for the inclusion but it has no least upper bound because an infinite multiplicity is not allowed.

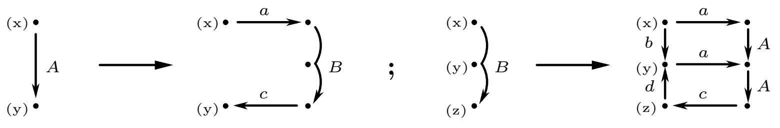


FIGURE 2.4. A (deterministic) graph grammar.

2.2 Graph grammars

A hypergraph *grammar* R is a finite set of rules of the form $fx_1 \ldots x_{\varrho(f)} \longrightarrow H$ where $fx_1 \ldots x_{\varrho(f)}$ is a hyperarc joining pairwise distinct vertices $x_1 \neq \ldots \neq x_{\varrho(f)}$ and H is a finite multi-hypergraph; we denote by $N_R := \{f \in F \mid fX \in \mathrm{Dom}(R)\}$ the *non-terminals* of R, the labels of the left hand sides; by $T_R := \{f \in F - N_R \mid \exists P \in \mathrm{Im}(R)(fX \in P)\}$ the *terminals* of R, the labels of R which are not non-terminals; by $F_R := N_R \cup T_R$ the labels of R; and by $\varrho(R) := \max\{\varrho(A) \mid A \in N_R\}$ the *arity* of R, the maximal arity of its non-terminals.

We use grammars to generate simple graphs (without multiplicity). Hence in the following, we may assume that any terminal hyperarc of any right hand side is of multiplicity 1, otherwise we replace R by

$$\{(X, \langle H \rangle) \mid (X, H) \in R\}$$

where $\langle H \rangle$ is obtained from H by removing the multiplicity of the terminal hyperarcs:

$$\langle H \rangle := H_{|N_R V_H^*} \cup (H \cap T_R V_H^*).$$

Remark that multiplicities of non-terminal hyperarcs are usually not introduced when working with graph grammars. As explained in the next subsection, they are in all generality necessary to ensure the unicity of the graph generated (see also Figure 2.6). In the next section, we shall see that any graph grammar can be transformed into an equivalent grammar where multiplicities do not need to be taken into account.

Starting from any hypergraph, we want a grammar to generate a unique hypergraph up to isomorphism. So we restrict ourselves to *deterministic* grammars, meaning that there is only one rule per non-terminal:

$$(X, H), (Y, K) \in R \wedge X(1) = Y(1) \Longrightarrow (X, H) = (Y, K).$$

For any rule $X \longrightarrow H$, we say that $V_X \cap V_H$ are the *inputs* of H and $\bigcup\{V_Y \mid Y \in H \wedge Y(1) \in N_R\}$ are the *outputs* of H. We shall use upper-case letters $A, B, C, \ldots$ for non-terminals and lower-case letters $a, b, c \ldots$ for terminals. We say that R is a *graph grammar* if the terminals are of arity

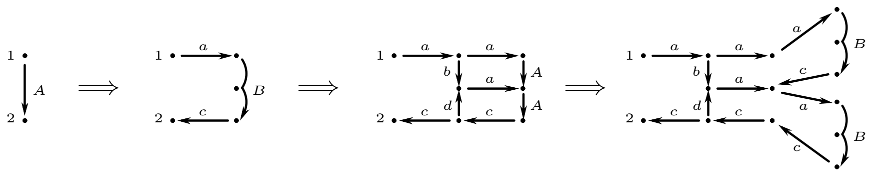


FIGURE 2.5. Parallel rewritings according to the grammar of Figure 2.4.

1 or 2. An example is given in Figure 2.4 where we have

$$N_R = \{A, B\}, \qquad T_R = \{a, b, c, d\}, \qquad \varrho(R) = 3,$$

and the inputs of the first and second rule are x, y and x, y, z, respectively.

Given a grammar R, the *rewriting* $\underset{R}{\longrightarrow}$ is the binary relation between multi-hypergraphs defined as follows: M rewrites into N, written $M \underset{R}{\longrightarrow} N$, if we can choose a non-terminal hyperarc $X = As_1 \dots s_p$ in M and a rule $Ax_1 \dots x_p \longrightarrow H$ in R such that N can be obtained by replacing X by H in M and by removing the multiplicity of terminal hyperarcs:

$$N = \langle (M - X) + h(H) \rangle$$

for some function h mapping each x_i to s_i, and the other vertices of H injectively to vertices outside of M; this rewriting is denoted by $M \underset{R,X}{\longrightarrow} N$. The rewriting $\underset{R,X}{\longrightarrow}$ of a hyperarc X is extended in an obvious way to the rewriting $\underset{R,E}{\longrightarrow}$ of any multi-subset E of non-terminal hyperarcs. A *complete parallel rewriting* $\underset{R}{\Longrightarrow}$ is the rewriting according to the multi-subset of all non-terminal hyperarcs: $M \underset{R}{\Longrightarrow} N$ if $M \underset{R,E}{\longrightarrow} N$ where E is the multi-subset of all non-terminal hyperarcs of M.

For instance, the first three steps of the parallel derivation from the graph $\{Axy, 1x, 2y\}$ according to the grammar of Figure 2.4 are depicted in Figure 2.5.

The *derivation* $\underset{R}{\Longrightarrow}^*$ is the reflexive and transitive closure for composition of the parallel rewriting $\underset{R}{\Longrightarrow}$ (*i. e.* $G \underset{R}{\Longrightarrow}^* H$ if H is obtained from G by a consecutive sequence of parallel rewritings). We can now define the graphs generated by deterministic graph grammars.

2.3 Regular graphs

Intuitively the graph (up to isomorphism) generated by a deterministic graph grammar R from a finite graph G_0 is the limit of any infinite se-

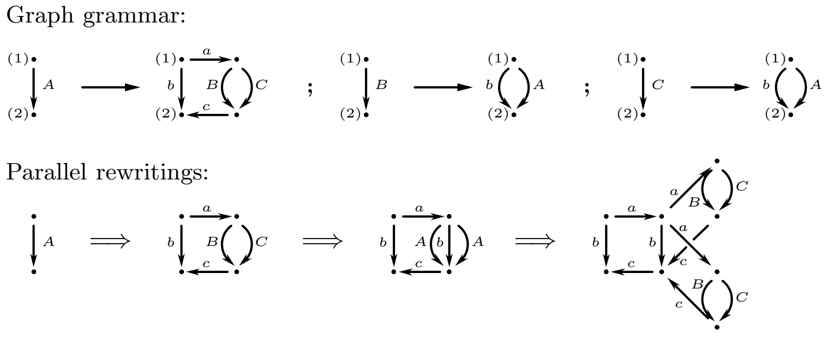


FIGURE 2.6. Parallel rewritings producing multiplicity.

quence of rewritings starting from G_0 where every non-terminal is eventually rewritten. Formally, to a sequence $(G_i)_{i\geq 0}$ of finite multi-hypergraphs such that

1. for all $i \geq 0$, $G_i \underset{R,X_i}{\longrightarrow} G_{i+1}$, and
2. for all $X \in G_i$ with $X(1) \in N_R$, there exists $j \geq i$ such that $X = X_j$,

we associate the limit $\bigcup_{i\geq 0}[G_i]$ where for a hypergraph M, we designate by $[M] := M \cap T_R V_M^*$ designates the (simple) set of terminal hyperarcs of M.

Note that the sequence $(G_i)_{i\geq 0}$ can be not increasing contrary to the sequence $([G_i])_{i\geq 0}$; in particular, even if $\bigcup_{i\geq 0}[G_i]$ is finite, the sequence $(G_i)_{i\geq 0}$ is not necessarily ultimately constant. It is easy to check that this limit does not depend on the order of the rewriting. In particular, we can use the parallel rewriting $\underset{R}{\Longrightarrow}$ which provides a canonical rewriting order similar to the leftmost rewriting for context-free grammars. The example in Figure 2.6 illustrates that without multiplicities, the unicity of the limit graph no longer holds.

We shall see in next section that, though multiplicities are crucial in ensuring the unicity of the generated graph, they can be omitted provided that the grammar respects a certain normal form (see Subsection 3.2).

A hypergraph G is *generated by a grammar* R from a hypergraph H if G is isomorphic to a hypergraph in the following set:

$$R^\omega(H) := \Big\{ \bigcup_{n\geq 0}[H_n] \Big| H_0 = H \wedge \forall n \geq 0 (H_n \underset{R}{\Longrightarrow} H_{n+1}) \Big\};$$

note that the vertices of H appear in any hypergraph of $R^\omega(H)$. For instance by continuing infinitely the derivation of Figure 2.5, we get a graph depicted

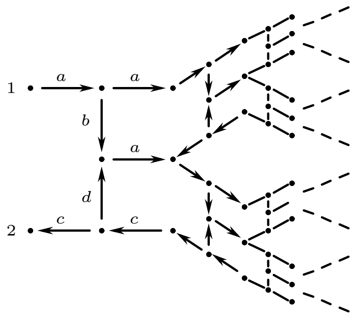


FIGURE 2.7. Graph generated by the grammar of Figure 2.4.

in Figure 2.7. Note that the definition of $R^\omega(H)$ does not fix a particular naming of the vertices of the graph generated by R. A canonical naming is provided in Section 3.5.

A *regular hypergraph* is a hypergraph generated by a (deterministic) grammar from a finite hypergraph. The regular hypergraphs are the *hyperedge replacement equational hypergraphs* in the sense of [7], which are defined as the smallest solutions of finite systems of equations involving a set of hypergraph operators.

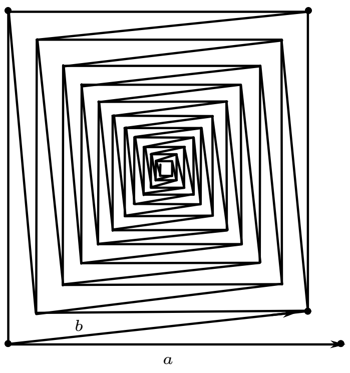


FIGURE 2.8. A regular graph.

A *regular graph* is a regular hypergraph which is a graph: it is generated by a graph grammar from a finite hypergraph. We give some examples of regular graphs. The grammar R reduced to the unique rule

$$A1234 \longrightarrow \{a21, b25, A2345\}$$

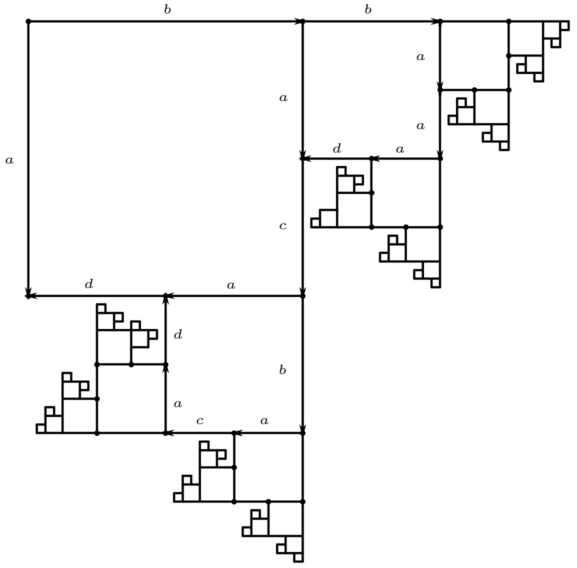
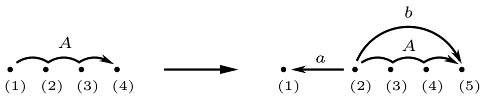
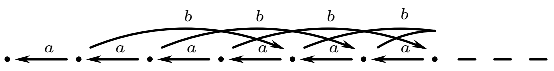


FIGURE 2.9. Another regular graph.

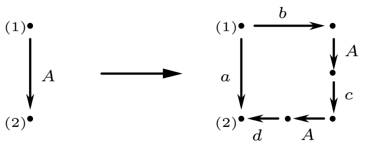
and represented below:



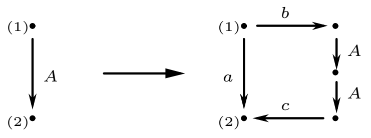
generates from its left hand side the following regular graph:



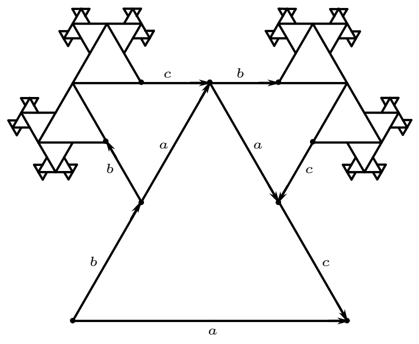
which can be drawn without crossing edges as the regular graph in Figure 2.8. Another example of graph grammar is the grammar reduced to the following rule:



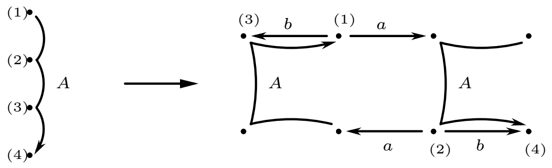
generating from its left hand side the regular graph in Figure 2.9. The grammar reduced to the following rule:



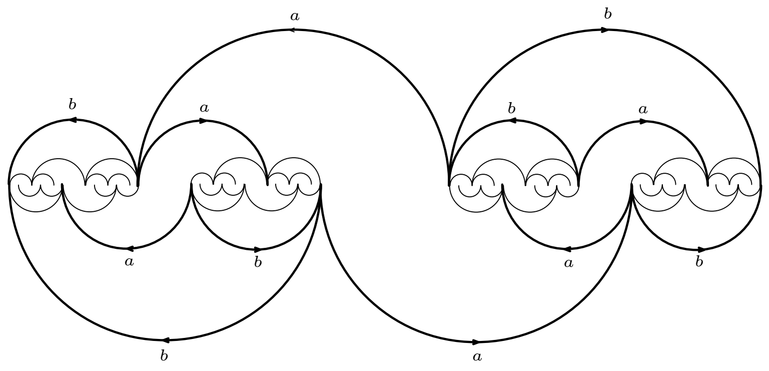
generates from its left hand side the following regular graph:



Finally the graph grammar reduced to the following rule:



generates from its left hand side the regular graph below, where each vertex is of infinite in-degree:



3 Normalizations of graph grammars

In this section, we present several elementary transformations to normalize hypergraph grammars. The first normalization gives an equivalent grammar with a constant axiom such that each non-terminal is accessible and generates a non empty graph which is connected except for the axiom and possibly another constant (cf. Proposition 3.5).

This normalization is extended to get ride of multiplicities both in the definition of the graph grammar and in its derivation relation. To ensure that multiplicities are not needed, we ask that any non-terminal hyperarc appearing in a right hand side of a rule contains a vertex which is not an input (cf. Proposition 3.10). We extend this second normalization by separating as much as possible for each right hand side the inputs and the outputs (cf. Theorem 3.12). All these basic transformations are expressed in a powerful and natural way as fixed point computations. These normalizations are used to derive properties on the generated graphs: any regular graph has a finite number of non-isomorphic connected components, and a finite number of vertex degrees (cf. Propositions 3.4 and 3.13). Finally we give a canonical vertex naming for the regular graphs (cf. Subsection 3.5).

3.1 Reduced and connected form

We begin by transforming any grammar into a reduced form. We say that a grammar R is *reduced* if $R = \varnothing$ or there exists a constant non-terminal $Z \in \mathrm{Dom}(R) \cap F_0$ called the *axiom* such that the following three conditions are satisfied:

(i) for all $H \in \mathrm{Im}(R)$, $Z \notin F_H$

(ii) for all $A \in N_R$ there exists H such that $Z \underset{R}{\Longrightarrow}^* H$ and $A \in F_H$

(iii) $R^\omega(X) \neq \varnothing$ for every $X \in \mathrm{Dom}(R)$;

the axiom is a non-terminal constant which by condition (i) does not appear in the right hand sides of the rules, condition (ii) means that each non-terminal is accessible from the axiom, and condition (iii) expresses that R generates a non empty hypergraph from any non-terminal hyperarc. By condition (iii), the grammar $\varnothing$ (with no rule) is the unique reduced grammar generating the empty graph $\varnothing$. By conditions (i) and (ii), a non empty reduced grammar has a unique axiom. For instance the grammar of Figure 2.4 is not reduced, but it becomes reduced by adding the rule $Z \longrightarrow Axy$.

We say that a hypergraph G is *generated by a reduced grammar* R if $R = G = \varnothing$ or if the reduced grammar R is non empty and generates G from its axiom.

Lemma 3.1. Any regular hypergraph can be generated in an effective way by a reduced grammar.

Proof. Let G be a hypergraph generated by a deterministic grammar R from a finite multi-hypergraph G_0.

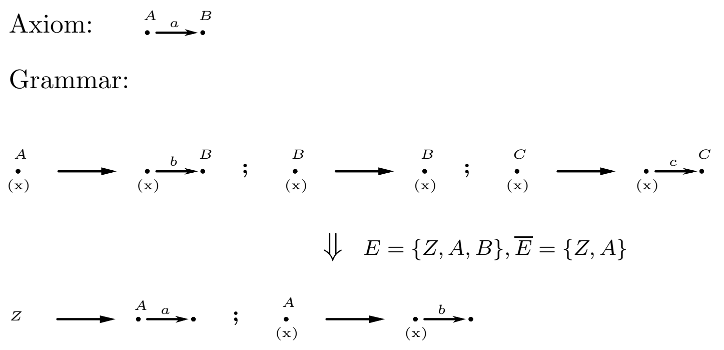


FIGURE 3.1. Reduction of a grammar.

We take a new constant $Z \in F_0 - F_R$ and we complete R into $\overline{R} := R \cup \{(Z, G_0)\}$. The set $E := \bigcup\{F_K \cap N_{\overline{R}} \mid Z \underset{\overline{R}}{\Longrightarrow}^* K\}$ of *accessible* non-terminals from Z is the least fixed point of the equation

$$E = \{Z\} \cup \{Y(1) \in N_R \mid \exists (X, H) \in \overline{R}(X(1) \in E \wedge Y \in H)\}.$$

The set $\overline{E} := \{X(1) \in E \mid \overline{R}^{\omega}(X) \neq \{\varnothing\}\}$ of *productive* accessible non-terminals is the least fixed point of the equation

$$\overline{E} = \{X(1) \in E \mid \exists P((X, P) \in \overline{R} \wedge P \cap (\overline{E} \cup T_{\overline{R}})V_P^* \neq \varnothing)\}.$$

The following grammar

$$S := \{(X, P_{|(\overline{E} \cup T_{\overline{R}})V_P^*}) \mid (X, P) \in \overline{R} \wedge X(1) \in \overline{E}\}$$

is reduced and generates G. Q.E.D. (Lemma 3.1)

The “standard” construction in the proof of Lemma 3.1 is illustrated in Figure 3.1. Another form of grammar is to be *proper*:

$$V_X \subseteq V_G \text{ for all } X \in \mathrm{Dom}(R) \text{ and for all } G \in R^{\omega}(X)$$

meaning that any vertex of the left hand side X of any rule, is a vertex of its right hand side and is a vertex of a terminal hyperarc of any graph obtained by a derivation from X. For instance the grammar of Figure 2.4 is proper but the following grammar:

$$Axy \longrightarrow \{axz, Azy\}$$

is not proper because the vertex y of Axy does not belong to any graph of $R^{\omega}(Axy)$.

Lemma 3.2. Any regular hypergraph can be generated in an effective way by a proper and reduced grammar.

Proof. Let G be a regular hypergraph. We may assume $G \neq \varnothing$ because $\varnothing$ is a proper and reduced grammar generating $\varnothing$. By Lemma 3.1, G is generated by a reduced grammar R from its axiom Z.

For every rule $Ax_1 \dots x_{\varrho(A)} \longrightarrow P_A$ in R, we define the set $\operatorname{Keep}(A)$ of indices $1 \leq i \leq \varrho(A)$ such that x_i is *useful*:

$$x_i \in V_G \text{ for all } G \in R^\omega(Ax_1 \dots x_{\varrho(A)})$$

This collection of sets $\operatorname{Keep}(A)$ is the least fixed point of the following recursive system:

$$\begin{aligned}\operatorname{Keep}(A) := \{i \in [\varrho(A)] \mid {} & P_A \cap T_R V_{P_A}^* x_i V_{P_A}^* \neq \varnothing \vee \\ & \exists BY \in P_A (B \in N_R \wedge \exists 1 \leq j \leq |Y| (Y(j) = x_i \wedge j \in \operatorname{Keep}(B)))\}.\end{aligned}$$

To each $A \in N_R$, we associate a new symbol A' of arity $|\operatorname{Keep}(A)|$. To each non-terminal hyperarc $Ay_1 \dots y_{\varrho(A)}$ (with $A \in N_R$), we associate the following hyperarc:

$$h(Ay_1 \dots y_{\varrho(A)}) := A' y_{i_1} \dots y_{i_p}$$

with $\{i_1, \dots, i_p\} = \operatorname{Keep}(A)$ and $i_1 < \dots < i_p$. We complete h by the identity: $h(X) := X$ for any terminal hyperarc X. Finally we extend h by union to any multi-hypergraph H: $h(H) := \{h(X) \mid X \in H\}$. We define a grammar

$$h(R) := \{(h(X), h(H)) \mid (X, H) \in R\}.$$

The grammar $h(R)$ is proper, reduced and generates G from its axiom $h(Z) = Z'$. Q.E.D. (Lemma 3.2)

The construction in the proof of Lemma 3.2 is illustrated in Figure 3.2.

We now want to generate regular hypergraphs using grammars in two parts: a set of rules producing only connected graphs, and a set of rules whose left hand sides are constants. Note that for any reduced grammar generating a connected hypergraph, the axiom is the unique non-terminal of null arity. A *connected grammar* R is a proper grammar such that

$$\text{for all } X \in \operatorname{Dom}(R) - F_0 \text{ and all } G \in R^\omega(X), G \text{ is connected.}$$

In particular for every rule $(X, H) \in R$ with $X \notin F_0$, we have $H \cap F_0 = \varnothing$. Let us extend Lemma 3.2.

Lemma 3.3. Any regular hypergraph can be generated in an effective way by a connected and reduced grammar.

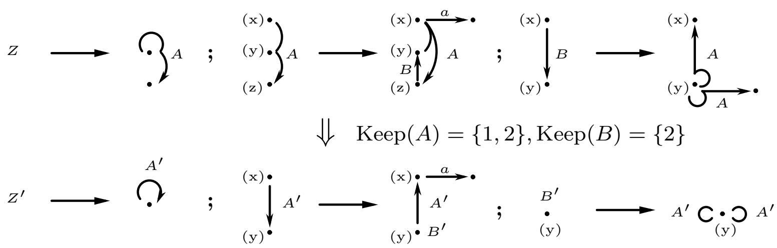


FIGURE 3.2. Transformation of a grammar into a proper grammar.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Lemma 3.2, G is generated by a proper and reduced grammar R from its axiom Z. For every rule $Ax_1 \ldots x_{\varrho(A)} \longrightarrow H_A$ in R and for every $1 \leq i \leq \varrho(A)$, we associate the set $\mathrm{Con}(A,i)$ of vertices in H_A which are connected to x_i in $R^\omega(Ax_1 \ldots x_{\varrho(A)})$. This collection of sets $\mathrm{Con}(A,i)$ is the least fixed point of the following recursive system:

$$\mathrm{Con}(A,i) = \{x_i\} \cup \bigcup \{V_X \mid X \in H_A \wedge X(1) \in T_R \wedge V_X \cap \mathrm{Con}(A,i) \neq \varnothing\}$$
$$\cup \bigcup \{X(j) \mid \exists Y \in \mathrm{Dom}(R)$$
$$(Y(1)X \in H_A \wedge \exists k(X(k) \in \mathrm{Con}(A,i) \wedge x_j \in \mathrm{Con}(Y(1),k)))\}.$$

We complete these sets by defining for any $A \in N_R$ the set

$$\mathrm{Con}(A) := \{\mathrm{Con}(A,i) \mid 1 \leq i \leq \varrho(A)\} \cup \{\varnothing\}.$$

To each non-terminal hyperarc $X \in \mathrm{Dom}(R)$ and to each $P \in \mathrm{Con}(X(1))$, we associate a new symbol $X(1)_P$ of arity $|P \cap \{x_1, \ldots, x_{\varrho(X(1))}\}|$, and the hyperarc

$$X_P := X(1)_P x_{i_1} \ldots x_{i_p}$$

with $\{x_{i_1}, \ldots, x_{i_p}\} = P \cap \{x_1, \ldots, x_{\varrho(X(1))}\}$ and $i_1 < \ldots < i_p$.
In particular $X_\varnothing = X(1)_\varnothing$ is a constant. This permits to define the following grammar:

$$I := \{(X, \{X_P \mid P \in \mathrm{Con}(X(1))\}) \mid X \in \mathrm{Dom}(R)\}$$

which splits each $X \in \mathrm{Dom}(R)$ into hyperarcs according to $\mathrm{Con}(X(1))$. For each rule (X,H) of R, there is a unique hypergraph K_X such that $H \underset{I}{\Longrightarrow} K_X$, and we denote

$$\langle\!\langle X \rangle\!\rangle := V_{K_X} - \bigcup \mathrm{Con}(X(1))$$

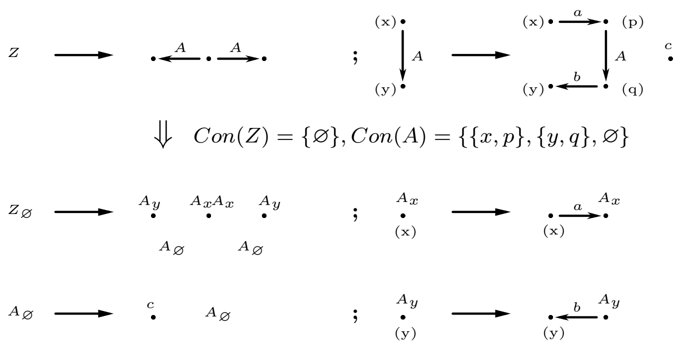
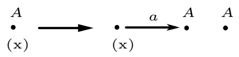

$\Downarrow$ $Con(Z) = \{\varnothing\}, Con(A) = \{\{x,p\},\{y,q\},\varnothing\}$

FIGURE 3.3. From a proper grammar to a connected grammar.

Grammar:



Graph generated from its non-terminal:

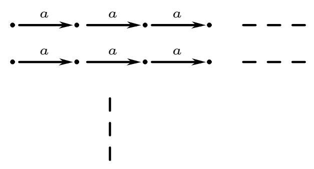


FIGURE 3.4. A non connected regular graph.

the set of vertices of H_X which are not connected to an input (a vertex in V_X). The following grammar

$$S := \{(X_P, (K_X)_{|P} - F_0) \mid X \in \mathrm{Dom}(R) \wedge P \in \mathrm{Con}(X(1)) - \{\varnothing\}\} \\ \cup \{(X_\varnothing, (K_X)_{|\langle\langle X\rangle\rangle} \mid X \in \mathrm{Dom}(R)\}$$

generates from X_P the connected component of $R^\omega(X)$ containing $P \neq \varnothing$, and S generates from $X_\varnothing$ the remaining part of $R^\omega(X)$. In particular $G \in S^\omega(Z_\varnothing)$. The grammar S is connected but it is not necessarily reduced. However by applying Lemma 3.1, we get an equivalent connected and reduced grammar of axiom $Z_\varnothing$. Q.E.D. (Lemma 3.3)

The transformation of Lemma 3.3 is illustrated in Figure 3.3.

A regular graph can have an infinite number of connected components as shown in Figure 3.4. An even simpler example is given by the graph grammar reduced to the unique rule $Z \longrightarrow \{axy, Z\}$ which generates from the constant Z the infinite repetition of an a-arc. However these two regular graphs have only a unique connected component up to isomorphism. Let us generalize this property.

Proposition 3.4. A regular hypergraph has a finite number of non-isomorphic connected components.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Lemma 3.3, G is generated by a connected and reduced grammar R from its axiom Z. We restrict R to the grammar $S := \{(X, H) \in R \mid X \notin F_0\}$. The grammar S preserves connectivity:

$$S^\omega(K) \text{ is connected for any connected hypergraph } K \notin N_R \cap F_0.$$

Any connected component of G is isomorphic to a hypergraph of the following set:

$$\bigcup\{S^\omega(K) \mid \exists H \in \mathrm{Im}(R)(K \text{ connected component of } H - (N_R \cap F_0))\}$$

which has a finite number of non-isomorphic hypergraphs.

Q.E.D. (Proposition 3.4)

Let us now normalize the constant rules: A grammar R is *strongly reduced* if R is a reduced grammar with at most two non-terminal constants (*i. e.* $|N_R \cap F_0| \leq 2$), and such that

$$(X, H) \in R \wedge X \notin F_0 \Longrightarrow H \cap F_0 = \varnothing.$$

Note that this last condition is already satisfied if R is connected. Let us extend Lemma 3.3.

Proposition 3.5. Any regular hypergraph can be generated in an effective way by a connected and strongly reduced grammar.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Lemma 3.3, G is generated by a connected and reduced grammar R from its axiom Z. We extract in R the following constant rules:

$$R_0 := \{(X, Y) \mid \exists H((X, H) \in R \wedge X \in F_0 \wedge Y \in F_H \cap N_R \cap F_0)\}$$

in order to determine the following subset of "non-repetitive" constant non-terminals:

$$\mathrm{NRep} := \{A \in N_R \cap F_0 \mid (A, A) \notin R_0^+\} - \{Z\}.$$

First we restrict R to the rules of its non-repetitives constant non-terminals:

$$I := \{(X,H) \in R \mid X \in \mathrm{NRep}\}.$$

To each $X \in \mathrm{NRep}$, we derive a hypergraph H_X such that

$$X \underset{I}{\Longrightarrow}^* H_X \wedge F_{H_X} \cap \mathrm{NRep} = \varnothing$$

and we define the following grammar:

$$I' := \{(X, H_X) \mid X \in \mathrm{NRep}\}.$$

By rewriting according to I', we remove the non-repetitive constant non-terminals in R. For each $X \in F_0 - \mathrm{NRep}$, we associate a hypergraph H'_X such that

$$X\ R \circ \underset{I'}{\Longrightarrow} H'_X$$

with $V_{H'_X} \cap V_{H'_Y} = \varnothing$ for every $X \neq Y$ in $F_0 - \mathrm{NRep}$.
The grammar

$$S := \{(X,H) \in R \mid X \notin F_0\} \cup \{(X, H'_X) \mid X \in F_0 - \mathrm{NRep}\}$$

remains connected, reduced and generates G from its axiom Z. The set of "repetitive" constant non-terminals is

$$\mathrm{Rep} := \{A \mid (A,A) \in R_0^+\} = (N_R \cap F_0) - (\mathrm{NRep} \cup \{Z\})$$

If $\mathrm{Rep} = \varnothing$ then S suits with $N_R \cap F_0 = \{Z\}$. Otherwise we take a new constant $Y \neq Z$ and the following graphs:

$$K_0 := (H'_Z)_{|-F_0}, \text{ the image of } Z \text{ in } S \text{ without constants,}$$
$$\text{and} \quad K := \bigcup\{(H'_X)_{|-F_0} \mid X \in \mathrm{Rep}\}.$$

The grammar

$$S' := \{(X,H) \in S \mid X \notin F_0\} \cup \{(Z, K_0 \cup \{Y\}), (Y, K \cup \{Y\})\}$$

remains connected and S' generates G from Z. By restriction to the accessible non-terminals from Z using Lemma 3.1, we get an equivalent grammar which is strongly reduced. Q.E.D. (Proposition 3.5)

The transformation of Proposition 3.5 is illustrated in Figure 3.5.

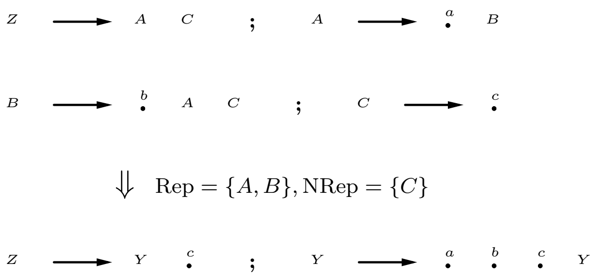


FIGURE 3.5. From a reduced grammar into a strongly reduced one.

3.2 Discarding the multiplicity

In this section, we construct for any reduced grammar a finite graph of its output dependencies by rewritings from the axiom. This permits to decide whether every derivation from the axiom is only on simple hypergraphs (without multiplicity). Then we present a normal form that allows to get ride of multiplicity. In a first time in Lemma 3.8, we show that any grammar is equivalent to one where right hand sides are hypergraphs and not multi-hypergraphs. In a second time, we show in Proposition 3.10 that any grammar is equivalent to a grammar where each non-terminal hyperarc appearing in a right hand side contains a vertex which is not an input. For a grammar in this normal form, the generated graph can be defined using only hypergraphs and not multi-hypergraphs.

Let R be any reduced grammar. An *output link* C of R is a multi-hypergraph of at most two hyperarcs which are non-terminals and with a common vertex:

$$|C| \leq 2 \wedge F_C \subseteq N_R \wedge (X, Y \in C \Longrightarrow V_X \cap V_Y \neq \varnothing);$$

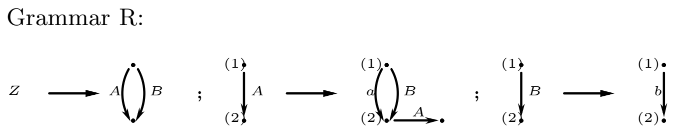
we denote $[C]_\sim := \{D \mid C \sim D\}$ the closure of C by isomorphism. The *output dependency graph* $\mathrm{Out}(R)$ of R is

$$\mathrm{Out}(R) := G_{|\{s \mid [Z]_\sim \underset{G}{\longrightarrow}^* s\}}$$

the graph G below and restricted to its vertices accessible from $[Z]_\sim$:

$$G := \big\{[C]_\sim \longrightarrow [D]_\sim \mid C, D \text{ output links } \wedge \exists H$$
$$(C \underset{R}{\longrightarrow} H \wedge D \subseteq H \wedge (|D| = 1 \Rightarrow D \text{ connected component of } H - [H]))\big\}.$$

In Figure 3.6, we give the output dependency graph of a reduced grammar. We say that a grammar R is *without multiplicity* if R is reduced and every



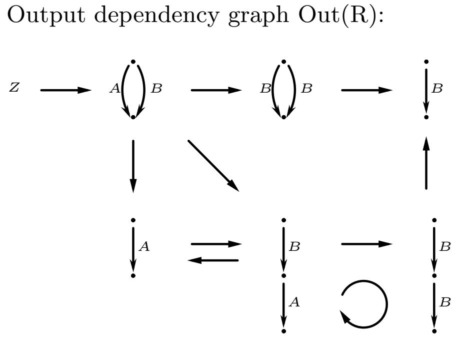


FIGURE 3.6. Output dependency graph of a grammar.

vertex of Out(R) is a simple hypergraph. Thus

$$R \text{ is without multiplicity } \iff (\forall H(Z \underset{R}{\longrightarrow}^* H \implies H \text{ simple})).$$

In particular, any grammar without multiplicity is simple.

We now want to transform any grammar into an equivalent grammar without multiplicity. We start with preliminary normal forms presented in Lemma 3.6 and 3.7. We say that a grammar R is *growing* if $R = \varnothing$ or R generates an infinite hypergraph from each left hand side, except possibly from its axiom Z:

$$\text{for all } X \in \text{Dom}(R) - \{Z\} \text{ and } G \in R^\omega(X), \text{ we have } |G| = \omega.$$

Lemma 3.6. Any regular hypergraph can be generated in an effective way by a growing, connected and strongly reduced grammar.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Proposition 3.5, G is generated by a connected and strongly reduced grammar R from its axiom Z. We define two binary relations R_0 and R_1 on the non-terminal set N_R as follows:

$$R_0 := \{(X(1), Y(1)) \mid \exists H((X,H) \in R \wedge Y \in H \wedge Y(1) \in N_R)\}$$
$$R_1 := \{(X(1), Y(1)) \mid \exists H((X,H) \in R \wedge Y \in H \wedge Y(1) \in N_R \wedge V_Y - V_X \neq \varnothing)\}$$

Then the set $E := \{A \mid \exists B((A,B) \in R_0^* \wedge (B,B) \in R_1^+)\}$ is the set of non-terminals $X(1)$ with $X \in \mathrm{Dom}(R)$ such that the graphs of $R^\omega(X)$ are infinite. We begin with the grammar

$$I_0 := \{(X, \varnothing) \mid X \in \mathrm{Dom}(R) \wedge X(1) \in N_R - E\}.$$

Having constructed a grammar I_n for $n \geq 0$, we define a deterministic grammar I_{n+1} with $\mathrm{Dom}(I_{n+1}) = \mathrm{Dom}(I_0)$ and

$$I_{n+1} \subseteq \{(X,H) \mid X\ R \circ \underset{I_n}{\Longrightarrow} H\}.$$

Note that the right hand sides of the grammars I_n do not contain any non-terminal hyperarc. We finish with the grammar $I = I_m$ for $m = \min\{n \mid I_n = I_{n+1}\}$. Thus I is a grammar with $\mathrm{Dom}(I) = \{X \in \mathrm{Dom}(R) \mid X(1) \in N_R - E\}$ and for every $(X,H) \in I$, H is finite and $H \in R^\omega(X)$.

From I, we construct a deterministic grammar S such that

$$S \subseteq \{(X,H) \mid X\ R \circ \underset{I}{\Longrightarrow} H \wedge X(1) \in E\}.$$

This grammar S is growing, connected and by restriction to the accessible non-terminals from Z, it is strongly reduced. Q.E.D. (Lemma 3.6)

We say that a grammar R is *strict* if

$$V_H - V_X \neq \varnothing \text{ for any } (X,H) \in R$$

any rule has at least one non-input vertex in its right hand side. Starting from a growing grammar, it is enough to write every right hand side until the grammar is strict.

Lemma 3.7. Any regular hypergraph can be generated in an effective way by a strict, connected and strongly reduced grammar.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Lemma 3.6, G is generated by a growing, connected and strongly reduced grammar R from its axiom Z. As R is growing, we derive each right hand side of S until we get a non input vertex. We define

$$S_0 := \{(X,H) \in R \mid V_X \neq V_H\}$$

and having defined S_n, we construct a maximal deterministic grammar

$$S_{n+1} \subseteq S_n \cup \{(X,H) \mid X \in \mathrm{Dom}(R) - \mathrm{Dom}(S_n) \wedge X\ R \circ \underset{S_n}{\Longrightarrow} H \wedge V_X \neq V_H\}$$

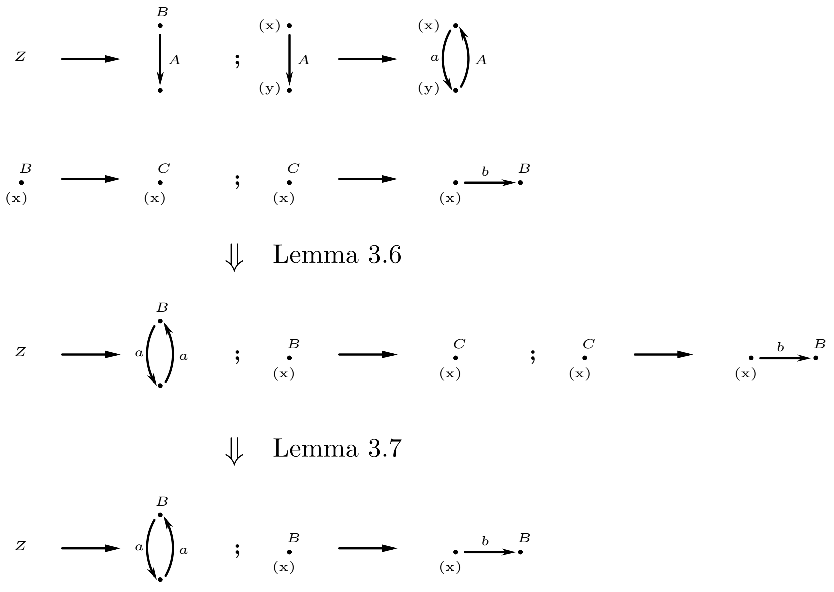


FIGURE 3.7. Transformation of a grammar into a strict grammar.

to get $S := S_m$ for $m = \min\{n \mid \forall(X,H) \in S_n(V_X \neq V_H)\}$. This grammar S is strict and generates G from its axiom Z. Furthermore S remains connected and becomes strongly reduced by restriction to the accessible non-terminals from Z. Q.E.D. (Lemma 3.7)

The transformations of Lemma 3.6 and Lemma 3.7 are illustrated in Figure 3.7.

To generate a regular (simple) hypergraph, we can avoid multiplicity in the grammar. Precisely, a *simple grammar* is a grammar where each right hand side is a (simple) hypergraph.

Lemma 3.8. Any regular hypergraph can be generated in an effective way by a simple, strict, connected and strongly reduced grammar.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Lemma 3.7, G is generated by a strict, connected and strongly reduced grammar R from its axiom Z. To each non-terminal $A \in N_R - \{Z\}$, we associate its maximal multiplicity:

$$m(A) := \max\{H(X) \mid H \in \mathrm{Im}(R) \wedge X \in H \wedge X(1) = A\},$$

and we take new non-terminals $A_1, \ldots, A_{m(A)}$ of arity $\varrho(A)$. This allows us to replace each right hand side $H \in \mathrm{Im}(R)$ by the following simple

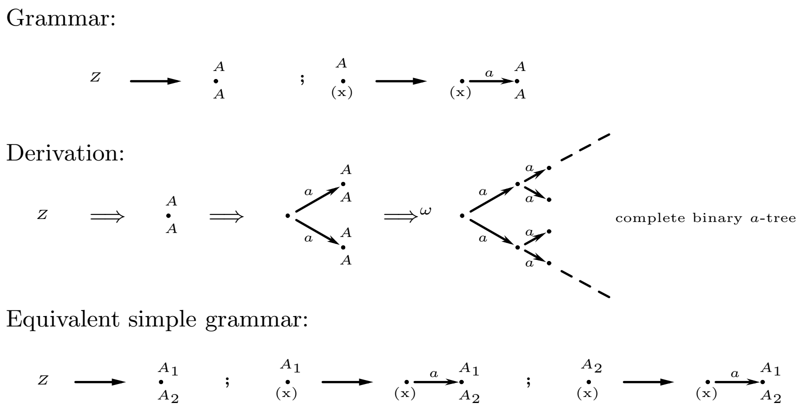


FIGURE 3.8. Transformation of a grammar into a simple grammar.

hypergraph:

$$\begin{aligned} H' := &\{X \mid X \in H \wedge X(1) \in T_R\} \\ &\cup \{X(1)_i X(2) \ldots X(|X|) \mid X \in H \cap N_R V_H^* \wedge 1 \leq i \leq H(X)\}. \end{aligned}$$

We obtain the grammar

$$\begin{aligned} S := &\{(Z, H') \mid (Z, H) \in R\} \\ &\cup \{(X(1)_i X(2) \ldots X(|X|), H') \mid (X, H) \in R \wedge 1 \leq i \leq m(X(1))\} \end{aligned}$$

This grammar S is simple, strict, connected, strongly reduced and generates G from its axiom Z. Q.E.D. (Lemma 3.8)

The transformation of Lemma 3.8 is illustrated in Figure 3.8.

To generate a regular hypergraph, we also want to reduce the rewriting steps to (simple) hypergraphs. This is not possible in general as shown in Figures 2.6 and 3.9. However any regular hypergraph can be generated by a simple hypergraph grammar whose rewriting steps are restricted to simple hypergraphs. A grammar R is *non-terminal outside* if for any rule $X \longrightarrow H$, any non-terminal hyperarc $Y \in H$ with $Y(1) \in N_R$ has a vertex which is not an input: $V_Y - V_X \neq \emptyset$. The grammar of Figure 3.4 is non-terminal outside and the grammar of Figure 3.9 is not. With the property of being strongly reduced, we have removed the multiplicity by parallel rewritings for constants. The non-terminal outside property removes the multiplicity by parallel rewritings for non-constant non-terminals.

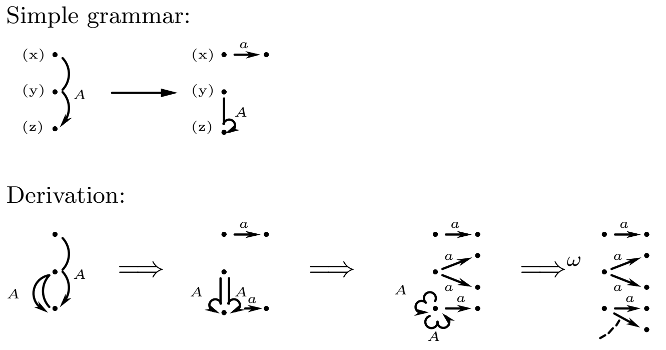


FIGURE 3.9. Multiplicity by parallel rewritings.

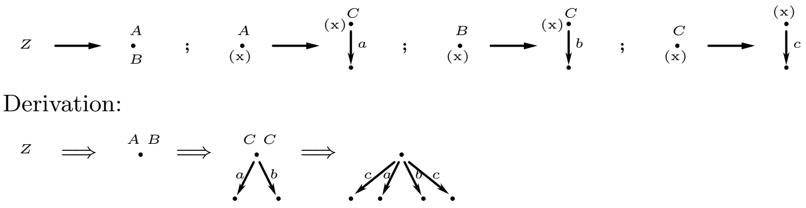


FIGURE 3.10. A graph grammar which is not non-terminal outside.

Lemma 3.9. Any non-terminal outside, simple and strongly reduced grammar is without multiplicity.

Proof. Let R be any non-terminal outside, simple and strongly reduced grammar. By induction, we verify that the rewriting $\underset{R}{\longrightarrow}$ preserves the property $P(H)$ of a hypergraph H to be simple and with at most one non-terminal constant:

$$P(H) \wedge H \underset{R}{\longrightarrow} K \Longrightarrow P(K).$$

Let X be the left hand side of the applied rule. If X is a constant then the implication is due to R being simple and strongly reduced. If X is not a constant then the implication is due to R being simple, strongly reduced and non-terminal outside. Q.E.D. (Lemma 3.9)

In Figure 3.10, we give another simpler grammar which is not non-terminal outside and for which the generated graph is obtained by parallel rewritings with multiplicity. We can transform any grammar into an equivalent simple non-terminal outside grammar.

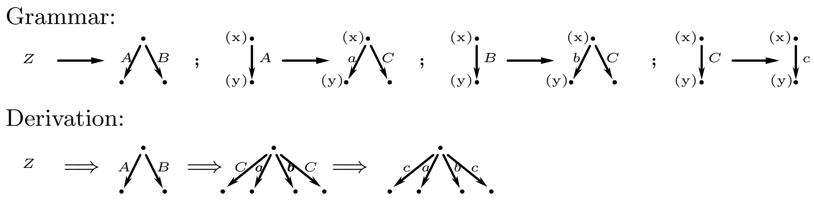


Figure 3.11. Transformation of the grammar of Figure 3.10.

Proposition 3.10. Any regular hypergraph can be generated in an effective way by a non-terminal outside, simple, connected and strongly reduced grammar.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Lemma 3.8, G is generated by a simple, strict, connected and strongly reduced grammar R from its axiom Z. Recall that a connected grammar is proper.

We transform R by incrementing the arity of non constant non-terminal hyperarcs. For each non-terminal $A \in N_R - F_0$, we take a new symbol A' of arity $\varrho(A') = \varrho(A) + 1$. For each $(X, H) \in R$ with $X \notin F_0$, there exists a vertex $x_X \in V_H - V_X$ because R is strict, and we define the hyperarc

$$X' := X(1)'X(2)\ldots X(|X|)x_X.$$

For each $H \in \mathrm{Im}(R)$ and for each $Y \in H$, we define the following hyperarc:

$$Y' := \begin{cases} Y & \text{if } Y(1) \notin N_R - F_0 \\ Y(1)'Y(2)\ldots Y(|Y|)y_Y & \text{if } Y(1) \in N_R - F_0; \end{cases}$$

where y_Y is a new vertex (not in R with $y_Y \neq y_Z$ for $Y \neq Z$). By union, we extend to $H' := \{Y' \mid Y \in H\}$.

It remains to take

$$S := \{(X, H') \in R \mid (X, H) \in R \wedge X \in F_0\} \\ \cup \{(X', H') \mid (X, H) \in R \wedge X \notin F_0\}.$$

The grammar S remains simple, connected and strongly reduced of axiom Z. And S is non-terminal outside and generates G. q.e.d. (Proposition 3.10)

The transformation of Proposition 3.10 is illustrated in Figure 3.11.

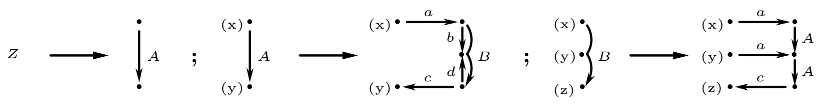


FIGURE 3.12. From the grammar of Figure 2.4 to a terminal outside one.

3.3 Separating the inputs with the outputs

We want to extend Proposition 3.10 by separating as much as possible in every right hand side of the grammar input and output vertices. However we can observe that if a vertex of a left hand side X is of infinite degree in $R^\omega(X)$ then it must be also an output. We shall show that a grammar can be transformed into an equivalent one such that the non-output vertices of every left hand side X are the inputs of finite degree in $R^\omega(X)$.

A grammar R is *terminal outside* if for any rule $X \longrightarrow H$, any terminal hyperarc $Y \in H$ with $Y(1) \in T_R$ has a vertex which is not an input: $V_Y - V_X \neq \varnothing$. An *outside grammar* is a terminal outside and non-terminal outside grammar.

Lemma 3.11. Any regular hypergraph can be generated in an effective way by an outside, simple, connected and strongly reduced grammar.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Lemma 3.1, G is generated by a reduced grammar R from its axiom Z. By least fixed point, we define the grammar I such that $\mathrm{Dom}(I) = \mathrm{Dom}(R)$ and

$$I = \{(X, H \cap T_R V_X^*) \mid X\ R\circ \underset{I}{\Longrightarrow} H\}.$$

We define grammars

$$J := \{(X, H \cup \{X\}) \mid (X,H) \in I \wedge X \neq Z\}, \text{ and}$$

$$S := \{(Z,H) \mid Z\ R\circ \underset{J}{\Longrightarrow} H\} \cup \{(X, H_{|-T_R V_X^*}) \mid X\ R\circ \underset{J}{\Longrightarrow} H\}.$$

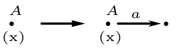
For any $X \in \mathrm{Dom}(R) - \{Z\}$,

$$S^\omega(X) = \{K - T_R V_X^* \mid K \in R^\omega(X)\}$$

hence $S^\omega(Z) = R^\omega(Z)$. Furthermore S is terminal outside but not necessary reduced (due to condition (iii)).

By applying the previous constructions, the obtained grammar remains terminal outside and becomes non-terminal outside, simple, connected and strongly reduced. Q.E.D. (Lemma 3.11)

Grammar:



Graph generated from its unique non-terminal:

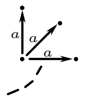


Figure 3.13. A regular graph of infinite degree.

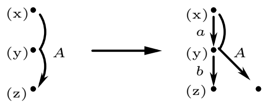


Figure 3.14. A grammar which is not degree-outside.

In Figure 3.12, we apply the construction of the proof of Lemma 3.11 to the grammar of Figure 2.4 completed with the rule $Z \longrightarrow Axy$. In the last figure of Section 2 and in Figure 3.9, we have regular graphs with vertices of infinite degree. In Figure 3.13, we give another regular graph of infinite degree. We shall see that there is no regular hypergraph of finite degree which is not of bounded degree. To compute the vertex degrees of a hypergraph, we separate in the right hand sides of a grammar the outputs from the inputs of finite degree. A *degree-outside* grammar R is a grammar such that the vertices of any right hand side which are inputs and outputs are the input vertices of infinite degree in the generated graph:

$$\forall (X,H) \in R \; \forall Y \in H \cap N_R V_H^* (V_X \cap V_Y \subseteq \{s \in V_X \mid \mathrm{d}_{R^\omega(X)}(s) = \omega\}).$$

The grammar of Figure 3.13 is degree-outside but the grammar in Figure 3.14 is not: x is both an input and an output but is of finite degree in the generated graph. A degree-outside and reduced grammar generating a hypergraph of finite degree is called an *input-separated grammar*: for each right hand side, any input is not an output. A grammar which is outside and degree-outside is a *complete outside grammar*.

Theorem 3.12. Any regular hypergraph can be generated in an effective way by a complete outside, simple, connected and strongly reduced grammar.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Lemma 3.11, G is generated by an outside, simple, connected and strongly reduced grammar R from its axiom Z. For any hypergraph H and any $P \subseteq V_H$, we denote

$$[H,P] := |\{(Y,i) \mid Y \in H \wedge Y(1) \in N_R \wedge 2 \leq i \leq |Y| \wedge Y(i) \in P\}|$$

the number of non-terminal links in H on vertices in P.

To get from R a degree-outside grammar, we derive each right hand side until we cannot separate outputs from inputs. We begin with the initial grammar $S_0 := R$; having constructed a grammar S_n with $n \geq 0$, we associate to each rule $(X,H) \in S_n$ a hypergraph K_X such that

$$H \underset{S_n}{\longrightarrow} K_X \wedge [K_X, V_X] < [H, V_X]$$

if such a hypergraph exists, otherwise $K_X = H$; and we define the grammar

$$S_{n+1} := \{(X, K_X) \mid X \in \mathrm{Dom}(R)\}.$$

We finish with the grammar

$$S := S_m \text{ for } m = \min\{n \mid S_n = S_{n+1}\}.$$

This grammar S is complete outside, simple, connected and generates G from Z. And S becomes strongly reduced by restriction to the accessible non-terminals from Z. Q.E.D. (Theorem 3.12)

Note that the transformation of Theorem 3.12 applied directly to the grammar of Figure 3.14 which is not terminal outside, and completed with the rule $Z \longrightarrow \{A123\}$, leaves the grammar unchanged. In Figure 3.15, we apply the transformation of Theorem 3.12 to a suitable grammar. The transformation of Theorem 3.12 is illustrated in Figure 3.16. The regular graph of Figure 3.16 has only two possible vertex degrees: 3 and ω. Let us generalize this property.

Proposition 3.13.

a) Any regular hypergraph has a finite number of vertex degrees, hence is either of infinite degree or of bounded degree.

b) The class of regular hypergraphs is closed under colouring of vertices whose degree belongs to a given subset of $\mathbb{N} \cup \{\omega\}$.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Theorem 3.12, G is generated by a complete outside, simple, connected and strongly reduced grammar R from its axiom Z. We can assume that the non-input vertices of the right hand sides are distinct:

$$\forall (X,H),(Y,K) \in R \text{ with } X \neq Y ((V_H - V_X) \cap (V_K - V_Y) = \varnothing)$$

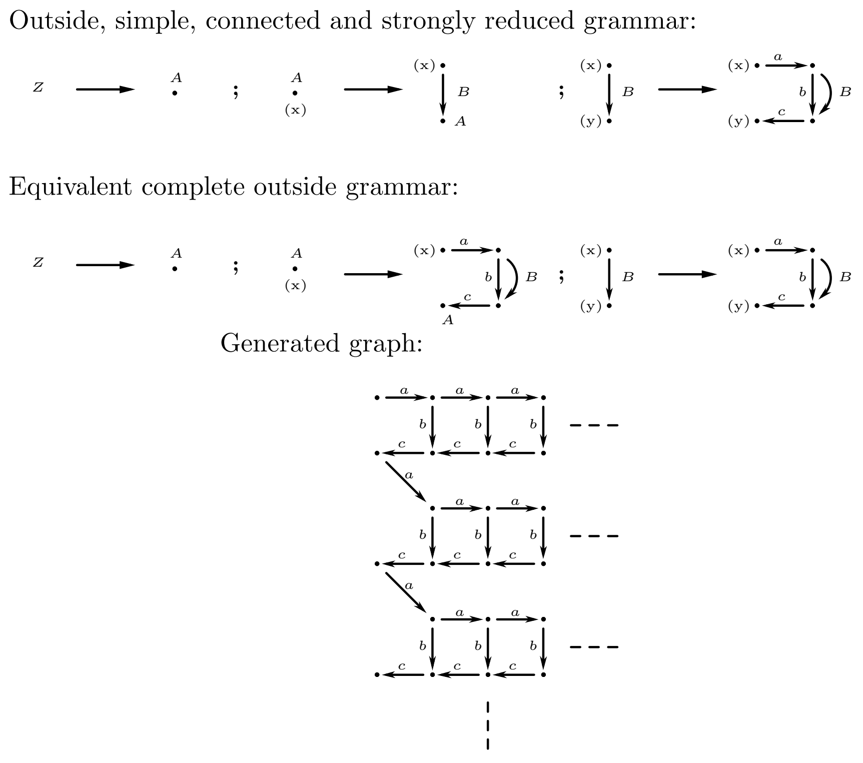

Outside, simple, connected and strongly reduced grammar:
Z
A
(x)
(y)
B
a
b
c
Equivalent complete outside grammar:
Generated graph:

FIGURE 3.15. Transformation of Theorem 3.12.

and we denote by E the finite set of non-input vertices in R:

$$E := \bigcup \{V_H - V_X \mid (X,H) \in R\}.$$

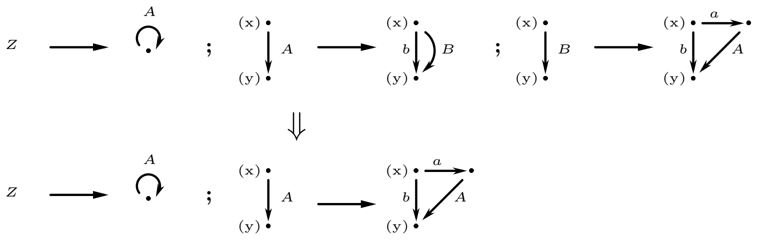
Let us prove Property a). For each rule $(X,H) \in R$, we take a hypergraph K such that $H \underset{R}{\Longrightarrow} K$ and for every vertex $s \in V_H - V_X$, we define

$$d(s) := \begin{cases} \omega & \text{if } \exists Y \in K(Y(1) \in N_R \wedge s \in V_Y) \\ \mathrm{d}_{[K]}(s) & \text{otherwise.} \end{cases}$$

The vertex degrees of G form the set $\{d(s) \mid s \in E\}$ which is finite and computable.

Let us prove Property b). Let $P \subseteq \mathbb{N} \cup \{\omega\}$ and # a colour. We want to construct a grammar generating

$$G_P := G \cup \{\#s \mid s \in V_G \wedge d(s) \in P\}.$$



generating the graph:

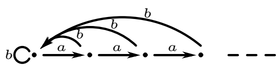


FIGURE 3.16. Transformation of a grammar into a degree-outside one.

To each rule $(X, H) \in R$, we associate the hypergraph:

$$H' := H \cup \{\#s \mid s \in V_H - V_X \wedge \mathrm{d}_{R^\omega(H)}(s) \in P\}.$$

So the grammar $\{(X, H') \mid (X, H) \in R\}$ generates G_P from Z.

Q.E.D. (Proposition 3.13)

3.4 Separating the outputs

This last normalization subsection permits to get grammars separating for each right hand side the vertices of the non-terminals. A grammar R is *output-separated* if for any right hand side, distinct non-terminal hyperarcs have no common vertex and any non-terminal hyperarc has distinct vertices: for any $H \in \mathrm{Im}(R)$ and any $X, Y \in H \cap N_R V_H^*$,

$$|V_X| = \varrho(X(1)) \wedge (X \neq Y \Rightarrow V_X \cap V_Y = \varnothing).$$

Note that any output-separated grammar is without multiplicity. Theorem 3.12 cannot be extended to get grammars which are also output-separated. However we give a general sufficient condition on any reduced grammar R that allows to transform R into an equivalent output-separated grammar. To any hypergraph H labelled in $N_R \cup T_R$, we denote

$$\mathrm{Comp}(H) := \{[C]_\sim \mid C \text{ connected component of } H_{|-T_R V_H^*}\}$$

the family of the connected components (up to isomorphism) of the set of non-terminal hyperarcs of H, and

$$\mathrm{Comp}(R) := \bigcup \{\mathrm{Comp}(H) \mid Z \underset{R}{\Longrightarrow}^* H\}.$$

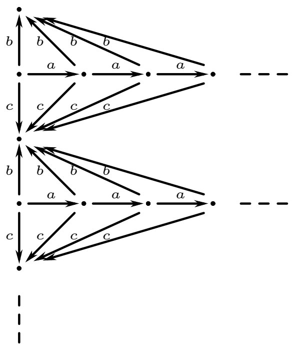


FIGURE 3.17. Regular graph not given by an output-separated grammar.

We say that R is *output-separable* if Comp(R) is finite. This notion is illustrated in Figure 3.18. Any input-separated grammar R (reduced and degree-outside grammar with Gen(R) of finite degree) is output-separable:

$$\mathrm{Comp}(R) = \{\{Z\}\} \cup \{[C]_\sim \mid \exists H \in \mathrm{Im}(R)\ (C \text{ connected component of } H_{|-T_R V_H^*})\}.$$

Any graph generated by an output-separable grammar can be generated by an output-separated grammar.

Lemma 3.14. Any output-separable grammar can be transformed into an equivalent output-separated grammar.

Proof. Let R be any output-separable grammar: R is reduced and Comp(R) is finite. Denoting m the cardinality of Comp(R), we take hypergraphs $H_1, \ldots, H_m$ such that

$$\{[H_1]_\sim, \ldots, [H_m]_\sim\} = \mathrm{Comp}(R).$$

The axiom Z of R satisfies $[Z]_\sim = \{Z\}$ hence $Z \in \{H_1, \ldots, H_m\}$. For each $1 \le i \le m$, we take a new symbol A_i of arity $\varrho(A_i) = |V_{H_i}|$, we denote

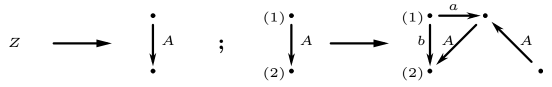
$$\{s_{i,1}, \ldots, s_{i,\varrho(A_i)}\} = V_{H_i}$$

we take a hypergraph K_i such that $H_i \underset{R}{\Longrightarrow} K_i$ and let

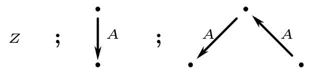
$$C_{i,1}, \ldots, C_{i,n_i} \text{ be the connected components of } (K_i)_{|-T_R V_{K_i}^*}.$$

By definition of Comp(R) and for every $1 \le i \le m$ and $1 \le j \le n_i$, there is a unique $1 \le i_j \le m$ such that $C_{i,j}$ is isomorphic to H_{i_j}, and we take an

Output-separable grammar R:



Comp(R):



Non output-separable grammar:

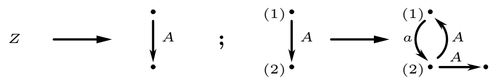


FIGURE 3.18. Output separation for grammars.

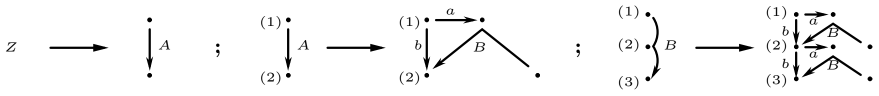


FIGURE 3.19. Output-separated grammar from the first grammar of Figure 3.18.

isomorphism $h_{i,j}$ from H_{i_j} to $C_{i,j}$: $H_{i_j} \overset{h_{i,j}}{\sim} C_{i,j}$. We define the grammar S having for each $1 \leq i \leq m$, the following rule:

$$A_i s_{i,1} \ldots s_{i,\varrho(A_i)} \longrightarrow [K_i] \cup \{A_{i_j} h_{i,j}(s_{i_j,1}) \ldots h_{i,j}(s_{i_j,\varrho(A_{i_j})}) \mid 1 \leq j \leq n_i\}.$$

So S is output-separated and $S^\omega(Z) = R^\omega(Z)$. Q.E.D. (Lemma 3.14)

The construction of the proof of Lemma 3.14 is illustrated in Figure 3.19. Lemma 3.14 permits to extend Theorem 3.12 to any regular graph of finite degree.

Corollary 3.15. Any regular hypergraph of finite degree can be generated in an effective way by a grammar which is input and output separated, connected and strongly reduced.

By generation by distance from a vertex of any connected regular graph of finite degree, we shall get in next section a grammar normal form stronger than in Corollary 3.15 (cf. Theorem 4.6). The condition of a grammar to be output-separable is effective.

Lemma 3.16. We can decide whether a reduced grammar is output-separable.

Proof. Left as a simple exercise on grammars. Q.E.D.

Henceforth and considering Proposition 3.10, we assume that any grammar is reduced, proper and without multiplicity.

3.5 Canonical regular graphs

A grammar R generates from a hypergraph K a family $R^\omega(K)$ of isomorphic hypergraphs. We present here a canonical way to extract a representant $\mathrm{Gen}(R,K)$ in this family. A vertex s of $\mathrm{Gen}(R,K)$ is the word of the path of the non-terminals plus the non-input vertex which are used to get s by rewritings. Up to a label renaming with adding rules, we assume that K and each right hand side of R has no two non-terminal hyperarcs with the same label: for every $H \in \{K\} \cup \mathrm{Im}(R)$,

$$Y, Y' \in H \wedge Y \neq Y' \wedge Y(1), Y'(1) \in N_R \Longrightarrow Y(1) \neq Y'(1).$$

We denote by V_R the vertex set of K plus the set of non input vertices of the right hand sides of R:

$$V_R := V_K \cup \bigcup \{V_H - V_X \mid (X,H) \in R\}.$$

To each word $u \in N_R^*$, we associate for each non-terminal $A \in N_R$ a new symbol A_u of arity $\varrho(A)$, and for each hyperarc X, we define

$$X_u := \begin{cases} X & \text{if } X(1) \notin N_R \\ X(1)_u X(2) \dots X(|X|) & \text{if } X(1) \in N_R, \end{cases}$$

that we extend by union to any hypergraph H: $H_u := \{X_u \mid X \in H\}$. To each rule $(X,H) \in R$ and hyperarc Y with $Y(1) = X(1)_u$, $u \in N_R^*$ and $V_Y \subset N_R^* V_R$, we associate the finite hypergraph $\widehat{Y} := h(H)_{uX(1)}$ where h is the function defined for every vertex $r \in V_H$ by

$$h(r) := \begin{cases} Y(i) & \text{if } r = X(i) \text{ for some } 2 \leq i \leq |X| \\ uX(1)r & \text{otherwise.} \end{cases}$$

Beginning with the hypergraph $H_0 = K_\varepsilon$ and having defined H_n for $n \geq 0$, we construct

$$H_{n+1} := [H_n] \cup \{\widehat{Y} \mid Y \in H_n \wedge \exists u \in N_R^* (Y(1) \in (N_R)_u)\}$$

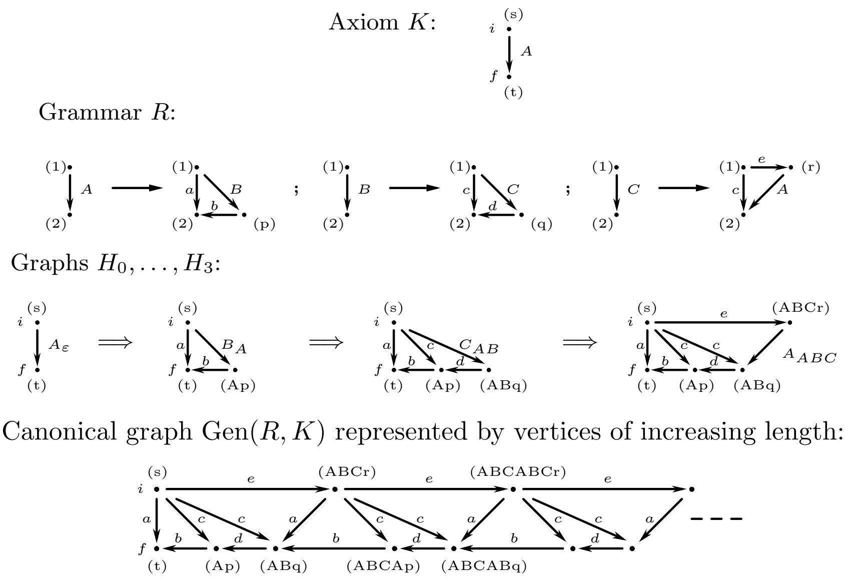


FIGURE 3.20. Canonical graph generated by a grammar.

in order to define the following terminal hypergraph:

$$\mathrm{Gen}(R,K) := \bigcup_{n\geq 0} [H_n].$$

Such a hypergraph is generated by R from K: $\mathrm{Gen}(R,K) \in R^{\omega}(K)$. In Figure 3.20, we illustrate the previous construction.

The vertex set of $\mathrm{Gen}(R,K)$ is regular because

$$V_{\mathrm{Gen}(R,K)} = V_K \cup \bigcup \{L_A \mid A \in F_K \cap N_R\}$$

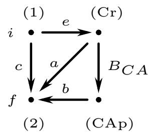
where the family of languages L_A for all $A \in N_R$ is the least fixed point of the following system: for each $(X,H) \in N_R$,

$$L_{X(1)} = X(1).\big((V_H - V_X) \cup \bigcup \{L_A \mid A \in F_H \cap N_R\}\big).$$

For the grammar R of Example 3.20, we have

$$L_A = A(\{p\} \cup L_B); \quad L_B = B(\{q\} \cup L_C); \quad L_C = C(\{r\} \cup L_A)$$

hence $V_{\mathrm{Gen}(R)} = \{s,t\} \cup L_A = \{s,t\} \cup (ABC)^*\{Ap, ABq, ABCr\}$



$$\begin{tikzcd}
\overset{(1)}{i} \arrow[r, "e"] \arrow[d, "c"'] & (Cr) \arrow[d, "B_{CA}"] \arrow[dl, "a"'] \\
\underset{(2)}{f} & (CAp) \arrow[l, "b"]
\end{tikzcd}$$

FIGURE 3.21. $\mathrm{Gen}_{CAp}(R, C12)$ for the grammar R of Figure 3.20.

For any non-empty finite $\varnothing \neq E \subseteq V_{\mathrm{Gen}(R,K)}$, we define the *least approximant* $\mathrm{Gen}_E(R, K)$ of $\mathrm{Gen}(R, K)$ whose vertex set contains E, which is the hypergraph obtained from K by a minimal number of rewritings to generate all vertices in E. Precisely we begin with $H_0 = K_\varepsilon$; having defined H_n for $n \geq 0$, either we can choose $Y \in H_n$ with $Y(1) \in (N_R)_u$ for some $u \in (N_R)^*$ such that $E \cap u(N_R)^+V_R \neq \varnothing$, and we take $H_{n+1} = (H_n - \{Y\}) \cup \widehat{Y}$ or if such a Y does not exist, we finish with $\mathrm{Gen}_E(R, K) = H_n$. In Figure 3.21, the least approximant of $\mathrm{Gen}(R, C12)$ containing $E = \{CAp\}$ is depicted, where R is taken from Figure 3.20. Note that the hypergraphs H_n given to define $\mathrm{Gen}(R, K) = \bigcup_{n\geq 0}[H_n]$ are approximants:

$$H_n = \mathrm{Gen}_{E_n}(R, K) \text{ for } E_n = \{v \in V_{\mathrm{Gen}(R,K)} \mid |v| \leq n + 1\}.$$

The *canonical graph* of a reduced grammar R of axiom Z is $\mathrm{Gen}(R) := \mathrm{Gen}(R, Z)$.

4 Generation by distance

In the previous section, we have considered transformations of grammars into equivalent normalized grammars. We now investigate transformations to get grammars generating hypergraphs by vertices of increasing distance from a given colour, either by accessibility or by non-oriented accessibility.

4.1 Regularity by restriction

The regularity of a graph is preserved by restriction to the vertices having a given colour.

Proposition 4.1. The class of regular hypergraphs is closed under restriction to the vertices having a colour in a given set.

Proof. Let $G \neq \varnothing$ be a regular hypergraph. By Theorem 3.12, G is generated by an outside, simple, connected and strongly reduced grammar R from its axiom Z. Let P be a colour set. We want to construct a grammar generating

$$G_P := G_{|\{s \in G \mid \exists c \in P (cs \in G)\}}.$$

We can restrict P to a unique colour #, otherwise we take a new colour d to colour all the vertices of G having a colour in P, then we do the restriction of

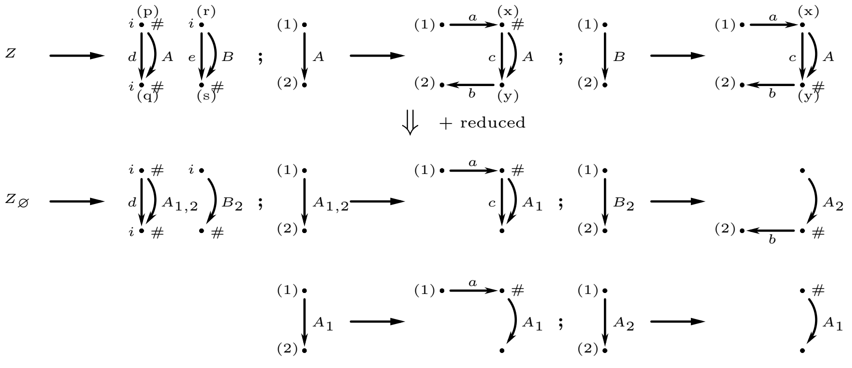


FIGURE 4.1. Grammar transformation for the restriction to colour #.

G to the vertices coloured by d and we remove this colour. To each $A \in N_R$ and each $I \subseteq [\varrho(A)]$, we associate a new non-terminal A_I of arity $\varrho(A)$. For each rule $(X, H) \in R$ and $I \subseteq [\varrho(X(1))]$, we define the hypergraph

$$\begin{aligned} H_I := \{Y \in H \mid Y(1) \in T_R \wedge \forall 1 < i \leq |Y| (\#Y(i) \in H \vee Y(i) \in [X, I])\} \\ \cup \{B_J Y \mid B \in N_R \wedge BY \in H \wedge J = \{j \mid 1 \leq j \leq |Y| \wedge \\ (\#Y(j) \in H \vee Y(j) \in [X, I])\}\} \end{aligned}$$

with $[X, I] := \{X(i+1) \mid i \in I\}$. Thus the grammar

$$\{(A_I X, H_I) \mid A \in N_R \wedge (AX, H) \in R \wedge I \subseteq [\varrho(A)]\}$$

generates $G_{\#}$ from $Z_{\varnothing}$. Q.E.D. (Proposition 4.1)

By Propositions 3.13 and 4.1, the regular graphs are closed by restriction to a given set of degrees. The construction of the proof of Proposition 4.1 is illustrated in Figure 4.1.

4.2 Regularity by graduation

A *graduation* g of a hypergraph G is a mapping from V_G into $\mathbb{N}$ such that only finitely many vertices have the same value by g *i. e.* g^{-1} is locally finite: $g^{-1}(n) = \{s \in V_G \mid g(s) = n\}$ is finite for every $n \geq 0$. We shall define the regularity of a hypergraph by vertices of increasing graduation.

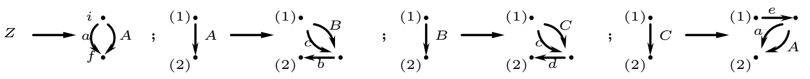
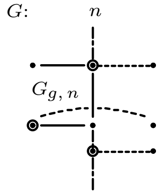


FIGURE 4.2. Generating the graph of Figure 3.20 by (length −2).

Precisely for every $n \geq 0$, we denote

$$\begin{aligned} G_{g,n} &:= G_{|\{s|g(s)\leq n\}} \\ &= \{X \in G \mid g(X(2)) \leq n \wedge \ldots \wedge g(X(|X|)) \leq n\} \\ \text{and } \partial_{g,n}G &:= \{s \mid g(s) \leq n \wedge \\ &\qquad \exists X \in G(s \in V_X \wedge \exists t \in V_X(g(t) > n))\} \\ &= \{s \in V_{G-G_{g,n}} \mid g(s) \leq n\} \end{aligned}$$

the nth *frontier* of G by g. This is illustrated by the following diagram:



where $G_{g,n}$ contains all edges depicted by a full line and $\partial_{g,n}G$ is the set of circled vertices; note that

$$V_{G_{g,n}} \cap V_{G-G_{g,n}} \subseteq \partial_{g,n}G.$$

We say that a hypergraph G is *regular by g* if there exists a terminal outside grammar R such that for every $n \geq 0$, R generates from its axiom Z by $n+1$ parallel rewritings the hypergraph $G_{g,n}$ of terminal hyperarcs, plus a set of non-terminal hyperarcs of vertex set $\partial_{g,n}G$

$$\forall n \geq 0\ \exists H(Z \underset{R}{\Longrightarrow}^{n+1} H \wedge [H] = G_{g,n} \wedge V_{H-[H]} = \partial_{g,n}G);$$

we also say that R generates G according to g. Observe that if G is connected and $G_{g,m} \neq \varnothing$, we have for $n \geq 0$,

$$\partial_{g,m+n}G = \varnothing \Longleftrightarrow G_{g,m+n} = G.$$

When V_G is a language, then word length may be used as a graduation. For instance, the canonical graph of Figure 3.20 is regular by length.

Proposition 4.2. Any canonical hypergraph is regular by length.

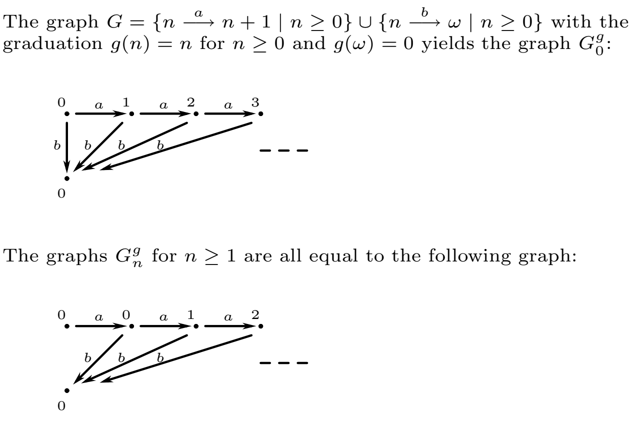

The graph $G = \{n \xrightarrow{a} n+1 \mid n \geq 0\} \cup \{n \xrightarrow{b} \omega \mid n \geq 0\}$ with the graduation $g(n) = n$ for $n \geq 0$ and $g(\omega) = 0$ yields the graph G_0^g:
0
1
2
3
a
a
a
b
b
b
b
0
The graphs G_n^g for $n \geq 1$ are all equal to the following graph:
0
0
1
2
a
a
a
b
b
b
0

FIGURE 4.3. Graph decomposition.

Proof. Let R be a grammar. Let us construct a grammar generating $\mathrm{Gen}(R)$ by length. By Lemma 3.11, we get a terminal outside grammar S with the same canonical hypergraph: $\mathrm{Gen}(S) = \mathrm{Gen}(R)$. As S is outside, S generates $\mathrm{Gen}(S)$ by length minus 2. By denoting Z the axiom of S and by adding two new constant symbols Z_0, Z_1, we complete S into $S \cup \{(Z_0, Z_1), (Z_1, Z)\}$ which generates $\mathrm{Gen}(S)$ by length (from Z_0). Q.E.D. (Proposition 4.2)

Proposition 4.2 implies that any regular hypergraph is regular by some graduation. A dual way to express the regularity by graduation is by decomposition: we remove iteratively on the graph the vertices with graduation less than $1, 2, \ldots$. The decomposition allows to avoid the explicit use of grammars. The *decomposition* at level $n \geq 0$ of a hypergraph G by a graduation g is the following hypergraph:

$$G_n^g := (G - G_{g,n-1}) \cup \{\max\{0, g(s) - n\}s \mid s \in V_{G - G_{g,n-1}}\}$$

obtained from G by removing $G_{g,n-1}$ with $G_{g,-1} = \varnothing$ and by colouring any remaining vertex s by the integer $\max\{0, g(s) - n\}$ (assuming that G has no integer colour otherwise we must use a new integer colour: p' for each $p \geq 0$). In particular $G_0^g = G \cup \{g(s)s \mid s \in V_G\}$. We give an example in Figure 4.3.

We say that a hypergraph G is *finitely decomposable* by a graduation g if the disjoint union

$$\sum_{n \geq 0} G_n^g := \{X(1)(X(2), n) \ldots (X(|X|), n) \mid n \geq 0 \wedge X \in G_n^g\}$$

only has a finite number of non-isomorphic connected components. For instance the graph G of Figure 4.3 is finitely decomposable by its graduation g with only two non isomorphic connected components: G_0^g and G_1^g. Another example is the complete binary tree

$$T := \{u \xrightarrow{a} ua \mid u \in \{a,b\}^*\} \cup \{u \xrightarrow{b} ub \mid u \in \{a,b\}^*\}$$

which is finitely decomposable by length: $T_0^{|\,|}$ and for every $n \geq 1$, any connected component of $T_n^{|\,|}$ is isomorphic to $T_1^{|\,|}$.

A last example is the semiline $\mathbb{N}$ which is regular but is not finitely decomposable using the graduation associating to $n \geq 0$ the $n+1$st prime number. By definition the vertices of G_n^g coloured by 0 are vertices of G coloured by some $i \leq n$:

$$\{s \mid 0s \in G_n^g\} \subseteq \{s \in V_G \mid g(s) \leq n\}$$

which is finite. In particular any connected component C of $\sum_{n\geq 0} G_n^g$ has a finite set $V_{C,0} = \{s \in V_C \mid 0s \in C\}$ of vertices coloured by 0. So any hypergraph G finitely decomposable by g is *bounded connected* by g in the following sense:

- there exists $b \geq 0$ such that $|V_{C,0}| \leq b$ for every connected component C of $\sum_{n\geq 0} G_n^g$,

or, equivalently,

- there exists $b \geq 0$ such that $|\{s \in V_C \mid g(s) \leq n\}| \leq b$ for all $n \geq 0$ and for every connected component C of $G - G_{g,n-1}$.

It follows that finite decomposition is a less powerful notion than regularity (by some graduation). The regular graph G of Figure 3.17 has no finite decomposition because it is not bounded connected by any graduation g: the decomposition G_n^g at any level n has a connected component containing all the vertices of infinite (in-)degree.

The finite decomposition of a hypergraph G by a graduation g also imposes that G has finitely many connected components. It is due to the fact that G_0^g has a finite number of non isomorphic connected components, and no connected component can be infinitely repeated because g is locally finite.

Any finite decomposition can be done by a grammar generating what we remove; the converse is true when the hypergraph is bounded connected by the graduation and has only a finite number of connected components.

Proposition 4.3. Given a graduation g of a hypergraph G, G is finitely decomposable by g if and only if G is regular by g and G is bounded connected with finitely many connected components.

Graph $G = \{(m,n) \xrightarrow{a} (m,n+1) \mid m,n \geq 0\}$

Graduation $g(m,n) = m+n$

G_0^g:

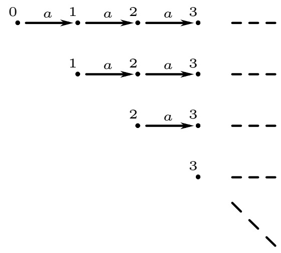


FIGURE 4.4. Graph regular by graduation, bounded connected, but not finitely decomposable.

Proof. $\Longrightarrow$: Let G be a hypergraph finitely decomposable by a graduation g. As already mentioned, G is bounded connected by g and G has only a finite number of connected components. It remains to be shown that G is regular by g. Recall that for any $n \geq 0$,

$$G_n^g := (G - G_{g,n-1}) \cup \{\max\{0, g(s)-n\}s \mid s \in V_{G-G_{g,n-1}}\}$$

with $G_{-1}^g = \varnothing$. We define

$$\widehat{G}_n^g := (G - G_{g,n}) \cup \{\max\{0, g(s)-n\}s \mid s \in V_{G-G_{g,n}}\}$$

obtained from G_n^g by removing the hyperarcs whose vertices are all coloured by 0 (only a finite number) and then by removing the isolated vertices coloured by 0. Let E be a maximal set of non-isomorphic connected components of $\{\widehat{G}_n^g \mid n \geq 0\}$. By hypothesis $\{G_n^g \mid n \geq 0\}$ has a finite number of non-isomorphic connected components, hence E is finite. For each $C \in E$, we order the set $V_{C,0}$ of vertices of C coloured by 0:

$$\{\langle C,1\rangle, \ldots, \langle C, |V_{C,0}|\rangle\} = V_{C,0},$$

and we take a new symbol $[C]$ of arity $|V_{C,0}|$. Note that for every $n \geq 0$,

$$G_{n+1}^g = \left(\widehat{G}_n^g - \{cs \in \widehat{G}_n^g \mid c \in \mathbb{N}\}\right) \cup \{\max\{0, c-1\}s \mid cs \in \widehat{G}_n^g \wedge c \in \mathbb{N}\}.$$

To each $C \in E$, we associate the hypergraph

$$C' := (C - \mathbb{N}V_C) \cup \{\max\{0, c-1\}s \mid cs \in C \wedge c \in \mathbb{N}\}$$

which is isomorphic to a connected component of $\{G^g_n \mid n \geq 0\}$, and we define

$$E' := \{C' \mid C \in E\} \cup \{G^g_0\}.$$

For each $C \in E'$, the connected components of

$$C - \{X \in C \mid V_X \subseteq V_{C,0} \wedge X(1) \notin \mathbb{N}\}$$

and not reduced to a vertex coloured by 0, are denoted by $C_1, \ldots, C_{n_C}$. For each $1 \leq i \leq n_C$, there is an isomorphism h_i from C_i to a unique $D_i \in E$. To each $C \in E'$, we associate the hypergraph

$$\begin{aligned}\langle\!\langle C \rangle\!\rangle := \{X \in C \mid X(1) \notin \mathbb{N} \wedge V_X \subseteq V_{C,0}\}\\ \cup \{[D_i]h_i^{-1}(\langle D_i, 1\rangle) \ldots h_i^{-1}(\langle D_i, |V_{D_i,0}|\rangle) \mid 1 \leq i \leq n_C\}.\end{aligned}$$

Finally the following outside grammar:

$$R := \{(Z, \langle\!\langle G^g_0 \rangle\!\rangle)\} \cup \{([C]\langle C, 1\rangle \ldots \langle C, |V_{C,0}|\rangle, \langle\!\langle C' \rangle\!\rangle) \mid C \in E\}$$

generates G from Z and according to g.

$\Longleftarrow$: Assume that G is regular by g, bounded connected by g and has a finite number of connected components. We want to show that G is finitely decomposable by g. We can assume without loss of generality that G only has one connected component,*i. e.* that G is connected.

There exists an integer b such that for any connected component C of $\sum_{n\geq 0} G^g_n$, $|V_{C,0}| \leq b$. Consider an outside grammar R generating G by g from its axiom Z. By the transformation of Lemma 3.3 splitting any hyperarc into connected hyperarcs, we can assume that R is connected. Consider an infinite parallel derivation

$$Z \underset{R}{\Longrightarrow} H_0 \ldots H_n \underset{R}{\Longrightarrow} H_{n+1} \underset{R}{\Longrightarrow} \ldots$$

For every $n \geq 0$, we have

$$[H_n] = G_{g,n} \text{ and } V_{H_n - [H_n]} = \partial_{g,n} G$$

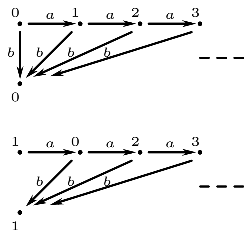
hence

$$\begin{aligned}\{s \mid 0s \in G^g_n\} &= \{s \in V_{G - G_{g,n-1}} \mid g(s) \leq n\}\\ &\supseteq \{s \in V_{G - G_{g,n}} \mid g(s) \leq n\}\end{aligned}$$

thus

$$V_{H_n - [H_n]} = \partial_{g,n} G = \{s \in V_{G - G_{g,n}} \mid g(s) \leq n\} \subseteq \{s \mid 0s \in G^g_n\}.$$

Gradued graph of finite decomposition



Grammar:

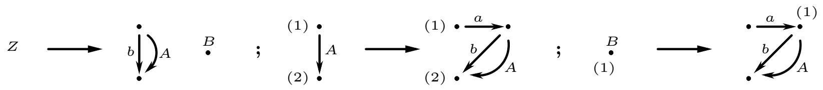


FIGURE 4.5. Grammar for a finitely decomposable graph.

Then for any $n \geq 0$ and any connected component K of $H_n - [H_n]$, $|V_K| \leq b$. It follows that $\{H_n - [H_n] \mid n \geq 0\}$ has a finite number of non isomorphic connected components, and we take a maximal set E of non isomorphic connected components. Consequently E is finite and the $R^\omega(K)$ for any $K \in E$ are, up to isomorphism, the connected components of $\{G - G_{g,n} \mid n \geq 0\}$.

For each $K \in E$, we take $K = K_0 \underset{R}{\Longrightarrow} K_1 \ldots K_n \underset{R}{\Longrightarrow} K_{n+1} \underset{R}{\Longrightarrow} \ldots$ a derivation generating the hypergraph $K' := \bigcup_{n \geq 0}[K_n]$ which we complete by an integer colouring as follows:

$$\overline{K} := K' \cup \{\min\{n \mid s \in V_{K_{n+1}}\}s \mid s \in V_{K'}\}$$

So $\{\overline{K} \mid K \in E\}$ are up to isomorphism the connected components of $\{G_n^g \mid n > 0\}$. Hence G is finitely decomposable by g. Q.E.D. (Proposition 4.3)

The transformation of the necessary condition of Proposition 4.3 is illustrated in Figure 4.5.

4.3 Regularity by accessibility

A usual problem in graph theory is the accessibility problem. This problem consists in computing the set of vertices accessible from a given initial set. Here we transform any grammar into another one generating the same graph plus a colouring of the vertices accessible from (vertices with) a given colour (cf. Proposition 4.4). This grammar transformation is expressed by least

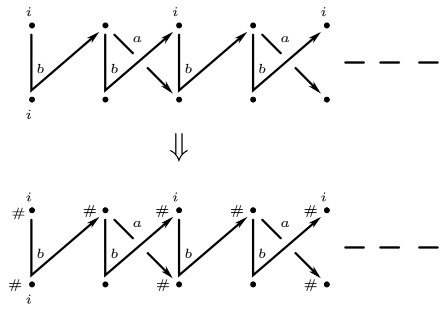


FIGURE 4.6. Computation of the vertices accessible from i.

fixpoint on the grammar. Finally we give a rooted regular graph of finite degree which cannot be generated by accessibility.

The *accessible vertex* set $\mathrm{Acc}(G,i)$ of a hypergraph G from a colour i is the smallest subset of V_G containing the set $V_{G,i}$ of vertices coloured by i and closed under the following accessibility property:

$$fv_1 \ldots v_{\varrho(f)} \in G \wedge \varrho(f) > 1 \wedge v_1,\ldots,v_{\varrho(f)-1} \in \mathrm{Acc}(G,i) \\ \Longrightarrow v_{\varrho(f)} \in \mathrm{Acc}(G,i)$$

Equivalently $\mathrm{Acc}(G,i)$ is the least solution of the following equation:

$$\mathrm{Acc}(G,i) = V_{G,i} \cup \mathrm{Succ}_G(\mathrm{Acc}(G,i))$$

for the following *successor* relation:

$$\mathrm{Succ}_G(E) := \{v \mid FE^+v \cap G \neq \varnothing\} \text{ for any } E \subseteq V_G.$$

So a hyperarc realises an “and” boolean function: we access via a hyperarc $fv_1 \ldots v_{\varrho(f)}$ its last vertex $v_{\varrho(f)}$ if we have accessed all its other vertices $v_1,\ldots,v_{\varrho(f)-1}$. A hypergraph G is *accessible* from a colour i if $\mathrm{Acc}(G,i) = V_G$. For instance the hypergraph $G = \{fxyz, gxy, hx, c\}$ of Figure 2.1 is accessible from h: $\mathrm{Acc}(G,h) = \{x,y,z\}$, but the hypergraph $G = \{ix, jy\}$ is not accessible from a unique colour.

We say that a vertex r of a hypergraph G is a *root* if $\mathrm{Acc}(G \cup \{ir\}, i) = V_G$ for i a new colour: $i \notin F_G$. Let us mark by a given colour # the accessible vertices of any regular hypergraph: we shall transform any grammar R generating a hypergraph G into another grammar generating $G \cup \{\#v \mid v \in \mathrm{Acc}(G,i)\}$. This is illustrated in Figure 4.6. The method simply translates the least fixed point defining $\mathrm{Acc}(G,i)$ to a least fixed point on the grammar generating G.

Proposition 4.4. The class of regular hypergraphs is effectively closed under accessible colouring.

Proof. Let R be a grammar of axiom Z generating a hypergraph G. For colours $\iota, \#$, we want to construct a grammar generating $G \cup \{\#v \mid v \in \mathrm{Acc}(G,\iota)\}$. Let $1,\ldots,\varrho(R)$ be the vertices of the left hand sides of R: up to renaming, we assume that each left hand side $X \in \mathrm{Dom}(R)$ of R is of the form $X = X(1)1\ldots\varrho(X(1))$. To each rule $A1\ldots\varrho(A) \longrightarrow H_A$ in R and each $I \subseteq [\varrho(A)]$, we associate the set $\mathrm{Acc}(A,I)$ of vertices in V_{H_A} which are accessible from I and the vertices coloured by ι in a(ny) graph of $R^\omega(H_A)$. This family of sets $\mathrm{Acc}(A,I)$ is the least fixed point of the following recursive system:

$$\begin{aligned}\mathrm{Acc}(A,I) := I \cup \{v \mid \iota v \in H_A\}&\\ \cup \{v \in V_{H_A} \mid T_R(\mathrm{Acc}(A,I))^+ v \cap H_A \neq \varnothing\}&\\ \cup \{Y(i) \mid \exists B \in N_R(BY \in H_A \wedge 1 \leq i \leq |Y| \wedge&\\ i \in \mathrm{Acc}(B,\{j \mid Y(j) \in \mathrm{Acc}(A,I)\}))\}.&\end{aligned}$$

Precisely we take a linear order on the set

$$M := \{(A,I) \mid A \in N_R \wedge I \subseteq [\varrho(A)]\}$$

and we define

$$E := \Big\{ \prod_{(A,I)\in M} P_{A,I} \mid \forall A \in N_R\ \forall I \subseteq J \subseteq [\varrho(A)](P_{A,I} \subseteq P_{A,J})\Big\}.$$

So E is a complete finite set for the inclusion componentwise whose smallest element is $\vec{\varnothing} = (\varnothing,\ldots,\varnothing)$. Then we define the mapping $f\colon E \longrightarrow E$ by

$$\begin{aligned}\Big(f\Big(\prod_{(B,J)\in M} P_{B,J}\Big)\Big)_{A,I} := I \cup \{v \mid \iota v \in H_A\}&\\ \cup \{v \in V_{H_A} \mid T_R P_{A,I}^+ v \cap H_A \neq \varnothing\}&\\ \cup \{Y(i) \mid \exists B \in N_R(BY \in H_A \wedge&\\ 1 \leq i \leq |Y| \wedge i \in P_{B,\{j \mid Y(j) \in P_{A,I}\}})\}.&\end{aligned}$$

Thus f is monotonous:

$$(\forall (A,I) \in M(P_{A,I} \subseteq Q_{A,I})) \Longrightarrow f\Big(\prod_{(A,I)\in M} P_{A,I}\Big) \subseteq f\Big(\prod_{(A,I)\in M} Q_{A,I}\Big).$$

As E is finite, f is continuous and by the Knaster-Tarski theorem:

$$\bigcup_{n\geq 0} f^n(\vec{\varnothing}) \text{ is the least fixed point of } f.$$

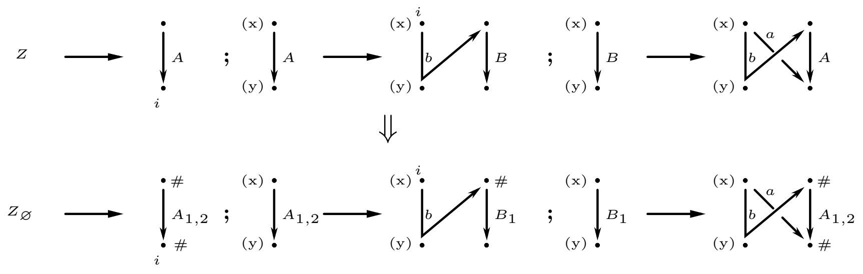


FIGURE 4.7. Colouring from i for the grammar of Figure 4.6.

So we define for every $(A, I) \in M$,

$$\mathrm{Acc}(A, I) := \Big(\bigcup_{n\geq 0} f^n(\vec{\varnothing})\Big)_{A,I}.$$

To each (A, I), we associate a new non-terminal A_I of arity $\varrho(A)$, and we define the following grammar:

$$S := \{(A_I 1 \ldots \varrho(A), H_{A,I}) \mid A \in N_R \wedge I \subseteq [\varrho(A)]\}$$

where

$$H_{A,I} := (H_A \cap T_R V_{H_A}^*) \cup \{\#v \mid v \in \mathrm{Acc}(A, I) - [\varrho(A)]\} \\ \cup \{B_{\{j \mid Y(j) \in \mathrm{Acc}(A,I)\}} Y \mid BY \in H_A \wedge B \in N_R\}.$$

with a restriction to the rules whose non-terminals are accessible from $Z_\varnothing$. Thus S generates from $Z_\varnothing$ the hypergraph $G \cup \{\#v \mid v \in \mathrm{Acc}(G, \iota)\}$.

Q.E.D. (Proposition 4.4)

The construction in the proof of Proposition 4.4 is illustrated in Figure 4.7.

The colouring by accessibility of a hypergraph G is a particular case of *regular colouring* by a finite hypergraph H whose vertices are colours *i. e.* $V_H \subset F_1$, and is the hypergraph defined as the least fixed point of the equation:

$$G \otimes H := G \cup \{c_{\varrho(f)} v_{\varrho(f)} \mid \exists f v_1 \ldots v_{\varrho(f)} \in G \; \exists f c_1 \ldots c_{\varrho(f)} \in H \\ (c_1 v_1, \ldots, c_{\varrho(f)-1} v_{\varrho(f)-1} \in G \otimes H)\}.$$

In particular

$$G \cup \{\#v \mid v \in \mathrm{Acc}(G, i)\} \\ = G \otimes \big(\{i\#\} \cup \{f\# \ldots \# \mid f \in F_G \wedge \varrho(f) > 1\}\big).$$

Let us extend Proposition 4.4 to any regular colouring.

Proposition 4.5. The class of regular hypergraphs is effectively closed under regular colouring.

Proof. We adapt the proof of Proposition 4.4. Let H be a finite hypergraph with $V_H \subset F_1$. Let R be a grammar of axiom Z generating a hypergraph G. We assume that the rule associated to any $A \in N_R$ is of the form:

$$A1 \ldots \varrho(A) \longrightarrow H_A.$$

To each $A \in N_R$ and $I \subseteq V_H[\varrho(A)]$, we associate the terminal hypergraph $\mathrm{Acc}(A, I)$ such that the family of these hypergraphs is the least fixed point of the following recursive system:

$$\begin{aligned}\mathrm{Acc}(A, I) := I \cup [H_A] \cup \big(\mathrm{Acc}(A, I) \otimes H\big)& \\ \cup \{cY(i) \mid \exists B \in N_R(BY \in H_A \wedge 1 \leq i \leq |Y| \wedge& \\ ci \in \mathrm{Acc}(B, \{dj \mid dY(j) \in \mathrm{Acc}(A, I)\}))\}.&\end{aligned}$$

To each (A, I), we associate a new non-terminal A_I of arity $\varrho(A)$, and we define the following grammar:

$$S := \{(A_I 1 \ldots \varrho(A), H_{A,I}) \mid A \in N_R \wedge I \subseteq V_H[\varrho(A)]\}$$

where

$$\begin{aligned}H_{A,I} := \big(\mathrm{Acc}(A, I) - V_H[\varrho(A)]\big)& \\ \cup \{B_{\{dj \mid dY(j) \in \mathrm{Acc}(A,I)\}} Y \mid BY \in H_A \wedge B \in N_R\}.&\end{aligned}$$

Thus S generates from $Z_\varnothing$ the hypergraph $G \otimes H$. Q.E.D. (Proposition 4.5)

We now consider the generation by accessibility. Taking any hypergraph G (whose vertices are) accessible from a given colour i, we map each vertex s to the minimum path length $g(s)$ to access s from i; precisely and inductively

$$\begin{aligned}g^{-1}(0) &= V_{G,i} \\ g^{-1}(n+1) &= \mathrm{Succ}_G(g^{-1}(\leq n)) - g^{-1}(\leq n)\end{aligned}$$

where $g^{-1}(\leq n) := g^{-1}(0) \cup \ldots \cup g^{-1}(n)$. For instance the graph of Figure 3.20 is regular by accessibility as shown in Figure 4.8.

Note that any hypergraph which is regular by accessibility is of finite out-degree and has a finite number of vertices coloured by the initial colour. In Figure 4.9, we give a regular graph of finite degree, accessible from a colour, and which is not regular by accessibility from this colour.

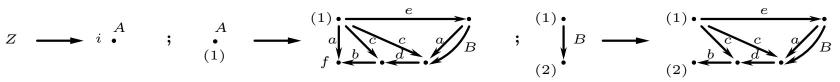


FIGURE 4.8. Generating the graph of Figure 3.20 by accessibility from i.

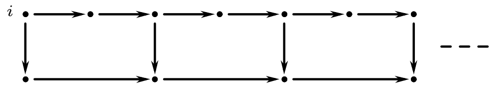


FIGURE 4.9. Regular graph not regular by accessibility from i.

4.4 Regularity by distance

Another usual graduation is the *distance* from a given vertex set E:

$$\mathrm{d}_G(s,E) := \min\{\mathrm{d}_G(s,t) \mid t \in E\}$$
$$\text{where } \mathrm{d}_G(s,t) := \min(\{n \mid s \underset{G}{\longleftrightarrow}^{n} t\} \cup \{\omega\}).$$

For instance the regular graph of Figure 2.7 remains regular by distance from the vertices coloured by 1 or 2 using outside grammar of Figure 4.10. We denote by $\mathrm{d}_G(s,i) := \mathrm{d}_G(s,V_{G,i})$ the distance in a hypergraph G of a vertex s to the set of vertices coloured by i. Note that the nth frontier of G by distance from i satisfies

$$\partial_{g,n}G = \{s \in V_{G-G_{d,n}} \mid d(s,i) = n\}.$$

We say that G is *finitely connected* by i if there is only a finite number of vertices coloured by i, and from which all vertices are connected: $V_{G,i}$ is finite and $d(s,i) < \omega$ for any $s \in V_G$. Any grammar generating a hypergraph G of finite degree and finitely connected from a colour i, can be transformed in an effective way into a grammar generating G by distance from i. Such a graph G is also bounded connected by distance.

Theorem 4.6. Any finitely connected regular hypergraph of finite degree is finitely decomposable by distance.

Proof. In part (i), we introduce the notion of frontier and of interface that allow to uniquely characterize any subset of hyperarcs in a hypergraph. Taking a regular hypergraph G finitely connected and of finite degree, we construct in part (ii) the *canonical* grammar generating G by distance. Part (iii) shows that this canonical grammar is indeed finite. Using (i)–(iii), we got that G is regular by distance. In (iv), we show that G is bounded

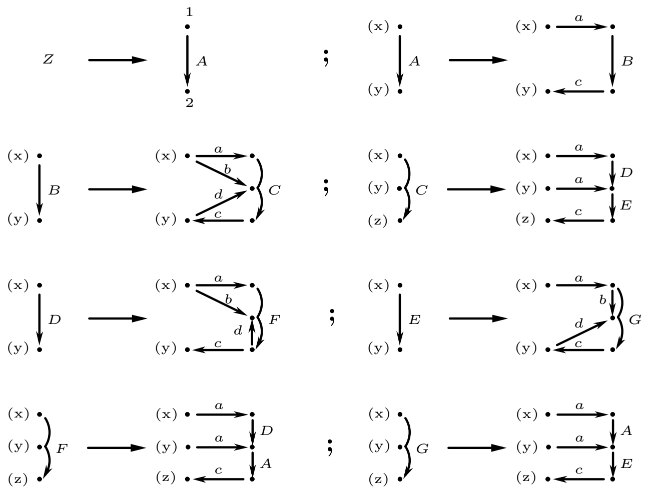


FIGURE 4.10. Grammar generating the graph of Figure 2.7 by distance.

connected by distance, and hence using Proposition 4.3, we deduce that G is finitely decomposable by distance.

(i) Let G be any hypergraph. Consider any sub-hypergraph $H \subset G$ such that for any connected component C of $G, H \cap C \neq C$. Such a hypergraph H is characterized by its *frontier*:

$$\mathrm{Fr}_G(H) := V_H \cap V_{G-H}$$

and by its *interface*:

$$\begin{aligned}\mathrm{In}_G(H) &:= \{X \in H \mid V_X \cap \mathrm{Fr}_G(H) \neq \varnothing\}\\ &= \{X \in H \mid V_X \cap V_{G-H} \neq \varnothing\};\end{aligned}$$

in particular $\mathrm{Fr}_G(H) \subseteq V_{\mathrm{In}_G(H)}$. The charaterization of H by $\mathrm{Fr}_G(H)$ and $\mathrm{In}_G(H)$ follows by this equality:

$$H = G\langle \mathrm{In}_G(H), \mathrm{Fr}_G(H)\rangle$$

where for any $K \subseteq G$ and any $P \subseteq V_G$, the hypergraph $G\langle K, P\rangle$ is the least fixed point of the following equation:

$$G\langle K, P\rangle = K \cup \{X \in G \mid V_X \cap V_{G_{\langle K,P\rangle}} \neq \varnothing \wedge V_X \cap P = \varnothing\}.$$

(ii) Let R be a grammar generating a finite degree hypergraph G finitely connected by a colour ι. We want to show that G is regular by distance d

from ι:

$$d(s) := d(s,\iota) \text{ for any vertex } s \text{ of } G.$$

By Theorem 3.12, we can assume that R is complete outside and connected. Up to a label renaming with adding rules, we assume that each right hand side has no two non-terminal hyperarcs with the same label, and we denote by V_R the set of non input vertices of the right hand sides of R:

$$V_R := \bigcup \{V_H - V_X \mid (X,H) \in R\}.$$

Let $Z = H_0 \underset{R}{\Longrightarrow} H_1 \ldots H_n \underset{R}{\Longrightarrow} H_{n+1} \underset{R}{\Longrightarrow} \ldots$ be the derivation generating $\mathrm{Gen}(R)$: $\bigcup_{n\geq 0}[H_n] = \mathrm{Gen}(R)$. As the set $V_{G,\iota}$ of vertices of G coloured by ι is finite, we denote by m the minimal derivation length to get all the vertices of G coloured by ι:

$$m := \min\{n \mid \forall p > n((H_p - H_n) \cap \iota V_{H_p} = \varnothing)\}.$$

As G is of finite degree and R is degree-outside, each rule of R has no output which is an input, hence

$$\mathrm{Gen}(R)_{d,n} \subseteq [H_{m+n}] \text{ for every } n \geq 0.$$

For every $n \geq 0$, we get

$$\partial_{d,n} \mathrm{Gen}(R) = \{s \in V_{H_{m+n} - \mathrm{Gen}(R)_{d,n}} \mid d(s) = n\}.$$

For every $n \geq 0$, we denote by $\{P_{n,1},\ldots,P_{n,r_n}\}$ the partition of $\partial_{d,n}\mathrm{Gen}(R)$ into connected vertices of $\mathrm{Gen}(R) - \mathrm{Gen}(R)_{d,n}$ *i. e.* of $H_{m+n} - \mathrm{Gen}(R)_{d,n}$, and for every $1 \leq i \leq r_n$,

$$\begin{aligned} K_{n,i} &:= \{X \in \mathrm{Gen}(R) - \mathrm{Gen}(R)_{d,n} \mid V_X \cap P_{n,i} \neq \varnothing\} \\ &= \{X \in [H_{m+n+1}] - \mathrm{Gen}(R)_{d,n} \mid V_X \cap P_{n,i} \neq \varnothing\}. \end{aligned}$$

Thus for every $n \geq 0$,

$$\mathrm{Gen}(R) - \mathrm{Gen}(R)_{d,n} = \bigcup_{i=1}^{r_n} \mathrm{Gen}(R)\langle K_{n,i}, P_{n,i}\rangle.$$

The left *residual* of $C \subseteq \mathrm{Gen}(R)$ by $u \in N_R^*$ is

$$u^{-1}C := \{fu_1 \ldots u_{\varrho(f)} \mid f(uu_1)\ldots(uu_{\varrho(f)}) \in C\}$$

and p_C is the greatest common prefix in N_R^* of the vertices of C. We take a linear ordering $<$ on $N_R \cup V_R$ that we extend on $N_R^* V_R$ by length lexicographic order. For any $n \geq 0$ and $1 \leq i \leq r_n$, we define $p_{n,i} := p_{K_{n,i}}$ and we define the hyperarc

$$X_{n,i} := (p_{n,i}^{-1}K_{n,i}, p_{n,i}^{-1}P_{n,i})s_1 \ldots s_q$$

with $\{s_1, \ldots, s_q\} = P_{n,i}$ and $s_1 > \ldots > s_q$; note that the label is a pair of a finite graph with a vertex subset. We define the grammar $S := \bigcup_{n \geq 0} S_n$ with

$$S_0 := \{(Z, \mathrm{Gen}(R)_{d,0} \cup \{X_{0,1}, \ldots, X_{0,r_0}\})\}$$

and, for all $n \geq 0$, $S_{n+1} := S_n \cup T$ where T contains all pairs

$$(X_{n,i}, K_{n,i} \cup \bigcup \{X_{n+1,j} \mid P_{n+1,j} \cap V_{K_{n,i}} \neq \varnothing\})$$

with $1 \leq i \leq r_n \wedge X_{n,i}(1) \notin N_{S_n}$. The finiteness of S is shown in (iii). For any $n \geq 0$ and $1 \leq i \leq r_n$, S generates from $X_{n,i}$ and by distance from ι the connected component of $\mathrm{Gen}(R) - \mathrm{Gen}(R)_{d,n}$ containing $P_{n,i}$. Thus S generates from Z the hypergraph $\mathrm{Gen}(R)$ by distance from ι.

(iii) Let us show that S is finite. This is obtained by giving a bound b such that $\mathrm{d}_{\mathrm{Gen}(R)}(s,t) \leq b$ for any $n \geq 0$, any connected component C of $\mathrm{Gen}(R) - \mathrm{Gen}(R)_{d,n}$ and any $s,t \in V_C \cap \partial_{d,n} \mathrm{Gen}(R)$. It is sufficient to extract such a bound for any $n \geq n_0$ with n_0 the smallest integer such that $\mathrm{Gen}(R)_{d,n_0} \supseteq [H_m]$. As R is a connected grammar, we take the following integer:

$$c := \max\{\mathrm{d}_{R^\omega(H)}(s,t) \mid H \in \mathrm{Im}(R) \wedge s,t \in V_H\}.$$

Let $n \geq n_0$. Let C be a connected component of $\mathrm{Gen}(R) - \mathrm{Gen}(R)_{d,n}$ and let $s,t \in V_C$ with $d(s) = n = d(t)$. We take a vertex z of C of minimal length. As $z \in V_C$, we have $d(z) \geq n$. By definition of $\mathrm{Gen}(R)$, $z = wr$ for $w \in N_R^*$ and r a vertex of a right hand side of R.

Consider an undirected path of minimal length from s (resp. t) to ι; such a path goes through a vertex $x = wp$ (resp. $y = wq$) for some vertex p (resp. q) of a right hand side of R. Hence

$$d(x,y) \leq c, d(x,z) \leq c, d(y,z) \leq c$$

for distances on $\mathrm{Gen}(R)$. Thus

$$d(s,x) + d(x) = d(s) \leq d(z) \leq d(z,x) + d(x) \leq c + d(x)$$

so $d(s,x) \leq c$. Similarly $d(t,y) \leq c$. Finally

$$d(s,t) \leq d(s,x) + d(x,y) + d(y,t) \leq 3c.$$

Finally $b = 3c$ fits (for any $n \geq n_0$).

(iv) By Proposition 4.3, it remains to verify that G is bounded connected by d. Let C be a connected component of G^d_{n+1} for some $n \geq 0$. So $C' := C - \mathbb{N}V_C$ is a connected component of $G - G_{d,n}$ with

$$V_{C,0} = V_{C'} \cap \partial_{d,n} G.$$

By (iii) we get $\mathrm{d}_G(s,t) \leq b$ for any $s, t \in V_{C,0}$. As G is of finite degree, let D be the maximum degree of its vertices. Thus for any connected component C of $\sum_{n\geq 1} G^d_n$, we have

$$|V_{C,0}| \leq D^0 + D^1 + \ldots + D^b$$

meaning that G is bounded connected by d. Q.E.D. (Theorem 4.6)

The generation by distance is illustrated in Figure 4.11 with $x > y$ and $p > q$ and

$$\begin{aligned}
C &= \big(\{p \xrightarrow{a} Ax, Ay \xrightarrow{b} q\}, \{p,q\}\big)\\
D &= \big(\{r \xrightarrow{a} Bx, r \xrightarrow{e} s\}, \{r\}\big)\\
E &= \big(\{x \xrightarrow{a} Ax, Ay \xrightarrow{b} y\}, \{x,y\}\big)\\
F &= \big(\{Bx \xrightarrow{a} BAx, Bx \xrightarrow{c} By, By \xrightarrow{b} s\}, \{Bx,s\}\big)\\
G &= \big(\{Ax \xrightarrow{a} AAx, Ax \xrightarrow{c} Ay, Ay \xrightarrow{b} y\}, \{Ax,y\}\big).
\end{aligned}$$

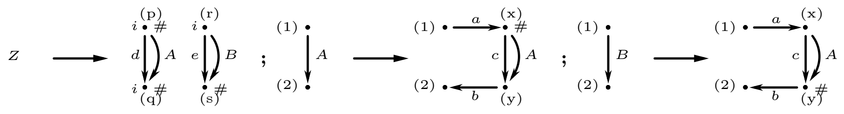
5 Graph grammars and pushdown automata

A pushdown automaton is a particular case of a labelled word rewriting system whose rules are only applied by suffix. Pushdown automata even in a weak form and the rewriting systems define the same graphs by suffix rewriting, which are exactly the regular graphs of bounded degree (cf. Theorem 5.11).

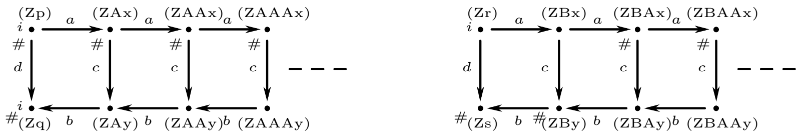
5.1 Suffix transition graphs

A labelled word rewriting system is just a finite uncoloured graph whose vertices are words. Its set of unlabelled suffix transitions is the suffix rewriting relation, whose transitive closure is a rational relation (cf. Proposition 5.2). Its set of labelled suffix transitions is called a suffix graph. Any regular restriction of this graph is regular by length (cf. Theorem 5.6). Conversely any regular graph of finite degree is a regular restriction of a suffix graph (cf. Theorem 5.8).

Taking grammar R of Figure 4.1



its canonical graph Gen(R) is


(Zp) a (ZAx) a (ZAAx) a (ZAAAx)
i # d # c # c # c
i #(Zq) b (ZAy) b (ZAAy) b (ZAAAy)
(Zr) a (ZBx) a (ZBAx) a (ZBAAx)
i d c # c # c
#(Zs) b #(ZBy) b (ZBAy) b (ZBAAy)

The construction of Theorem 4.6 gives the grammar:

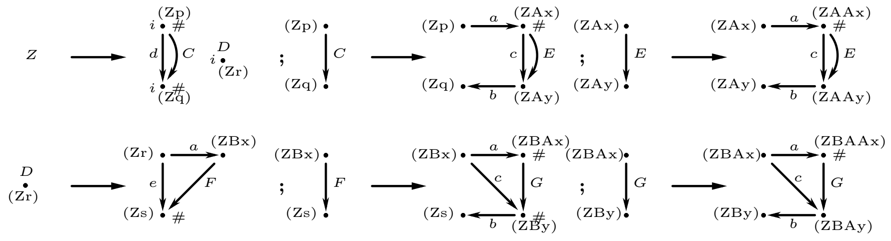

Z
(Zp) i # d C i # (Zq)
D i (Zr)
(Zp) C (Zq)
(Zp) a (ZAx) # c E (Zq) b (ZAy)
(ZAx) E (ZAy)
(ZAx) a (ZAAx) # c E (ZAy) b (ZAAy)
D (Zr)
(Zr) a (ZBx) e F (Zs) #
(ZBx) F (Zs)
(ZBx) a (ZBAx) # c G (Zs) b (ZBy) #
(ZBAx) G (ZBy)
(ZBAx) a (ZBAAx) # c G (ZBy) b (ZBAy)

Figure 4.11. Generation by distance.

We fix a countable set T of symbols, called *terminals*. A labelled word *rewriting system* S is a finite subset of $N^* \times T \times N^*$ where N is an arbitrary alphabet of *non-terminals*; we write $u \xrightarrow[S]{a} v$ for $(u, a, v) \in S$, and define

$$\begin{aligned}
\mathrm{Dom}(S) &:= \{u \mid \exists a \in T \ \exists v \in N^*(u \xrightarrow[S]{a} v)\} && \text{its } \textit{left hand sides}, \\
\mathrm{Im}(S) &:= \{v \mid \exists a \in T \ \exists u \in N^*(u \xrightarrow[S]{a} v)\} && \text{its } \textit{right hand sides}, \\
W_S &:= \mathrm{Dom}(S) \cup \mathrm{Im}(S) && \text{the } \textit{words} \text{ of } S, \\
N_S &:= \{u(i) \mid u \in W_S \wedge 1 \leq i \leq |u|\} && \text{its } \textit{non-terminals}, \\
T_S &:= \{a \in T \mid \exists u, v \in N^*(u \xrightarrow[S]{a} v)\} && \text{its } \textit{terminals}.
\end{aligned}$$

Rewritings in a rewriting system are generally defined as applications of rewriting rules in every context. We are only concerned with suffix rewriting. Given a rewriting system S and a terminal $a \in T_S$, we call *labelled suffix* *rewriting* $\underset{S}{\overset{a}{\longrightarrow\!\!|}}$ the binary relation on N_S^* defined by

$$wu \underset{S}{\overset{a}{\longrightarrow\!\!|}} wv \text{ for any } u \xrightarrow[S]{a} v \text{ and } w \in N_S^*.$$

Example 5.1. Consider the rewriting system $S = \{\varepsilon \xrightarrow{1} ab, bab \xrightarrow{2} ab\}$. We have

$$bb \underset{S}{\overset{1}{\longrightarrow\!\!|}} bbab \underset{S}{\overset{2}{\longrightarrow\!\!|}} bab \underset{S}{\overset{2}{\longrightarrow\!\!|}} ab \underset{S}{\overset{1}{\longrightarrow\!\!|}} abab \underset{S}{\overset{2}{\longrightarrow\!\!|}} aab \ldots$$

For any rewriting system S, the unlabelled *suffix rewriting* is

$$\underset{S}{\longrightarrow\!\!|} \; := \bigcup_{a \in T_S} \underset{S}{\overset{a}{\longrightarrow\!\!|}} \; = \{wu \longrightarrow wv \mid u \xrightarrow[S]{a} v \wedge w \in N_S^*\}$$

and its reflexive and transitive closure (by composition) $\underset{S}{\longrightarrow\!\!|}^*$ is the *suffix derivation.* In Example 5.1, we have $bb \underset{S}{\longrightarrow\!\!|}^* bb$ and $bb \underset{S}{\longrightarrow\!\!|}^* ab$. We denote by

$$\underset{S}{\longrightarrow\!\!|}^+ \; = \; \underset{S}{\longrightarrow\!\!|} \circ \underset{S}{\longrightarrow\!\!|}^*$$

the transitive closure of $\underset{S}{\longrightarrow\!\!|}$. A well-known property is that the set of words deriving by suffix from a given word is a regular language, and a finite automaton accepting it is effectively constructible [2]. This property remains true starting from any regular set of words. More generally, the suffix derivation is itself a rational relation: it can be recognized by a transducer *i. e.* a finite automaton labelled by pairs of words.

Proposition 5.2 (Caucal, [4])**.** The suffix derivation of any word rewriting system is effectively a rational relation.

Proof. We give here a construction improved by Carayol.

(i) Let N be any alphabet. For any $P \subseteq N^*$ and for any word $u \in N^*$, we denote by $u \downarrow P$ the set of irreducible words obtained from u by derivation according to $P \times \{\varepsilon\}$:

$$u \downarrow P := \{v \mid u \underset{P\times\{\varepsilon\}}{\longrightarrow}^* v, \underset{P\times\{\varepsilon\}}{\not\longrightarrow}\}.$$

We extend by union $\downarrow P$ to any language $L \subseteq N^*$:

$$L \downarrow P := \bigcup \{u \downarrow P \mid u \in L\}.$$

A standard result due to Benois [1] is that for P regular, the operation $\downarrow P$ preserves regularity:

$$L, P \in \mathrm{Rat}(N^*) \Longrightarrow L \downarrow P \in \mathrm{Rat}(N^*).$$

Precisely, we have

$$L \downarrow P = \underset{P\times\{\varepsilon\}}{\longrightarrow}^{*}(L) - N^*PN^*$$

It remains to show that the image $\underset{P\times\{\varepsilon\}}{\longrightarrow}^{*}(L)$ of L by the derivation $\underset{P\times\{\varepsilon\}}{\longrightarrow}^{*}$ is regular. This property is true even if P is not regular. Precisely and for L regular, there is a *finite automaton* $A \subseteq Q \times N \times Q$ recognizing L from an *initial state* $i \in Q$ to a subset $F \subseteq Q$ of *final states*: $L(A, i, F) = L$. By adding iteratively ε-transitions between states linked by a path labelled in P, we complete A into an automaton B which is the least fixpoint of the following equation:

$$B = A \cup \{p \xrightarrow{\varepsilon} q \mid \exists u \in P(p \underset{B}{\overset{u}{\Longrightarrow}} q)\}.$$

Note that we can refine B by saturating A with only elementary ε-transitions:

$$\begin{aligned} B = A \cup \{p \xrightarrow{\varepsilon} q \mid p \neq q \wedge \exists a \in P \cap N(p \underset{A}{\overset{a}{\longrightarrow}} q)\} \\ \cup \{p \xrightarrow{\varepsilon} q \mid p \neq q \wedge \exists aub \in P(a, b \in N \wedge p \underset{A}{\overset{a}{\longrightarrow}} \underset{B}{\overset{u}{\Longrightarrow}} \underset{A}{\overset{b}{\longrightarrow}} q)\}. \end{aligned}$$

So $L(B, i, F) = \underset{P\times\{\varepsilon\}}{\longrightarrow}^{*}(L)$.

(ii) We denote N_S by N and to each letter $x \in N$, we associate a new symbol $\overline{x} \notin N$ with $\overline{x} \neq \overline{y}$ for $x \neq y$. Let $\overline{N} := \{\overline{x} \mid x \in N\}$. We extend the operation $\overline{}$ by morphism to all words $u = x_1 \ldots x_n$ *i. e.* $\overline{u} = \overline{x_1} \ldots \overline{x_n}$. Recall that the mirror $\widetilde{u}$ of any word $u = x_1 \ldots x_n$ is the word $\widetilde{u} = x_n \ldots x_1$. The following set is regular:

$$\left[\{\widetilde{\overline{u}}v \mid \exists a(u \underset{S}{\overset{a}{\longrightarrow}} v)\}^*\right) \downarrow \{x\overline{x} \mid x \in N\}\Big] \cap \overline{N}^* N^*$$

meaning that we can apply by suffix a rule (u, v) by producing on the right v after having removed u on the right (using $\downarrow \{x\overline{x} \mid x \in N\}$). This set can be written as a finite union $\bigcup_{i\in I} \overline{U}_i V_i$ where $U_i, V_i \in \mathrm{Rat}(N^*)$ for all $i \in I$. Taking the following relation:

$$\overline{S} := \bigcup_{i\in I} \widetilde{U_i} \times V_i$$

it is easy to verify that the suffix derivation according to S is the suffix rewriting according to $\overline{S}$:

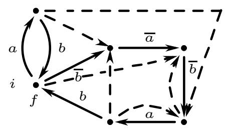
$$\underset{S}{\longrightarrow\!\!\!\mid}^{*} \;=\; \underset{\overline{S}}{\longrightarrow\!\!\!\mid}\;.$$

It follows that $\underset{S}{\longrightarrow\!\!\!\mid}^{*}$ is an effective rational relation. In particular starting from $I \in \mathrm{Rat}(N^*)$, we have

$$\underset{S}{\longrightarrow\!\!\!\mid}^{*}(I) = \underset{\overline{S}}{\longrightarrow\!\!\!\mid}(I) = \mathrm{Im}\big(\underset{\overline{S}}{\longrightarrow\!\!\!\mid} \cap I \times N^*\big) \in \mathrm{Rat}(N^*)$$

Q.E.D. (Proposition 5.2)

Taking the system $S = \{\varepsilon \xrightarrow{1} ab, bab \xrightarrow{2} ab\}$ of Example 5.1, the construction of Proposition 5.2 gives the following finite automaton where the dashed arrows are ε-transitions:



which gives the suffix derivation of S:

$$\underset{S}{\longrightarrow\!\!\!\mid}^{*} = \{\varepsilon\} \times (a^+b)^* \cup b^+ab \times (a^+b)^+ \cup b^+ \times (a^+b)^+.$$

To any rewriting system S, we associate its *suffix graph*:

$$\mathrm{Suff}(S) := \{wu \xrightarrow{a} wv \mid u \underset{S}{\xrightarrow{a}} v \wedge w \in N_S^*\} = N_S^*.S$$

which is the set of its suffix transitions. For instance the suffix graph of $\{x \xrightarrow{a} \varepsilon, x \xrightarrow{b} x^4\}$ is the regular graph of Figure 2.8. The suffix graph of $\{x \xrightarrow{a} \varepsilon, x \xrightarrow{b} zxyx, y \xrightarrow{c} \varepsilon, z \xrightarrow{d} \varepsilon\}$ restricted to the set $(z + zxy)^*(\varepsilon + x)$ of its vertices accessible from x is the graph of Figure 2.9. The suffix transition graphs of word rewriting systems have bounded degree.

Lemma 5.3. The suffix graph of any rewriting system has bounded degree, and has a finite number of non isomorphic connected components.

Proof. Let S be any labelled word rewriting system.

(i) Let us verify that $\mathrm{Suff}(S) = N_S^*.S$ has bounded degree. Let w be any vertex of this graph. As we can at most apply all the rules of S, the out-degree of w is bounded by the number of rules: $d^+(w) \leq |S|$. Note that the inverse of $N_S^*.S$ is the suffix graph of the inverse of S:

$$\big(N_S^*.S\big)^{-1} = N_S^*.S^{-1},$$

so the in-degree of w is its out-degree for $N_S^*.S^{-1}$, hence

$$d^-(w) \leq |S^{-1}| = |S|.$$

Finally the degree of w satisfies: $d(w) = d^+(w) + d^-(w) \leq 2|S|$.

(ii) We show that $N_S^*.S$ has a finite number of non isomorphic connected components. Let H be any connected component of $N_S^*.S$. Let $w \in N_S^*$ such that

$$w.W_S \cap V_H \neq \varnothing \text{ and of length } |w| \text{ minimal.}$$

Such a word w is unique because it is prefix of all the vertices of H: by definition of w, there is $u \in W_S$ such that $wu \in V_H$; by induction on the length of any derivation $wu \underset{H \cup H^{-1}}{\Longrightarrow}{}^* v$, w is prefix of v. By removing this common prefix to the vertices of H, we obtain the graph

$$w^{-1}H := \{u \xrightarrow{a} v \mid wu \underset{H}{\xrightarrow{a}} wv\}$$

which is isomorphic to H and has a vertex in W_S which is finite. So the set of connected components of Suff(S) is finite up to isomorphism.
Q.E.D. (Lemma 5.3)

By Proposition 3.4, the second property of Lemma 5.3 is a particular case of the fact that any suffix graph is regular.

Proposition 5.4. The suffix graph of any rewriting system can be generated by a one-rule graph grammar from its left hand side.

Proof. Let S be any labelled word rewriting system. Let

$$E := \{y \mid \exists x \neq \varepsilon (xy \in W_S)\}$$

be the set of strict suffixes of the words of S. We take a label Y of arity $n = |E|$ and let $\{e_1, \ldots, e_n\} = E$. We define the grammar R restricted to the following rule:

$$Ye_1 \ldots e_n \longrightarrow S \cup \{Y(xe_1)\ldots(xe_n) \mid x \in N_S\}.$$

So $N_S^*.S$ is generated by R from its left hand side: $N_S^*.S \in R^\omega(Ye_1 \ldots e_n)$.
Q.E.D. (Proposition 5.4)

Taking the system $S = \{\varepsilon \xrightarrow{1} ab, bab \xrightarrow{2} ab\}$ of Example 5.1 and by applying the construction of Proposition 5.4, we get the one-rule grammar shown in Figure 5.1 generating the suffix graph of S. The regularity of any suffix graph is preserved by any regular restriction.

Corollary 5.5. Any regular restriction of a suffix graph is a regular graph.

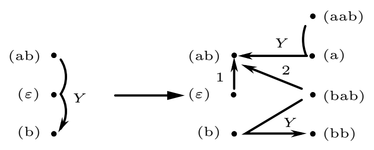


FIGURE 5.1. Generating the suffix graph of the system of Example 5.1.

Proof. Let S be any labelled word rewriting system and let $P \in \mathrm{Rat}(N_S^*)$ be any regular language. We want to show that $\mathrm{Suff}(S)_{|P}$ is a regular graph. We can assume that each non-terminal of S is not a terminal and is an edge label: $N_S \subset F_2 - T_S$. We complete S into the following word rewriting system:

$$\overline{S} := S \cup \{\varepsilon \xrightarrow{x} x \mid x \in N_S\}.$$

It follows that

$$\mathrm{Suff}(\overline{S}) = \mathrm{Suff}(S) \cup \{u \xrightarrow{x} ux \mid u \in N_S^* \wedge x \in N_S\}.$$

As P is regular, there exists a finite graph H labelled in N_S which recognizes P from an initial vertex i to a vertex subset F: $L(H, i, F) = P$. We can assume that the vertices of H are vertex colours: $V_H \subset F_1$. By Proposition 5.4, $\mathrm{Suff}(\overline{S})$ is a regular graph. We take a new colour $\iota \in F_1 - V_H$. By Proposition 4.5, the graph

$$G := \mathrm{Suff}(\overline{S}) \cup \big(\{\iota\varepsilon\} \otimes (H \cup \{\iota i\})\big)$$

remains regular. By removing in G the arcs labelled in N_S, we get the graph

$$G' := G - V_G \times N_S \times V_G$$

which is regular (it suffices to remove the arcs labelled in N_S in the grammar generating G). By Proposition 4.1, the restriction of G' to the vertices coloured in F is again a regular graph G''. By removing all vertex colours from G'', we get $\mathrm{Suff}(S)_{|P}$ which is regular. Q.E.D. (Corollary 5.5)

Theorem 5.6. Any regular restriction of a suffix graph is regular by length.

Proof. We begin as in Corollary 5.5. Let S be any labelled word rewriting system and let $P \in \mathrm{Rat}(N_S^*)$ be any regular language. We want to show that $\mathrm{Suff}(S)_{|P}$ is regular by vertex length. We can assume that each non-terminal of S is not a terminal and is a label colour: $N_S \subset F_1 - T_S$, we complete S into the following word rewriting system:

$$\overline{S} := S \cup \{\varepsilon \xrightarrow{x} x \mid x \in N_S\}, \text{ and get}$$

$$\mathrm{Suff}(\overline{S}) = \mathrm{Suff}(S) \cup \{u \xrightarrow{x} ux \mid u \in N_S^* \wedge x \in N_S\}.$$

In particular $V_{\mathrm{Suff}(\overline{S})} = N_S^*$ and we define

$$m := \max\{|u| \mid u \in W_{\overline{S}}\}.$$

As P is regular, there is a finite complete graph H labelled in N_S which recognizes P from an initial vertex ι to a vertex subset F: $L(H, \iota, F) = P$. We can assume that the vertices of H are vertex colours: $V_H \subset F_1$. We define

$$H(P) := \{cu \mid u \in P \wedge \iota \underset{H}{\overset{u}{\Longrightarrow}} c\} \text{ for any } P \subseteq N_S^*.$$

(i) Let us show that $\mathrm{Suff}(\overline{S}) \cup H(N_S^*)$ is regular by length. For any $n \geq 0$, we define

$$S_n := \{zx \xrightarrow{a} zy \mid x \underset{\overline{S}}{\xrightarrow{a}} y \wedge \min\{|zx|, |zy|\} \leq n < \max\{|zx|, |zy|\}\}$$

in such a way that

$$\mathrm{Suff}(\overline{S}) - \mathrm{Suff}(\overline{S})_{|\,|,n} = \mathrm{Suff}(S_n).$$

For every $n \geq 0$, we get

$$\partial_{|\,|,n} \mathrm{Suff}(\overline{S}) = \{u \in N_S^*.\big(\mathrm{Dom}(S_n) \cup \mathrm{Im}(S_n)\big) \mid |u| \leq n\}$$

and we can compute $\{P_{n,1}, \ldots, P_{n,r_n}\}$ the partition of $\partial_{|\,|,n} \mathrm{Suff}(\overline{S})$ into connected vertices of $\mathrm{Suff}(\overline{S}) - \mathrm{Suff}(\overline{S})_{|\,|,n}$, and for every $1 \leq i \leq r_n$,

$$K_{n,i} := \{u \underset{\mathrm{Suff}(S_n)}{\xrightarrow{a}} v \mid \{u, v\} \cap P_{n,i} \neq \varnothing \wedge \max\{|u|, |v|\} = n + 1\}.$$

Thus with the notation (i) of the proof of Theorem 4.6, we have for every $n \geq 0$,

$$\mathrm{Suff}(\overline{S}) - \mathrm{Suff}(\overline{S})_{|\,|,n} = \bigcup_{i=1}^{r_n} \mathrm{Suff}(\overline{S})\langle K_{n,i}, P_{n,i}\rangle.$$

We take a linear ordering $<$ on N_S that we extend on N_S^* by length-lexicographic order. For any $n \geq 0$ and $1 \leq i \leq r_n$, we take

$$p_{n,i} := \min\{|u| - m \mid u \in P_{n,i} \wedge |u| \geq m\}$$

which is a common prefix of the words in $P_{n,i}$, and we define the hyperarc $X_{n,i} := p_{n,i}^{-1} H(P_{n,i}) s_1 \ldots s_q$ with $\{s_1, \ldots, s_q\} = P_{n,i}$ and $s_1 < \ldots < s_q$; note that the label is a finite set of coloured vertices. We define the grammar

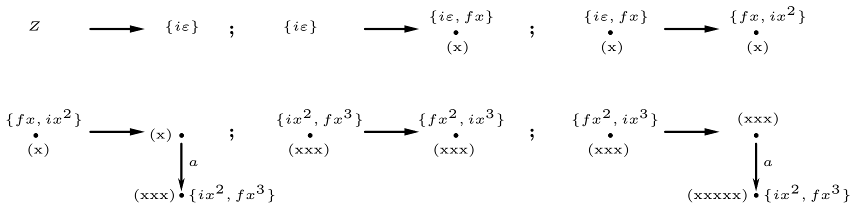


FIGURE 5.2. Generation by length of a regular restriction of a suffix graph.

$R := \bigcup_{n\geq 0} R_n$ with $R_0 := \{(Z, (S \cap \{\varepsilon\} \times T_S \times \{\varepsilon\}) \cup \{\iota\varepsilon, X_{0,1}\}\}$ and, for all $n \geq 0$, $R_{n+1} := R_n \cup S$ where S contains all pairs

$$(X_{n,i}, K_{n,i} \cup H(V_{K_{n,i}} - P_{n,i}) \cup \bigcup\{X_{n+1,j} \mid P_{n+1,j} \cap V_{K_{n,i}} \neq \varnothing\})$$

with $1 \leq i \leq r_n \wedge X_{n,i}(1) \notin N_{R_n}$. The finiteness of R is shown in (ii).

For any $n \geq 0$ and any $1 \leq i \leq r_n$, R generates from $X_{n,i}$ and by vertex length, the connected component of $\big(\mathrm{Suff}(\overline{S}) - \mathrm{Suff}(\overline{S})_{|\,|,n}\big) \cup H(\{u \in N_S^* \mid |u| > n\})$ containing $P_{n,i}$. Thus R generates from axiom Z the graph $\mathrm{Suff}(\overline{S}) \cup H(N_S^*)$ by vertex length.

(ii) Let us show that R is finite. It is sufficient to show that $\{p_{n,i}^{-1}P_{n,i} \mid n \geq 0 \wedge 1 \leq i \leq r_n\}$ is finite. Let $n \geq 0$ and $1 \leq i \leq r_n$. We show that any word in $p_{n,i}^{-1}P_{n,i}$ has length at most $2m$. Let $u, v \in P_{n,i}$. We have $|u| \leq n$. There exist $z \in N_S^*$ and $x \xrightarrow[S_n \cup S_n^{-1}]{a} y$ with $v = zx$ and $|zy| > n$. Hence $|u| - |v| = |u| - |zy| \leq n - (n - |y|) = |y| \leq m$. Assume now that v is of minimal length. Either $|v| \leq m$, so $p_{n,i} = \varepsilon$ and thus $|p_{n,i}^{-1}u| = |u| \leq m + |v| \leq 2m$. Or $|v| > m$, then $v = wx$ for some w and $|x| = m$. Thus $p_{n,i} = w$ and $|p_{n,i}^{-1}u| - |x| = |u| - |v| \leq m$ hence $|p_{n,i}^{-1}u| \leq m + |x| = 2m$.

(iii) It remains to end as in the proof of Corollary 5.5. We remove in R the arcs labelled in N_S and by Proposition 4.1, we restrict to the vertices coloured by F. Then we remove the colours and apply Lemma 3.2 to get a grammar generating $\mathrm{Suff}(S)_{|L}$ by length. Q.E.D. (Theorem 5.6)

Starting with the system $S = \{\varepsilon \xrightarrow{a} xx\}$ and the language $L = x(xx)^*$ recognized by the complete automaton $\{i \xrightarrow{x} f, f \xrightarrow{x} i\}$ from i to f, the construction of Theorem 5.6 yields the grammar shown in Figure 5.2, which generates $\mathrm{Suff}(S)_{|L} = \{x^{2n+1} \xrightarrow{a} x^{2n+3} \mid n \geq 0\}$ by length.

In Subsection 3.5, we have associated to any grammar R a representant $\mathrm{Gen}(R)$ of its set of generated graphs. Any vertex of $\mathrm{Gen}(R)$ is the word of the non-terminals used to get it. This allows us to express Gen(R) as a suffix graph when it is of bounded degree.

Lemma 5.7. Any grammar R generating a bounded degree uncoloured graph, can be transformed into a word rewriting system S such that any connected component (resp. any accessible subgraph) of Gen(R) is a connected component (resp. accessible subgraph) of Suff(S).

Proof. To define Gen(R) simply, we assume that each right hand side has no two non-terminal hyperarcs with the same label. We assume that the rule of any $A \in N_R$ is of the form: $A1 \dots \varrho(A) \longrightarrow H_A$. We write V_R the set of non input vertices of the right hand sides of R:

$$V_R := \bigcup \{ V_{H_A} - [\varrho(A)] \mid A \in N_R \}.$$

To each $A \in N_R$, let S_A be a graph of vertex set $V_{S_A} \subset N_R^+ V_R \cup [\varrho(A)]$ labelled in T_R such that the family of graphs S_A is the least fixed point of the following equations:

$$S_A = A \cdot \big([H_A] \cup \bigcup \{ S_B[Y(1), \dots, Y(\varrho(B))] \mid BY \in H_A \wedge B \in N_R \} \big)$$

where for any $A \in N_R$, for any graph G of vertex set $V_{S_A} \subset N_R^* V_R \cup [\varrho(A)]$ labelled in T_R and for any $a_1, \dots, a_{\varrho(A)} \in V_R \cup [\varrho(R)]$, the *substitution* $G[a_1, \dots, a_{\varrho(A)}]$ is the graph obtained from G by replacing in its vertices each $i \in [\varrho(A)]$ by a_i:

$$G[a_1, \dots, a_{\varrho(A)}] := \{ u[a_1, \dots, a_{\varrho(A)}] \xrightarrow{a} v[a_1, \dots, a_{\varrho(A)}] \mid u \xrightarrow[G]{a} v \}$$

with

$$u[a_1, \dots, a_{\varrho(A)}] := \begin{cases} a_i & \text{if } u = i \in [\varrho(A)] \\ u & \text{otherwise;} \end{cases}$$

and where the *addition* $A \cdot G$ is defined by

$$A \cdot G := \{ A \cdot (u \xrightarrow{a} v) \mid u \xrightarrow[G]{a} v \}$$

and with $A \cdot (u \xrightarrow{a} v)$ defined by

$$\begin{cases} u \xrightarrow{a} v & \text{if } u, v \in [\varrho(A)] \vee u, v \notin [\varrho(A)] \cup V_R \\ Au \xrightarrow{a} v & \text{if } u \notin [\varrho(A)] \wedge v \in [\varrho(A)] \\ u \xrightarrow{a} Av & \text{if } u \in [\varrho(A)] \wedge v \notin [\varrho(A)] \\ Au \xrightarrow{a} Av & \text{if } u, v \notin [\varrho(A)] \wedge (u \in V_R \vee v \in V_R). \end{cases}$$

The system $S = S_Z$ is suitable, for Z the axiom of R:

$$S_Z = \{ u \xrightarrow{a} v \mid \min\{|u|, |v|\} = 2 \wedge \exists w (wu \xrightarrow[\mathrm{Gen}(R)]{a} wv) \}.$$

Q.E.D. (Lemma 5.7)

Taking the grammar of Figure 3.20, the construction of Lemma 5.7 yields

$$\begin{aligned}
S_Z &= Z \cdot (S_A[s,t]) \\
S_A &= A \cdot (\{1 \xrightarrow{a} 2, p \xrightarrow{b} 2\} \cup S_B[1,p]) \\
S_B &= B \cdot (\{1 \xrightarrow{c} 2, q \xrightarrow{d} 2\} \cup S_C[1,q]) \\
S_C &= C \cdot (\{1 \xrightarrow{c} 2, 1 \xrightarrow{e} r\} \cup S_A[r,2])
\end{aligned}$$

hence

$$\begin{aligned}
S_Z = \{&Zs \xrightarrow{a} Zt, Zs \xrightarrow{c} ZAp, Zs \xrightarrow{c} ZABq, Zs \xrightarrow{e} ZABCr\} \\
&\cup \{ZAp \xrightarrow{b} Zt, ABq \xrightarrow{d} Ap, BCr \xrightarrow{a} Bq, BCAp \xrightarrow{b} Bq\} \\
&\cup \{Cr \xrightarrow{c} CAp, CBq \xrightarrow{d} Cp, Cr \xrightarrow{c} CABq, Cr \xrightarrow{e} CABCr\}
\end{aligned}$$

Corollary 5.5 (or Theorem 5.6) and Lemma 5.7 imply the equality between the classes of suffix graphs and uncoloured regular graphs of bounded degree.

Theorem 5.8. Considering the suffix graphs of labelled word rewriting systems, their connected components are the connected regular graphs of bounded degree, their accessible subgraphs are the rooted regular graphs of bounded degree, their regular restrictions are the regular graphs of bounded degree.

Proof. **(i)** Let S be any word rewriting system. Let v be any vertex of $\mathrm{Suff}(S)$ *i.e.* $v \in N_S^*(\mathrm{Dom}(S) \cup \mathrm{Im}(S))$. By Proposition 5.2, the set of vertices accessible from v is the regular language $\underset{S}{\longrightarrow\!\!\!\shortmid}^{*}(v)$, and the vertex set of the connected component of $\mathrm{Suff}(S)$ containing v is the regular language $\underset{S\cup S^{-1}}{\longrightarrow\!\!\!\shortmid}^{*}(v)$. By Corollary 5.5, any regular restriction (resp. any accessible subgraph, any connected component) of $\mathrm{Suff}(S)$ is an uncoloured (resp. rooted, connected) regular graph of bounded degree.

(ii) Let R be any grammar generating an uncoloured graph of bounded degree. Let S be the word rewriting system constructed from R by Lemma 5.7. In 5.1, we have seen that $\mathrm{Gen}(R)$ has a regular vertex set. By Lemma 5.7,

$$\mathrm{Gen}(R) = \mathrm{Suff}(S)_{|V_{\mathrm{Gen}(R)}}$$

hence $\mathrm{Gen}(R)$ is a regular restriction of a suffix graph. Furthermore by Lemma 5.7, if $\mathrm{Gen}(R)$ is connected (resp. rooted) then it is a connected component (resp. accessible subgraph) of $\mathrm{Suff}(S)$. Q.E.D. (Theorem 5.8)

We now restrict as much as possible the word rewriting systems to define the same suffix graphs.

5.2 Weak pushdown automata

A (real-time) *pushdown automaton* S over the alphabet T of terminals is a particular word rewriting system: S is a finite subset of $PQ \times T \times P^*Q$ where P, Q are disjoint alphabets of respectively *stack letters* and *states*; we denote by

$$P_S := \{u(i) \mid 1 \leq i \leq |u| \wedge \exists q \in Q(uq \in W_S)\} \qquad \text{the } \textit{stack letters},$$
$$Q_S := \{q \mid \exists u \in P^*, uq \in W_S\} \qquad \text{the } \textit{states} \text{ of } S.$$

A *configuration* of S is a word in $P_S^*.Q_S$: a stack word followed by a state. The *transition graph* of S is the set of its transitions restricted to its configurations:

$$\mathrm{Tr}(S) := \{wu \xrightarrow{a} wv \mid u \xrightarrow[S]{a} v \wedge w \in P_S^*\} = P_S^*.S$$

It is also the suffix graph of S restricted to its configurations.

Note that a pushdown automaton is essentially a labelled word rewriting system whose left hand sides are of length 2 and such that the rules are only applied by suffix. A symmetrical way to normalize both sides of the rules of a rewriting system is given by a *weak pushdown automaton* S which is a finite set of rules of the form:

$$p \xrightarrow{a} q \text{ or } p \xrightarrow{a} xq \text{ or } xp \xrightarrow{a} q \text{ with } x \in P, p, q \in Q, a \in T$$

where P and Q are disjoint alphabets of stack letters and states; we also write P_S and Q_S for respectively the stack letters and the states (appearing in the rules) of S. The *transition graph* of S is also the set of its (suffix) transitions restricted to its configurations: $\mathrm{Tr}(S) := P_S^*.S$. We define the same suffix graphs by normalizing labelled word rewriting systems as pushdown automata or weak pushdown automata.

Theorem 5.9. The suffix graphs of labelled word rewriting systems, the transition graphs of pushdown automata, and the transition graphs of weak pushdown automata, have up to isomorphism the same connected components, the same accessible subgraphs and the same regular restrictions.

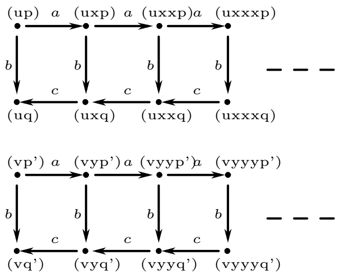
Proof. **(i)** Let S be any weak pushdown automaton. Let us construct a pushdown automaton $\overline{S}$ simulating S: the connected components (resp. accessible subgraphs, regular restrictions) of $\mathrm{Tr}(S)$ are connected components (resp. accessible subgraphs, regular restrictions) of $\mathrm{Tr}(\overline{S})$. We take a new symbol $\perp$ and we define the pushdown automaton:

$$\overline{S} := \{yp \xrightarrow{a} yxq \mid p \xrightarrow[S]{a} xq \wedge y \in N_S \cup \{\perp\}\}$$
$$\cup \{yp \xrightarrow{a} yq \mid p \xrightarrow[S]{a} q \wedge y \in N_S \cup \{\perp\}\}$$
$$\cup \{xp \xrightarrow{a} q \mid xp \xrightarrow[S]{a} q\}.$$

Weak pushdown automaton:

$$\begin{array}{ll} p \xrightarrow{a} xp & p' \xrightarrow{a} yp' \\ p \xrightarrow{b} q & p' \xrightarrow{b} q' \\ xq \xrightarrow{c} q & yq' \xrightarrow{c} q' \end{array}$$

Transition graph:



for any $u \in \{x, y\}^* y \cup \{\varepsilon\}$

for any $v \in \{x, y\}^* x \cup \{\varepsilon\}$

FIGURE 5.3. The transition graph of a weak pushdown automaton.

Thus $P_{\overline{S}} = P_S \cup \{\bot\}$ and $Q_{\overline{S}} = Q_S$. Furthermore

$$u \underset{\mathrm{Tr}(S)}{\xrightarrow{a}} v \Longleftrightarrow \bot u \underset{\mathrm{Tr}(\overline{S})}{\xrightarrow{a}} \bot v \text{ for any } u, v \in P_S^*.Q_S.$$

It follows that for any $L \in \mathrm{Rat}((P_S \cup Q_S)^*) \cap P_S^*.Q_S$ written by abuse of notation as $\mathrm{Rat}(P_S^*.Q_S)$,

$$\mathrm{Tr}(S)_{|L} = \mathrm{Tr}(\overline{S})_{|\bot L}$$

and for any vertex v of $\mathrm{Tr}(S)$ *i. e.* $v \in P_S^*.W_S$, the connected component (resp. accessible subgraph) of $\mathrm{Tr}(S)$ containing v (resp. from v) is the connected component (resp. accessible subgraph) of $\mathrm{Tr}(\overline{S})$ containing (resp. from) $\bot v$.

(ii) Let S be any pushdown automaton. Thus S is simulated by itself as a rewriting system over $P_S \cup Q_S$ because

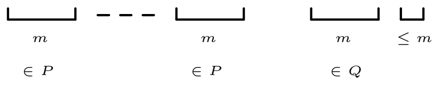
$$\mathrm{Tr}(S)_{|L} = \mathrm{Suff}(S)_{|L} \text{ for any } L \in \mathrm{Rat}(P_S^*.Q_S)$$

and for any $v \in P_S^*.W_S$, the connected component (resp. accessible subgraph) of $\mathrm{Tr}(S)$ containing v (resp. from v) is the connected component (resp. accessible subgraph) of $\mathrm{Suff}(S)$ containing (resp. from) v.

(iii) Let S be any labelled word rewriting system. We want to simulate S by a weak pushdown automaton $\overline{S}$. Let m be the greatest length of the words of S:

$$m := \max\{|u| \mid u \in W_S\}.$$

As in (i), we take a new symbol $\perp$ to mark on the left the words over N_S. Any word in $\perp N_S^*$ is decomposed from left to right into m blocks (the last block being of length $\leq m$):



by using the two bijections: i from $N_S^m \cup \perp N_S^{m-1}$ to a new alphabet P and j from $\{\perp w \mid w \in N_S^* \wedge |w| < 2m\} \cup \{w \in N_S^* \mid m < |w| \leq 2m\}$ to a new alphabet Q, and according to the injection k defined from $N_S^* \cup \perp N_S^*$ into $P^*.Q$ by

$$k(u) := \begin{cases} \varepsilon & \text{if } u = \varepsilon \\ j(u) & \text{if } u \in \mathrm{Dom}(j) \\ i(w)k(v) & \text{if } u = wv \notin \mathrm{Dom}(j) \wedge |w| = m. \end{cases}$$

For every $n \geq 0$, we denote by $f(n) := \lceil \frac{n}{m} \rceil$ the (minimal) number of blocs of length m necessary to contain n letters. By applying (by suffix) any rule of S, we can add or delete at most m letters, hence

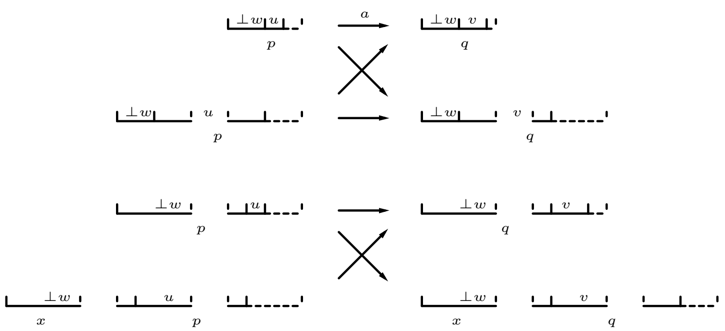
$$|f(|u|) - f(|v|)| \leq 1 \text{ for any } u \xrightarrow[S]{a} v.$$

We define the weak pushdown automaton $\overline{S} := \overline{S}' \cup \overline{S}''$ with

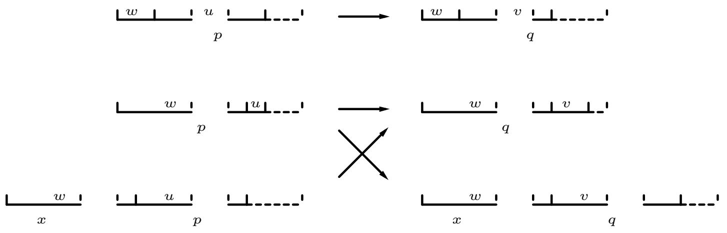
$$\overline{S}' := \{k(\perp wu) \xrightarrow{a} k(\perp wv) \mid u \xrightarrow[S]{a} v \wedge w \in N_S^* \wedge f(\perp wu) + f(\perp wv) \leq 5\}$$

$$\overline{S}'' := \{k(wu) \xrightarrow{a} k(wv) \mid u \xrightarrow[S]{a} v \wedge w \in N_S^* \wedge 4 \leq f(wu) + f(wv) \leq 5\}$$

We illustrate below the different types of rules for $\overline{S}'$:



We illustrate below the different types of rules for $\overline{S}''$:



Note that we have

$$u \xrightarrow[\perp N_S^*.S]{a} v \Longrightarrow k(u) \xrightarrow[P^*.\overline{S}]{a} k(v)$$

$$u \in \perp N_S^* \wedge k(u) \xrightarrow[P^*.\overline{S}]{a} w \Longrightarrow \exists v (u \xrightarrow[\perp N_S^*.S]{a} v \wedge k(v) = w)$$

$$v \in \perp N_S^* \wedge w \xrightarrow[P^*.\overline{S}]{a} k(v) \Longrightarrow \exists u (u \xrightarrow[\perp N_S^*.S]{a} v \wedge k(u) = w).$$

It follows that the image by k of the connected component of $\perp N_S^*.S$ containing $\perp u$ is equal to the connected component of $P^*.\overline{S}$ containing $k(\perp u)$. Furthermore the accessible subgraph from $\perp u$ of $\perp N_S^*.S$ is equal to the accessible subgraph from $k(\perp u)$ of $P^*.\overline{S}$. We also deduce that the suffix graph $\mathrm{Suff}(S) = N_S^*.S$ is isomorphic to

$$k(\perp N_S^*.S) = \overline{S}' \cup i(\perp N_S^{m-1}).(i(N_S^m))^*.\overline{S}'' = (P^*.\overline{S})_{|k(\perp N_S^*)},$$

hence $N_S^*.S$ is not isomorphic to $P^*.\overline{S}$ (we need a restriction). More generally we have

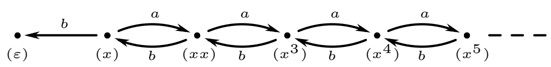
$$k\big((\perp N_S^*.S)_{|\perp M}\big) = (P^*.\overline{S})_{|k(\perp M)} \text{ for any } M \subseteq N_S^*$$

and if $M \in \mathrm{Rat}(N_S^*)$ then $k(\perp M) \in \mathrm{Rat}(P^*.Q)$. Consequently any regular restriction of $N_S^*.S$ is isomorphic to a regular restriction of the transition graph of the weak pushdown automaton $\overline{S}$. Q.E.D. (Theorem 5.9)

Let us illustrate the construction of the proof (iii) of Theorem 5.9 applied to the labelled word rewriting system:

$$S = \{x \xrightarrow{a} xx, x \xrightarrow{b} \varepsilon\}.$$

Its suffix graph $\mathrm{Suff}(S)$ is the following rooted graph:



Note that $L(\mathrm{Suff}(S), x, \varepsilon)$ is the Łukasiewicz language. By applying the construction (iii) of Theorem 5.9, the greatest length of S is $m = 2$. Its set of states is $Q = \{1, 2, 3, 4, p, q\}$ with the following bijection j:

$$\begin{array}{rlrlrl} \perp & \longmapsto 1; & \perp x & \longmapsto 2; & \perp xx & \longmapsto 3; \\ \perp xxx & \longmapsto 4; & xxx & \longmapsto p; & xxxx & \longmapsto q \end{array}$$

and its set of pushdown letters is $P = \{y, z\}$ with the bijection i:

$$xx \longmapsto y; \qquad\qquad \perp x \longmapsto z$$

By coding the arcs of $\mathrm{Suff}(S)$ restricted to $\{\varepsilon, \ldots, x^5\}$, we get the following weak pushdown automaton $\overline{S}$:

$$\left|\begin{array}{rlrlrl} 2 & \xrightarrow{b} 1; & 2 & \xrightarrow{a} 3; & 3 & \xrightarrow{b} 2 \\ 3 & \xrightarrow{a} 4; & 4 & \xrightarrow{b} 3; & 4 & \xrightarrow{a} zp \\ zp & \xrightarrow{b} 4; & p & \xrightarrow{a} q; & q & \xrightarrow{b} p \\ q & \xrightarrow{a} yp; & yp & \xrightarrow{b} q & & \end{array}\right.$$

Its transition graph $\mathrm{Tr}(\overline{S})$ accessible from 2 (or connected to 2) is the following:

$$\begin{tikzcd} \underset{(1)}{\bullet} & \underset{(2)}{\bullet} \arrow[l, "b"'] \arrow[r, bend left, "a"] & \underset{(3)}{\bullet} \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & \underset{(4)}{\bullet} \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & \underset{(zp)}{\bullet} \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & \underset{(zq)}{\bullet} \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & \underset{(zyp)}{\bullet} \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & \underset{(zyq)}{\bullet} \arrow[l, bend left, "b"] \arrow[r, bend left, "a"] & \underset{(zyyp)}{\bullet} \arrow[l, bend left, "b"] \arrow[r, dash, dashed] & {} \end{tikzcd}$$

The use of weak pushdown automata, instead of word rewriting systems or of pushdown automata, allows simpler constructions. For instance, let us restrict Theorem 5.6 to weak pushdown automata.

Proposition 5.10. Let S be a weak pushdown automaton. Let H be a finite deterministic graph labelled in P_S and coloured in Q_S recognizing from a vertex i the configuration language:

$$L = \{uq \mid u \in P_S^* \wedge q \in Q_S \wedge i \underset{H}{\overset{u}{\Longrightarrow}} s \wedge qs \in H\}.$$

Thus $\mathrm{Suff}(S)_{|L}$ is generated by length by a grammar with $|V_H| + 1$ rules.

Proof. Let $Q_S = \{q_1, \ldots, q_n\}$ be the set of states of S. We associate to any vertex s of H a new label $[s]$ of arity n and we define the grammar R with the axiom rule

$$Z \longrightarrow \{[i]q_1 \ldots q_n\} \cup \{p \underset{S}{\overset{a}{\longrightarrow}} q \mid pi, qi \in H\},$$

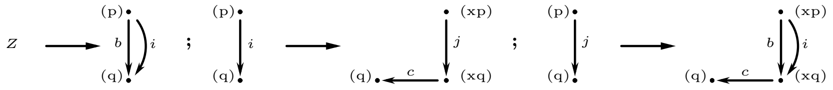


FIGURE 5.4. Regular restriction of a weak pushdown graph.

and for any vertex s of H, we take the following rule:

$$\begin{aligned}[s]q_1 \dots q_n \longrightarrow \{xp \xrightarrow{a} xq \mid p \xrightarrow[S]{a} q \wedge \exists t(s \xrightarrow[H]{x} t \wedge pt, qt \in H)\} \\ \cup \{p \xrightarrow[S]{a} xq \mid \exists t(s \xrightarrow[H]{x} t \wedge ps, qt \in H)\} \\ \cup \{xp \xrightarrow[S]{a} q \mid \exists t(s \xrightarrow[H]{x} t \wedge pt, qs \in H)\} \\ \cup \{[t](xq_1)\dots(xq_n) \mid s \xrightarrow[H]{x} t\}\end{aligned}$$

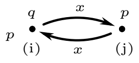
Thus R generates by length $(P_S^*.S)_L$ from its axiom Z.

Q.E.D. (Proposition 5.10)

Taking the weak pushdown automaton of Figure 5.3 restricted to the system

$$S = \{p \xrightarrow{a} xp, p \xrightarrow{b} q, xq \xrightarrow{c} q\}$$

and the regular language $L = (xx)^*p \cup x^*q$ of configuration recognized from vertex i by the following finite deterministic automaton:



The construction of Proposition 5.10 gives the grammar shown in Figure 5.4 which generates $\mathrm{Suff}(S)_L$ by length.

5.3 Main result

Finally we put together Theorem 5.8 and Theorem 5.9, and we recall Theorem 4.6 and Theorem 5.6.

Theorem 5.11. The suffix graphs of labelled word rewriting systems, the transition graphs of pushdown automata, and the transition graphs of weak pushdown automata, have up to isomorphism

- the same connected components: the connected regular graphs of bounded degree,
- the same accessible subgraphs: the rooted regular graphs of bounded degree,

- the same regular restrictions: the regular graphs of bounded degree.

These graphs are regular by length, and also by distance when they are connected.

All these equivalences are effective. Note that by Theorem 4.6 (or Proposition 4.3), the regularity by distance for the connected graphs coincides with the finite decomposition by distance.

Theorem 5.12 (Muller-Schupp, [8])**.** The connected components of pushdown automata are the connected graphs of bounded degree having a finite decomposition by distance.

This result has been expressed with the usual pushdown automata which are intermediate devices between the general labelled word rewriting systems (applied by suffix) and the weak pushdown automata. Furthermore the finite decomposition by distance for the connected graphs of bounded degree is a normal form of the regularity.

6 Languages

Any graph G traces the language $L(G,i,f)$ of the labels of its paths from a colour i to a colour f. By Theorem 5.11, the regular graphs trace exactly the context-free languages, and by restriction to path grammars, we give directly a context-free grammar generating the path labels of any regular graph (cf. Propositions 6.2 and 6.3). Finally we verify that the deterministic regular graphs trace exactly the deterministic context-free languages (cf. Proposition 6.5).

6.1 Path grammars

The regular languages are the languages recognized by the finite automata:

$$\mathrm{Rat}(T^*) := \{L(G,i,f) \mid G \text{ finite} \wedge F_G \cap F_2 \subseteq T \wedge i,f \in F_1\}$$

and the context-free languages, which are the languages recognized by the pushdown automata, are the languages recognized by the regular graphs:

$$\mathrm{Alg}(T^*) := \{L(G,i,f) \mid G \text{ regular} \wedge F_G \cap F_2 \subseteq T \wedge i,f \in F_1\}.$$

This equality follows by Theorem 5.11 because by adding ε-transitions, we can transform any regular graph G into a regular graph $\overline{G}$ of bounded degree recognizing the same language: $L(G,i,f) = L(\overline{G},i,f)$.

Let us give a simple construction to get directly a context-free grammar generating the recognized language of a regular graph. In fact and contrary to the previous sections, we just need transformations preserving the recognized language but not the structure. First by adding ε-transitions, we can start from a unique vertex to end to a unique vertex. More precisely, let R be a grammar and H be a finite hypergraph such that $R^\omega(H)$ are only coloured graphs. For any colours i, f, we denote

$$L(R, H, i, f) := L(\mathrm{Gen}(R, H), i, f)$$

the label set of the paths from i to f of any generated graph by R from H, or in particular for the canonical graph $\mathrm{Gen}(R, H)$ defined in 3.5. For Z the axiom of R, we also write

$$L(R, i, f) := L(R, Z, i, f) = L(\mathrm{Gen}(R), i, f).$$

We say that R is an *initial grammar* for the colours i, f when only the right hand side H of Z is coloured by i, f, and i, f colour a unique vertex:

$$|H \cap iV_H| = 1 = |H \cap fV_H|.$$

Lemma 6.1. For any grammar R and colours i, f, we can get an initial grammar S labelled in $F_R \cup \{\varepsilon\}$ and recognizing the same language: $L(R, i, f) = L(S, i, f)$.

Proof. Let R be any grammar generating from its axiom Z a coloured graph G. To any non-terminal $A \in N_R - \{Z\}$, we associate a new symbol A' of arity $\varrho(A) + 2$. We take two new vertices p, q which are not vertices of R. We define the following grammar:

$$\begin{aligned} S := \{(Z, K' \cup \{ip, fq\}) \mid (Z, K) \in R\} & \\ \cup \{(A'Xpq, K') \mid (AX, K) \in R \wedge A \neq Z\} & \end{aligned}$$

where for any hypergraph $K \in \mathrm{Im}(S)$, the graph K' is the following:

$$\begin{aligned} K' := \{s \underset{K}{\xrightarrow{a}} t \mid a \in T_R\} \cup \{A'Xpq \mid AX \in K \wedge A \in N_R\} & \\ \cup \{p \xrightarrow{\varepsilon} s \mid is \in K\} \cup \{s \xrightarrow{\varepsilon} q \mid fs \in K\}. & \end{aligned}$$

Assuming that $p, q \notin V_G$, S generates from its axiom Z the following graph:

$$H := (G - F_1V_G) \cup \{ip, fq\} \cup \{p \xrightarrow{\varepsilon} s \mid is \in G\} \cup \{s \xrightarrow{\varepsilon} q \mid fs \in G\}$$

satisfying $L(G, i, f) = L(H, i, f)$ *i. e.* $L(R, i, f) = L(S, i, f)$. Note that for G having an infinite number of initial (resp. final) vertices, the vertex p (resp. q) in H is of infinite out-degree (resp. in-degree). By adding new ε-arcs, we can avoid these infinite degrees. Q.E.D. (Lemma 6.1)

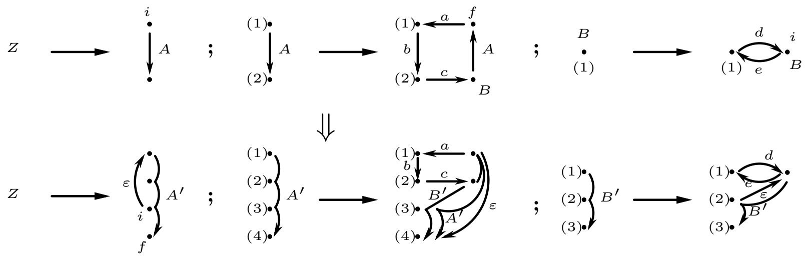


FIGURE 6.1. Same language from an initial vertex to a final vertex.

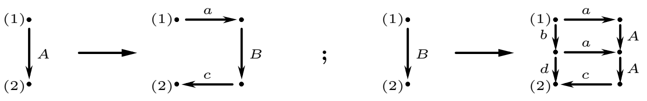


FIGURE 6.2. An acyclic path grammar.

In Figure 6.1 we illustrate the construction of Lemma 6.1. To preserve the recognized language of a regular graph (and not the structure), we can restrict to grammars having only arcs (of arity 2). A *path grammar* R is a deterministic graph grammar without axiom and whose each rule is of the form $A12 \longrightarrow H$ where H is a finite set of arcs having no arc of source 2 and no arc of goal 1. In Figure 6.2, we give a path grammar which is *acyclic*: each right hand side is an acyclic graph. For any path grammar R, any $A \in N_R$ and any derivation

$$A12 = H_0 \underset{R}{\Longrightarrow} H_1 \dots H_n \underset{R}{\Longrightarrow} \dots$$

we define the following languages, where $n \geq 0$:

$$\begin{aligned} L_n(R,A) &:= L(H_n,1,2) && \subseteq (N_R \cup T_R)^* \qquad \forall n \geq 0 \\ L(R,A) &:= \bigcup_{m\geq 0} \big(L_m(R,A) \cap T_R^*\big) && \subseteq T_R^*. \end{aligned}$$

Proposition 6.2 (Caucal-Hieu, [6]). For any grammar R and colours i, f, we can get a path grammar S recognizing from a non-terminal A the language $L(S,A) = L(R,i,f)$.

Proof. **(i)** We assume that each rule of R is of the form: $A1 \dots \varrho(A) \longrightarrow H_A$ for any $A \in N_R$. Let Z be the axiom of R. By Lemma 6.1, we suppose that R is initial: i (resp. f) colours only H_Z (not the other right hand sides of R) and on a unique vertex $\overline{p}$ (resp. $\overline{q} \neq \overline{p}$). We assume that 0 is not a vertex of R and we take a new set of labels of arity 2:

$$\{A_{i,j} \mid A \in N_R \wedge 1 \leq i,j \leq \varrho(A)\} \cup \{Z'\}.$$

We define the *splitting* $\prec G \succ$ of any $(T_R \cup N_R)$-hypergraph G as being the graph:

$$\begin{aligned}\prec G \succ := \{s \xrightarrow{a} t \mid ast \in G \wedge a \in T_R\} \\ \cup \{s \xrightarrow{A_{i,j}} t \mid A \in N_R \wedge 1 \leq i,j \leq \varrho(A) \wedge \\ \exists s_1, \ldots, s_{\varrho(A)}(As_1 \ldots s_{\varrho(A)} \in G \wedge s = s_i \wedge t = s_j)\}\end{aligned}$$

and for $p, q \in V_G$ and $P \subseteq V_G$ with $0 \notin V_G$, we define for $p \neq q$

$$\begin{aligned}G_{p,q,P} &:= \{s \xrightarrow[\prec G \succ]{a} t \mid t \neq p \wedge s \neq q \wedge s,t \notin P\}_{|\{s|p \longrightarrow^* s \longrightarrow^* q\}} \\ G_{p,p,P} &:= \big(\{s \xrightarrow[\prec G \succ]{a} t \mid t \neq p \wedge s,t \notin P\} \\ &\qquad \cup \{s \xrightarrow{a} 0 \mid s \xrightarrow[\prec G \succ]{a} p\}\big)_{|\{s|p \longrightarrow^* s \longrightarrow^* 0\}}.\end{aligned}$$

This allows us to define the splitting of R as the following path grammar:

$$\begin{aligned}\prec R \succ := \{A_{i,j}12 \longrightarrow h_{i,j}\big((H_A)_{i,j,[\varrho(A)]-\{i,j\}}\big) \mid A \in N_R \wedge 1 \leq i,j \leq \varrho(A)\} \\ \cup \{Z'12 \longrightarrow \big(h\langle H_Z \rangle\big)_{|\{s|1 \longrightarrow^* s \longrightarrow^* 2\}}\}\end{aligned}$$

where $h_{i,j}$ is the vertex renaming of $(H_A)_{i,j,[\varrho(A)]-\{i,j\}}$ defined by

$$\begin{aligned}&h_{i,j}(i) = 1, \quad h_{i,j}(j) = 2, \quad h_{i,j}(x) = x \text{ otherwise}, \quad \text{for } i \neq j \\ &h_{i,i}(i) = 1, \quad h_{i,i}(0) = 2, \quad h_{i,i}(x) = x \text{ otherwise},\end{aligned}$$

and h is the vertex renaming of H_Z defined by

$$h(\overline{p}) = 1, \qquad h(\overline{q}) = 2, \qquad h(x) = x \text{ otherwise}.$$

We then put $\prec R \succ$ into a reduced form.

(ii) Let us show that $L(R, i, f) = L(\prec R \succ, Z')$. For any $A \in N_R$, we take a derivation

$$A1 \ldots \varrho(A) = H_0 \underset{R}{\Longrightarrow} H_1 \underset{R}{\Longrightarrow} \ldots H_n \underset{R}{\Longrightarrow} \ldots$$

we write $H_\omega = \bigcup_{n \geq 0}[H_n]$ and for every $1 \leq i,j \leq \varrho(A)$ and $0 \leq n \leq \omega$, we define the following languages:

$$\begin{aligned}L_n(R, A, i, j) &:= L\big((H_n)_{i,j,[\varrho(A)]-\{i,j\}}, i, j\big) \qquad \text{for } i \neq j \\ L_n(R, A, i, i) &:= L\big((H_n)_{i,i,[\varrho(A)]-\{i\}}, i, 0\big).\end{aligned}$$

Note that for any $A \in N_R$, any $i, j \in [\varrho(A)]$ and $n \geq 0$, we have

$$\begin{aligned} & L_n(R, A, i, j) \subseteq \big(T_R \cup \{A_{p,q} \mid A \in N_R \wedge p, q \in [\varrho(A)]\}\big)^* \\ \textit{and} \qquad & L_\omega(R, A, i, j) \subseteq T_R^*. \end{aligned}$$

Let us verify that for any $A \in N_R$ and $1 \leq i, j \leq \varrho(A)$, we have

$$L_\omega(R, A, i, j) = L(\prec R \succ, A_{i,j}).$$

As $L_\omega(R, A, i, j) = \bigcup_{n \geq 0} \big(L_n(R, A, i, j) \cap T_R^*\big)$, it is sufficient to prove by induction on $n \geq 0$ that

$$L_n(R, A, i, j) = L_n(\prec R \succ, A_{i,j}).$$

"$n = 0$": we have $L_0(R, A, i, j) = \{A_{i,j}\} = L_0(\prec R \succ, A_{i,j})$.
"$n = 1$": we have

$$L_1(R, A, i, j) = L((H_A)_{i,j,[\varrho(A)]-\{i,j\}}, i, j) = L_1(\prec R \succ, A_{i,j}).$$

"$n \Longrightarrow n + 1$":

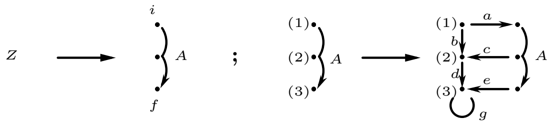
$$\begin{aligned} L_{n+1}(R, A, i, j) &= L_1(R, A, i, j)[L_n(R, B, p, q)/B_{p,q}] \\ &= L_1(\prec R \succ, A_{i,j})[L_n(\prec R \succ, B_{p,q})/B_{p,q}] \quad \text{by ind. hyp.} \\ &= L_{n+1}(\prec R \succ, A_{i,j}). \end{aligned}$$

Finally we have

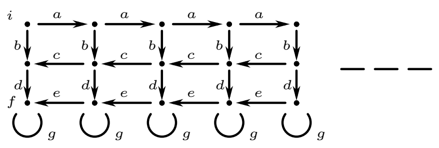
$$\begin{aligned} L(R, i, f) &= L(\text{Gen}(R), i, f) \\ &= L(\prec H_Z \succ, \overline{p}, \overline{q})[L_\omega(R, A, i, j)/A_{i,j}] \\ &= L_1(\prec R \succ, Z')[L(\prec R \succ, A_{i,j})/A_{i,j}] \\ &= L(\prec R \succ, Z'). \end{aligned}$$

Q.E.D. (Proposition 6.2)

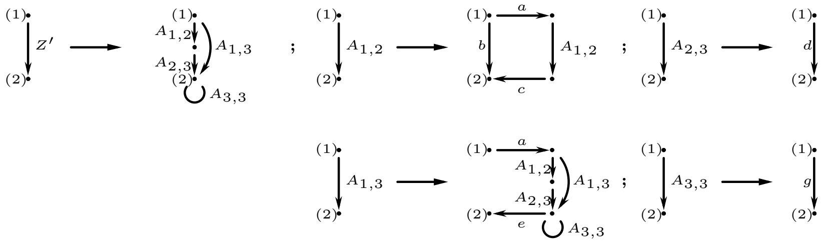
Let us illustrate Proposition 6.2 starting from the following grammar R:



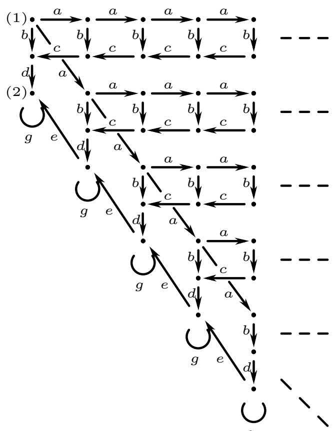
Its generated graph $R^\omega(Z)$ is given below.



Proposition 6.2 splits grammar R into the following grammar $\prec\!R\!\succ$ in reduced form:



We get the following generated graph $\prec\!R\!\succ^{\omega}(Z'12)$:



Obviously the graphs $R^{\omega}(Z)$ and $\prec\!R\!\succ^{\omega}(Z'12)$ are not isomorphic but by Proposition 6.2 they recognize the same language:

$$L(R^{\omega}(Z), i, f) = L(\prec\!R\!\succ, Z') = \{a^{m+n}bc^{m}d(eg^{*})^{n} \mid m, n \geq 0\}.$$

We now show that path grammars are language-equivalent to context-free grammars (on words). Recall that a *context-free grammar* P is a finite binary relation on words in which each left hand side is a letter called a non-terminal, and the remaining letters of P are terminals. By denoting N_P and T_P the respective sets of non-terminals and terminals of P, the rewriting $\underset{P}{\longrightarrow}$ according to P is the binary relation on $(N_P \cup T_P)^*$ defined by

$$UAV \underset{P}{\longrightarrow} UWV \text{ if } (A,W) \in P \text{ and } U,V \in (N_P \cup T_P)^*.$$

The derivation $\underset{P}{\longrightarrow}^*$ is the reflexive and transitive closure of $\underset{P}{\longrightarrow}$ with respect to composition. The language $L(P,U)$ generated by P from any $U \in (N_P \cup T_P)^*$ is the set of terminal words deriving from U:

$$L(P,U) := \{u \in T_P^* \mid U \underset{P}{\longrightarrow}^* u\}.$$

Path grammars and context-free grammars are language-equivalent with linear time translations.

Proposition 6.3 (Caucal-Dinh, [6])**.**

a) We can transform in linear time any path grammar R into a context-free grammar $\widehat{R}$ such that $L(R,A) = L(\widehat{R},A)$ for any $A \in N_R$.

b) We can transform in linear time any context-free grammar P into an acyclic path grammar $\vec{P}$ such that $L(P,A) = L(\vec{P},A)$ for any $A \in N_P$.

Proof. **(i)** The first transformation is analogous to the translation of any finite automaton into an equivalent right linear grammar. To each non-terminal $A \in N_R$, we take a vertex renaming h_A of $R(A12)$ such that $h_A(1) = A$ and $h_A(2) = \varepsilon$, and the image $\mathrm{Im}(h_A) - \{\varepsilon\}$ is a set of symbols with $\mathrm{Im}(h_A) \cap \mathrm{Im}(h_B) = \{\varepsilon\}$ for any $B \in N_R - \{A\}$. We define the following context-free grammar:

$$\widehat{R} := \{(h_A(s), ah_A(t)) \mid \exists A \in N_R (s \underset{R(A12)}{\xrightarrow{a}} t)\}.$$

Note that each right side of $\widehat{R}$ is a word of length at most 2, and the number of non-terminals of $\widehat{R}$ depends on the description length of R:

$$|N_{\widehat{R}}| = \Big(\sum_{A \in N_R} |V_{R(A12)}|\Big) - |N_R|.$$

For instance, the path grammar of Figure 6.2 is transformed into the following context-free grammar:

$$\begin{array}{lll} A = aC; & C = BD; & D = c \\ B = aF + bE; & E = aG + d; & F = AG \\ G = AH; & H = c & \end{array}$$

(ii) For the second transformation, we have $N_{\vec{P}} = N_P$ and for each $A \in N_P$, its right hand side in $\vec{P}$ is the set of distinct paths from 1 to 2 labelled by the right hand sides of A in P. We translate the context-free grammar P into the following acyclic path grammar:

$$\vec{P} := \{(A12, H_A) \mid A \in \mathrm{Dom}(P)\}$$

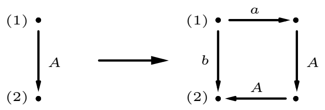
such that for each non-terminal $A \in \mathrm{Dom}(P)$, the graph H_A is the set of right hand sides of A in P starting from 1 and ending to 2:

$$\begin{aligned} H_A := \{1 \xrightarrow{B} (B,V) \mid (A,BV) \in P \wedge |B| = 1 \wedge V \neq \varepsilon\} \\ \cup \{(U,BV) \xrightarrow{B} (UB,V) \mid (A,UBV) \in P \wedge |B| = 1 \wedge U,V \neq \varepsilon\} \\ \cup \{(U,B) \xrightarrow{B} 2 \mid (A,UB) \in P \wedge |B| = 1 \wedge U \neq \varepsilon\} \\ \cup \{1 \xrightarrow{B} 2 \mid (A,B) \in P \wedge |B| = 1\} \\ \cup \{1 \xrightarrow{\varepsilon} 2 \mid (A,\varepsilon) \in P\}. \end{aligned}$$

Note that $N_P = N_{\vec{P}}$ and $T_P = T_{\vec{P}} - \{\varepsilon\}$. For instance the context-free grammar $\{(A,aAA),(A,b)\}$ generating from A the Łukasiewicz language, is translated into the acyclic path grammar reduced to the unique rule:

$$A12 \longrightarrow \{1 \xrightarrow{b} 2, 1 \xrightarrow{a} (a,AA), (a,AA) \xrightarrow{A} (aA,A), (aA,A) \xrightarrow{A} 2\}$$

and represented below:



Q.E.D. (Proposition 6.3)

Note that by using the two transformations of Proposition 6.3, we can transform in linear time any path grammar into a language equivalent acyclic path grammar. By Proposition 6.2 and Proposition 6.3 a), the recognized languages of regular graphs are generated by context-free grammars. The converse is true by Proposition 6.3 b).

Corollary 6.4. The regular graphs recognize exactly the context-free languages.

6.2 Deterministic languages

We now focus on the deterministic regular graphs. We say that a coloured graph G is *deterministic from* a colour i if i colours a unique vertex of G, and two arcs with the same source have distinct labels:

$$|G \cap iV_G| = 1 \text{ and } (p \xrightarrow[G]{a} q \wedge p \xrightarrow[G]{a} r \Longrightarrow q = r).$$

The languages recognized by the deterministic regular graphs

$$\mathrm{DAlg}(T^*) := \{L(G,i,f) \mid G \text{ regular and deterministic from } i \\ \wedge F_G \cap F_2 \subseteq T \wedge i,f \in F_1\}$$

are the languages recognized by the deterministic pushdown automata.

Proposition 6.5. The deterministic regular graphs recognize exactly the deterministic context-free languages.

Proof. Recall that a *deterministic pushdown automaton* S over an alphabet T of terminals is a finite subset of $PQ \times (T \cup \{\varepsilon\}) \times P^*Q$ where P, Q are disjoint alphabets of respectively stack letters and states, and such that S is deterministic for any $a \in T \cup \{\varepsilon\}$:

$$(xp \xrightarrow{a} uq \wedge xp \xrightarrow{a} vr) \Longrightarrow uq = vr$$

and each left-hand side of an ε-rule is not the left-hand side of a terminal rule:

$$(xp \xrightarrow{\varepsilon} uq \wedge xp \xrightarrow{a} vr) \Longrightarrow a = \varepsilon.$$

The language $L(\mathrm{Tr}(S), xp, F)$ recognized by S starting from an initial configuration xp and ending to a regular set $F \subseteq P^*Q$ of final configurations, is a *deterministic context-free language.*

(i) Let us verify that $L(\mathrm{Tr}(S), xp, F)$ is traced by a deterministic regular graph. We take two colours i and f. By Proposition 5.4 or more precisely by Corollary 5.5, the following coloured graph:

$$G := \mathrm{Tr}(S) \cup \{i(xp)\} \cup \{fu \mid u \in F\}$$

is a regular graph. Let R be a grammar generating G. We define the following grammar:

$$R' := \{(X', H') \mid (X, H) \in R\}$$

where for any hyperarc $fs_1 \ldots s_n$ of R, we associate the hyperarc

$$(fs_1 \ldots s_n)' := f[s_1] \ldots [s_n]$$

that we extend by union to any right hand side H of R:

$$H' := \{Y' \mid Y \in H\}$$

and such that for any vertex $s \in V_H$,

$$[s] := \{t \mid t \xrightarrow[{[H]}]{\varepsilon}{}^{*} s \vee s \xrightarrow[{[H]}]{\varepsilon}{}^{*} t\}.$$

Thus R' is without ε-arc and

$$L(R', i, f) = L(R, i, f) = L(G, i, f) = L(\mathrm{Tr}(S), xp, F).$$

(ii) Let i, f be colours and R be a grammar such that $\mathrm{Gen}(R)$ is deterministic from i. We want to show that $L(R, i, f)$ is a deterministic context-free language. By Proposition 4.4 (and 4.1), we assume that $\mathrm{Gen}(R)$ is accessible from i. By Lemma 3.11, we can assume that R is terminal outside. For any rule $(X, H) \in R$, we define

$$\mathrm{Out}(X(1)) := \{i \mid 1 < i \leq \varrho(X(1)) \wedge \exists a(X(i) \xrightarrow[\mathrm{Gen}(R,X)]{a})\}$$

the ranks of the input vertices which are source of an arc in the generated graph from X. Precisely $\big(\mathrm{Out}(A)\big)_{A \in N_R}$ is the least fixed point of the system: for each $(X, H) \in R$,

$$\begin{aligned}\mathrm{Out}(X(1)) = \{i \mid \exists a(X(i) \xrightarrow[{[H]}]{a})\}& \\ \cup \{i \mid \exists Y \in H \cap N_R V_H^* \ \exists j \in \mathrm{Out}(Y(1))(X(i) = Y(j))\}&.\end{aligned}$$

We rewrite non-terminal hyperarcs in the right hand sides of R until all the terminal arcs of input source are produced. We begin with the grammar:

$$R_0 := R$$

and having constructed a grammar R_n for $n \geq 0$, we choose a rule $(X, H) \in R_n$ and a non-terminal hyperarc $Y \in H \cap N_R V_H^*$ such that

$$V_X \cap \{Y(i) \mid i \in \mathrm{Out}(Y(1))\} \neq \varnothing$$

and we rewrite Y in H to get a hypergraph K *i. e.* $H \xrightarrow[R_n, Y]{} K$ in order to replace H by K in R_n:

$$R_{n+1} := (R_n - \{(X, H)\}) \cup \{(X, K)\}.$$

If such a choice is not possible, we finish with $\overline{R} := R_n$. As $\mathrm{Gen}(R)$ is deterministic, it is of bounded out-degree, hence $\overline{R}$ exists. By construction, $\overline{R}$ is equivalent to R:

$$R^\omega(X) = \overline{R}^\omega(X) \text{ for any } X \in \mathrm{Dom}(R) = \mathrm{Dom}(\overline{R}).$$

Furthermore $\overline{R}$ satisfies the following property:

$$\forall (X,H) \in \overline{R}\ \forall Y \in H \cap N_{\overline{R}} V_H^* \ (V_X \cap \{Y(i) \mid i \in \mathrm{Out}(Y(1))\} = \varnothing)$$

meaning that any input which is a vertex of a non-terminal hyperarc Y cannot be a source of an arc in the generated graph from Y. For each rule $(X,H) \in \overline{R}$, we denote

$$\mathrm{InOut}(X(1)) := \bigcup \{V_X \cap V_Y \mid Y \in H \wedge Y(1) \in N_{\overline{R}}\}$$

the set of input-output vertices; and for each $s \in \mathrm{InOut}(X(1))$, we take a new vertex $s' \notin V_H$ and to any non-terminal hyperarc $Y \in H$ with $Y(1) \in N_{\overline{R}}$, we associate the hyperarc $Y' = Y(1)Y(2)' \ldots Y(|Y|)'$ with $s' := s$ for any $s \in V_H - \mathrm{InOut}(X(1))$. We define the grammar R' by associating to each rule $(X,H) \in \overline{R}$, the following rule:

$$X \longrightarrow [H] \cup \{Y' \mid Y \in H \wedge Y(1) \in N_{\overline{R}}\} \cup \{s' \xrightarrow{\varepsilon} s \mid s \in \mathrm{InOut}(X(1))\}.$$

Thus $L(R,i,f) = L(R',i,f)$ and the graph $\mathrm{Gen}(R')$ is of finite degree, deterministic over $T_R \cup \{\varepsilon\}$ and such that any source of an ε-arc is not source of an arc labelled in T_R. By Theorem 5.11, $\mathrm{Gen}(R')$ is the transition graph of a pushdown automaton S accessible from an initial configuration c_0 with a regular set F of final configurations:

$$\mathrm{Gen}(R') = \mathrm{Tr}(S)_{\{c \mid c_0 \longrightarrow^* c\}} \cup \{ic_0\} \cup \{fc \mid c \in F\}.$$

Finally S is a deterministic pushdown automaton recognizing the language:

$$L(\mathrm{Tr}(S), i, F) = L(R', i, f) = L(R, i, f).$$

Q.E.D. (Proposition 6.5)

Due to a lack of space (and time), we have only presented a first (and partial) survey on deterministic graph grammars. After defining suitable normal forms, we explored the notion of regularity of a graph with respect to a finite-index graduation of its vertices.

Together with a generic representation of grammar-generated graphs, this yields a canonical representation of any given regular graph. These definitions and techniques constitute a basic toolkit for conveniently manipulating deterministic graph grammars. As an illustration, we were able to prove in a self-contained way several known structural results concerning regular graphs, the most important being their links with the transition graphs of pushdown automata.

This is only a first step in studying deterministic graph grammars, and many interesting developments remain to be explored. We hope that this paper might encourage further work on the subject. In particular, we believe that grammars will prove an invaluable tool in extending finite graph theory to the class of regular graphs, as well as finite automata theory to some sub-families of context-free languages. Some efforts in these directions have already begun to appear [5, 6]. Other leads for further research concern the use of grammars as a tool for more general computations (a particular case is Proposition 4.4), and the design of geometrical proofs for results related to context-free languages (e.g. the standard pumping lemma).

Let us conclude with a natural question: how can one extend deterministic graph grammars in order to generate the structure of infinite automata [10], in particular those associated to pushdown automata using stack of stacks [11, 3]?

References

[1] M. Benois. Parties rationnelles du groupe libre. *C. R. Acad. Sci. Paris Sér. A-B*, 269:A1188–A1190, 1969.

[2] J. R. Büchi. Regular canonical systems. *Arch. Math. Logik Grundlagenforsch.*, 6:91–111 (1964), 1964.

[3] A. Carayol. *Automates infinis, logiques et langages.* PhD thesis, University of Rennes 1, 2006.

[4] D. Caucal. On the regular structure of prefix rewriting. In A. Arnold, editor, *CAAP*, volume 431 of *Lecture Notes in Computer Science*, pages 87–102. Springer, 1990.

[5] D. Caucal. Synchronization of pushdown automata. In O. H. Ibarra and Z. Dang, editors, *Developments in Language Theory*, volume 4036 of *Lecture Notes in Computer Science*, pages 120–132. Springer, 2006.

[6] D. Caucal and Dinh Trong Hieu. Path algorithms on regular graphs. In E. Csuhaj-Varjú and Z. Ésik, editors, *FCT*, volume 4639 of *Lecture Notes in Computer Science*, pages 199–212. Springer, 2007.

[7] B. Courcelle. The monadic second-order logic of graphs, ii: Infinite graphs of bounded width. *Mathematical Systems Theory*, 21(4):187–221, 1989.

[8] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theor. Comput. Sci.*, 37:51–75, 1985.

[9] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific, 1997.

[10] W. Thomas. A short introduction to infinite automata. In W. Kuich, G. Rozenberg, and A. Salomaa, editors, *Developments in Language Theory*, volume 2295 of *Lecture Notes in Computer Science*, pages 130–144. Springer, 2001.

[11] W. Thomas. Constructing infinite graphs with a decidable mso-theory. In B. Rovan and P. Vojtás, editors, *MFCS*, volume 2747 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2003.

Quantifier-free definable graph operations preserving recognizability[*]

Bruno Courcelle

Laboratoire Bordelais de Recherche en Informatique
and Institut Universitaire de France
Université Bordeaux 1
351, cours de la Libération
33405 Talence cedex, France
`courcell@labri.fr`

Abstract

We show that an operation on graphs, and more generally, on relational structures that has an inverse definable by a monadic second-order transduction preserves the family of recognizable sets.

1 Introduction

Several algebras of graphs, and more generally of relational structures, can be defined in terms of disjoint union as unique binary operation and of several unary operations defined by quantifier-free formulas. These algebras are the basis of the extension to graphs and hypergraphs of the *theory of formal languages* in a universal algebra setting.

In every algebra, one can define two families of subsets, the family of *equational sets* which generalizes the family of context-free languages, and the family of *recognizable sets* which generalizes the family of recognizable languages. Equational sets are defined as least solutions of systems of recursive set equations and not in terms of rewriting rules. Recognizable sets are defined in terms of finite congruences and not in terms of finite automata. These purely algebraic definitions which are due to Mezei and Wright [8] have the advantage of being applicable to every algebra, whereas rewriting systems and finite automata cannot. One obtains definitions of "context-free" sets of graphs which avoid the cumbersome analysis of the confluence of particular graph rewriting systems. The basic definitions and facts regarding these notions can be found in [2, 5, 6, 7].

[*] There has been a long cooperation between the Logic and Computer Science groups in RWTH and in LaBRI, which started in 1984 with the visit of W. Thomas in Bordeaux as invited speaker to the Colloquium on Trees (CAAP). This note extends my article with A. Blumensath, which is a recent outcome of this cooperation, and hopefully, not the last one. I thank A. Blumensath for helpful comments.

Certain closure properties of the families of equational and recognizable sets are valid at the most general level. In particular, the family of equational sets of an algebra $\mathbf{M}$ is closed under union, intersection with the recognizable sets and under the operations of this algebra. For an example, the concatenation of two equational (i.e., context-free) languages is equational. The family of recognizable sets of an algebra $\mathbf{M}$ is closed under union, intersection and difference, and under the inverses of *unary derived operations* (the operations defined by finite terms over the signature of $\mathbf{M}$). The family of recognizable languages (alternatively called *rational* or *regular*) is also closed under concatenation, but this is not a special case of a general algebraic property, by contrast with the case of equational languages. In a general algebra, the family of recognizable sets is not always closed under the operations of the algebra. That these closure properties are true depends on particular properties of the considered algebra.

Which properties of an algebra ensure that the family of recognizable sets is closed under the operations of the algebra?

Two types of answers can be given: algebraic and logical answers. Algebraic answers have been given in [4], an article motivated by the study of the so-called *Hyperedge Replacement (HR) algebra of graphs and hypergraphs*, that is connected in a natural way to the notion of *tree-width* [6]. The results of the article [4] can be applied to the case of languages in a quite simple way: the property of words that $uv = wx$ if and only if there exists a word z such that $u = wz$ and $zv = x$, or $uz = w$ and $v = zx$ implies that the concatenation of two recognizable languages is recognizable, by a proof that uses only finite congruences and no construction of automata.

Another important case is that of an associative and commutative operation, a useful example being the disjoint union of graphs and relational structures denoted by $\oplus$. The corresponding (commutative) concatenation of subsets preserves recognizability because the equality $u \oplus v = w \oplus x$ is equivalent to the existence of y_1, y_2, y_3, y_4 such that $u = y_1 \oplus y_2$, $v = y_3 \oplus y_4$, $w = y_1 \oplus y_3$ and $x = y_2 \oplus y_4$.

The article [4] establishes that the family of HR-recognizable sets of graphs is closed under the operations of the HR-algebra. One might think that these results would extend without difficulties to the somewhat similar *Vertex Replacement (VR) algebra of graphs* (which we define below). However this is not the case as we shall see in the next section.

In the present article, we do not answer the above question in full generality, but we give a sufficient condition for algebras of finite relational structures (hence also of finite graphs) whose operations are disjoint union and unary operations defined by quantifier-free formulas, that we call *quantifier-free definable operations.* We are particularly interested by these algebras because every *monadic second-order definable set* of finite relational structures is recognizable (see Theorem 5.1 below). Our main result (Theorem 5.4) is a direct consequence of a result of [2]. It relates the preservation of recognizability in the algebra of relational structures under a unary operation to the existence an inverse for this operation that is a *monadic second-order transduction.* The present article continues the exploration done in particular in [1, 2, 3, 6, 7] of the deep links between algebraic and logical properties, more precisely here, between recognizability and monadic second-order logic.

2 The VR-algebra of simple graphs.

Graphs are finite, simple (without multiple edges), directed, and loop-free. Let C be a countable set of labels containing the set of nonnegative integers. A *C-graph* is a graph G given with a total mapping lab_G from its vertex set V_G to C. Hence G is defined as a triple $\langle V_G, \mathrm{edg}_G, \mathrm{lab}_G\rangle$ where edg_G is the binary edge relation. We call $\mathrm{lab}_G(v)$ the *label* of a vertex v. We denote by $\pi(G)$ the finite set $\mathrm{lab}_G(V_G) \subseteq C$, and we call it the *type* of G. The operations on C-graphs are the following ones:

1. We define a constant $\mathbf{1}$ to denote an isolated vertex labelled by 1.
2. For $i, j \in C$ with $i \neq j$, we define a unary function $\mathrm{add}_{i,j}$ such that
$$\mathrm{add}_{i,j}(\langle V_G, \mathrm{edg}_G, \mathrm{lab}_G\rangle) = \langle V_G, \mathrm{edg}'_G, \mathrm{lab}_G\rangle$$
where edg'_G is edg_G augmented with the set of pairs (u, v) such that $\mathrm{lab}_G(u) = i$ and $\mathrm{lab}_G(v) = j$. In order to add undirected edges (considered as pairs of opposite directed edges), we take
$$\mathrm{add}_{i,j}(\mathrm{add}_{j,i}(\langle V_G, \mathrm{edg}_G, \mathrm{lab}_G\rangle)).$$
3. We let also $\mathrm{ren}_{i\to j}$ be the unary function such that
$$\mathrm{ren}_{i\to j}(\langle V_G, \mathrm{edg}_G, \mathrm{lab}_G\rangle) = \langle V_G, \mathrm{edg}_G, \mathrm{lab}'_G\rangle$$
where $\mathrm{lab}'_G(v) = j$ if $\mathrm{lab}_G(v) = i$, and $\mathrm{lab}'_G(v) = \mathrm{lab}_G(v)$, otherwise. This mapping relabels into j every vertex label i.
4. Finally, we use the binary operation $\oplus$ that makes the union of disjoint copies of its arguments. Hence the graph $G \oplus H$ is well-defined up to isomorphism.

We denote by F^{VR} the countable set of all these operations, including the constant $\mathbf{1}$. The *VR-algebra* has for domain the set $\mathcal{G}$ of all isomorphism classes of C-graphs and the operations of F^{VR}. A well-formed term t written with the symbols of F^{VR} defines a C-graph $G = \mathrm{val}(t)$, actually a graph up to isomorphism. However, $\mathrm{val}(t)$ can be defined as a "concrete" graph with vertex set $Occ_{\mathbf{1}}(t)$ the set of occurrences in t of the constant $\mathbf{1}$.

A set of C-graphs L is *VR-recognizable* if there exists an F^{VR}-congruence $\approx$ on $\mathcal{G}$ such that

1. $G \approx H$ implies $\pi(G) = \pi(H)$
2. for each finite subset D of C, the congruence $\approx$ has finitely many equivalence classes of graphs of type D,
3. L is the union of a finite set of equivalence classes of $\approx$.

We shall prove below that the disjoint union and the renaming operations $\mathrm{ren}_{i \to j}$ preserve VR-recognizability. (A more complicated proof can be based on the algebraic lemmas of [4].) However :

Proposition 2.1. The operation $\mathrm{add}_{a,b}$ does not preserve recognizability. The operation that deletes all edges does not either.

Proof. Here is a counter-example. One takes the set L of finite directed graphs G of type $\{a, b\}$ consisting of pairwise nonadjacent edges linking one vertex labelled by a to one vertex labelled by b. Hence, we have as many a-labelled vertices as b-labelled ones. This set is definable in monadic second-order logic (and even in first-order logic) hence is VR-recognizable by a general theorem (see [3, 6], Theorem 5.1 below). The set $K = \mathrm{add}_{a,b}(L)$ consists of complete bipartite graphs $K_{n,n}$. And this set is not recognizable, because otherwise, so would be the set of terms of the form $\mathrm{add}_{a,b}([\mathbf{a} \oplus (\mathbf{a} \oplus (...\mathbf{a}))..)] \oplus [\mathbf{b} \oplus (...(\mathbf{b} \oplus \mathbf{b})..)])$ having n occurrences of $\mathbf{a}$ defined as $\mathrm{ren}_{1 \to a}(\mathbf{1})$ and n occurrences of $\mathbf{b}$ defined as $\mathrm{ren}_{1 \to b}(\mathbf{1})$ with $n > 0$. By a standard pumping argument this set is not recognizable. The proof is similar for the operation that deletes all edges. One uses the terms $[\mathbf{a} \oplus (\mathbf{a} \oplus (...\mathbf{a}))..)] \oplus [\mathbf{b} \oplus (...(\mathbf{b} \oplus \mathbf{b})..)]$. Q.E.D.

We now describe the logical setting that will help to investigate recognizability. We formulate it not only for graphs but for finite relational structures.

3 Relational structures and monadic second-order logic

Let $R = \{A, B, C, ...\}$ be a finite set of *relation symbols* each of them given with a nonnegative integer $\varrho(A)$ called its *arity*. We denote by $\mathcal{STR}(R)$ the set of *finite* R-structures $S = \langle D_S, (A_S)_{A \in R} \rangle$ where $A_S \subseteq D_S^{\varrho(A)}$ if $A \in R$

is a relation symbol, and D_S is the *domain* of S. If R consists of relation symbols of arity one or two we say that the structures in $\mathcal{STR}(R)$ are *binary.* Binary structures can be seen as vertex- and edge-labelled graphs. If we have several binary relations say A, B, C, the corresponding graphs have edges with labels A, B, C.

Monadic Second-order logic (MS logic for short) is the extension of *First-Order logic* with variables denoting subsets of the domains of the considered structures and atomic formulas of the form $x \in X$ expressing the membership of x in a set X. We shall denote by $\mathrm{MS}(R, W)$ the set of *Monadic second-order* formulas written with the set R of relation symbols and having their free variables in a set W consisting of *first-order and set variables.*

As a typical and useful example, we give an MS formula with free variables x and y expressing that (x, y) belongs to the reflexive and transitive closure of a binary relation A :

$$\forall X(x \in X \wedge \forall u, v[(u \in X \wedge A(u,v)) \Longrightarrow v \in X] \Longrightarrow y \in X).$$

If the relation A is not given in the considered structures but is defined by an MS formula, then one replaces $A(u, v)$ by this formula with appropriate substitutions of variables.

A subset of $\mathcal{STR}(R)$ is *MS-definable* if it is the set of finite models of a *monadic second-order sentence*, i.e., of an MS formula without free variables. Such a set is closed under isomorphism.

4 Monadic second-order transductions

Monadic second-order formulas can be used to define transformations of graphs and relational structures. As in language theory, a binary relation $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{B}$ where $\mathcal{A}$ and $\mathcal{B}$ are sets of words, graphs or relational structures is called a *transduction*: $\mathcal{A} \to \mathcal{B}$. An *MS transduction* is a transduction specified by MS formulas. It transforms a structure S, given with an n-tuple of subsets of its domain called the *parameters*, into a structure T, the domain of which is a subset of $D_S \times [k]$, (where $[k] = \{1, ..., k\}$). It is *noncopying* if $k = 1$. The general definition can be found in [1, 2, 6]. We only define noncopying MS transductions which are needed in this article.

We let R and Q be two finite sets of relation symbols. Let W be a finite set of set variables, called *parameters.* A (Q, R)-*definition scheme* is a tuple of formulas of the form $\Delta = (\varphi, \psi, (\theta_A)_{A \in Q})$ where $\varphi \in \mathrm{MS}(R, W), \psi \in \mathrm{MS}(R, W \cup \{x_1\})$, and $\theta_A \in \mathrm{MS}(R, W \cup \{x_1, \cdots, x_{\varrho(A)}\})$, for $A \in Q$.

These formulas are intended to define a structure T in $\mathcal{STR}(Q)$ from a structure S in $\mathcal{STR}(R)$. Let $S \in \mathcal{STR}(R)$, let γ be a W-assignment in S.

A Q-structure T with domain $D_T \subseteq D_S$ is *defined in* (S, γ) by Δ if

1. $(S, \gamma) \models \varphi$
2. $D_T = \{d \mid d \in D_S, (S, \gamma, d) \models \psi\}$
3. for each A in Q : $A_T = \{(d_1, \cdots, d_t) \in D_T^t \mid (S, \gamma, d_1, \cdots, d_t) \models \theta_A\}$, where $t = \varrho(A)$.

Since T is associated in a unique way with S, γ and Δ whenever it is defined, i.e., whenever $(S, \gamma) \models \varphi$, we can use the functional notation $\mathrm{def}_\Delta(S, \gamma)$ for T. The *transduction defined by* Δ is the binary relation :

$$\mathrm{def}_\Delta := \{(S, T) \mid T = \mathrm{def}_\Delta(S, \gamma) \text{ for some } W\text{-assignment } \gamma \text{ in } S\}.$$

A transduction $f \subseteq \mathcal{STR}(R) \times \mathcal{STR}(Q)$ is a noncopying *MS transduction* if it is equal to def_Δ (up to isomorphism) for some (Q, R)-definition scheme Δ. We shall also write functionally $\mathrm{def}_\Delta(S) := \{\mathrm{def}_\Delta(S, \gamma) \mid \gamma$ is a W-assignment in $S\}$. A definition scheme without parameters defines a *parameterless MS transduction*, which is actually a partial function: $\mathcal{STR}(R) \longrightarrow \mathcal{STR}(Q)$.

A *quantifier-free definable operation (a QF operation in short)* is a parameterless noncopying MS-transduction $\delta : \mathcal{STR}(R) \longrightarrow \mathcal{STR}(Q)$ defined by a scheme $\Delta = (\varphi, \psi, (\theta_A)_{A \in Q})$ such that the formula φ is equivalent to `True`, and the formulas θ_A are without quantifiers (whence also without set variables). This implies that δ is total. Furthermore, we say that such an operation is *nondeleting* if the formula ψ is equivalent to `True`. This condition implies that the domains of S and of $\delta(S)$ are the same.

A labelled graph $\langle V_G, \mathrm{edg}_G, \mathrm{lab}_G \rangle$ of type contained in D will be represented by the relational structure $\lfloor G \rfloor = \langle V_G, \mathrm{edg}_G, p_{aG}, ..., p_{dG} \rangle$ where $D = \{a, ..., d\}$ and $p_{xG}(u)$ is true if and only if $\mathrm{lab}_G(u) = x$. Through this representation, the unary operations $\mathrm{add}_{a,b}$ and $\mathrm{ren}_{a \to b}$ are quantifier-free. This means that for some QF operation α, we have $\alpha(\lfloor G \rfloor) = \lfloor \mathrm{add}_{a,b}(G) \rfloor$ for all graphs G of type contained in D, and similarly for $\mathrm{ren}_{a \to b}$.

The composition of two transductions is defined as their composition as binary relations. If they are both partial functions, then one obtains the composition of these functions. The *inverse image* of a set $L \subseteq \mathcal{STR}(Q)$ under a transduction $\delta : \mathcal{STR}(R) \longrightarrow \mathcal{STR}(Q)$ is the set of elements S of $\mathcal{STR}(R)$ such that $\delta(S) \cap L$ is not empty. It is denoted by $\delta^{-1}(L)$. (Equality of structures is understood up to isomorphism, hence $\delta^{-1}(L)$ is closed under isomorphisms.)

Proposition 4.1 (Courcelle, [6]).

1. The composition of two MS transductions is an MS transduction.
2. The inverse image of an MS-definable set of structures under an MS transduction is MS-definable.

5 The many-sorted algebra of relational structures

We now make the family of sets $\mathcal{STR}(R)$ for all relational signatures R into a many-sorted algebra **STR**, where each R is a sort and $\mathcal{STR}(R)$ is the corresponding domain. Here are the operations. First we define a disjoint union $\oplus : \mathcal{STR}(R)\times \mathcal{STR}(Q) \longrightarrow \mathcal{STR}(R \cup Q)$ for each pair of sorts (R, Q) (using the same notation for all of these operations). Then we also let in the signature all QF operations : $\mathcal{STR}(R) \longrightarrow \mathcal{STR}(Q)$ for all pairs of sorts (R, Q). For each pair (R, Q) there are actually only finitely many such operations (see [7, Appendix A]). We take the constant $*$ denoting the structure in $\mathcal{STR}(\varnothing)$ with a single element. We could actually take other constants, this would not affect the results stated below because recognizability does not depend on the set of constants. We let F^{QF} be this signature. The notation refers to the role of QF operations.

A subset of $\mathcal{STR}(R)$ is QF-*recognizable* if it is a (finite) union of classes of an F^{QF}-congruence on **STR** (equivalent elements must have the same sort) that has finitely many classes in each domain $\mathcal{STR}(R)$.

The labelled graphs having a type included in a finite set D are represented by relational structures $\lfloor G \rfloor = \langle V_G, \mathrm{edg}_G, p_{aG}, ..., p_{dG}\rangle$ in $\mathcal{STR}(\{\mathrm{edg}\} \cup \{p_a, ..., p_d\})$ where $D = \{a, ..., d\}$. A set of labelled graphs is VR-recognizable if and only if it is QF-recognizable, and it is VR-equational if and only if it is QF-equational [2, Theorem 68].

Theorem 5.1 (Courcelle, [3, 6])**.** If a subset of $\mathcal{STR}(R)$ is MS-definable, then it is QF-recognizable.

Theorem 5.2 (Blumensath-Courcelle, [2, Theorem 51])**.** The inverse image of a QF-recognizable set of relational structures under an MS transduction is QF-recognizable.

The following definition is new.

Definition 5.3. Let θ be a mapping that associates with every structure S in $\mathcal{STR}(R)$ a structure T in $\mathcal{STR}(Q)$ with same domain. It is *MS-invertible* if there exists a noncopying and nondeleting MS transduction ξ with set of parameters W such that, for all structures S and T:

1. if $\theta(S) = T$, then there exists a W-assignment γ such that $\xi(T, \gamma) = S$,
2. for every W-assignment γ such that $\xi(T, \gamma)$ is defined, we have $\theta(\xi(T, \gamma)) = T$.

As an example, we can observe that the operation $\mathrm{ren}_{a\longrightarrow b}$ is MS-invertible. Let $H = \mathrm{ren}_{a\longrightarrow b}(G)$ be obtained from G by replacing each vertex label a by b. This means that the sets X and Y of vertices labelled by a and by b are made into a unique set $X \cup Y$, the set of vertices of H labelled by b. To recover G from H, it is enough to use a set parameter Z that guesses, among the vertices labelled by b those which were originally labelled by a. Clearly, for each set Z of vertices labelled by b, one obtains a graph G such that $H = \mathrm{ren}_{a \longrightarrow b}(G)$, and every such G is of this form.

On the contrary, the operation $\mathrm{add}_{a,b}$ is not MS-invertible: the inverse MS-transduction would need to guess a set of edges to be deleted. This is not possible without using edge set quantifications, which is not what we are doing here (but can be done in relation with the HR-algebra, see [1, 6]). However, the restriction of $\mathrm{add}_{a,b}$ to the set of graphs that have no edge from an a-labelled vertex to a b-labelled one is MS-invertible, and its inverse MS-transduction is parameterless.

Theorem 5.4. Every MS-invertible mapping preserves QF-recognizability.

Proof. Let θ be an MS-invertible mapping : $\mathcal{STR}(R) \longrightarrow \mathcal{STR}(Q)$ with inverse MS transduction ξ, using a set of parameters W. Let $L \subseteq \mathcal{STR}(R)$ be recognizable. We claim that $\theta(L) = \xi^{-1}(L)$, which will yield the result by Theorem 5.2.

If $T = \theta(S), S \in L$ there exists a W-assignment γ such that $\xi(T, \gamma) = S$, hence T belongs to $\xi^{-1}(L)$. Conversely, if $T \in \xi^{-1}(L)$, then $\xi(T, \gamma) \in L$ for some W-assignment γ hence $\theta(\xi(T, \gamma)) = T$ and $T \in \theta(L)$. Q.E.D.

The proof of [2, Theorem 51] uses the fact that the QF operation that deletes a unary relation preserves recognizability [2, Proposition 58]. Such an operation is clearly MS-invertible. The proof of [2, Proposition 58] is done with the algebraic techniques of [4]. (Since recognizability is an algebraic notion, algebraic constructions must be used somewhere.)

Note that the same proof yields that MS-invertible QF operations preserve MS-definability, whereas a QF operation like $\mathrm{add}_{a,b}$ does not.

Question 5.5. Which QF operations are MS-invertible?

It does not seem easy to give necessary and sufficient conditions. We have already given examples and counter-examples (with help of Proposition 2.1). The operation that relabels a binary symbol, say A into B, does not preserve recognizability. The proof is as in Proposition 2.1. The following is a related question.

Question 5.6. Does there exist a QF operation that is not MS-invertible but preserves QF-recognizability?

We now consider in a similar way the disjoint union $\oplus : \mathcal{STR}(R) \times \mathcal{STR}(Q) \longrightarrow \mathcal{STR}(R \cup Q)$. Let mark be a unary relation not in $R \cup Q$. Let us define the *marked* disjoint union $\oplus_{\mathrm{mark}} : \mathcal{STR}(R) \times \mathcal{STR}(Q) \longrightarrow$

$\mathcal{STR}(R \cup Q \cup \{\text{mark}\})$, such that $S \oplus_{\text{mark}} T = S \oplus T$ augmented with $\text{mark}(u)$ for every u in the domain of T. It clear that there are two parameterless QF operations ξ_1 and ξ_2 such that for every structure Z

1. $\xi_1(Z)$ and $\xi_2(Z)$ are defined if and only if $Z = S \oplus_{\text{mark}} T$ for some S in $\mathcal{STR}(R)$ and some T in $\mathcal{STR}(Q)$,

2. and if this is the case S and T as in 1. are unique and
$$Z = \xi_1(Z) \oplus_{\text{mark}} \xi_2(Z).$$

Theorem 5.7. Disjoint union preserves QF-recognizability.

Proof. Let $L \subseteq \mathcal{STR}(R)$ and $K \subseteq \mathcal{STR}(Q)$ be recognizable. Let $M = L \oplus_{\text{mark}} K$. We claim that $M = \xi_1^{-1}(L) \cap \xi_2^{-1}(K)$.

If $Z = S \oplus_{\text{mark}} T \in M$, $S \in L$, $T \in K$, then $\xi_1(Z) = S$ and $\xi_2(Z) = T$, hence $Z \in \xi_1^{-1}(S)$ and $Z \in \xi_2^{-1}(T)$, $Z \in \xi_1^{-1}(L) \cap \xi_2^{-1}(K)$. Conversely, if $Z \in \xi_1^{-1}(L) \cap \xi_2^{-1}(K)$ then $\xi_1(Z) = S \in L$ and $\xi_2(Z) = T \in K$ and $Z = S \oplus_{\text{mark}} T \in L \oplus_{\text{mark}} K = M$. This proves the claim, and by Theorem 5.2, $\xi_1^{-1}(L)$ and $\xi_2^{-1}(K)$ are recognizable and so is their intersection M.

The image of M under the QF operation that deletes mark is recognizable by [2, Proposition 58], and this image is $L \oplus K$. Q.E.D.

A similar proof shows that disjoint union preserves MS-definability.

The family of recognizable sets of relational structures is thus preserved under disjoint union and MS-invertible QF operations. These operations form a subsignature $\mathrm{F}^{\text{inv-QF}}$ of F^{QF}. From general facts discussed in depth in [2], it follows that the $\mathrm{F}^{\text{inv-QF}}$-equational sets form a subfamily of the QF-equational ones, and that the QF-recognizable sets form a subfamily of the $\mathrm{F}^{\text{inv-QF}}$-recognizable ones. If those two inclusions are equalities, then we say that the signatures $\mathrm{F}^{\text{inv-QF}}$ and F^{QF} are *equivalent.*

Question 5.8. Is the signature $\mathrm{F}^{\text{inv-QF}}$ equivalent to F^{QF}?

Let us first go back to the case of the VR-algebra. The signature F^{VR} is equivalent to the restriction to graphs of the signature F^{QF} ([3] and [7, Theorem 4.5]). Furthermore, one can eliminate from F^{VR} the operations $\text{add}_{a,b}$ and replace them by *derived operations* of the form $G \otimes_\lambda H = \lambda(G \oplus H)$ where λ is a composition of $\text{add}_{a,b}$ operations and of relabellings that only create edges between G and H (and not inside G or H). One obtains an algebra of graphs with the same recognizable sets [7, Proposition 4.9] and the same equational sets. For each operation $\otimes_\lambda$ a pair of inverse MS-transductions like ξ_1 and ξ_2 for $\oplus$ can be defined so that the operations $\otimes_\lambda$ preserve recognizability. In this way we can handle the problem of the non-MS-invertibility of $\text{add}_{a,b}$.

Could we do the same for F^{QF}? There is another difficulty with the QF operations that delete relations of arity more than one, and those which rename them, because, as observed above, they are not MS-invertible. A subsignature of F^{QF} equivalent to it is defined in [2] but it uses these non-MS-invertible operations. We leave open Question 5.8.

As final comment, we observe that the result of [4] stating that the family of HR-recognizable sets of graphs and hypergraphs is closed under the operations of the HR-algebra can be proved by the tools used for Theorems 5.4 and 5.7.

References

[1] A. Blumensath, T. Colcombet, and C. Löding. Logical theories and compatible operations. In *Automata and Logic: History and Perspectives*, volume 2 of *Texts in Logics and Games*. Amsterdam University Press, Amsterdam, 2007. This volume.

[2] A. Blumensath and B. Courcelle. Recognizability, hypergraph operations, and logical types. *Inform. and Comput.*, 204(6):853–919, 2006.

[3] B. Courcelle. The monadic second-order logic of graphs vii: Graphs as relational structures. *Theor. Comput. Sci.*, 101(1):3–33, 1992.

[4] B. Courcelle. Recognizable sets of graphs: Equivalent definitions and closure properties. *Mathematical Structures in Computer Science*, 4(1):1–32, 1994.

[5] B. Courcelle. Basic notions of universal algebra for language theory and graph grammars. *Theor. Comput. Sci.*, 163(1&2):1–54, 1996.

[6] B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars*, pages 313–400. World Scientific, 1997.

[7] B. Courcelle and P. Weil. The recognizability of sets of graphs is a robust property. *Theor. Comput. Sci.*, 342(2-3):173–228, 2005.

[8] J. Mezei and J. B. Wright. Algebraic automata and context-free sets. *Information and Control*, 11(1/2):3–29, 1967.

First-order definable languages*

Volker Diekert[1]
Paul Gastin[2]

[1] Institut für Formale Methoden der Informatik
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart, Germany
diekert@fmi.uni-stuttgart.de

[2] Laboratoire Spécification et Vérification
École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex, France
Paul.Gastin@lsv.ens-cachan.fr

Abstract

We give an essentially self-contained presentation of some principal results for first-order definable languages over finite and infinite words. We introduce the notion of a *counter-free* Büchi automaton; and we relate counter-freeness to *aperiodicity* and to the notion of *very weak alternation.* We also show that aperiodicity of a regular ∞-language can be decided in polynomial space, if the language is specified by some Büchi automaton.

1 Introduction

The study of regular languages is one of the most important areas in formal language theory. It relates logic, combinatorics, and algebra to automata theory; and it is widely applied in all branches of computer sciences. Moreover it is the core for generalizations, e.g., to tree automata [26] or to partially ordered structures such as Mazurkiewicz traces [6].

In the present contribution we treat first-order languages over finite and infinite words. First-order definability leads to a subclass of regular languages and again: it relates logic, combinatorics, and algebra to automata theory; and it is also widely applied in all branches of computer sciences. Let us mention that first-order definability for Mazurkiewicz traces leads essentially to the same picture as for words (see, e.g., [5]), but nice characterizations for first-order definable sets of trees are still missing.

The investigation on first-order languages has been of continuous interest over the past decades and many important results are related to the efforts

* We would like to thank the anonymous referee for the detailed report.

of Wolfgang Thomas [31, 32, 33, 34, 35]. We also refer to his influential contributions in the handbooks of Theoretical Computer Science [36] and of Formal Languages [37].

We do not compete with these surveys. Our plan is more modest. We try to give a self-contained presentation of some of the principal characterizations of first-order definable languages in a single paper. This covers description with star-free expressions, recognizability by aperiodic monoids and definability in linear temporal logic. We also introduce the notion of a *counter-free* Büchi automaton which is somewhat missing in the literature so far. We relate counter-freeness to the aperiodicity of the transformation monoid. We also show that first-order definable languages can be characterized by very weak alternating automata using the concept of aperiodic automata. In some sense the main focus in our paper is the explanation of the following theorem.

Theorem 1.1. Let L be a language of finite or infinite words over a finite alphabet. Then the following assertions are equivalent:

1. L is first-order definable.
2. L is star-free.
3. L is aperiodic.
4. L is definable in the linear temporal logic LTL.
5. L is first-order definable with a sentence using at most 3 names for variables.
6. L is accepted by some counter-free Büchi automaton.
7. L is accepted by some aperiodic Büchi automaton.
8. L is accepted by some very weak alternating automaton.

Besides, the paper covers related results. The translation from first-order to LTL leads in fact to the pure future fragment of LTL, i.e., the fragment without any past tense operators. This leads to the separation theorem for first-order formulae in one free variable as we shall demonstrate in Section 9. We also show that aperiodicity (i.e., first-order definability) of a regular ∞-language can be decided in polynomial space, if the language is specified by some Büchi automaton.

Although the paper became much longer than expected, we know that much more could be said. We apologize if the reader's favorite theorem is not covered in our survey. In particular, we do not speak about varieties, and we gave up the project to cover principle results about the fragment of first-order logic which corresponds to unary temporal logic. These diamonds will continue to shine, but not here, and we refer to [30] for more background. As mentioned above, we use Büchi automata, but we do not discuss deterministic models such as deterministic Muller automata.

The history of Theorem 1.1 is related to some of the most influential scientists in computer science. The general scheme is that the equivalences above have been proved first for finite words. After that, techniques were developed to generalize these results to infinite words. Each time, the generalization to infinite words has been non-trivial and asked for new ideas. Perhaps, the underlying reason for this additional difficulty is due to the fact that the subset construction fails for infinite words. Other people may say that the difficulty arises from the fact that regular ω-languages are not closed in the Cantor topology. The truth is that combinatorics on infinite objects is more complicated.

The equivalence of first-order definability and star-freeness for finite words is due to McNaughton and Papert [19]. The generalization to infinite words is due to Ladner [15] and Thomas [31, 32]. These results have been refined, e.g. by Perrin and Pin in [24]. Based on the logical framework of Ehrenfeucht-Fraïssé-games, Thomas also related the quantifier depth to the so-called dot-depth hierarchy, [33, 35]. Taking not only the quantifier alternation into account, but also the length of quantifier blocks one gets even finer results as studied by Blanchet-Sadri in [2].

The equivalence of star-freeness and aperiodicity for finite words is due to Schützenberger [28]. The generalization to infinite words is due to Perrin [23] using the syntactic congruence of Arnold [1]. These results are the basis allowing to decide whether a regular language is first-order definable.

Putting these results together one sees that statements 1, 2, and 3 in Theorem 1.1 are equivalent. From the definition of LTL it is clear that linear temporal logic describes a fragment of FO^3, where the latter means the family of first-order definable languages where the defining sentence uses at most three names for variables. Thus, the implications from 4 to 5 and from 5 to 1 are trivial. The highly non-trivial step is to conclude from 1 (or 2 or 3) to 4. This is usually called Kamp's Theorem and is due to Kamp [13] and Gabbay, Pnueli, Shelah, and Stavi [9].

In this survey we follow the algebraic proof of Wilke which is in his habilitation thesis [38] and which is also published in [39]. Wilke gave the proof for finite words, only. In order to generalize it to infinite words we use the techniques from [5], which were developed to handle Mazurkiewicz traces. Cutting down this proof to the special case of finite or infinite words leads to the proof presented here. It is still the most complicated part in the paper, but again some of the technical difficulties lie in the combinatorics of infinite words which is subtle. Restricting the proof further to finite words, the reader might hopefully find the simplest way to pass from aperiodic languages to LTL. But this is also a matter of taste, of course.

Every first-order formula sentence can be translated to a formula in FO^3. This is sharp, because it is known that there are first-order properties which are not expressible in FO^2, which characterizes unary temporal logic [7] over infinite words.

The equivalence between definability in monadic second order logic, regular languages, and acceptance by Büchi automata is due to Büchi [3]. However, Büchi automata are inherently non-deterministic. In order to have deterministic automata one has to move to other acceptance conditions such as Muller or Rabin-Streett conditions. This important result is due to McNaughton, see [18]. Based on this, Thomas [32] extended the notion of deterministic counter-free automaton to deterministic counter-free automaton with Rabin-Streett condition and obtained thereby another characterization for first-order definable ω-languages. There is no canonical object for a minimal Büchi automaton, which might explain why a notion of counter-free Büchi automaton has not been introduced so far. On the other hand, there is a quite natural notion of counter-freeness as well as of aperiodicity for non-deterministic Büchi automata. (Aperiodic non-deterministic *finite* automata are defined in [16], too.) For non-deterministic automata, aperiodicity describes a larger class of automata, but both counter-freeness and aperiodicity can be used to characterize first-order definable ω-languages. This is shown in Section 11 and seems to be an original part in the paper.

We have also added a section about very weak alternating automata. The notion of *weak alternating automaton* is due to Muller, Saoudi, and Schupp [21]. A *very weak alternating automaton* is a special kind of weak alternating automaton and this notion has been introduced in the PhD thesis of Rhode [27] in a more general context of ordinals. (In the paper by Löding and Thomas [17] these automata are called *linear alternating*.) Section 13 shows that very weak alternating automata characterize first-order definability as well. More precisely, we have a cycle from 3 to 6 to 7 and back to 3, and we establish a bridge from 4 to 8 and from 8 to 7.

It was shown by Stern [29] that deciding whether a deterministic finite automaton accepts an aperiodic language over finite words can be done in polynomial space, i.e., in PSPACE. Later Cho and Huynh showed in [4] that this problem is actually PSPACE-complete. So, the PSPACE-hardness transfers to (non-deterministic) Büchi automata. It might belong to folklore that the PSPACE-upper bound holds for Büchi automata, too; but we did not find any reference. So we prove this result here, see Proposition 12.3.

As said above, our intention was to give simple proofs for existing results. But simplicity is not a simple notion. Therefore for some results, we present two proofs. The proofs are either based on a congruence lemma established for first-order logic in Section 10.1, or they are based on a splitting lemma established for star-free languages in Section 3.1. Depending on his background, the reader may wish to skip one approach.

2 Words, first-order logic, and basic notations

By P we denote a unary predicate taken from some finite set of *atomic propositions*, and $x, y, \ldots$ denote variables which represent positions in finite or infinite words. The syntax of first-order logic uses the symbol $\perp$ for *false* and has atomic formulae of type $P(x)$ and $x < y$. We allow Boolean connectives and first-order quantification. Thus, if φ and ψ are first-order formulae, then $\neg\varphi$, $\varphi \vee \psi$ and $\exists x \varphi$ are first-order formulae, too. As usual we have derived formulae such as $x \leq y$, $x = y$, $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$, $\forall x \varphi = \neg\exists x \neg\varphi$ and so on.

We let Σ be a finite *alphabet*. The relation between Σ and the set of unary predicates is that for each letter $a \in \Sigma$ and each predicate P the truth-value $P(a)$ must be well-defined. So, we always assume this. Whenever convenient we include for each letter a a predicate P_a such that $P_a(b)$ is *true* if and only if $a = b$. We could assume that all predicates are of the form P_a, but we feel more flexible of not making this assumption. If x is a position in a word with label $a \in \Sigma$, then $P(x)$ is defined by $P(a)$.

By Σ^* (resp. Σ^ω) we mean the set of finite (resp. infinite) words over Σ, and we let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. The length of a word w is denoted by $|w|$, it is a natural number or ω. A *language* is a set of finite or infinite words.

Formulae without free variables are *sentences*. A first-order sentence defines a subset of Σ^∞ in a natural way. Let us consider a few examples. We can specify that the first position is labeled by a letter a using $\exists x \forall y \; P_a(x) \wedge x \leq y$. We can say that each occurrence of a is immediately followed by b with the sentence $\forall x \; \neg P_a(x) \vee \exists y \; x < y \wedge P_b(y) \wedge \forall z \; \neg(x < z \wedge z < y)$. We can also say that the direct successor of each b is the letter a. Hence the language $(ab)^\omega$ is first-order definable. We can also say that a last position in a word exists and this position is labeled b. For $a \neq b$ this leads almost directly to a definition of $(ab)^*$. But $(aa)^*$ cannot be defined with a first-order sentence. A formal proof for this statement is postponed, but at least it should be clear that we cannot define $(aa)^*$ the same way as we did for $(ab)^*$, because we have no control that the length of a word in a^* is even.

The set of positions $\mathrm{pos}(w)$ is defined by $\mathrm{pos}(w) = \{i \in \mathbb{N} \mid 0 \leq i < |w|\}$. We think of $\mathrm{pos}(w)$ as a linear order where each position i is labeled with $\lambda(i) \in \Sigma$, and $w = \lambda(0)\lambda(1)\cdots$.

A *k-structure* means here a pair $(w, \overline{p})$, where $w \in \Sigma^\infty$ is a finite or infinite word and $\overline{p} = (p_1, \ldots, p_k)$ is a k-tuple of positions in $\mathrm{pos}(w)$. The set of all k-structures is denoted by $\Sigma^\infty_{(k)}$, and the subset of finite structures is denoted by $\Sigma^*_{(k)}$. For simplicity we identify Σ^∞ with $\Sigma^\infty_{(0)}$.

Let $\overline{x}$ be a k-tuple $(x_1, \ldots, x_k)$ of variables and φ be a first-oder formula where all free variables are in the set $\{x_1, \ldots, x_k\}$. The semantics of

$$(w, (p_1, \ldots, p_k)) \models \varphi$$

is defined as usual: It is enough to give a semantics to atomic formulae, and $(w, (p_1, \ldots, p_k)) \models P(x_i)$ means that the label of position p_i satisfies P, and $(w, (p_1, \ldots, p_k)) \models x_i < x_j$ means that position p_i is before position p_j, i.e., $p_i < p_j$.

With every formula we can associate its language by

$$\mathcal{L}(\varphi) = \left\{ (w, \overline{p}) \in \Sigma^{\infty}_{(k)} \;\middle|\; (w, \overline{p}) \models \varphi \right\}.$$

In order to be precise we should write $\mathcal{L}_{\Sigma,k}(\varphi)$, but if the context is clear, we omit the subscript Σ, k.

Definition 2.1. By $\mathrm{FO}(\Sigma^*)$ (resp. $\mathrm{FO}(\Sigma^\infty)$) we denote the set of first-order definable languages in Σ^* (resp. Σ^∞), and by FO we denote the family of all first-order definable languages. Analogously, we define families $\mathrm{FO}^n(\Sigma^*)$, $\mathrm{FO}^n(\Sigma^\infty)$, and FO^n by allowing only those formulae which use at most n different names for variables.

3 Star-free sets

For languages $K, L \subseteq \Sigma^\infty$ we define the *concatenation* by

$$K \cdot L = \{uv \mid u \in K \cap \Sigma^*, v \in L\}.$$

The n-th power of L is defined inductively by $L^0 = \{\varepsilon\}$ and $L^{n+1} = L \cdot L^n$. The *Kleene-star* of L is defined by $L^* = \bigcup_{n \geq 0} L^n$. Finally, the ω-iteration of L is

$$L^\omega = \{u_0 u_1 u_2 \cdots \mid u_i \in L \cap \Sigma^* \text{ for all } i \geq 0\}.$$

We are interested here in families of *regular languages*, also called *rational languages*. In terms of expressions it is the smallest family of languages which contains all finite subsets, which is closed under finite union and concatenation, and which is closed under the *Kleene-star* (and ω-power). The relation to finite automata (Büchi automata resp.) is treated in Section 11. For the main results on first-order languages the notion of a Büchi automaton is actually not needed.

The Kleene-star and the ω-power do not preserve first-order definability, hence we consider subclasses of regular languages. A language is called *star-free*, if we do not allow the Kleene-star, but we allow complementation. Therefore we have all Boolean operations. In terms of expressions the *class of star-free languages* is the smallest family of languages in Σ^∞ (resp. Σ^*)

which contains Σ^*, all singletons $\{a\}$ for $a \in \Sigma$, and which is closed under finite union, complementation and concatenation. It is well-known that regular languages are closed under complement[1], hence star-free languages are regular.

As a first example we note that for every $A \subseteq \Sigma$ the set A^* (of finite words containing only letters from A) is also star-free. We have:

$$A^* = \Sigma^* \setminus (\Sigma^*(\Sigma \setminus A)\Sigma^\infty).$$

In particular, $\{\varepsilon\} = \varnothing^*$ is star-free. Some other expressions *with star* are also in fact star-free. For example, for $a \neq b$ we obtain:

$$(ab)^* = (a\Sigma^* \cap \Sigma^* b) \setminus \Sigma^*(\Sigma^2 \setminus \{ab, ba\})\Sigma^*.$$

The above equality does not hold, if $a = b$. Actually, $(aa)^*$ is not star-free. The probably best way to see that $(aa)^*$ is not star-free, is to show (by structural induction) that for all star-free languages L there is a constant $n \in \mathbb{N}$ such that for all words x we have $x^n \in L$ if and only if $x^{n+1} \in L$. The property is essentially *aperiodicity* and we shall prove the equivalence between star-free sets and aperiodic languages later. Since $(ab)^*$ is star-free (for $a \neq b$), but $(aa)^*$ is not, we see that a *projection* of a star-free set is not star-free, in general.

Definition 3.1. By $\mathbf{SF}(\Sigma^*)$ (resp. $\mathbf{SF}(\Sigma^\infty)$) we denote the set of star-free languages in Σ^* (resp. Σ^∞), and by $\mathbf{SF}$ we denote the family of all star-free languages.

An easy exercise (left to the interested reader) shows that

$$\mathbf{SF}(\Sigma^*) = \{L \subseteq \Sigma^* \mid L \in \mathbf{SF}(\Sigma^\infty)\} = \{L \cap \Sigma^* \mid L \in \mathbf{SF}(\Sigma^\infty)\}.$$

3.1 The splitting lemma

A star-free set admits a canonical decomposition given a partition of the alphabet. This will be shown here and it is used to prove that first-order languages are star-free in Section 4 and for the separation theorem in Section 9. The alternative to this section is explained in Section 10, where the standard way of using the *congruence lemma* is explained, see Lemma 10.2. Thus, there is an option to skip this section.

Lemma 3.2. Let $A, B \subseteq \Sigma$ be disjoint subalphabets. If $L \in \mathbf{SF}(\Sigma^\infty)$ then we can write

$$L \cap B^*AB^\infty = \bigcup_{1 \leq i \leq n} K_i a_i L_i$$

where $a_i \in A$, $K_i \in \mathbf{SF}(B^*)$ and $L_i \in \mathbf{SF}(B^\infty)$ for all $1 \leq i \leq n$.

[1] We do not need this standard result here.

Proof. Since $B^*AB^\infty = \bigcup_{a \in A} B^*aB^\infty$, it is enough to show the result when $A = \{a\}$. The proof is by induction on the star-free expression and also on the alphabet size. (Note that $|B| < |\Sigma|$.). The result holds for the basic star-free sets:

- If $L = \{a\}$ with $a \in A$ then $L \cap B^*AB^\infty = \{\varepsilon\}a\{\varepsilon\}$.
- If $L = \{a\}$ with $a \notin A$ then $L \cap B^*AB^\infty = \emptyset a \emptyset$ (or we let $n = 0$).
- If $L = \Sigma^*$ then $L \cap B^*AB^\infty = B^*AB^*$.

The inductive step is clear for union. For concatenation, the result follows from

$$(L \cdot L') \cap B^*AB^\infty = (L \cap B^*AB^\infty) \cdot (L' \cap B^\infty) \cup (L \cap B^*) \cdot (L' \cap B^*AB^\infty).$$

It remains to deal with the complement $\Sigma^\infty \setminus L$ of a star-free set. By induction, we have $L \cap B^*aB^\infty = \bigcup_{1 \le i \le n} K_i a L_i$. If some K_i and K_j are not disjoint (for $i \neq j$), then we can rewrite

$$K_i a L_i \cup K_j a L_j = (K_i \setminus K_j) a L_i \cup (K_j \setminus K_i) a L_j \cup (K_i \cap K_j) a (L_i \cup L_j).$$

We can also add $(B^* \setminus \bigcup_i K_i) a \emptyset$ in case $\bigcup_i K_i$ is strictly contained in B^*. Therefore, we may assume that $\{K_i \mid 1 \le i \le n\}$ forms a partition of B^*. This yields:

$$(\Sigma^\infty \setminus L) \cap B^*aB^\infty = \bigcup_{1 \le i \le n} K_i a (B^\infty \setminus L_i).$$

Q.E.D.

4 From first-order to star-free languages

This section shows that first-order definable languages are star-free languages. The transformation is involved in the sense that the resulting expressions are much larger than the size of the formula, in general. The proof presented here is based on the splitting lemma. The alternative is again in Section 10.

Remark 4.1. The converse that star-free languages are first-order definable can be proved directly. Although strictly speaking we do not use this fact, we give an indication how it works. It is enough to give a sentence for languages of type $L = \mathcal{L}(\varphi) \cdot a \cdot \mathcal{L}(\psi)$. We may assume that the sentences φ and ψ use different variable names. Then we can describe L as a language $\mathcal{L}(\xi)$ where

$$\xi = \exists z\, P_a(z) \wedge \varphi_{<z} \wedge \psi_{>z},$$

where $\varphi_{<z}$ and $\psi_{>z}$ relativize all variables with respect to the position of z. We do not go into more details, because, as said above, we do not need this fact.

We have to deal with formulae having free variables. We provide first another semantics of a formula with free variables in a set of words over an extended alphabet allowing to encode the assignment. This will also be useful to derive the separation theorem in Section 9.

Let V be a finite set of variables. We define $\Sigma_V = \Sigma \times \{0,1\}^V$. (Do not confuse Σ_V with $\Sigma_{(k)}$ from above.) Let $w \in \Sigma^\infty$ be a word and σ be an assignment from the variables in V to the positions in w, thus $0 \leq \sigma(x) < |w|$ for all $x \in V$. The pair (w,σ) can be encoded as a word $\overline{(w,\sigma)}$ over Σ_V. More precisely, if $w = a_0a_1a_2\cdots$ then $\overline{(w,\sigma)} = (a_0,\tau_0)(a_1,\tau_1)(a_2,\tau_2)\cdots$ where for all $0 \leq i < |w|$ we have $\tau_i(x) = 1$ if and only if $\sigma(x) = i$. We let $\mathcal{N}_V \subseteq \Sigma_V^\infty$ be the set of words $\overline{(w,\sigma)}$ such that $w \in \Sigma^\infty$ and σ is an assignment from V to the positions in w. We show that $\mathcal{N}_V$ is star-free. For $x \in V$, let $\Sigma_V^{x=1}$ be the set of pairs (a,τ) with $\tau(x) = 1$ and let $\Sigma_V^{x=0} = \Sigma_V \setminus \Sigma_V^{x=1}$ be its complement. Then,

$$\mathcal{N}_V = \bigcap_{x\in V} (\Sigma_V^{x=0})^*\Sigma_V^{x=1}(\Sigma_V^{x=0})^\infty.$$

Given a first-order formula φ and a set V containing all free variables of φ, we define the semantics $[\![\varphi]\!]_V \subseteq \mathcal{N}_V$ inductively:

$$\begin{aligned}
[\![P_a(x)]\!]_V &= \{\overline{(w,\sigma)} \in \mathcal{N}_V \mid w = b_0b_1b_2\cdots \in \Sigma^\infty \text{ and } b_{\sigma(x)} = a\}\\
[\![x < y]\!]_V &= \{\overline{(w,\sigma)} \in \mathcal{N}_V \mid \sigma(x) < \sigma(y)\}\\
[\![\exists x, \varphi]\!]_V &= \{\overline{(w,\sigma)} \in \mathcal{N}_V \mid \exists i, 0 \leq i < |w| \wedge \overline{(w,\sigma[x \to i])} \in [\![\varphi]\!]_{V\cup\{x\}}\}\\
[\![\varphi \vee \psi]\!]_V &= [\![\varphi]\!]_V \cup [\![\psi]\!]_V\\
[\![\neg\varphi]\!]_V &= \mathcal{N}_V \setminus [\![\varphi]\!]_V.
\end{aligned}$$

Proposition 4.2. Let φ be a first-order formula and V be a set of variables containing the free variables of φ. Then, $[\![\varphi]\!]_V \in \mathbf{SF}(\Sigma_V^\infty)$.

Proof. The proof is by induction on the formula. We have

$$\begin{aligned}
[\![P_a(x)]\!]_V &= \mathcal{N}_V \cap (\Sigma_V^* \cdot \{(a,\tau) \mid \tau(x) = 1\} \cdot \Sigma_V^\infty)\\
[\![x < y]\!]_V &= \mathcal{N}_V \cap (\Sigma_V^* \cdot \Sigma_V^{x=1} \cdot \Sigma_V^* \cdot \Sigma_V^{y=1} \cdot \Sigma_V^\infty).
\end{aligned}$$

The induction is trivial for disjunction and negation since the star-free sets form a Boolean algebra and $\mathcal{N}_V$ is star-free. The interesting case is existential quantification $[\![\exists x, \varphi]\!]_V$.

We assume first that $x \notin V$ and we let $V' = V \cup \{x\}$. By induction, $[\![\varphi]\!]_{V'}$ is star-free and we can apply Lemma 3.2 with the sets $A = \Sigma_{V'}^{x=1}$ and $B = \Sigma_{V'}^{x=0}$. Note that $\mathcal{N}_{V'} \subseteq B^*AB^\infty$. Hence, $[\![\varphi]\!]_{V'} = [\![\varphi]\!]_{V'} \cap B^*AB^\infty$ and we obtain $[\![\varphi]\!]_{V'} = \bigcup_{1\leq i\leq n} K_i'a_i'L_i'$ where $a_i' \in A$, $K_i' \in \mathbf{SF}(B^*)$ and $L_i' \in \mathbf{SF}(B^\infty)$ for all i. Let $\pi : B^\infty \to \Sigma_V^\infty$ be the *bijective renaming* defined by $\pi(a,\tau) = (a, \tau_{\restriction V})$. Star-free sets are not preserved by projections but indeed they are preserved by bijective renamings. Hence, $K_i = \pi(K_i') \in \mathbf{SF}(\Sigma_V^*)$ and $L_i = \pi(L_i') \in \mathbf{SF}(\Sigma_V^\infty)$. We also rename $a_i' = (a,\tau)$ into $a_i = (a, \tau_{\restriction V})$. We have $[\![\exists x, \varphi]\!]_V = \bigcup_{1 \le i \le n} K_i a_i L_i$ and we deduce that $[\![\exists x, \varphi]\!]_V \in \mathbf{SF}(\Sigma_V^\infty)$.

Finally, if $x \in V$ then we choose a new variable $y \notin V$ and we let $U = (V \setminus \{x\}) \cup \{y\}$. From the previous case, we get $[\![\exists x, \varphi]\!]_U \in \mathbf{SF}(\Sigma_U^\infty)$. To conclude, it remains to rename y to x. Q.E.D.

Corollary 4.3. We have:

$$\mathrm{FO}(\Sigma^*) \subseteq \mathbf{SF}(\Sigma^*) \text{ and } \mathrm{FO}(\Sigma^\infty) \subseteq \mathbf{SF}(\Sigma^\infty).$$

5 Aperiodic languages

Recall that a monoid $(M, \cdot)$ is a non-empty set M together with a binary operation $\cdot$ such that $((x \cdot y) \cdot z) = (x \cdot (y \cdot z))$ and with a neutral element $1 \in M$ such that $x \cdot 1 = 1 \cdot x = x$ for all x, y, z in M. Frequently we write xy instead of $x \cdot y$.

A *morphism* (or *homomorphism*) between monoids M and M' is a mapping $h : M \to M'$ such that $h(1) = 1$ and $h(x \cdot y) = h(x) \cdot h(y)$.

We use the algebraic notion of *recognizability* and the notion of *aperiodic* languages. Recognizability is defined as follows. Let $h : \Sigma^* \to M$ be a morphism to a finite monoid M. Two words $u, v \in \Sigma^\infty$ are said to be h-similar, denoted by $u \sim_h v$, if for some $n \in \mathbb{N} \cup \{\omega\}$ we can write $u = \prod_{0 \le i < n} u_i$ and $v = \prod_{0 \le i < n} v_i$ with $u_i,\ v_i \in \Sigma^+$ and $h(u_i) = h(v_i)$ for all $0 \le i < n$. The notation $u = \prod_{0 \le i < n} u_i$ refers to an *ordered product*, it means a factorization $u = u_0 u_1 \cdots$. In other words, $u \sim_h v$ if either $u = v = \varepsilon$, or $u, v \in \Sigma^+$ and $h(u) = h(v)$ or $u, v \in \Sigma^\omega$ and there are factorizations $u = u_0 u_1 \cdots$, $v = v_0 v_1 \cdots$ with $u_i, v_i \in \Sigma^+$ and $h(u_i) = h(v_i)$ for all $i \ge 0$.

The transitive closure of $\sim_h$ is denoted by $\approx_h$; it is an equivalence relation. For $w \in \Sigma^\infty$, we denote by $[w]_h$ the equivalence class of w under $\approx_h$. Thus,

$$[w]_h = \{u \mid u \approx_h w\}\,.$$

In case that there is no ambiguity, we simply write $[w]$ instead of $[w]_h$. Note that there are three cases $[w] = \{\varepsilon\}$, $[w] \subseteq \Sigma^+$, and $[w] \subseteq \Sigma^\omega$.

Definition 5.1. We say that a morphism $h : \Sigma^* \to M$ *recognizes* L, if $w \in L$ implies $[w]_h \subseteq L$ for all $w \in \Sigma^\infty$.

Thus, a language $L \subseteq \Sigma^\infty$ is *recognized* by h if and only if L is saturated by $\approx_h$ (or equivalently by $\sim_h$). Note that we may assume that a recognizing morphism $h : \Sigma^* \to M$ is surjective, whenever convenient.

Since M is finite, the equivalence relation $\approx_h$ is of finite index. More precisely, there are at most $1+|M|+|M|^2$ classes. This fact can be derived by some standard Ramsey argument about infinite monochromatic subgraphs. We repeat the argument below in order to keep the article self-contained, see also [3, 12, 25]. It shows the existence of a so-called *Ramsey factorization.*

Lemma 5.2. Let $h : \Sigma^* \to M$ be a morphism to a finite monoid M and $w = u_0u_1u_2\cdots$ be an infinite word with $u_i \in \Sigma^+$ for $i \geq 0$. Then there exist $s, e \in M$, and an increasing sequence $0 < p_1 < p_2 < \cdots$ such that the following two properties hold:

1. $se = s$ and $e^2 = e$.

2. $h(u_0 \cdots u_{p_1-1}) = s$ and $h(u_{p_i} \cdots u_{p_j-1}) = e$ for all $0 < i < j$.

Proof. Let $E = \{(i,j) \in \mathbb{N}^2 \mid i < j\}$. We consider the mapping $c : E \to M$ defined by $c(i,j) = h(u_i \cdots u_{j-1})$. We may think that the pairs (i,j) are (edges of an infinite complete graph and) *colored* by $c(i,j)$. Next we wish to color an infinite set of positions.

We define inductively a sequence of infinite sets $\mathbb{N} = N_0 \supset N_1 \supset N_2 \cdots$ and a sequence of natural numbers $n_0 < n_1 < n_2 < \cdots$ as follows. Assume that N_p is already defined and infinite. (This is true for $p = 0$.) Choose any $n_p \in N_p$, e.g., $n_p = \min N_p$. Since M is finite and N_p is infinite, there exists $c_p \in M$ and an infinite subset $N_{p+1} \subset N_p$ such that $c(n_p, m) = c_p$ for all $m \in N_{p+1}$. Thus, for all $p \in \mathbb{N}$ infinite sets N_p are defined and for every position n_p we may choose the color c_p. Again, because M is finite, one color must appear infinitely often. This color is called e and it is just the (idempotent) element of M we are looking for. Therefore we find a strictly increasing sequence $p_0 < p_1 < p_2 < \cdots$ such that $c_{p_i} = e$ and hence $e = h(u_{p_i} \cdots u_{p_j-1})$ for all $0 \leq i < j$. Note that $e = c(n_{p_0}, n_{p_2}) = c(n_{p_0}, n_{p_1})c(n_{p_1}, n_{p_2}) = e^2$. Moreover, if we set $s = h(u_0 \cdots u_{p_1-1})$, we obtain

$$s = c(0, n_{p_1}) = c(0, n_{p_0})c(n_{p_0}, n_{p_1}) = c(0, n_{p_0})c(n_{p_0}, n_{p_1})c(n_{p_1}, n_{p_2}) = se.$$

This is all we need. Q.E.D.

The lemma implies that for each (infinite) word w we may choose some $(s, e) \in M \times M$ with $s = se$ and $e = e^2$ such that $w \in h^{-1}(s)\,(h^{-1}(e))^\omega$. This establishes that $\approx_h$ has at most $|M|^2$ classes $[w]$ where w is infinite; and this in turn implies the given bound $1 + |M| + |M|^2$.

Pairs $(s, e) \in M \times M$ with $s = se$ and $e = e^2$ are also called *linked pair.*

Remark 5.3. The existence of a Ramsey factorization implies that a language $L \subseteq \Sigma^\omega$ recognized by a morphism h from Σ^* to some finite monoid M

can be written as a finite union of languages of type UV^ω, where $U, V \subseteq \Sigma^*$ are recognized by h and where moreover $U = h^{-1}(s)$ and $V = h^{-1}(e)$ for some $s, e \in M$ with $se = s$ and $e^2 = e$. In particular, we have $UV \subseteq U$ and $VV \subseteq V$. Since $\{\varepsilon\}^\omega = \{\varepsilon\}$, the statement holds for $L \subseteq \Sigma^*$ and $L \subseteq \Sigma^\infty$ as well.

A (finite) monoid M is called *aperiodic*, if for all $x \in M$ there is some $n \in \mathbb{N}$ such that $x^n = x^{n+1}$.

Definition 5.4. A language $L \subseteq \Sigma^\infty$ is called *aperiodic*, if it is recognized by some morphism to a finite and aperiodic monoid. By $\mathbf{AP}(\Sigma^*)$ (resp. $\mathbf{AP}(\Sigma^\infty)$) we denote the set of aperiodic languages in Σ^* (resp. Σ^∞), and by $\mathbf{AP}$ we denote the family of aperiodic languages.

6 From star-freeness to aperiodicity

Corollary 4.3 (as well as Proposition 10.3) tells us that all first-order definable languages are star-free. We want to show that all star-free languages are recognized by aperiodic monoids. Note that the trivial monoid recognizes the language Σ^*, actually it recognizes all eight Boolean combinations of $\{\varepsilon\}$ and Σ^ω.

Consider next a letter a. The smallest recognizing monoid of the singleton $\{a\}$ is aperiodic, it has just three elements $1, a, 0$ with $a \cdot a = 0$ and 0 is a *zero*, this means $x \cdot y = 0$ as soon as $0 \in \{x, y\}$.

Another very simple observation is that if L_i is recognized by a morphism $h_i : \Sigma^* \to M_i$ to some finite (aperiodic) monoid M_i, $i = 1, 2$, then (the direct product $M_1 \times M_2$ is aperiodic and) the morphism

$$h : \Sigma^* \to M_1 \times M_2,\ w \mapsto (h_1(w), h_2(w))$$

recognizes all Boolean combinations of L_1 and L_2.

The proof of the next lemma is rather technical. Its main part shows that the family of recognizable languages is closed under concatenation. Aperiodicity comes into the picture only at the very end in a few lines. There is alternative way to prove the following lemma. In Section 11 we introduce non-deterministic counter-free Büchi automata which can be used to show the closure under concatenation as well, see Lemma 11.3.

Lemma 6.1. Let $L \subseteq \Sigma^*$ and $K \subseteq \Sigma^\infty$ be aperiodic languages. Then $L \cdot K$ is aperiodic.

Proof. As said above, we may choose a single morphism $h : \Sigma^* \to M$ to some finite aperiodic monoid M, which recognizes both L and K.

The set of pairs $(h(u), h(v))$ with $u, v \in \Sigma^*$ is finite (bounded by $|M|^2$) and so its power set S is finite, too. We shall see that there is a monoid structure on some subset of S such that this monoid recognizes $L \cdot K$.

To begin with, let us associate with $w \in \Sigma^*$ the following set of pairs:

$$g(w) = \{(h(u), h(v)) \mid w = uv\}\,.$$

The finite set $g(\Sigma^*) \subseteq S$ is in our focus. We define a multiplication by:

$$\begin{aligned} g(w) \cdot g(w') &= g(ww') \\ &= \{(h(wu'), h(v')) \mid w' = u'v'\} \cup \{(h(u), h(vw')) \mid w = uv\}\,. \end{aligned}$$

The product is well-defined. To see this, observe first that $(h(u), h(v)) \in g(w)$ implies $h(w) = h(u)h(v)$ since h is a morphism. Thus, the set $g(w)$ *knows* the element $h(w)$. Second, $h(wu') = h(w)h(u')$ since h is a morphism. Hence, we can compute $\{(h(wu'), h(v')) \mid w' = u'v'\}$ from $g(w)$ and $g(w')$. The argument for the other component is symmetric.

By the very definition of g, we obtain a morphism

$$g : \Sigma^* \to g(\Sigma^*).$$

In order to see that g recognizes $L \cdot K$ consider $u \in L \cdot K$ and v such that we can write $u = \prod_{0 \le i < n} u_i$ and $v = \prod_{0 \le i < n} v_i$ with u_i, $v_i \in \Sigma^+$ and $g(u_i) = g(v_i)$ for all $0 \le i < n$. We have to show $v \in L \cdot K$. We have $u \in L \cdot K = (L \cap \Sigma^*) \cdot K$. Hence, for some index j we can write $u_j = u'_j u''_j$ with

$$\left(\prod_{0 \le i < j} u_i\right) u'_j \in L \quad \text{and} \quad u''_j \left(\prod_{j < i < n} u_i\right) \in K.$$

Now, $g(u_i) = g(v_i)$ implies $h(u_i) = h(v_i)$. Moreover, $u_j = u'_j u''_j$ implies $(h(u'_j), h(u''_j)) \in g(u_j) = g(v_j)$. Hence we can write $v_j = v'_j v''_j$ with $h(u'_j) = h(v'_j)$ and $h(u''_j) = h(v''_j)$. Therefore

$$\left(\prod_{0 \le i < j} v_i\right) v'_j \in L \quad \text{and} \quad v''_j \left(\prod_{j < i < n} v_i\right) \in K$$

and $v \in L \cdot K$, too.

It remains to show that the resulting monoid is indeed aperiodic. To see this choose some $n > 0$ such that $x^n = x^{n+1}$ for all $x \in M$. Consider any element $g(w) \in g(\Sigma^*)$. We show that $g(w)^{2n} = g(w)^{2n+1}$. This is straightforward:

$$g(w)^{2n} = g(w^{2n}) = \left\{(h(w^k u), h(vw^m)) \;\middle|\; w = uv,\, k + m = 2n - 1\right\}.$$

If $k + m = 2n - 1$ then either $k \ge n$ or $m \ge n$. Hence, for each pair, we have either $(h(w^k u), h(vw^m)) = (h(w^{k+1} u), h(vw^m))$ or $(h(w^k u), h(vw^m)) = (h(w^k u), h(vw^{m+1}))$. The result follows. Q.E.D.

Proposition 6.2. We have $\mathbf{SF} \subseteq \mathbf{AP}$ or more explicitly:

$$\mathbf{SF}(\Sigma^*) \subseteq \mathbf{AP}(\Sigma^*) \text{ and } \mathbf{SF}(\Sigma^\infty) \subseteq \mathbf{AP}(\Sigma^\infty).$$

Proof. Aperiodic languages form a Boolean algebra. We have seen above that $\mathbf{AP}$ contains Σ^* and all singletons $\{a\}$, where a is a letter. Thus, star-free languages are aperiodic by Lemma 6.1. Q.E.D.

7 From LTL to FO^3

The syntax of $\mathrm{LTL}_\Sigma[\mathsf{XU}, \mathsf{YS}]$ is given by

$$\varphi ::= \bot \mid a \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \,\mathsf{XU}\, \varphi \mid \varphi \,\mathsf{YS}\, \varphi,$$

where a ranges over Σ. When there is no ambiguity, we simply write LTL for $\mathrm{LTL}_\Sigma[\mathsf{XU}, \mathsf{YS}]$. We also write $\mathrm{LTL}_\Sigma[\mathsf{XU}]$ for the pure future fragment where only the *next-until* modality XU is allowed.

In order to give a semantics to an LTL formula we identify each $\varphi \in$ LTL with some first-order formula $\varphi(x)$ in at most one free variable. The identification is done by structural induction. $\top$ and $\bot$ still denote the truth value *true* and *false*, the formula a becomes $a(x) = P_a(x)$. The formulae *neXt-Until* and *Yesterday-Since* are defined by:

$$(\varphi \,\mathsf{XU}\, \psi)(x) = \exists z:\ x < z \wedge \psi(z) \wedge \forall y:\ x < y < z \rightarrow \varphi(y).$$
$$(\varphi \,\mathsf{YS}\, \psi)(x) = \exists z:\ x > z \wedge \psi(z) \wedge \forall y:\ x > y > z \rightarrow \varphi(y).$$

It is clear that each LTL formula becomes under this identification a first-order formula which needs at most three different names for variables. For simplicity let us denote this fragment by FO^3, too. Thus, we can write $\mathrm{LTL} \subseteq \mathrm{FO}^3$.

As usual, we may use derived formulas such as $\mathsf{X}\,\varphi = \bot \,\mathsf{XU}\, \varphi$ (read *neXt* φ), $\varphi \,\mathsf{U}\, \psi = \psi \vee (\varphi \wedge (\varphi \,\mathsf{XU}\, \psi))$ (read φ *Until* ψ), $\mathsf{F}\,\varphi = \top \,\mathsf{U}\, \varphi$ (read *Future* φ), etc.

Since $\mathrm{LTL} \subseteq \mathrm{FO}^3$ a model of an LTL_Σ formula φ is a word $v = a_0a_1a_2\cdots \in A^\infty \setminus \{\varepsilon\}$ together with a position $0 \le i < |v|$ (the alphabet A might be different from Σ).

For a formula $\varphi \in \mathrm{LTL}_\Sigma$ and an alphabet A, we let

$$\mathcal{L}_A(\varphi) = \{v \in A^\infty \setminus \{\varepsilon\} \mid v, 0 \models \varphi\}.$$

We say that a language $L \subseteq A^\infty$ is definable in LTL_Σ if $L \setminus \{\varepsilon\} = \mathcal{L}_A(\varphi)$ for some $\varphi \in \mathrm{LTL}_\Sigma$. Note that the empty word ε cannot be a model of a formula. To include the empty word, it will be convenient to consider for any letter c (not necessarily in A), the language

$$\mathcal{L}_{c,A}(\varphi) = \{v \in A^\infty \mid cv, 0 \models \varphi\}.$$

Remark 7.1. When we restrict to the pure future fragment $\mathrm{LTL}_\Sigma[\mathsf{XU}]$ the two approaches define almost the same class of languages. Indeed, for each formula $\varphi \in \mathrm{LTL}_\Sigma[\mathsf{XU}]$, we have $\mathcal{L}_A(\varphi) = \mathcal{L}_{c,A}(\mathsf{X}\,\varphi) \setminus \{\varepsilon\}$. Conversely, for each formula φ there is a formula $\overline{\varphi}$ such that $\mathcal{L}_A(\overline{\varphi}) = \mathcal{L}_{c,A}(\varphi) \setminus \{\varepsilon\}$. The translation is simply $\overline{\varphi \,\mathsf{XU}\, \psi} = \varphi \,\mathsf{U}\, \psi$, $\overline{c} = \top$ and $\overline{a} = \bot$ if $a \neq c$, and as usual $\overline{\neg\varphi} = \neg\overline{\varphi}$ and $\overline{\varphi \vee \psi} = \overline{\varphi} \vee \overline{\psi}$.

8 From AP to LTL

8.1 A construction on monoids

The passage from **AP** to LTL is perhaps the most difficult step in completing the picture of first-order definable languages. We shall use an induction on the size of the monoid M, for this we recall first a construction due to [5].

For a moment let M be any monoid and $m \in M$ an element. Then $mM \cap Mm$ is obviously a subsemigroup, but it may not have a neutral element. Hence it is not a monoid, in general. Note that, if $m \neq 1_M$ and M is aperiodic, then $1_M \notin mM \cap Mm$. Indeed, assume that $1_M \in mM$ and write $1_M = mx$ with $x \in M$. Hence $1_M = m^n x^n$ for all n, and for some $n \geq 0$ we have $m^n = m^{n+1}$. Taking this n we see:

$$1_M = m^n x^n = m^{n+1} x^n = m(m^n x^n) = m 1_M = m.$$

Therefore $|mM \cap Mm| < |M|$, if M is aperiodic and if $m \neq 1_M$.

It is possible to define a new product $\circ$ such that $mM \cap Mm$ becomes a monoid where m is a neutral element: We let

$$xm \circ my = xmy$$

for $xm, my \in mM \cap Mm$. This is well-defined since $xm = x'm$ and $my = my'$ imply $xmy = x'my'$. The operation is associative and $m \circ z = z \circ m = z$. Hence $(mM \cap Mm, \circ, m)$ is indeed a monoid. Actually it is a divisor of M. To see this consider the submonoid $N = \{x \in M \mid xm \in mM\}$. (Note that N is indeed a submonoid of M.) Clearly, the mapping $x \mapsto xm$ yields a surjective morphism from $(N, \cdot, 1_M)$ onto $(mM \cap Mm, \circ, m)$, which is therefore a homomorphic image of the submonoid N of M. In particular, if M is aperiodic, then $(mM \cap Mm, \circ, m)$ is aperiodic, too. The construction is very similar to a construction of what is known as *local algebra*, see [8, 20]. Therefore we call $(mM \cap Mm, \circ, m)$ the *local divisor* of M at the element m.

8.2 Closing the cycle

Proposition 8.1. We have $\mathbf{AP} \subseteq \mathbf{LTL}$. More precisely, let $L \subseteq \Sigma^\infty$ be a language recognized by an aperiodic monoid M.

(1) We can construct a formula $\varphi \in \mathrm{LTL}_\Sigma[\mathsf{XU}]$ such that $L \setminus \{\varepsilon\} = \mathcal{L}_\Sigma(\varphi)$.

(2) For any letter c (not necessarily in Σ), we can construct a formula $\varphi \in \mathrm{LTL}_\Sigma[\mathsf{XU}]$ such that $L = \mathcal{L}_{c,\Sigma}(\varphi)$.

Proof. Note first that (1) follows from (2) by Remark 7.1. The proof of (2) is by induction on $(|M|, |\Sigma|)$ (with lexicographic ordering). Let $h : \Sigma^* \to M$ be a morphism to the aperiodic monoid M. The assertion of Proposition 8.1 is almost trivial if $h(c) = 1_M$ for all $c \in \Sigma$. Indeed, in this case, the set L is a Boolean combination of the sets $\{\varepsilon\}$, Σ^+ and Σ^ω which are easily definable in $\mathrm{LTL}_\Sigma[\mathsf{XU}]$: we have $\{\varepsilon\} = \mathcal{L}_{c,\Sigma}(\neg \mathsf{X} \top)$, $\Sigma^+ = \mathcal{L}_{c,\Sigma}(\mathsf{X}\,\mathsf{F} \neg \mathsf{X} \top)$ and $\Sigma^\omega = \mathcal{L}_{c,\Sigma}(\neg \mathsf{F} \neg \mathsf{X} \top)$. Note that when $|M| = 1$ or $|\Sigma| = 0$ then we have $h(c) = 1_M$ for all $c \in \Sigma$ and this special case ensures the base of the induction.

In the following, we fix a letter $c \in \Sigma$ such that $h(c) \neq 1_M$ and we let $A = \Sigma \setminus \{c\}$. We define the c-factorization of a word $v \in \Sigma^\infty$. If $v \in (A^*c)^\omega$ then its c-factorization is $v = v_0 c v_1 c v_2 c \cdots$ with $v_i \in A^*$ for all $i \geq 0$. If $v \in (A^*c)^* A^\infty$ then its c-factorization is $v = v_0 c v_1 c \cdots v_{k-1} c v_k$ where $k \geq 0$ and $v_i \in A^*$ for $0 \leq i < k$ and $v_k \in A^\infty$.

Consider two new disjoint alphabets $T_1 = \{h(u) \mid u \in A^*\}$ and $T_2 = \{[u]_h \mid u \in A^\infty\}$. Let $T = T_1 \uplus T_2$ and define the mapping $\sigma : \Sigma^\infty \to T^\infty$ by $\sigma(v) = h(v_0)h(v_1)h(v_2)\cdots \in T_1^\omega$ if $v \in (A^*c)^\omega$ and its c-factorization is $v = v_0 c v_1 c v_2 c \cdots$, and $\sigma(v) = h(v_0)h(v_1)\cdots h(v_{k-1})[v_k]_h \in T_1^* T_2$ if $v \in (A^*c)^* A^\infty$ and its c-factorization is $v = v_0 c v_1 c \cdots v_{k-1} c v_k$.

Lemma 8.2. Let $L \subseteq \Sigma^\infty$ be a language recognized by h. There exists a language $K \subseteq T^\infty$ which is definable in $\mathrm{LTL}_T[\mathsf{XU}]$ and such that $L = \sigma^{-1}(K)$.

Proof. We have seen that the local divisor $M' = h(c)M \cap Mh(c)$ is an aperiodic monoid with composition $\circ$ and neutral element $h(c)$. Moreover, $|M'| < |M|$ since $h(c) \neq 1_M$. Let us define a morphism $g : T^* \to M'$ as follows. For $m = h(u) \in T_1$ we define $g(m) = h(c)mh(c) = h(cuc)$. For $m \in T_2$ we let $g(m) = h(c)$, which is the neutral element in M'.

Let $K_0 = \{[u]_h \mid u \in L \cap A^\infty\} \subseteq T_2$. We claim that $L \cap A^\infty = \sigma^{-1}(K_0)$. One inclusion is clear. Conversely, let $v \in \sigma^{-1}(K_0)$. There exists $u \in L \cap A^\infty$ such that $\sigma(v) = [u]_h \in T_2$. By definition of σ, this implies $v \in A^\infty$ and $v \approx_h u$. Since $u \in L$ and L is recognized by h, we get $v \in L$ as desired.

For $n \in T_1$ and $m \in T_2$, let $K_{n,m} = nT_1^* m \cap n[n^{-1}\sigma(L) \cap T_1^* m]_g$ and let $K_1 = \bigcup_{n \in T_1, m \in T_2} K_{n,m}$. We claim that $L \cap (A^*c)^+ A^\infty = \sigma^{-1}(K_1)$. Let first $v \in L \cap (A^*c)^+ A^\infty$ and write $v = v_0 c v_1 \cdots c v_k$ its c-factorization. With $n = h(v_0)$ and $m = [v_k]_h$ we get $\sigma(v) \in K_{n,m}$. Conversely, let $v \in \sigma^{-1}(K_{n,m})$ with $n \in T_1$ and $m \in T_2$. We have $v \in (A^*c)^+ A^\infty$ and its c-factorization is $v = v_0 c v_1 \cdots c v_k$ with $k > 0$, $h(v_0) = n$ and $[v_k]_h = m$. Moreover, $x = h(v_1)\cdots h(v_{k-1})[v_k]_h \in [n^{-1}\sigma(L) \cap T_1^* m]_g$ hence we find $y \in T_1^* m$ with $g(x) = g(y)$ and $ny \in \sigma(L)$. Let $u \in L$ be such that $\sigma(u) = ny \in nT_1^*m$. Then $u \in (A^*c)^+A^\infty$ and its c-factorization is $u = u_0cu_1\cdots cu_\ell$ with $\ell > 0$, $h(u_0) = n$ and $[u_\ell]_h = m$. By definition of g, we get $h(cv_1c\cdots cv_{k-1}c) = g(x) = g(y) = h(cu_1c\cdots cu_{\ell-1}c)$. Using $h(v_0) = n = h(u_0)$ and $[v_k]_h = m = [u_\ell]_h$, we deduce that $v \approx_h u$. Since $u \in L$ and L is recognized by h, we get $v \in L$ as desired.

For $n \in T_1$, let $K_{n,\omega} = nT_1^\omega \cap n[n^{-1}\sigma(L)\cap T_1^\omega]_g$ and let $K_2 = \bigcup_{n\in T_1} K_{n,\omega}$. As above, we shall show that $L \cap (A^*c)^\omega = \sigma^{-1}(K_2)$. So let $v \in L \cap (A^*c)^\omega$ and consider its c-factorization $v = v_0cv_1cv_2\cdots$. With $n = h(v_0)$, we get $\sigma(v) \in K_{n,\omega}$. To prove the converse inclusion we need some auxiliary results.

First, if $x \sim_g y \sim_g z$ with $x \in T^\omega$ and $|y|_{T_1} < \omega$ then $x \sim_g z$. Indeed, in this case, we find factorizations $x = x_0x_1x_2\cdots$ and $y = y_0y_1y_2\cdots$ with $x_i \in T^+$, $y_0 \in T^+$ and $y_i \in T_2^+$ for $i > 0$ such that $g(x_i) = g(y_i)$ for all $i \geq 0$. Similarly, we find factorizations $z = z_0z_1z_2\cdots$ and $y = y'_0y'_1y'_2\cdots$ with $z_i \in T^+$, $y'_0 \in T^+$ and $y'_i \in T_2^+$ for $i > 0$ such that $g(z_i) = g(y'_i)$ for all $i \geq 0$. Then, we have $g(x_i) = g(y_i) = h(c) = g(y'_i) = g(z_i)$ for all $i > 0$ and $g(x_0) = g(y_0) = g(y'_0) = g(z_0)$ since y_0 and y'_0 contain all letters of y from T_1 and g maps all letters from T_2 to the neutral element of M'.

Second, if $x \sim_g y \sim_g z$ with $|y|_{T_1} = \omega$ then $x \sim_g y' \sim_g z$ for some $y' \in T_1^\omega$. Indeed, in this case, we find factorizations $x = x_0x_1x_2\cdots$ and $y = y_0y_1y_2\cdots$ with $x_i \in T^+$, and $y_i \in T^*T_1T^*$ such that $g(x_i) = g(y_i)$ for all $i \geq 0$. Let y'_i be the projection of y_i to the subalphabet T_1 and let $y' = y'_0y'_1y'_2\cdots \in T_1^\omega$. We have $g(y_i) = g(y'_i)$, hence $x \sim_g y'$. Similarly, we get $y' \sim_g z$.

Third, if $\sigma(u) \sim_g \sigma(v)$ with $u, v \in (A^*c)^\omega$ then $cu \approx_h cv$. Indeed, since $u, v \in (A^*c)^\omega$, the c-factorizations of u and v are of the form $u_1cu_2c\cdots$ and $v_1cv_2c\cdots$ with $u_i, v_i \in A^*$. Using $\sigma(u) \sim_g \sigma(v)$, we find new factorizations $u = u'_1cu'_2c\cdots$ and $v = v'_1cv'_2c\cdots$ with $u'_i, v'_i \in (A^*c)^*A^*$ and $h(cu'_ic) = h(cv'_ic)$ for all $i > 0$. We deduce

$$\begin{aligned} cu = (cu'_1c)u'_2(cu'_3c)u'_4\cdots \sim_h (cv'_1c)u'_2(cv'_3c)u'_4\cdots &= cv'_1(cu'_2c)v'_3(cu'_4c)\cdots \\ &\sim_h cv'_1(cv'_2c)v'_3(cv'_4c)\cdots = cv. \end{aligned}$$

We come back to the proof of $\sigma^{-1}(K_{n,\omega}) \subseteq L \cap (A^*c)^\omega$. So let $u \in \sigma^{-1}(K_{n,\omega})$. We have $u \in (A^*c)^\omega$ and $\sigma(u) = nx \in nT_1^\omega$ with $x \in [n^{-1}\sigma(L)\cap T_1^\omega]_g$. Let $y \in T_1^\omega$ be such that $x \approx_g y$ and $ny \in \sigma(L)$. Let $v \in L$ with $\sigma(v) = ny$. We may write $u = u_0cu'$ and $v = v_0cv'$ with $u_0, v_0 \in A^*$, $h(u_0) = n = h(v_0)$, $u', v' \in (A^*c)^\omega$, $x = \sigma(u')$ and $y = \sigma(v')$. Since $x \approx_g y$, using the first two auxiliary results above and the fact that the mapping $\sigma : (A^*c)^\omega \to T_1^\omega$ is surjective, we get $\sigma(u') \sim_g \sigma(w_1) \sim_g \cdots \sim_g \sigma(w_k) \sim_g \sigma(v')$ for some $w_1, \ldots, w_k \in (A^*c)^\omega$. From the third auxiliary result, we get $cu' \approx_h cv'$. Hence, using $h(u_0) = h(v_0)$, we obtain $u = u_0cu' \approx_h v_0cv' = v$. Since $v \in L$ and L is recognized by h, we get $u \in L$ as desired.

Finally, let $K = K_0 \cup K_1 \cup K_2$. We have already seen that $L = \sigma^{-1}(K)$. It remains to show that K is definable in $\mathrm{LTL}_T[\mathsf{XU}]$. Let $N \subseteq T^\infty$, then, by definition, the language $[N]_g$ is recognized by g which is a morphism to the aperiodic monoid M' with $|M'| < |M|$. By induction on the size of the monoid, we deduce that for all $n \in T_1$ and $N \subseteq T^\infty$ there exists $\varphi \in \mathrm{LTL}_T[\mathsf{XU}]$ such that $[N]_g = \mathcal{L}_{n,T}(\varphi)$. We easily check that $n\mathcal{L}_{n,T}(\varphi) = \mathcal{L}_T(n \wedge \varphi)$. Therefore, the language $n[N]_g$ is definable in $\mathrm{LTL}_T[\mathsf{XU}]$. Moreover, K_0, nT_1^*m and nT_1^ω are obviously definable in $\mathrm{LTL}_T[\mathsf{XU}]$. Therefore, K is definable in $\mathrm{LTL}_T[\mathsf{XU}]$. Q.E.D. (Lemma 8.2)

Let $b \in \Sigma$ be a letter. For a nonempty word $v = a_0a_1a_2 \cdots \in \Sigma^\infty \setminus \{\varepsilon\}$ and a position $0 \le i < |v|$, we denote by $\mu_b(v,i)$ the largest factor of v starting at position i and not containing the letter b except maybe a_i. Formally, $\mu_b(v,i) = a_ia_{i+1}\cdots a_\ell$ where $\ell = \max\{k \mid i \le k < |v| \text{ and } a_j \ne b \text{ for all } i < j \le k\}$.

Lemma 8.3 (Lifting)**.** For each formula $\varphi \in \mathrm{LTL}_\Sigma[\mathsf{XU}]$, there exists a formula $\overline{\varphi}^b \in \mathrm{LTL}_\Sigma[\mathsf{XU}]$ such that for each $v \in \Sigma^\infty \setminus \{\varepsilon\}$ and each $0 \le i < |v|$, we have $v, i \models \overline{\varphi}^b$ if and only if $\mu_b(v,i), 0 \models \varphi$.

Proof. The construction is by structural induction on φ. We let $\overline{a}^b = a$. Then, we have $\overline{\neg\varphi}^b = \neg\overline{\varphi}^b$ and $\overline{\varphi \vee \psi}^b = \overline{\varphi}^b \vee \overline{\psi}^b$ as usual. For next-until, we define $\overline{\varphi \,\mathsf{XU}\, \psi}^b = (\neg b \wedge \overline{\varphi}^b) \,\mathsf{XU}\, (\neg b \wedge \overline{\psi}^b)$.

Assume that $v, i \models \overline{\varphi \,\mathsf{XU}\, \psi}^b$. We find $i < k < |v|$ such that $v, k \models \neg b \wedge \overline{\psi}^b$ and $v, j \models \neg b \wedge \overline{\varphi}^b$ for all $i < j < k$. We deduce that $\mu_b(v,i) = a_ia_{i+1}\cdots a_\ell$ with $\ell > k$ and that $\mu_b(v,i), k-i \models \psi$ and $\mu_b(v,i), j-i \models \varphi$ for all $i < j < k$. Therefore, $\mu_b(v,i), 0 \models \varphi \,\mathsf{XU}\, \psi$ as desired. The converse can be shown similarly. Q.E.D. (Lemma 8.3)

Lemma 8.4. For all $\xi \in \mathrm{LTL}_T[\mathsf{XU}]$, there exists a formula $\widetilde{\xi} \in \mathrm{LTL}_\Sigma[\mathsf{XU}]$ such that for all $v \in \Sigma^\infty$ we have $cv, 0 \models \widetilde{\xi}$ if and only if $\sigma(v), 0 \models \xi$.

Proof. The proof is by structural induction on ξ. The difficult cases are for the constants $m \in T_1$ or $m \in T_2$.

Assume first that $\xi = m \in T_1$. We have $\sigma(v), 0 \models m$ if and only if $v = ucv'$ with $u \in A^* \cap h^{-1}(m)$. The language $A^* \cap h^{-1}(m)$ is recognized by the restriction $h_{\restriction A} : A^* \to M$. By induction on the size of the alphabet, we find a formula $\varphi_m \in \mathrm{LTL}_A[\mathsf{XU}]$ such that $\mathcal{L}_{c,A}(\varphi_m) = A^* \cap h^{-1}(m)$. We let $\widetilde{m} = \overline{\varphi_m}^c \wedge \mathsf{XF}\, c$. By Lemma 8.3, we have $cv, 0 \models \widetilde{m}$ if and only if $v = ucv'$ with $u \in A^*$ and $\mu_c(cv, 0), 0 \models \varphi_m$. Since $\mu_c(cv,0) = cu$, we deduce that $cv, 0 \models \widetilde{m}$ if and only if $v = ucv'$ with $u \in \mathcal{L}_{c,A}(\varphi_m) = A^* \cap h^{-1}(m)$.

Next, assume that $\xi = m \in T_2$. We have $\sigma(v) \models m$ if and only if $v \in A^\infty \cap m$ (note that letters from T_2 can also be seen as equivalence classes which are subsets of Σ^∞). The language $A^\infty \cap m$ is recognized by the restriction $h_{\restriction A}$. By induction on the size of the alphabet, we find a formula $\psi_m \in \mathrm{LTL}_A[\mathsf{XU}]$ such that $\mathcal{L}_{c,A}(\psi_m) = A^\infty \cap m$. Then, we let $\widetilde{m} = \overline{\psi_m}^{c} \wedge \neg\,\mathsf{XF}\,c$ and we conclude as above.

Finally, we let $\widetilde{\neg\xi} = \neg\widetilde{\xi}$, $\widetilde{\xi_1 \vee \xi_2} = \widetilde{\xi_1} \vee \widetilde{\xi_2}$ and for the modality next-until we define $\widetilde{\xi_1 \,\mathsf{XU}\, \xi_2} = (\neg c \vee \widetilde{\xi_1})\,\mathsf{U}\,(c \wedge \widetilde{\xi_2})$.

Assume that $\sigma(v), 0 \models \xi_1 \,\mathsf{XU}\, \xi_2$ and let $0 < k < |\sigma(v)|$ be such that $\sigma(v), k \models \xi_2$ and $\sigma(v), j \models \xi_1$ for all $0 < j < k$. Let $v_0 c v_1 c v_2 c \cdots$ be the c-factorization of v. Since the logics $\mathrm{LTL}_T[\mathsf{XU}]$ and $\mathrm{LTL}_\Sigma[\mathsf{XU}]$ are pure future, we have $\sigma(v), k \models \xi_2$ if and only if $\sigma(v_k c v_{k+1} \cdots), 0 \models \xi_2$ if and only if (by induction) $c v_k c v_{k+1} \cdots, 0 \models \widetilde{\xi_2}$ if and only if $cv, |cv_0 \cdots cv_{k-1}| \models \widetilde{\xi_2}$. Similarly, $\sigma(v), j \models \xi_1$ if and only if $cv, |cv_0 \cdots cv_{j-1}| \models \widetilde{\xi_1}$. Therefore, $cv, 0 \models \widetilde{\xi_1 \,\mathsf{XU}\, \xi_2}$. The converse can be shown similarly. Q.E.D. (Lemma 8.4)

We conclude now the proof of Proposition 8.1. We start with a language $L \subseteq \Sigma^\infty$ recognized by h. By Lemma 8.2, we find a formula $\xi \in \mathrm{LTL}_T[\mathsf{XU}]$ such that $L = \sigma^{-1}(\mathcal{L}_T(\xi))$. Let $\widetilde{\xi}$ be the formula given by Lemma 8.4. We claim that $L = \mathcal{L}_{c,\Sigma}(\widetilde{\xi})$. Indeed, for $v \in \Sigma^\infty$, we have $v \in \mathcal{L}_{c,\Sigma}(\widetilde{\xi})$ if and only if $cv, 0 \models \widetilde{\xi}$ if and only if (Lemma 8.4) $\sigma(v), 0 \models \xi$ if and only if $\sigma(v) \in \mathcal{L}_T(\xi)$ if and only if $v \in \sigma^{-1}(\mathcal{L}_T(\xi)) = L$. Q.E.D. (Proposition 8.1)

9 The separation theorem

As seen in Section 7, an $\mathrm{LTL}_\Sigma[\mathsf{YS}, \mathsf{XU}]$ formula φ can be viewed as a first-order formula with one free variable. The converse, in a stronger form, is established in this section.

Proposition 9.1. For all first-order formulae ξ in one free variable we find a finite list $(K_i, a_i, L_i)_{i=1,\ldots,n}$ where each $K_i \in \mathbf{SF}(\Sigma^*)$ and each $L_i \in \mathbf{SF}(\Sigma^\infty)$ and a_i is a letter such that for all $u \in \Sigma^*$, $a \in \Sigma$ and $v \in \Sigma^\infty$ we have

$$(uav, |u|) \models \xi \text{ if and only if } u \in K_i, a = a_i \text{ and } v \in L_i \text{ for some } 1 \leq i \leq n.$$

Proof. By Proposition 4.2, with $V = \{x\}$ we have $[\![\xi]\!]_V \in \mathbf{SF}(\Sigma_V^\infty)$. Hence, we can use Lemma 3.2 with $A = \Sigma_V^{x=1}$ and $B = \Sigma_V^{x=0}$. Note that $\mathcal{N}_V = B^* A B^\infty$. Hence, we obtain

$$[\![\xi]\!]_V = \bigcup_{i=1,\ldots,n} K_i' \cdot a_i' \cdot L_i'$$

with $a_i' \in A$, $K_i' \in \mathbf{SF}(B^*)$ and $L_i' \in \mathbf{SF}(B^\infty)$ for all i. Let $\pi : B^\infty \to \Sigma^\infty$ be the *bijective renaming* defined by $\pi(a, \tau) = a$. Star-free sets are preserved by injective renamings. Hence, we can choose $K_i = \pi(K_i') \in \mathbf{SF}(\Sigma^*)$ and $L_i = \pi(L_i') \in \mathbf{SF}(\Sigma^\infty)$. Note also that $a_i' = (a_i, 1)$ for some $a_i \in \Sigma$. Q.E.D.

Theorem 9.2 (Separation)**.** Let $\xi(x) \in \mathrm{FO}_\Sigma(<)$ be a first-order formula with one free variable x. Then, $\xi(x) = \zeta(x)$ for some LTL formula $\zeta \in \mathrm{LTL}_\Sigma[\mathsf{YS}, \mathsf{XU}]$. Moreover, we can choose for ζ a disjunction of conjunctions of pure past and pure future formulae:

$$\zeta = \bigvee_{1 \le i \le n} \psi_i \wedge a_i \wedge \varphi_i$$

where $\psi_i \in \mathrm{LTL}_\Sigma[\mathsf{YS}]$, $a_i \in \Sigma$ and $\varphi_i \in \mathrm{LTL}_\Sigma[\mathsf{XU}]$. In particular, every first-order formula with one free variable is equivalent to some formula in FO^3.

Note that we have already established a weaker version which applies to first-order sentences. Indeed, if ξ is a first-order sentence, then $\mathcal{L}(\varphi)$ is star-free by Proposition 10.3, hence aperiodic by Proposition 6.2, and finally definable in LTL by Proposition 8.1. The extension to first-order formulae with one free variable will also use the previous results.

Proof. By Proposition 9.1 we find for each ξ a finite list $(K_i, a_i, L_i)_{i=1,\ldots,n}$ where each $K_i \in \mathbf{SF}(\Sigma^*)$ and each $L_i \in \mathbf{SF}(\Sigma^\infty)$ and a_i is a letter such that for all $u \in \Sigma^*$, $a \in \Sigma$ and $v \in \Sigma^\infty$ we have

$$(uav, |u|) \models \xi \text{ if and only if } u \in K_i, a = a_i \text{ and } v \in L_i \text{ for some } 1 \le i \le n.$$

For a finite word $b_0 \cdots b_m$ where b_j are letters we let $\overleftarrow{b_0 \cdots b_m} = b_m \cdots b_0$. This means we read words from right to left. For a language $K \subseteq \Sigma^*$ we let $\overleftarrow{K} = \{\overleftarrow{w} \mid w \in K\}$. Clearly, each $\overleftarrow{K_i}$ is star-free. Therefore, using Propositions 6.2 and 8.1, for each $1 \le i \le n$ we find $\widehat{\psi_i}$ and $\varphi_i \in \mathrm{LTL}_\Sigma[\mathsf{XU}]$ such that $\mathcal{L}_{a_i}(\widehat{\psi_i}) = \overleftarrow{K_i}$ and $\mathcal{L}_{a_i}(\varphi_i) = L_i$. Replacing all operators XU by YS we can transform $\widehat{\psi_i} \in \mathrm{LTL}_\Sigma[\mathsf{XU}]$ into a formula $\psi_i \in \mathrm{LTL}_\Sigma[\mathsf{YS}]$ such that $(a\overleftarrow{w}, 0) \models \widehat{\psi_i}$ if and only if $(wa, |w|) \models \psi_i$ for all $wa \in \Sigma^+$. In particular, $K_i = \{w \in \Sigma^* \mid wa_i, |w| \models \psi_i\}$.

It remains to show that $\xi(x) = \zeta(x)$ where $\zeta = \bigvee_{1 \le i \le n} \psi_i \wedge a_i \wedge \varphi_i$. Let $w \in \Sigma^\infty \setminus \{\varepsilon\}$ and p be a position in w.

Assume first that $(w, p) \models \xi(x)$ and write $w = uav$ with $|u| = p$. We have $u \in K_i$, $a = a_i$ and $v \in L_i$ for some $1 \le i \le n$. We deduce that $ua_i, |u| \models \psi_i$ and $a_i v, 0 \models \varphi_i$. Since ψ_i is pure past and φ_i is pure future, we deduce that $ua_i v, |u| \models \psi_i \wedge a_i \wedge \varphi_i$. Hence we get $w, p \models \zeta$.

Conversely, assume that $w, p \models \psi_i \wedge a_i \wedge \varphi_i$ for some i. As above, we write $w = ua_i v$ with $|u| = p$. Since ψ_i is pure past and φ_i is pure future, we deduce that $ua_i, |u| \models \psi_i$ and $a_i v, 0 \models \varphi_i$. Therefore, $u \in K_i$ and $v \in L_i$. We deduce that $(w, p) \models \xi(x)$. Q.E.D.

10 Variations

This section provides an alternative way to establish the bridge from first-order to star freeness and an alternative proof for Theorem 9.2.

There is a powerful tool to reason about first-oder definable languages which we did not discuss: Ehrenfeucht-Fraïssé-games. These games lead to an immediate proof of a congruence lemma, which is given in Lemma 10.2 below. On the other hand, in our context, it would be the only place where we could use the power of Ehrenfeucht-Fraïssé-games, therefore we skip this notion and we use Lemma 10.1 instead.

Before we continue we introduce a few more notations. The *quantifier depth* $\mathrm{qd}(\varphi)$ of a formula φ is defined inductively. For the atomic formulae $\bot$, P, and $x < y$ it is zero, the use of the logical connectives does not increase it, it is the maximum over the operands, but adding a quantifier in front increases the quantifier depth by one. For example, the following formula in one free variable y has quantifier depth two:

$$\forall x\ (\exists y\, P(x) \wedge \neg P(y)) \vee (\exists z\, P(z) \wedge (x < z) \vee (z < y))$$

By $\mathrm{FO}_{m,k}$ we mean the set of all formulae of quantifier depth at most m and where the free variables are in the set $\{x_1, \ldots, x_k\}$, and FO_m is a short-hand of $\mathrm{FO}_{m,0}$; it is the set of sentences of quantifier-depth at most m.

We say that formulae $\varphi, \psi \in \mathrm{FO}_{m,k}$ are equivalent if $\mathcal{L}(\varphi) = \mathcal{L}(\psi)$ (for all Σ). Since the set of unary predicates is finite, there are, up to equivalence, only finitely many formulae in $\mathrm{FO}_{m,k}$ as soon as k and m are fixed. This is true for $m = 0$, because over any finite set of formulae there are, up to equivalence, only finitely many Boolean combinations. For $m > 0$ we have, by induction, only finitely many formulae of type $\exists x_{k+1}\, \varphi$ where φ ranges over $\mathrm{FO}_{m-1,k+1}$. A formula in $\mathrm{FO}_{m,k}$ is a Boolean combination over such formulae, as argued for $m = 0$ there are only finitely many choices.

10.1 The congruence lemma

Recall that $\Sigma^\infty_{(k)}$ means the set of pairs $(w, \overline{p})$, where $w \in \Sigma^\infty$ is a finite or infinite word and $\overline{p} = (p_1, \ldots, p_k)$ is a k-tuple of positions in $\mathrm{pos}(w)$. If we have $(u, \overline{p}) \in \Sigma^*_{(k)}$ and $(v, \overline{q}) \in \Sigma^\infty_{(\ell)}$, then we can define the concatenation in the natural way by shifting $\overline{q}$:

$$(u, \overline{p}) \cdot (v, \overline{q}) = (uv, p_1, \ldots, p_k, |u| + q_1, \ldots, |u| + q_\ell) \in \Sigma^\infty_{(k+\ell)}.$$

For each k and m and $(w, \overline{p}) \in \Sigma^\infty_{(k)}$ we define classes as follows:

$$[(w, \overline{p})]_{m,k} = \bigcap_{\varphi \in \mathrm{FO}_{m,k} \mid (w,\overline{p}) \models \varphi} \mathcal{L}(\varphi).$$

For $k = 0$ we simply write $[w]_{m,0}$. Since $\mathrm{qd}(\varphi) = \mathrm{qd}(\neg\varphi)$ and $\mathcal{L}(\neg\varphi) = \Sigma^{\infty}_{(k)} \setminus \mathcal{L}(\varphi)$ we obtain

$$\begin{aligned}[(w,\overline{p})]_{m,k} &= \bigcap_{\varphi\in\mathrm{FO}_{m,k} \mid (w,\overline{p})\models\varphi} \mathcal{L}(\varphi) \\ &= \bigcap_{\varphi\in\mathrm{FO}_{m,k} \mid (w,\overline{p})\models\varphi} \mathcal{L}(\varphi) \quad \setminus \bigcup_{\varphi\in\mathrm{FO}_{m,k} \mid (w,\overline{p})\not\models\varphi} \mathcal{L}(\varphi).\end{aligned}$$

Note that $(u',\overline{p}') \in [(u,\overline{p})]_{m,k}$ if and only if $(u,\overline{p}) \models \varphi \iff (u',\overline{p}') \models \varphi$ for all $\varphi \in \mathrm{FO}_{m,k}$ if and only if $[(u',\overline{p}')]_{m,k} = [(u,\overline{p})]_{m,k}$.

Lemma 10.1. Let $[(u,\overline{p})]_{m,k} = [(u',\overline{p}')]_{m,k}$ with $m \geq 1$, $\overline{p} = (p_1,\ldots,p_k)$, and $\overline{p}' = ({p'}_1,\ldots,{p'}_k)$. Then for all positions $p_{k+1} \in \mathrm{pos}(u)$ there exists a position $p'_{k+1} \in \mathrm{pos}(u')$ such that

$$[(u,(p_1,\ldots,p_{k+1}))]_{m-1,k+1} = [(u',({p'}_1,\ldots,{p'}_{k+1}))]_{m-1,k+1}.$$

Proof. Choose some $p_{k+1} \in \mathrm{pos}(u)$. We are looking for a position $p'_{k+1} \in \mathrm{pos}(u')$ such that for all $\psi \in \mathrm{FO}_{m-1,k+1}$ we have $(u,(p_1,\ldots,p_{k+1})) \models \psi$ if and only if $(u',({p'}_1,\ldots,{p'}_{k+1})) \models \psi$.

Consider the following finite (up to equivalence) conjunction:

$$\Psi = \bigwedge_{\psi\in\mathrm{FO}_{m-1,k+1} \mid (u,(p_1,\ldots,p_{k+1}))\models\psi} \psi.$$

We have $(u,(p_1,\ldots,p_{k+1})) \models \Psi$, $\mathrm{qd}(\exists x_{k+1}\Psi) \leq m$ and $(u,\overline{p}) \models \exists x_{k+1}\Psi$. Hence $(u',\overline{p}') \models \exists x_{k+1}\Psi$; and therefore there is some $p'_{k+1} \in \mathrm{pos}(u')$ such that $(u',({p'}_1,\ldots,{p'}_{k+1})) \models \Psi$.

Finally, for each $\psi \in \mathrm{FO}_{m-1,k+1}$, either Ψ implies ψ or Ψ implies $\neg\psi$, because either $(u,(p_1,\ldots,p_{k+1})) \models \psi$ or $(u,(p_1,\ldots,p_{k+1})) \models \neg\psi$. Hence, if $(u,(p_1,\ldots,p_{k+1})) \models \psi$, then $(u',({p'}_1,\ldots,{p'}_{k+1})) \models \psi$, too. If $(u,(p_1,\ldots,p_{k+1})) \models \neg\psi$, then $(u',({p'}_1,\ldots,{p'}_{k+1})) \models \neg\psi$, too. The result follows. Q.E.D.

The next lemma is known as *congruence lemma.*

Lemma 10.2. Let $[(u,\overline{p})]_{m,k} = [(u',\overline{p}')]_{m,k}$ and $[(v,\overline{q})]_{m,\ell} = [(v',\overline{q}')]_{m,\ell}$, where u and u' are finite words. Then we have

$$[(u,\overline{p})\cdot(v,\overline{q})]_{m,k+\ell} = [(u',\overline{p}')\cdot(v',\overline{q}')]_{m,k+\ell}.$$

Proof. We have to show that for all $\varphi \in \mathrm{FO}_{m,k}$ we have $(u,\overline{p})\cdot(v,\overline{q}) \models \varphi$ if and only if $(u',\overline{p}')\cdot(v',\overline{q}') \models \varphi$. Since we get Boolean combinations for free, we may assume that φ is of the form $\exists x_{k+1}\psi$ or an atomic formula.

If $\varphi = P(x_i)$ and $i \leq k$, then we have $(u, \overline{p}) \cdot (v, \overline{q}) \models P(x_i)$ if and only if $(u, \overline{p}) \models P(x_i)$ and the result follows. The case $i > k$ is symmetric.

If $\varphi = x_i < x_j$, assume first $i \leq k$. If, in addition, $j > k$, then $(u, \overline{p}) \cdot (v, \overline{q}) \models x_i < x_j$ is true, otherwise $i, j \leq k$ and we see that $(u, \overline{p}) \cdot (v, \overline{q}) \models x_i < x_j$ if and only if $(u, \overline{p}) \models x_i < x_j$. The case $i > k$ is similar.

It remains to deal with $\varphi = \exists x_{k+1} \psi$. Assume $(u, \overline{p}) \cdot (v, \overline{q}) \models \varphi$. We have to show that $(u', \overline{p}') \cdot (v', \overline{q}') \models \varphi$. Assume first that there is some position $p_{k+1} \in \mathrm{pos}(u)$ such that

$$(u, (p_1, \ldots, p_{k+1})) \cdot (v, \overline{q}) \models \psi.$$

By Lemma 10.1 there is some position $p'_{k+1} \in \mathrm{pos}(u')$ such that

$$[(u, (p_1, \ldots, p_{k+1}))]_{m-1,k+1} = [(u', ({p'}_1, \ldots, {p'}_{k+1}))]_{m-1,k+1}.$$

We have $\mathrm{qd}(\psi) \leq m - 1$, hence by induction on m we deduce

$$(u', ({p'}_1, \ldots, {p'}_{k+1})) \cdot (v', \overline{q}') \models \psi$$

This in turn implies

$$(u', \overline{p}') \cdot (v', \overline{q}') \models \exists x_{k+1} \psi.$$

The case where $(u, \overline{p}) \cdot (v, (q_1, \ldots, q_{\ell+1})) \models \psi$ for some position $q_{\ell+1}$ in v is similar. Q.E.D.

10.2 From FO to SF and separation via the congruence lemma

It is convenient to define a *dot-depth hierarchy*. The Boolean combinations of Σ^* are of dot-depth zero. In order to define the m-th level of the dot-depth hierarchy, $m \geq 1$, one forms the Boolean closure of the languages $K \cdot a \cdot L$, where $a \in \Sigma$ and K, L are of level at most $m - 1$. Note that there are only finitely many languages of level m.

Proposition 10.3. Let $m \geq 0$ and $\varphi \in \mathrm{FO}_m$ be a sentence with quantifier-depth at most m. Then we find a star-free language L of level at most m in the dot-depth hierarchy such that $\mathcal{L}(\varphi) = L$.

Proof. We perform an induction on m. The case $m = 0$ is trivial since the only sentences are $\top$ and $\bot$. Hence let $m > 0$. By definition,

$$[w]_{m-1,0} = \bigcap_{\psi \in \mathrm{FO}_{m-1} | w \models \psi} \mathcal{L}(\psi).$$

By induction on m we may assume that $[w]_{m-1,0}$ is star-free of dot-depth $m - 1$. Consider next a sentence $\varphi \in \mathrm{FO}_m$. We want to show that $\mathcal{L}(\varphi)$ is of dot-depth m. Languages of dot-depth m form a Boolean algebra, thus by structural induction it is enough to consider a sentence $\varphi = \exists x \psi$. Consider the following union:

$$T = \bigcup_{(uav,|u|) \models \psi} [u]_{m-1,0} \cdot a \cdot [v]_{m-1,0}.$$

Since $[u]_{m-1,0}$ and $[v]_{m-1,0}$ are star-free sets of dot-depth $m-1$, there are finitely many sets $[u]_{m-1,0} \cdot a \cdot [v]_{m-1,0}$ in the union above. In fact, it is a star-free expression of dot-depth m.

It remains to show that $\mathcal{L}(\varphi) = T$. Let $w \in \mathcal{L}(\varphi) = \mathcal{L}(\exists x \psi)$. We find a position in w and a factorization $w = uav$ such that $(uav, |u|) \models \psi$. Since $u \in [u]_{m-1,0}$ and $v \in [v]_{m-1,0}$, we have $uav \in T$, hence $\mathcal{L}(\varphi) \subseteq T$.

The converse follows by a twofold application of the congruence lemma (Lemma 10.2): Indeed, let $u' \in [u]_{m-1,0}$ and $v' \in [v]_{m-1,0}$ then

$$\begin{aligned}
[(u'a, |u'|)]_{m-1,1} &= [(u') \cdot (a, 0)]_{m-1,1} \\
&= [(u) \cdot (a, 0)]_{m-1,1} = [(ua, |u|)]_{m-1,1} \\
[(u'av', |u'|)]_{m-1,1} &= [(u'a, |u'|) \cdot (v')]_{m-1,1} \\
&= [(ua, |u|) \cdot (v)]_{m-1,1} = [(uav, |u|)]_{m-1,1}.
\end{aligned}$$

Therefore, $(uav, |u|) \models \psi$ implies $(u'av', |u'|) \models \psi$ and this implies $u'av' \models \exists x \psi$. Thus, $T \subseteq \mathcal{L}(\Psi)$. Q.E.D.

The congruence lemma yields an alternative way to show Proposition 9.1 (and hence the separation theorem, Theorem 9.2) too.

Proof of Proposition 9.1 based on Lemma 10.2. Let $\mathrm{qd}(\xi) = m$ for some $m \geq 0$. As in the proof of Proposition 10.3 define a language:

$$T = \bigcup_{(uav,|u|) \models \xi} [u]_{m,0} \cdot a \cdot [v]_{m,0}.$$

The union is finite and the classes $[u]_{m,0} \cap \Sigma^*$ and $[v]_{m,0}$ are first-order definable. First-order definable languages are star-free by Proposition 10.3. Thus, we can rewrite T as desired:

$$T = \bigcup_{i=1,\ldots,n} K_i \cdot a_i \cdot L_i.$$

Moreover, the proof of Proposition 10.3 has actually shown that $(uav, |u|) \models \xi$ if and only if $u \in K_i$, $a = a_i$ and $v \in L_i$ for some $1 \leq i \leq n$.

For convenience, let us repeat the argument. If $(uav, |u|) \models \xi$, then we find an index i such that $u \in K_i$, $a = a_i$, and $v \in L_i$. For the converse, let

$u' \in K_i$, $a' = a_i$, and $v' \in L_i$ for some i. We have to show $(u'a'v, |u'|) \models \xi$. By definition of T, we have $u' \in K_i = [u]_{m,0} \cap \Sigma^*$, $a' = a$, and $v' \in L_i = [v]_{m,0}$ for some $(uav, |u|) \models \xi$. The congruence lemma (Lemma 10.2) applied twice yields:

$$[(a'v', 0)]_{m,1} = [(a', 0) \cdot (v')]_{m,1} = [(a, 0) \cdot (v)]_{m,1} = [(av, 0)]_{m,1}.$$
$$[(u'a'v', |u'|)]_{m,1} = [(u') \cdot (a'v', 0)]_{m,1} = [(u) \cdot (av, 0)]_{m,1} = [(uav, |u|)]_{m,1}.$$

We deduce $(u'a'v, |u'|) \models \xi$. Q.E.D.

11 Counter-free and aperiodic Büchi automata

There is a standard way to introduce recognizable languages with finite automata. Since we deal with finite and infinite words we use Büchi automata with two acceptance conditions, one for finite words and the other for infinite words. A *Büchi automaton* is given as a tuple

$$\mathcal{A} = (Q, \Sigma, \delta, I, F, R),$$

where Q is a finite set of *states* and δ is a relation:

$$\delta \subseteq Q \times \Sigma \times Q.$$

The set $I \subseteq Q$ is called the set of *initial states*, the sets $F, R \subseteq Q$ consist of *final* and *repeated* states respectively.

If δ is the graph of a partially defined function from $Q \times \Sigma$ to Q and if in addition $|I| \leq 1$, then the automaton is called *deterministic*.

A *path* means in this section a finite or infinite sequence

$$\pi = p_0, a_0, p_1, a_1, p_2, a_2, \ldots$$

such that $(p_i, a_i, p_{i+1}) \in \delta$ for all $i \geq 0$. We say that the path is *accepting*, if it starts in an initial state $p_0 \in I$ and either it is finite and ends in a final state from F or it is infinite and visits infinitely many repeated states from R. The label of the above path π is the word $u = a_0a_1a_2 \cdots \in \Sigma^\infty$. The language *accepted* by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$ and is defined as the set of words which appear as label of an accepting path. We have $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\infty$. Languages of the form $\mathcal{L}(\mathcal{A})$ are called *regular* or *regular ω-languages*, if $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$

McNaughton and Papert have introduced the classical notion of a *counter-free* deterministic finite automaton, [19]. They showed that counter-freeness captures star-freeness (hence aperiodicity) for languages over finite words. Our aim is to give a natural notion of counter-freeness for non deterministic (Büchi) automata such that a language $L \subseteq \Sigma^\infty$ is aperiodic if and only if it can be accepted by a counter-free Büchi automaton. To the best of our knowledge, all previous extensions to infinite words used deterministic automata.

If $p, q \in Q$ are states of $\mathcal{A}$, then we let $L_{p,q}$ be the set of labels of finite paths from p to q.

Definition 11.1. A Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F, R)$ is called *counter-free*, if $u^m \in L_{p,p}$ implies $u \in L_{p,p}$ for all states $p \in Q$, words $u \in \Sigma^*$, and $m \geq 1$.

Note that the definition is taking only the underlying transition relation δ into account, but does not depend on the sets I, F, or R. For deterministic automata counter-freeness coincides with the standard notion as introduced in [19]. We start with the classical result of [19] on finite words.

Lemma 11.2. Let $L \subseteq \Sigma^*$ be a language of finite words recognized by a morphism h from Σ^* to some finite aperiodic monoid M. Then the minimal deterministic automaton recognizing L is counter-free.

Proof. The states of the minimal deterministic automaton recognizing L can be written as

$$L(u) = u^{-1}L = \{w \in \Sigma^* \mid uw \in L\}$$

with $u \in \Sigma^*$ and all transitions have the form $(L(u), a, L(ua))$. Assume that $L(uv^m) = L(u)$ for some $m \geq 1$. Then we can take m as large as we wish and since M is aperiodic we may assume that $x^{m+1} = x^m$ for all $x \in M$. Since h recognizes L, we deduce that $uv^m w \in L$ if and only if $uv^{m+1}w \in L$ for all $w \in \Sigma^*$, i.e., $L(uv^m) = L(uv^{m+1})$. Using $L(uv^m) = L(u)$ we obtain,

$$L(u) = L(uv^m) = L(uv^{m+1}) = L((uv^m)v) = L(uv).$$

Hence, the automaton is counter-free. Q.E.D.

Lemma 11.3. Let $L \subseteq \Sigma^*$ and $L' \subseteq \Sigma^\infty$ be accepted by counter-free automata. Then $L \cdot L'$ can be accepted by some counter-free automaton.

Proof. Trivial, just consider a usual construction showing that regular languages are closed under concatenation. Essentially, the new automaton is the disjoint union of the two automata with additional transitions allowing to switch from the first one to the second one. Therefore, a loop in the new automaton is either a loop in the first one or a loop in the second one. Thus, we have no *new* loops and hence the result. Q.E.D.

Proposition 11.4. Let $L \subseteq \Sigma^\infty$ be recognized by a morphism $h : \Sigma^* \to M$ to some finite aperiodic monoid M. Then we find a counter-free Büchi automaton $\mathcal{A}$ with $L = \mathcal{L}(\mathcal{A})$.

Proof. By Remark 5.3 we can write L as a finite union of languages of type UV^ω, where U and V are aperiodic languages of finite words and where moreover $V = h^{-1}(e)$ for some idempotent $e \in M$. By a simple construction on monoids we may actually assume that $h^{-1}(1) = \{\varepsilon\}$ and then in turn that $e \neq 1$. Hence without restriction we have $V \subseteq \Sigma^+$. The *union* of two counter-free Büchi automata is counter-free and recognizes the union of the accepted languages. Therefore we content to construct a counter-free Büchi automaton for the language UV^ω. By Lemmata 11.2 and 11.3 it is enough to find a counter-free automaton for V^ω. The trick is that V^ω can be accepted by some deterministic Büchi automaton. Define the *witness* W by

$$W = V \cdot (V \setminus V\Sigma^+).$$

The language W is aperiodic. By Lemma 11.2, its minimal automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F, \varnothing)$ is counter-free. View this automaton as a deterministic Büchi automaton $\mathcal{A}' = (Q, \Sigma, \delta, I, \varnothing, F)$ where final states are now repeated states. (It is also counter-free according to Definition 11.1, because it is deterministic.)

The automaton $\mathcal{A}'$ accepts those infinite strings where infinitely many prefixes are in W. We want to show that this coincides with V^ω. Clearly, $w \in V^\omega$ implies that w has infinitely many prefixes in W. We show that the converse holds, too. Let $w \in \Sigma^\omega$ and w_i be a list of infinitely many prefixes in W. For each w_i choose some factorization $w_i = u_i v_i$ with $u_i \in V$ and $v_i \in V \setminus V\Sigma^+$. Note there might be several such factorizations. However, if $w_i \neq w_j$, then we cannot have $u_i = u_j$, because otherwise v_i is a strict prefix of v_j or vice versa. Thus, we find infinitely many u_i and by switching to some infinite subsequence we may assume

$$u_1 < u_1 v_1 < u_2 < u_2 v_2 < u_3 < u_3 v_3 \cdots$$

where $\leq$ means the prefix relation. For all i we can write $u_{i+1} = u_i v_i v_i'$. We have

$$e = h(u_{i+1}) = h(u_i v_i v_i') = e \cdot e \cdot h(v_i') = e \cdot h(v_i') = h(v_i) \cdot h(v_i') = h(v_i v_i').$$

Hence

$$w = u_1 (v_1 v_1')(v_2 v_2')(v_3 v_3') \cdots \in V^\omega.$$

Therefore, V^ω is accepted by the counter-free Büchi automaton $\mathcal{A}'$. Q.E.D.

To prove that conversely, a language accepted by a counter-free Büchi automaton is aperiodic, we shall use a weaker notion. The following definition coincides with the one given in [16, Definition 3.1] for non-deterministic finite automata in the context of finite transducers.

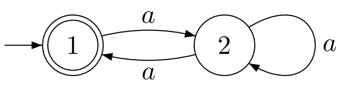


FIGURE 1. The non-deterministic Büchi automaton $\mathcal{A}_1$

Definition 11.5. A Büchi automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F, R)$ is called *aperiodic*, if for some $m \geq 1$ we have:

$$u^m \in L_{p,q} \iff u^{m+1} \in L_{p,q}$$

for all states $p, q \in Q$ and words $u \in \Sigma^*$.

Lemma 11.6. Let $\mathcal{A}$ be a Büchi automaton.

1. If $\mathcal{A}$ is counter-free, then $\mathcal{A}$ is aperiodic.

2. If $\mathcal{A}$ is deterministic and aperiodic, then $\mathcal{A}$ is counter-free.

Proof. 1. Let $u^{m+1} \in L_{p,q}$. If m is large enough, we find $m+1 = k_1 + \ell + k_2$ with $\ell \geq 2$ and a state s such that $u^{k_1} \in L_{p,s}$, $u^\ell \in L_{s,s}$, and $u^{k_2} \in L_{s,q}$. Since the automaton is counter-free, we obtain $u \in L_{s,s}$ and therefore $u^m \in L_{p,q}$. Similarly, we can show that $u^m \in L_{p,q}$ implies $u^{m+1} \in L_{p,q}$.

2. Let $u^m \in L_{p,p}$ for some $m \geq 1$. Then $u^m \in L_{p,p}$ for m as large as we wish. Since the automaton is aperiodic we have $u^m, u^{m+1} \in L_{p,p}$ for some m large enough. Since the automaton is deterministic, we deduce that $u \in L_{p,p}$, too. Q.E.D.

Remark 11.7. Consider the non-deterministic Büchi automaton $\mathcal{A}_1$ of Figure 1 which accepts $\{a^\omega\}$. The automaton $\mathcal{A}_1$ is aperiodic, but not counter-free.

The *transformation monoid* $T(\mathcal{A})$ of $\mathcal{A}$ is realized as a submonoid of Boolean matrices. More precisely, let $\mathcal{A}$ have n states. We consider the monoid $\mathbb{B}^{n\times n}$ of $n \times n$ matrices over the finite commutative semiring $\mathbb{B} = \{0, 1\}$ with max as addition and the natural multiplication as product. For every word u we define a matrix $t(u) \in \mathbb{B}^{n\times n}$ by:

$$t(u)[p, q] = 1 \iff u \in L_{p,q}.$$

The mapping $t : \Sigma^* \to \mathbb{B}^{n\times n}$ is a monoid morphism, because $t(\varepsilon)$ is the identity matrix and we have for all $u, v \in \Sigma^*$:

$$t(u \cdot v)[p, q] = \sum_{r\in Q} t(u)[p, r] \cdot t(v)[r, q].$$

The transition monoid of $\mathcal{A}$ is $T(\mathcal{A}) = t(\Sigma^*) \subseteq \mathbb{B}^{n\times n}$.

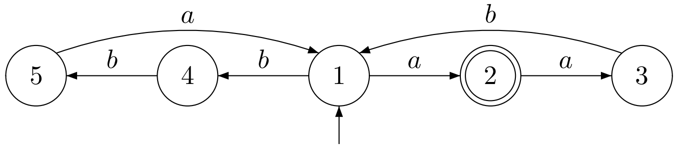


FIGURE 2. The deterministic and counter-free Büchi automaton $\mathcal{A}_2$

Remark 11.8. In terms of the transition monoid, Definition 11.5 says that a Büchi automaton $\mathcal{A}$ is aperiodic if and only if the monoid $T(\mathcal{A})$ is aperiodic.

The problem is that the morphism t to the transition monoid of $\mathcal{A}$ does not recognize $\mathcal{L}(\mathcal{A})$, in general. Indeed consider the deterministic automaton $\mathcal{A}_2$ on Figure 2 where the only repeated state is 2. The automaton accepts the language

$$L = \{w \in \{aab, bba\}^\omega \mid \text{the factor } aa \text{ appears infinitely often}\}.$$

Consider the matrix $t(aab)$ for which all entries are 0 except $t(aab)[1,1] = 1$. We have $t(aab) = t(bba)$, but $(aab)^\omega \in L$ and $(bba)^\omega \notin L$. Thus t does not recognize L.

It is therefore somewhat surprising that aperiodicity of $T(\mathcal{A})$ implies that $\mathcal{L}(\mathcal{A})$ is an aperiodic language. This is proved in Proposition 11.11, below.

We still need another concept. In Büchi's original proof that regular ω-languages are closed under complementation (see [3]) he used a finer congruence than given by the morphism t. To reflect this, we switch from the Boolean semiring $\mathbb{B}$ to the finite commutative semiring $K = \{0, 1, \infty\}$. The semiring structure of K is given by $x + y = \max\{x, y\}$ and the natural multiplication with the convention $0 \cdot \infty = 0$.

In order to take repeated states into account we let $R_{p,q} \subseteq L_{p,q}$ be the set of labels of *nonempty* and finite paths from p to q, *which use a repeated state at least once.* For every word u we define a matrix $h(u) \in K^{n \times n}$ by:

$$h(u)[p,q] = \begin{cases} 0 & \text{if } u \notin L_{p,q}, \\ 1 & \text{if } u \in L_{p,q} \setminus R_{p,q}, \\ \infty & \text{if } u \in R_{p,q}. \end{cases}$$

For the Büchi automaton $\mathcal{A}_2$ in Figure 2 we have $h(aab)[1,1] = \infty$, whereas $h(bba)[1,1] = 1$. For all other entries we have $h(aab)[p,q] = h(bba)[p,q] = 0$.

Note that $h(\varepsilon)$ is the identity matrix. In the semiring $K^{n \times n}$ we have as usual:

$$h(u \cdot v)[p,q] = \sum_{r \in Q} h(u)[p,r] \cdot h(v)[r,q].$$

Hence, $h : \Sigma^* \to K^{n\times n}$ is a monoid morphism and we can check easily that h recognizes $\mathcal{L}(\mathcal{A})$. The submonoid $BT(\mathcal{A}) = h(\Sigma^*) \subseteq K^{n\times n}$ is called either *Büchi's transition monoid* of $\mathcal{A}$ or the *ω-transition monoid* of $\mathcal{A}$. We obtain Büchi's result [3]:

Proposition 11.9. For every Büchi automaton $\mathcal{A}$ the morphism $h : \Sigma^* \to BT(\mathcal{A})$ onto the ω-transition monoid of $\mathcal{A}$ recognizes $\mathcal{L}(\mathcal{A})$.

Corollary 11.10. A language in $L \subseteq \Sigma^\infty$ can be accepted by some Büchi automaton if and only if it can be recognized by some morphism to some finite monoid.

Proof. Proposition 11.9 gives one direction. Conversely, assume that L is recognized by a morphism h from Σ^* to some finite monoid M. By Remark 5.3, L is a finite union of languages of type UV^ω, where $U, V \subseteq \Sigma^*$ are recognized by h. These sets are accepted by finite deterministic automata with M as set of states. Standard constructions on Büchi automata for union, concatenation, and ω-power yield the result. Q.E.D.

It also follows that regular ω-languages are closed under complementation, since recognizable languages are closed under complementation by definition (as they are unions of equivalence classes).

Proposition 11.11. Let $L \subseteq \Sigma^\infty$ a language. The following are equivalent.

1. There is a counter-free Büchi automaton $\mathcal{A}$ with $L = \mathcal{L}(\mathcal{A})$.
2. There is an aperiodic Büchi automaton $\mathcal{A}$ with $L = \mathcal{L}(\mathcal{A})$.
3. The language L is aperiodic.

Proof. $1 \Rightarrow 2$: Trivial by Lemma 11.6.1.

$2 \Rightarrow 3$: Let $\mathcal{A}$ have n states and consider Büchi's morphism $h : \Sigma^* \to K^{n\times n}$ as above. We show that the submonoid $BT(\mathcal{A}) = h(\Sigma^*) \subseteq K^{n\times n}$ is aperiodic. More precisely, we show for all states p, q and words u that $h(u^{2m})[p,q] = h(u^{2m+1})[p,q]$ as soon as m large enough.

Since the automaton is aperiodic we find a suitable m with $u^m \in L_{p,q}$ if and only if $u^{m+1} \in L_{p,q}$ for all states p, q and words u. We immediately get

$$h(u^{2m})[p,q] \geq 1 \iff h(u^{2m+1})[p,q] \geq 1.$$

Assume now that $h(u^{2m})[p,q] = \infty$. Then for some r we have $h(u^{2m})[p,q] = h(u^m)[p,r] \cdot h(u^m)[r,q]$ and by symmetry we may assume $h(u^m)[r,q] = \infty$ and $h(u^m)[p,r] \neq 0$. This implies $h(u^{m+1})[p,r] \neq 0$ and therefore $h(u^{2m+1})[p,q] = h(u^{m+1})[p,r] \cdot h(u^m)[r,q] = \infty$. Similarly, we can show that $h(u^{2m+1})[p,q] = \infty$ implies $h(u^{2m})[p,q] = \infty$.

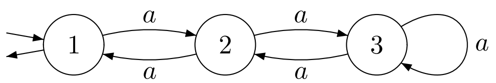


FIGURE 3. Aperiodicity does not imply counter-freeness for minimal size NFA.

Thus we have seen that $h(u^{2m})[p,q] = h(u^{2m+1})[p,q]$ for all $u \in \Sigma^*$ and all states p, q. This shows that L is recognized by some aperiodic monoid (of size at most 3^{n^2}).

$3 \Rightarrow 1$: This is the contents of Proposition 11.4. Q.E.D.

The automaton $\mathcal{A}_2$ above is counter-free, and this notion does not depend on final or repeated states. In particular, the languages $\{aab, bba\}^\omega$ and $\{aab, bba\}^*$ are further examples of aperiodic languages.

We conclude this section with several remarks concerning counter-freeness for Büchi automata.

Remark 11.12. If $L \subseteq \Sigma^\infty$ is aperiodic, then we actually find some Büchi automaton $\mathcal{A}$ with $L = \mathcal{L}(\mathcal{A})$, where for all states $p \in Q$, words $u \in \Sigma^*$, and $m \geq 1$ the following two conditions hold:

1. If $u^m \in L_{p,p}$, then $u \in L_{p,p}$.
2. If $u^m \in R_{p,p}$, then $u \in R_{p,p}$.

This is true, because all crucial constructions in the proof of Proposition 11.4 were done for deterministic automata. If an automaton is deterministic, then Condition 1 implies Condition 2, because if $u^m \in R_{p,p}$ and $u \in L_{p,p}$, then the path labeled by u^m from p to p visits the same states as the path labeled by u from p to p. For non-deterministic automata the second condition is a further restriction of counter-free automata.

Remark 11.13. For finite words, counter-freeness of the minimal automaton of a language $L \subseteq \Sigma^*$ characterizes aperiodicity of L. There is no canonical minimal Büchi automaton for languages of infinite words, but we may ask whether counter-freeness of a non-deterministic automaton of minimal size also characterizes aperiodicity. The answer is negative. Indeed, consider the language $L = \left\{\varepsilon, a^2\right\} \cup a^4a^*$ which is aperiodic and accepted by the 3-state automaton in Figure 3. This automaton is not counter-free since $a^2 \in L_{1,1}$ but $a \notin L_{1,1}$. We can check that L cannot be accepted by a 2-state automaton.

Remark 11.14. Let $\mathcal{A} = (Q, \Sigma, \delta, I)$ be a non-deterministic automaton and let $\mathcal{B} = (2^Q, \Sigma, \delta_{\mathcal{B}}, \{I\})$ be its (deterministic) subset automaton. Note

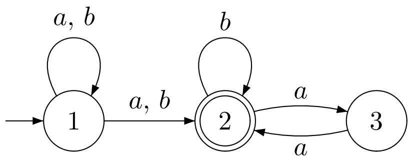


FIGURE 4. The Büchi automaton $\mathcal{A}$ accepting $\Sigma^+\{a^2, b\}^\omega$.

that, in this definition, we do not restrict to the accessible subsets from I. First, we prove that if $\mathcal{A}$ is counter-free, then so is $\mathcal{B}$. Assume that $\delta(X, u^m) = X$ for some $X \subseteq Q$, $u \in \Sigma^+$ and $m > 0$. Then, for each $p \in X$ we find some $p' \in X$ with $p \in \delta(p', u^m)$. Iterating these backward paths, we find $q \in X$ such that

$$q \xrightarrow{u^{jm}} q \xrightarrow{u^{km}} p$$

Since $\mathcal{A}$ is counter-free, it follows $q \xrightarrow{u} q$. Hence, $p \in \delta(X, u^{1+km}) = \delta(X, u)$. We have proved $X \subseteq \delta(X, u)$. It follows by induction that $\delta(X, u) \subseteq \delta(X, u^m) = X$. Therefore, $\mathcal{B}$ is counter-free.

Next, we show that if $\mathcal{B}$ is counter-free then $\mathcal{A}$ is aperiodic. Let $x \in T(\mathcal{A})$ be in the transition monoid of $\mathcal{A}$: $x = t(u)$ for some $u \in \Sigma^*$. We have $x^m = x^{m+k}$ for some $m, k > 0$. Let $X = x^m(Q) = \delta(Q, u^m)$. Since $x^m = x^{m+k}$ we have $\delta(X, u^k) = X$ and we deduce $\delta(X, u) = X$ since $\mathcal{B}$ is counter-free. Therefore, $x^m = x^{m+1}$ and we have shown that $T(\mathcal{A})$ is aperiodic.

Therefore, counter-freeness of the *full* subset automaton is another sufficient condition for aperiodicity. But, for this to hold over infinite words, it is important not to restrict to the subsets accessible from I. Indeed, let $\Sigma = \{a, b\}$ with $a \neq b$ and consider the language:

$$L = \Sigma^+\{a^2, b\}^\omega.$$

The non-deterministic 3-state Büchi automaton $\mathcal{A}$ in Figure 4 accepts L with $I = \{1\}$, $F = \varnothing$ and $R = \{2\}$ (an easy exercise shows that there is no deterministic Büchi automaton accepting L). The subset automaton restricted to the subsets reachable from $\{1\}$ is depicted in Figure 5. This automaton is counter-free, but L is not aperiodic.

12 Deciding aperiodicity in polynomial space

This section is devoted to a construction which shows that aperiodicity is decidable (in polynomial space) for recognizable languages. Thus, all properties mentioned in Theorem 1.1 are decidable for a regular ∞-languages.

Our aim is an optimal algorithm in a complexity theoretical meaning, and the best we can do is to find a polynomial space bounded algorithm.

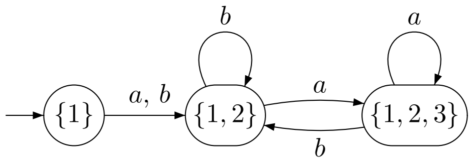


FIGURE 5. The subset automaton $\mathcal{B}$ of $\mathcal{A}$ restricted to reachable states.

This is indeed optimal, because PSPACE-hardness is known by [4]. It should be noted that our PSPACE-upper bound is not a formal consequence of [29] or any other reference we are aware of, because [29] deals only with deterministic automata over finite words. Moreover, our approach is not based on the syntactic congruence of Arnold [1]. Instead we start with any recognizing morphism and we consider its maximal aperiodic quotient. We check whether this monoid still recognizes the same language. This is possible in polynomial space, as we shall demonstrate below. We need an algebraic construction first.

Proposition 12.1. Let $h_1 : \Sigma^* \to M_1$ be a surjective morphism onto a finite monoid M_1 which recognizes L and let $m \geq |M_1|$. Let M_1' be the quotient of the monoid M_1 by the congruence generated by $\{x^m = x^{m+1} \mid x \in M_1\}$ and let $h_1' : \Sigma^* \to M_1'$ be the canonical morphism induced by h_1. Then L is aperiodic if and only if h_1' recognizes L.

Proof. First, If h_1' recognizes L, then L is aperiodic since M_1' is aperiodic by construction.

Conversely, if L is aperiodic, then there is some surjective morphism $h_2 : \Sigma^* \to M_2$ which recognizes L and where M_2 is aperiodic. We first show that L is also recognized by a *quotient* monoid M of both M_1 and M_2. This means that M is a homomorphic image of M_1 as well as of M_2.

$$\begin{tikzcd}
 & \Sigma^* \arrow[dl, "h_1"'] \arrow[dr, "h_2"] \arrow[dd, "h"] & \\
M_1 \arrow[dr, "\bar{h}_1"'] & & M_2 \arrow[dl, "\bar{h}_2"] \\
 & M &
\end{tikzcd}$$

We define the relation $H \subseteq \Sigma^* \times \Sigma^*$ by:

$$H = \{(u, v) \mid h_1(u) = h_1(v) \vee h_2(u) = h_2(v)\}.$$

The transitive closure H^+ of H is an equivalence relation, and easily seen to be a congruence. Thus, we can define the quotient monoid M of Σ^* by H^+. We have a canonical morphism $h : \Sigma^* \to M$ and $|M| \leq \min\{|M_1|, |M_2|\}$.

Since $h_i(u) = h_i(v)$ implies $h(u) = h(v)$ for all $u, v \in \Sigma^*$, the morphism h factorizes through M_1 and M_2 as shown in the diagram above: $h = \bar{h}_i \circ h_i$ for $i = 1, 2$.

We show that h recognizes L, too. First, we note that $H^+ = H^\ell$ where $\ell = \min\{|M_1|, |M_2|\}$. Indeed, if $u_0 \; H \; u_1 \cdots H \; u_k$ with $k \geq |M_1|$ then we find $0 \leq i < j \leq k$ with $h_1(u_i) = h_1(u_j)$ and we obtain $(u_0, u_k) \in H^{k-(j-i)}$. Now, consider some $u = \prod_{0 \leq i < n} u_i$ and $v = \prod_{0 \leq i < n} v_i$ with u_i, $v_i \in \Sigma^+$ such that $(u_i, v_i) \in H$ for all $0 \leq i < n$. Since $H^+ = H^\ell$ it is enough to see that $u \in L$ implies $v \in L$. Now, for all $0 \leq i < n$ there is $w_i \in \{u_i, v_i\}$ with $h_1(u_i) = h_1(w_i)$ and $h_2(w_i) = h_2(v_i)$. Since h_1 recognizes L, we have $u \in L$ implies $\prod_{0 \leq i < n} w_i \in L$, and this implies $v \in L$ since h_2 recognizes L.

The monoid M as constructed above is aperiodic, because it is a quotient monoid of M_2. But $|M| \leq |M_1| \leq m$, hence $x^m = x^{m+1}$ for all $x \in M$. By definition, M_1' is the quotient of the monoid M_1 by the congruence generated by $\{x^m = x^{m+1} \mid x \in M_1\}$. Since M satisfies all equations $x^m = x^{m+1}$, the morphism $\bar{h}_1 : M_1 \to M$ factorizes through M_1': $\bar{h}_1 = \bar{h}_1' \circ g$ where g is the canonical morphism from M_1 to M_1'.

$$\begin{tikzcd}
 & & \Sigma^* \arrow[lld, bend right, "h_1'"'] \arrow[ld, "h_1"'] \arrow[dd, "h"] \arrow[rd, "h_2"] & \\
M_1' \arrow[rrd, bend right, "\bar{h}_1'"'] & M_1 \arrow[l, "g"'] \arrow[rd, "\bar{h}_1"'] & & M_2 \arrow[ld, "\bar{h}_2"] \\
 & & M &
\end{tikzcd}$$

By definition, $h_1' = g \circ h_1$ and we deduce that $h = \bar{h}_1' \circ h_1'$. Hence, $h_1'(u) = h_1'(v)$ implies $h(u) = h(v)$ for all $u, v \in \Sigma^*$. Since h recognizes L, this implies that h_1' recognizes L, too. Q.E.D.

From Proposition 12.1, we can derive easily a pure decidability result. Indeed, if we start with a language L recognized by a Büchi automaton $\mathcal{A}$ with n states, we know that L is aperiodic if and only if it is recognized by some aperiodic monoid with at most 3^{n^2} elements. Hence, we can guess a recognizing morphism h from Σ^* to an aperiodic monoid M of size at most 3^{n^2}, guess a set P of linked pairs, compute a Büchi automaton $\mathcal{A}'$ recognizing $L' = \bigcup_{(s,e) \in P} h^{-1}(s) h^{-1}(e)^\omega$ using Corollary 11.10, and finally check whether $L = L'$ starting from $\mathcal{A}, \mathcal{A}'$ and using complementations, intersections and an emptiness tests.

The complexity of this algorithm is not optimal. In order to derive a PSPACE algorithm, we first establish the following characterization.

Proposition 12.2. Let $h : \Sigma^* \to M$ be a surjective morphism that recognizes $L \subseteq \Sigma^\infty$. Let $g : M \to M'$ be a surjective morphism. Then, $h' = g \circ h$

recognizes L if and only if for all $s, e, s', e' \in M$ such that $g(s) = g(s')$ and $g(e) = g(e')$ we have

$$h^{-1}(s)h^{-1}(e)^\omega \subseteq L \quad \Longleftrightarrow \quad h^{-1}(s')h^{-1}(e')^\omega \subseteq L$$

Intuitively, this means that the set of linked pairs associated with L is saturated by g.

Proof. Assume first that h' recognizes L. Let $s, e, s', e' \in M$ with $g(s) = g(s')$ and $g(e) = g(e')$ and assume that $h^{-1}(s)h^{-1}(e)^\omega \subseteq L$. Since h is surjective, we find $u, v, u', v' \in \Sigma^*$ such that $h(u) = s$, $h(v) = e$, $h(u') = s'$ and $h(v') = e'$. From the hypothesis, we get $h'(u) = h'(u')$ and $h'(v) = h'(v')$. Now, $uv^\omega \in h^{-1}(s)h^{-1}(e)^\omega \subseteq L$. Since h' recognizes L we deduce $u'v'^\omega \in h^{-1}(s')h^{-1}(e')^\omega \cap L$. Since h recognizes L we obtain $h^{-1}(s')h^{-1}(e')^\omega \subseteq L$.

Conversely, let $u = u_0u_1u_2 \cdots \in L$ and $v = v_0v_1v_2 \cdots$ with $u_i, v_i \in \Sigma^+$ and $h'(u_i) = h'(v_i)$ for all $i \geq 0$. We have to show that $v \in L$. Grouping factors u_i and v_i using Lemma 5.2, we find new factorizations $u = u'_0u'_1u'_2 \cdots$ and $v = v'_0v'_1v'_2 \cdots$ which satisfy in addition $h(u'_i) = e$ and $h(v'_i) = e'$ for all $i > 0$. Let $s = h(u'_0)$ and $s' = h(v'_0)$. We have $g(s) = h'(u'_0) = h'(v'_0) = g(s')$ and similarly $g(e) = g(e')$. Now, $u \in h^{-1}(s)h^{-1}(e)^\omega \cap L \neq \varnothing$ and since h recognizes L we get $h^{-1}(s)h^{-1}(e)^\omega \subseteq L$. We deduce that $v \in h^{-1}(s')h^{-1}(e')^\omega \subseteq L$. Q.E.D.

Proposition 12.3. We can decide in PSPACE whether the accepted language $L \subseteq \Sigma^\infty$ of a given Büchi automaton $\mathcal{A}$ is aperiodic.

Proof. Let $h : \Sigma^* \to K^{n\times n}$ be Büchi's morphism and let $M = BT(\mathcal{A}) = h(\Sigma^*)$ so that $h : \Sigma^* \to M$ is surjective and recognizes $L = \mathcal{L}(\mathcal{A})$. Let g be the canonical morphism from M to the quotient M' of M by the congruence generated by $\{x^m = x^{m+1} \mid x \in M\}$ with $m = 3^{n^2} \geq |M|$.

It is enough to design a non-deterministic polynomial space algorithm which finds out that L is not aperiodic. By Propositions 12.1 and 12.2, we have to check whether there exist four elements $s, e, s', e' \in M$ such that $g(s) = g(s')$, $g(e) = g(e')$, $h^{-1}(s)h^{-1}(e)^\omega \subseteq L$ and $h^{-1}(s')h^{-1}(e')^\omega \nsubseteq L$. By definition of M', this is equivalent to the existence of $u, v, w, x, y, z \in M$ and $\varepsilon_1, \varepsilon_2, \varepsilon_3, \varepsilon_4 \in \{0, 1\}$ with $h^{-1}(s)h^{-1}(e)^\omega \subseteq L$ and $h^{-1}(s')h^{-1}(e')^\omega \nsubseteq L$ where $s = uv^{m+\varepsilon_1}w$, $e = xy^{m+\varepsilon_2}z$, $s' = uv^{m+\varepsilon_3}w$ and $e' = xy^{m+\varepsilon_4}z$.

We have $h^{-1}(s)h^{-1}(e)^\omega \subseteq L$ if and only if there are $p \in I$, $q \in Q$ such that $(se^k)[p, q] \geq 1$ and $e^\ell[q, q] = \infty$ for some $k, \ell \leq n$. Indeed, if the right hand side holds then we find an accepting run in $\mathcal{A}$ for some word $u \in h^{-1}(s)h^{-1}(e)^\omega$. Hence, we have $h^{-1}(s)h^{-1}(e)^\omega \cap L \neq \varnothing$ and since h recognizes L we deduce that $h^{-1}(s)h^{-1}(e)^\omega \subseteq L$. Conversely, let $u = u_0u_1u_2 \ldots \in L$ with $h(u_0) = s$ and $h(u_i) = e$ for $i > 0$. Consider an accepting run for u:

$$p \xrightarrow{u_0} q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} q_2 \cdots$$

Since this run is accepting, we find k such that a repeated state is visited in the path $q_k \xrightarrow{u_{k+1}} q_{k+1}$ and $q_k = q_{k+\ell}$ for some $\ell > 0$. Removing loops we may assume that $k < n$ and $\ell \leq n$. We get the result with $q = q_k$.

Therefore, we have the following algorithm.

1. Guess six matrices $u, v, w, x, y, z \in M$ and guess four values ε_1, ε_2, ε_3, ε_4 in $\{0, 1\}$ (with, if one wishes, $\varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_4 = 1$).
2. Compute $s = uv^{m+\varepsilon_1}w$, $e = xy^{m+\varepsilon_2}z$, $s' = uv^{m+\varepsilon_3}w$ and $e' = xy^{m+\varepsilon_4}z$.
3. Check that $h^{-1}(s)h^{-1}(e)^\omega \subseteq L$ and $h^{-1}(s')h^{-1}(e')^\omega \nsubseteq L$.

Computing x^m with $x \in M$ can be done with $\mathcal{O}(\log m) = \mathcal{O}(n^2)$ products of $n \times n$ matrices. Hence, steps 2 and 3 can be done in deterministic polynomial time, once the matrices $u, v, w, x, y, z \in M$ are known. It remains to explain how to guess in PSPACE an element $x \in M = h(\Sigma^*)$. As a matter of fact, it is here[2] where we need the full computational power of PSPACE. To do this, we guess a sequence $a_1, a_2, \ldots a_i \in \Sigma$ letter after letters and simultaneously we compute the sequence

$$h(a_1), h(a_1a_2), \ldots, h(a_1a_2 \cdots a_i).$$

We remember only the last element $h(a_1a_2 \cdots a_j)$ before we guess the next letter a_{j+1} and compute the next matrix. We stop with some $i \leq 3^{n^2}$ and we let $x = h(a_1a_2 \cdots a_i)$ be the last computed matrix. Q.E.D.

In some cases it is extremely easy to see that a language is not aperiodic. For example, $(aa)^*$ is recognized by the cyclic group $\mathbb{Z}/2\mathbb{Z}$ of two elements. Every aperiodic quotient of a group is trivial. But the trivial monoid cannot recognize $(aa)^*$.

13 Very weak alternating automata

For a finite set Q we mean by $\mathbb{B}^+(Q)$ the *non-empty positive* Boolean combinations of elements of Q, e.g., $p \wedge (q \vee r)$. We write $P \models \xi$, if a subset $P \subseteq Q$ *satisfies* a formula $\xi \in \mathbb{B}^+(Q)$. By definition, $P \models p$ if and only if $p \in P$. As a consequence, we have for instance $\{p, r\} \models p \wedge (q \vee r)$ and $\{p, r, s\} \models p \wedge (q \vee r)$, but $\{q, r\} \not\models p \wedge (q \vee r)$. Note that $\varnothing \not\models \xi$ since we use non-empty positive Boolean combinations, only. The *satisfiability relation* is monotone. This means, if $P \subseteq P'$ and $P \models \xi$, then $P' \models \xi$, too.

An *alternating* automaton is a tuple $\mathcal{A} = (Q, \Sigma, \delta, I, F, R)$ where

[2] For the interested reader, the test $x \in h(\Sigma^*)$ is PSPACE-hard, in general [10, Problem MS5]. This problem is closely related to the intersection problem of regular languages, where the PSPACE–hardness is due to Kozen [14].

- Q is a finite set of states,
- Σ is a finite alphabet,
- $I \in \mathbb{B}^+(Q)$ is the (alternating) initial condition,
- $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ is the (alternating) transition function (for instance, $\delta(p,a) = (p \wedge (q \vee r)) \vee (q \wedge s)$ is a possible transition),
- $F \subseteq Q$ is the subset of *final* states,
- and $R \subseteq Q$ is the subset of *repeated* states.

A run of $\mathcal{A}$ over some word $w = a_0a_1a_2 \cdots \in \Sigma^\infty$ is a Q-labeled forest (V, E, ρ) with $E \subseteq V \times V$ and $\rho : V \to Q$ such that

- the set of roots $\{z \mid E^{-1}(z) = \varnothing\}$ satisfy the initial condition:
$$\rho(\{z \mid E^{-1}(z) = \varnothing\}) \models I,$$
- each node satisfies the transition relation: for all $x \in V$ of depth n, i.e., such that $x \in E^n(z)$ where $z \in V$ is the root ancestor of x, we have $n \leq |w|$ and if $n < |w|$ then x is not a leaf and $\rho(E(x)) \models \delta(\rho(x), a_n)$.

If the word w is finite then the run is *accepting*, if each leaf x satisfies $\rho(x) \in F$. If the word w is infinite then the run is *accepting*, if every infinite branch visits R infinitely often. Since we use *nonempty* boolean combinations of states for the transition function, if w is finite then each leaf must be of depth $|w|$ and if w is infinite then each maximal branch must be infinite. We denote by $\mathcal{L}(\mathcal{A})$ the set of words $w \in \Sigma^\infty$ for which there is some accepting run of $\mathcal{A}$.

An alternating automaton $\mathcal{A}$ is called *very weak*, if there is a partial order relation $\leq$ on Q such that the transition function is *non-increasing*, i.e., for each $p, q \in Q$ and $a \in \Sigma$, if q occurs in $\delta(p,a)$ then $q \leq p$. Clearly, we can transform the partial ordering into a linear ordering without changing the condition of being very weak[3]. The next proposition shows that every first-order definable language can be accepted by some very weak automaton. The converse is shown in Proposition 13.3.

Proposition 13.1. For any formula $\xi \in \mathrm{LTL}_\Sigma(\mathsf{XU})$, we can construct a very weak alternating automaton $\mathcal{A}$ over Σ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\xi)$.

[3] In [17] a very weak automaton is therefore called a *linear alternating automaton.*

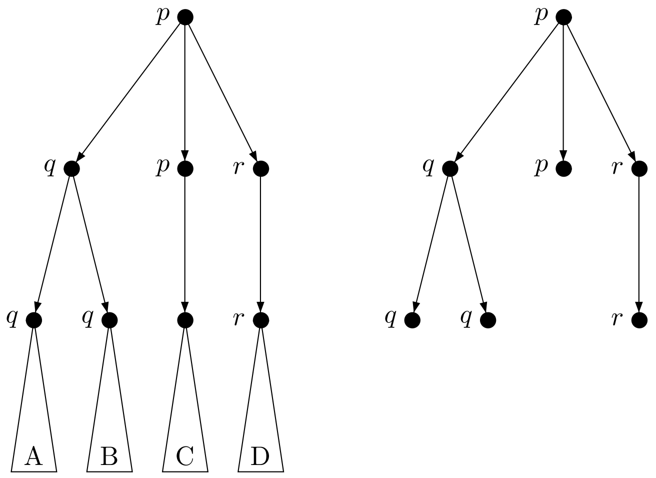


FIGURE 6. A run on the left and on the right the new tree with fresh leaves.

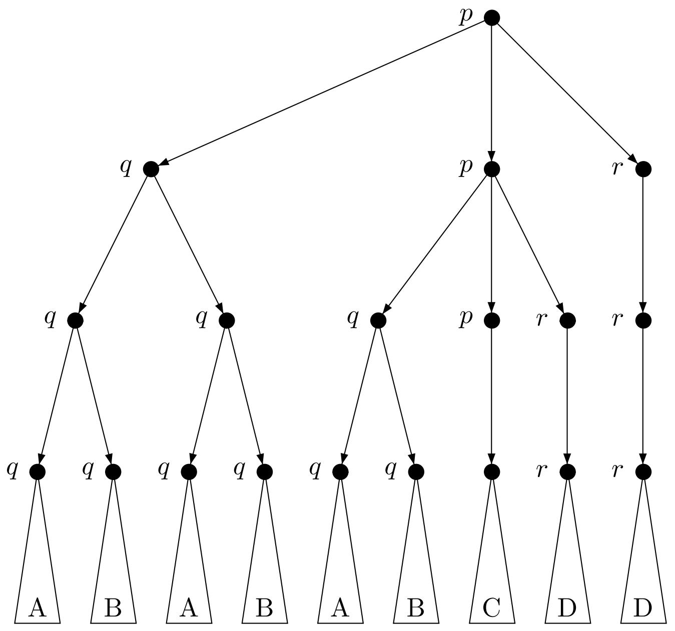


FIGURE 7. The new run with leaves on level $m + 1$.

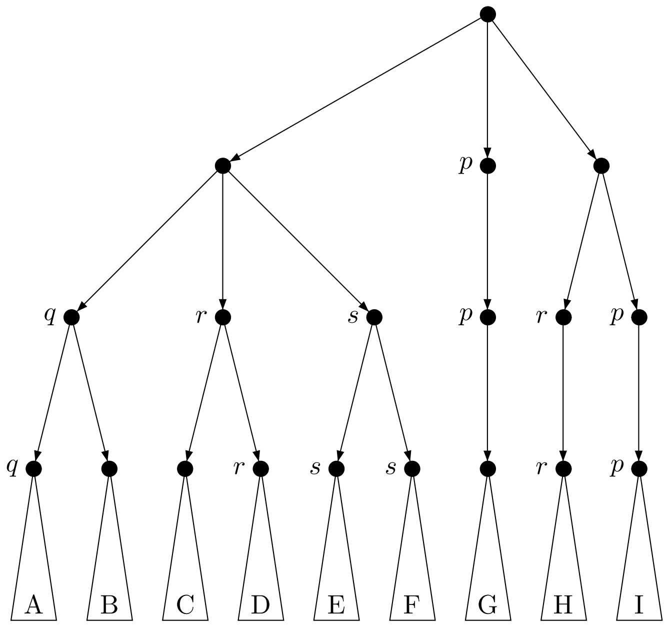


FIGURE 8. Another run with leaves on level $m + 1$.

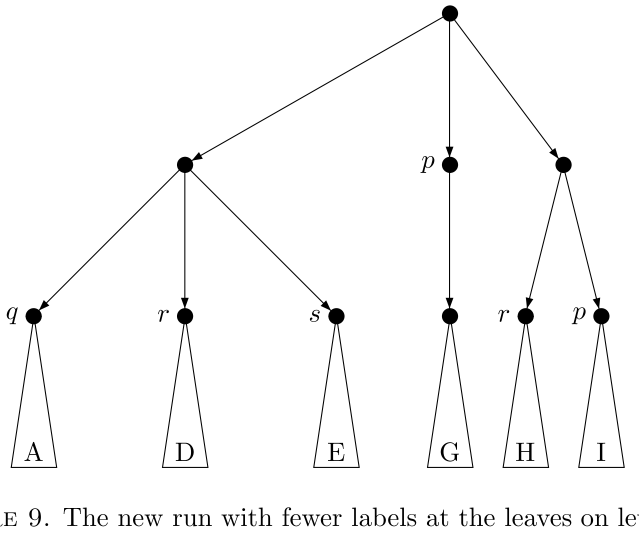


FIGURE 9. The new run with fewer labels at the leaves on level m.

Proof. First, we push the negations down to the constants. For this we need a dual for each operator. Clearly, $\vee$ and $\wedge$ are dual to each other. The dual of next-until is *next-release* which is defined by

$$\varphi \,\mathsf{XR}\, \psi = \neg(\neg\varphi \,\mathsf{XU}\, \neg\psi).$$

Hence, the semantics of *next-release* is given by

$$(\varphi \,\mathsf{XR}\, \psi)(x) = \forall z : \; x < z \rightarrow \psi(z) \vee \exists y : \; x < y < z \wedge \varphi(y).$$

Note that this is always true at the last position of a finite word: for all $v \in \Sigma^+$, we have $v, |v| - 1 \models \varphi \,\mathsf{XR}\, \psi$ for all formulae φ and ψ. One may also notice that

$$\varphi \,\mathsf{XR}\, \psi = \mathsf{X}\,\mathsf{G}\,\psi \vee (\psi \,\mathsf{XU}\, (\varphi \wedge \psi)).$$

All $\mathrm{LTL}_\Sigma(\mathsf{XU})$ formulae can be rewritten in *positive normal form* following the syntax

$$\varphi ::= \bot \mid \top \mid a \mid \neg a \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \,\mathsf{XU}\, \varphi \mid \varphi \,\mathsf{XR}\, \varphi.$$

Transforming a formula into positive normal form does not increase its size, and the number of temporal operators remains unchanged.

So, let ξ be an LTL formula in positive normal form. We define the alternating automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F, R)$ as follows:

- The set Q of states consists of $\bot$, $\top$, END and the sub-formulae of ξ of the form a, $\neg a$, $\varphi \,\mathsf{XU}\, \psi$ or $\varphi \,\mathsf{XR}\, \psi$. Here, END means that we have reached the end of a finite word. Note that each sub-formula of ξ is in $\mathbb{B}^+(Q)$.
- The initial condition is $I = \xi$ itself.
- The transition function is defined by

$$\begin{aligned}
\delta(a, b) &= \begin{cases} \top & \text{if } b = a \\ \bot & \text{otherwise} \end{cases} \\
\delta(\neg a, b) &= \begin{cases} \bot & \text{if } b = a \\ \top & \text{otherwise} \end{cases} \\
\delta(\bot, a) &= \bot \\
\delta(\top, a) &= \top \\
\delta(\varphi \,\mathsf{XU}\, \psi, a) &= \psi \vee (\varphi \wedge \varphi \,\mathsf{XU}\, \psi) \\
\delta(\varphi \,\mathsf{XR}\, \psi, a) &= \mathrm{END} \vee (\psi \wedge (\varphi \vee \varphi \,\mathsf{XR}\, \psi)) \\
\delta(\mathrm{END}, a) &= \bot
\end{aligned}$$

- The set of final states is $F = \{\top, \mathrm{END}\}$.
- The repeated states are the next-release sub-formulae of ξ together with $\top$.

Using the sub-formula partial ordering, we see that the alternating automaton $\mathcal{A}$ is very weak. We can also easily check that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\xi)$. Note that in a run over an infinite word, each infinite branch is ultimately labeled $\top$ or $\bot$ or with a XU or XR formula. A state $\varphi \,\mathsf{XU}\, \psi$ is rejecting since if a branch is ultimately labeled with this state, this means that the eventuality ψ was not checked. On the other hand, $\varphi \,\mathsf{XR}\, \psi$ is accepting since if a branch is ultimately labeled with this state then ψ is ultimately true for this word. Q.E.D.

As we see below, it is easy to transform a very weak alternating automaton into a Büchi automaton. We follow the construction of [11]. However, for this purpose it is convenient to generalize the acceptance conditions. A *generalized Büchi automaton* is a tuple

$$\mathcal{A} = (Q, \Sigma, \delta, I, F, T_1, \ldots, T_r)$$

where Q is a finite set of states, Σ is a finite alphabet,

$$\delta \subseteq Q \times \Sigma \times Q$$

is the non deterministic transition relation, $I \subseteq Q$ is the subset of initial states, $F \subseteq Q$ is the subset of final states, and $T_1, \ldots, T_r \subseteq \delta$ defines the accepting conditions. An infinite run $q_0, a_1, q_1, a_2, q_2, \cdots$ is accepted by $\mathcal{A}$ if for each $1 \le i \le r$, some transition in T_i occurs infinitely often in the run. Hence, the acceptance condition is generalized in two respects. First, it uses accepting *transitions* instead of accepting *states*. Second it allows a conjunction of Büchi's conditions. Obviously, each generalized Büchi automaton can be transformed into an equivalent *classical* Büchi automaton.

From a very weak alternating automaton, we construct an equivalent generalized Büchi automaton as follows. Let $\mathcal{A} = (Q, \Sigma, \delta, I, F, R)$ be a very weak alternating automaton. We define $\mathcal{A}' = (Q', \Sigma, \delta', I', F', (T_f)_{f \notin R})$ by

- $Q' = 2^Q$,
- $I' = \{P \subseteq Q \mid P \models I\}$,
- $(P, a, P') \in \delta'$ if and only if $P' \models \bigwedge_{p \in P} \delta(p, a)$,
- $F' = 2^F$ is the set of final states,

- for each $p \notin R$ we have an accepting condition

$$T_p = \{(P, a, P') \mid p \notin P \text{ or } P' \setminus \{p\} \models \delta(p, a)\}.$$

Proposition 13.2. The automata $\mathcal{A}$ and $\mathcal{A}'$ accept the same language.

The proof that $\mathcal{A}$ and $\mathcal{A}'$ accept the same language is a little bit technical, but not very hard. Details are left to the reader or can be found in [22].

We now state and prove the converse of Proposition 13.1.

Proposition 13.3. Let $L \subseteq \Sigma^\infty$ be accepted by some very weak alternating automaton. Then L is aperiodic.

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, I, F, R)$ be a very weak alternating automaton. For a word u and subsets P and P' of Q we write

$$P \overset{u}{\Longrightarrow} P',$$

if $\mathcal{A}$ has a run (V, E, ρ) over u, where P is the set of labels of the roots and P' is the set of labels of the leaves on level $|u|$. This means that in the corresponding generalized Büchi automaton $\mathcal{A}'$ there is path from state P to state P', which is labeled by the word u.

Let $m = |Q|$, we want to show that $P \overset{u^m}{\Longrightarrow} P'$ if and only if $P \overset{u^{m+1}}{\Longrightarrow} P'$ for all words u and subsets P and P'. This implies that the transformation monoid of $\mathcal{A}'$ is aperiodic. Then, we conclude that languages accepted by very weak alternating automata are always aperiodic in a similar way as in the proof of Proposition 11.11, (because the generalized accepting condition can be easily incorporated in that proof).

First, assume that $P \overset{u^m}{\Longrightarrow} P'$ and let us see that $P \overset{u^{m+1}}{\Longrightarrow} P'$, too. This is true if u is the empty word. Hence we may assume that $|u| \geq 1$. Let (V, E, ρ) be the forest which corresponds to this run. We assume that $P = \{p\}$ and that (V, E, ρ) is tree. This is not essential, but it simplifies the picture a little bit. To simplify the picture further, we assume that $u = a$ is in fact a letter. Formally, we replace E by $E^{|u|}$ and we restrict the new forest to the tree which has the same root as (V, E, ρ). Note that the set of leaves which were on level $|u^m|$ before are now exactly the leaves on level $|m|$. Hence the assumption $u = a$ is justified.

Since $m = |Q|$ we find on each branch from the root to leaves a first node which has the same label as its parent node. This happens because the automaton is very weak and therefore the ordering on the way down never increases. We cut the tree at these nodes and these nodes are called *fresh leaves*. See Figure 6, where the fresh leaves have labels q, q, p, and r from left-to-right.

Now, at each fresh leaf we glue the original sub tree of its parent node. We obtain a new tree of height $m+1$ which has as the set of labels at level $m+1$ exactly the same labels as before the labels at level m in the original tree. (See Figure 7.) It is clear that the new tree is a run over u^{m+1} and thus, $P \overset{u^{m+1}}{\Longrightarrow} P'$ as desired.

For the other direction, assume that $P \overset{u^{m+1}}{\Longrightarrow} P'$ and let (V, E, ρ) be a forest which corresponds to this run. Just as above we may assume that (V, E, ρ) is a tree and that u is a letter. This time we go down from the root to leaves and we cut at the first node, where the node has the same label as one of its children. See Figure 8. Now, we glue at these new leaves the original sub tree of one of its children which has the same label.

We obtain a new tree of height m such that each label at the leaves on level m appeared before as a label on some leaf of the original tree (V, E, ρ) at level $m+1$, see Figure 9.

Thus, $P \overset{u^{m}}{\Longrightarrow} P''$ for some subset $P'' \subseteq P'$. But the satisfiability relation is monotone; therefore $P \overset{u^{m}}{\Longrightarrow} P'$, too. Thus, indeed $P \overset{u^{m}}{\Longrightarrow} P'$ if and only if $P \overset{u^{m+1}}{\Longrightarrow} P'$ for $m = |Q|$. Q.E.D.

References

[1] A. Arnold. A syntactic congruence for rational omega-language. *Theor. Comput. Sci.*, 39:333–335, 1985.

[2] F. Blanchet-Sadri. Some logical characterizations of the dot-depth hierarchy and applications. *J. Comput. System Sci.*, 51(2):324–337, 1995.

[3] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr .)*, pages 1–11, Stanford, Calif., 1962. Stanford Univ. Press.

[4] S. Cho and D. T. Huynh. Finite-automaton aperiodicity is pspace-complete. *Theor. Comput. Sci.*, 88(1):99–116, 1991.

[5] V. Diekert and P. Gastin. Pure future local temporal logics are expressively complete for mazurkiewicz traces. *Inf. Comput.*, 204(11):1597–1619, 2006.

[6] V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1995.

[7] K. Etessami, M. Y. Vardi, and Th. Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

[8] A. Fernández López and M. Tocón Barroso. The local algebras of an associative algebra and their applications. In J. Misra, editor, *Applicable Mathematics in the Golden Age*, pages 254–275, New Delhi, India, 2002. Narosa.

[9] D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal basis of fairness. In *POPL*, pages 163–173, 1980.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[11] P. Gastin and D. Oddoux. Fast ltl to büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

[12] P. Gastin and A. Petit. Infinite traces. In *The book of traces*, pages 393–486. World Sci. Publ., River Edge, NJ, 1995.

[13] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Los Angeles (California), 1968.

[14] D. Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266. IEEE, 1977.

[15] R. E. Ladner. Application of model theoretic games to discrete linear orders and finite automata. *Information and Control*, 33(4):281–303, 1977.

[16] C. Lautemann, P. McKenzie, T. Schwentick, and H. Vollmer. The descriptive complexity approach to logcfl. *J. Comput. Syst. Sci.*, 62(4):629–652, 2001.

[17] C. Löding and W. Thomas. Alternating automata and logics over infinite words. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *IFIP TCS*, volume 1872 of *Lecture Notes in Computer Science*, pages 521–535. Springer, 2000.

[18] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9(5):521–530, 1966.

[19] R. McNaughton and S. Papert. *Counter-free automata.* The M.I.T. Press, Cambridge, Mass.-London, 1971. With an appendix by William Henneman, M.I.T. Research Monograph, No. 65.

[20] K. Meyberg. Lectures on algebras and triple systems, 1972.

[21] D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating automata. the weak monadic theory of the tree, and its complexity. In L. Kott, editor, *ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 275–283. Springer, 1986.

[22] D. Oddoux. *Utilisation des automates alternants pour un model-checking efficace des logiques temporelles linaires.* PhD thesis, Université Paris 7 (France), 2003.

[23] D. Perrin. Recent results on automata and infinite words. In M. Chytil and V. Koubek, editors, *MFCS*, volume 176 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 1984.

[24] D. Perrin and J.-E. Pin. First-order logic and star-free sets. *J. Comput. Syst. Sci.*, 32(3):393–406, 1986.

[25] D. Perrin and J.-E. Pin. *Infinite Words. Automata, Semigroups, Logic and Games*, volume 141 of *Pure and Applied Mathematics.* Elsevier, Amsterdam, 2004.

[26] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[27] S. Rhode. *Alternating automata and the temporal logic of ordinals.* PhD Thesis, University of Illinois, Urbana Campaign II, 1997.

[28] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[29] J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985.

[30] P. Tesson and D. Thérien. Diamonds are forever: the variety DA. In *Semigroups, algorithms, automata and languages (Coimbra, 2001)*, pages 475–499. World Sci. Publ., River Edge, NJ, 2002.

[31] W. Thomas. Star-free regular sets of omega-sequences. *Information and Control*, 42:148–156, 1979.

[32] W. Thomas. A combinatorial approach to the theory of omega-automata. *Information and Control*, 48(3):261–283, 1981.

[33] W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25(3):360–376, 1982.

[34] W. Thomas. An application of the Ehrenfeucht-Fraïssé game in formal language theory. *Mém. Soc. Math. France (N.S.)*, 16:11–21, 1984. Logic (Paris, 1983).

[35] W. Thomas. A concatenation game and the dot-depth hierarchy. In E. Börger, editor, *Computation Theory and Logic*, volume 270 of *Lecture Notes in Computer Science*, pages 415–426. Springer, 1987.

[36] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 133–192. Elsevier, Amsterdam, 1990.

[37] W. Thomas. Languages, automata, and logic. In *Handbook of formal languages, Vol. 3*, pages 389–455. Springer, Berlin, 1997.

[38] Th. Wilke. *Classifying Discrete Temporal Properties.* Habilitationsschrift, Universität Kiel, Apr. 1998.

[39] Th. Wilke. Classifying discrete temporal properties. In *STACS 99 (Trier)*, volume 1563 of *Lecture Notes in Comput. Sci.*, pages 32–46, Berlin, 1999. Springer.

Matrix-based complexity functions and recognizable picture languages*

Dora Giammarresi[1]
Antonio Restivo[2]

[1] Dipartimento di Matematica
Università di Roma “Tor Vergata”
via della Ricerca Scientifica
00133 Roma, Italy
giammarr@mat.uniroma2.it

[2] Dipartimento di Matematica e Applicazioni
Università di Palermo
via Archirafi, 34
90123 Palermo, Italy
restivo@dipmat.math.unipa.it

Abstract

The aim of this paper is to shed new light onto the relations between the complement problem and the unambiguity in the family of recognizable picture languages. It is known that, contrary to the one-dimensional case, the family REC of recognizable picture languages is not closed under complementation and that the family UREC of unambiguous recognizable picture languages is a proper subfamily of REC. The interest to investigate the relations between these two facts was raised by Wolfgang Thomas. In this paper we present a novel general framework to study such a problem, by introducing some complexity functions on pictures languages.

1 Introduction

Picture (two-dimensional) languages were studied using different approaches and perspectives since the sixties as the natural counterpart in two dimensions of (one-dimensional) string languages. In 1991, a unifying point of view was presented in [6] where the family of *tiling recognizable picture languages* is defined (see also [7]). The definition of recognizable picture language takes as starting point a well known characterization of recognizable string languages in terms of local languages and projections. Namely, any recognizable string language can be obtained as projection of a local string language defined over a larger alphabet. Such notion can be extended

* We heartily thank Oliver Matz for his careful readings and suggestions.

in a natural way to the two-dimensional case: more precisely, local picture languages are defined by means of a set of square arrays of side-length two (called *tiles*) that represents the only allowed blocks of that size in the pictures of the language (with special treatment for border symbols). Then, we say that a two-dimensional language is tiling recognizable if it can be obtained as a projection of a local picture language. The family of all tiling recognizable two-dimensional languages is called REC. Remark that, when we consider strings as particular pictures (that is pictures in which one side has length one), this definition of recognizability coincides with the one for the strings, i.e. the definition given in terms of finite automata. Further the definition of class REC turns out to be robust because it inherits most of the important properties from the class of regular string languages (see also [8]). Moreover tiling recognizable picture languages have been considered and appreciated in the picture processing and pattern recognition fields (see [14]). Finally the approach to recognizability in terms of tiling systems is very close to that one proposed by Woflgang Thomas in the more general context of graphs (cf. [11, 17]).

A crucial difference between the recognizability of string languages and the one of picture languages in REC arises directly from its definition. The definition of recognizability in terms of local languages and projections is implicitly non-deterministic (notice that in the one-dimensional case a tiling system corresponds in general to a non-deterministic automaton). This fact is strengthened by another result: the class REC is not closed under complementation. As a consequence, we infer that it is not possible to eliminate the non-determinism from this model without losing in power of recognition (as long as deterministic versions allow complementation). Problems on deterministic tiling systems are considered in [1]. If we denote by co-REC the family of languages whose complement is in REC, we have that REC is strictly included in REC $\cup$ co-REC.

In this scenario, related to the problem of defining a subset of REC closed under complement, *unambiguity* plays a central role as intermediate notion between determinism and non-determinism. As determinism, unambiguity corresponds to the existence of a *unique* process of computation, but while determinism is a "local" notion, unambiguity is a "global" one. Recall that, for regular string languages, the three notions of determinism, non-determinism and unambiguity coincide while in more general structures this is not true (see for instance [13]). *Unambiguous recognizable two-dimensional languages* have been introduced in [6], and their family referred to as UREC. Informally, a picture language belongs to UREC when it admits an unambiguous tiling system, i.e. such that every picture has a unique counter-image in its corresponding local language. In [2] several problems on class UREC are studied and it is proved that UREC is strictly included in REC. Very recently, in [12], unambiguous recognizable picture languages are considered in relation to picture series definable by some weighted logic.

In this paper we present a novel general framework to study properties of recognizable picture languages and then use it to study the relations between classes REC ∪ co-REC, REC and UREC. The strict inclusions among these classes have been proved in [4], [10], [2], respectively, using ad-hoc techniques. Here we propose again those results in a unified formalism and proof method with the major intent of establishing relations between the complement problem and unambiguity in the family of recognizable picture languages. Remark that the interest for such relations was also raised by Wolfgang Thomas in [13].

We introduce some complexity functions on picture languages and combine two main techniques. First, following the approach of O. Matz in [10], we consider, for each positive integer m, the set $L(m)$ of pictures of a language L having one dimension (say the vertical one) of size m. Language $L(m)$ can be viewed as a string language over the alphabet (of the columns) $\Sigma^{m,1}$. The idea is then to measure the complexity of the picture language L by evaluating the grow rate, with respect to m, of some numerical parameters of $L(m)$. In order to specify such numerical parameters we make use, as a second technique, of the Hankel matrix of a string language. The parameters are indeed expressed in terms of some elementary matrix-theoretic notions of the Hankel matrices of the string languages $L(m)$. In particular, we consider here three parameters: the number of different rows, the rank, and the maximal size of a permutation submatrix.

We prove three main theorems that establish some bounds on corresponding complexity functions based on those three parameters, respectively. Then, as applications for those bounds we analyze the complexity functions of some examples of picture languages. Interestingly the languages we propose have quite similar definitions based on a combination of the existence or non-existence of duplicate columns in the pictures either for a single column or for all the columns. By means of those languages we re-prove the strict inclusions of families REC ∪ co-REC, REC and UREC.

Moreover we show an example of a language in REC that does not belong to UREC and whose complement is not in REC. This language introduces further discussions on relations between unambiguity and non-closure under complement.

The paper is organized as follows. We start, in Section 2, by introducing some basic two-dimensional languages terminology and definitions and recalling the technique due to O. Matz to reduce a picture language to a family of string languages on the columns alphabets. Then in Section 3 we introduce our novel technique by defining complexity functions based on

Hankel matrices. In Section 4 we recall all definitions and properties of the family REC of tiling recognizable picture languages. Our main results are proved in Section 5 while in Section 6 we apply them to some picture languages in order to establish some separation results. Finally, in Section 7 we discuss some further directions for the introduced techniques and propose some related questions. For sake of completeness, we report here most of proofs of the results we cite.

2 Picture languages

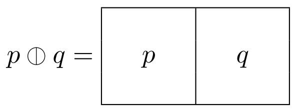
In this section we introduce some definitions about two-dimensional languages and their operations. More notations and definitions can be found in [7].

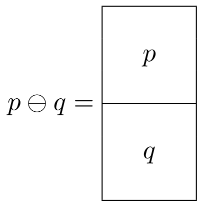
Let Σ be a finite alphabet. A *picture* (or *two-dimensional string*) over Σ is a two-dimensional rectangular array of elements of Σ. Given a picture p, let $p(i,j)$ denote the symbol in p with coordinates (i,j), moreover the *size* of p is given by a pair (m,n) where m and n are the number of rows and columns of p, respectively. The set of all pictures over Σ of size (x,y) for all $x,y \geq 1$ is denoted by Σ^{++} and a *picture (two-dimensional) language* over Σ is a subset of Σ^{++}. Remark that in this paper we do not consider the case of empty pictures (i.e. pictures where the number of rows and/or columns can be zero). The set of all pictures over Σ of fixed size (m,n), with $m,n \geq 1$ is denoted by $\Sigma^{m,n}$. We give a first example of a picture language.

Example 2.1. Let L be the language of square pictures over an alphabet Σ, that is:

$$L = \{\, p \mid p \text{ has size } (n,n),\ n > 0 \,\}.$$

We now recall the classical concatenation operations between pictures and picture languages. Let p and q be two pictures over an alphabet Σ, of size (m,n) and (m',n') with $m,n,m',n' > 0$, respectively. The *column concatenation* of p and q (denoted by $p \varobar q$) and the *row concatenation* of p and q (denoted by $p \ominus q$) are partial operations, defined only if $m = m'$ and if $n = n'$, respectively and are given by:





As done in the string language theory, these definitions of picture concatenations can be extended to define two-dimensional language concatenations. If

L_1, L_2 are picture languages over an alphabet Σ, the *column concatenation* of L_1 and L_2 is defined by

$$L_1 \mathbin{⦶} L_2 = \{x \mathbin{⦶} y \mid x \in L_1 \text{ and } y \in L_2\}$$

Similarly, the *row concatenation* of L_1 and L_2 is defined by

$$L_1 \ominus L_2 = \{x \ominus y \mid x \in L_1 \text{ and } y \in L_2\}$$

Furthermore, by iterating the concatenation operations, we obtain the column and row *closure* or *star*. More precisely: the *column closure* of L (denoted by $L^{*⦶}$) and the *row closure* of L (denoted by $L^{*\ominus}$) are defined respectively as

$$L^{*⦶} = \bigcup_i L^{i⦶} \quad \text{and} \quad L^{*\ominus} = \bigcup_i L^{i\ominus}$$

where $L^{1⦶} = L$, $L^{n⦶} = L^{(n-1)⦶} \mathbin{⦶} L$ and $L^{1\ominus} = L$, $L^{n\ominus} = L^{(n-1)\ominus} \ominus L$.

We conclude this section by describing a technique, introduced by O. Matz in [10], that associates to a given picture language L an infinite sequence $(L(m))_{m\geq 1}$ of string languages. Let $L \subseteq \Sigma^{++}$ be a picture language. For any $m \geq 1$, we consider the subset $L(m) \subseteq L$ containing all pictures with exactly m rows. Such language $L(m)$ can be viewed as a string language over the alphabet $\Sigma^{m,1}$ of the columns, i.e. words in $L(m)$ have a "fixed height m". For example, if

$$p = \boxed{\begin{array}{ccccc} a & b & b & a & a \\ a & a & b & b & a \\ b & b & a & b & a \\ a & a & a & a & b \end{array}} \in L$$

then the word

$$w = \begin{bmatrix} a \\ a \\ b \\ a \end{bmatrix} \begin{bmatrix} b \\ a \\ b \\ a \end{bmatrix} \begin{bmatrix} b \\ b \\ a \\ a \end{bmatrix} \begin{bmatrix} a \\ b \\ b \\ a \end{bmatrix} \begin{bmatrix} a \\ b \\ b \\ a \end{bmatrix} \begin{bmatrix} a \\ a \\ a \\ b \end{bmatrix}$$

belongs to the string language $L(4)$ over the alphabet of columns

$$\Sigma^{4,1} = \left\{ \begin{bmatrix} x \\ y \\ s \\ t \end{bmatrix} \mid x, y, s, t \in \Sigma \right\}.$$

Observe that studying the sequence $(L(m))_{m\geq 1}$ of string languages corresponding to a picture languages L does not capture the whole structure of L because in some sense it takes into account only its horizontal dimension. Nevertheless it will be very useful to state some conditions for the recognizability of the picture language L.

3 Hankel matrices and complexity functions

In this section we introduce a novel tool to study picture languages based on combining two main techniques: the Matz's technique described above (that associates to a given picture language L an infinite sequence $(L(m))_{m\geq 1}$ of string languages) and the technique that describes a string language by means of its Hankel matrix. As results there will be the definitions of some complexity functions for picture languages that will be used to state some necessary conditions on recognizable picture languages.

Hankel matrices were firstly introduced in [16] in the context of formal power series (see also [3] and [15]). Moreover they are used under different name in communication complexity (see [9]).

Definition 3.1. Let $S \subseteq A^*$ be a string language. The Hankel matrix of S is the infinite boolean matrix $H_S = [h_{xy}]_{x\in A^*, y\in A^*}$ where

$$h_{xy} = \begin{cases} 1 & \text{if } xy \in S \\ 0 & \text{if } xy \notin S. \end{cases}$$

Therefore both the rows and the columns of H_S are indexed by the set of strings in A^* and the $1s$ in the matrix gives the description of language S in the way described above.

Given an Hankel matrix H_S, we call *submatrix of* H_S a matrix K_S specified by a pair of languages (U, V), with $U, V \subseteq A^*$, that is obtained by intersecting all rows and all columns of H_S that are indexed by the strings in U and V, respectively. Moreover, given two Hankel submatrices K_S^1 and K_S^2, their intersection is the submatrix specified by the intersections of the corresponding index sets respectively.

Moreover we recall some further notations on matrices. A *permutation* matrix is a boolean matrix that has exactly one 1 in each row and in each column. Usually when dealing with permutation matrices, one makes a correspondence between a permutation matrix $D = [d_{ij}]$ of size n with a permutation function $\sigma = I\!N \longrightarrow I\!N$ by assuming that $d_{ij} = 1 \Leftrightarrow j = \sigma(i)$.

Finally we recall that the *rank* of a matrix is the size of the biggest submatrix with non-null determinant (with respect to field $\mathbb{Z}$). Alternatively, the rank is defined as the maximum number of row or columns that are linearly independent. Then, observe that, by definition, the rank of a permutation matrix coincides with its size.

Given a picture language L over the alphabet Σ, we can associate to L an infinite sequence $(H_L(m))_{m\geq 1}$ of matrices, where each $H_L(m)$ is the Hankel matrix of string language $L(m)$ associated to L.

We can define the following functions from the set of natural numbers $\mathbb{N}$ to $\mathbb{N} \cup \infty$.

Definition 3.2. Let L be a picture language.

i) The *row complexity function* $R_L(m)$ gives the number of distinct rows of the matrix $H_L(m)$;

ii) The *permutation complexity function* $P_L(m)$ gives the size of the maximal permutation matrix that is a submatrix of $H_L(m)$;

iii) The *rank complexity function* $K_L(m)$ gives the rank of the matrix $H_L(m)$.

Notice the all the functions $R_L(m)$, $P_L(m)$ and $K_L(m)$ defined above are independent from the order of the rows (columns, resp.) of the Hankel matrix $H_L(m)$. In the sequel we shall use any convenient order for the set of strings that index the rows and the columns. We can immediately state the following lemma.

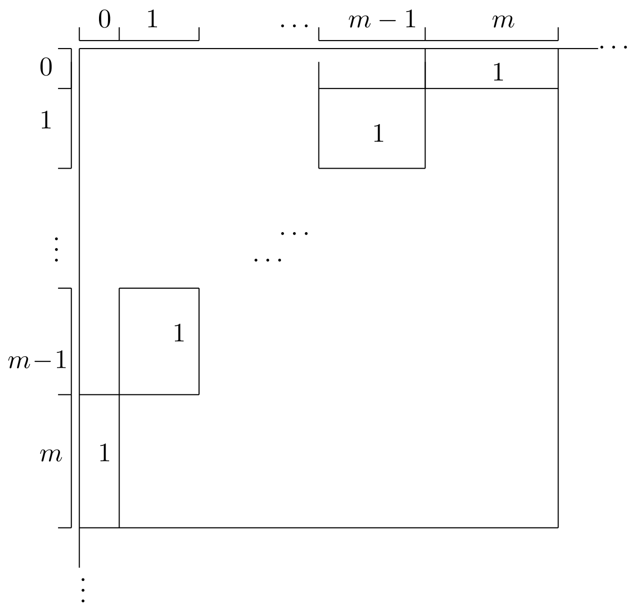
Lemma 3.3. Given a picture language L, for each $m \in \mathbb{N}$:

$$P_L(m) \leq K_L(m) \leq R_L(m).$$

Proof. The rank of a matrix is the size of the biggest submatrix whose rows are linearly independent and therefore the rank is greater than or equal to the size of any permutational submatrix (recall that the rank of a permutational matrix is equal to its size).

Moreover if two rows are linearly independent they should be different and therefore $K_L(m) \leq R_L(m)$. Q.E.D.

Example 3.4. Consider the language L of squares over a two-letters alphabet $\Sigma = \{a, b\}$ described in Example 2.1. Observe that, for each $m \geq 0$, $L(m)$ is the finite language of all possible strings of length m over the alphabet of the columns $\Sigma^{m,1}$. Then consider the Hankel matrix of $L(m)$: it has all its 1s in the positions indexed by pairs (x, y) of strings such that $|x| + |y| = m$. Now assume that the strings that index the rows and the columns of the Hankel matrix are ordered by length: we can have some nonzero positions only in the upper-right portion of $H_L(m)$ that are indexed by all possible strings of length $\leq m$ on the alphabet $\Sigma^{m,1}$, included the empty word. More specifically, in this portion the matrix $H_L(m)$ has all 0s with the exception of a chain of rectangles of all 1s from the top-right to the bottom left corner. This is represented in the following figure where the numbers $0, 1, \ldots, m-1, m$ indicate the length of the index words.



It is easy to verify that the number of different rows in $H_L(m)$ is equal to $m+1$ and this is also the number of rows of a permutation submatrix and this is also the rank of $H_L(m)$.

Then for this language it holds that for all positive m:

$$P_L(m) = K_L(m) = R_L(m) = m + 1.$$

Example 3.5. As generalization of the above Example 3.4, consider the language L of pictures over an alphabet Σ of size $(n, f(n))$ where $f(n)$ is a non-negative function defined on the set of natural numbers, that is:

$$L = \{\, p \,|\, p \text{ is of size } (n, f(n)\}.$$

Similar arguments as in the above example show that, for each $m \geq 0$, language $L(m)$ is a finite language (it contains all strings of length $f(m)$ over the alphabet of the columns $\Sigma^{m,1}$) and then, for all positive m: $P_L(m) = K_L(m) = R_L(m) = f(m) + 1$.

Example 3.6. Consider the language L of pictures over an alphabet Σ of size $(n, 2n)$ such that the two square halves are equal, that is:

$$L = \{\, p \oplus p \,|\, p \text{ is a square}\}.$$

Again, as in the Example 3.4, for each $m \geq 0$, language $L(m)$ is a finite language (it contains all strings of length $2m$ over the alphabet of the columns $\Sigma^{m,1}$ of the form ww). Then, doing all the calculations, one obtains that, for all positive m, $P_L(m)$, $K_L(m)$ and $R_L(m)$ are all of the same order of complexity $O(\sigma^{m^2})$, where σ is the number of symbols in the alphabet Σ.

4 Recognizable picture languages

In this section we recall definitions and basic properties of tiling recognizable two-dimensional languages firstly introduced in 1992 in [6]. We recall the definition of *local* and *recognizable* picture languages and the corresponding family LOC and REC. We state and discuss closure properties of REC under concatenations and Boolean operations. Furthermore, we give the definition of unambiguous recognizable picture languages and of class UREC. The notations used together with all the results and proofs mentioned here can be found in [7].

In order to describe scanning or recognizing strategies for pictures, it is needed to identify the symbols on the boundary. Then, for any picture p of size (m, n), we consider picture $\widehat{p}$ of size $(m+2, n+2)$ obtained by surrounding p with a special *boundary symbol* $\# \notin \Sigma$. We call *tile* a square picture of dimension $(2, 2)$ and given a picture p we denote by $B_{2,2}(p)$ the set of all blocks of p of size $(2, 2)$.

Let Γ be a finite alphabet. A two-dimensional language $L \subseteq \Gamma^{++}$ is *local* if there exists a finite set Θ of tiles over the alphabet $\Gamma \cup \{\#\}$ such that $L = \{x \in \Gamma^{++} \mid B_{2,2}(\widehat{x}) \subseteq \Theta\}$. We shall write $L = L(\Theta)$. Therefore tiles in Θ represent all the *allowed blocks* of size $(2, 2)$ for the pictures in L. The family of local picture languages will be denoted by LOC. We now give an example of a local two-dimensional language.

Example 4.1. Let $\Gamma = \{0, 1\}$ be an alphabet and let Θ be the following set of tiles over Γ.

$$\Theta = \left\{ \begin{array}{cccccc}
\begin{array}{|c|c|}\hline 0 & \# \\ \hline 1 & \# \\ \hline\end{array} &
\begin{array}{|c|c|}\hline 0 & \# \\ \hline 0 & \# \\ \hline\end{array} &
\begin{array}{|c|c|}\hline \# & \# \\ \hline 0 & 0 \\ \hline\end{array} &
\begin{array}{|c|c|}\hline \# & \# \\ \hline 0 & 1 \\ \hline\end{array} &
\begin{array}{|c|c|}\hline \# & \# \\ \hline \# & 1 \\ \hline\end{array} &
\begin{array}{|c|c|}\hline \# & \# \\ \hline 0 & \# \\ \hline\end{array} \\[2em]
\begin{array}{|c|c|}\hline \# & 1 \\ \hline \# & 0 \\ \hline\end{array} &
\begin{array}{|c|c|}\hline \# & 0 \\ \hline \# & 0 \\ \hline\end{array} &
\begin{array}{|c|c|}\hline 0 & 0 \\ \hline \# & \# \\ \hline\end{array} &
\begin{array}{|c|c|}\hline 0 & 1 \\ \hline \# & \# \\ \hline\end{array} &
\begin{array}{|c|c|}\hline \# & 0 \\ \hline \# & \# \\ \hline\end{array} &
\begin{array}{|c|c|}\hline 1 & \# \\ \hline \# & \# \\ \hline\end{array} \\[2em]
\begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & 1 \\ \hline\end{array} &
\begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 1 \\ \hline\end{array} &
\begin{array}{|c|c|}\hline 0 & 1 \\ \hline 0 & 0 \\ \hline\end{array} &
\begin{array}{|c|c|}\hline 0 & 0 \\ \hline 0 & 0 \\ \hline\end{array} & &
\end{array} \right\}$$

The language $L(\Theta)$ is the language of square pictures (i.e. pictures of size (n, n) with $n \geq 2$) in which all diagonal positions (i.e. those of the form (i, i))

carry symbol 1, whereas the remaining positions carry symbol 0. That is, pictures as the following:

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Notice that the language of squares over a one-letter alphabet is not a local language because there is no "local strategy" to compare the number of rows and columns using only one symbol.

Let Γ and Σ be two finite alphabets. A mapping $\pi : \Gamma \to \Sigma$ will be in the sequel called *projection.* The projection $\pi(p)$ of $p \in \Gamma^{++}$ of size (m, n) is the picture $p' \in \Sigma^{++}$ such that $p'(i, j) = \pi(p(i, j))$ for all $1 \leq i \leq m$, $1 \leq j \leq n$. Similarly, if $L \subseteq \Gamma^{++}$ is a picture language over Γ, we indicate by $\pi(L)$ the projection of language L, i.e. $\pi(L) = \{p' | p' = \pi(p),\, p \in L\} \subseteq \Sigma^{++}$.

A quadruple $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ is called *tiling system* if Σ and Γ are finite alphabets, Θ is a finite set of tiles over $\Gamma \cup \{\#\}$ and $\pi : \Gamma \to \Sigma$ is a projection. Therefore, a tiling system is composed by a local language over Γ (defined by the set Θ) and a projection $\pi : \Gamma \longrightarrow \Sigma$. A two-dimensional language $L \subseteq \Sigma^{++}$ is *tiling recognizable* if there exists a tiling system $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ such that $L = \pi(L(\Theta))$. Moreover, we shall refer to $L' = L(\Theta)$ as an *underling local language* for L and to Γ as a *local alphabet* for L. Let $p \in L$, if $p' \in L'$ is such that $\pi(p') = p$, we refer to p' as a *counter-image* of p in the underling local language L'.

The family of all two-dimensional languages that are *tiling recognizable* is denoted by REC. We give here some examples to which we shall refer in the sequel.

Example 4.2. Let L be the language of square pictures (i.e. pictures of size (n, n)) over one-letter alphabet $\Sigma = \{a\}$. To show that language L is in REC we remark that it can be obtained as projection of language in Example 4.1 by mean of projection $\pi(0) = \pi(1) = a$.

Example 4.3. Let L be the language of pictures p whose first column is equal to the last one. We have that $L \in$ REC. Indeed we can define a tiling system where the information on each letter of the first column of p is brought along horizontal direction, using some subscripts, to the last column of p. More precisely, we use a local alphabet $\Gamma = \{x_y \mid x, y \in \Sigma\}$ with $x, y \in \Sigma$ (the subscripts y are used to recall the symbols in the first column of a picture), the projection $\pi(x_y) = x$. The set of tiles is such that, for $p \in L$ and some i, j we have that if $p_{i,1} = y$ and $p_{i,j} = x$ then $p'_{i,j} = x_y$

with $\pi(p') = p$. The tiles of the left border must be of the form $\begin{array}{|c|c|}\hline \# & z_z \\ \hline \# & t_t \\ \hline \end{array}$, the tiles of the right border must be of the form $\begin{array}{|c|c|}\hline z_z & \# \\ \hline t_t & \# \\ \hline \end{array}$, whereas the "middle tiles" must be of the form $\begin{array}{|c|c|}\hline z_z & s_z \\ \hline t_t & r_t \\ \hline \end{array}$. Here below it is given an example of a picture $p \in L \subseteq \{a, b\}^*$ together with a corresponding local picture p'.

$$p = \begin{array}{|c|c|c|c|c|}\hline b & b & a & b & b \\ \hline a & a & b & a & a \\ \hline b & a & a & a & b \\ \hline a & b & b & b & a \\ \hline a & b & b & b & a \\ \hline \end{array} \qquad p' = \begin{array}{|c|c|c|c|c|}\hline b_b & b_b & a_b & b_b & b_b \\ \hline a_a & a_a & b_a & a_a & a_a \\ \hline b_b & a_b & a_b & a_b & b_b \\ \hline a_a & b_a & b_a & b_a & a_a \\ \hline a_a & b_a & b_a & b_a & a_a \\ \hline \end{array}$$

We remark that a tiling system $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ for a picture language is in some sense a generalization to the two-dimensional case of an automaton that recognizes a string language. Indeed, in one-dimensional case, the quadruple $(\Sigma, \Gamma, \Theta, \pi)$ corresponds exactly to the state-graph of the automaton: the alphabet Γ is in a one-to-one correspondence with the edges, the set Θ describes the edges adjacency, the mapping π gives the labelling of the edges in the automaton. Then, the set of words of the underlying local language defined by set Θ corresponds to all accepting paths in the state-graph and its projection by π gives the language recognized by the automaton. As consequence, when rectangles degenerate in strings the definition of recognizability coincides with the classical one for strings (cf. [5]).

4.1 Closure properties of family REC

The family REC is closed with respect to different types of operations. We recall the following theorems without proof (the interested reader can consult [7]).

Theorem 4.4. The family REC is closed under alphabetic projection.

Theorem 4.5. The family REC is closed under row and column concatenation and under row and column stars operations.

Theorem 4.6. The family REC is closed under union and intersection.

As immediate application of this closure properties we have that, as we do in the string case, we can define recognizable languages by means of *picture regular expressions* starting from finite languages containing a single picture of one symbol and using operations of union, intersection, row and column concatenations and closures and projection. We see this in the following examples.

Example 4.7. Let us consider again language L in Example 4.3 of pictures such that the first column is equal to the last one. We showed that $L \in$ REC by giving explicitly a tiling system for it. It is also easy to show that L can be obtained by using concatenations and star operations as follows:

$$L = \bigcup_{a \in \Sigma} (a \oplus \Sigma^{*\oplus} \oplus a))^{*\ominus}$$

where a denotes the size $(1,1)$ picture containing symbol a.

Example 4.8. Let L be the language of pictures p of size (m,n) with the property " $\exists 1 \leq i,j \leq n$ such that the i-th column of p is equal to the j-th column of p". Observe that

$$L = \Sigma^{++} \oplus L' \oplus \Sigma^{++}$$

where L' is the language of pictures with the first column equal to the last one given the the above Example 4.3.

Given two string languages $S, T \subseteq \Sigma^*$, we define the *row-column combination* of S, T to be a picture language $L = S \oplus T$ as the set of all pictures p such that all rows of p belongs to language S and all columns of p belongs to language T. Notice that we can write $L = S^{\ominus *} \cap T^{\oplus *}$, then, as consequence of above closure properties, it holds the following corollary.

Corollary 4.9. If $S, T \subseteq \Sigma^*$ are recognizable string languages then picture language $L = S \oplus T \in$ REC.

We use this result in the next example.

Example 4.10. Let L be the language of pictures p over an alphabet Σ of size (m,n) with the property "$\exists 1 \leq i \leq n$ such that the i-th column of p is different from all the other columns of p". We show that L is in REC.

It is convenient to define a new alphabet Δ. Assume that Σ has σ symbols. Then, for each $s \in \Sigma$ we define a new alphabet Σ_s obtained by adding a subscript s to each element of Σ and define a new alphabet $\Delta = \bigcup_{s \in \Sigma} \Sigma_s$. Let $x, s \in \Sigma$.

We now consider two string languages $L(h)$ and $L(v)$ over the alphabet $\Sigma \cup \Delta$:

$$L(h) = \bigcup_{s \in \Sigma} \Sigma_s^* s \Sigma_s^*$$

$$L(v) = \Sigma^* \cup \left(\bigcup_{x \neq s} (\Delta^* x_s \Delta^*) \right).$$

and a projection $\pi : \Sigma \cup \Delta \longrightarrow \Sigma$ that erases subscripts (whenever there are). Then one can verify that $L = \pi(L(h) \oplus L(v))$ and hence, by Theorem 4.4 and Corollary 4.9, $L \in$ REC.

All those closure properties confirm the close analogy with the one-dimensional case. The big difference regards the complement operation. In [7], using a combinatorial argument, it is showed that language in Example 3.6 is not tiling recognizable while it is not difficult to write a picture regular expressions for its complement. This proves the following theorem.

Theorem 4.11. REC is *not* closed under complement.

As consequence of this theorem, it is interesting to consider the family REC $\cup$ co-REC of picture languages L such that either L itself or its complement ${}^{\mathsf{C}}L$ is tiling recognizable. Observe that REC is strictly included in REC $\cup$ co-REC. In Section 5 we shall state a necessary condition for a language to be in REC $\cup$ co-REC.

4.2 Unambiguous Recognizable Languages

The definition of recognizability in terms of local languages and projections is implicitly non-deterministic. This can be easily understood if we refer to the one-dimensional case: if no particular constraints are given for the set Θ, the tiling system $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ corresponds in general to a non-deterministic automaton. Moreover Theorem 4.11 shows that is not possible to eliminate non-determinism from this definition (as long as determinism allows complementation).

All these results motivated the definition of the class of *unambiguous recognizable two-dimensional language* firstly given in [6]. Informally, a tiling system is unambiguous if every picture has a unique counter-image in its corresponding local language. Let $L \subseteq \Sigma^{++}$ be a two-dimensional language.

Definition 4.12. A tiling system $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ is an *unambiguous tiling system* for $L = L(\mathcal{T})$ if and only if for any picture $x \in L$ there exists a *unique* local picture $y \in L(\Theta)$ such that $x = \pi(y)$.

An alternative definition for *unambiguous tiling system* is that function π extended to $\Gamma^{++} \rightarrow \Sigma^{++}$ is injective. Observe that an unambiguous tiling system can be viewed as a generalization in two dimensions of the definition of unambiguous automaton that recognizes a string language.

A recognizable two-dimensional language $L \subseteq \Sigma^{++}$ is *unambiguous* if and only if it admits an unambiguous tiling system $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$. We denote by UREC the family of all unambiguous recognizable two-dimensional languages. Obviously it holds true that UREC $\subseteq$ REC.

In [2], it is shown that it *undecidable* whether a given tiling system is unambiguous. Furthermore some closure properties of UREC are proved. The main result in [2] shows that UREC is strictly contained in REC and therefore that there exist languages that are inherently ambiguous. In Section 5 we shall re-state a necessary condition for a language to be in UREC.

5 Recognizability conditions based on complexity functions

In this section we state three theorems that give necessary conditions for a picture language to be in REC $\cup$ co-REC, REC and UREC, respectively. Although these theorems are re-formulations of corresponding ones given in [4], [10], [2], respectively, here all the results are given in this unifying matrix-based framework that allows to make connections among these results that before appeared unrelated.

We first report a lemma given in [10]. Let L be a recognizable picture languages and let $\mathcal{T} = (\Sigma, \Gamma, \Theta, \pi)$ a tiling system recognizing L.

Lemma 5.1. For all $m > 1$ there exists a finite automaton $\mathcal{A}(m)$ with γ^m states that recognizes string language $L(m)$, where $\gamma = |\Gamma \cup \{\#\}|$.

Proof. For any positive integer m, we define the non-deterministic finite automaton $\mathcal{A}(m) = (\Sigma^{1,m}, Q_m, I_m, F_m, \delta_m)$ where $\Sigma^{1,m}$ is the alphabet of the columns of height m over Σ; the set of states Q_m is the set of all possible columns of m symbols in $\Gamma \cup \{\#\}$ therefore $|Q_m| = \gamma$. The set of initial states corresponds to the columns adjacent to the left border while the set of final states F_m contains all the columns of border symbols. The transitions from a given state p to state q are defined by using the adjacency allowed by the set of local tiles. Then, by construction it holds that $\mathcal{A}(m)$ accepts exactly $L(m)$. Q.E.D.

The construction of the automaton in the above proof implies directly the following corollary.

Corollary 5.2. If $L \in$ UREC, then $\mathcal{A}(m)$ is unambiguous.

We can now state the first necessary condition for picture recognizability.

Theorem 5.3. If $L \in$ REC $\cup$ co-REC then there exists a positive integer γ such that, for all $m > 0$, $R_L(m) \leq 2^{\gamma^m}$

Proof. Consider two rows of the Hankel matrix $H_L(m)$ indexed by the words x and y respectively. It is easy to see that such two rows are equal if and only if x and y are in the same Nerode equivalence class of $L(m)$.(Recall that, see also [5], given a language $L \subseteq \Sigma^*$, two words u and v are in the same Nerode equivalence class of L if $\forall w \in \Sigma^* : uw \in L \Leftrightarrow vw \in L$). Thus the number $R_L(m)$ of different rows of $H_L(m)$ coincides with the number of classes of the Nerode equivalence and therefore it corresponds to the number of states of the minimal DFA recognizing $L(m)$. The number of states of the DFA recognizing $L(m)$ obtained by determinization of the NFA $\mathcal{A}(m)$ is at most $2^{|Q_m|} = 2^{\gamma^m}$.

Moreover observe that if ${}^{\complement}L$ is the complement of a given language L, the Hankel matrix of ${}^{\complement}L$ can be obtained by taking the Hankel matrix of L and changing all 0s in 1s and all 1s in 0s. Then the two matrices have the same number of distinct rows, i.e.: $R_L(m) = R_{{}^{\complement}L(m)}$. The thesis follows. Q.E.D.

We now state a necessary condition for a language to be tiling recognizable.

Theorem 5.4. If $L \in \mathrm{REC}$ then there exists a positive integer γ such that, for all $m > 0$, $P_L(m) \leq \gamma^m$.

Proof. Consider a permutation matrix that is a submatrix of $H_L(M)$. Let $x_1, x_2, \ldots, x_n$ be the words that index its rows and let $y_{\sigma(1)}, y_{\sigma(2)}, \ldots, y_{\sigma(n)}$ be the words that index its columns, where σ is the permutation that represents the matrix. To prove the statement of the theorem it suffices to show that $n \leq \gamma^m$ for some γ. Recall that by definition of Hankel matrix, one has the following two conditions:

$$x_i y_{\sigma(i)} \in L \quad \text{for } 1 \leq i \leq n \tag{1}$$

$$x_j y_{\sigma(i)} \notin L \quad \text{for } i \neq j \tag{2}$$

For any i, $1 \leq i \leq n$, denote by $S_i \subseteq Q_m$ the set of states q of the automaton $\mathcal{A}(m)$ such that there exists a path from a starting state to q with label x_i. For condition (1), there exists a state $p_i \in S_i$ such that in $\mathcal{A}(m)$ there is a path with label $y_{\sigma(i)}$ from p_i to an accepting state. Observe that $p_i \notin S_j$ for *all* $j \neq i$ otherwise in the automaton there would be an accepting path for the word $x_j y_{\sigma(i)}$ against condition (2). of the considered permutation matrix. This implies that the number of such sets S_i is at most the number of states of Q_m, that is $n \leq |Q_m| = \gamma^m$ Q.E.D.

For the third theorem we need some additional notations and definitions on matrices. Let M be a boolean matrix. A *1-monocromatic* submatrix of M is any submatrix of M whose elements are all equal to 1. Let $S = \{M_1, M_2, \ldots, M_n\}$ be a set of 1-monocromatic submatrices of M: S is an *exact cover* for M if, for any element $m_{ij} \in M$ such that $m_{ij} = 1$, there exists an integer t such that $m_{ij} \in M_t$ and moreover $M_r \cap M_s = \emptyset$ for $r \neq s$. For instance, consider the Hankel matrix in Example 3.4. Each rectangle of 1s is an Hankel 1-monocromatic submatrix while all $m+1$ rectangles together are an exact cover for the matrix. Denote by $\tau(M)$ the minimal cardinality of an exact cover of M. We now report the following lemma.

Lemma 5.5. Let M be a boolean matrix. Then: $\mathrm{rank}(M) \leq \tau(M)$.

Proof. Let $S = \{M_1, M_2, \ldots, M_{\tau(M)}\}$ be an exact cover of minimal size of matrix M. Let M'_i denote the matrix (of same size of M) obtained from M by changing in 0 all the elements not belonging to M_i. It is easy to verify that all those matrices M'_i have rank 1. Moreover observe that

$$M = M'_1 + M'_2 + \ldots + M'_{\tau(M)}.$$

Then, by using the well known linear algebra fact that the rank of the sum of some matrices in not greater than the sum of the ranks of those matrices, we have that:

$$\operatorname{rank}(M) \leq \operatorname{rank}(M'_1) + \operatorname{rank}(M'_2) + \ldots + \operatorname{rank}(M'_{\tau(M)}) = \tau(M).$$

Q.E.D.

We can now state our third necessary condition.

Theorem 5.6. If $L \in$ UREC then there exists a positive integer γ such that, for all $m > 0$, $K_L(m) \leq \gamma^m$

Proof. Consider the NFA $\mathcal{A}(m)$ for the string language $L(m)$ defined in Lemma 5.1. Observe that, by Corollary 5.2, $\mathcal{A}(m)$ is unambiguous. For every state $q \in Q_m$ consider the sets U_q and V_q of words defined as follows:

- $u \in U_q$ if there exists a path in $\mathcal{A}(m)$ from an initial state to state q with label u;
- $v \in V_q$ if there exists a path in $\mathcal{A}(m)$ from state q to a final state with label v.

Now take the Hankel matrix $H_L(m)$ of language $L(m)$ and consider the submatrix M_q corresponding to language pair (U_q, V_q). M_q is a 1-monocromatic submatrix of $H_L(m)$ because $uv \in L$ for all $u \in U_q$ and all $v \in V_q$.

The set $\mathcal{S}_{\mathcal{A}(m)} = \{M_q \,|\, q \in Q_m\}$ is an exact cover of $H_L(m)$. Indeed any 1 in $H_L(m)$ is in a position corresponding to a row indexed by a word u and a column indexed by a word v such that $uv \in L$ and then it belongs to an element of $\mathcal{S}_{\mathcal{A}(m)}$. Moreover, for the unambiguity of $\mathcal{A}(m)$, it follows that any pair of elements of $\mathcal{S}_{\mathcal{A}(m)}$ has empty intersection. Then, using Lemma 5.5, we can conclude that

$$K_L(m) = \operatorname{rank}(H_L(m)) \leq \tau(H_L(m)) \leq |\mathcal{S}_{\mathcal{A}(m)}| = |Q_m| = \gamma^m.$$

Q.E.D.

6 Separation results

In this section we state some separation results for the classes of recognizable picture languages here considered. We start by showing that there exist languages L such that are neither L nor ${}^{\complement}L$ are recognizable.

Let L_f be a picture language over Σ with $|\Sigma| = \sigma$ of pictures of size $(n, f(n))$ where f is a non-negative function over $\mathbb{N}$. In Example 3.5 it is remarked that $R_{L_f}(m) = f(m)+1$. Then, if we choose a function "greater" than the bound in Theorem 5.3, we obtain the following.

Corollary 6.1. Let $f(n)$ be a function that has asymptotical growth rate greater than 2^{γ^n}, then $L_f \notin \text{REC} \cup \text{co-REC}$.

We now consider four examples of picture languages defined over a given alphabet Σ with $|\Sigma| = \sigma \geq 2$. Those examples will be checked for the inequalities of the Theorems 5.3, 5.4, 5.6 of previous section and used to separate classes REC∪co-REC, REC and UREC. It is interesting to observe that these four languages have very similar definitions as if they were a variation on a theme. Their properties are based on a combination of the existence or non-existence of duplicate columns in the pictures either for a single column or for all the columns. Surprisingly all those variations suffice to separate the introduced recognizable classes. The languages are the following:

$L_{\forall 1} = \{p \in \Sigma^{++} \mid$ *all* columns of p appear *once* in $p\}$

$L_{\forall 2} = \{p \in \Sigma^{++} \mid$ *all* columns of p appear at least *twice* in $p\}$

$L_{\exists 2} = \{p \in \Sigma^{++} \mid$ *there exists* a column in p that appears at least *twice* in $p\}$

$L_{\exists 1} = \{p \in \Sigma^{++} \mid$ *there exists* a column in p that appears only *once* in $p\}$

Notice that language $L_{\exists 2}$ is the language already introduced in Example 4.8 while $L_{\exists 1}$ is the language already introduced in Example 4.10; the remaining two languages are their complements. More precisely: $L_{\forall 1} = {}^{\complement}L_{\exists 2}$ and $L_{\forall 2} = {}^{\complement}L_{\exists 1}$. By using the inequalities on the complexity functions given in the previous section, we shall prove that $L_{\forall 2} \notin$ REC, $L_{\forall 1} \notin$ REC and $L_{\exists 2} \notin$ UREC.

In the proof of the following results we make use of submatrices of the Hankel matrix $H_L(m)$ specified by row's and column's indices in the following set of strings over the column alphabet $\Sigma^{m,1} = \{c_1, c_2, \ldots, c_{\sigma^m}\}$:

$$S^{(m)} = \{c_{i_1} c_{i_2} c_{i_3} \ldots c_{i_k} \mid 1 \leq i_1 < i_2 < i_3 \ldots < i_k \leq \sigma^m\}.$$

Observe that there is a bijection between the set of words $S^{(m)}$ and the family of subsets of $\Sigma^{m,1}$. So one has that $|S^{(m)}| = 2^{\sigma^m}$.

Lemma 6.2. For all $m \geq 1$, $P_{L_{\forall 2}}(m) \geq 2^{\sigma^m}$.

Proof. Consider the Hankel matrix $H_{L_{\forall 2}}(m)$ and its submatrix having row and column indices in the set $S^{(m)}$. We show that such submatrix is an identity matrix (and thus a permutation matrix). Indeed, for any element h_{xy} of the submatrix, with $x, y \in S^{(m)}$ one has that $h_{xy} = 1$ if $xy \in L_{\forall 2}$ and this is true if and only if $x = y$. Then, the remark that the size of this submatrix is 2^{σ^m} concludes the proof. Q.E.D.

From Theorem 5.4 the above Lemma 6.2 one derives the following.

Corollary 6.3. $L_{\forall 2} \notin$ REC.

Consider now language $L_{\exists 1}$ and the complexity function $R_{L_{\exists 1}}(m)$. Then

$$R_{L_{\exists 1}}(m) = R_{L_{\forall 2}}(m) \geq P_{L_{\forall 2}}(m).$$

Since we have proved in Section 4.1 (Example 4.10) that $L_{\exists 1} \in$ REC, from Lemma 6.2 we derive the following.

Corollary 6.4. The bound given in Theorem 5.3 is tight.

We now consider language $L_{\forall 1}$. We prove the following.

Lemma 6.5. For all $m \geq 1$, $P_{L_{\forall 1}}(m) \geq \begin{pmatrix} \sigma^m \\ \frac{\sigma^m}{2} \end{pmatrix}$.

Proof. Consider the following subset $T^{(m)}$ of $S^{(m)}$

$$T^{(m)} = \left\{ c_{i_1} c_{i_2} c_{i_3} \ldots c_{i_k} \mid 1 \leq i_1 < i_2 < i_3 \ldots < i_k \leq \sigma^n,\ k = \frac{\sigma^m}{2} \right\}.$$

Notice that we are implicitly assuming that σ is even: in the opposite case everything can be done similarly but with a bit of more technicality. It is easy to verify that there is a bijection between $T^{(m)}$ and the family of subsets of $\Sigma^{m,1}$ with size $\frac{\sigma^m}{2}$. Therefore $|T^{(m)}| = \begin{pmatrix} \sigma^m \\ \frac{\sigma^m}{2} \end{pmatrix}$.

Consider now the Hankel matrix $H_{L_{\forall 1}}(m)$ and its submatrix having row and column indices in the set $T^{(m)}$. We show that such submatrix is a permutation matrix.

Recall that the elements of $T^{(m)}$ correspond to the subsets of $\Sigma^{m,1}$ with size $\frac{\sigma^m}{2}$, and that any subset of size $\frac{\sigma^m}{2}$ has a unique complement of the same size. This means that, denoting by h_{xy} the element of the Hankel matrix $H_{L_{\forall 1}}(m)$ for $x, y \in T^{(m)}$, one has that $h_{xy} = 1$ if $xy \in L_{\forall 1}$ and this is true if and only if x and y correspond to complementary sets in $\Sigma^{m,1}$. Thus the submatrix is a permutation matrix and the thesis follows. Q.E.D.

By Theorem 5.4 and the above Lemma 6.5 we derive the following.

Corollary 6.6. $L_{\forall 1} \notin \mathrm{REC}$.

Let us now consider the language $L_{\exists 2}$. We prove the following.

Lemma 6.7. For all $m \geq 1$, $K_{L_{\exists 2}}(m) \geq 2^{\sigma^m} - 1$.

Proof. Consider as in Lemma 6.2 the Hankel matrix $H_{L_{\exists 2}}(m)$ and its submatrix, here denoted by $M(m)$, having both row and column indices in the set $S^{(m)}$. To easily compute the rank of $M(m)$ it is useful to introduce a total order in the strings of $S^{(m)}$.

We need the following notations and definitions. Given a sequence $S = (x_1, \ldots, x_n)$ of strings and a string z, denote by Sz the sequence $(x_1 z, \ldots, x_n z)$. If $T = (y_1, \ldots, y_m)$ is another sequence of strings, denote by (S, T) the sequence composed by elements of S followed by the elements of T, i.e. the sequence $(S, T) = (x_1, \ldots, x_n, y_1, \ldots, y_m)$. Further recall that we are considering strings on the alphabet of columns $\Sigma^{m,1} = \{c_1, c_2, \ldots, c_{\sigma^m}\}$.

With these notations we can define the sequence $S^{(m)}$ by induction on the index k of the number of elements in $\Sigma^{m,1}$ involved in the definitions of the strings. The definition is the following:

$$\begin{aligned} S_0^{(m)} &= (\varepsilon) \\ S_k^{(m)} &= (S_{k-1}^{(m)}, S_{k-1}^{(m)} c_k) \end{aligned}$$

for $k = 1, \ldots, \sigma^m$. We have that $S^{(m)} = S_{\sigma^m}^{(m)}$. For instance:

$$\begin{aligned} S_0^{(m)} &= (\varepsilon) \\ S_1^{(m)} &= (\varepsilon, c_1) \\ S_2^{(m)} &= (\varepsilon, c_1, c_2, c_1 c_2) \\ S_3^{(m)} &= (\varepsilon, c_1, c_2, c_1 c_2, c_3, c_1 c_3, c_2 c_3, c_1 c_2 c_3) \end{aligned}$$

By ordering the elements of $S^{(m)}$ as above, the matrix $M(m)$ assumes a particular shape. For instance, the submatrix of $M(m)$ whose row's and column's indices belongs to $S_3^{(m)}$ is represented below.

$$\begin{array}{c|cccccccc|} & \varepsilon & c_1 & c_2 & c_1c_2 & c_3 & c_1c_3 & c_2c_3 & c_1c_2c_3 \\ \hline \varepsilon & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ c_2 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ c_1c_2 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ c_3 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ c_1c_3 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ c_2c_3 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ c_1c_2c_3 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline \end{array}$$

Let $M'(m)$ denote the matrix obtained from $M(m)$ by deleting the first row and the first column. One can show that $M'(m)$ has 1 in all counter-diagonal positions and in all positions below it. In fact, by construction of $M'(m)$, the pair of strings (x, y) corresponding to the rows and columns indices of these positions are such that there exists a symbol $c_i \in \Sigma^{m,1}$ that occurs both in x and y, and hence $xy \in L_{\exists 2}$. Moreover all positions immediately above the counter-diagonal contain 0 since the pair (x, y) of strings corresponding to these positions have no symbols in common. By elementary matrix computations, one has that the determinant det $M'(m) \neq 0$. Since the size of $M'(m)$ is $2^{\sigma^m} - 1$, the thesis follows. Q.E.D.

By Theorem 5.6 and the above Lemma 6.7 we derive the following.

Corollary 6.8. $L_{\exists 2} \notin$ UREC.

We conclude by collecting all the results of this section and state the following separation result.

Theorem 6.9. UREC $\subsetneq$ REC $\subsetneq$ REC $\cup$ co-REC.

Proof. Language $L_{\exists 2}$ separate UREC from REC. In fact, it does not belongs to UREC by Corollary 6.8 while in Example 4.8 it is shown $L_{\exists 2} \in$ REC.

Furthermore language $L_{\forall 2}$ separates families REC and REC $\cup$ co-REC. In fact, from Corollary 6.3, we have that language $L_{\forall 2} \notin$ REC while its complement ${}^{\mathsf{C}}L_{\forall 2} = L_{\exists 1} \in$ REC (see Example 4.10). Q.E.D.

7 Final remarks and open questions

We proposed a unifying framework based on Hankel matrices to deal with recognizable picture languages. As result, we stated three necessary conditions for the classes REC $\cup$ co-REC, REC and UREC. The first natural question that arises regards the non-sufficiency of such statements, more specifically the possibility of refining them to get sufficient conditions. Observe that the technique we used of reducing a picture language L in a sequence of string languages $(L(m))_{m>0}$ on the columns alphabets $\Sigma^{m,1}$ allows to take into account the "complexity" of a picture language along only the horizontal dimension. For instance, consider the languages $L'_{\exists 1}, L'_{\exists 2}, L'_{\forall 1}, L'_{\forall 2}$ obtained by exchanging the word "column" with "rows" in the definitions of corresponding languages given the the previous section. For those languages the necessary conditions we gave are meaningless, nevertheless it is easy to figure out a corresponding technique that, given a picture language L, consider the sequence of string languages $(L'(n))_{n>0}$ on the rows alphabets $\Sigma^{1,n}$ and then consider the Hankel matrices of such languages. Then the question is whether by combining conditions that use such both techniques along the two dimensions we could get strong conditions for the recognizability of the given picture language.

The novelty of these matrix-based complexity functions gives a common denominator to study relations between the *complement problem* and *unambiguity* in this family of recognizable picture languages. In 1994, in the more general context of graphs Wolfgang Thomas *et. al.* had pointed the close relations between these two concepts. In particular, paper [13] ends with the following question formulated specifically for grids graphs and a similar notion of recognizability (here, we report it in our terminology and context).

Question 7.1. Let $L \subseteq \Sigma^{++}$ be a language in REC such that also ${}^{\complement}L \in$ REC. Does this imply that $L \in$ UREC?

As far as we know, there are no negative examples for this question. On the other hand, we have seen a language $L_{\exists 2}$ that belongs to REC such that its complement $L_{\forall 1}$ does not and $L_{\exists 2}$ is not in UREC. Then we can formulate another question.

Question 7.2. Let $L \subseteq \Sigma^{++}$ be a language in REC such that ${}^{\complement}L \notin$ REC. Does this imply that $L \notin$ UREC?

As further work we believe that this matrix-based complexity function technique to discriminate class of languages could be refined to study relations between closure under complement and unambiguity. Notice that a positive answer to any of a single question above does not imply that UREC is closed under complement. Moreover observe that the two problems can be rewritten as whether REC $\cap$ co-REC $\subseteq$ UREC and whether UREC $\subseteq$ REC $\cap$ co-REC, respectively, i.e. they correspond to verify two inverse inclusions. As consequence, if both conjectures were true then we would conclude not only that UREC is closed under complement but also that it is the *largest* subset of REC closed under complement.

References

[1] M. Anselmo, D. Giammarresi, and M. Madonia. From determinism to non-determinism in recognizable two-dimensional languages. In T. Harju, J. Karhumäki, and A. Lepistö, editors, *Developments in Language Theory*, volume 4588 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2007.

[2] M. Anselmo, D. Giammarresi, M. Madonia, and A. Restivo. Unambiguous recognizable two-dimensional languages. *Theor. Inform. Appl.*, 40(2):277–293, 2006.

[3] J. Berstel and C. Reutenauer. *Rational series and their languages*, volume 12 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Berlin, 1988.

[4] J. Cervelle. Langages de figures. Technical report, Ecole Normale Supériure de Lyon, Dept de Mathématiques et Informatique, 1997. Rapport de Stage.

[5] S. Eilenberg. *Automata, languages, and machines. Vol. A.* Academic Press [A subsidiary of Harcourt Brace Jovanovich, Publishers], New York, 1974. Pure and Applied Mathematics, Vol. 58.

[6] D. Giammarresi and A. Restivo. Recognizable picture languages. *IJPRAI*, 6(2&3):241–256, 1992.

[7] D. Giammarresi and A. Restivo. Two-dimensional languages. In *Handbook of formal languages, Vol. 3*, pages 215–267. Springer, Berlin, 1997.

[8] D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Inf. Comput.*, 125(1):32–45, 1996.

[9] J. Hromkovic, S. Seibert, J. Karhumäki, H. Klauck, and G. Schnitger. Communication complexity method for measuring nondeterminism in finite automata. *Inf. Comput.*, 172(2):202–217, 2002.

[10] O. Matz. On piecewise testable, starfree, and recognizable picture languages. In M. Nivat, editor, *FoSSaCS*, volume 1378 of *Lecture Notes in Computer Science*, pages 203–210. Springer, 1998.

[11] O. Matz and W. Thomas. The monadic quantifier alternation hierarchy over graphs is infinite. In *LICS*, pages 236–244, 1997.

[12] I. Mäurer. Weighted picture automata and weighted logics. In B. Durand and W. Thomas, editors, *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 313–324. Springer, 2006.

[13] A. Potthoff, S. Seibert, and W. Thomas. Nondeterminism versus determinism of finite automata over directed acyclic graphs. *Bull. Belg. Math. Soc. Simon Stevin*, 1(2):285–298, 1994. Journées Montoises (Mons, 1992).

[14] S. C. Reghizzi and M. Pradella. A SAT-based parser and completer for pictures specified by tiling. *Pattern Recognition.* To appear.

[15] A. Salomaa and M. Soittola. *Automata-theoretic aspects of formal power series.* Springer-Verlag, New York, 1978. Texts and Monographs in Computer Science.

[16] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.

[17] W. Thomas. On logics, tilings, and automata. In J. L. Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 441–454. Springer, 1991.

Applying Blackwell optimality: priority mean-payoff games as limits of multi-discounted games[*]

Hugo Gimbert[1]
Wiesław Zielonka[2]

[1] Laboratoire d'informatique
École polytechnique
91128 Palaiseau Cedex, France
gimbert@lix.polytechnique.fr

[2] Laboratoire d'Informatique Algorithmique: Fondements et Applications
Université Paris Diderot, Paris 7
Case 7014
75205 Paris Cedex 13, France
zielonka@liafa.jussieu.fr

Abstract

We define and examine priority mean-payoff games—a natural extension of parity games. By adapting the notion of Blackwell optimality borrowed from the theory of Markov decision processes we show that priority mean-payoff games can be seen as a limit of special multi-discounted games.

1 Introduction

One of the major achievements of the theory of stochastic games is the result of Mertens and Neyman [15] showing that the values of mean-payoff games are the limits of the values of discounted games. Since the limit of the discounted payoff is related to Abel summability while the mean-payoff is related to Cesàro summability of infinite series, and classical abelian and tauberian theorems establish tight links between these two summability methods, the result of Mertens and Neyman, although technically very difficult, comes with no surprise.

In computer science similar games appeared with the work of Gurevich and Harrington [12] (games with Muller condition) and Emerson and Jutla [5] and Mostowski [16] (parity games).

However discounted and mean-payoff games also seem very different from Muller/parity games. The former, inspired by economic applications, are

[*] Partially supported by the French ANR-SETI project AVERISS

games with real valued payments, the latter, motivated by logics and automata theory, have only two outcomes, the player can win or lose.

The theory of parity games was developed independently from the theory of discounted/mean-payoff games [11] even though it was noted by Jurdziński [14] that deterministic parity games on finite arenas can be reduced to mean-payoff games[1].

Recently de Alfaro, Henzinger and Majumdar [3] presented results that indicate that it is possible to obtain parity games as an appropriate limit of multi-discounted games. In fact, the authors of [3] use the language of the μ-calculus rather than games, but as the links between μ-calculus and parity games are well-known since the advent [5], it is natural to wonder how discounted μ-calculus from [3] can be reflected in games.

The aim of this paper is to examine in detail the links between discounted and parity games suggested by [3]. In our study we use the tools and methods that are typical for classical game theory but nearly never used for parity games. We want to persuade the reader that such tools, conceived for games inspired by economic applications, can be successfully applied to games that come from computer science.

As a by-product we obtain a new class of games—priority mean-payoff games — that generalise in a very natural way parity games but contrary to the latter allow to quantify the gains and losses of the players.

The paper is organised as follows.

In Section 2 we introduce the general framework of deterministic zero-sum infinite games used in the paper, we define optimal strategies, game values and introduce positional (i.e. memoryless) strategies.

In Section 3 we present discounted games. Contrary to classical game theory where there is usually only one discount factor, for us it is crucial to work with multi-discounted games where the discount factor can vary from state to state.

Section 4 is devoted to the main class of games examined in this paper—priority mean-payoff games. We show that for these games both players have optimal positional strategies (on finite arenas).

In classical game theory there is a substantial effort to refine the notion of optimal strategies. To this end Blackwell [2] defined a new notion of optimality that allowed him a fine-grained classification of optimal strategies for mean-payoff games. In Section 5 we adapt the notion of Blackwell optimality to our setting. We use Blackwell optimality to show that in some strong sense priority mean-payoff games are a limit of a special class of multi-discounted games.

[1] But this reduction seems to be proper for deterministic games and not possible for perfect information stochastic games.

The last Section 6 discusses briefly some other applications of Blackwell optimality.

Since the aim of this paper is not only to present new results but also to familiarize the computer science community with methods of classical game theory we have decided to make this paper totally self-contained. We present all proofs, even the well-known proof of positionality of discounted games[2]. For the same reason we also decided to limit ourselves to deterministic games. Similar results can be proved for perfect information stochastic games [10, 9] but the proofs become much more involved. We think that the deterministic case is still of interest and has the advantage of beeing accessible through elementary methods.

The present paper is an extended and improved version of [8].

2 Games

An *arena* is a tuple $\mathcal{A} = (S_1, S_2, A)$, where S_1 and S_2 are the sets of *states* that are controlled respectively by player 1 and player 2, A is the set of *actions*.

By $S = S_1 \cup S_2$ we denote the set of all states. Then $A \subseteq S \times S$, i.e. each action $a = (s', s'') \in A$ is a couple composed of the *source state* $\text{source}(a) = s'$ and the *target state* $\text{target}(a) = s''$. In other words, an arena is just a directed graph with the set of vertices S partitioned onto S_1 and S_2 with A as the set of edges.

An action a is said to be *available* at state s if $\text{source}(a) = s$ and the set of all actions available at s is denoted by $A(s)$.

We consider only arenas where the set of states is finite and such that for each state s the set $A(s)$ of available actions is non-empty.

A *path* in arena $\mathcal{A}$ is a finite or infinite sequence $p = s_0 s_1 s_2 \ldots$ of states such that for all i, $(s_i, s_{i+1}) \in A$. The first state is the source of p, $\text{source}(p) = s_0$, if p is finite then the last state is the target of p, $\text{target}(p)$.

Two players 1 and 2 play on $\mathcal{A}$ in the following way. If the current state s is controlled by player $P \in \{1, 2\}$, i.e. $s \in S_P$, then player P chooses an action $a \in A(s)$ available at s, this action is executed and the system goes to the state $\text{target}(a)$.

Starting from an initial state s_0, the infinite sequence of consecutive moves of both players yields an infinite sequence $p = s_0 s_1 s_2 \ldots$ of visited states. Such sequences are called *plays*, thus plays in this game are just infinite paths in the underlying arena $\mathcal{A}$.

[2] But this can be partially justified since we need positionality of multi-discounted games while in the literature usually simple discounted games are treated. We should admit however that passing from discounted to multi-discounted games needs only minor obvious modifications.

We shall also use the term “a finite play” as a synonym of “a finite path” but “play” without any qualifier will always denote an infinite play/path.

A *payoff mapping*

$$u : S^\omega \to \mathbb{R} \tag{1.1}$$

maps infinite sequences of states to real numbers. The interpretation is that at the end of a play p player 1 receives from player 2 the *payoff* $u(p)$ (if $u(p) < 0$ then it is rather player 2 that receives from player 1 the amount $|u(p)|$).

A *game* is couple $(\mathcal{A}, u)$ composed of an arena and a payoff mapping.

The obvious aim of player 1 (the maximizer) in such a game is to maximize the received payment, the aim of player 2 (the minimizer) is opposite, he wants to minimize the payment paid to his adversary.

A strategy of a player P is his plan of action that tells him which action to take when the game is at a state $s \in S_P$. The choice of the action can depend on the whole past sequence of moves.

Therefore a *strategy* for player 1 is a mapping

$$\sigma : \{p \mid p \text{ a finite play with } \mathrm{target}(p) \in S_1\} \longrightarrow S \tag{1.2}$$

such that for each finite play p with $s = \mathrm{target}(p) \in S_1$, $(s, \sigma(p)) \in A(s)$.

Strategy σ of player 1 is said to be *positional* if for every state $s \in S_1$ and every finite play p with $\mathrm{target}(p) = s$, $\sigma(p) = \sigma(s)$. Thus the action chosen by a positional strategy depends only on the current state, previously visited states are irrelevant. Therefore a positional strategy of player 1 can be identified with a mapping

$$\sigma : S_1 \to S \tag{1.3}$$

such that for all $s \in S_1, (s, \sigma(s)) \in A(s)$.

A finite or infinite play $p = s_0 s_1 \ldots$ is said to be *consistent* with a strategy σ of player 1 if, for each $i \in \mathbb{N}$ such that $s_i \in S_1$, we have $(s_i, \sigma(s_0 \ldots s_i)) \in A$.

Strategies, positional strategies and consistent plays are defined in the analogous way for player 2 with S_2 replacing S_1.

In the sequel Σ and $\mathcal{T}$ will stand for the set of strategies for player 1 and player 2 while Σ_p and $\mathcal{T}_p$ are the corresponding sets of positional strategies.

The letters σ and τ, with subscripts or superscripts if necessary, will be used to denote strategies of player 1 and player 2 respectively.

Given a pair of strategies $\sigma \in \Sigma$ and $\tau \in \mathcal{T}$ and an initial state s, there exists a unique infinite play in arena $\mathcal{A}$, denoted $p(s, \sigma, \tau)$, consistent with σ and τ and such that $s = \mathrm{source}(p(s, \sigma, \tau))$.

Definition 2.1. Strategies $\sigma^\sharp \in \Sigma$ and $\tau^\sharp \in \mathcal{T}$ are *optimal* in the game $(\mathcal{A}, u)$ if

$$\forall s \in S, \forall \sigma \in \Sigma, \forall \tau \in \mathcal{T},$$
$$u(p(s,\sigma,\tau^\sharp)) \leq u(p(s,\sigma^\sharp,\tau^\sharp)) \leq u(p(s,\sigma^\sharp,\tau)) \ . \quad (1.4)$$

Thus if strategies $\sigma^\sharp$ and $\tau^\sharp$ are optimal then the players do not have any incentive to change them unilaterally: player 1 cannot increase his gain by switching to another strategy σ while player 2 cannot decrease his losses by switching to another strategy τ.

In other words, if player 2 plays according to $\tau^\sharp$ then the best response of player 1 is to play with $\sigma^\sharp$, no other strategy can do better for him. Conversely, if player 1 plays according to $\sigma^\sharp$ then the best response of player 2 is to play according to $\tau^\sharp$ as no other strategy does better to limit his losses.

We say that a payoff mapping u *admits optimal positional strategies* if for all games $(\mathcal{A}, u)$ over *finite arenas* there exist optimal positional strategies for both players. We should emphasize that the property defined above is a property of the payoff mapping and not a property of a particular game, we require that both players have optimal positional strategies for *all possible games* over finite arenas.

It is important to note that zero-sum games that we consider here, i.e. the games where the gain of one player is equal to the loss of his adversary, satisfy the exchangeability property for optimal strategies:

for any two pairs of optimal strategies $(\sigma^\sharp, \tau^\sharp)$ and $(\sigma^\star, \tau^\star)$, the pairs $(\sigma^\star, \tau^\sharp)$ and $(\sigma^\sharp, \tau^\star)$ are also optimal and, moreover,

$$u(p(s,\sigma^\sharp,\tau^\sharp)) = u(p(s,\sigma^\star,\tau^\star)) \ ,$$

i.e. the value of $u(p(s,\sigma^\sharp,\tau^\sharp))$ is independent of the choice of the optimal strategies—this is *the value of the game* $(\mathcal{A}, u)$ *at state* s.

We end this general introduction with two simple lemmas.

Lemma 2.2. Let u be a payoff mapping admitting optimal positional strategies for both players.

(A) Suppose that $\sigma \in \Sigma$ is any strategy while $\tau^\sharp \in \mathcal{T}_p$ is positional. Then there exists a positional strategy $\sigma^\sharp \in \Sigma_p$ such that

$$\text{for all } s \in S, \quad u(p(s,\sigma,\tau^\sharp)) \leq u(p(s,\sigma^\sharp,\tau^\sharp)) \ . \quad (1.5)$$

(B) Similarly, if $\tau \in \mathcal{T}$ is any strategy and $\sigma^\sharp \in \Sigma_p$ a positional strategy then there exists a positional strategy $\tau^\sharp \in \mathcal{T}_p$ such that

$$\text{for all } s \in S, \quad u(p(s,\sigma^\sharp,\tau^\sharp)) \leq u(p(s,\sigma^\sharp,\tau)) \ .$$

Proof. We prove (A), the proof of (B) is similar. Take any strategies $\sigma \in \Sigma$ and $\tau^\sharp \in \mathcal{T}_p$. Let $\mathcal{A}'$ be a subarena of $\mathcal{A}$ obtained by restricting the actions of player 2 to the actions given by the strategy $\tau^\sharp$, i.e. in $\mathcal{A}'$ the only possible strategy for player 2 is the strategy $\tau^\sharp$. The actions of player 1 are not restricted, i.e. in $\mathcal{A}'$ player 1 has the same available actions as in $\mathcal{A}$, in particular σ is a valid strategy of player 1 on $\mathcal{A}'$. Since u admits optimal positional strategies, player 1 has an optimal positional strategy $\sigma^\sharp$ on $\mathcal{A}'$. But (1.5) is just the optimality condition of $\sigma^\sharp$ on $\mathcal{A}'$. Q.E.D.

Lemma 2.3. Suppose that the payoff mapping u admits optimal positional strategies. Let $\sigma^\sharp \in \Sigma_p$ and $\tau^\sharp \in \mathcal{T}_p$ be positional strategies such that

$$\forall s \in S, \forall \sigma \in \Sigma_p, \forall \tau \in \mathcal{T}_p,$$
$$u(p(s,\sigma,\tau^\sharp)) \leq u(p(s,\sigma^\sharp,\tau^\sharp) \leq u(p(s,\sigma^\sharp,\tau)) \ , \quad (1.6)$$

i.e. $\sigma^\sharp$ and $\tau^\sharp$ are optimal in the class of positional strategies. Then $\sigma^\sharp$ and $\tau^\sharp$ are optimal in the class of all strategies.

Proof. Suppose that

$$\exists \tau \in \mathcal{T}, \quad u(p(s,\sigma^\sharp,\tau)) < u(p(s,\sigma^\sharp,\tau^\sharp)) \ . \quad (1.7)$$

By Lemma 2.2 (B) there exists a positional strategy $\tau^\star \in \mathcal{T}_p$ such that $u(p(s,\sigma^\sharp,\tau^\star)) \leq u(p(s,\sigma^\sharp,\tau)) < u(p(s,\sigma^\sharp,\tau^\sharp))$, contradicting (1.6). Thus $\forall \tau \in \mathcal{T}, u(p(s,\sigma^\sharp,\tau^\sharp)) \leq u(p(s,\sigma^\sharp,\tau))$. The left hand side of (1.4) can be proved in a similar way. Q.E.D.

3 Discounted Games

Discounted games were introduced by Shapley [19] who proved that stochastic discounted games admit stationary optimal strategies. Our exposition follows very closely the original approach of [19] and that of [17]. Nevertheless we present a complete proof for the sake of completeness.

Arenas for discounted games are equipped with two mappings defined on the set S of states: the *discount mapping*

$$\lambda : S \longrightarrow [0,1)$$

associates with each state s a discount factor $\lambda(s) \in [0,1)$ and the *reward mapping*

$$r : S \longrightarrow \mathbb{R} \quad (1.8)$$

maps each state s to a real valued reward $r(s)$.

The payoff mapping

$$u_\lambda : S^\omega \longrightarrow \mathbb{R}$$

for discounted games is defined in the following way: for each play $p = s_0 s_1 s_2 \ldots \in S^\omega$

$$\begin{aligned} u_\lambda(p) = & \quad (1-\lambda(s_0))r(s_0) + \lambda(s_0)(1-\lambda(s_1))r(s_1) + \\ & \quad \lambda(s_0)\lambda(s_1)(1-\lambda(s_2))r(s_2) + \ldots \\ = & \quad \sum_{i=0}^{\infty} \lambda(s_0)\ldots\lambda(s_{i-1})(1-\lambda(s_i))r(s_i) \ . \end{aligned} \tag{1.9}$$

Usually when discounted games are considered it is assumed that there is only one discount factor, i.e. that there exists $\lambda \in [0,1)$ such that $\lambda(s) = \lambda$ for all $s \in S$. But for us it is essential that discount factors depend on the state.

It is difficult to give an intuitively convincing interpretation of (1.9) if we use this payoff mapping to evaluate infinite games. However, there is a natural interpretation of (1.9) in terms of stopping games, in fact this is the original interpretation given by Shapley [19].

In *stopping games* the nature introduces an element of uncertainty. Suppose that at a stage i a state s_i is visited. Then, before the player controlling s_i is allowed to execute an action, a (biased) coin is tossed to decide if the game stops or if it will continue. The probability that the game stops is $1 - \lambda(s_i)$ (thus $\lambda(s_i)$ gives the probability that the game continues). Let us note immediately that since we have assumed that $0 \le \lambda(s) < 1$ for all $s \in S$, the stopping probabilities are strictly positive therefore the game actually stops with probability 1 after a finite number of steps.

If the game stops at s_i then player 1 receives from player 2 the payment $r(s_i)$. This ends the game, there is no other payment in the future.

If the game does not stop at s_i then there is no payment at this stage and the player controlling the state s_i is allowed to choose an action to execute[3].

Now note that $\lambda(s_0)\ldots\lambda(s_{i-1})(1-\lambda(s_i))$ is the probability that the game have not stopped at any of the states $s_0, \ldots, s_{i-1}$ but it does stop at state s_i. Since this event results in the payment $r(s_i)$ received by player 1, Eq. (1.9) gives in fact the *payoff expectation* for a play $s_0 s_1 s_2 \ldots$.

Shapley [19] proved[4] that

Theorem 3.1 (Shapley)**.** Discounted games $(\mathcal{A}, u_\lambda)$ over finite arenas admit optimal positional strategies for both players.

[3] More precisely, if the nature does not stop the game then the player controlling the current state *is obliged* to execute an action, players cannot stop the game by themselves.

[4] In fact, Shapley considered a much larger class of stochastic games.

Proof. Let $\mathbb{R}^S$ be the vector space consisting of mappings from S to $\mathbb{R}$. For $f \in \mathbb{R}^S$, set $||f|| = \sup_{s\in S} |f(s)|$. Since S is finite $||\cdot||$ is a norm for which $\mathbb{R}^S$ is complete. Consider an operator $\Psi : \mathbb{R}^S \longrightarrow \mathbb{R}^S$, for $f \in \mathbb{R}^S$ and $s \in S$,

$$\Psi[f](s) = \begin{cases} \max_{(s,s')\in A(s)}(1-\lambda(s))r(s) + \lambda(s)f(s') & \text{if } s \in S_1 \\ \min_{(s,s')\in A(s)}(1-\lambda(s))r(s) + \lambda(s)f(s') & \text{if } s \in S_2 \end{cases} .$$

$\Psi[f](s)$ can be seen as the value of a one shot game that gives the payoff $(1-\lambda(s))r(s)+\lambda(s)f(s')$ if the player controlling the state s choses an action $(s, s') \in A(s)$.

We can immediately note that Ψ is monotone, if $f \geq g$ then $\Psi[f] \geq \Psi[g]$, where $f \geq g$ means that $f(s) \geq g(s)$ for all states $s \in S$.
Moreover, for any positive constant c and $f \in \mathbb{R}^S$

$$\Psi[f] - c\lambda\mathbf{1} \leq \Psi[f - c\cdot\mathbf{1}] \quad\text{and}\quad \Psi[f + c\cdot\mathbf{1}] \leq \Psi[f] + c\lambda\mathbf{1} \ , \tag{1.10}$$

where $\mathbf{1}$ is the constant mapping, $\mathbf{1}(s) = 1$ for each state s, and $\lambda = \sup_{s\in S}\lambda(s)$.
Therefore, since

$$f - ||f-g||\cdot\mathbf{1} \leq g \leq f + ||f-g||\cdot\mathbf{1} \ ,$$

we get

$$\Psi[f] - \lambda||f-g||\cdot\mathbf{1} \leq \Psi[g] \leq \Psi[f] + \lambda||f-g||\cdot\mathbf{1} \ ,$$

implying

$$||\Psi[f] - \Psi[g]|| \leq \lambda||f-g|| \ .$$

By the Banach contraction principle, Ψ has a unique fixed point $w \in \mathbb{R}^S$, $\Psi[w] = w$. From the definition of Ψ we can see that this unique fixed point satisfies the inequalities

$$\forall s \in S_1, \forall (s,s') \in A(s), \quad w(s) \geq (1-\lambda(s))r(s) + \lambda(s)w(s') \tag{1.11}$$

and

$$\forall s \in S_2, \forall (s,s') \in A(s), \quad w(s) \leq (1-\lambda(s))r(s) + \lambda(s)w(s') \ . \tag{1.12}$$

Moreover, for each $s \in S$ there is an action $\xi(s) = (s,s') \in A(s)$ such that

$$w(s) = (1-\lambda(s))r(s) + \lambda(s)w(s') \ . \tag{1.13}$$

We set $\sigma^\sharp(s) = \xi(s)$ for $s \in S_1$ and $\tau^\sharp(s) = \xi(s)$ for $s \in S_2$ and we show that $\sigma^\sharp$ and $\tau^\sharp$ are optimal for player 1 and 2. Suppose that player 1 plays according to the strategy $\sigma^\sharp$ while player 2 according to some strategy τ. Let $p(s_0,\sigma^\sharp,\tau)=s_0s_1s_2\ldots$. Then, using (1.12) and (1.13), we get by induction on k that

$$w(s_0)\le\sum_{i=0}^{k}\lambda(s_0)\ldots\lambda(s_{i-1})(1-\lambda(s_i))r(s_i)+\lambda(s_0)\ldots\lambda(s_k)w(s_{k+1})\ .$$

Tending k to infinity we get

$$w(s_0)\le u_\lambda(p(s_0,\sigma^\sharp,\tau))\ .$$

In a similar way we can establish that for any strategy σ of player 1,

$$w(s_0)\ge u_\lambda(p(s_0,\sigma,\tau^\sharp))$$

and, finally, that

$$w(s_0)=u_\lambda(p(s_0,\sigma^\sharp,\tau^\sharp))\ ,$$

proving the optimality of $\sigma^\sharp$ and $\tau^\sharp$. Q.E.D.

4 Priority mean-payoff games

In mean-payoff games the players try to optimize (maximize/minimize) the mean value of the payoff received at each stage. In such games the *reward mapping*

$$r:S\longrightarrow\mathbb{R} \tag{1.14}$$

gives, for each state s, the payoff received by player 1 when s is visited. The payoff of an infinite play is defined as the mean value of daily payments:

$$u_m(s_0s_1s_2\ldots)=\limsup_k\frac{1}{k+1}\sum_{i=0}^{k}r(s_i)\ , \tag{1.15}$$

where we take lim sup rather than the simple limit since the latter may not exist. As proved by Ehrenfeucht and Mycielski [4], such games admit optimal positional strategies; other proofs can be found for example in [1, 7].

We slightly generalize mean-payoff games by equipping arenas with a new mapping

$$w:S\longrightarrow\mathbb{R}_+ \tag{1.16}$$

associating with each state s a *strictly positive* real number $w(s)$, the *weight* of s. We can interpret $w(s)$ as the amount of time spent at state s each time when s is visited. In this setting $r(s)$ should be seen as the payoff by a time unit when s is visited, thus the mean payoff received by player 1 is

$$u_m(s_0s_1s_2\ldots)=\limsup_k\frac{\sum_{i=0}^{k}w(s_i)r(s_i)}{\sum_{i=0}^{k}w(s_i)}\ . \tag{1.17}$$

Note that in the special case when the weights are all equal to 1, the weighted mean value (1.17) reduces to (1.15).

As a final ingredient we add to our arena a *priority mapping*

$$\pi : S \longrightarrow \mathbb{Z}_+ \tag{1.18}$$

giving a positive integer *priority* $\pi(s)$ of each state s.

We define the *priority* of a play $p = s_0s_1s_2\ldots$ as the *smallest* priority appearing infinitely often in the sequence $\pi(s_0)\pi(s_1)\pi(s_2)\ldots$ of priorities visited in p:

$$\pi(p) = \liminf_i \pi(s_i) \ . \tag{1.19}$$

For any priority a, let $\mathbf{1}_a : S \longrightarrow \{0,1\}$ be the indicator function of the set $\{s \in S \mid \pi(s) = a\}$, i.e.

$$\mathbf{1}_a(s) = \begin{cases} 1 & \text{if } \pi(s) = a \\ 0 & \text{otherwise.} \end{cases} \tag{1.20}$$

Then the priority mean payoff of a play $p = s_0s_1s_2\ldots$ is defined as

$$u_{\mathrm{pm}}(p) = \limsup_k \frac{\sum_{i=0}^{k} \mathbf{1}_{\pi(p)}(s) \cdot w(s_i) \cdot r(s_i)}{\sum_{i=0}^{k} \mathbf{1}_{\pi(p)}(s_i) \cdot w(s_i)} \ . \tag{1.21}$$

In other words, to calculate priority mean payoff $u_{\mathrm{pm}}(p)$ we take weighted mean payoff but with the weights of all states having priorities different from $\pi(p)$ shrunk to 0. (Let us note that the denominator $\sum_{i=0}^{k} \mathbf{1}_{\pi(p)}(s_i) \cdot w(s_i)$ is different from 0 for k large enough, in fact it tends to infinity since $\mathbf{1}_{\pi(p)}(s_i) = 1$ for infinitely many i. For small k the numerator and the denominator can be equal to 0 and then, to avoid all misunderstanding, it is convenient to assume that the indefinite value $0/0$ is equal to $-\infty$.)

Suppose that for all states s,

- $w(s) = 1$ and
- $r(s)$ is 0 if $\pi(s)$ is even, and $r(s)$ is 1 if $\pi(s)$ is odd.

Then the payoff obtained by player 1 for any play p is either 1 if $\pi(p)$ is odd, or 0 if $\pi(p)$ is even. If we interpret the payoff 1 as the victory of player 1, and payoff 0 as his defeat then such a game is just the usual parity game [5, 11].

It turns out that

Theorem 4.1. For any arena $\mathcal{A}$ the priority mean-payoff game $(\mathcal{A}, u_{\mathrm{pm}})$ admits optimal positional strategies for both players.

There are many possible ways to prove Theorem 4.1, for example by adapting the proofs of positionality of mean payoff games from [4] and [1] or by verifying that u_{pm} satisfies sufficient positionality conditions given in [7]. Below we give a complete proof based mainly on ideas from [7, 20].

A payoff mapping is said to be *prefix independent* if for each play p and for each factorization $p = xy$ with x finite we have $u(p) = u(y)$, i.e. the payoff does not depend on finite prefixes of a play. The reader can readily persuade herself that the priority mean payoff mapping u_{pm} is prefix independent.

Lemma 4.2. Let u be a prefix-independent payoff mapping such that both players have optimal positional strategies $\sigma^\sharp$ and $\tau^\sharp$ in the game $(\mathcal{A}, u)$. Let $\mathrm{val}(s) = p(s, \sigma^\sharp, \tau^\sharp)$, $s \in S$, be the game value for an initial state s.

For any action $(s,t) \in A$,

(1) if $s \in S_1$ then $\mathrm{val}(s) \geq \mathrm{val}(t)$,

(2) if $s \in S_2$ then $\mathrm{val}(s) \leq \mathrm{val}(t)$,

(3) if $s \in S_1$ and $\sigma^\sharp(s) = t$ then $\mathrm{val}(s) = \mathrm{val}(t)$,

(4) if $s \in S_2$ and $\tau^\sharp(s) = t$ then $\mathrm{val}(s) = \mathrm{val}(t)$.

Proof. (1). This is quite obvious. If $s \in S_1$, $(s,t) \in A$ and $\mathrm{val}(s) < \mathrm{val}(t)$ then for a play starting at s player 1 could secure for himself at least $\mathrm{val}(t)$ by executing first the action (s,t) and next playing with his optimal strategy. But this contradicts the definition of $\mathrm{val}(s)$ since from s player 2 has a strategy that limits his losses to $\mathrm{val}(s)$.

The proof of (2) is obviously similar.

(3). We know by (1) that if $s \in S_1$ and $\sigma^\sharp(s) = t$ then $\mathrm{val}(s) \geq \mathrm{val}(t)$. This inequality cannot be strict since from t player 2 can play in such a way that his loss does not exceed $\mathrm{val}(t)$.

(4) is dual to (1). Q.E.D.

Proof of Theorem 4.1. We define the size of an arena $\mathcal{A}$ to be the difference $|A| - |S|$ of the number of actions and the number of states and we carry the proof by induction on the size of $\mathcal{A}$. Note that since for each state there is at least one available action the size of each arena is ≥ 0.

If for each state there is only one available action then the number of actions is equal to the number of states, the size of $\mathcal{A}$ is 0, and each player has just one possible strategy, both these strategies are positional and, obviously, optimal.

Suppose that both players have optimal positional strategies for arenas of size $< k$ and let $\mathcal{A}$ be of size k, $k \geq 1$.

Then there exists a state with at least two available actions. Let us fix such a state t, we call it the *pivot*. We assume that t is controlled by player 1

$$t \in S_1 \tag{1.22}$$

(the case when it is controlled by player 2 is symmetric).

Let $A(t) = A_L(t) \cup A_R(t)$ be a partition of the set $A(t)$ of actions available at t onto two disjoint non-empty sets. Let $\mathcal{A}_L$ and $\mathcal{A}_R$ be two arenas, we call them left and right arenas, both of them having the same states as $\mathcal{A}$, the same reward, weight and priority mappings and the same available actions for all states different from t. For the pivot state t, $\mathcal{A}_L$ and $\mathcal{A}_R$ have respectively $A_L(t)$ and $A_R(t)$ as the sets of available actions. Thus, since $\mathcal{A}_L$ and $\mathcal{A}_R$ have less actions than $\mathcal{A}$, their size is smaller than the size of $\mathcal{A}$ and, by induction hypothesis, both players have optimal positional strategies: $(\sigma^\sharp_L, \tau^\sharp_L)$ on $\mathcal{A}_L$ and $(\sigma^\sharp_R, \tau^\sharp_R)$ on $\mathcal{A}_R$.

We set $\mathrm{val}_L(s) = u_{\mathrm{pm}}(p(s, \sigma^\sharp_L, \tau^\sharp_L))$ and $\mathrm{val}_R(s) = u_{\mathrm{pm}}(p(s, \sigma^\sharp_R, \tau^\sharp_R))$ to be the values of a state s respectively in the left and the right arena.

Without loss of generality we can assume that for the pivot state t

$$\mathrm{val}_L(t) \le \mathrm{val}_R(t) \ . \tag{1.23}$$

We show that this implies that

$$\text{for all } s \in S, \quad \mathrm{val}_L(s) \le \mathrm{val}_R(s) \ . \tag{1.24}$$

Suppose the contrary, i.e. that the set

$$X = \{s \in S \mid \mathrm{val}_L(s) > \mathrm{val}_R(s)\}$$

is non-empty. We define a positional strategy σ^* for player 1

$$\sigma^*(s) = \begin{cases} \sigma^\sharp_L(s) & \text{if } s \in X \cap S_1 \\ \sigma^\sharp_R(s) & \text{if } s \in (S \setminus X) \cap S_1. \end{cases} \tag{1.25}$$

Note that, since the pivot state t does not belong to X, for $s \in X \cap S_1$, $\sigma^\sharp_L(s)$ is valid action for player 1 not only in $\mathcal{A}_L$ but also in $\mathcal{A}_R$, therefore the strategy σ^* defined above is a valid positional strategy on the arena $\mathcal{A}_R$. We claim that

For games on $\mathcal{A}_R$ starting at a state $s_0 \in X$ strategy σ^* guarantees that player 1 wins at least $\mathrm{val}_L(s_0)$ (against any strategy of player 2).
(1.26)

Suppose that we start a game on $\mathcal{A}_R$ at a state s_0 and player 1 plays according to σ^* while player 2 uses any strategy τ. Let

$$p(s_0, \sigma^*, \tau) = s_0 s_1 s_2 \ldots \tag{1.27}$$

be the resulting play. We define

$$\text{for all } s \in S, \quad \mathrm{val}(s) = \begin{cases} \mathrm{val}_L(s) & \text{for } s \in X, \\ \mathrm{val}_R(s) & \text{for } s \in S \setminus X. \end{cases} \tag{1.28}$$

We shall show that the sequence $\mathrm{val}(s_0), \mathrm{val}(s_1), \mathrm{val}(s_2), \ldots$ is nondecreasing,

$$\text{for all } i, \quad \mathrm{val}(s_i) \le \mathrm{val}(s_{i+1}) \ . \tag{1.29}$$

Since strategies $\sigma_L^\sharp$ and $\sigma_R^\sharp$ are optimal in $\mathcal{A}_L$ and $\mathcal{A}_R$, Lemma 4.2 and (1.28) imply that for all i

$$\mathrm{val}(s_i) = \mathrm{val}_L(s_i) \le \mathrm{val}_L(s_{i+1}) \quad \text{if } s_i \in X, \tag{1.30}$$

and

$$\mathrm{val}(s_i) = \mathrm{val}_R(s_i) \le \mathrm{val}_R(s_{i+1}) \quad \text{if } s_i \in S \setminus X. \tag{1.31}$$

To prove (1.29) there are four cases to examine:

(1) Suppose that s_i and s_{i+1} belong to X. Then $\mathrm{val}(s_{i+1}) = \mathrm{val}_L(s_{i+1})$ and (1.29) follows from (1.30).

(2) Suppose that s_i and s_{i+1} belong to $S \setminus X$. Then $\mathrm{val}(s_{i+1}) = \mathrm{val}_R(s_{i+1})$ and now (1.29) follows from (1.31).

(3) Let $s_i \in X$ and $s_{i+1} \in S \setminus X$. Then (1.29) follows from (1.30) and from the fact that $\mathrm{val}_L(s_{i+1}) \le \mathrm{val}_R(s_{i+1}) = \mathrm{val}(s_{i+1})$.

(4) Let $s_i \in S \setminus X$ and $s_{i+1} \in X$. Then $\mathrm{val}_R(s_{i+1}) < \mathrm{val}_L(s_{i+1}) = \mathrm{val}(s_{i+1})$, which, by (1.31), implies (1.29). Note that in this case we have the strict inequality $\mathrm{val}(s_i) < \mathrm{val}(s_{i+1})$.

This finishes the proof of (1.29).

Since the set $\{\mathrm{val}(s) \mid s \in S\}$ is finite, (1.29) implies that the sequence $\mathrm{val}(s_i), i = 0, 1, \ldots,$ is ultimately constant. But examining the case (4) above we have established that each passage from $S \setminus X$ to X strictly increases the value of val. Thus from some stage n onward all states s_i, $i \ge n$, are either in X or in $S \setminus X$. Therefore, according to (1.25), from the stage n onward player 1 always plays either $\sigma_L^\sharp$ or $\sigma_R^\sharp$ and the optimality of both strategies assures that he wins at least $\mathrm{val}(s_n)$, i.e.

$$u_{\mathrm{pm}}(p(s_0, \sigma^*, \tau)) = u_{\mathrm{pm}}(s_0 s_1 \ldots) = \\ u_{\mathrm{pm}}(s_n s_{n+1} s_{n+2} \ldots) \ge \mathrm{val}(s_n) \ge \mathrm{val}(s_0).$$

In particular, if $s_0 \in X$ then using strategy σ^* player 1 secures for himself the payoff of at least $\text{val}(s_0) = \text{val}_L(s_0)$ against any strategy of player 2, which proves (1.26). On the other hand, the optimality of $\tau_R^\sharp$ implies that player 2 can limit his losses to $\text{val}_R(s_0)$ by using strategy $\tau_R^\sharp$. But how player 1 can win at least $\text{val}_L(s_0)$ while player 2 loses no more than $\text{val}_R(s_0)$ if $\text{val}_L(s_0) > \text{val}_R(s_0)$ for $s_0 \in X$? We conclude that the set X is empty and (1.24) holds.

Now our aim is to prove that (1.23) implies that the strategy $\sigma_R^\sharp$ is optimal for player 1 not only in $\mathcal{A}_R$ but also for games on the arena $\mathcal{A}$. Clearly player 1 can secure for himself the payoff of at least $\text{val}_R(s)$ by playing according to $\sigma_R^\sharp$ on $\mathcal{A}$. We should show that he cannot do better. To this end we exhibit a strategy $\tau^\sharp$ for player 2 that limits the losses of player 2 to $\text{val}_R(s)$ on the arena $\mathcal{A}$.

At each stage player 2 will use either his positional strategy $\tau_L^\sharp$ optimal in $\mathcal{A}_L$ or strategy $\tau_R^\sharp$ optimal in $\mathcal{A}_R$. However, in general neither of these strategies is optimal for him in $\mathcal{A}$ and thus it is not a good idea for him to stick to one of these strategies permanently, he should rather adapt his strategy to the moves of his adversary. To implement the strategy $\tau^\sharp$ player 2 will need one bit of memory (the strategy $\tau^\sharp$ we construct here is *not* positional). He uses this memory to remember if at the last passage through the pivot state t player 1 took an action of $A_L(t)$ or an action of $A_R(t)$. In the former case player 2 plays using the strategy $\tau_L^\sharp$, in the latter case he plays using the strategy $\tau_R^\sharp$. In the periods between two passages through t player 2 does not change his strategy, he sticks either to $\tau_L^\sharp$ or to $\tau_R^\sharp$, he switches from one of these strategies to the other only when compelled by the action taken by player 1 during the last visit at the pivot state[5]. It remains to specify which strategy player 2 uses until the first passage through t and we assume that it is the strategy $\tau_R^\sharp$.

Let $s_0 \in S$ be an initial state and let σ be some, not necessarily positional, strategy of 1 for playing on $\mathcal{A}$. Let

$$p(s_0, \sigma, \tau^\sharp) = s_0 s_1 s_2 \dots \tag{1.32}$$

be the resulting play. Our aim is to show that

$$u_{\text{pm}}(p(s_0, \sigma, \tau^\sharp)) \leq \text{val}_R(s_0) \ . \tag{1.33}$$

[5] Note the intuition behind the strategy $\tau^\sharp$: If at the last passage through the pivot state t player 1 took an action of $A_L(t)$ then, at least until the next visit to t, the play is like the one in the game $\mathcal{A}_L$ (all actions taken by the players are actions of $\mathcal{A}_L$) and then it seems reasonable for player 2 to respond with his optimal strategy on $\mathcal{A}_L$. On the other hand, if at the last passage through t player 1 took an action of $A_R(t)$ then from this moment onward until the next visit to t we play like in $\mathcal{A}_R$ and then player 2 will respond with his optimal strategy on $\mathcal{A}_R$.

If $p(s_0, \sigma, \tau^\sharp)$ never goes through t then $p(s_0, \sigma, \tau^\sharp)$ is in fact a play in $\mathcal{A}_R$ consistent with $\tau^\sharp_R$ which immediately implies (1.33).

Suppose now that $p(s_0, \sigma, \tau^\sharp)$ goes through t and let k be the first stage such that $s_k = t$. Then the initial history $s_0 s_1 \ldots s_k$ is consistent with $\tau^\sharp_R$ which, by Lemma 4.2, implies that

$$\mathrm{val}_R(t) \leq \mathrm{val}_R(s_0) \ . \tag{1.34}$$

If there exists a stage n such that $s_n = t$ and player 2 does not change his strategy after this stage[6], i.e. he plays from the stage n onward either $\tau^\sharp_L$ or $\tau^\sharp_R$ then the suffix play $s_n s_{n+1} \ldots$ is consistent with one of these strategies implying that either $u_{\mathrm{pm}}(s_n s_{n+1} \ldots) \leq \mathrm{val}_L(t)$ or $u_{\mathrm{pm}}(s_n s_{n+1} \ldots) \leq \mathrm{val}_R(t)$. But $u_{\mathrm{pm}}(s_n s_{n+1} \ldots) = u_{\mathrm{pm}}(p(s_0, \sigma, \tau^\sharp))$ and thus (1.34) and (1.23) imply (1.33).

The last case to consider is when player 2 switches infinitely often between $\tau^\sharp_R$ and $\tau^\sharp_L$.

In the sequel we say that a non-empty sequence of states z contains only actions of $\mathcal{A}_R$ if for each factorization $z = z' s' s'' z''$ with $s', s'' \in S$, (s', s'') is an action of $\mathcal{A}_R$. (Obviously, there is in a similar definition for $\mathcal{A}_L$.)

Since now we consider the case when the play $p(s_0, \sigma, \tau^\sharp)$ contains infinitely many actions of $A_L(t)$ and infinitely many actions of $A_R(t)$ there exists a unique infinite factorization

$$p(s_0, \sigma, \tau^\sharp) = x_0 x_1 x_2 x_3 \ldots \ , \tag{1.35}$$

such that

- each x_i, $i \geq 1$, is non-empty and begins with the pivot state t,
- each path $x_{2i} t$, $i = 0, 1, 2, \ldots$ contains only actions of $\mathcal{A}_R$ while
- each path $x_{2i+1} t$ contains only actions of $\mathcal{A}_L$.

(Intuitively, we have factorized the play $p(s_0, \sigma, \tau^\sharp)$ according to the strategy used by player 2.)

Let us note that the conditions above imply that

$$x_R = x_2 x_4 x_6 \ldots \quad \text{and} \quad x_L = x_1 x_3 x_5 \ldots \ . \tag{1.36}$$

are infinite paths respectively in $\mathcal{A}_R$ and $\mathcal{A}_L$.

Moreover, x_R is a play consistent with $\tau^\sharp_R$ while x_L is consistent with $\tau^\sharp_L$. By optimality of strategies $\tau^\sharp_R$, $\tau^\sharp_L$,

$$u_{\mathrm{pm}}(x_R) \leq \mathrm{val}_R(t) \quad \text{and} \quad u_{\mathrm{pm}}(x_L) \leq \mathrm{val}_L(t) \ . \tag{1.37}$$

[6] In particular this happens if $p(s_0, \sigma, \tau^\sharp)$ goes finitely often through t.

It is easy to see that path priorities satisfy $\pi(x_R) \geq \pi(p(s_0, \sigma, \tau^\sharp))$ and $\pi(x_L) \geq \pi(p(s_0, \sigma, \tau^\sharp))$ and at most one of these inequalities is strict.

(1) If $\pi(x_R) > \pi(p(s_0, \sigma, \tau^\sharp))$ and $\pi(x_L) = \pi(p(s_0, \sigma, \tau^\sharp))$ then there exists m such that all states in the suffix $x_{2m}x_{2m+2}x_{2m+4}\ldots$ of x_R have priorities greater than $\pi(p(s_0, \sigma, \tau^\sharp))$ and do not contribute to the payoff $u_{\mathrm{pm}}(x_{2m}x_{2m+1}x_{2m+2}x_{2m+3}\ldots)$.

This and the prefix-independence property of u_{pm} imply

$$u_{\mathrm{pm}}(p(s_0, \sigma, \tau^\sharp)) = u_{\mathrm{pm}}(x_{2m}x_{2m+1}x_{2m+2}x_{2m+3}\ldots) = \\ u_{\mathrm{pm}}(x_{2m+1}x_{2m+3}\ldots) = u_{\mathrm{pm}}(x_L) \leq \mathrm{val}_L(s_0) \leq \mathrm{val}_R(s_0),$$

where the first inequality follows from the fact that x_L is consistent with the optimal strategy $\tau_L^\sharp$.

(2) If $\pi(x_L) > \pi(p(s_0, \sigma, \tau^\sharp))$ and $\pi(x_R) = \pi(p(s_0, \sigma, \tau^\sharp))$ then we get in a similar way $u_{\mathrm{pm}}(p(s_0, \sigma, \tau^\sharp)) = u_{\mathrm{pm}}(x_R) \leq \mathrm{val}_R(s_0)$.

(3) Let $a = \pi(x_R) = \pi(p(s_0, \sigma, \tau^\sharp)) = \pi(x_L)$. For a sequence $t_0t_1\ldots t_l$ of states we define

$$F_a(t_0\ldots t_l) = \sum_{i=1}^{l} \mathbf{1}_a(t_i) \cdot w(t_i) \cdot r(t_i)$$

and

$$G_a(t_0\ldots t_l) = \sum_{i=1}^{l} \mathbf{1}_a(t_i) \cdot w(t_i),$$

where $\mathbf{1}_a$ is defined in (1.20). Thus for an infinite path p, $u_{\mathrm{pm}}(p) = \limsup_i F_a(p_i)/G_a(p_i)$, where p_i is the prefix of length i of p.

Take any $\varepsilon > 0$. Eq. (1.37) implies that for all sufficiently long prefixes y_L of x_L, $F_a(y_L)/G_a(y_L) \leq \mathrm{val}_L(t) + \varepsilon \leq \mathrm{val}_R(t) + \varepsilon$ and similarly for all sufficiently long prefixes y_R of x_R, $F_a(y_R)/G_a(y_R) \leq \mathrm{val}_R(t) + \varepsilon$. Then we also have

$$\frac{F_a(y_R) + F_a(y_L)}{G_a(y_R) + G_a(y_L)} \leq \mathrm{val}_R(t) + \varepsilon \quad . \tag{1.38}$$

If y is a proper prefix of the infinite path $x_1x_2x_3\ldots$ then

$$y = x_1x_2\ldots x_{2i-1}x'_{2i}x'_{2i+1} \quad ,$$

where

- either x'_{2i} is a prefix of x_{2i} and x'_{2i+1} is empty or
- $x'_{2i} = x_{2i}$ and x'_{2i+1} is a prefix of x_{2i+1}

(and x_i are as in factorization (1.35)). Then $y_R = x_2x_4 \dots x'_{2i}$ is a prefix of x_R while $y_L = x_1x_3 \dots x_{2i-1}x'_{2i+1}$ is a prefix of x_L. If the length of y tends to ∞ then the lengths of y_R and y_L tend to ∞. Since $G_a(y) = G_a(y_R) + G_a(y_L)$ and $F_a(y) = F_a(y_R)+G_a(y_L)$ Eq. (1.38) implies that $G_a(y)/F_a(y) \leq \mathrm{val}_R(t) + \varepsilon$. Since the last inequality holds for all sufficiently long finite prefixes of $x_1x_2x_3 \dots$ we get that $u_{\mathrm{pm}}(p(s_0, \sigma, \tau^\sharp)) = u_{\mathrm{pm}}(x_1x_2x_3 \dots) \leq \mathrm{val}_R(s_0) + \varepsilon$. As this is true for all $\varepsilon > 0$ we have in fact $u_{\mathrm{pm}}(p(s_0, \sigma, \tau^\sharp)) \leq \mathrm{val}_R(s_0)$.

This finishes the proof that if player 2 plays according to strategy $\tau^\sharp$ then his losses do not extend $\mathrm{val}_R(s_0)$.

We can conclude that strategies $\sigma^\sharp_R$ and $\tau^\sharp$ are optimal on $\mathcal{A}$ and for each initial state s the value of a game on $\mathcal{A}$ is the same as in $\mathcal{A}_R$.

Note however that while player 1 can use his optimal positional strategy $\sigma^\sharp_R$ to play optimally on $\mathcal{A}$ the situation is more complicated for player 2. The optimal strategy that we have constructed for him is not positional and certainly if we pick some of his optimal positional strategies on $\mathcal{A}_R$ then we cannot guarantee that it will remain optimal on $\mathcal{A}$.

To obtain an optimal positional strategy for player 2 we proceed as follows:

If for each state $s \in S_2$ controlled by player 2 there is only one available action then player 2 has only one strategy ($\tau^\sharp_R = \tau^\sharp_L$). Thus in this case player 2 needs no memory.

If there exists a state $t \in S_2$ with at least two available actions then we take this state as the pivot and by the same reasoning as previously we find a pair of optimal strategies $(\sigma^*, \tau^\sharp)$ such that $\tau^\sharp$ is positional while σ^* may need one bit of memory to be implemented.

By exchangeability property of optimal strategies we can conclude that $(\sigma^\sharp, \tau^\sharp)$ is a couple of optimal positional strategies. Q.E.D.

5 Blackwell optimality

Let us return to discounted games. In this section we examine what happens if, for all states s, the discount factors $\lambda(s)$ tend to 1 or, equivalently, the stopping probabilites tend to 0.

When all discount factors are equal and tend to 1 with the same rate then the value of discounted game tends to the value of a simple mean-payoff game, this is a classical result examined extensively by many authors in the context of stochastic games, see [6] and the references therein.

What happens however if discount factors tend to 1 with different rates for different states? To examine this limit we assume in the sequel that arenas for discounted games are equipped not only with a reward mapping $r : S \longrightarrow \mathbb{R}$ but also with a priority mapping $\pi : S \longrightarrow \mathbb{Z}_+$ and a weight mapping $w : S \longrightarrow (0,1]$, exactly as for priority mean-payoff games of Section 4.

Let us take $\beta \in (0,1]$ and assume that the stopping probability of each state s is equal to $w(s)\beta^{\pi(s)}$, i.e. the discount factor is

$$\lambda(s) = 1 - w(s)\beta^{\pi(s)} \ . \tag{1.39}$$

Note that with these discount factors, for two states s and s', $\pi(s) < \pi(s')$ iff $1 - \lambda(s') = o(1 - \lambda(s))$ for $\beta \downarrow 0$.

If (1.39) holds then the payoff mapping (1.9) can be rewritten in the following way, for a play $p = s_0 s_1 s_2 \ldots$,

$$u_\beta(p) = \sum_{i=0}^{\infty} (1 - w(s_0)\beta^{\pi(s_0)}) \ldots (1 - w(s_{i-1})\beta^{\pi(s_{i-1})})\beta^{\pi(s_i)} w(s_i) r(s_i) \ . \tag{1.40}$$

Let us fix a finite arena $\mathcal{A}$. Obviously, it depends on the parameter β which positional strategies are optimal in the games with payoff (1.40). It is remarkable that for β sufficiently close to 0 the optimality of positional strategies does not depend on β any more. This phenomenon was discovered, in the framework of Markov decision processes, by David Blackwell [2] and is now known under the name of Blackwell optimality.

We shall say that positional strategies $(\sigma^\sharp, \tau^\sharp) \in \Sigma \times \mathcal{T}$ are *β-optimal* if they are optimal in the discounted game $(\mathcal{A}, u_\beta)$.

Definition 5.1. Strategies $(\sigma^\sharp, \tau^\sharp) \in \Sigma \times \mathcal{T}$ are Blackwell optimal in a game $(\mathcal{A}, u_\beta)$ if they are β-optimal for all β in an interval $0 < \beta < \beta_0$ for some constant $\beta_0 > 0$ (β_0 depends on the arena $\mathcal{A}$).

Theorem 5.2.

(a) For each arena $\mathcal{A}$ there exists $0 < \beta_0 < 1$ such that if $\sigma^\sharp, \tau^\sharp$ are β-optimal positional strategies for players 1 and 2 for some $\beta \in (0, \beta_0)$ then they are β-optimal for all $\beta \in (0, \beta_0)$, i.e. they are Blackwell optimal.

(b) If $\sigma^\sharp, \tau^\sharp$ are positional Blackwell optimal strategies then they are also optimal for the priority mean-payoff game $(\mathcal{A}, u_{\mathrm{pm}})$.

(c) For each state s, $\lim_{\beta \downarrow 0} \mathrm{val}(\mathcal{A}, s, u_\beta) = \mathrm{val}(\mathcal{A}, s, u_{\mathrm{pm}})$, where $\mathrm{val}(\mathcal{A}, s, u_\beta)$ and $\mathrm{val}(\mathcal{A}, s, u_{\mathrm{pm}})$ are the values of, respectively, the β-discounted game and the priority mean-payoff game.

The remaining part of this section is devoted to the proof of Theorem 5.2.

Lemma 5.3. Let p be an ultimately periodic infinite sequence of states. Then $u_\beta(p)$ is a rational function[7] of β and

$$\lim_{\beta\downarrow 0} u_\beta(p) = u_{\mathrm{pm}}(p) \ . \tag{1.41}$$

Proof. First of all we need to extend the definition (1.40) to finite sequences of states, if $x = s_0 s_1 \dots s_l$ then $u_{\mathrm{pm}}(x)$ is defined like in (1.40) but with the sum taken from 0 to l.

Let $p = xy^\omega$ be an ultimately periodic sequence of states, where x, y are finite sequences of states, y non-empty. Directly from (1.40) we obtain that, for $x = s_0 \dots s_l$,

$$u_\beta(p) = u_\beta(x) + (1 - w(s_0)\beta^{\pi(s_0)}) \dots (1 - w(s_l)\beta^{\pi(s_l)}) u_\beta(y^\omega) \ . \tag{1.42}$$

For any polynomial $f(\beta) = \sum_{i=0}^{l} a_i \beta^i$ the *order*[8] of f is the *smallest* j such that $a_j \neq 0$. By definition the order of the zero polynomial is $+\infty$.

Now note that $u_\beta(x)$ is just a polynomial of β of order strictly greater than 0, which implies that $\lim_{\beta\downarrow 0} u_\beta(x) = 0$. Thus $\lim_{\beta\downarrow 0} u_\beta(p) = \lim_{\beta\downarrow 0} u_\beta(y^\omega)$. On the other hand, $u_{\mathrm{pm}}(p) = u_{\mathrm{pm}}(y^\omega)$. Therefore it suffices to prove that

$$\lim_{\beta\downarrow 0} u_\beta(y^\omega) = u_{\mathrm{pm}}(y^\omega) \ . \tag{1.43}$$

Suppose that $y = t_0 t_1 \dots t_k$, $t_i \in S$. Then

$$u_\beta(y^\omega) = u_\beta(y) \sum_{i=0}^{\infty} [(1 - w(t_0)\beta^{\pi(t_0)}) \cdots (1 - w(t_k)\beta^{\pi(t_k)})]^i = \frac{u_\beta(y)}{1 - (1 - w(t_0)\beta^{\pi(t_0)}) \cdots (1 - w(t_k)\beta^{\pi(t_k)})} \ . \tag{1.44}$$

Let $a = \min\{\pi(t_i) \mid 0 \leq i \leq k\}$ be the priority of y, $L = \{l \mid 0 \leq l \leq k \ \text{ and } \ \pi(t_l) = a\}$. Now it suffices to observe that the right hand side of (1.44) can be rewritten as

$$u_\beta(y^\omega) = \frac{\sum_{l\in L} w(t_l) r(t_l) \beta^a + f(\beta)}{\sum_{l\in L} w(t_l)\beta^a + g(\beta)} \ ,$$

where f and g are polynomials of order greater than a. Therefore

$$\lim_{\beta\downarrow 0} u_\beta(y^\omega) = \frac{\sum_{l\in L} w(t_l) r(t_l)}{\sum_{l\in L} w(t_l)} \ . \tag{1.45}$$

However, the right hand side of (1.45) is the value of $u_{\mathrm{pm}}(y^\omega)$. Q.E.D.

[7] The quotient of two polynomials.

[8] Not to be confounded with the degre of f which is te greatest j such that $a_j \neq 0$.

Proof of Theorem 5.2. The proof of condition (a) given below follows very closely the one given in [13] for Markov decision processes.

Take a sequence (β_n), $\beta_n \in (0,1]$, such that $\lim_{n\to\infty}\beta_n = 0$. Since for each β_n there is at least one pair of β_n-optimal positional strategies and there are only finitely many positional strategies for a finite arena $\mathcal{A}$, passing to a subsequence of (β_n) if necessary, we can assume that there exists a pair of positional strategies $(\sigma^\sharp, \tau^\sharp)$ that are β_n-optimal for all β_n.

We claim that there exists $\beta_0 > 0$ such that $(\sigma^\sharp, \tau^\sharp)$ are β-optimal for all $0 < \beta < \beta_0$.

Suppose the contrary. Then there exists a state s and a sequence (γ_m), $\gamma_m \in (0,1]$, such that $\lim_{m\to\infty}\gamma_m = 0$ and, for each m, either $\sigma^\sharp$ or $\tau^\sharp$ is not γ_m-optimal. Therefore, for each m,

(i) either player 1 has a strategy $\sigma^\star_m$ such that

$$u_{\gamma_m}(p(s,\sigma^\sharp,\tau^\sharp)) < u_{\gamma_m}(p(s,\sigma^\star_m,\tau^\sharp)),$$

(ii) or player 2 has a strategy $\tau^\star_m$ such that

$$u_{\gamma_m}(p(s,\sigma^\sharp,\tau^\star_m)) < u_{\gamma_m}(p(s,\sigma^\sharp,\tau^\sharp)).$$

Due to Lemma 2.2, all the strategies $\sigma^\star_m$ and $\tau^\star_m$ can be chosen to be positional and since the number of positional strategies is finite, taking a subsequence of (γ_m) if necessary, we can assume that

(1) either there exist a state s, a positional strategy $\sigma^\star \in \Sigma_p$ and a sequence (γ_m), $\gamma_m \downarrow 0$, such that

$$u_\beta(p(s,\sigma^\sharp,\tau^\sharp)) < u_\beta(p(s,\sigma^\star,\tau^\sharp)) \quad \text{for all } \beta = \gamma_1,\gamma_2,\dots\ , \tag{1.46}$$

(2) or there exist a state s, a positional strategy $\tau^\star \in \mathcal{T}_p$ and a sequence (γ_m), $\gamma_m \downarrow 0$, such that

$$u_\beta(p(s,\sigma^\sharp,\tau^\star)) < u_\beta(p(s,\sigma^\sharp,\tau^\sharp)) \quad \text{for all } \beta = \gamma_1,\gamma_2,\dots\ . \tag{1.47}$$

Suppose that (1.46) holds.
The choice of $(\sigma^\sharp, \tau^\sharp)$ guarantees that

$$u_\beta(p(s,\sigma^\star,\tau^\sharp)) \le u_\beta(p(s,\sigma^\sharp,\tau^\sharp)) \quad \text{for all } \beta = \beta_1,\beta_2,\dots\ . \tag{1.48}$$

Consider the function

$$f(\beta) = u_\beta(p(s,\sigma^\star,\tau^\sharp)) - u_\beta(p(s,\sigma^\sharp,\tau^\sharp)). \tag{1.49}$$

By Lemma 5.3, for $0 < \beta < 1$, $f(\beta)$ is a rational function of β. But from (1.46) and (1.48) we can deduce that when β tends to 0 then $f(\beta) \leq 0$ infinitely often and $f(\beta) > 0$ infinitely often. This is possible for a rational function f only if this function is identicaly equal to 0, contradicting (1.46). In a similar way we can prove that (1.47) entails a contradiction. We conclude that $\sigma^\sharp$ and $\tau^\sharp$ are Blackwell optimal.

To prove condition (b) of Theorem 5.2 suppose the contrary, i.e. that there are positional Blackwell optimal strategies $(\sigma^\sharp, \tau^\sharp)$ that are not optimal for the priority mean-payoff game. This means that there exists a state s such that either

$$u_{\mathrm{pm}}(p(s, \sigma^\sharp, \tau^\sharp)) < u_{\mathrm{pm}}(p(s, \sigma, \tau^\sharp)) \tag{1.50}$$

for some strategy σ of player 1 or

$$u_{\mathrm{pm}}(p(s, \sigma^\sharp, \tau)) < u_{\mathrm{pm}}(p(s, \sigma^\sharp, \tau^\sharp)) \tag{1.51}$$

for some strategy τ of player 2. Since priority mean-payoff games have optimal positional strategies, by Lemma 2.2, we can assume without loss of generality that σ and τ are positional. Suppose that (1.50) holds. As $\sigma, \sigma^\sharp, \tau^\sharp$ are positional the plays $p(s, \sigma^\sharp, \tau^\sharp)$ and $p(s, \sigma, \tau^\sharp)$ are ultimately periodic, by Lemma 5.3, we get

$$\begin{aligned} \lim_{\beta \downarrow 0} u_\beta(p(s, \sigma^\sharp, \tau^\sharp)) &= u_{\mathrm{pm}}(p(s, \sigma^\sharp, \tau^\sharp)) \\ &< u_{\mathrm{pm}}(p(s, \sigma, \tau^\sharp)) = \lim_{\beta \downarrow 0} u_\beta(p(s, \sigma, \tau^\sharp)). \end{aligned} \tag{1.52}$$

However, inequality (1.52) implies that there exists $0 < \beta_0$ such that

$$\text{for all } \beta < \beta_0, \quad u_\beta(p(s, \sigma^\sharp, \tau^\sharp)) < u_\beta(p(s, \sigma, \tau^\sharp)) \ ,$$

in contradiction with the Blackwell optimality of $(\sigma^\sharp, \tau^\sharp)$. Similar reasoning shows that also (1.51) contradicts the Blackwell optimality of $(\sigma^\sharp, \tau^\sharp)$. i This also shows that

$$\lim_{\beta \downarrow 0} \mathrm{val}(\mathcal{A}, s, u_\beta) = \lim_{\beta \downarrow 0} u_\beta(p(s, \sigma^\sharp, \tau^\sharp)) = u_{\mathrm{pm}}(p(s, \sigma^\sharp, \tau^\sharp)) = \mathrm{val}(\mathcal{A}, s, u_{\mathrm{pm}}),$$

i.e., condition (c) of Theorem 5.2 holds as well. Q.E.D.

Let us note that there is another known link between parity and discounted games: Jurdziński [14] has shown how parity games can be reduced to mean-payoff games and it is well-known that the value of mean-payoff games is a limit of the value of discounted games, see [15] or [21] for the particular case of deterministic games. However, the reduction of [14] does not seem to extend to priority mean-payoff games and, more significantly, it also fails for perfect information stochastic games. Note also that [21] concentrates only on value approximation and the issue of Blackwell optimality of strategies in not touched at all.

6 Final remarks

6.1 Interpretation of infinite games

In real life all systems have a finite life span: computer systems become obsolete, economic environment changes. Therefore it is reasonable to ask if infinite games are pertinent as models of such systems. This question is discussed for example in [18].

If there exists a family of payoff mappings u_n such that $u_n : S^n \longrightarrow \mathbb{R}$ is defined for paths of length n (n-stage payoff) and the payoff $u(s_0 s_1 \ldots)$ for an infinite play is a limit of $u_n(s_0 s_1 \ldots s_{n-1})$ when the number of stages n tends to ∞ then we can say that infinite games are just approximations of finite games where the length of the game is very large or not precisely known. This interpretation is quite reasonable for simple mean-payoff games for example, where the payoff for infinite plays is a limit of n stage mean-payoff. However such an interpretation fails for priority mean-payoff games and for parity games where no appropriate n-stage payoff mappings exist.

However the stopping (or discounted) games offer another attractive probabilistic interpretation of priority mean-payoff games. For sufficiently small β if we consider a stopping game with the stopping probabilities $w(s)\beta^{\pi(s)}$ for each state s then Theorem 5.2 states that optimal positional strategies for the stopping game are optimal for the priority mean-payoff game. Moreover, the value of the stopping game tends to the value of the priority mean-payoff game when β tends to 0. And the stopping game is a finite game but in a probabilistic rather than deterministic sense, such a game stops with probability 1. Thus we can interpret infinite priority mean-payoff games as an approximation of stopping games where the stopping probabilities are very small. We can also see that smaller priorities are more significant since the corresponding stopping probabilities are much greater: $w(s)\beta^{\pi(s)} = o(w(t)\beta^{\pi(t)})$ if $\pi(s) > \pi(t)$.

6.2 Refining the notion of optimal strategies for priority mean-payoff games

Optimal strategies for parity games (and generally for priority mean-payoff games) are under-selective. To illustrate this problem let us consider the game of Figure 6.2.

For this game all strategies of player 1 guarantee him the payment 1. Suppose however that the left loop contains $2^{1000000}$ states while the right loop only 3 states. Then, intuitively, it seems that the positional strategy choosing always the small right loop is much more advantageous for player 1 than the positional strategy choosing always the big left loop. But with the traditional definition of optimality for parity games one strategy is as good as the other.

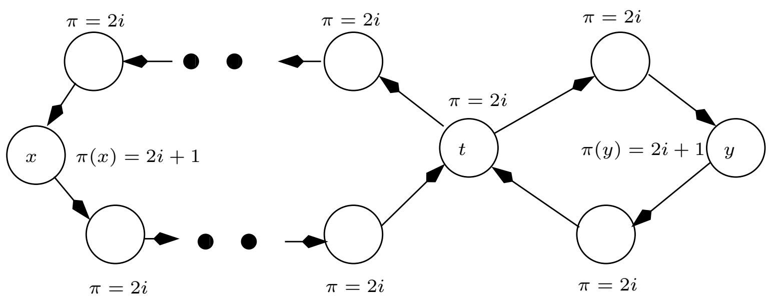


FIGURE 1. The left and the right loop contain one state, x and y respectively, with priority $2i + 1$, all the other states have priority $2i$. The weight of all states is 1. The reward for x and for y is 1 and 0 for all the other states. This game is in fact a parity (Büchi) game, player 1 gets payoff 1 if one of the states $\{x, y\}$ is visited infinitely often and 0 otherwise.

On the other hand, Blackwell optimality clearly distinguishes both strategies, the discounted payoff associated with the right loop is strictly greater than the payoff for the left loop.

Let us note that under-selectiveness of simple mean-payoff games originally motivated the introduction of the Blackwell's optimality criterion [2]. Indeed, the infinite sequence of rewards $100, 0, 0, 0, 0, \ldots$ gives, at the limit, the mean-payoff 0, the same as an infinite sequence of 0. However it is clear that we prefer to get once 100 even if it is followed by an infinite sequence of 0 than to get 0 all the time.

6.3 Evaluating β_0.

Theorem 5.2 is purely existential and does not provide any evaluation of the constant β_0 appearing there. However it is not difficult to give an elementary estimation for β_0, at least for deterministic games considered in this paper. We do not do it here since the bound for β_0 obtained this way does not seem to be particularly enlightening.

The preceding subsection discussing the meaning of the Blackwell optimality raises the question what is the complexity of finding Blackwell optimal strategies. This question remains open. Note that if we can find efficiently Blackwell optimal strategies then we can obviously find efficiently optimal strategies for priority mean-payoff games and, in particular, for parity games. But the existence of a polynomial time algorithm solving parity games is a well-known open problem.

References

[1] H. Björklund, S. Sandberg, and S. G. Vorobyov. Memoryless determinacy of parity and mean payoff games: a simple proof. *Theor. Comput. Sci.*, 310(1-3):365–378, 2004.

[2] D. Blackwell. Discrete dynamic programming. *Ann. Math. Statist.*, 33:719–726, 1962.

[3] L. de Alfaro, T. A. Henzinger, and R. Majumdar. Discounting the future in systems theory. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 1022–1037. Springer, 2003.

[4] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *Internat. J. Game Theory*, 8(2):109–113, 1979.

[5] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, pages 368–377. IEEE, 1991.

[6] J. Filar and K. Vrieze. *Competitive Markov decision processes.* Springer-Verlag, New York, 1997.

[7] H. Gimbert and W. Zielonka. When can you play positionally? In J. Fiala, V. Koubek, and J. Kratochvíl, editors, *MFCS*, volume 3153 of *Lecture Notes in Computer Science*, pages 686–697. Springer, 2004.

[8] H. Gimbert and W. Zielonka. Deterministic priority mean-payoff games as limits of discounted games. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 312–323. Springer, 2006.

[9] H. Gimbert and W. Zielonka. Limits of multi-discounted markov decision processes. In *LICS*, pages 89–98. IEEE Computer Society, 2007.

[10] H. Gimbert and W. Zielonka. Perfect information stochastic priority games. In L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, editors, *ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 850–861. Springer, 2007.

[11] E. Grädel, W. Thomas, and Th. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[12] Y. Gurevich and L. Harrington. Trees, automata, and games. In *STOC*, pages 60–65. ACM, 1982.

[13] A. Hordijk and A. A. Yushkevich. Blackwell optimality. In *Handbook of Markov decision processes*, volume 40 of *Internat. Ser. Oper. Res. Management Sci.*, pages 231–267. Kluwer Acad. Publ., Boston, MA, 2002.

[14] M. Jurdzinski. Deciding the winner in parity games is in up $\cap$ co-up. *Inf. Process. Lett.*, 68(3):119–124, 1998.

[15] J.-F. Mertens and A. Neyman. Stochastic games. *Internat. J. Game Theory*, 10(2):53–66, 1981.

[16] A. Mostowski. Games with forbidden positions. Technical Report 78, Uniwersytet Gdański, Instytut Matematyki, 1991.

[17] A. Neyman. From Markov chains to stochastic games. In *Stochastic games and applications (Stony Brook, NY, 1999)*, volume 570 of *NATO Sci. Ser. C Math. Phys. Sci.*, pages 9–25. Kluwer Acad. Publ., Dordrecht, 2003.

[18] M. J. Osborne and A. Rubinstein. *A course in game theory.* MIT Press, Cambridge, MA, 1994.

[19] L. S. Shapley. Stochastic games. *Proceedings Nat. Acad. of Science USA*, 39:1095–1100, 1953.

[20] W. Zielonka. An invitation to play. In J. Jedrzejowicz and A. Szepietowski, editors, *MFCS*, volume 3618 of *Lecture Notes in Computer Science*, pages 58–70. Springer, 2005.

[21] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996.

Logic, graphs, and algorithms[*]

Martin Grohe

Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
grohe@informatik.hu-berlin.de

Abstract

Algorithmic meta theorems are algorithmic results that apply to whole families of combinatorial problems, instead of just specific problems. These families are usually defined in terms of logic and graph theory. An archetypal algorithmic meta theorem is Courcelle's Theorem [9], which states that all graph properties definable in monadic second-order logic can be decided in linear time on graphs of bounded tree width.

This article is an introduction into the theory underlying such meta theorems and a survey of the most important results in this area.

1 Introduction

In 1990, Courcelle [9] proved a fundamental theorem stating that graph properties definable in monadic second-order logic can be decided in linear time on graphs of bounded tree width. This is the first in a series of *algorithmic meta theorems*. More recent examples of such meta theorems state that all first-order definable properties of planar graphs can be decided in linear time [42] and that all first-order definable optimisation problems on classes of graphs with excluded minors can be approximated in polynomial time to any given approximation ratio [19]. The term "meta theorem" refers to the fact that these results do not describe algorithms for specific problems, but for whole families of problems, whose definition typically has a logical and a structural (usually graph theoretical) component. For example, Courcelle's Theorem is about *monadic second-order logic* on *graphs of bounded tree width*.

[*] I would like to thank Bruno Courcelle, Arnaud Durand, Sang-Il Oum, Stéphan Thomassé for patiently answering various questions I had while writing this survey. Thanks to Isolde Adler, Albert Atserias, Yijia Chen, Anuj Dawar, Reinhard Diestel, Jörg Flum, Magdalena Grüber, Stephan Kreutzer, Nicole Schweikardt for valuable comments on earlier drafts of the survey.

This article is an introductory survey on algorithmic meta theorems. Why should we care about such theorems? First of all, they often provide a quick way to prove that a problem is solvable efficiently. For example, to show that the 3-colourability problem can be solved in linear time on graphs of bounded tree width, we observe that 3-colourability is a property of graphs definable in monadic second-order logic and apply Courcelle's theorem. Secondly, and more substantially, algorithmic meta theorems yield a better understanding of the scope of general algorithmic techniques and, in some sense, the limits of tractability. In particular, they clarify the interactions between logic and combinatorial structure, which is fundamental for computational complexity.

The general form of algorithmic meta theorems is:

> All *problems* definable in a certain *logic* on a certain class of *structures* can be solved *efficiently.*

Problems may be of different types, for example, they may be optimisation or counting problems, but in this article we mainly consider decision problems. We briefly discuss other types of problems in Section 7.2. *Efficient solvability* may mean, for example, polynomial time solvability, linear or quadratic time solvability, or fixed-parameter tractability. We shall discuss this in detail in Section 2.3. Let us now focus on the two main ingredients of the meta theorems, logic and structure.

The two *logics* that, so far, have been considered almost exclusively for meta theorems are first-order logic and monadic second-order logic. Techniques from logic underlying the theorems are Feferman-Vaught style composition lemmas, automata theoretic techniques, and locality results such as Hanf's Theorem and Gaifman's Theorem.

The *structures* in algorithmic meta theorems are usually defined by graph theoretic properties. Actually, to ease the presentation, the only structures we shall consider in this survey are graphs. Many of the meta theorems are tightly linked with *graph minor theory.* This deep theory, mainly developed by Robertson and Seymour in a long series of papers, describes the structure of graphs with excluded minors. It culminates in the graph minor theorem [75], which states that every class of graphs closed under taking minors can be characterised by a finite set of excluded minors. The theory also has significant algorithmic consequences. Robertson and Seymour [73] proved that every class of graphs that is closed under taking minors can be recognised in cubic time. More recently, results from graph minor theory have been combined with algorithmic techniques that had originally been developed for planar graphs to obtain polynomial time approximation schemes and fixed parameter tractable algorithms for many standard optimisation problems on families of graphs with excluded minors. The methods developed in this context are also underlying the more advanced algorithmic meta theorems.

There are some obvious similarities between algorithmic meta theorems and results from *descriptive complexity theory*, in particular such results from descriptive complexity theory that also involve restricted classes of graphs. As an example, consider the theorem stating that fixed-point logic with counting captures polynomial time on graphs of bounded tree width [49], that is, a property of graphs of bounded tree width is definable in fixed-point logic with counting if and only if it is decidable in polynomial time. Compare this to Courcelle's Theorem. Despite the similarity, there are two crucial differences: On the one hand, Courcelle's Theorem is weaker as it makes no completeness claim, that is, it does not state that *all* properties of graphs of bounded tree width that are decidable in linear time are definable in monadic second-order logic. On the other hand, Courcelle's Theorem is stronger in its algorithmic content. Whereas it is very easy to show that all properties of graphs (not only graphs of bounded tree width) definable in fixed-point logic with counting are decidable in polynomial time, the proof of Courcelle's theorem relies on substantial algorithmic ideas like the translation of monadic second-order logic over trees into tree automata [80] and a linear time algorithm for computing tree decompositions [5]. In general, algorithmic meta theorems involve nontrivial algorithms, but do not state completeness, whereas in typical results from descriptive complexity, the algorithmic content is limited, and the nontrivial part is completeness. But there is no clear dividing line. Consider, for example, Papadimitriou and Yannakakis's [66] well known result that all optimisation problems in the logically defined class MAXSNP have a constant factor approximation algorithm. This theorem does not state completeness, but technically it is much closer to Fagin's Theorem [36], a central result of descriptive complexity theory, than to the algorithmic meta theorems considered here. In any case, both algorithmic meta theorems and descriptive complexity theory are branches of finite model theory, and there is no need to draw a line between them.

When I wrote this survey, it was my goal to cover the developments up to the most recent and strongest results, which are concerned with monadic second-order logic on graphs of bounded rank width and with first-order logic on graphs with excluded minors. The proofs of most theorems are at least sketched, so that we hope that the reader will not only get an impression of the results, but also of the techniques involved in their proofs.

2 The basics

$\mathbb{R}$, $\mathbb{Q}$, $\mathbb{Z}$, and $\mathbb{N}$ denote the sets of real numbers, rational numbers, integers, and natural numbers (that is, positive integers), respectively. For a set

$S \subseteq \mathbb{R}$, by $S_{\geq 0}$ we denote the set of nonnegative numbers in S. For integers m, n, by $[m, n]$ we denote the interval $\{m, m+1, \ldots, n\}$, which is empty if $n < m$. Furthermore, we let $[n] = [1, n]$. The power set of a set S is denoted by 2^S, and the set of all k-element subsets of S by $\binom{S}{k}$.

2.1 Graphs

A *graph* G is a pair $(V(G), E(G))$, where $V(G)$ is a finite set whose elements are called *vertices* and $E(G) \subseteq \binom{V(G)}{2}$ is a set of unordered pairs of vertices, which are called *edges*. Hence graphs in this paper are always *finite*, *undirected*, and *simple*, where simple means that there are no loops or parallel edges. If $e = \{u, v\}$ is an edge, we say that the vertices u and v are *adjacent*, and that both u and v are *incident* with e. A graph H is a *subgraph* of a graph G (we write $H \subseteq G$) if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If $E(H) = E(G) \cap \binom{V(H)}{2}$, then H is an *induced* subgraph of G. For a set $W \subseteq V(G)$, we write $G[W]$ to denote the induced subgraph $\big(W, E(G) \cap \binom{W}{2}\big)$ and $G \setminus W$ to denote $G[V(G) \setminus W]$. For a set $F \subseteq E$, we let $G[\![F]\!]$ be the subgraph $\big(\bigcup F, F\big)$. Here $\bigcup F$ denote the union of all edges in F, that is, the set of all vertices incident with at least one edge in F. We call $G[\![F]\!]$ the subgraph of G *generated* by F; note that it is not necessarily an induced subgraph of G. The *union* of two graphs G and H is the graph $G \cup H = (V(G) \cup V(H), E(G) \cup E(H))$, and the *intersection* $G \cap H$ is defined similarly. The *complement* of a graph $G = (V, E)$ is the graph $\overline{G} = \big(V, \binom{V}{2} \setminus E\big)$. There is a unique *empty graph* $(\emptyset, \emptyset)$. For $n \geq 1$, we let K_n be the complete graph with n vertices. To be precise, let us say $K_n = \big([n], \binom{[n]}{2}\big)$. We let $K_{n,m}$ be the complete bipartite graph with parts of size m, n, respectively.

Occasionally, we consider (vertex) labelled graphs. A *labelled graph* is a tuple

$$G = \big(V(G), E(G), P_1(G), \ldots, P_\ell(G)\big),$$

where $P_i(G) \subseteq V(G)$ for all $i \in [\ell]$. The symbols P_i are called *labels*, and if $v \in P_i(G)$ we say that v is *labelled* by P_i. Subgraphs, union, and intersection extend to labelled graphs in a straightforward manner. The *underlying graph* of a labelled graph G is $(V(G), E(G))$. Whenever we apply graph theoretic notions such as connectivity to labelled graphs, we refer to the underlying graph.

The *order* $|G|$ of a graph G is the number of vertices of G. We usually use the letter n to denote the order of a graph. The *size* of G is the number $||G|| = |G| + |E(G)|$. Up to a constant factor, this is the size of the adjacency list representation of G under a uniform cost model.

$\mathcal{G}$ denotes the class of all graphs. For every class $\mathcal{C}$ of graphs, we let $\mathcal{C}_{lb}$ be the class of all labelled graphs whose underlying graph is in $\mathcal{C}$. A *graph invariant* is a mapping defined on the class $\mathcal{G}$ of all graphs that is invariant under isomorphisms. All graph invariants considered in this paper are integer valued. For a graph invariant $f : \mathcal{G} \to \mathbb{Z}$ and a class $\mathcal{C}$ of graphs, we say that $\mathcal{C}$ *has bounded* f if there is a $k \in \mathbb{Z}$ such that $f(G) \leq k$ for all $G \in \mathcal{C}$.

Let $G = (V, E)$ be a graph. The *degree* $\deg^G(v)$ of a vertex $v \in V$ is the number of edges incident with v. We omit the superscript ^{G} if G is clear from the context. The *(maximum) degree* of G is the number

$$\Delta(G) = \max\{\deg(v) \mid v \in V\}.$$

The *minimum degree* $\delta(G)$ is defined analogously, and the *average degree* $d(G)$ is $2|E(G)|/|V(G)|$. Observe that $||G|| = O(d(G) \cdot |G|)$. Hence if a class $\mathcal{C}$ of graphs has bounded average degree, then the size of the graphs in $\mathcal{C}$ is linearly bounded in the order. In the following, "degree" of a graph, without qualifications, always means "maximum degree".

A *path* in $G = (V, E)$ of *length* $n \geq 0$ *from* a vertex v_0 *to* a vertex v_n is a sequence $v_0, \ldots, v_n$ of distinct vertices such that $\{v_{i-1}, v_i\} \in E$ for all $i \in [n]$. Note that the length of a path is the number of edges on the path. Two paths are *disjoint* if they have no vertex in common. G is *connected* if it is nonempty and for all $v, w \in V$ there is a path from v to w. A *connected component* of G is a maximal (with respect to $\subseteq$) connected subgraph. G is k*-connected*, for some $k \geq 1$, if $|V| > k$ and for every $W \subseteq V$ with $|W| < k$ the graph $G \setminus W$ is connected.

A *cycle* in a graph $G = (V, E)$ of *length* $n \geq 3$ is a sequence $v_1 \ldots v_n$ of distinct vertices such that $\{v_n, v_1\} \in E$ and $\{v_{i-1}, v_i\} \in E$ for all $i \in [2, n]$. A graph G is *acyclic*, or a *forest*, if it has no cycle. G is a *tree* if it is acyclic and connected. It will be a useful convention to call the vertices of trees *nodes*. A node of degree at most 1 is called a *leaf*. The set of all leaves of a tree T is denoted by $L(T)$. Nodes that are not leaves are called *inner nodes*. A *rooted tree* is a triple $T = (V(T), E(T), r(T))$, where $(V(T), E(T))$ is a tree and $r(T) \in V(T)$ is a distinguished node called the *root*. A node t of a rooted tree T is the *parent* of a node u, and u is a *child* of t, if t is the predecessor of u on the unique path from the root $r(T)$ to u. Two nodes that are children of the same parent are called *siblings*. A *binary tree* is a rooted tree T in which every node has either no children at all or exactly two children.

2.2 Logic

I assume that the reader has some background in logic and, in particular, is familiar with first-order predicate logic. To simplify matters, we only consider logics over (labelled) graphs, even though most results mentioned in this survey extend to more general structures. Let us briefly review the syntax and semantics of *first-order logic* FO and *monadic second-order logic*

MSO. We assume that we have an infinite supply of *individual variables*, usually denoted by the lowercase letters x, y, z, and an infinite supply of *set variables*, usually denoted by uppercase letters X, Y, Z. *First-order formulas* in the language of graphs are built up from atomic formulas $E(x, y)$ and $x = y$ by using the usual Boolean connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), and $\leftrightarrow$ (bi-implication) and existential quantification $\exists x$ and universal quantification $\forall x$ over individual variables. Individual variables range over vertices of a graph. The atomic formula $E(x, y)$ expresses adjacency, and the formula $x = y$ expresses equality. From this, the semantics of first-order logic is defined in the obvious way. First-order formulas over labelled graphs may contain additional atomic formulas $P_i(x)$, meaning that x is labelled by P_i. If a label P_i does not appear in a labelled graph G, then we always interpret $P_i(G)$ as the empty set. In *monadic second-order formulas*, we have additional atomic formulas $X(x)$ for set variables X and individual variables x, and we admit existential and universal quantification over set variables. Set variables are interpreted by sets of vertices, and the atomic formula $X(x)$ means that the vertex x is contained in the set X.

The *free* individual and set variables of a formula are defined in the usual way. A *sentence* is a formula without free variables. We write $\varphi(x_1, \ldots, x_k, X_1, \ldots, X_\ell)$ to indicate that φ is a formula with free variables among $x_1, \ldots, x_k$, $X_1, \ldots, X_\ell$. We use this notation to conveniently denote substitutions and assignments to the variables. If $G = (V, E)$ is a graph, $v_1, \ldots, v_k \in V$, and $W_1, \ldots, W_\ell \subseteq V$, then we write $G \models \varphi(v_1, \ldots, v_k, W_1, \ldots, W_\ell)$ to denote that $\varphi(x_1, \ldots, x_k, X_1, \ldots, X_\ell)$ holds in G if the variables x_i are interpreted by the vertices v_i and the variables X_i are interpreted by the vertex sets W_i.

Occasionally, we consider monadic second-order formulas that contain no second-order quantifiers, but have free set variables. We view such formulas as first-order formulas, because free set variables are essentially the same as labels (unary relation symbols). An example of such a formula is the formula $\mathrm{dom}(X)$ in Example 2.1 below. We say that a formula $\varphi(X)$ is *positive in* X if X only occurs in the scope of an even number of negation symbols. It is *negative in* X if X only occurs in the scope of an odd number of relation symbols. We freely use abbreviations such as $\bigwedge_{i=1}^{k} \varphi_i$ instead of $(\varphi_1 \wedge \ldots \wedge \varphi_k)$ and $x \neq y$ instead of $\neg x = y$.

Example 2.1. A *dominating set* in a graph $G = (V, E)$ is a set $S \subseteq V$ such that for every $v \in V$, either v is in S or v is adjacent to a vertex in S.
The following first-order sentence dom_k says that a graph has a dominating

set of size k:

$$\mathrm{dom}_k = \exists x_1 \dots \exists x_k \big(\bigwedge_{1 \le i < j \le k} x_i \ne x_j \wedge \forall y \bigvee_{i=1}^{k} \big(y = x_i \vee E(y, x_i)\big)\big).$$

The following formula $\mathrm{dom}(X)$ says that X is a dominating set:

$$\mathrm{dom}(X) = \forall y \Big(X(y) \vee \exists z \big(X(z) \wedge E(z, y) \big) \Big).$$

More precisely, for every graph G and every subset $S \subseteq V(G)$ it holds that $G \models \mathrm{dom}(S)$ if and only if S is a dominating set of G.

Example 2.2. The following monadic second-order sentences *conn* and *acyc* say that a graph is connected and acyclic, respectively:

$$\begin{aligned} \mathit{conn} = \exists x\, x = x \wedge \forall X \Big(\big(\exists x\, X(x) \wedge \forall x \forall y \big((X(x) \wedge E(x,y)) \to X(y) \big) \big) \\ \to \forall x X(x) \Big), \\ \mathit{acyc} = \neg \exists X \Big(\exists x\, X(x) \wedge \forall x \big(X(x) \\ \to \exists y_1 \exists y_2 \big(y_1 \ne y_2 \wedge E(x, y_1) \wedge E(x, y_2) \wedge X(y_1) \wedge X(y_2) \big) \big) \Big). \end{aligned}$$

The sentence *acyc* is based on the simple fact that a graph has a cycle if and only if it has a nonempty induced subgraph in which every vertex has degree at least 2. Then the sentence *tree* = *conn* $\wedge$ *acyc* says that a graph is a tree.

The *quantifier rank* of a first-order or monadic second-order formula φ is the nesting depth of quantifiers in φ. For example, the quantifier rank of the formula *acyc* in Example 2.2 is 4. Let G be a graph and $\bar{v} = (v_1, \dots, v_k) \in V(G)^k$, for some nonnegative integer k. For every $q \ge 0$, the *first-order q-type of $\bar{v}$ in G* is the set $\mathrm{tp}_q^{\mathrm{FO}}(G, \bar{v})$ of all first-order formulas $\varphi(x_1, \dots, x_k)$ of quantifier rank at most q such that $G \models \varphi(v_1, \dots, v_k)$. The *monadic second-order q-type of $\bar{v}$ in G*, $\mathrm{tp}_q^{\mathrm{MSO}}(G, \bar{v})$ is defined analogously. As such, types are infinite sets, but we can syntactically *normalise* formulas in such a way that there are only finitely many normalised formulas of fixed quantifier rank and with a fixed set of free variables, and that every formula can effectively be transformed into an equivalent normalised formula of the same quantifier rank. We represent a type by the set of normalised formulas it contains. There is a fine line separating decidable and undecidable properties of types and formulas. For example, it is decidable whether a formula is contained in a type: We just normalise the formula and test if it is equal to one of the normalised formulas in the type. It is undecidable whether a set of

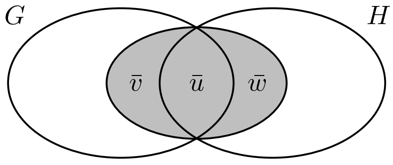


FIGURE 2.1. An illustration of Lemma 2.3

normalised formulas actually is (more precisely: represents) a type. To see this, remember that types are satisfiable by definition and that the satisfiability of first-order formulas is undecidable.

For a tuple $\bar{v} = (v_1, \dots, v_k)$, we sloppily write $\{\bar{v}\}$ to denote the set $\{v_1, \dots, v_k\}$. It will always be clear from the context whether $\{\bar{v}\}$ refers to the set $\{v_1, \dots, v_k\}$ or the 1-element set $\{(v_1, \dots, v_k)\}$. For tuples $\bar{v} = (v_1, \dots, v_k)$ and $\bar{w} = (w_1, \dots, w_\ell)$, we write $\bar{v}\bar{w}$ to denote their concatenation $(v_1, \dots, v_k, w_1, \dots, w_\ell)$. We shall heavily use the following "Feferman-Vaught style" composition lemma.

Lemma 2.3. Let tp be one of $\mathrm{tp}^{\mathrm{FO}}, \mathrm{tp}^{\mathrm{MSO}}$. Let G, H be labelled graphs and $\bar{u} \in V(G)^k$, $\bar{v} \in V(G)^\ell$, $\bar{w} \in V(H)^m$ such that $V(G) \cap V(H) = \{\bar{u}\}$ (cf. Figure 2.1). Then for all $q \geq 0$, $\mathrm{tp}_q(G \cup H, \bar{u}\bar{v}\bar{w})$ is determined by $\mathrm{tp}_q(G, \bar{u}\bar{v})$ and $\mathrm{tp}_q(H, \bar{u}\bar{w})$. Furthermore, there is an algorithm that computes $\mathrm{tp}_q(G \cup H, \bar{u}\bar{v}\bar{w})$ from $\mathrm{tp}_q(G, \bar{u}\bar{v})$ and $\mathrm{tp}_q(H, \bar{u}\bar{w})$.

Let me sketch a proof of this lemma for first-order types. The version for monadic second-order types can be proved similarly, but is more complicated (see, for example, [58]).

Proof sketch. Let G, H be labelled graphs and $\bar{u} \in V(G)^k$ such that $V(G) \cap V(H) = \{\bar{u}\}$. By induction on φ, we prove the following claim:

Claim: Let $\varphi(\bar{x}, \bar{y}, \bar{z})$ be a first-order formula of quantifier rank q, where $\bar{x}$ is a k-tuple and $\bar{y}$, $\bar{z}$ are tuples of arbitrary length. Then there is a Boolean combination $\Phi(\bar{x}, \bar{y}, \bar{z})$ of expressions $G \models \psi(\bar{x}, \bar{y})$ and $H \models \chi(\bar{x}, \bar{z})$ for formulas ψ, χ of quantifier rank at most q, such that for all tuples $\bar{v}$ of vertices of G and $\bar{w}$ of vertices of H of the appropriate lengths it holds that

$$G \cup H \models \varphi(\bar{u}, \bar{v}, \bar{w}) \iff \Phi(\bar{u}, \bar{v}, \bar{w}).$$

Here $\Phi(\bar{u}, \bar{v}, \bar{w})$ denotes the statement obtained from $\Phi(\bar{x}, \bar{y}, \bar{z})$ by substituting $\bar{u}$ for $\bar{x}$, $\bar{v}$ for $\bar{y}$, and $\bar{w}$ for $\bar{z}$.

Furthermore, the construction of Φ from φ is effective.

The claim holds for atomic formulas, because there are no edges from $V(G) \setminus V(H)$ to $V(H) \setminus V(G)$ in $G \cup H$. It obviously extends to Boolean combinations of formulas. So suppose that $\varphi(\bar{x}, \bar{y}, \bar{z}) = \exists x_0 \psi(\bar{x}, x_0, \bar{y}, \bar{z})$. Let $\bar{v}, \bar{w}$ be tuples in G, H of the appropriate lengths. By the induction hypothesis, there are $\Psi_1(\bar{x}, \bar{y}y_0, \bar{z})$ and $\Psi_2(\bar{x}, \bar{y}, \bar{z}z_0)$ such that

$$\begin{aligned} & G \cup H \models \varphi(\bar{u}, \bar{v}, \bar{w}) \\ \iff & \exists v_0 \in V(G)\ \Psi_1(\bar{u}, \bar{v}v_0, \bar{w}) \text{ or } \exists w_0 \in V(H)\ \Psi_2(\bar{u}, \bar{v}, \bar{w}w_0). \end{aligned}$$

We may assume that Ψ_1 is of the form

$$\bigvee_{i=1}^{m} \big(G \models \chi_i(\bar{x}, \bar{y}y_0) \wedge H \models \xi_i(\bar{x}, \bar{z})\big).$$

Hence $\exists v_0 \in V(G)\ \Psi_1(\bar{u}, \bar{v}v_0, \bar{w})$ is equivalent to

$$\bigvee_{i=1}^{m} \big(\exists v_0 \in V(G)\ G \models \chi_i(\bar{u}, \bar{v}v_0) \wedge H \models \xi_i(\bar{u}, \bar{w})\big).$$

We let $\Phi_1 = \bigvee_{i=1}^{m} \big(G \models \exists y_0 \chi_i(\bar{x}, \bar{y}y_0) \wedge H \models \xi_i(\bar{x}, \bar{z})\big)$. Similarly, we define a Φ_2 from Ψ_2, and then we let $\Phi = \Phi_1 \vee \Phi_2$.

Clearly, the claim implies the statements of the lemma. Q.E.D.

2.3 Algorithms and complexity

I assume that the reader is familiar with the basics of the design and analysis of algorithms. We shall make extensive use of the Oh-notation. For example, we shall denote the class of all polynomially bounded functions of one variable n by $n^{O(1)}$. I also assume that the reader is familiar with standard complexity classes such as PTIME, NP, and PSPACE and with concepts such as reducibility between problems and hardness and completeness for complexity classes. If not specified otherwise, reductions are always polynomial time many-one reductions. The following example illustrates our notation for introducing algorithmic problems.

Example 2.4. The *dominating set problem* is defined as follows:

DOMINATING-SET	
Instance.	A graph G and a natural number k
Problem.	Decide if G has a dominating set of size k

It is well-known that DOMINATING-SET is NP-complete.

We are mainly interested in algorithms for and the complexity of *model checking problems*. For every logic L and every class $\mathcal{C}$ of graphs, we let:

MC(L, $\mathcal{C}$)	
Instance.	A sentence φ of L and a graph $G \in \mathcal{C}$
Problem.	Decide if $G \models \varphi$

If $\mathcal{C}$ is the class of all graphs, we just write MC(L).

Example 2.5. Example 2.1 shows that Dominating-Set is reducible to MC(FO). Hence MC(FO) is NP-hard. As MC(FO) is trivially reducible to MC(MSO), the latter is also NP-hard.

Fact 2.6 (Vardi, [81])**.** MC(FO) and MC(MSO) are PSPACE-complete.

This fact is often phrased as: "The *combined complexity* of FO resp. MSO is PSPACE-complete." Combined complexity refers to both the sentence and the graph being part of the input of the model checking problem. Two principal ways of dealing with the hardness of model checking problems are restrictions of the logics and restrictions of the classes of input graphs. In this survey, we shall only consider restrictions of the classes of input graphs. As for restrictions of the logics, let me just mention that the model checking problem remains NP-hard even for the fragment of first-order logic whose formulas are the *positive primitive formulas*, that is, existentially quantified conjunctions of atomic formulas. On the other hand, the model checking problem is in polynomial time for the *bounded variable fragments* of first-order logic [82].

Unfortunately, restricting the class of input graphs does not seem to improve the complexity, because the hardness result in Fact 2.6 can be strengthened: Let G be any graph with at least two vertices. Then it is PSPACE-hard to decide whether a given FO-sentence φ holds in the fixed graph G. Of course this implies the corresponding hardness result for MSO. Hence not only the combined complexity, but also the *expression complexity* of FO and MSO is PSPACE-complete. Expression complexity refers to the problem of deciding whether a given sentence holds in a fixed graph. The reason for the hardness result is that in graphs with at least two vertices we can take atoms of the form $x = y$ to represent Boolean variables and use this to reduce the PSPACE-complete satisfiability problem for *quantified Boolean formulas* to the model checking problem. Let us explicitly state the following consequence of this hardness result, where we call a class of graphs *nontrivial* if it contains at least one graph with at least two vertices.

Fact 2.7. For every nontrivial class $\mathcal{C}$ of graphs, the problems MC(FO, $\mathcal{C}$) and MC(MSO, $\mathcal{C}$) are PSPACE-hard.

So what can we possibly gain by restricting the class of input graphs of our model checking problems? As there are no polynomial time algorithms (unless PTIME = PSPACE) even for very simple classes $\mathcal{C}$ of input graphs, we have to relax our notion of "tractability". A drastic way of doing this is to consider *data complexity* instead of combined complexity, that is, consider the complexity of evaluating a fixed sentence of the logic in a given graph. The following fact implies that the data complexity of FO is in PTIME:

Fact 2.8. There is an algorithm that solves MC(FO) in time $O(k^2 \cdot n^k)$, where n denotes the order of the input graph G and k the length of the input sentence φ.

Even though FO and MSO have the same combined complexity and the same expression complexity, the following example shows that the two logics differ in their data complexity:

Example 2.9. It is easy to see that there is an MSO-formula *3-col* saying that a graph is 3-colourable. As the 3-colourability problem is NP-complete, this shows that the data complexity of MSO is NP-hard.

There are, however, nontrivial classes $\mathcal{C}$ of graphs such that the data complexity of MSO restricted to $\mathcal{C}$ is in PTIME. As we shall see later, an example of such a class is the class of all trees. Thus things are starting to get interesting.

Still, while we have seen that polynomial combined complexity is too restrictive, polynomial data complexity may be too liberal as a notion of tractability. Recall from the introduction that this survey is about *algorithmic meta theorems*, that is, uniform tractability results for classes of algorithmic problems defined in terms of logic. Fact 2.8 implies such a meta theorem: *Every graph property definable in first-order logic can be decided in polynomial time.* A serious draw back of this result is that it does not bound the degrees of the polynomial running times of algorithms deciding first-order properties. An important justification for PTIME being a reasonable mathematical model of the class of "tractable" (that is, efficiently solvable) problems is that most problems solvable in polynomial time are actually solvable by algorithms whose running time is bounded by polynomials of low degree, usually not more than three. However, this is not the case for parameterized families of polynomial time definable problems such as the family of first-order definable graph properties, for which the degree of the polynomials is unbounded. Or more plainly, even for a property that is defined by a fairly short first-order sentence, say, of length $k = 10$, an algorithm deciding this property in time $O(n^{10})$ hardly qualifies as efficient. A much more useful meta theorem would state that first-order definable graph properties can be decided "uniformly" in polynomial time, that is, in time bounded by polynomials of a fixed degree. Unfortunately, such a theorem does not seem to hold, at least not for first-order definable properties of the class of all graphs.

The appropriate framework for studying such questions is that of *parameterized complexity theory* [28, 39, 63]. A *parameterized problem* is a pair (P, κ), where P is a decision problem in the usual sense and κ is a polynomial time computable mapping that associates a natural number, called the *parameter*, with each instance of P.[1]

Here we are mainly interested in model checking problems parameterized by the length of the input formula. For a logic L and a class $\mathcal{C}$ of graphs, we let:

p-MC(L, $\mathcal{C}$)	
Instance.	A sentence φ of L and a graph $G \in \mathcal{C}$
Parameter.	$\lvert\varphi\rvert$
Problem.	Decide if $G \models \varphi$

A parameterized problem (P, κ) is *fixed-parameter tractable* if there is an algorithm deciding whether an instance x is in P in time

$$f(\kappa(x)) \cdot |x|^c, \tag{2.1}$$

for some computable function f and some constant c. We call an algorithm that achieves such a running time an *fpt algorithm*. Slightly imprecisely, we call f the *parameter dependence* of the algorithm and c the *exponent*. An fpt algorithm with exponent 1 is called a *linear fpt algorithm*. Similarly, fpt algorithms with exponents 2 and 3 are called *quadratic* and *cubic*. FPT denotes the class of all parameterized problems that are fixed-parameter tractable.

Hence a parameterized model checking problem is fixed-parameter tractable if and only if it is "uniformly" in polynomial time, in the sense discussed above. (By requiring the function f bounding the running time to be computable, we impose a slightly stronger uniformity condition than above. This is inessential, but technically convenient.)

Parameterized complexity theory is mainly concerned with the distinction between running times like $O(2^k \cdot n)$ (fpt) and $O(n^k)$ (not fpt). Running times of the latter type yield the parameterized complexity class XP. Intuitively, a problem is in XP if it can be solved by an algorithm whose running time is polynomial for fixed parameter values. Formally, XP is the class of all parameterized problems that can be decided in time

$$O(|x|^{f(\kappa(x))}),$$

[1] At some places in this paper (the first time in Remark 3.19) we are dealing with "parameterized problems" where the parameterization is not polynomial time computable. Whenever this appears here, the parameterization is computable by an fpt algorithm (see below), and this is good enough for our purposes. The same issue is also discussed in Section 11.4 of [39].

for some computable function f. Hence essentially, the parameterized model checking problem for a logic is in XP if and only if the data complexity of the logic is polynomial time. The class XP strictly contains FPT; this is an easy consequence of the time hierarchy theorem.

There is an appropriate notion of *fpt reduction* and a wide range of parameterized complexity classes between FPT and XP.

Example 2.10. A *clique* in a graph is the vertex set of a complete subgraph. The *parameterized clique problem* is defined as follows:

p-CLIQUE	
Instance.	A graph G and a natural number k
Parameter.	k
Problem.	Decide if G has a clique of size k

It is easy to see that p-CLIQUE $\in$ XP. It can be proved that p-CLIQUE is complete for the parameterized complexity class W[1] under fpt reductions [27].

Example 2.11. The *parameterized dominating set problem* is defined as follows:

p-DOMINATING-SET	
Instance.	A graph G and a natural number k
Parameter.	k
Problem.	Decide if G has a dominating set of size k

It is easy to see that p-DOMINATING-SET $\in$ XP. It can be proved that p-DOMINATING-SET is complete for the parameterized complexity class W[2] under fpt reductions [26].

The parameterized complexity classes W[1] and W[2] form the first two levels of the so-called *W-hierarchy* of classes between FPT and XP. Yet another parameterized complexity class, located between the W-hierarchy and XP, is called AW[$*$]. Thus we have

$$\mathrm{FPT} \subseteq \mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \mathrm{W}[3] \subseteq \cdots \subseteq \mathrm{AW}[*] \subseteq \mathrm{XP}.$$

It is conjectured that all containments between the classes are strict.

Fact 2.12 (Downey-Fellows-Taylor, [29])**.** p-MC(FO) is AW[$*$]-complete under fpt reductions.

Observe that by Example 2.9, p-MC(MSO) is not even in XP unless PTIME = NP.

This concludes our brief introduction to parameterized complexity theory. For proofs of all results mentioned in this section, I refer the reader to [39].

3 Monadic second-order logic on tree-like classes of graphs

The model checking problem for monadic second-order logic turns out to be tractable on trees and graph classes that are sufficiently similar to trees. A well-known measure for the similarity of a graph with a tree is *tree width*. In this article, however, we shall work with *branch width* instead. The tree width and branch width of a graph are the same up to a factor of 3/2, so the results are essentially the same. Some of the results, including Courcelle's theorem, may sound unfamiliar this way, but the reader can substitute "tree" for "branch" almost everywhere, and the results will remain true (up to constant factors, which we usually disregard anyway). Using branch width instead of tree width may make this article a bit more interesting for those who do not want to read the definition of tree width for the 100th time. However, the main reason for working with branch width is that it combines nicely with the other graph invariant that we shall study in this section, *rank width*. Indeed, both branch width and rank width of a graph are instances of the same abstract notion of branch width of a set function.

3.1 Trees

Let $\mathcal{T}$ denote the class of all trees. Recall that then $\mathcal{T}_{lb}$ denotes the class of labelled trees.

Theorem 3.1 (Folklore)**.** p-MC(MSO, $\mathcal{T}_{lb}$) is solvable by a linear fpt algorithm.

We sketch two proofs of this theorem. Even though one may view them as "essentially the same", the first is more natural from an algorithmic point of view, while the second will be easier to generalise later.

First proof sketch. Using a standard encoding of arbitrary trees in binary trees via the "first-child/next-sibling" representation, we can reduce the model checking problem for monadic second-order logic on arbitrary labelled trees to the model checking problem for monadic second-order logic on labelled binary trees. By a well-known theorem due to Thatcher and Wright [80], we can effectively associate a (deterministic) bottom-up tree automaton A_φ with every MSO-sentence φ over binary trees such that a binary tree T satisfies φ if and only if the automaton A_φ accepts T. By simulating the run of A_φ on T, it can be checked in linear time whether A_φ accepts a tree T. Q.E.D.

Second proof sketch. Again, we first reduce the model checking problem to binary trees. Let T be a labelled binary tree, and let φ be a monadic second-order sentence, say, of quantifier rank q. For every $t \in V(T)$, let T_t be the subtree of T rooted in t. Starting from the leaves, our algorithm computes $\mathrm{tp}_q^{\mathrm{MSO}}(T_t, t)$ for every $t \in T$, using Lemma 2.3. Then it decides if $\varphi \in \mathrm{tp}_q^{\mathrm{MSO}}(T, r)$ for the root r of T. Q.E.D.

The fpt algorithms described in the two proofs of Theorem 3.1 are linear in the size of the input trees. Clearly, this is optimal in terms of n (up to a constant factor). But what about the parameter dependence, that is, the function f in an fpt running time $f(k) \cdot n$? Recall that a function $f : \mathbb{N}^n \to \mathbb{N}$ is *elementary* if it can be formed from the successor function, addition, subtraction, and multiplication using composition, projections, bounded addition of the form $\sum_{\ell \le m} g(n_1, \dots, n_k, \ell)$, and bounded multiplication of the form $\prod_{\ell \le m} g(n_1, \dots, n_k, \ell)$. Let $\exp^{(h)}$ denote the h-fold exponentiation defined by $\exp^{(0)}(n) = n$ and $\exp^{(h)}(n) = 2^{\exp^{(h-1)}(n)}$ for all $n, h \in \mathbb{N}$. It is easy to see that $\exp^{(h)}$ is elementary for all $h \ge 0$ and that if a function $f : \mathbb{N} \to \mathbb{N}$ is elementary then there is an $h \ge 0$ such that $f(n) \le \exp^{(h)}(n)$ for all $n \in \mathbb{N}$. It is well known that there is no elementary function f such that the number of states of the smallest automaton A_φ equivalent to an MSO-formula φ of length k is at most $f(k)$. It follows that the parameter dependence of our automata based fpt algorithm for p-MC(MSO, $\mathcal{T}$) is non-elementary. Similarly, the number of monadic second-order q-types is nonelementary in terms of q, and hence the type based fpt algorithm also has a nonelementary parameter dependence. But this does not rule out the existence of other fpt algorithms with a better parameter dependence. The following theorem shows that, under reasonable complexity theoretic assumptions, no such algorithms exist, not even for first-order model checking:

Theorem 3.2 (Frick-Grohe, [43]).

1. Unless PTIME = NP, there is no fpt algorithm for p-MC(MSO, $\mathcal{T}$) with an elementary parameter dependence.

2. Unless FPT = W[1], there is no fpt algorithm for p-MC(FO, $\mathcal{T}$) with an elementary parameter dependence.

As almost all classes $\mathcal{C}$ of graphs we shall consider in the following contain the class $\mathcal{T}$ of trees, we have corresponding lower bounds for the model checking problems on these classes $\mathcal{C}$. The only exception are classes of graphs of bounded degree, but even for such classes, we have a triply exponential lower bound [43] (cf. Remark 4.13).

3.2 Branch decompositions

We first introduce branch decompositions in an abstract setting and then specialise them to graphs in two different ways.

3.2.1 Abstract branch decompositions

Let A be a nonempty finite set and $\kappa : 2^A \to \mathbb{R}$. In this context, the function κ is often called a *connectivity function.* A *branch decomposition* of (A, κ) is a pair (T, β) consisting of a binary tree T and a bijection $\beta : L(T) \to A$. (Recall that $L(T)$ denotes the set of leaves of a tree T.) We inductively define a mapping $\widetilde{\beta} : V(T) \to 2^A$ by letting

$$\widetilde{\beta}(t) = \begin{cases} \{\beta(t)\} & \text{if } t \text{ is a leaf,} \\ \widetilde{\beta}(t_1) \cup \widetilde{\beta}(t_2) & \text{if } t \text{ is an inner node with children } t_1, t_2. \end{cases}$$

The *width* of the branch decomposition (T, κ) is defined to be the number

$$\text{width}(T, \kappa) = \max\left\{\kappa(\widetilde{\beta}(t)) \;\middle|\; t \in V(T)\right\},$$

and the *branch width* of (A, κ), denoted by $\text{bw}(A, \kappa)$, is defined to be the minimum of the widths of all branch decompositions of (A, κ). We extend the definition of branch width to empty ground sets A by letting $\text{bw}(\varnothing, \kappa) = \kappa(\varnothing)$ for all $\kappa : \{\varnothing\} \to \mathbb{R}$. Note that $(\varnothing, \kappa)$ does not have a branch decomposition, because the empty graph, not being connected, is not a tree.

Usually, the connectivity functions κ considered for branch decompositions are integer-valued, symmetric, and submodular. A function $\kappa : 2^A \to \mathbb{R}$ is *symmetric* if $\kappa(B) = \kappa(A \setminus B)$ for all $B \subseteq A$, and it is *submodular* if

$$\kappa(B) + \kappa(C) \geq \kappa(B \cup C) + \kappa(B \cap C) \tag{3.1}$$

for all $B, C \subseteq A$.

Example 3.3. Let $A \subseteq \mathbb{R}^n$ be finite. For every $B \subseteq A$, let $r(B)$ be the dimension of the linear subspace of $\mathbb{R}^n$ generated by B, or equivalently, the rank of the matrix with column vectors B (defined to be 0 if $B = \varnothing$). Define $\kappa_{\text{lin}} : 2^A \to \mathbb{Z}$ by

$$\kappa_{\text{lin}}(B) = r(B) + r(A \setminus B) - r(A).$$

κ_{lin} measures the dimension of the intersection of the subspace generated by B and the subspace generated by $A \setminus B$. It is easy to see that κ_{lin} is symmetric and submodular.

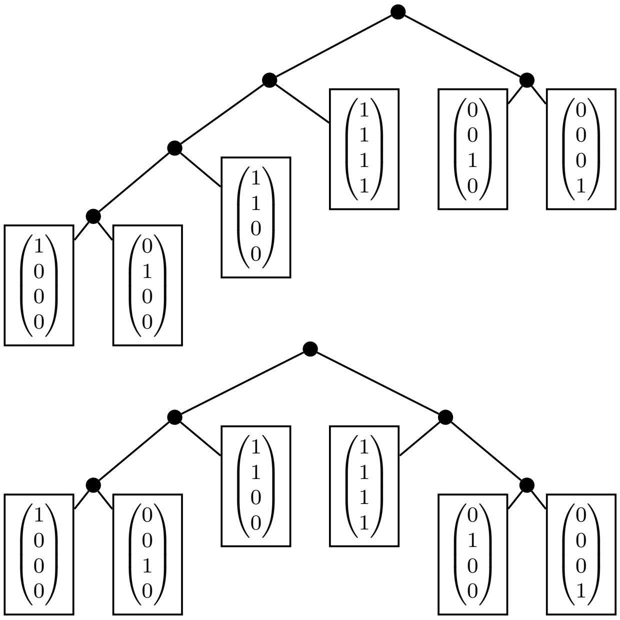

$\begin{pmatrix}1\\1\\1\\1\end{pmatrix}$
$\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$
$\begin{pmatrix}1\\1\\0\\0\end{pmatrix}$
$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$
$\begin{pmatrix}1\\1\\0\\0\end{pmatrix}$
$\begin{pmatrix}1\\1\\1\\1\end{pmatrix}$
$\begin{pmatrix}1\\0\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\1\\0\end{pmatrix}$
$\begin{pmatrix}0\\1\\0\\0\end{pmatrix}$
$\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$

FIGURE 3.1. Two branch decompositions of (A, κ_{lin}) from Example 3.3

For example, let

$$A = \left\{ \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\end{pmatrix}, \begin{pmatrix}1\\1\\0\\0\end{pmatrix}, \begin{pmatrix}1\\1\\1\\1\end{pmatrix} \right\} \subseteq \mathbb{R}^4.$$

Figure 3.1 shows two branch decompositions of (A, κ_{lin}). I leave it as an exercise for the reader to verify that the first decomposition has width 1 and the second has width 2. Observe that $\text{bw}(A, \kappa_{\text{lin}}) = 1$, because every branch decomposition (T, β) of (A, κ_{lin}) has a leaf $t \in L(T)$ with $\beta(t) = (1, 1, 1, 1)^T$, and we have $\kappa_{\text{lin}}(\{(1, 1, 1, 1)^T\}) = 1$.

Example 3.4. Again, let $A \subseteq \mathbb{R}^n$. Now, for $B \subseteq A$ let $d(B)$ be the dimension of the affine subspace of $\mathbb{R}^n$ spanned by B (defined to be -1 if $B = \emptyset$), and let

$$\kappa_{\text{aff}}(B) = d(B) + d(A \setminus B) - d(A).$$

It is not hard to prove that κ_{aff} is also symmetric and submodular.

Figure 3.2 shows an example of a set $A = \{a, b, c, d, e, f, g, h\} \subseteq \mathbb{R}^2$ and a branch decomposition of (A, κ_{aff}) of width 1.

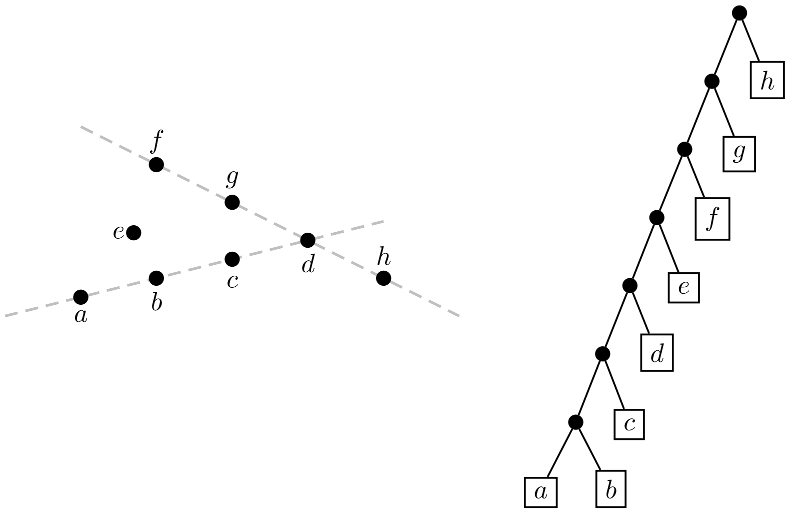


FIGURE 3.2. A set of A of eight points in the plane and a branch decomposition of (A, κ_{aff}) of width 1

Example 3.5. The previous two examples have a common generalisation, which is known as the branch width of *matroids*.[2] Let M be a matroid with base set A and rank function r_M. Then the function $\kappa : 2^A \to \mathbb{Z}$ defined by

$$\kappa_M(B) = r_M(B) + r_M(A \setminus B) - r_M(A)$$

is known as the *connectivity function* of the matroid.[3] Obviously, κ_M is symmetric, and as the rank function r_M is submodular, κ_M is also submodular.

Before we return to graphs, let us state a very general algorithmic result, which shows that approximately optimal branch decompositions can be computed by an fpt algorithm. The proof of this theorem is beyond the scope of this survey. It is based on a deep algorithm for minimizing submodular functions due to Iwata, Fleischer, and Fujishige [52].

When talking about algorithms for branch decompositions, we have to think about how the input of these algorithms is specified. Let $\mathcal{A}$ be a class of pairs (A, κ), where $\kappa : 2^A \to \mathbb{Z}$ is symmetric and submodular and takes only nonnegative values. We call $\mathcal{A}$ a *tractable class of connectivity functions*, if we have a representation of the pairs $(A, \kappa) \in \mathcal{A}$ such that,

[2] Readers who do not know anything about matroids should not worry. This example is the only place in this survey where they appear.

[3] Often, the connectivity function is defined by $\kappa_M(B) = r_M(B) + r_M(A \setminus B) - r_M(A) + 1$, but this difference is inessential here.

given the representation of (A, κ), the ground set A can be computed in polynomial time, and for every $B \subseteq A$, the value $\kappa(B)$ can be computed in polynomial time.

For example, if $\mathcal{A}$ is the class of pairs $(A, \kappa_{\mathrm{lin}})$, where A is a finite set of vectors over some finite field or the field of rationals and κ_{lin} is the linear connectivity function, then we can represent a pair $(A, \kappa_{\mathrm{lin}})$ simply by a matrix whose columns are the vectors in A. For the graph based examples that we shall describe next, the underlying graph is always used as a representation.

Theorem 3.6 (Oum-Seymour, [65]). Let $\mathcal{A}$ be a tractable class of connectivity functions. Then there is an fpt algorithm that, given $(A, \kappa) \in \mathcal{A}$ and a parameter $k \in \mathbb{N}$, computes a branch decomposition of (A, κ) of width at most $3k$ if $\mathrm{bw}(A, \kappa) \leq k$. If $\mathrm{bw}(A, \kappa) > k$, the algorithm may still compute a branch decomposition of (A, κ) of width at most $3k$, or it may simply halt without an output.[4]

3.2.2 Branch decompositions of graphs

Let $G = (V, E)$ be a graph. For a set $F \subseteq E$, we define the *boundary* of F to be the set ∂F of all vertices of G incident both with an edge in F and with an edge in $E \setminus F$. We define a function $\kappa_G : 2^E \to \mathbb{Z}$ by letting $\kappa_G(F) = |\partial F|$ for every $F \subseteq E$. It is not hard to verify that κ_G is symmetric and submodular. A *branch decomposition* of G is a branch decomposition of (E, κ_G), and the *branch width* $\mathrm{bw}(G)$ of G is the branch width of (E, κ_G).

Example 3.7. Figure 3.3 shows an example of a graph and branch decomposition of this graph of width 5.

Example 3.8 (Robertson-Seymour, [72]). For every $n \geq 3$, the complete graph K_n on n-vertices has branch width $\lceil 2n/3 \rceil$.

We omit the proof of the lower bound. For the upper bound, we partition the vertex set of K_n into three parts V_1, V_2, V_3 of size $\lceil n/3 \rceil$ or $\lfloor n/3 \rfloor$, and we partition the edge set into three sets E_{12}, E_{23}, E_{13} such that edges in E_{ij} are only incident with vertices in $V_i \cup V_j$. Then we take arbitrary branch decompositions of the three subgraphs $G_{ij} = (V_i \cup V_j, E_{ij})$ and join them together as indicated in Figure 3.4.

Note that the construction of the previous example actually shows that every n-vertex graph has branch width at most $\lceil 2n/3 \rceil$.

Example 3.9 (Robertson-Seymour, [72]). A graph has branch width 0 if and only if it has maximum degree at most 1. A graph has branch width 1 if and only if it has at least one vertex of degree greater than 1, and every

[4] An fpt algorithm of this type is known as an *fpt approximation algorithm* [7].

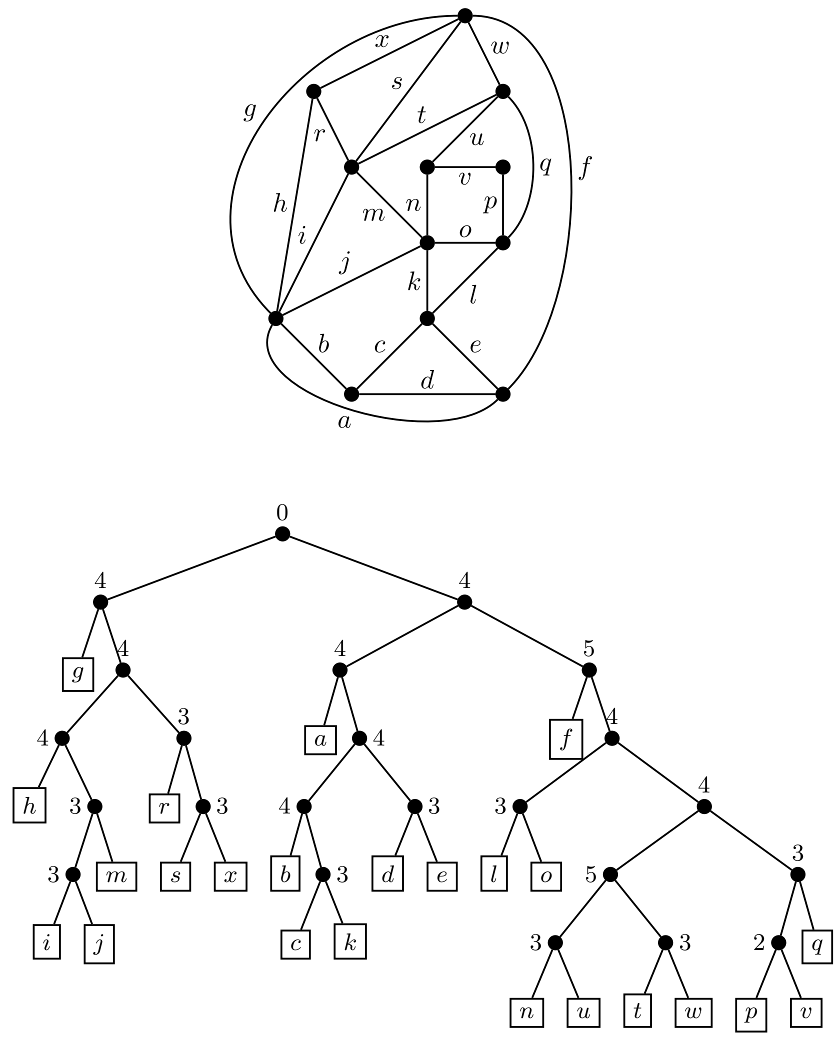


FIGURE 3.3. A graph with a branch decomposition of width 5. The numbers at the nodes indicate the size of the boundary of the edges in the subtree below that node.

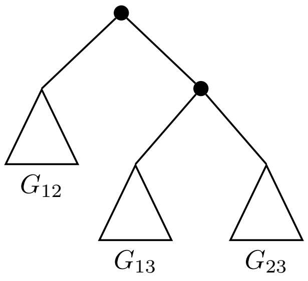


FIGURE 3.4. A branch decomposition of a clique (see Example 3.8)

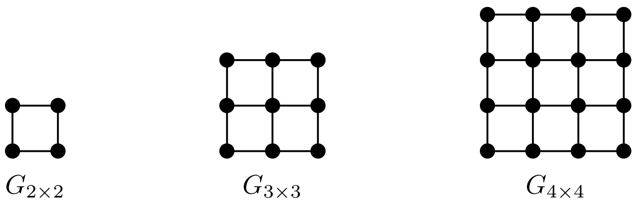


FIGURE 3.5. The $(n \times n)$-grids for $n = 2, 3, 4$

connected component has at most one vertex of degree greater than 1. Trees and cycles have branch width at most 2.

Let me suggest it as an exercise for the reader to prove these simple facts.

Example 3.10 (Robertson-Seymour, [72]). For all $n \geq 2$, the $n \times n$-*grid*

$$G_{n\times n} = \Big([n] \times [n], \big\{\{(i_1, j_1), (i_2, j_2)\} \bigm| |i_1 - i_2| + |j_1 - j_2| = 1\big\}\Big)$$

(cf. Figure 3.5) has branch width n.

Branch width is closely related to the more familiar *tree width.* In fact, it is not very hard to prove the following inequalities for all graphs G [72]:

$$\mathrm{bw}(G) \leq \mathrm{tw}(G) + 1 \leq \max\big\{(3/2) \cdot \mathrm{bw}(G), 2\big\}, \tag{3.2}$$

where $\mathrm{tw}(G)$ denotes the tree width of G.

As the connectivity functions κ_G are symmetric and submodular, approximately optimal branch decompositions can be computed by the general purpose algorithm of Theorem 3.6. However, for the special case of branch decompositions of graphs, better algorithms are known:

Theorem 3.11 (Bodlaender-Thilikos, [6]). There is an algorithm that, given a graph G and a $k \in \mathbb{N}$, decides if $\mathrm{bw}(G) \leq k$ and computes a branch

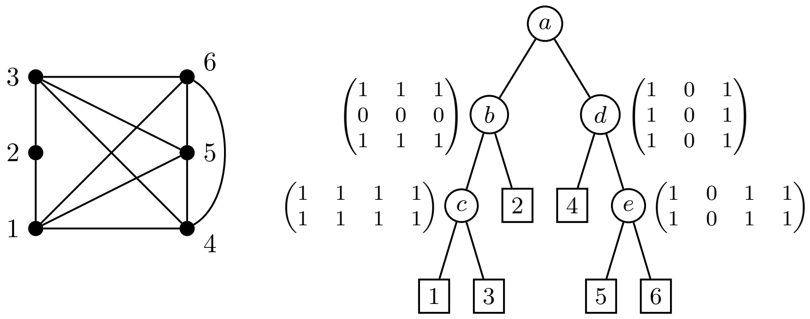


FIGURE 3.6. A graph with a rank decomposition of width 1. For later reference, we have named the nodes of the tree

decomposition of G of width at most k if this is the case in time

$$f(k) \cdot n,$$

where $n = |V(G)|$, for some computable function f.

3.2.3 Rank decompositions of graphs

Whereas branch width is based on decompositions of the edge set of a graph, for rank width we decompose its vertex set. For a graph $G = (V, E)$ and subsets $U, W \subseteq V$ of its vertex set, we let $M_G(U, W)$ be the $|U| \times |W|$-matrix with entries m_{uw} for $u \in U, w \in W$, where

$$m_{uw} = \begin{cases} 1 & \text{if } \{u, w\} \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Hence $M_G(V, V)$ is just the adjacency matrix of G. We view $M_G(U, W)$ as a matrix over the field GF(2) and let $\operatorname{rk}(M_G(U, W))$ be its row rank over GF(2). Now we define a connectivity function $\rho_G : 2^V \to \mathbb{N}$ by

$$\rho_G(U) = \operatorname{rk}\big(M_G(U, V \setminus U)\big)$$

for all $U \subseteq V$. Since the row rank and column rank of a matrix coincide, the function ρ_G is symmetric, and it is not hard to prove that it is submodular. A *rank decomposition* of G is a branch decomposition of (V, ρ_G), and the *rank width* $\operatorname{rw}(G)$ of G is the rank width of (V, ρ_G).

Example 3.12. Figure 3.6 shows an example of a graph and a rank decomposition of this graph of width 1.

It is easy to prove that rank width can be bounded in terms of branch width. The following theorem, which gives a tight bound, is not so obvious:

Theorem 3.13 (Oum [64])**.** For every graph G it holds that $\mathrm{rw}(G) \leq \max\{1, \mathrm{bw}(G)\}$.

The following example shows that the rank width of a graph can be substantially smaller than the branch width, and that it can also be the same.

Example 3.14. It is easy to see that every rank decomposition of a complete graph has width 1. Combined with Example 3.8, this shows that the branch width and rank width of a graph can differ by a factor $\Omega(n)$, where n denotes the number of vertices.

Let $I(K_n)$ be the graph obtained from the complete n-vertex graph K_n by subdividing all edges once, that is, by replacing every edge by a path of length 2. $I(K_n)$ is the *incidence graph* of K_n. Then if $n \geq 3$ and $n \equiv 0, 1 \mod 3$ we have $\mathrm{rw}(I(K_n)) = \mathrm{bw}(I(K_n)) = \lceil (2/3) \cdot n \rceil$ [64].

Example 3.15. It can be shown that the rank width of an $(n \times n)$-grid is at least $\lceil n/2 - 2 \rceil$ (follows from [64]). Hence grids have both large branch width and large rank width.

As for the branch width of graphs, there is an algorithm for computing rank width that is more efficient than the general purpose algorithm of Theorem 3.6.

Theorem 3.16 (Hlineny-Oum, [51])**.** There is an algorithm that, given a graph G and a $k \in \mathbb{N}$, decides if $\mathrm{rw}(G) \leq k$ and computes a rank decomposition of G of width at most k if this is the case in time

$$f(k) \cdot n^3,$$

where $n = |V(G)|$, for some computable function f.

Rank width is related to the graph invariant *clique width* [17], which is defined in terms of a graph algebra: The clique width $\mathrm{cw}(G)$ of a graph G is the least number of constant symbols required in a term in this algebra describing the graph G. Oum and Seymour [65] proved that for every graph G it holds that

$$\mathrm{rw}(G) \leq \mathrm{cw}(G) \leq 2^{rw(G)+1} - 1.$$

In particular, this implies that a class of graphs has bounded rank width if and only if it has bounded clique width.

3.3 Courcelle's Theorems

For every $k \geq 1$, let $\mathcal{B}_k$ be the class of all graphs of branch width at most k and $\mathcal{R}_k$ the class of all graphs of rank width at most k. The following theorem is usually formulated in terms of tree width, but by (3.2) the following "branch width version" is equivalent.

Courcelle's Theorem (Courcelle, [9])**.** For every k, the problem p-MC(MSO, $\mathcal{B}_k$) is solvable by a linear fpt algorithm.

As for Theorem 3.1, we sketch two proofs. The first is a reduction to Theorem 3.1, whereas the second is a generalisation of the second proof of Theorem 3.1.

First proof sketch. Let us fix $k \geq 1$. We reduce the model checking problem on the class $\mathcal{B}_k$ to that on labelled trees and then apply Theorem 3.1. We associate with each graph $G \in \mathcal{B}_k$ a labelled tree T^+ and with each MSO-sentence φ over graphs a sentence φ^+ over labelled trees such that $G \models \varphi \iff T^+ \models \varphi^+$. We shall do this in such a way that T^+ is computable from G in linear time and that φ^+ is computable from φ. Then our model checking algorithm proceeds as follows: Given $G \in \mathcal{B}_k$ and $\varphi \in$ MSO, it computes T^+ and φ^+ and then tests if T^+ satisfies φ^+ using the algorithm of Theorem 3.1.

The mapping $G \mapsto T^+$ will not be canonical, i.e., isomorphic graphs G will not necessarily yield isomorphic trees T^+. The tree T^+ will depend on the specific representation of the input graph G and on the algorithm we use to compute a branch decomposition of this input graph. Note that this does not affect the correctness of our algorithm.

We construct T^+ from G as follows: Without loss of generality we assume that G has no isolated vertices. We first compute a branch decomposition (T, β) of G of width at most k, which can be done in linear time by Theorem 3.11. Then we define a labelling of T that allows us to reconstruct G from the labelled tree T^+ *within MSO*. Formally, we define the labelling in such a way that G is *MSO-interpretable* in T^+. Then we can construct φ^+ from φ using the method of syntactic interpretations (see [32, 12]).

We assume that T is an ordered binary tree, that is, each inner node has a left and a right child. Recall that, for a node t of T, $\tilde{\beta}(t)$ is the set of all edges e of G such that $e = \beta(u)$ for some leaf u of T that appears in the subtree rooted at t. Let $B_t = \partial\tilde{\beta}(t)$ be the boundary of $\tilde{\beta}(t)$, that is, the set of all vertices incident with an edge in $\tilde{\beta}(t)$ and with an edge in $E(G) \setminus \tilde{\beta}(t)$. Since the width of (T, β) is at most k we have $|B_t| \leq k$ for all nodes t. The labelling of the tree T^+ encodes for every inner node t with left child t_1 and right child t_2 how B_t intersects the sets B_{t_1} and B_{t_2}. We assume some linear order of the vertices of G. Then there will be labels P_{1ij}, for $i, j \in [k]$, indicating that the ith vertex in B_{t_1} is equal to the jth vertex in B_t, and similarly labels P_{2ij} for t_2. Note that $B_t \subseteq B_{t_1} \cup B_{t_2}$, so these labels "determine" B_t. We do not label the leaves.

For each leaf t, the set B_t consists of the two endpoints of the edge $\beta(t)$ (unless one or both endpoints have degree 1). It is easy to write down four MSO-sentences $eq_{ij}(x, y)$, for $i, j \in \{0, 1\}$, such that for all leaves u, t of T

we have $T^+ \models eq_{ij}(u,v)$ if and only if the ith vertex in B_u is equal to the jth vertex in B_t. Recalling our assumption that G has no isolated vertices, it is now easy to reconstruct G from T^+ within MSO. Q.E.D.

Second proof sketch. Let G be a graph of branch width k, and let φ be an MSO-sentence, say, of quantifier rank q. We compute a branch decomposition (T,β) of G of width k. We fix some linear order on the vertices of G. For every $t \in V(T)$ we let $\bar{b}_t$ be the ordered tuple of the elements of $\partial\tilde{\beta}(t)$. Recall that for a subset $B \subseteq E(G)$, by $G[\![B]\!]$ we denote the subgraph $(\bigcup B, B)$ generated by B.

Starting from the leaves we inductively compute $\mathrm{tp}_q(G[\![\tilde{\beta}(t)]\!], \bar{b}_t)$ for all $t \in V(T)$, applying Lemma 2.3 at every node. For this to work, it is important that for all nodes t with children t_1 and t_2 it holds that

$$\begin{aligned} &V\big(G[\![\tilde{\beta}(t_1)]\!] \cap G[\![\tilde{\beta}(t_2)]\!]\big) \subseteq \partial\tilde{\beta}(t_1) \cup \partial\tilde{\beta}(t_2) \\ \text{and } &\partial\tilde{\beta}(t) \subseteq \partial\tilde{\beta}(t_1) \cup \partial\tilde{\beta}(t_2). \end{aligned}$$

Finally, we check if $\varphi \in \mathrm{tp}_q(G[\![\tilde{\beta}(r)]\!], \bar{b}_r)$ for the root r. (Note that $\bar{b}_r$ is actually the empty tuple, but this does not matter.) Q.E.D.

The following theorem was first proved by Courcelle [8, 11] in a version phrased in terms of certain graph grammars. Later, a version for clique width was proved by Courcelle, Makowsky, and Rotics [14], and finally the relation between clique width and rank width was established by Oum and Seymour [65].

Theorem 3.17 (Courcelle-Makowsky-Oum-Rotics-Seymour, [8, 65, 11, 14])**.** For every k, p-MC(MSO, $\mathcal{R}_k$) is solvable by a cubic fpt algorithm.

Proof sketch. The proof follows the same strategy as the first proof of Courcelle's Theorem: We fix k. For every graph $G \in \mathcal{R}_k$ we construct a labelled tree T^* such that G can be reconstructed from T^* within MSO. Then using the method of syntactic interpretations, for every MSO-sentence φ over graphs we obtain an MSO-sentence φ^* over labelled trees such that $G \models \varphi \iff T^* \models \varphi^*$.

T^* is obtained by suitably labelling the tree T of a rank decomposition (T,β) of G of width k. The difficulty here is to encode G in a labelling of T that uses only finitely many labels. Let t be an inner node of T with children t_1 and t_2. For $i = 1,2$, let $U_i = \tilde{\beta}(t_i)$. Furthermore, let $U = U_1 \cup U_2$ and $W = V \setminus U$. Then $\tilde{\beta}(t) = U$, and the matrices at the nodes t_1, t_2, t can be written as

$$\begin{aligned} M(U_1, V \setminus U_1) &= \begin{pmatrix} M(U_1, U_2) & M(U_1, W) \end{pmatrix}, \\ M(U_2, V \setminus U_2) &= \begin{pmatrix} M(U_2, U_1) & M(U_2, W) \end{pmatrix}, \\ M(U, V \setminus U) &= \begin{pmatrix} M(U_1, W) \\ M(U_2, W) \end{pmatrix}. \end{aligned}$$

Note that $M(U_2, U_1)$ is the transpose of $M(U_1, U_2)$. (We omit the subscript $_G$ for the matrices $M_G(\cdot, \cdot)$.)

For every node $t \in V(T)$ we compute a set B_t of at most k vertices of G such that the rows corresponding to the vertices in B_t form a basis of the row space of the matrix $M(U, V \setminus U)$, where $U = \tilde{\beta}(t)$. We define a labelling of the (inner) nodes of T as follows: Let t be an inner node with children t_1 and t_2 and $U_1 = \tilde{\beta}(t_1)$, $U_2 = \tilde{\beta}(t_2)$, $U = U_1 \cup U_2 = \tilde{\beta}(t)$. Then at t the labelling encodes

- the matrix $M(B_{t_1}, B_{t_2})$,
- for $i = 1, 2$ and each $v \in B_{t_i}$ a representation of the row of $M(U, V \setminus U)$ corresponding to v as a linear combination of vectors of the basis corresponding to B_t over the field GF(2).

Note that this amounts to at most $3k^2$ bits of information: The matrix requires at most k^2 bits, and a linear combination of k vectors over GF(2) requires k bits.

We now describe how the graph G can be reconstructed from the labelled tree T^*. The vertices of G correspond to the leaves of T^*. To find out whether there is an edge between a vertex v_1, say, with $v_1 = \beta(u_1)$ and a vertex v_2, say with $v_2 = \beta(u_2)$, we proceed as follows: Let t be the first common ancestor of u_1 and u_2, and let t_1 and t_2 be the children of t such that u_i is a descendant of t_i, for $i = 1, 2$. Let $U_i = \tilde{\beta}(t_i)$ and $U = U_1 \cup U_2 = \tilde{\beta}(t)$. Then $v_i \in U_i$. Note that $B_{u_i} = \{v_i\}$, because the matrices at the leaves only have one row. Hence, using the labelling, we can recursively find a representation of the row of the matrix $M(U_i, V \setminus U_i)$ corresponding to v_i as a linear combination of the rows corresponding to B_{t_i}. Then we can use the matrix $M(B_{t_1}, B_{t_2})$, which is also part of the labelling, to compute the entry $m_{v_1 v_2}$ of the matrix $M(U_1, U_2)$, and this entry tells us whether there is an edge between v_1 and v_2. The following example illustrates this construction. Q.E.D.

Example 3.18. Consider the graph G and branch decomposition displayed in Figure 3.6. We define the "bases" as follows:

t	1	2	3	4	5	6	a	b	c	d	e
B_t	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$	$\{6\}$	$\varnothing$	$\{1\}$	$\{1\}$	$\{4\}$	$\{5\}$

Then for example, at node b the following information is stored: The matrix

$$M(\{1\},\{2\}) = (1),$$

and a representation of the rows $r_1 = (1\ 1\ 1)$ and $r_2 = (0\ 0\ 0)$ of the matrix $M(\{1,2,3\},\{4,5,6\})$ in terms of the row r_1:

$$r_1 = 1 \cdot r_1, \quad r_2 = 0 \cdot r_1.$$

To determine whether there is an edge, say, between between $v_1 = 3$ and $v_2 = 5$ we take the least common ancestor of the two leaves, a with its two children b and d. The representation of row $r_3 = (1\ 1\ 1)$ of $M(\{1,2,3\},\{4,5,6\})$ with respect to $B_b = \{1\}$ is $r_3 = 1 \cdot r_1$, and the representation of row $r_5 = (1\ 0\ 1)$ of $M(\{4,5,6\},\{1,2,3\})$ with respect to $B_d = \{4\}$ is $r_5 = 1 \cdot r_4$. Hence $m_{35} = 1 \cdot 1 \cdot m_{14} = 1$, that is, there is an edge between 3 and 5.

It follows from Theorem 3.2 that the parameter dependence of the fpt algorithms in the previous two theorems has to be nonelementary.

We close this section with two remarks about strengthenings of the two theorems:

Remark 3.19. Our proofs yield stronger theorems than stated: Not only is the MSO model checking problem fixed-parameter tractable on every class of graphs whose branch width is bounded, but actually the following doubly parameterized model checking problem is fixed-parameter tractable:

Instance.	A sentence $\varphi \in$ MSO and a graph G
Parameter.	$\lvert\varphi\rvert + \mathrm{bw}(G)$
Problem.	Decide if $G \models \varphi$

The same is true for rank width.

Remark 3.20. It is easy to see that both theorems can be extended to labelled graphs.

Courcelle's Theorem even holds for a stronger monadic second order logic, denoted by MSO$_2$, that admits quantification not only over sets of vertices of a graph, but also over sets of edges. This stronger result can easily be derived from the (labelled) version of Courcelle's Theorem. Define the *incidence graph* $I(G)$ of a graph G to be the graph (V_I, E_I), where $V_I = V(G) \cup E(G)$ and $E_I = \big\{\{v,e\} \bigm| v \in e\big\}$. It is not hard to see that for every graph G of branch width at least 2 it holds that $\mathrm{bw}(G) = \mathrm{bw}(I(G))$. Furthermore, every MSO$_2$-formula over G can be translated to an MSO-formula over the labelled incidence graph $(I(G), P)$, where $P = E(G)$

(The labelling is not really needed, but convenient.) Hence it follows from Courcelle's Theorem that p-MC$(\mathrm{MSO}_2, \mathcal{B}_k)$ has a linear fpt algorithm for every $k \geq 1$.

This does not work for rank width, because the rank width of the incidence graph can be much larger than that of the original graph. Surprisingly, the rank width of the incidence graph of a graph is closely related to the branch width of the original graph. Oum [64] proved that

$$\mathrm{bw}(G) - 1 \leq \mathrm{rw}(I(G)) \leq \mathrm{bw}(G)$$

for every graph G with at least one vertex of degree 2.

4 First-order logic on locally tree-like classes of graphs

There is not much hope for extending the tractability of monadic second-order model checking to further natural classes of graphs such as planar graphs or graphs of bounded degree. Indeed, the MSO-definable 3-colourability problem is NP-complete even when restricted to planar graphs of degree 4. For first-order logic, however, the model checking problem is tractable on much larger classes of graphs. Seese [77] showed that first-order model checking admits a linear fpt algorithm on all classes of bounded degree. Later Frick and Grohe [42] proved the same for planar graphs, essentially by the general approach that we shall describe in this section. The crucial property of first-order logic that we exploit is its *locality.*

4.1 The locality of first-order logic

Let $G = (V, E)$ be a graph. The *distance* $\mathrm{dist}^G(v, w)$ between two vertices $v, w \in V$ is the length of the shortest path from v to w. For every $v \in V$ and $r \in \mathbb{N}$, the *r-neighbourhood* of v in G is the set

$$N_r^G(v) = \{w \in V \mid \mathrm{dist}^G(v, w) \leq r\}$$

of all vertices of distance at most r from v. For a set $W \subseteq V$, we let $N_r^G(W) = \bigcup_{w \in W} N_r^G(w)$. We omit the superscript ^{G} if G is clear from the context. The *radius* of a connected graph G is the least r for which there is a vertex $v \in V(G)$ such that $V(G) \subseteq N_r(v)$. The radius of a disconnected graph is ∞.

Observe that distance is definable in first-order logic, that is, for every $r \geq 0$ there is a first-order formula $\mathrm{dist}_{\leq r}(x, y)$ such that for all graphs G and $v, w \in V(G)$,

$$G \models \mathrm{dist}_{\leq r}(v, w) \iff \mathrm{dist}(v, w) \leq r.$$

In the following, we shall write $\mathrm{dist}(x, y) \leq r$ instead of $\mathrm{dist}_{\leq r}(x, y)$ and $\mathrm{dist}(x, y) > r$ instead of $\neg\mathrm{dist}_{\leq r}(x, y)$.

A first-order formula $\varphi(x_1,\ldots,x_k)$ is *r-local* if for every graph G and all $v_1,\ldots,v_k \in V(G)$ it holds that

$$G \models \varphi(v_1,\ldots,v_k) \iff G\big[N_r(\{v_1,\ldots,v_k\})\big] \models \varphi(v_1,\ldots,v_k).$$

This means that it only depends on the r-neighbourhood of a vertex tuple whether an r-local formula holds at this tuple. A formula is *local* if it is r-local for some r.

A *basic local sentence* is a first-order sentence of the form

$$\exists x_1 \ldots \exists x_k \left(\bigwedge_{1\le i<j\le k} \mathrm{dist}(x_i,x_j) > 2r \wedge \bigwedge_{i=1}^{k} \varphi(x_i) \right),$$

where $\varphi(x)$ is r-local. In particular, for every local formula $\varphi(x)$ the sentence $\exists x\ \varphi(x)$ is a basic local sentence.

Gaifman's Locality Theorem (Gaifman, [44])**.** Every first-order sentence is equivalent to a Boolean combination of basic local sentences.

Furthermore, there is an algorithm that computes a Boolean combination of basic local sentences equivalent to a given first-order sentence.

We shall illustrate the following proof sketch in Example 4.2 below. To appreciate the cleverness of the proof, the reader may try to find a Boolean combination of basic local sentences equivalent to the simple sentence $\varphi = \exists x \exists y \big(\neg E(x,y) \wedge P(x) \wedge Q(y)\big)$ considered in the example before reading the proof.

Proof sketch. The proof is by structural induction on first-order formulas. To enable this induction, we need to prove a stronger statement that also includes formulas with free variables. We say that a first-order formula is in *Gaifman normal form (GNF)* if it is a Boolean combination of basic local sentences and local formulas.

Claim 4.1. Every first-order formula is equivalent to a formula in GNF.

Proof. The claim is trivial for atomic formulas, because all atomic formulas are 0-local. It obviously extends to Boolean combinations of formulas. Universal quantification can be reduced to existential quantfication and negation. The only remaining case is that of existentially quantified formulas

$$\varphi(\bar{x}) = \exists y\ \psi(\bar{x},y),$$

where $\psi(\bar{x},y)$ is in GNF. We may assume that $\psi(\bar{x},y)$ is of the form

$$\bigvee_{i=1}^{m} \big(\chi_i \wedge \xi_i(\bar{x},y)\big),$$

where each χ_i is a Boolean combination of basic local sentences and each $\xi_i(\bar{x}, y)$ is local. Here we use the simple observation that a Boolean combination of local formulas is local. Then $\varphi(\bar{x})$ is equivalent to the formula

$$\bigvee_{i=1}^{m} \big(\chi_i \wedge \exists y\, \xi_i(\bar{x}, y)\big).$$

It remains to prove that each formula

$$\varphi'(\bar{x}) = \exists y\, \xi(\bar{x}, y),$$

where $\xi(\bar{x}, y)$ is local, is equivalent to a formula in GNF. Let $r \geq 0$ such that $\xi(\bar{x}, y)$ is r-local. We observe that $\varphi'(\bar{x})$ is equivalent to the formula

$$\exists y\big(\mathrm{dist}(\bar{x}, y) \leq 2r + 1 \wedge \xi(\bar{x}, y)\big) \vee \exists y\big(\mathrm{dist}(\bar{x}, y) > 2r + 1 \wedge \xi(\bar{x}, y)\big), \quad (4.1)$$

where $\mathrm{dist}(\bar{x}, y) \leq 2r + 1$ abbreviates $\bigvee_i \mathrm{dist}(x_i, y) \leq 2r + 1$. The first formula in the disjunction (4.1) is $(3r + 1)$-local. Hence we only need to consider the second, $\exists y\big(\mathrm{dist}(\bar{x}, y) > 2r + 1 \wedge \xi(\bar{x}, y)\big)$. Using Lemma 2.3 and the r-locality of $\xi(\bar{x}, y)$, it is not hard to see that this formula is equivalent to a Boolean combination of formulas of the form

$$\zeta(\bar{x}) \wedge \exists y\big(\mathrm{dist}(\bar{x}, y) > 2r + 1 \wedge \eta(y)\big),$$

where $\zeta(\bar{x})$ and $\eta(y)$ are r-local. Let $r' = 2r + 1$. It remains to prove that

$$\varphi''(\bar{x}) = \exists y\big(\mathrm{dist}(\bar{x}, y) > r' \wedge \eta(y)\big)$$

is equivalent to a formula in GNF. This is the core of the whole proof. Suppose that $\bar{x} = (x_1, \ldots, x_k)$. Let G be a graph and $\bar{v} = (v_1, \ldots, v_k) \in V(G)^k$. When does $G \models \varphi''(\bar{v})$ hold? Clearly, it holds if there are $w_1, \ldots, w_{k+1}$ of pairwise distance greater than $2r'$ such that $G \models \eta(w_j)$ for all j, because each r'-neighbourhood $N_{r'}(v_i)$ contains at most one w_j and hence there is at least one w_j of distance greater than r' from all the v_i. For $\ell \geq 1$, let

$$\theta_\ell = \exists y_1 \ldots \exists y_\ell \Big(\bigwedge_{1 \leq i < j \leq \ell} \mathrm{dist}(y_i, y_j) > 2r' \wedge \eta(y_i) \Big).$$

Note that θ_ℓ is a basic local sentence. We have just seen that θ_{k+1} implies $\varphi''(\bar{x})$. But of course $\varphi''(\bar{x})$ may also hold if θ_{k+1} does not. Let us return to our graph G and the tuple $\bar{v} \in V(G)^k$. Let $\ell \geq 1$ be maximum such that $G \models \theta_\ell$ and suppose that $\ell \leq k$. In the following case distinction, we shall determine when $G \models \varphi''(\bar{v})$.

Case 1: There are no $w_1, \ldots, w_\ell \in N_{r'}(\{\bar{v}\})$ of pairwise distance greater than $2r'$ such that $G \models \eta(w_j)$ for all j.

As $G \models \theta_\ell$, this implies that there is at least one $w \notin N_{r'}(\{\bar{v}\})$ such that $G \models \eta(w)$. Hence $G \models \varphi''(\bar{v})$.

Case 2: There is a $w \in N_{3r'}(\bar{v})$ such that $w \notin N_{r'}(\bar{v})$ and $G \models \eta(w)$. Then, trivially, $G \models \varphi''(\bar{v})$.

Case 3: Neither Case 1 nor Case 2, that is, there are $w_1, \dots, w_\ell \in N_{r'}(\{\bar{v}\})$ of pairwise distance greater than $2r'$ such that $G \models \eta(w_j)$ for all j, and there is no $w \in N_{3r'}(\bar{v}) \setminus N_{r'}(\bar{v})$ such that $G \models \eta(w)$.

Then $G \not\models \varphi''(\bar{v})$. To see this, suppose for contradiction that there is a $w \in V(G)$ such that $w \notin N_{r'}(\{\bar{v}\})$ and $G \models \eta(w)$. Then $w \notin N_{3r'}(\{\bar{v}\})$ and therefore $\mathrm{dist}(w_j, w) > 2r'$ for all $j \in [\ell]$. Thus $G \models \theta_{\ell+1}$, which contradicts the maximality of ℓ.

Hence $G \models \varphi''(\bar{v})$ if any only if we are in Case 1 or 2. Note that the conditions describing these cases can be defined by local formulas, say, $\gamma_{\ell,1}(\bar{x})$ and $\gamma_{\ell,2}(\bar{x})$. Thus if $G \models \theta_\ell \wedge \neg\theta_{\ell+1}$, then $G \models \varphi''(\bar{v})$ if and only if $G \models \gamma_{\ell,1}(\bar{v}) \vee \gamma_{\ell,2}(\bar{v})$.

Overall, $\varphi''(\bar{x})$ is equivalent to the formula

$$\theta_{k+1} \vee \bigvee_{\ell=1}^{k} \Big(\theta_\ell \wedge \neg\theta_{\ell+1} \wedge \big(\gamma_{\ell,1}(\bar{x}) \vee \gamma_{\ell,2}(\bar{x})\big)\Big),$$

which is in GNF. Q.E.D. (Claim 4.1)

It is not hard to show that our construction yields an algorithm that computes a formula in GNF equivalent to a given first-order formula. Q.E.D.

Example 4.2. Let us follow the proof of Gaifman's theorem and construct a Boolean combination of basic local sentences equivalent to the sentence

$$\varphi = \exists x \exists y \big(\neg E(x,y) \wedge P(x) \wedge Q(y)\big),$$

which is a sentence over labelled graphs with labels P and Q.

The quantifier free formula $\varphi_0(x,y) = \big(\neg E(x,y) \wedge P(x) \wedge Q(y)\big)$ is 0-local. Hence we start the construction with the formula

$$\varphi_1(x) = \exists y \big(\neg E(x,y) \wedge P(x) \wedge Q(y)\big).$$

$\varphi_1(x)$ is equivalent to the formula

$$\varphi_1' = P(x) \wedge \exists y \big(\neg E(x,y) \wedge Q(y)\big).$$

Splitting $\exists y \big(\neg E(x,y) \wedge Q(y)\big)$ with respect to the distance between x and y as in (4.1) (with $r = 0$) and simplifying the resulting formula, we obtain

$$P(x) \wedge \Big(Q(x) \vee \exists y \big(\mathrm{dist}(x,y) > 1 \wedge Q(y)\big)\Big).$$

It remains to consider the formula $\varphi_1''(x) = \exists y\big(\mathrm{dist}(x,y) > 1 \wedge Q(y)\big)$. Following the proof of Gaifman's theorem (with $\varphi'' = \varphi_1''$, $\eta(y) = Q(y)$, $r = 0$, and $k = 1$), we obtain the following equivalent formula in GNF:

$$\begin{aligned}\varphi_1'''(x) = \theta_2 \vee \Big(\theta_1 \wedge \neg\theta_2 \wedge \big(\neg\, \exists y(\mathrm{dist}(x,y) \le 1 \wedge Q(y))\\ \vee \exists y(\mathrm{dist}(x,y) \le 3 \wedge \mathrm{dist}(x,y) > 1 \wedge Q(y))\big)\Big)\end{aligned}$$

where $\theta_1 = \exists y_1 Q(y_1)$ and $\theta_2 = \exists y_1 \exists y_2\big(\mathrm{dist}(y_1,y_2) > 2 \wedge Q(y_1) \wedge Q(y_2)\big)$. Hence $\varphi_1(x)$ is equivalent to the formula $P(x) \wedge \big(Q(x) \vee \varphi_1'''(x)\big)$. The step from $\varphi_1(x)$ to $\varphi = \exists x \varphi_1(x)$ is simple, because there are no free variables left. By transforming the formula $P(x) \wedge \big(Q(x) \vee \varphi_1'''(x)\big)$ into disjunctive normal form and pushing the existential quantfier inside, we obtain the formula:

$$\begin{aligned}&\exists x\big(P(x) \wedge Q(x)\big)\\ \vee&\big(\exists x\ P(x) \wedge \theta_2\big)\\ \vee&\Big(\exists x\big(P(x) \wedge \neg\exists y(\mathrm{dist}(x,y) \le 1 \wedge Q(y))\big) \wedge \theta_1 \wedge \neg\theta_2\Big)\\ \vee&\Big(\exists x\big(P(x) \wedge \exists y(\mathrm{dist}(x,y) \le 3 \wedge \mathrm{dist}(x,y) > 1 \wedge Q(y))\big) \wedge \theta_1 \wedge \neg\theta_2\Big).\end{aligned}$$

Observe that this is indeed a Boolean combination of basic local sentences equivalent to φ. A slightly simpler Boolean combination of basic local sentences equivalent to φ is constructed in Example 3 of [50] by a different technique.

It has recently been proved in [20] that the translation of a first-order sentence into a Boolean combination of basic local sentences may involve a nonelementary blow-up in the size of the sentence.

4.2 Localisations of graph invariants

Recall that $\mathcal{G}$ denotes the class of all graphs. For every graph invariant $f : \mathcal{G} \to \mathbb{N}$ we can define its *localisation* $\ell_f : \mathcal{G} \times \mathbb{N} \to \mathbb{N}$ by

$$\ell_f(G,r) = \max\Big\{ f\big(G[N_r(v)]\big) \;\Big|\; v \in V(G)\Big\}.$$

Hence to compute $\ell_f(G,r)$, we apply f to every r-neighbourhood in G and then take the maximum. We say that a class $\mathcal{C}$ of graphs has *locally bounded* f if there is a computable[5] function $g : \mathbb{N} \to \mathbb{N}$ such that $\ell_f(G,r) \le g(r)$ for all $G \in \mathcal{C}$ and all $r \in \mathbb{N}$.

[5] It would be more precise to call this notion "effectively locally bounded f", but this would make the terminology even more awkward.

Example 4.3. One of the simplest graph invariants is the order of a graph. Observe that a class of graphs has locally bounded order if and only if it has bounded degree.

Moreover, if a class $\mathcal{C}$ has bounded degree then it has locally bounded f for every computable graph invariant f.

In this section, we are mainly interested in the localisation of branch width. Maybe surprisingly, there are several natural classes of graphs of locally bounded branch width. We start with two trivial examples and then move on to more interesting ones:

Example 4.4. Every class of graphs of bounded branch width has locally bounded branch width.

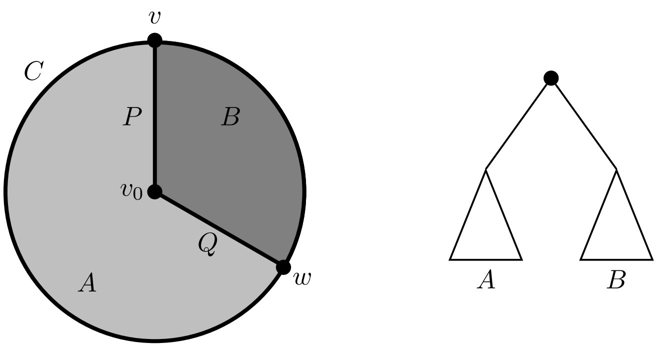
Example 4.5. Every class of graphs of bounded degree has locally bounded branch width. This follows immediately from Example 4.3.

Example 4.6 (Robertson-Seymour-Tamaki, [70, 78])**.** The class of planar graphs has locally bounded branch width. More precisely, a planar graph of radius r has branch width at most $2r + 1$.

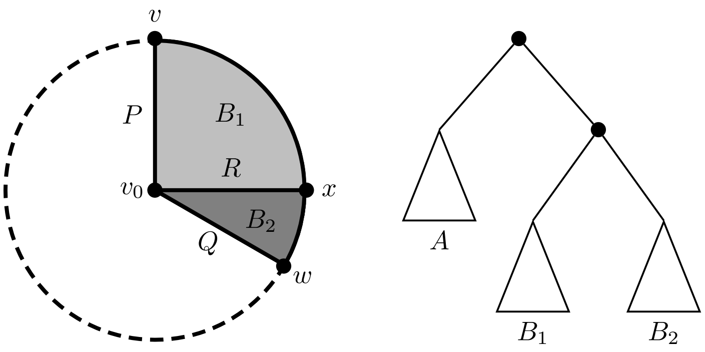
Let me sketch the proof. Let G be a planar graph of radius r, and let v_0 be a vertex such that $V(G) \subseteq N_r(v_0)$. We show how to recursively partition the edge set of G in such a way that at each stage, the boundary of each part has cardinality at most $2r + 1$. This will give us a branch decomposition of width at most $2r + 1$.

Without loss of generality we may assume that G is 2-connected; if it is not, we first decompose it into its 2-connected blocks. Figure 4.1 illustrates the following steps. We fix a planar embedding of G, and let C be the exterior cycle. We pick two vertices v, w on C and shortest paths P, Q from v_0 to v, w, respectively. Then we cut along P and Q. This gives us a partition of $E(G)$ into two parts whose boundary is contained in $V(P \cup Q)$. We can add the edges in $E(P \cup Q)$ arbitrarily to either of the two parts. Now we consider each of the parts separately. The boundary cycle consists of P, Q, and a piece of the cycle C. If this piece of C is just one edge, we can split it off and then further decompose the rest. Otherwise, we pick a vertex x on the piece of C and a shortest path R from v_0 to x. We obtain two new parts with boundaries $V(P \cup R)$ and $V(Q \cup R)$. We partition these new parts recursively until they only consist of their boundaries, and then we partition the rest arbitrarily. Of course this proof sketch omits many details and special cases. For example, the vertex v_0 could be on the exterior cycle to begin with. I leave it to the reader to work out these details.

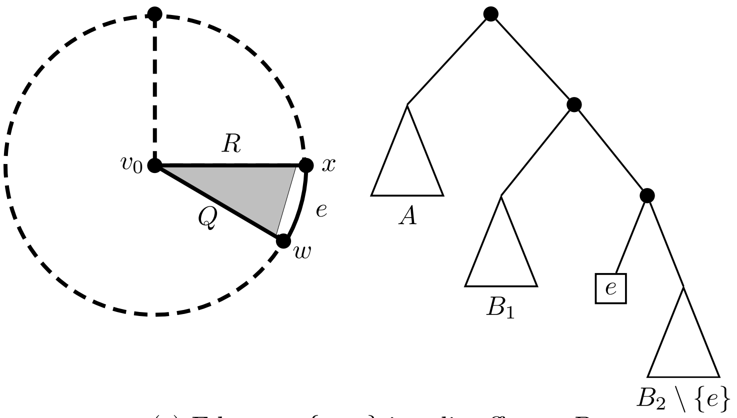
The branch decomposition in Figure 3.3 was obtained by this method. Note that the graph has radius 2, with centre v_0 being the vertex incident with the edges m and j. The initial paths P and Q have edge sets



(a) The graph is cut along PQ



(b) Part B is cut again along R



(c) Edge $e = \{w, x\}$ is split off part B_2

FIGURE 4.1. Schematic branch decomposition of a planar graph

$E(P) = \{s, m\}$ and $E(Q) = \{j\}$. The right part consists of the edges $a, b, c, k, d, e, f, l, o, n, u, t, w, p, v, q$. The edges of $P \cup Q$ were added to the left part. In the next step, the right part was split along the path R with $E(R) = \{k, e\}$. The right part of this split consists of the edges $f, l, o, n, u, t, w, p, v, q$. The edge f immediately can be split off, and the new boundary cycle is w, q, l, k, m, s. The new splitting path consists of the edge o, et cetera.

Example 4.7 (Eppstein, [34]). The *genus* of a graph is the minimum genus of an orientable or nonorientable surface the graph can be embedded into. For every k, the class of all graphs of genus at most k has locally bounded branch width. Moreover, for every k the class of all graphs of *crossing number* at most k has locally bounded branch width.

In the next example, we shall construct an artificial class of graphs of locally bounded branch width. It serves as an illustration that the global structure of graphs of locally bounded branch width can be quite complicated. In particular, this example shows that there are classes of graphs of locally bounded branch width and of unbounded average degree. Recall that if a class $\mathcal{C}$ of graphs has unbounded average degree then the size of the graphs in $\mathcal{C}$ is superlinear in their order. The graph classes in all previous examples have bounded average degree and thus size linear in the order. For planar graphs and graphs of bounded genus, this follows from Euler's formula.

Example 4.8 (Frick-Grohe, [42]). Recall that the *girth* of a graph is the length of its shortest cycle, and the *chromatic number* is the least number of colours needed to colour the graph in such a way that no two adjacent vertices receive the same colour. We shall use the well-known fact, due to Erdös [35], that for all $g, k \geq 1$ there exist graphs of girth greater than g and chromatic number greater than k. The proof of this fact (see [2]) shows that we can effectively construct such a graph $G_{g,k}$ for given g and k.

Then for every $k \geq 1$, every graph $G_{k,k}$ must have a subgraph H_k of minimum degree at least k; otherwise we could properly colour G with k colours by a straightforward greedy algorithm (see [25], Corollary 5.2.3). Let $H_k \subseteq G_{k,k}$ be such a subgraph. As a subgraph of $G_{k,k}$ the graph H_k still has girth greater than k.

Let $\mathcal{C} = \{H_k \mid k \geq 1\}$. Then $\mathcal{C}$ has unbounded minimum degree and hence unbounded average degree. Nevertheless, $\mathcal{C}$ has locally bounded branch width. To see this, simply observe that the r-neighbourhood of every vertex in a graph of girth greater than $2r + 1$ is a tree. As the branch width of a tree is at most 2, for every graph $H \in \mathcal{C}$ and every $r \geq 1$ we have

$$\ell_{\mathrm{bw}}(H, r) \leq \max\Big(\big\{\, \mathrm{bw}(H_k) \bigm| k \leq 2r + 1\big\} \cup \{2\}\Big).$$

4.3 Model checking algorithms

Theorem 4.9. Let f be a graph invariant such that the following parameterization of the model checking problem for first-order logic is fixed-parameter tractable:

p-MC(FO, f)	
Instance.	A sentence $\varphi \in \mathrm{FO}$ and a labelled graph G
Parameter.	$\lvert\varphi\rvert + f(G)$
Problem.	Decide if $G \models \varphi$

So for every class $\mathcal{C}$ of graphs of locally bounded f, the problem p-MC(FO, $\mathcal{C}$) is fixed-parameter tractable.

The proof of the theorem relies on Gaifman's Locality Theorem and the following lemma:

Lemma 4.10 (Frick-Grohe, [42])**.** Let f and $\mathcal{C}$ be as in Theorem 4.9. Then the following problem is fixed-parameter tractable:

Instance.	A labelled graph $G = (V, E, P) \in \mathcal{C}_{lb}$ and $k, r \in \mathbb{N}$
Parameter.	$k + r$
Problem.	Decide if there are vertices $v_1, \dots, v_k \in P$ such that $\mathrm{dist}(v_i, v_j) > 2r$ for $1 \leq i < j \leq k$

For simplicity, we only prove the lemma for graph invariants f that are *induced-subgraph-monotone*, that is, for all graphs G and induced subgraphs $H \subseteq G$ we have $f(H) \leq f(G)$. Note that both branch width and rank width are induced-subgraph-monotone.

Proof sketch of Lemma 4.10. Given $G = (V, E, P)$ and $k, r \in \mathbb{N}$, we first compute a maximal (with respect to inclusion) set $S \subseteq P$ of vertices of pairwise distance greater than $2r$. If $|S| \geq k$, then we are done.

Otherwise, we know that $P \subseteq N_{2r}(S)$. Let H be the induced subgraph of G with vertex set $N_{3r}(S)$. As $|S| < k$, the radius of each connected component of H is at most $(3r + 1) \cdot k$. Hence, because f is induced-subgraph-monotone,

$$f(H) \leq \ell_f(G, (3r + 1) \cdot k) \leq g((3r + 1) \cdot k),$$

where g is a function witnessing that $\mathcal{C}$ has locally bounded f.

Since $P \subseteq N_{2r}(S)$ and $V(H) = N_{3r}(S)$, for all vertices $v, w \in P$ it holds that $\mathrm{dist}^G(v, w) > 2r$ if and only if $\mathrm{dist}^H(v, w) > 2r$. Hence it remains to check whether H contains k vertices labelled P of pairwise distance greater than $2r$. This is equivalent to saying that H satisfies the first-order sentence

$$\exists x_1 \dots \exists x_k \left(\bigwedge_{1 \le i < j \le k} \mathrm{dist}(x_i, x_j) > 2r \wedge \bigwedge_{i=1}^{k} P(x_i) \right).$$

We can use an fpt algorithm for p-MC(FO, f) to check this. Q.E.D.

Proof sketch of Theorem 4.9. Let $G = (V, E) \in \mathcal{C}$ and $\varphi \in$ FO. We first transform φ into an equivalent Boolean combination of basic local sentences. Then we check separately for each basic local sentence in this Boolean combination whether it is satisfied by G and use the results to determine whether φ holds.

So let us consider a basic local sentence

$$\psi = \exists x_1 \dots \exists x_k \left(\bigwedge_{1 \le i < j \le k} \mathrm{dist}(x_i, x_j) > 2r \wedge \bigwedge_{i=1}^{k} \chi(x_i) \right),$$

where $\chi(x)$ is r-local. For each vertex v of G we check whether $G[N_r(v)]$ satisfies $\chi(v)$ using an fpt algorithm for p-MC(FO, f). We can do this within the desired time bounds because $f(G[N_r(v)]) \le \ell_f(G, r)$. If $G[N_r(v)]$ satisfies $\chi(v)$, we label v by P. To determine whether G satisfies ψ, we have to check whether the labelled graph (V, E, P) has k vertices in P of pairwise distance greater than $2r$. By Lemma 4.10, this can be done by an fpt algorithm. Q.E.D.

Corollary 4.11 (Frick-Grohe, [42])**.** For every class $\mathcal{C}$ of graphs of locally bounded branch width, p-MC(FO, $\mathcal{C}$) is fixed-parameter tractable.

Corollary 4.12. For every class $\mathcal{C}$ of graphs of locally bounded rank width, p-MC(FO, $\mathcal{C}$) is fixed-parameter tractable.

Let me close this section with a few remarks on the running time of the model checking algorithms.

Remark 4.13. We first look at the exponent of the fpt algorithms. An analysis of the algorithms described above shows that for every class $\mathcal{C}$ of locally bounded f we obtain an fpt algorithm for p-MC(FO, $\mathcal{C}$) with exponent $c+1$, where c is the exponent of an fpt algorithm for p-MC(FO, f). Hence for classes of locally bounded branch width, this yields a quadratic fpt algorithm, and for classes of locally bounded rank width, it yields an fpt algorithm with exponent four.

For classes $\mathcal{C}$ of locally bounded branch width, the exponent can be brought arbitrarily close to 1; more precisely, for every $\varepsilon > 0$ there is an fpt

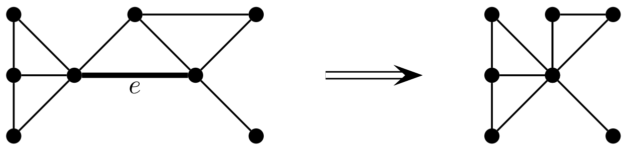


FIGURE 5.1. Contraction of edge e

algorithm for p-MC(FO, $\mathcal{C}$) with a running time of $f(k) \cdot |G|^{1+\varepsilon}$ [42]. Note that we cannot hope to find an fpt algorithm that is linear in the order for general classes of locally bounded branch width, because by Example 4.8 there are classes $\mathcal{C}$ of locally bounded branch width and unbounded average degree, which implies that the size of the graphs in $\mathcal{C}$ is not linearly bounded in the order (and thus an algorithm that is linear in the order cannot even read the whole input graph). It is an open question whether for every class $\mathcal{C}$ of graphs of locally bounded branch width there is an fpt algorithm p-MC(FO, $\mathcal{C}$) that is linear in the size $||G||$ of the input graph.

For specific classes $\mathcal{C}$, such as the class of planar graphs and classes of bounded genus or bounded degree, it is known that there are fpt algorithms that are linear in the order [42, 77].

Finally, let us look at the parameter dependence of the fpt algorithms. In general, it is again nonelementary by Theorem 3.2, because our classes contain the class of all trees. However, classes of graphs of bounded degree do not contain all trees, and it turns out that for such classes there are fpt algorithms with an elementary parameter dependence. For the class $\mathcal{D}_k$ of graphs of degree at most $k \geq 3$, there is a linear fpt algorithm for p-MC(FO, $\mathcal{D}_k$) with a triply exponential parameter dependence, and there is a matching lower bound, which even holds on labelled binary trees [43].

5 Digression: Graph minor theory

A graph H is a *minor* of a graph G if H can be obtained from G by deleting vertices, deleting edges, and contracting edges. *Contracting* an edge means removing the edge, identifying its two end vertices, and possibly removing the resulting parallel edges. Figure 5.1 illustrates this. We write $H \preceq G$ if H is isomorphic to a minor of G. A *minor mapping* from H to G is a mapping μ that associates with each $v \in V(H)$ a connected subgraph $\mu(v) \subseteq G$ and with each $e \in E(H)$ an edge $\mu(e) \in E(G)$ such that:

- for all $v \neq w$, the graphs $\mu(v)$ and $\mu(w)$ are vertex disjoint;
- for all $e = \{v, w\} \in E(H)$, the edge $\mu(e)$ is incident to a vertex $v' \in V(\mu(v))$ and a vertex $w' \in V(\mu(w))$.

It is easy to see that $H \preceq G$ if and only if there is a minor mapping from H to G. Observe that the graphs $\mu(v)$ of a minor mapping μ can be chosen

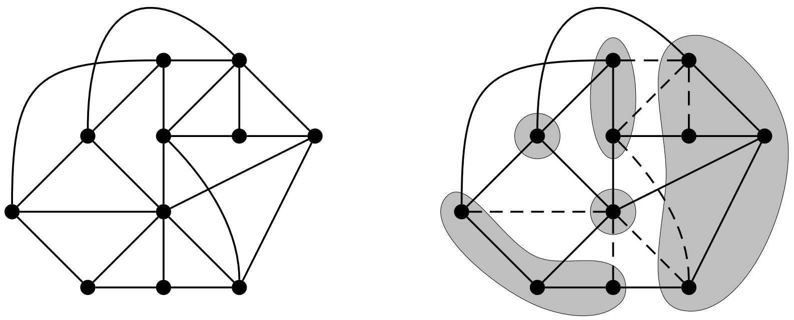


FIGURE 5.2. An image of K_5 in a nonplanar graph

to be trees. If μ is a minor mapping from H to G, we call the graph

$$\mu(H) = \Big(\bigcup_{v \in V(H)} V(\mu(v)), \bigcup_{v \in V(H)} E(\mu(v)) \cup \big\{ \mu(e) \bigm| e \in E(H) \big\} \Big)$$

an *image* of H in G.[6] Figure 5.2 shows an example.

For every graph H, we let

$$\mathcal{X}(H) = \{G \mid H \not\preceq G\}.$$

We say that a class $\mathcal{C}$ of graphs *excludes* H if $\mathcal{C} \subseteq \mathcal{X}(H)$. For a class $\mathcal{H}$ of graphs, we let

$$\mathcal{X}(\mathcal{H}) = \bigcap_{H \in \mathcal{H}} \mathcal{X}(H) = \{G \mid H \not\preceq G \text{ for all } H \in \mathcal{H}\}.$$

A class $\mathcal{C}$ of graphs is *minor-closed* if for every graph $G \in \mathcal{C}$ and every $H \preceq G$ it holds that $H \in \mathcal{C}$. Observe that a class $\mathcal{C}$ of graphs is minor-closed if and only if it can be *defined by excluding minors*, that is, there is a class $\mathcal{H}$ such that $\mathcal{C} = \mathcal{X}(\mathcal{H})$ (just take $\mathcal{H} = \mathcal{G} \setminus \mathcal{C}$). Robertson and Seymour proved that every minor-closed class of graphs can actually be defined by excluding finitely many minors:

Graph Minor Theorem (Robertson-Seymour, [75])**.** For every minor-closed class $\mathcal{C}$ of graphs there is a finite class $\mathcal{F}$ of graphs such that

$$\mathcal{C} = \mathcal{X}(\mathcal{F}).$$

Many natural classes of graphs are minor-closed:

Example 5.1. Every cycle can be contracted to a triangle K_3. Hence the class of forests (acyclic graphs) is precisely $\mathcal{X}(K_3)$.

[6] In the literature, the term "model" is used instead of "image". We prefer "image" here to avoid confusion with "models" in the logical sense.

Example 5.2. For every $k \geq 1$, the class $\mathcal{B}_k$ of all graphs of branch width k is minor-closed. Let me suggest it as an exercise for the reader to prove this. Furthermore, it holds that $\mathcal{B}_2 = \mathcal{X}(K_4)$ [72].

Example 5.3. *Series-parallel graphs* and *outerplanar graphs* exclude K_4. It can be shown that $\mathcal{X}(K_4)$ is precisely the class of all graphs that are subgraphs of series-parallel graphs (see [25], Exercise 7.32). $\mathcal{X}(\{K_4, K_{2,3}\})$ is the class of outerplanar graphs (see [25], Exercise 4.20).

Example 5.4. By Kuratowski's well-known theorem [55] (or, more precisely, by a variant due to Wagner [83]), the class of planar graphs is $\mathcal{X}(\{K_5, K_{3,3}\})$.

Example 5.5. For every $k \geq 0$, the class of all graphs of genus k is minor-closed.

Note that all previous examples of minor-closed classes also have locally bounded branch width. But this is a coincidence, as the following example shows.

Example 5.6. A graph G is an *apex graph* if there is a vertex $v \in V(G)$ such that $G \setminus \{v\}$ is planar. The class of all apex graphs is minor-closed.

The class of apex graphs does not have locally bounded branch width. To see this, consider the "pyramid graphs" P_n obtained from the $(n \times n)$-grid $G_{n\times n}$ by adding a new vertex and connecting it to all vertices of the grid. Obviously, the pyramid graphs are apex graphs, and for every $n \geq 1$ we have

$$\ell_{\mathrm{bw}}(P_n, 1) \geq \mathrm{bw}(G_{n\times n}) \geq n,$$

where the second inequality holds by Example 3.10.

Example 5.7. A graph is *knot free* if it can be embedded into $\mathbb{R}^3$ in such a way that no cycle of the graph is knotted in a nontrivial way. It is easy to see that the class of all knot free graphs is minor-closed.

Similarly, the class of all graphs that can be embedded into $\mathbb{R}^3$ in such a way that no pair of cycles is linked is minor-closed.

Let me also mention a "non-example": The class of all graphs of crossing number $k \geq 1$ is *not* minor-closed.

5.1 Structure theory

The proof of the graph minor theorem relies on a deep structure theory for classes of graphs with excluded minors. While it is far beyond the scope of this survey to describe this theory in adequate detail, or even give a precise statement of the main structural result, I would like to give the reader a glimpse of the theory, because the model checking algorithms for graphs with excluded minors heavily rely on it. Let me start with a disclaimer: The following intuitive remarks may make a nice story, but they do not always reflect the actual proofs and thus should be taken with some care.

Suppose we have a class $\mathcal{C}$ with excluded minors. Then $\mathcal{C} \subseteq \mathcal{X}(K_k)$ for some k, because every graph is a minor of some complete graph. We fix $\mathcal{C}$ and k for the rest of this section. We want to describe the structure of the graphs in $\mathcal{C}$ by "decomposing" them into "simple" building blocks. We shall define later what exactly we mean by "decomposing" a graph. For now, let us just remark that if a graph has bounded branch width, then we can decompose it into pieces of bounded size. Thus we are mainly interested in classes $\mathcal{C}$ of unbounded branch width. The following theorem, which is one of the fundamental results of the whole theory, gives us a handle on the structure of graphs of unbounded branch width:

Excluded Grid Theorem (Robertson-Seymour, [71])**.** There is a computable function f such that for every $k \geq 1$ and every graph G, if $\mathrm{bw}(G) \geq f(k)$ then $G_{k\times k} \preceq G$.

A proof of this theorem can be found in [25].

The Excluded Grid Theorem tells us that if our class $\mathcal{C}$ has unbounded branch width, then the graphs in $\mathcal{C}$ contain large grids as minors. Now we can try to use these large grids as "coordinate systems" and describe the structure of the graphs relative to the grids. So suppose we have a graph $G \in \mathcal{C}$ with a large grid minor, and let $H \subseteq G$ be the image of a large grid. Let us further assume that G is highly connected; if it is not we first decompose it into highly connected parts and then consider each of them separately. We come back to this decomposition process later. We think of the grid as embedded into the plane and the rest of G being glued onto H. It can be proved now that $G\backslash H$ must be glued onto H in a fairly "orderly" way: If there are many pairwise far apart "crossings" in the interior of G then we can find a K_k-minor in G, which is impossible because $G \in \mathcal{C} \subseteq \mathcal{X}(K_k)$. Here a crossing consists of two pairwise disjoint paths with endpoints v_1, v_3 and v_2, v_4 respectively, such that v_1, v_2, v_3, v_4 occur in this clockwise order on some cycle of the grid. Figure 5.3 shows a grid with two crossings. This leaves us with the following structure: There is a bounded number of vertices, called *apices*, that are connected to the grid in an arbitrary fashion. After removing the apices, there still may be many crossings, but they must be grouped together into a bounded number of small regions, called *vortices.* Apart from the apices and the vortices, the rest of G must fit nicely into the planar structure of the grid, that is, the components of $G \setminus H$ are planar pieces, each of which can be embedded into a "square" of the grid. However, so far we have only talked about the interior of the grid. There may be connections between different parts of the exterior cycle of

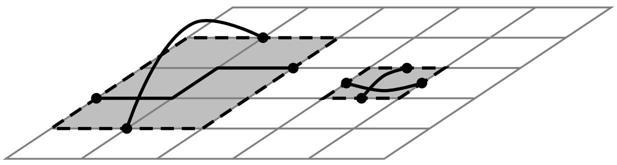


FIGURE 5.3. A grid with two crossings

the grid, but they cannot be too wild either, because otherwise we could find a large clique minor again. We can subdivide the exterior cycle into a bounded number of segments and stick some of these together. This gives us a graph that can be embedded into a surface of bounded genus (recall that every surface can be obtained by gluing together edges of a convex polygon in the plane). Thus after removing a bounded number of apices and vortices, G can be embedded into a surface of bounded genus. We say that G has *almost bounded genus.* We assumed that G is highly connected; if it is not then we can decompose it into pieces with this property. This is Robertson and Seymour's main structure theorem [74]: *For every class $\mathcal{C}$ of graphs with an excluded minor, the graphs in $\mathcal{C}$ can be decomposed into graphs that have almost bounded genus.*

Let us now make it precise what we mean by "decomposing" a graph. Intuitively, we want to recursively split the graph along small separators until there no longer are small separators and the graph is highly connected. But if we do this, we lose too much structure in the decomposition process, because two vertices that are far apart on one side of the partition may be close together on the other side and hence in the original graph. Thus "locality", and similarly "connectivity", may be destroyed in the decomposition process, and this is something we would like to avoid. We take a very drastic approach: Whenever we separate a graph, on both sides we add edges between all vertices in the separator.

We call a graph G a *clique sum* of graphs G_1 and G_2 (and write $G = G_1 \oplus G_2$) if $G_1 \cap G_2$ is a complete graph, $V(G) = V(G_1) \cup V(G_2)$, $E(G) \subseteq E(G_1) \cup E(G_2)$, and $E(G_1) \setminus E(G) \subseteq E(G_2)$, $E(G_2) \setminus E(G) \subseteq E(G_1)$. Thus G is a subgraph of $G_1 \cup G_2$ obtained by possibly deleting some of the edges in $G_1 \cap G_2$. Figure 5.4 illustrates this. Note that we are slightly abusing notation here because there may be several non-isomorphic graphs G such that $G = G_1 \oplus G_2$.

A *clique sum decomposition* of a graph G is a pair (T, γ) consisting of a binary tree T and a mapping γ that associates a graph $\gamma(t)$ with every node $t \in V(T)$ such that $\gamma(r) = G$ for the root r of T and $\gamma(t) = \gamma(t_1) \oplus \gamma(t_2)$ for all nodes t with children t_1, t_2. Figure 5.5 shows an example

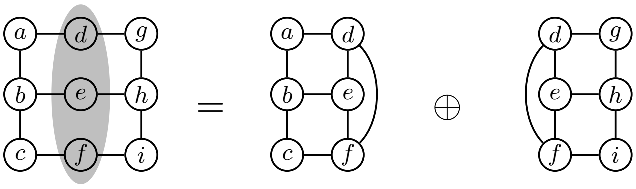


FIGURE 5.4. A clique sum

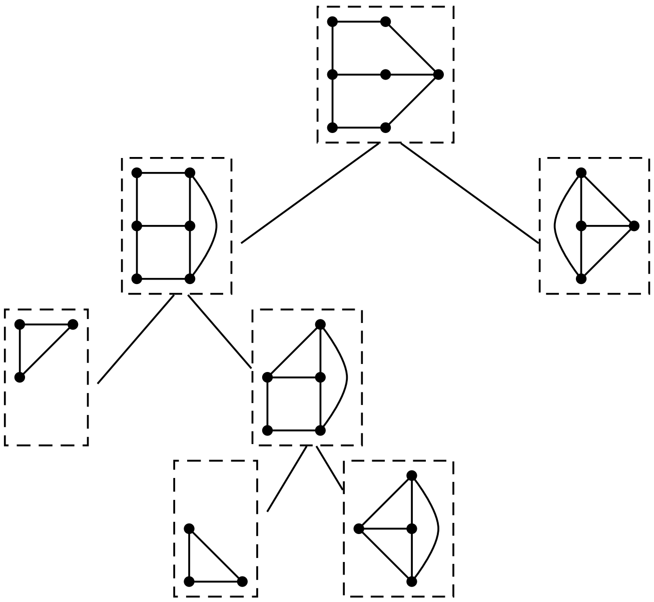


FIGURE 5.5. A clique sum decomposition

of a clique sum decomposition of a graph. The decomposition in Figure 5.5 is *complete* in the sense that the graphs at the leaves cannot be decomposed any further. In general, the clique sum decompositions we are interested in are not necessarily complete.

We call the graphs $\gamma(t)$ in a clique sum decomposition (T, γ) the *parts* of the decomposition and the parts $\gamma(t)$ for the leaves t the *atomic parts*, or just *atoms*. (T, γ) is a clique sum decomposition *over* a class $\mathcal{A}$ of graphs if all atoms of (T, γ) belong to $\mathcal{A}$. We call a graph *decomposable over* $\mathcal{A}$ if it has a clique sum decomposition over $\mathcal{A}$ and denote the class of all graphs that are decomposable over $\mathcal{A}$ by $\mathcal{D}(\mathcal{A})$.

Example 5.8. Let $k \geq 1$, and let $\mathcal{O}_k$ be the class of all graphs of order at most k. If a graph G is decomposable over $\mathcal{O}_k$, then $\mathrm{bw}(G) \leq \max\{k, 2\}$. Let me suggest it as an exercise for the reader to verify this simple fact.

Conversely, it is not too hard to prove that if a graph has branch width at most k, then it is decomposable over $\mathcal{O}_{\lceil (3/2)\cdot k\rceil}$.

Let me remark that a graph has tree width k if and only if it is decomposable over $\mathcal{O}_{k+1}$. This follows from the fact that a graph has tree width at most k if and only if it is a subgraph of a chordal graph of clique number $k+1$ (see Corollary 12.3.12 of [25]). The result for branch width then follows by (3.2).

I leave it as an exercise to prove the following simple lemma:

Lemma 5.9. If a class $\mathcal{A}$ of graphs is minor-closed, then the class $\mathcal{D}(\mathcal{A})$ is also minor-closed.

Robertson and Seymour's structure theorem for classes of graphs with excluded minors can now be stated slightly more precisely as follows: *For every class $\mathcal{C}$ of graphs with an excluded minor there is a class $\mathcal{A}$ of graphs that have almost bounded genus such that $\mathcal{C} \subseteq \mathcal{D}(\mathcal{A})$.* Of course this still leaves it open what exactly is meant by "almost bounded genus". We refer the curious reader to the last chapter of Diestel's book [25] for a more comprehensive introduction to the theory, or to Robertson and Seymour's original article [74].

We close this section by stating a simplified version of a Robertson and Seymour's structure theorem that will be sufficient for our purposes. Recall that ℓ_{bw} denotes the localization of branch width. Minor-closed classes of locally bounded branch width are particularly well behaved. Eppstein [33, 34] proved that a minor closed class $\mathcal{C}$ has locally bounded branch width if and only if it does not contain all apex graphs (recall the definition of apex graphs from Example 5.6). Demaine and Hajiaghayi [22] proved that if a class of graphs has locally bounded branch width, then there actually is a linear bound on the local branch width, that is, there is a $\lambda \geq 1$ such that for all $G \in \mathcal{C}$ and for all $r \geq 1$ it holds that $\ell_{\mathrm{bw}}(G,r) \leq \lambda \cdot r$. This motivates the definition of the following classes of graphs, for every $\lambda \geq 1$:

$$\mathcal{L}_\lambda = \big\{ G \bigm| \ell_{\mathrm{bw}}(H,r) \leq \lambda \cdot r \text{ for all } H \preceq G \big\}.$$

For every $\mu \geq 0$, we define a class of graphs that are "μ-close" to $\mathcal{L}_\lambda$:

$$\mathcal{L}_{\lambda,\mu} = \big\{ G \bigm| \exists X \subseteq V(G) :\ |X| \leq \mu \text{ and } G \setminus X \in \mathcal{L}_\lambda \big\}.$$

Theorem 5.10 (Grohe, [47])**.** For every class $\mathcal{C}$ with excluded minors, there exist nonnegative integers λ, μ such that

$$\mathcal{C} \subseteq \mathcal{D}(\mathcal{L}_{\lambda,\mu}).$$

To obtain this result from Robertson and Seymour's structure theorem, one only has to prove that graphs of almost bounded genus are in $\mathcal{L}_{\lambda,\mu}$ for suitable λ, μ. This is not very difficult.

5.2 Algorithms

Before we get back to model checking problems, let me briefly describe some other algorithmic applications of graph minor theory. Consider the following two parameterized problems:

p-DISJOINT-PATHS	
Instance.	A graph G and vertices $s_1, t_1, \dots, s_k, t_k \in V(G)$
Parameter.	k
Problem.	Decide if there are pairwise disjoint paths P_i, for $i \in [k]$, from s_i to t_i in G

p-MINOR	
Instance.	Graph G, H
Parameter.	$\lvert H \rvert$
Problem.	Decide if $H \preceq G$

For neither of the two problems, it is even obvious that they belong to the class XP, that is, can be solved in polynomial time for fixed k, $|H|$, respectively. For DISJOINT-PATHS, this was a long standing open problem posed by Garey and Johnson [45]. Robertson and Seymour proved that both problems are fixed-parameter tractable:

Theorem 5.11 (Robertson-Seymour, [73]). p-DISJOINT-PATHS and p-MINOR have cubic fpt algorithms.

The reader may wonder why we combine both problems in one theorem. The reason is that they are both special cases of the more general *rooted minor problem.* A *rooted graph* is a tuple $(G, v_1, \dots, v_k)$, where G is a graph and $v_1, \dots, v_k \in V(G)$, and a rooted graph $(H, w_1, \dots, w_k)$ is a *rooted minor* of a rooted graph $(G, v_1, \dots, v_k)$ if there is a minor map μ from H into G such that $v_i \in V(\mu(w_i))$ for all $i \in [k]$. The parameterized problem p-ROOTED-MINOR is defined as p-MINOR, but for rooted graphs. I leave it to the reader to reduce p-DISJOINT-PATHS to p-ROOTED-MINOR. Robertson and Seymour proved that p-ROOTED-MINOR has a cubic fpt algorithm.

To get an idea of the proof it is easiest to look at the disjoint paths problem. Suppose we are given a graph G and $s_1, t_1, \dots, s_k, t_k \in V(G)$. Let us further assume, to simplify the presentation, that G is $2k$-connected. If $K_{3k} \preceq G$, then we know that there are disjoint paths from the s_is to the t_is: As the graph is $2k$-connected, by Menger's theorem we can find disjoint paths from $s_1, t_1, \dots, s_k, t_k$ to an image of K_{3k}. Then in the image of K_{3k}, we can connect the pieces in the right way because all connections are there. This is not entirely trivial, because we only have an image of

K_{3k} and not a subgraph, but it can be done. So now we can assume that $K_{3k} \not\preceq G$, and we can apply the structure theory for graphs with excluded K_{3k}. If the branch width of G is bounded, we can solve the disjoint paths problem easily, for example, by applying Courcelle's theorem. If the branch width is large, then by the Excluded Grid Theorem, we can find a large grid in G. By the arguments described above, we can now find a small set of vertices such that after removing these vertices, the whole graph G fits nicely into the planar structure of the grid. Passing to a smaller grid if necessary, we may assume that all the s_i and t_i are outside the grid. Now it can be proved that if there are disjoint paths from s_i to t_i for all $i \in [k]$, then there are such paths that avoid the middle vertex of the grid (say, the grid has odd order). Intuitively, it is plausible that if we have a very large grid and k disjoint paths traversing the grid, then we can always re-route them to avoid the middle vertex. Proving this formally turns out to be the most difficult part of the whole proof [68, 69]. It builds on the full structure theory described in the previous section. However, once this is done, we know that we can delete the middle vertex of the grid and obtain a smaller graph G' such that there are disjoint paths from s_i to t_i for all $i \in [k]$ in G if and only if there are such paths in G'. We repeatedly delete "irrelevant" vertices this way until we obtain a graph of bounded branch width, and then we solve the problem on this graph. This completes our outline of the proof of Theorem 5.11.

Combined with the Graph Minor Theorem, Theorem 5.11 has the following stunning consequence.

Corollary 5.12. Every minor-closed class $\mathcal{C}$ of graphs is decidable in cubic time.

Note that a priori there is no reason why every minor-closed class $\mathcal{C}$ of graphs should be decidable at all. Remarkably, Corollary 5.12 just claims the existence of algorithms, without actually giving us the algorithms. For example, by Example 5.7 it implies the existence of a cubic time algorithm for deciding whether a graph is knot free. But we still do not know such an algorithm! The reason is that we do not know a finite family of excluded minors defining the class of knot free graphs. Corollary 5.12 is constructive in the sense that if we are given a finite family of excluded minors that defines the class $\mathcal{C}$, then we can construct a cubic time algorithm deciding $\mathcal{C}$. However, for many minor-closed classes we do not know such a finite family.

In recent years, there has been a substantial body of work on algorithms for graph problems restricted to graph classes with excluded minors or even generalisations of such classes [1, 21, 23, 24, 47, 53]. The algorithmic meta theorems presented in the following section should be seen in this context as an attempt to get a more global view on the potentials of algorithmic graph minor theory.

We close this section with a lemma that we shall need in the next section.

Lemma 5.13. For every minor-closed class $\mathcal{A}$ of graphs there is an algorithm that, given a graph $G \in \mathcal{D}(\mathcal{A})$, computes a clique sum decomposition of G over $\mathcal{A}$ in time $O(n^5)$.

Note that, in particular, the lemma implies an algorithmic version of Theorem 5.10: For every class $\mathcal{C}$ with excluded minors there is a polynomial time algorithm that, given a graph in $\mathcal{C}$, computes a clique sum decomposition of G over $\mathcal{L}_{\lambda,\mu}$.

Proof sketch of Lemma 5.13. Recall that if we write $G = G_1 \oplus G_2$, this implies that $V(G_1 \cap G_2)$ induces a clique in both G_1 and G_2, but not necessarily in G. If it also induces a clique in G, and hence $G = G_1 \cup G_2$, we call the clique sum *simplicial.* We call a clique sum decomposition (T, γ) a *simplicial decomposition* if the clique sums at all nodes of T are simplicial. We call a simplicial decomposition *complete* if its atoms can not be decomposed any further. Simplicial decompositions are much easier to handle than clique sum decompositions. Tarjan [79] showed that a separating clique of a graph can be found in quadratic time. This implies that a complete simplicial decomposition of a graph can be found in cubic time.

Observe that if a graph G has a clique sum decomposition over $\mathcal{A}$, then some supergraph $G' \supseteq G$ with the same vertex set has a simplicial decomposition over $\mathcal{A}$. As $\mathcal{A}$ is closed under taking subgraphs, we may actually assume that this simplicial decomposition is complete.

To compute a clique sum decomposition of a graph G over $\mathcal{A}$, we proceed as follows: We add a maximal set of edges to G so that the resulting graph G' is still in the class $\mathcal{D}(\mathcal{A})$. We can do this in time $O(n^5)$, testing membership in the minor-closed class $\mathcal{D}(\mathcal{A})$ in cubic time for every potential edge. Then we compute a complete simplicial decomposition of the graph G'. This also gives us a clique sum decomposition of G. Q.E.D.

6 First-order logic on graph classes with excluded minors

Let $\mathcal{C}$ be a class of graphs with excluded minors. Our goal is to design an fpt algorithm for the first-order model checking problem on $\mathcal{C}$. Recall that by Theorem 5.10, the graphs in $\mathcal{C}$ are decomposable into graphs that "almost" have locally bounded branch width, where almost means after removing a bounded number of vertices. We know how to deal with graphs of locally bounded branch width, and it is not hard to extend this to graphs of almost locally bounded branch width. Moreover, we know how to deal with tree structured graphs. By combining these things, so it seems, it should not be too hard to obtain the desired result. This is true, but there are technical difficulties to overcome.

We say that a tuple $\bar{v}$ of vertices of a graph G *induces a clique* in G if $G[\{\bar{v}\}]$ is a complete graph. We write $G = G' \oplus_{\bar{v}} H$ to denote that G is a clique sum of graphs G' and H with $V(G') \cap V(H) = \{\bar{v}\}$. For tuples $\bar{v}_1, \ldots, \bar{v}_m$ of vertices in G' and graphs $H_1, \ldots, H_m$, we may write $G' \oplus_{\bar{v}_1} H_1 \oplus_{\bar{v}_2} \ldots \oplus_{\bar{v}_m} H_m$; the order of the summation of the H_is does not matter. In the following, types are always first-order types, and we write tp instead of $\mathrm{tp}^{\mathrm{FO}}$. Let me remark that of the two lemmas below that are concerned with computing types, Lemma 6.1 also holds for MSO-types instead of FO-types, whereas the Lemma 6.2 only holds for FO-types.

To see that the parameterized problems in Lemmas 6.1 and 6.2 are well-defined, suppose that we have labelled graphs G, G', $H_1, \ldots, H_m$ and tuples $\bar{v}_0, \ldots, \bar{v}_m$ of vertices of G' such that $G = G' \oplus_{\bar{v}_1} H_1 \oplus_{\bar{v}_2} \ldots \oplus_{\bar{v}_m} H_m$. Then it follows from Lemma 2.3 that $\mathrm{tp}_q(G, \bar{v}_0)$ only depends on the types $\mathrm{tp}_q(H_1, \bar{v}_1), \ldots, \mathrm{tp}_q(H_m, \bar{v}_m)$ and not on the actual graphs H_i. That is, for all graphs $H_1', \ldots, H_m'$ with $V(G' \cap H_i') = \{\bar{v}_i\}$ and $\mathrm{tp}_q(H_i', \bar{v}_i) = \mathrm{tp}_q(H_i, \bar{v}_i)$ it holds that

$$\mathrm{tp}_q(G' \oplus_{\bar{v}_1} H_1' \oplus_{\bar{v}_2} \ldots \oplus_{\bar{v}_m} H_m') = \mathrm{tp}_q(G, \bar{v}_0).$$

Lemma 6.1. The following problem is fixed parameter tractable:

Instance.	A labelled graph G' of branch width k, tuples $\bar{v}_i \in V(G')^{k_i}$ for $i \in [0, m]$ that induce cliques in G', and q-types $\Theta_1, \ldots, \Theta_m$
Parameter.	q
Problem.	Compute the type $\mathrm{tp}_q(G, \bar{v}_0)$ for all graphs $G = G' \oplus_{\bar{v}_1} H_1 \oplus_{\bar{v}_2} \ldots \oplus_{\bar{v}_m} H_m$, where the H_i are graphs with $\mathrm{tp}_q(H_i, \bar{v}_i) = \Theta_i$ for all $i \in [m]$

Proof sketch. The proof is similar to the second proof of Courcelle's Theorem: We take a branch decomposition of G'. Starting at the leaves, we compute the types of the boundaries of all nodes. To accomodate for the graphs H_i, we label some of the leaves of the branch decomposition with the cliques $\bar{v}_i$, for $i \in [m]$, instead of edges of G'. The type that is passed from such a leaf to its parent in the computation is Θ_i. In order to obtain the type $\mathrm{tp}_q(G, \bar{v}_0)$ and not just $\mathrm{tp}_q(G, ())$ (the type of the empty tuple) at the root, at each node t of the decomposition we compute the type of a tuple consisting of the vertices in the boundary $\partial\tilde{\beta}(t)$ together with all vertices of the subgraph $G'[\![\tilde{\beta}(t)]\!]$ that appear in the tuple $\bar{v}_0$ (instead of just the vertices in $\partial\tilde{\beta}(t)$). Q.E.D.

Lemma 6.2. For all λ, μ, the following problem is fixed-parameter tractable:

Instance.	A labelled graph $G' \in \mathcal{L}_{\lambda,\mu}$, tuples $\bar{v}_i \in V(G')^{k_i}$ for $i \in [0, m]$ that induce cliques in G', and q-types $\Theta_1, \dots, \Theta_m$
Parameter.	q
Problem.	Compute the type $\mathrm{tp}_q(G, \bar{v}_0)$ for all graphs $G = G' \oplus_{\bar{v}_1} H_1 \oplus_{\bar{v}_2} \dots \oplus_{\bar{v}_m} H_m$, where the H_i are graphs with $\mathrm{tp}_q(H_i, \bar{v}_i) = \Theta_i$ for all $i \in [m]$

Proof sketch. We prove the statement by induction on μ. For $\mu = 0$, that is, graphs in $\mathcal{L}_\lambda$, it can be proved similarly to Theorem 4.9 (using Lemma 6.1 locally).

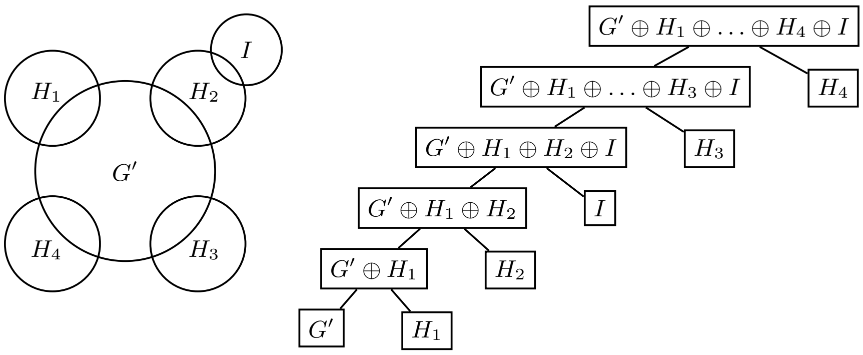
So let $\mu > 0$. Suppose we are given an instance of the problem. We observe that the graph G' contains a vertex w such that $G' \setminus \{w\} \in \mathcal{L}_{\lambda,\mu-1}$. As $\mathcal{L}_{\lambda,\mu-1}$ is minor-closed and hence decidable in cubic time by Corollary 5.12, we can find such a vertex in time $O(n^4)$. We define a new labelled graph G^* by deleting the vertex w and labelling all vertices adjacent to w in G' with a new label P. We then translate every formula ψ of quantifier rank at most q into a formula ψ^* such that $G \models \psi(\bar{v}_0) \iff G^* \models \psi^*(\bar{v}_0)$. As $G^* \in \mathcal{L}_{\lambda,\mu-1}$, we can apply the induction hypothesis to check if $G^* \models \psi^*(\bar{v}_0)$, and this way we can compute the type of $\bar{v}_0$ in G. Q.E.D.

Theorem 6.3 (Flum-Grohe, [38])**.** For every class $\mathcal{C}$ of graphs with an excluded minor, the problem p-MC(FO, $\mathcal{C}$) is fixed-parameter tractable.

Proof sketch. Let $G \in \mathcal{C}$ and $\varphi \in$ FO, say, of quantifier rank q. Let $\lambda, \mu \geq 0$ such that $\mathcal{C} \subseteq \mathcal{D}(\mathcal{L}_{\lambda,\mu})$. Using Lemma 5.13, we compute a clique sum decomposition (T, γ) of G over $\mathcal{L}_{\lambda,\mu}$.

Now the obvious idea is to compute the q-types of the "boundary tuples" for the parts $\gamma(t)$ in the decomposition in a bottom-up fashion, similarly to the second proof of Courcelle's Theorem. Unfortunately, this simple idea does not work, because a clique sum decomposition is not as well-behaved as a branch decomposition, and the boundaries of the parts may have unbounded size. It may even happen that an atom of the decomposition (corresponding to a leaf of the tree) intersects all other atoms. Figure 6.1 illustrates this.

Observe that a graph in $\mathcal{L}_{\lambda,\mu}$ cannot contain a clique with more than $k = \lceil (3/2) \cdot \lambda + \mu \rceil$ vertices. Hence for all nodes t of T with children t_1, t_2, we must have $V(\gamma(t_1) \cap \gamma(t_2)) \leq k$, because $V(\gamma(t_1) \cap \gamma(t_2))$ is a clique in the $\gamma(t_i)$, and this clique will appear in some atom of the decomposition. Let us fix some order of the vertices of G. For every inner node t with



I
H_1
H_2
G'
H_4
H_3
$G' \oplus H_1 \oplus \ldots \oplus H_4 \oplus I$
$G' \oplus H_1 \oplus \ldots \oplus H_3 \oplus I$
H_4
$G' \oplus H_1 \oplus H_2 \oplus I$
H_3
$G' \oplus H_1 \oplus H_2$
I
$G' \oplus H_1$
H_2
G'
H_1

FIGURE 6.1. The left hand side shows a graph and the right hand side a clique sum decomposition of this graph where the atom G' intersects four other atoms and the atom H_2 intersects two other atoms

children t_1, t_2, we let $\bar{c}_t$ be the ordered tuple that contains the elements of $V(\gamma(t_1) \cap \gamma(t_2))$.

Our algorithm proceeds recursively, that is, "top-down", instead of "bottom up" as the algorithm in the proof of Courcelle's Theorem, to compute the types of the tuples $\bar{c}_t$. Let us start at the root r of T. Our goal is to compute the q-type of the empty tuple in G. Suppose that the clique sum at r is $G = G_1 \oplus G_2$. We now want to compute the q-type of the tuple $\bar{c}_r$ in both G_1 and G_2; from that we easily get the q-type of the empty tuple in G using Lemma 2.3. So let us continue by computing the q-type of $\bar{c}_r$ in G_1. Suppose the children of t_1 are t_{11} and t_{12}. Let $\bar{c}_1 = \bar{c}_{t_1}$. Now we have a problem: To determine the q-type of $\bar{c}_r$ in G_1, it does not suffice to compute the q-types of $\bar{c}_1$ in G_{11} and G_{12}, because $\bar{c}_r$ and $\bar{c}_1$ may be disjoint tuples. It seems that we have to compute the q-type of the longer tuple $\bar{c}_1\bar{c}_r$ in both graphs. But clearly we cannot afford the tuples to get longer at every recursion level. Now recall that $\{\bar{c}_r\}$ is a clique in G_1. Hence it is either contained in $\{\bar{c}_1\} = V(G_{11}) \cap V(G_{12})$, in which case we have no problem anyway, or it is contained in precisely one of the two graphs G_{11}, G_{12}. Suppose $\bar{c}_r$ is contained in G_{12}. Then we first compute the q-type Θ of the tuple $\bar{c}_1$ in G_{11}. Now we have to compute the type of $\bar{c}_r$ in the graph $G_1 = G_{11} \oplus G_{12}$. That is, we are in the situation where we have to compute the type of a tuple $\bar{v}$ of vertices of a graph G' in a graph $G' \oplus_{\bar{v}'} H$ for some (and hence all) graph(s) H with $\text{tp}_q(H, \bar{v}') = \Theta$. Furthermore, we know that $\bar{v}, \bar{v}'$ induce cliques in G'. The general problem we have to solve recursively at all nodes of the decomposition tree is the following:

Compute the q-type of a tuple $\bar{v}_0$ of vertices of a graph G' in a graph $G' \oplus_{\bar{v}_1} H_1 \oplus_{\bar{v}_2} \ldots \oplus_{\bar{v}_m} H_m$ for some (and hence all) graph(s) H_i with

$\mathrm{tp}_q(H_i, \bar{c}_i) = \Theta_i$. Here all the tuples $\bar{v}_i$ have length at most k, and they induce cliques in G'.

At the leaves we can use Lemma 6.2 to do this. At the inner nodes, we proceed as described for the node t_1 above. Q.E.D.

The proof of the theorem actually shows that for all classes $\mathcal{C}$ with excluded minors, p-MC(FO, $\mathcal{C}$) has an fpt algorithm with exponent at most 5. Hence, the exponent is independent of the class $\mathcal{C}$. Thus we have "almost" proved that there is an fpt algorithm for the model checking problem parameterized both by formula size and the size of the excluded minor. With considerable additional effort, we can get rid of the "almost" in this statement. Let me explain where the difficulties are and, in very general terms, how they are resolved.

Let us first make the statement precise. We define a new graph invariant *excluded minor order (emo)* by letting

$$\mathrm{emo}(G) = \min\{|H| \mid H \not\preceq G\}$$

for every graph G. Note that $\mathrm{emo}(G) = \min\{n \mid K_n \not\preceq G\}$ and that a class $\mathcal{C}$ excludes a minor if and only if it has bounded excluded minor order. Our goal is to prove that the following problem is fixed-parameter tractable:

p-MC(FO, emo)	
Instance.	A graph G and a sentence $\varphi \in$ FO
Parameter.	$\lvert\varphi\rvert + \mathrm{emo}(G)$
Problem.	Decide if $G \models \varphi$

We have already proved that for every k there is an fpt algorithm A_k with exponent 5 for the first-order model checking problem on the class of all graphs of excluded minor order at most k. The problem is that the family A_k of algorithms is *nonuniform*, that is, we have a different algorithm for every k. To prove that p-MC(FO, emo) is fixed-parameter tractable, we need a uniform family A_k, or equivalently, a single algorithm A that takes k as an additional input. The family of algorithms we construct in the proof is nonuniform because we use Corollary 5.12 to get decision algorithms for the minor-closed classes $\mathcal{L}_{\lambda,\mu}$ (in the proof of Lemma 6.2) and $\mathcal{D}(\mathcal{L}_{\lambda,\mu})$ (in the proof of Lemma 5.13) for parameters λ, μ that depend on the excluded minor order of the input graph. If we could compute finite families of excluded minors characterising the classes $\mathcal{L}_{\lambda,\mu}$ and $\mathcal{D}(\mathcal{L}_{\lambda,\mu})$ from the parameters λ, μ, then we would be fine, but we currently do not know how to do this. Fortunately, there is an alternative approach that avoids Corollary 5.12 entirely. The application of Corollary 5.12 in the proof of Lemma 5.13

yielded an algorithm for computing a clique sum decomposition of a graph over $\mathcal{D}(\mathcal{L}_{\lambda,\mu})$. While we do not know how to compute such a decomposition uniformly in λ and μ, in [18] we found a way to compute, uniformly in λ, μ, a decomposition that is a sufficiently good approximation of the desired clique sum decomposition. The algorithm recursively splits the input graph along small separators that are sufficiently "balanced". The application of Corollary 5.12 in the proof of Lemma 6.2 was needed to find a set of at most μ vertices in a graph in $\mathcal{L}_{\lambda,\mu}$ whose removal left a graph in $\mathcal{L}_{\lambda}$. In [18], we found an fpt algorithm that, given a graph $G \in \mathcal{L}_{\lambda,\mu}$, computes a set $W \subseteq V(G)$ of at most μ vertices such that $G \setminus W \in \mathcal{L}_{\lambda'}$ for some λ' that is effectively bounded in terms of λ. This is good enough for our purposes. Putting everything together, we obtain the following result:

Theorem 6.4 (Dawar-Grohe-Kreutzer, [18])**.** p-MC(FO, emo) is fixed-parameter tractable.

We say that a class *locally excludes a minor* if it has locally bounded excluded minor order. Then combining Theorems 6.4 and 4.9, we get:

Corollary 6.5 (Dawar-Grohe-Kreutzer, [18])**.** For every class $\mathcal{C}$ locally excluding a minor, the problem p-MC(FO, $\mathcal{C}$) is fixed-parameter tractable.

7 Other logics and other problems

In this section, we briefly discuss some extensions of the main results mentioned in this survey to more powerful logics, and also to variants of the basic model checking problem.

7.1 Other logics

It is really not much that is known about algorithmic meta theorems for logics other than first-order and monadic second-order logic. Courcelle's Theorem and its variant for graphs of bounded rank width can be extended to the extension of monadic second order logic by modulo counting quantifiers [10, 12] (also see [58]), and clearly not to full binary second order logic.

As for the results for first-order logic, let us consider potential extensions of the model-checking results to *monadic transitive closure logic* and *monadic least fixed-point logic*. Both transitive closure logic and least fixed-point logic have been extensively studied in finite model theory [31, 56]. Their monadic fragments are strictly contained in monadic second-order logic, and they strictly contain first-order logic. (When we say that a logic *contains* another logic, we mean semantic containment, that is, L_1 *contains* L_2 if every formula of L_2 is logically equivalent to a formula of L_1. We say that L_1 *strictly contains* L_2 it L_1 contains L_2, but L_2 does not contain L_1.) Monadic transitive closure logic and monadic least fixed-point logic seem to mark the boundary of the range of logics to which the tractability results for first-order model checking can be extended.

Monadic transitive closure logic TC^1 is the extension of first-order logic by formulas of the form $[\mathrm{TC}_{x,y}\varphi](x,y)$, where φ is a formula with free variables among $\{x,y\}$. The free variables of the formula $[\mathrm{TC}_{x,y}\varphi](x,y)$ are x and y. It is allowed to nest TC-operators arbitrarily and interleave them with first-order quantifiers and connectives. However, we do not allow any other free variables than x and y in the formula φ in $[\mathrm{TC}_{x,y}\varphi](x,y)$. The semantics is defined as follows: If G is a (labelled) graph and $v,w \in V(G)$, then $G \models [\mathrm{TC}_{x,y}\varphi](v,w)$ if and only if there is an $m \geq 1$ and vertices $v_1,\dots,v_m \in V(G)$ such that $v = v_1, w = v_m$, and $G \models \varphi(v_i,v_{i+1})$ for all $i \in [m-1]$.

Example 7.1. The following TC^1-sentence states that a graph is connected:

$$\forall x \forall y [\mathrm{TC}_{x,y} E(x,y)](x,y).$$

It is known that there is no sentence of first-order logic defining connectivity (see, e.g., [31, 32, 56]).

Example 7.2. The following TC^1-sentence states that a graph has no cyclic walk of odd length and hence is bipartite

$$\neg\exists x \exists y \Big(\big[\mathrm{TC}_{x,y} \exists z \big(E(x,z) \wedge E(z,y)\big)\big](x,y) \wedge E(y,x)\Big).$$

Again, it is known that there is no sentence of first-order logic defining bipartiteness.

The logic TC^1 trivially contains FO, and it is strictly contained in MSO. As opposed to MSO, its data complexity is still in polynomial time (actually, in nondeterministic logarithmic space).

Theorem 7.3. Let $\mathcal{C}$ be a class of graphs that contains all planar graphs of degree at most 3. Then p-MC(TC^1,$\mathcal{C}$) is hard for the parameterized complexity class AW[$*$].

Proof sketch. We reduce the model checking problem for first-order logic on arbitrary graphs, which is known to be AW[$*$]-complete (by Theorem 2.12), to p-MC(TC^1,$\mathcal{C}$). Let G be a graph and φ a first-order sentence.

We start with constructing a drawing of G in the plane, which of course may involve edge crossings. We can find a drawing with at most polynomially many (in the number of vertices of G) crossings such that in each point of the plane at most 2 edges cross. We introduce five new labels P_1, P_2, Q_1, Q_2, R. We define a new labelled graph G_1 by labelling each vertex of the original graph G with P_1 and replacing each edge crossing in the

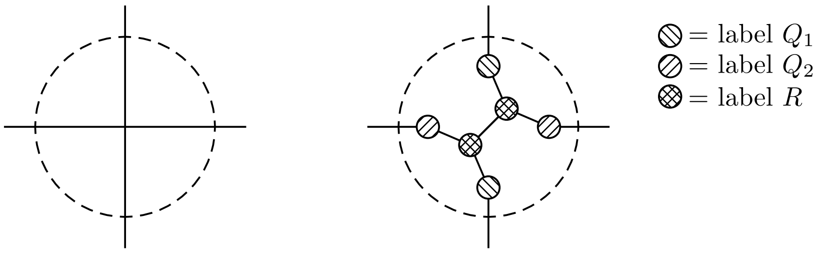


FIGURE 7.1. A gadget for edge crossings

drawing of G by a little gadget, as shown in Figure 7.1. Observe that the edge relation of the graph G can be defined in G_1 by a TC^1-formula (but not by an FO-formula, because an edge may cross many other edges). G_1 is planar, but may have degree greater than 3. We define a graph G_2 by replacing every vertex v of G_1 of degree d by a binary tree with exactly d leaves. With each leaf we associate one vertex w adjacent to v in G_1. We connect the leaf of the v-tree associated with w with the leaf of the w-tree associated with v. Then we identify v with the root of its tree, label it P_1, and label all other vertices of the tree P_2. Then the edge relation of G is also definable in G_2 by a TC^1-formula. We can use this formula to translate the formula φ into a TC^1-formula φ_2 such that

$$G \models \varphi \iff G_2 \models \varphi_2.$$

G_2 is a planar graph of degree at most 3, and it clearly can be computed from G in polynomial time. This gives us the desired reduction. Q.E.D.

Monadic least-fixed-point logic LFP^1 (see, e.g., [48, 76]) is the extension of first-order logic by formulas of the form $[\mathrm{LFP}_{x,X}\varphi](x)$, where φ is a first-order formula such that X only occurs positively in φ and φ has no free individual variables other than x. (It may have free set variables other than X.) The free variables of $[\mathrm{LFP}_{x,X}\varphi](x)$ are x and all free set variables of φ except X. To define the semantics, let $\varphi = \varphi(x, X, Y_1, \dots, Y_m)$. Let G be a labelled graph and $W_1, \dots, W_m \subseteq V(G)$, $v \in V(G)$. Then $G \models [\mathrm{LFP}_{x,X}\varphi(x, X, W_1, \dots, W_m)](v)$ if and only if v is in the least fixed point of the monotone operator $U \mapsto \{u \mid G \models \varphi(u, U, W_1, \dots, W_m)\}$ on $V(G)$. We call a formula in LFP^1 *restricted* if for every subformula of the form $[\mathrm{LFP}_{x,X}\varphi](x)$, the formula φ has no free set variables other than X. By LFP^1_r we denote the fragment of LFP^1 consisting of all restricted formulas.

The reason for requiring that a formula φ in the scope of a fixed-point operator $[\mathrm{LFP}_{x,X}\varphi](x)$ contains no free individual variables other than x is that otherwise even the restricted fragment of the logic would contain TC^1.

It can be shown that LFP^1 (as defined here) does not contain TC^1 and that, conversely, TC^1 does not contain LFP^1, not even LFP^1_r.

I was unable to come up with convincing examples of properties of plain graphs that are definable in LFP^1_r or LFP^1, but not in first-order logic. However, this changes when we admit more general structures. For example, on *Kripke structures*, that is, labelled directed graphs with one distinguished element, LFP^1 contains the modal μ-calculus. Here is another example:

Example 7.4. We can describe monotone Boolean circuits as labelled directed acyclic graphs, and assignments to the input gates by an additional label. It is easy to see that there is an LFP^1_r-formula stating that an assignment satisfies a circuit. This is not definable in first-order logic.

As we mentioned earlier, almost all results presented in this survey extend to arbitrary structures. In this context, the following tractability result is more interesting than it may seem in a purely graph theoretical context.

Theorem 7.5. Let $\mathcal{C}$ be a class of graphs such that p-$\mathrm{MC}(\mathrm{FO}, \mathcal{C}_{lb})$ is fixed-parameter tractable. Then p-$\mathrm{MC}(\mathrm{LFP}^1_r, \mathcal{C}_{lb})$ is fixed-parameter tractable.

Proof sketch. To evaluate a formula of the form $[\mathrm{LFP}_{x,X}\varphi](x)$, where $\varphi = \varphi(x, X)$ is first-order, in a graph G, we proceed as follows: We introduce a new label P. Initially, we set $P(G) = \emptyset$. Then we repeatedly compute the set of all $v \in V(G)$ such that $G \models \varphi(v, P(G))$ using an fpt algorithm for p-$\mathrm{MC}(\mathrm{FO}, \mathcal{C}_{lb})$ and set $P(G)$ to be the set of all these vertices. After at most $n = |G|$ steps, the computation reaches a fixed point, which consists precisely of all v such that $G \models [\mathrm{LFP}_{x,X}\varphi](v)$. Using this algorithm as a subroutine, we can easily model-check arbitrary sentences in LFP^1_r. Q.E.D.

Lindell [57] proved that for the classes $\mathcal{D}_k$ of graphs of degree at most k, the problem p-$\mathrm{MC}(\mathrm{LFP}^1_r, \mathcal{D}_k)$ even has a linear time fpt algorithm.

7.2 Generalised model checking problems

For a formula $\varphi(x_1, \ldots, x_k)$ and a graph G, by $\varphi(G)$ we denote the set of all tuples $(v_1, \ldots, v_k) \in V(G)^k$ such that $G \models \varphi(v_1, \ldots, v_k)$. For every logic L and class $\mathcal{C}$ of graphs, we may consider the following variants of the model checking problem p-$\mathrm{MC}(\mathrm{L}, \mathcal{C})$: The input always consists of a graph $G \in \mathcal{C}$ and a formula $\varphi \in \mathrm{L}$, possibly with free variables. The parameter is $|\varphi|$. The *decision problem* simply asks if $\varphi(G)$ is nonempty. For logics closed under existential quantification, this problem is equivalent to the model checking problem p-$\mathrm{MC}(\mathrm{L}, \mathcal{C})$. Therefore, we shall not consider it here anymore. The *construction problem* asks for a solution $\bar{v} \in \varphi(G)$ if there exists one. The *evaluation (or listing) problem* asks for all solutions, that is, for the whole set $\varphi(G)$. Finally, the *counting (or enumeration) problem* asks for the number $|\varphi(G)|$ of solutions. All these problems have natural applications.

The results on monadic second-order model checking on graphs of bounded branch width and bounded rank width (Theorems 3.3 and 3.17) can be extended to the corresponding construction and counting problems [3, 15, 37, 40]. For the evaluation problem, the situation is a bit more complicated because the size of the answer $\varphi(G)$ may be much larger than the size of the input (n^k for a graph of order n and a formula with k free variables), hence we cannot expect an algorithm that is fixed-parameter tractable. However, it has been proved that there is a linear time fpt algorithm for this problem if the running time is measured in terms of the input size plus the output size [16, 37]. Recently, it has been shown that there even is such an algorithm that does a linear (in terms of the input size) pre-computation and then produces solutions with delay bounded in terms of the parameter [4, 13].

Frick [40, 41] proved that the construction problem and counting problem for many classes of graphs of locally bounded branch width, including planar graphs and graphs of bounded degree, has a linear fpt algorithm. This is a nontrivial extension of the model checking results. Even for a simple first-order definable counting problem like the parameterized independent set counting problem ("Count the number of independent sets of size k in a graph."), say, on a class of graphs of bounded degree, it is not obvious how to solve it by an fpt algorithm. For the evaluation problem, again there are linear time fpt algorithms if the running time is measured in terms of the input size plus the output size [40]. For classes of graphs of bounded degree, Durand and Grandjean [30] proved that there is an fpt algorithm for the first-order evaluation problem that does a linear pre-computation and then produces solutions with delay bounded in terms of the parameter.

Finally, let us take a brief look at optimisation problems, which play a central role in complexity theory, but have not been studied very systematically in the context of meta theorems. Consider a first-order formula $\varphi(X)$ that is positive in a free set variable X. Such a formula naturally describes a minimisation problem: Given a graph G, find a set $S \subseteq V(G)$ of minimum size such that $G \models \varphi(S)$. Many natural minimisation problems on graphs can be described this way. An example is the minimum dominating set problem, which can be described by the formula dom(X) of Example 2.1. Similarly, formulas $\varphi(X)$ that are negative in X naturally describe maximisation problems. An example is the maximum independent set problem, which is described by the formula $ind(X) = \forall x \forall y(\neg X(x) \vee \neg X(y) \vee \neg E(x, y))$. We call such optimisation problems *first-order definable*. It was proved in [19] that the restriction of a first-order definable optimisation problem to a class of graphs with an excluded minor has a polynomial time approximation scheme, that is, can be approximated in polynomial time to any factor $(1 + \varepsilon)$, where $\varepsilon > 0$.

8 Concluding remarks and open questions

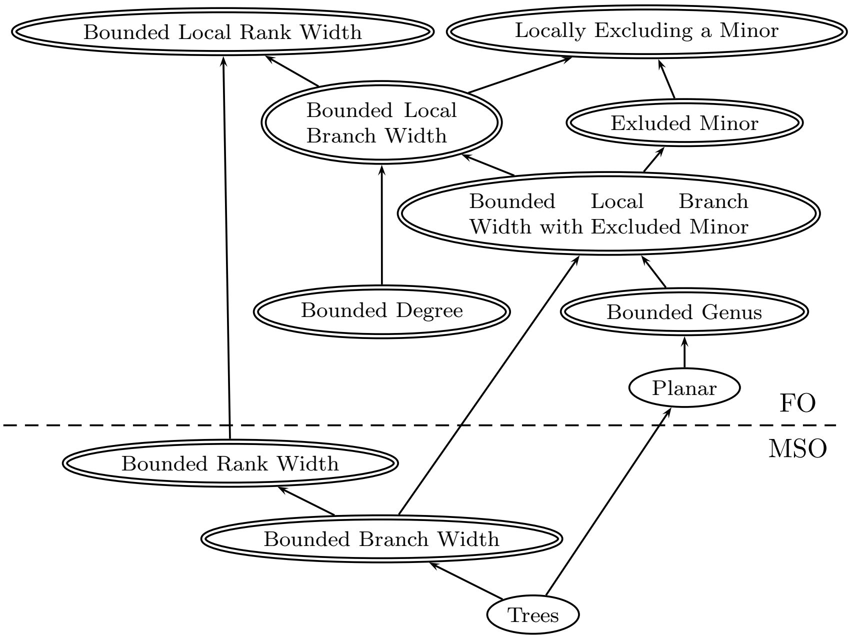


FIGURE 8.1. Classes of graphs with a tractable first-order model checking problems. Double-lined ellipses contain families of classes. Classes below the dashed line have a tractable monadic second-order model checking problem

Figure 8.1 gives an overview of the classes of graphs we have studied in this survey. Let me conclude by mentioning a few directions for further research that I find particularly promising:

8.1 Further tractable classes

Many of the classes of graphs considered in this survey, including all classes excluding a minor, have bounded average degree. It may be tempting to conjecture that first-order model checking is tractable on all classes of graphs of bounded average degree, but it is easy to see that this is not the case. As Stephan Kreutzer observed, it is not even the case for classes of bounded maximum average degree, where the *maximum average degree* of a graph G is the maximum of the average degrees of all subgraphs of G. To see this, just observe that model-checking on a graph G can be reduced to model-checking on its incidence graph (i.e., the graph obtained from G by subdividing each edge once), and that every incidence graph has maximum average degree at most 4.

Nešetřil and Ossona de Mendez [61] introduced a property of graph classes that may be viewed as a refinement of maximum average degree and that avoids such problems. Let G be a graph. The *radius* of a minor mapping μ from a graph H to G is the minimum of the radii of the subgraphs $G[\mu(v)]$, for $v \in V(H)$. We write $H \preceq_r G$ if there is a minor mapping of radius at most r from H to G. Note that $H \preceq_0 G$ if and only if H is a subgraph of G. The *greatest reduced average density (grad) of rank r of G* is the number

$$\nabla_r(G) = \max\left\{ \frac{|E(H)|}{|V(H)|} \,\middle|\, H \preceq_r G \right\}.$$

Note that $\nabla_0(G)$ is half the maximum average degree of G. Now a class $\mathcal{C}$ of graphs has *bounded expansion* if there is some function f such that $\nabla_r(G) \leq f(r)$ for all $G \in \mathcal{C}$ and $r \geq 0$. Nešetřil and Ossona de Mendez observed that every class of graphs excluding a minor has bounded expansion. It can be shown that there are classes of bounded expansion that do not exclude a minor, not even locally. Conversely, there are classes of bounded local tree width and hence classes locally excluding a minor that do not have bounded expansion. This follows from Example 4.8 and the fact that classes of bounded expansion have bounded average degree. I refer the reader to [60, 61, 62] for an introduction to classes of bounded expansion and an overview of their nice algorithmic properties.

Open Problem 8.1. Is p-MC(FO, $\mathcal{C}$) fixed-parameter tractable for every class $\mathcal{C}$ of graphs of bounded expansion?

There is no need to restrict the study of structural properties that facilitate efficient model checking to graph theoretic properties such as those predominant in this survey. For example, it would also be very interesting to study the complexity of model-checking problems on finite algebraic structures such as groups, rings, fields, lattices, et cetera.

Open Problem 8.2. Are p-MC(FO, $\mathcal{C}$) and p-MC(MSO, $\mathcal{C}$) fixed-parameter tractable for the classes $\mathcal{C}$ of finite groups, finite abelian groups, finite rings, finite fields?

8.2 Necessary conditions for tractability

The main results presented in this survey may be viewed as giving sufficient conditions for classes of graphs to have tractable first-order or monadic second-order model checking problems. *What are necessary conditions for tractability, and which classes have hard model checking problems?* Note that it is not easy to come up with structural conditions for hardness, because we can "cheat" and, for example, pad graphs that have a structure presumably making model checking difficult with a large number of isolated vertices. This makes the model checking problem "easier" just because it gives us more time to solve it. Thus we probably want to impose closure conditions on the classes of graphs we consider, such as being closed under taking subgraphs.

It follows from the Excluded Grid Theorem that for minor-closed classes $\mathcal{C}$ of graphs, p-MC(MSO, $\mathcal{C}$) is fixed-parameter tractable if and only if $\mathcal{C}$ has bounded branch width. Actually, this can be slightly strengthened to classes closed under taking topological minors. I do not know of any results beyond that. To stimulate research in this direction, let me state a conjecture:

Conjecture 8.3. Let $\mathcal{C}$ be a class of graphs that is closed under taking subgraphs. Suppose that the branch width of $\mathcal{C}$ is not poly-logarithmically bounded, that is, there is no constant c such that $\mathrm{bw}(G) \leq \log^c |G|$ for every $G \in \mathcal{C}$. Then p-MC(MSO, $\mathcal{C}$) is not fixed parameter tractable.

Of course, with current techniques we can only hope to prove this conjecture under some complexity theoretic assumption.

For first-order logic, I have much less intuition. Clearly, the present results are very far from optimal. Just as an illustration, observe that if a class $\mathcal{C}$ of graphs has a tractable first-order model checking problem, then so has the closure of $\mathcal{C}$ under complementation. (Recall that the *complement* of a graph $G = (V, E)$ is the graph $\bar{G} = \Big(V, \binom{V}{2} \setminus E\Big)$.) However, most of the classes we considered here are not closed under complementation.

8.3 Average case analysis

Instead of the worst case running time, it is also interesting to consider the average case. Here even the most basic questions are wide open. For $n \geq 1$, let $\mathcal{W}_n$ be a probability space of graphs with vertex set $[n]$. We say that a model checking algorithm is *fpt on average over* $\big(\mathcal{W}_n\big)_{n\geq 1}$ if its expected running time on input $G \in \mathcal{W}_n$ and φ is bounded by $f(|\varphi|) \cdot n^{O(1)}$, for some computable function f. For every function $p : \mathbb{N} \to [0,1]$ (here $[0,1]$ denotes an interval of real numbers), let $\mathcal{G}(n,p)$ denote the probability space of all graphs over $[n]$ with edge probability $p(n)$ (see, e.g., [2]). For a constant $c \in [0,1]$, we let $\mathcal{G}(n,c) = \mathcal{G}(n,p)$ for the constant function $p(n) = c$. In [46], I observed that for $p(n) = \min\{1, c/n\}$, where $c \in \mathbb{R}_{\geq 0}$ is a constant, there is a model checking algorithm for first-order logic that is fpt on average over $\big(\mathcal{G}(n,p)\big)_{n\geq 1}$.

Open Problem 8.4. Is there a model checking algorithm for first-order logic that is fpt on average over $\big(\mathcal{G}(n,1/2)\big)_{n\geq 1}$?

Let me suggest it as an exercise for the reader to design a model checking algorithm for existential first-order logic that is fpt on average over $\big(\mathcal{G}(n,1/2)\big)_{n\geq 1}$.

8.4 Structures of bounded rank width

Most of the results of this survey can easily be extended from classes $\mathcal{C}$ of graphs to the classes $\mathcal{C}_{\mathrm{str}}$ of arbitrary relational structures whose underlying graphs (Gaifman graphs) are in $\mathcal{C}$. However, this is not true for the results that involve rank width. It is not at all obvious what an appropriate notion of rank width for arbitrary structures could look like, and I think it is a challenging open problem to find such a notion.

8.5 Model checking for monadic least fixed-point logic

Conjecture 8.5. Let $\mathcal{C}$ be a class of graphs such that p-MC(FO,$\mathcal{C}_{\mathrm{lb}}$) is fixed-parameter tractable. Then p-MC(LFP^1,$\mathcal{C}_{\mathrm{lb}}$) is fixed-parameter tractable.

It will be difficult to prove this conjecture, because it is related to the notoriously open problem of whether the model checking problem for the modal μ-calculus is in polynomial time. But maybe the conjecture is wrong; refuting it might be more feasible.

References

[1] I. Abraham, C. Gavoille, and D. Malkhi. Compact routing for graphs excluding a fixed minor. In P. Fraigniaud, editor, *DISC*, volume 3724 of *Lecture Notes in Computer Science*, pages 442–456. Springer, 2005.

[2] N. Alon and J. H. Spencer. *The probabilistic method.* Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience [John Wiley & Sons], New York, second edition, 2000. With an appendix on the life and work of Paul Erdős.

[3] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[4] G. Bagan. Mso queries on tree decomposable structures are computable with linear delay. In Z. Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.

[5] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[6] H. L. Bodlaender and D. M. Thilikos. Constructive linear time algorithms for branchwidth. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP*, volume 1256 of *Lecture Notes in Computer Science*, pages 627–637. Springer, 1997.

[7] Y. Chen, M. Grohe, and M. Grüber. On parameterized approximability. In H. L. Bodlaender and M. A. Langston, editors, *IWPEC*, volume 4169 of *Lecture Notes in Computer Science*, pages 109–120. Springer, 2006.

[8] B. Courcelle. An axiomatic definition of context-free rewriting and its application to nlc graph grammars. *Theor. Comput. Sci.*, 55(2-3):141–181, 1987.

[9] 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 (B)*, pages 193–242. Elsevier Science Publishers, Amsterdam, 1990.

[10] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[11] B. Courcelle. The monadic second-order logic of graphs vii: Graphs as relational structures. *Theor. Comput. Sci.*, 101(1):3–33, 1992.

[12] B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars*, pages 313–400. World Scientific, 1997.

[13] B. Courcelle. Linear delay enumeration and monadic second-order logic, 2006. Available at http://www.labri.fr/perso/courcell/ActSci.html.

[14] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[15] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Applied Mathematics*, 108(1-2):23–52, 2001.

[16] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1&2):49–82, 1993.

[17] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[18] A. Dawar, M. Grohe, and S. Kreutzer. Locally excluding a minor. In *LICS*, pages 270–279. IEEE Computer Society, 2007.

[19] A. Dawar, M. Grohe, S. Kreutzer, and N. Schweikardt. Approximation schemes for first-order definable optimisation problems. In *LICS*, pages 411–420. IEEE Computer Society, 2006.

[20] A. Dawar, M. Grohe, S. Kreutzer, and N. Schweikardt. Model theory makes formulas large. In L. Arge, C. Cachin, T. Jurdzinski, and A. Tarlecki, editors, *ICALP*, volume 4596 of *Lecture Notes in Computer Science*, pages 913–924. Springer, 2007.

[21] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on graphs of bounded-genus and -minor-free graphs. In Munro [59], pages 830–839.

[22] E. D. Demaine and M. T. Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In Munro [59], pages 840–849.

[23] E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Decomposition, approximation, and coloring. In *FOCS*, pages 637–646. IEEE Computer Society, 2005.

[24] E. D. Demaine, M. T. Hajiaghayi, and K. Kawarabayashi. Algorithmic graph minor theory: Improved grid minor bounds and wagner's contraction. In S. K. Madria, K. T. Claypool, R. Kannan, P. Uppuluri, and M. M. Gore, editors, *ISAAC*, volume 4317 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2006.

[25] R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics.* Springer-Verlag, Berlin, third edition, 2005.

[26] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

[27] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness ii: On completeness for w[1]. *Theor. Comput. Sci.*, 141(1&2):109–131, 1995.

[28] R. G. Downey and M. R. Fellows. *Parameterized complexity.* Monographs in Computer Science. Springer-Verlag, New York, 1999.

[29] R. G. Downey, M. R. Fellows, and U. Taylor. The parameterized complexity of relational database queries and an improved characterization of $W[1]$. In *Combinatorics, complexity, & logic (Auckland, 1996)*, Springer Ser. Discrete Math. Theor. Comput. Sci., pages 194–213, Singapore, 1997. Springer.

[30] A. Durand and E. Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. on Comput. Log.* To appear.

[31] H.-D. Ebbinghaus and J. Flum. *Finite model theory.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, second edition, 1999.

[32] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical logic.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, second edition, 1994. Translated from the German by Margit Meßmer.

[33] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3), 1999.

[34] D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000.

[35] P. Erdős. Graph theory and probability. *Canad. J. Math.*, 11:34–38, 1959.

[36] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of computation (Proc. SIAM-AMS Sympos. Appl. Math., New York, 1973)*, pages 43–73. SIAM–AMS Proc., Vol. VII, Providence, R.I., 1974. Amer. Math. Soc.

[37] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[38] J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model-checking. *SIAM J. Comput.*, 31(1):113–145, 2001.

[39] J. Flum and M. Grohe. *Parameterized complexity theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

[40] M. Frick. *Easy Instances for Model Checking.* PhD thesis, Albert-Ludwigs-Universität Freiburg, 2001.

[41] M. Frick. Generalized model-checking over locally tree-decomposable classes. In H. Alt and A. Ferreira, editors, *STACS*, volume 2285 of *Lecture Notes in Computer Science*, pages 632–644. Springer, 2002.

[42] M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

[43] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

[44] H. Gaifman. On local and nonlocal properties. In *Proceedings of the Herbrand symposium (Marseilles, 1981)*, volume 107 of *Stud. Logic Found. Math.*, pages 105–135, Amsterdam, 1982. North-Holland.

[45] M. R. Garey and D. S. Johnson. *Computers and intractability.* W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[46] M. Grohe. Generalized model-checking problems for first-order logic. In A. Ferreira and H. Reichel, editors, *STACS*, volume 2010 of *Lecture Notes in Computer Science*, pages 12–26. Springer, 2001.

[47] M. Grohe. Local tree-width, excluded minors, and approximation algorithms. *Combinatorica*, 23(4):613–632, 2003.

[48] M. Grohe, S. Kreutzer, and N. Schweikardt. The expressive power of two-variable least fixed-point logics. In J. Jedrzejowicz and A. Szepietowski, editors, *MFCS*, volume 3618 of *Lecture Notes in Computer Science*, pages 422–434. Springer, 2005.

[49] M. Grohe and J. Mariño. Definability and descriptive complexity on databases of bounded tree-width. In C. Beeri and P. Buneman, editors, *ICDT*, volume 1540 of *Lecture Notes in Computer Science*, pages 70–82. Springer, 1999.

[50] M. Grohe and S. Wöhrle. An existential locality theorem. *Ann. Pure Appl. Logic*, 129(1-3):131–148, 2004.

[51] P. Hlinený and S. Oum. Finding branch-decompositions and rank-decompositions. In L. Arge, M. Hoffmann, and E. Welzl, editors, *ESA*, volume 4698 of *Lecture Notes in Computer Science*, pages 163–174. Springer, 2007.

[52] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.

[53] K. Kawarabayashi and B. Mohar. Approximating the list-chromatic number and the chromatic number in minor-closed and odd-minor-closed classes of graphs. In Kleinberg [54], pages 401–416.

[54] J. M. Kleinberg, editor. *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006.* ACM, 2006.

[55] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[56] L. Libkin. *Elements of finite model theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2004.

[57] S. Lindell. Computing monadic fixed-points in linear-time on doubly-linked data structures, 2005. Available at http://www.haverford.edu/cmsc/slindell/.

[58] J. A. Makowsky. Algorithmic uses of the feferman-vaught theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004.

[59] J. I. Munro, editor. *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*. SIAM, 2004.

[60] J. Nesetril and P. O. de Mendez. Linear time low tree-width partitions and algorithmic consequences. In Kleinberg [54], pages 391–400.

[61] J. Nešetřil and P. O. de Mendez. Grad and classes with bounded expansion I: Decompositions. *European J. Combin.*, 2007. To appear.

[62] J. Nešetřil and P. O. de Mendez. Grad and classes with bounded expansion II: Algorithmic aspects. *European J. Combin.*, 2007. To appear.

[63] R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

[64] S. Oum. Rank-width is less than or equal to branch-width, 2006. Available at http://www.math.uwaterloo.ca/˜sangil/.

[65] S. Oum and P. D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006.

[66] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.

[67] N. Robertson and P. D. Seymour. Graph minors I–XXIII. Appearing in *Journal of Combinatorial Theory, Series B* since 1982.

[68] N. Robertson and P. D. Seymour. Graph minors XXI. Graphs with unique linkages. *J. Combin. Theory Ser. B.* To appear.

[69] N. Robertson and P. D. Seymour. Graph minors XXII. Irrelevant vertices in linkage problems. To appear.

[70] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[71] N. Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986.

[72] N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991.

[73] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[74] N. Robertson and P. D. Seymour. Graph minors XVI. Excluding a non-planar graph. *J. Combin. Theory Ser. B*, 77:1–27, 1999.

[75] N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004.

[76] N. Schweikardt. On the expressive power of monadic least fixed point logic. *Theor. Comput. Sci.*, 350(2-3):325–344, 2006.

[77] D. Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996.

[78] H. Tamaki. A linear time heuristic for the branch-decomposition of planar graphs. In G. D. Battista and U. Zwick, editors, *ESA*, volume 2832 of *Lecture Notes in Computer Science*, pages 765–775. Springer, 2003.

[79] R. E. Tarjan. Decomposition by clique separators. *Discrete Math.*, 55(2):221–232, 1985.

[80] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[81] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146. ACM, 1982.

[82] M. Y. Vardi. On the complexity of bounded-variable queries. In *PODS*, pages 266–276. ACM Press, 1995.

[83] K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Math. Ann.*, 114(1):570–590, 1937.

Non-regular fixed-point logics and games

Stephan Kreutzer[1]
Martin Lange[2]

[1] Oxford University Computing Laboratory
Wolfson Building
Parks Road
Oxford, OX1 3QD, England
`kreutzer@comlab.ox.ac.uk`

[2] Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstraße 67
80538 München, Germany
`martin.lange@ifi.lmu.de`

Abstract

The modal μ-calculus is—despite strictly subsuming many other temporal logics—in some respect quite limited in expressive power: it is equi-expressive to the bisimulation-invariant fragment of Monadic Second-Order Logic over words, trees, or graphs. Hence, properties expressible in the modal μ-calculus are inherently regular.

Motivated by specifications that reach beyond the regularity bound, we introduce extensions of the modal μ-calculus that can define non-regular properties. We focus on two modal fixed-point logics: the Modal Iteration Calculus (MIC) which uses inflationary instead of least fixed-point quantifiers, and Fixed-Point Logic with Chop (FLC) which incorporates sequential composition into the modal μ-calculus. We compare these two different approaches to increased expressiveness. In particular, we show how a common set of specifications can be formalised in each of them and give an overview of known results.

The modal μ-calculus also enjoys a nice game-theoretic characterisation: its model checking problem is equivalent to the problem of solving a parity game. We also show how to characterise the model checking problems for MIC and FLC in this way, making use of appropriate extensions of parity games, namely stair parity and backtracking games.

1 Introduction

Modal and temporal logics. The most commonly used specification logics in the theory of computer aided verification are based on propositional modal logic augmented by temporal operators. Among those one can broadly distinguish between linear and branching time logics, depending on how they treat the temporal development of processes. The modal μ-calculus, $\mathcal{L}_\mu$ for short, provides a common generalization of most temporal logics. It is defined as the extension of basic propositional modal logic by rules to form the least and the greatest fixed point of definable monotone operators.

$\mathcal{L}_\mu$ is a *regular* logic in the sense that it can be translated into monadic second order logic (MSO) and therefore can only define regular classes of trees and their representations as transition systems. It is even equi-expressive to the bisimulation-invariant fragment of MSO over trees or graphs [9] and can therefore be seen as *the* regular branching time temporal logic.

Temporal logics such as LTL, CTL or CTL* are all embeddable into $\mathcal{L}_\mu$. They can express important properties—such as reachability, safety, liveness, fairness, etc.—and specifications in these languages can be verified automatically and in many cases also efficiently in process models. However, a number of natural properties of processes are no longer regular and therefore cannot be expressed in any of these logics. For instance, one cannot express that a specific event occurs in all possible execution traces at *the same time* [7], that every transmission is acknowledged, or that there are no more *returns* than *calls*.

To express these properties in a logic, the logic needs to be able to count to some extent, at least to compare cardinalities, i.e. it needs to incorporate non-regular properties. There are various potential ways of defining logics with non-regular features.

One option is to add a bisimulation preserving form of counting explicitly, i.e. to consider a modal analogue to first-order plus counting. Similarly, one could add specific operators for the tasks at hand, an operator to compare cardinalities, for instance. In this way, logics tailored towards specific tasks can be obtained.

Another possibility is to enrich the models over which a regular logic is interpreted with some extra information and let the operators of the logic make use of this. This has been done in the linear time temporal logic CaRet for example [1]. It is interpreted in an LTL-like fashion over infinite words that represent runs of recursive processes, i.e. positions in these words are marked with *call* and *return* symbols. CaRet then extends LTL by allowing its operators to access *return* positions that match the previous *call* position in the sense that in between the *calls* and *returns* form a balanced Dyck-language. This way, non-regularity is added into the meta-logic rather than the logic itself.

A different approach is to consider general purpose logics employing more expressive fixed-point constructs than least fixed points of monotone operators. This is the trait we follow in this paper. There are (at least) two ways in which the modal μ-calculus can be extended in this way: one can relax the restriction to monotone operators or one can stick to monotone operators but allow fixed-point inductions of higher order. We consider these options and introduce two modal fixed-point logics: (1) the Modal Iteration Calculus (MIC) which replaces least and greatest fixed points in $\mathcal{L}_\mu$ by inflationary and deflationary ones; and (2) Fixed-Point Logic with Chop (FLC) which extends $\mathcal{L}_\mu$ with an operator for sequential composition. This necessitates a higher-order semantics.

Non-regular properties. We illustrate these logics by a set of examples of non-regular properties, i.e. properties that cannot be expressed in $\mathcal{L}_\mu$.

The most obvious choices come from formal language theory. The first hurdle to take for a logic that wants to be able to express non-regular properties is the standard example of a context-free and non-regular language, i.e. $L = \{a^n b^n \mid n \geq 1\}$. Note that MIC and FLC are branching time logics, and hence, we shall look for formulas that are satisfied by a state if, and only if, it has a maximal outgoing path whose labels form a word in L. While this is a toy example, there are also formal languages which give rise to interesting program correctness properties. Let $\Sigma = \{a, b\}$ and consider the language L consisting of all words that do not have a prefix in which there are more b's than a's. It is easily seen to be non-regular but context-free, and it is the formal language basis of the aforementioned property about *calls* and *returns*. A suitable reformulation of this language in a formula of MIC or FLC would show that these logics can express properties of recursive processes like "no process is ended unless it has been started" etc. Note that this is also the same as absence of underflows in FIFO or LIFO buffers of unbounded size.

Non-regularity, however, need not be rooted in the theory of formal word languages. Branching time logics whose expressive power exceeds that of $\mathcal{L}_\mu$ may also be able to express properties that are unrelated to context-free languages. For example, the aforementioned *uniform inevitability* property – some event occurs in all executions at the same time—cannot be expressed by a finite tree automaton. As we shall see, it can be expressed in both MIC and FLC. Note that this is a generalization of the property of being bisimilar to a balanced tree—the globally occurring event is just a deadlock in this case.

Games. Closely related to modal logics are games since model checking problems for modal logics often have game-theoretic characterizations. Games in this context are played by two players who push a token along a path through the game arena formed by some product of the underlying structure and the syntax tree of the formula at hand. The logic influences the type of winning condition.

Modal logic for instance induces simple reachability games, while the fixed-point recursion mechanism in the modal μ-calculus requires games with winning conditions on infinite plays, namely parity games [18].

There is often a reverse connection between games and modal logics as well. Game graphs can be seen as labeled transition systems again, and it is reasonable to ask whether the winning regions—the parts from which one of the players has a winning strategy—can in turn be defined by a formula of that logic. This is the case for the modal μ-calculus and parity games.

As the logics considered here are proper extensions of $\mathcal{L}_\mu$, this gives an intuitive explanation of why simple parity games do not suffice to characterize their model checking problems. Instead, an interesting game model for the logics presented here is that of *stair parity games* which are played on the configuration graph of a visibly pushdown system [15]. The name is due to the fact that the parity condition is not evaluated on the whole of a play but only on that part that looks like stairs w.r.t. the stacks involved in these games. We show how the model checking problems for both MIC and FLC can be characterized by stair parity games.

Outline. The paper is organized as follows. Section 2 contains preliminary definitions about transition systems and recalls some necessary fixed-point theory and the modal μ-calculus. In Section 3 we then introduce MIC and FLC formally and give examples of formulas defining non-regular properties in these logics. At the end of this section we compare the two logics by giving an overview of the known complexity and expressivity results about them. Section 4 then defines stair parity games and shows how to characterize MIC's and FLC's model checking problems by them. We also introduce *backtracking games*, which are non-regular games extending ordinary parity games in a different way. They were originally introduced as game model for inflationary fixed-point logics. Finally, Section 5 concludes the paper with some remarks about further research.

2 Preliminaries

Labeled transition systems. For the remainder of this paper we fix a finite non-empty set $\mathcal{A}$ of *actions* and $\mathcal{P}$ of *proposition symbols.*

A *labeled transition system* is a structure $\mathcal{T} := (\mathcal{S}, \{\xrightarrow{a} : a \in \mathcal{A}\}, L)$, where $\mathcal{S}$ is a finite non-empty set of states, $\xrightarrow{a}$ is a binary relation on states for each $a \in \mathcal{A}$, and $L : \mathcal{S} \to 2^{\mathcal{P}}$ is a function labeling each state s with the set of propositions true at s.

Fixed-point theory. Let A be a set and $F : 2^A \to 2^A$ be a function. F is called *monotone* if $F(X) \subseteq F(Y)$ for all $X \subseteq Y \subseteq A$. A *fixed point* of F is any set $P \subseteq A$ such that $F(P) = P$. A *least fixed point of* F is a fixed point that is contained in any other fixed point of F.

It is a consequence of the Knaster-Tarski theorem [19] that every monotone function $F : 2^A \to 2^A$ has a least and a greatest fixed point, written as $\mathbf{lfp}(F)$ and $\mathbf{gfp}(F)$, which can be defined as

$$\mathbf{lfp}(F) \;:=\; \bigcap\{X \subseteq A : F(X) = X\} \;=\; \bigcap\{X \subseteq A : F(X) \subseteq X\},$$

and

$$\mathbf{gfp}(F) \;:=\; \bigcup\{X \subseteq A : F(X) = X\} \;=\; \bigcup\{X \subseteq A : F(X) \supseteq X\}.$$

Least fixed points of monotone operators can also be obtained inductively by the ordinal-indexed sequence X^α of subsets of A defined as

$$X^0 \;:=\; \varnothing \;, \quad X^{\alpha+1} \;:=\; F(X^\alpha) \;, \quad X^\kappa \;:=\; \bigcup_{\alpha<\kappa} X^\alpha$$

where κ is a limit ordinal. As F is monotone, this sequence of sets is increasing, i.e. for all α, β: if $\alpha < \beta$ then $X^\alpha \subseteq X^\beta$, and therefore reaches a fixed point X^∞, with $X^\infty := X^\alpha$ for the least ordinal α such that $X^\alpha = X^{\alpha+1}$. The fixed point X^∞ is called the *inductive fixed point of* F. Again it follows from Knaster and Tarski's theorem that for every monotone operator $F : 2^A \to 2^A$, the least and the inductive fixed point coincide.

Similarly, the greatest fixed point of a monotone operator can also be defined inductively by the following sequence of sets:

$$X^0 \;:=\; A \;, \quad X^{\alpha+1} \;:=\; F(X^\alpha) \;, \quad X^\kappa \;:=\; \bigcap_{\alpha<\kappa} X^\alpha$$

where, again, κ is a limit ordinal.

Least and greatest fixed points are dual to each other. For every operator F define the dual operator $F^d : X \mapsto (F(X^c))^c$ where $X^c := A \setminus X$. If F is monotone, then F^d is also monotone and we have that

$$\mathbf{lfp}(F) = (\mathbf{gfp}(F^d))^c \quad \text{and} \quad \mathbf{gfp}(F) = (\mathbf{lfp}(F^d))^c.$$

The modal μ-calculus. We briefly recall the definition of $\mathcal{L}_\mu$. Let $\mathcal{V}$ be a countable infinite set of variables. The formulas of $\mathcal{L}_\mu$ are given by the following grammar.

$$\varphi ::= q \mid \neg q \mid X \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid [a]\varphi \mid \langle a\rangle\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where $q \in \mathcal{P}$, $a \in \mathcal{A}$, and $X \in \mathcal{V}$. The semantics of $\mathcal{L}_\mu$ is that of basic modal logic where in addition formulas $\mu X.\varphi$ and $\nu X.\varphi$ are interpreted as follows. On any labeled transition system $\mathcal{T}$ with state set $\mathcal{S}$, an $\mathcal{L}_\mu$-formula $\varphi(X)$ with free variable $X \in \mathcal{V}$ induces an operator $F_\varphi : 2^{\mathcal{S}} \to 2^{\mathcal{S}}$ which takes a set U of states to the set $[\![\varphi]\!]^{\mathcal{T}}_{X\mapsto U}$. Here, we write $[\![\varphi]\!]^{\mathcal{T}}_{X\mapsto U}$ for the set of states from $\mathcal{T}$ at which the formula φ holds under the interpretation that interprets the variable X by the set U. As, by definition, X occurs only positively in φ, this operator is monotone. We define $[\![\mu X.\varphi]\!]^{\mathcal{T}} := \mathbf{lfp}(F_\varphi)$ and $[\![\nu X.\varphi]\!]^{\mathcal{T}} := \mathbf{gfp}(F_\varphi)$.

Notation 2.1. Sometimes we want to speak about transitions labeled with any action, and therefore use the abbreviations $\Diamond\varphi := \bigvee_{a\in\mathcal{A}}\langle a\rangle\varphi$, and $\Box\varphi := \bigwedge_{a\in\mathcal{A}}[a]\varphi$. We shall also use terms $\mathtt{tt} := q \vee \neg q$, $\mathtt{ff} := q \wedge \neg q$ for some $q \in \mathcal{P}$.

3 Non-regular logics

In this section we introduce two extensions of the modal μ-calculus by non-regular constructs. We first recall the *Modal Iteration Calculus*, introduced in [4] which incorporates inflationary fixed points into $\mathcal{L}_\mu$. In Section 3.2 we then introduce the *Fixed-Point Logic with Chop*, introduced in [16], based on extending $\mathcal{L}_\mu$ by sequential composition. To illustrate the logics and to help comparing them, we exhibit a set of examples and give formalizations for them in both logics.

3.1 The Modal Iteration Calculus

Informally, MIC is propositional modal logic ML, augmented with simultaneous inflationary fixed points.

3.1.1 Syntax and Semantics

Definition 3.1. Let $\mathcal{V}$ be a countable infinite set of variables. The formulas of the *Modal Iteration Calculus* (MIC) are given by the following grammar.

$$\varphi ::= q \mid X \mid \neg\varphi \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid [a]\varphi \mid \langle a\rangle\varphi \mid \mathbf{ifp}X.S \mid \mathbf{dfp}X.S$$

where $X \in \mathcal{V}$, $q \in \mathcal{P}$, $a \in \mathcal{A}$, and

$$S := \begin{cases} X_1 & \leftarrow & \varphi_1 \\ & \vdots & \\ X_k & \leftarrow & \varphi_k \end{cases}$$

is a *system* of rules with $\varphi_i \in$ MIC and $X_i \in \mathcal{V}$ for $1 \le i \le k$. If S consists of a single rule $X \leftarrow \varphi$ we simplify the notation and write $\mathbf{ifp}X.\varphi$ instead of $\mathbf{ifp}\ X.\{X \leftarrow \varphi\}$.

We define $\mathrm{Sub}(\varphi)$ as the set of subformulas of φ as usual. In particular, the variables X_i occurring on the left-hand side of rules in a system S as above count as subformulas. The semantics of the various operators are as in propositional modal logic with the semantics of **ifp** and **dfp** being as follows. On every transition system $\mathcal{T} := (\mathcal{S}, \{\xrightarrow{a} : a \in \mathcal{A}\}, L)$, the system S defines, for each ordinal α, a tuple $\bar{X}^\alpha = (X_1^\alpha, \ldots, X_k^\alpha)$ of sets of states, via the following inflationary induction:

$$\begin{aligned} X_i^0 &:= \varnothing, \\ X_i^{\alpha+1} &:= X_i^\alpha \cup [\![\varphi_i]\!]^{\mathcal{T}}_{\bar{X} \mapsto \bar{X}^\alpha}, \\ X_i^\kappa &:= \bigcup_{\alpha<\kappa} X_i^\alpha \end{aligned}$$

where κ is a limit ordinal. We call $(X_1^\alpha, \ldots, X_k^\alpha)$ the α-th stage of the inflationary induction of S on $\mathcal{T}$. As the stages are increasing (i.e. $X_i^\alpha \subseteq X_i^\beta$ for any $\alpha < \beta$), this induction reaches a fixed point $(X_1^\infty, \ldots, X_k^\infty)$. Now we put $[\![(\mathbf{ifp}\ X_i : S)]\!]^{\mathcal{T}} := X_i^\infty$.

The semantics of the deflationary fixed-point operator is defined analogously as the i-th component of the deflationary fixed point $(X_1^\infty, \ldots, X_k^\infty)$ obtained from the sequence $X_i^0 := \mathcal{S}$, $X_i^{\alpha+1} := X_i^\alpha \cap [\![\varphi_i]\!]^{\mathcal{T}}_{\bar{X} \mapsto \bar{X}^\alpha}$, and $X_i^\kappa := \bigcap_{\alpha<\kappa} X_i^\alpha$.

3.1.2 Properties Expressible in MIC

We demonstrate the Modal Iteration Calculus by some examples. It is immediately clear from the definition that every $\mathcal{L}_\mu$-formula is equivalent to a MIC-formula (by replacing every μ-operator by **ifp** and ν-operator by **dfp**). We shall therefore use least fixed points as well as inflationary fixed points in the examples below.

Example 3.2. Let us first consider the language $\{a^n b^n \mid n \geq 1\}$ mentioned above. We model a finite word by a transition system consisting of a simple path whose edges are labeled by the letters in the word. For example, the word $aabb \in L$ is modeled by the system $\bullet \xrightarrow{a} \bullet \xrightarrow{a} \bullet \xrightarrow{b} \bullet \xrightarrow{b} \bullet$. [1]

Using this encoding of words, the language L can be defined as follows. The formula $\varphi := \langle a \rangle \mathtt{tt} \wedge \mathtt{EF}(\langle b \rangle \mathtt{tt}) \wedge \neg \mathtt{EF}(\langle b \rangle \langle a \rangle \mathtt{tt})$ defines all words starting with an a, containing at least one b, and where all b's come after all a's, i.e. the language a^+b^+. Here, $\mathtt{EF}(\vartheta)$ is the $\mathcal{L}_\mu$ formula $\mu R.(\vartheta \vee \Diamond R)$ saying that a state satisfying φ is reachable. Within a^+b^+ the language L can then be defined by the formula

$$\neg \left(\mathbf{ifp} Z. \begin{cases} X & \leftarrow \langle b \rangle (\Box \mathtt{ff} \vee X) \\ Y & \leftarrow \langle a \rangle (\neg \langle a \rangle \mathtt{tt} \wedge \langle b \rangle \mathtt{tt}) \vee \langle a \rangle Y \\ Z & \leftarrow (\langle a \rangle Y \wedge \neg \mathtt{EF}(\langle a \rangle \langle b \rangle \neg X)) \vee (\neg \langle a \rangle Y \wedge \mathtt{EF}(\langle a \rangle \langle b \rangle X)) \end{cases} \right).$$

[1] There are two common ways of modeling a word by a transition system: labeling edges by letters, as we do it here, or labeling states by the corresponding letters. For MIC, the latter is often more convenient and helps to simplify formulas. For the logic FLC, which we shall consider below, the formalization used here is preferable. To unify the examples for MIC and FLC we prefer to use the edge labeling model for both logics.

We demonstrate the evaluation of the fixed points by the following two words $w_1 \in L$ and $w_2 \notin L$.

$$w_1 := 1 \xrightarrow{a} 2 \xrightarrow{a} 3 \xrightarrow{b} 4 \xrightarrow{b} 5 \qquad w_2 := 1 \xrightarrow{a} 2 \xrightarrow{a} 3 \xrightarrow{a} 4 \xrightarrow{b} 5 \xrightarrow{b} 6$$

$$\begin{array}{lcl} X^1 & := & \{4\} \\ Y^1 & := & \{2\} \\ Z^1 & := & \varnothing \\ \\ X^2 & := & \{3,4\} \\ Y^2 & := & \{1,2\} \\ Z^2 & := & \varnothing \end{array} \qquad \begin{array}{lcl} X^1 & := & \{5\} \\ Y^1 & := & \{3\} \\ Z^1 & := & \varnothing \\ \\ X^2 & := & \{4,5\} \\ Y^2 & := & \{2,3\} \\ Z^2 & := & \{1\} \\ & \ldots & \end{array}$$

Hence, w_1 satisfies the formula whereas w_2 does not. The idea is that at stage i of the fixed-point induction, X^i contains all states from which a b-labeled path of length i leads to a leaf. To define the induction on Y, let u be a state in $\mathcal{T}$ which has an incoming a-transition but only outgoing b-transitions, i.e.

$$\ldots \bullet \xrightarrow{a} u \xrightarrow{b} \bullet \ldots$$

Note that for words in a^+b^+ this state is unique. The state u is "in the middle" of the word. Then Y^i contains all states from which there is a path to u of length i labeled by a's.

Finally, a state occurs in Z if at some stage i its a-successor is in Y^i but the b-successor of u is not in X^i or vice versa. Hence, the root occurs in Z if the labels of the path leading from the root to the leaf is not a word in $a^n b^n$.

The example demonstrates a general technique of how counting can be implemented in MIC: we let an induction on a variable X start at a leaf and in each iteration proceed to a predecessor of a state already contained in X. At each stage i, X^i contains the states of distance at most i from a leaf. We can then use a formula $\neg X \wedge \Box X$ to define the states of distance exactly i from a leaf. This techniques is employed in various proofs showing expressibility and complexity results for MIC. We demonstrate it in the following example, where we define the class of transition systems bisimilar to a well-founded tree of finite height. Here the height of a leaf is 0 and the height of an inner node is the maximum height of its successors plus 1.

Example 3.3. Let $\mathcal{T} := (\mathcal{S}, \{\xrightarrow{a} : a \in \mathcal{A}\}, L)$ be a transition system and $s \in \mathcal{S}$. Then $s \in [\![\mu X.\Box X]\!]^{\mathcal{T}}$ if, and only if, there is no infinite path emerging from s, i.e. $(\mathcal{T}, s)$ is bisimilar to a well-founded tree—disregarding labels. Using a similar trick as in the previous example, we can define all nodes of infinite height in a well-founded tree. For this, consider the formula

$$\varphi \;:=\; \mathbf{ifp}\ Z.\begin{cases} X & \leftarrow \Box X \\ Y & \leftarrow X \\ Z & \leftarrow \big(\Box X \wedge \Diamond\neg Y\big) \vee \Box\mathtt{ff} \end{cases}$$

After $\alpha < \omega$ iterations, the stage X^α contains all nodes of height $< \alpha$ and Y^α contains all nodes of height $< \alpha - 1$. Hence, every node of finite height will at some point have all its successors in X but at least one successor outside of Y (except for the leaves which are included into Z by the disjunct $\Box\mathtt{ff}$) and therefore after ω iterations Z contains all nodes of finite height. However, as $X^\omega = Y^\omega$ a node r of height exactly ω will never occur in Z. Hence, a tree has finite height if, and only if, its root satisfies $\mu X.\Box X \wedge \neg\mathtt{EF}(\neg\varphi)$.

The next example shows how to define the class of transition systems bisimilar to balanced trees of finite height.[2]

Example 3.4. All that remains is to define in the class of trees of finite height the class of balanced trees. This is done by the formula

$$\neg\left(\mathbf{ifp}\ Y.\begin{cases} X & \leftarrow \Box X \\ Y & \leftarrow \Diamond X \wedge \Diamond\neg X \end{cases}\right).$$

Again, for $i > 0$, the i-th stage X^i contains all states from which no path of length $\geq i$ emerges. Hence, a state occurs in Y if it has two successors of different length.

Finally, we give an example showing that the class of all transition systems which are bisimilar to a word of finite length is MIC-definable.

Example 3.5. We have already seen in the previous examples that we can axiomatize transition systems bisimilar to balanced trees of finite height. So all that is left to do is to give a formula that defines in such trees that all paths carry the same labels. This is easily expressed by the formula

$$\neg\left(\mathbf{ifp}\ Y.\begin{cases} X & \leftarrow \Box X \\ Y & \leftarrow \bigwedge\limits_{a\in\mathcal{A}}\left(\begin{array}{l}\mathtt{EF}(\neg X \wedge \Box X \wedge \langle a\rangle X) \wedge \\ \mathtt{EF}(\neg X \wedge \Box X \wedge \bigvee\limits_{b\in\mathcal{A},a\neq b}\langle b\rangle X)\end{array}\right) \end{cases}\right).$$

Using similar tricks we can express buffer underflow in finite words, i.e. the context-free language

$$L := \{w \in \{a,b\}^* \mid \forall u,v : w = uv \Rightarrow |u|_b \leq |u|_a\}.$$

[2] Note that all we can hope for is to define trees of finite height, as finiteness itself is not preserved under bisimulation and hence not definable in *any* modal logic.

Here, $|u|_b$ denotes the number of b's in the word u and likewise for $|u|_a$. It is not known, however, whether the "buffer underflow" and "bisimilarity-to-a-word" formulas can be amended for infinite words as well. The problem is that there no longer is a natural starting point for fixed-point inductions.

3.2 Fixed-Point Logic with Chop

We proceed by introducing a different extension of the modal μ-calculus. It differs from MIC in that we again consider monotone inductions only, but the individual fixed-point stages are no longer sets of states but monotone functions from the complete lattice of all monotone operators over the state space.

3.2.1 Syntax and Semantics

Let $\mathcal{P}$ and $\mathcal{A}$ be as before, and $\mathcal{V}$ be a countable infinite set of variable names. Formulas of *Fixed-Point Logic with Chop* (FLC) over $\mathcal{P}$, $\mathcal{A}$ and $\mathcal{V}$ are given by the following grammar.

$$\varphi \ ::= \ q \mid \neg q \mid X \mid \tau \mid \langle a \rangle \mid [a] \mid (\varphi \vee \varphi) \mid (\varphi \wedge \varphi) \mid (\varphi ; \varphi) \mid \mu X.\varphi \mid \nu X.\varphi$$

where $q \in \mathcal{P}$, $a \in \mathcal{A}$, and $X \in \mathcal{V}$. We shall write σ for either μ or ν. In the following, we shall also omit parentheses and introduce the convention that ";" binds stronger than the Boolean operators which, in turn, bind stronger than fixed-point quantifiers.

The set of subformulas $\mathrm{Sub}(\varphi)$ of an FLC formula φ is defined as usual, for example $\mathrm{Sub}(\sigma X.\varphi) = \{\sigma X.\varphi\} \cup \mathrm{Sub}(\varphi)$, etc. Also, we assume that variables are quantified at most once in each formula. Hence, each φ comes with a function fp_φ which associates to each variable X in $\mathrm{Sub}(\varphi)$ its defining fixed-point formula $\sigma X.\psi$.

FLC extends the modal μ-calculus $\mathcal{L}_\mu$ with the sequential composition ("chop") operator $\_;\_$. Remember that variables in $\mathcal{L}_\mu$ formulas can only occur in rightmost positions within Boolean formulas, possibly prefixed by modal operators. This gives an intuitive explanation of the fact that the expressive power of $\mathcal{L}_\mu$ is restricted to regular languages of infinite words or trees—formulas of $\mathcal{L}_\mu$ resemble (alternating) right-linear grammars with modal operators as terminal symbols.

Variables in FLC formulas, however, can also be suffixed with modal operators through the use of sequential composition, e.g. $\langle a \rangle ; X ; \langle b \rangle$. Since this is supposed to generalize the restricted composition of modal operators with formulas on the right, there is no need to include formulas of the form $\langle a \rangle \varphi$ in FLC. Instead, this is supposed to be simulated by $\langle a \rangle ; \varphi$, and this is why modal operators are chosen as atomic formulas in FLC.

The semantics of the modal μ-calculus cannot simply be extended by clauses for the additional operators in FLC, in particular not for sequential composition. Remember that the semantics of $\mathcal{L}_\mu$ assigns to each formula and environment interpreting its free variables a *set of states* of the underlying LTS. In other words, the semantics of a $\mathcal{L}_\mu$ formula is a *predicate*.

In order to interpret sequential composition naturally, the semantics of FLC lifts the $\mathcal{L}_\mu$ semantics to the space of monotone functions of type $2^{\mathcal{S}} \to 2^{\mathcal{S}}$, where $\mathcal{S}$ is the state space of an LTS. Hence, FLC formulas get interpreted by *predicate transformers*. This allows sequential composition to be interpreted naturally using function composition.

Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} : a \in \mathcal{A}\}, L)$ be an LTS, and

$$2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}} \;\; := \;\; \{f : 2^{\mathcal{S}} \to 2^{\mathcal{S}} \mid \forall S, T \subseteq \mathcal{S} : \text{ if } S \subseteq T \text{ then } f(S) \subseteq f(T)\}$$

be the set of all monotone predicate transformers over $\mathcal{T}$. This can be ordered partially by the well-known pointwise order

$$f \sqsubseteq g \quad \text{iff} \quad \forall T \subseteq \mathcal{S} : f(T) \subseteq g(T)$$

In fact, $(2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}}, \sqsubseteq)$ forms a complete lattice with top and bottom elements $\top = \lambda T.\mathcal{S}$, $\bot = \lambda T.\varnothing$, as well as meets and joins $\bigsqcap$, $\bigsqcup$. The following is easily verified. Let $f_i \in 2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}}$, $i \in I$ for some set of indices I. Then

$$\bigsqcap_{i \in I} f_i \;\; := \;\; \lambda T. \bigcap_{i \in I} f_i(T) \qquad \bigsqcup_{i \in I} f_i \;\; := \;\; \lambda T. \bigcup_{i \in I} f_i(T)$$

are monotone too, and form the infimum, resp. supremum of $\{f_i \mid i \in I\}$ in $2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}}$.

This function space will now act as the domain of interpretation for FLC formulas. A formula $\varphi(X)$ with a free variable X gives rise to a second-order function $F_\varphi : (2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}}) \to (2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}})$ which is monotone itself w.r.t. the partial order $\sqsubseteq$. According to the Knaster-Tarski Theorem, least and greatest fixed points of such second-order functions exist uniquely in $2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}}$ and can be used to give meaning to formulas with fixed-point quantifiers just as it is done in the modal μ-calculus and first-order functions.

Let $\rho : \mathcal{V} \to (2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}})$ be an environment interpreting (free) variables by monotone predicate transformers. As usual, we write $\rho[X \mapsto f]$ to denote the environment that maps X to f and agrees with ρ on all other arguments. The semantics of an FLC formula w.r.t. an underlying LTS and the environment ρ is defined inductively as follows.

$$\begin{aligned}
[\![q]\!]^{\mathcal{T}}_\rho &:= \lambda T.\{s \in \mathcal{S} \mid q \in L(s)\} \\
[\![\neg q]\!]^{\mathcal{T}}_\rho &:= \lambda T.\{s \in \mathcal{S} \mid q \notin L(s)\} \\
[\![X]\!]^{\mathcal{T}}_\rho &:= \rho(X) \\
[\![\tau]\!]^{\mathcal{T}}_\rho &:= \lambda T.T \\
[\![\langle a \rangle]\!]^{\mathcal{T}}_\rho &:= \lambda T.\{s \in \mathcal{S} \mid \exists t \in \mathcal{S}, \text{ s.t. } s \xrightarrow{a} t \text{ and } t \in T\}
\end{aligned}$$

$$\begin{aligned}
\llbracket [a] \rrbracket_\rho^\mathcal{T} &:= \lambda T.\{s \in \mathcal{S} \mid \forall t \in \mathcal{S} : \text{ if } s \xrightarrow{a} t \text{ then } t \in T\} \\
\llbracket \varphi \vee \psi \rrbracket_\rho^\mathcal{T} &:= \llbracket \varphi \rrbracket_\rho^\mathcal{T} \sqcup \llbracket \psi \rrbracket_\rho^\mathcal{T} \\
\llbracket \varphi \wedge \psi \rrbracket_\rho^\mathcal{T} &:= \llbracket \varphi \rrbracket_\rho^\mathcal{T} \sqcap \llbracket \psi \rrbracket_\rho^\mathcal{T} \\
\llbracket \varphi ; \psi \rrbracket_\rho^\mathcal{T} &:= \lambda T.\llbracket \varphi \rrbracket_\rho^\mathcal{T} \big(\llbracket \psi \rrbracket_\rho^\mathcal{T}(T)\big) \\
\llbracket \mu X.\varphi \rrbracket_\rho^\mathcal{T} &:= \bigsqcap \{f \in 2^\mathcal{S} \rightarrowtail 2^\mathcal{S} \mid \llbracket \varphi \rrbracket_{\rho[X \mapsto f]}^\mathcal{T} \sqsubseteq f\} \\
\llbracket \nu X.\varphi \rrbracket_\rho^\mathcal{T} &:= \bigsqcup \{f \in 2^\mathcal{S} \rightarrowtail 2^\mathcal{S} \mid f \sqsubseteq \llbracket \varphi \rrbracket_{\rho[X \mapsto f]}^\mathcal{T}\}
\end{aligned}$$

Thus, the operators of FLC are simply translated into related operators on the lattice structure of the function space $2^\mathcal{S} \rightarrowtail 2^\mathcal{S}$ with τ being the identity function as the neutral element of the sequential composition operator.

Since FLC is supposed to be a program logic, it is necessary to explain when a single state satisfies a (closed) formula of FLC. Note that in the case of the modal μ-calculus this is simply done using the element relation on the semantics of the formula. This is clearly not possible if the semantics is a function. The usual *models*-relation is therefore—by arbitrary choice—defined as follows. Let $\mathcal{T}$ be an LTS with state set $\mathcal{S}$ and $s \in \mathcal{S}$.

$$\mathcal{T}, s \models_\rho \varphi \quad \text{iff} \quad s \in \llbracket \varphi \rrbracket_\rho^\mathcal{T}(\mathcal{S})$$

This gives rise to two different equivalence relations in FLC: two formulas φ and ψ are *strongly equivalent* if they have the same semantics.

$$\varphi \equiv \psi \quad \text{iff} \quad \text{for all LTS } \mathcal{T} \text{ and all } \rho : \mathcal{V} \to (2^\mathcal{S} \rightarrowtail 2^\mathcal{S}) : \llbracket \varphi \rrbracket_\rho^\mathcal{T} = \llbracket \psi \rrbracket_\rho^\mathcal{T}$$

On the other hand, they are *weakly equivalent* if they are satisfied by the same set of states in any LTS.

$$\begin{aligned}
\varphi \approx \psi \quad \text{iff} \quad & \text{for all LTS } \mathcal{T} \text{ with state set } \mathcal{S} \text{ and all } \rho : \mathcal{V} \to (2^\mathcal{S} \rightarrowtail 2^\mathcal{S}) : \\
& \llbracket \varphi \rrbracket_\rho^\mathcal{T}(\mathcal{S}) = \llbracket \psi \rrbracket_\rho^\mathcal{T}(\mathcal{S})
\end{aligned}$$

Clearly, strong equivalence is at least as strong as weak equivalence: $\equiv \subseteq \approx$—two functions that agree on all arguments certainly agree on a particular one. Here we are mainly interested in weak equivalence because it formalizes "expressing the same property", and it is therefore the right notion for comparing FLC to other logics like MIC w.r.t. expressive power.

3.2.2 Properties Expressible in FLC

In the following we want to exemplify the use of FLC by formalizing a few non-regular properties.

Example 3.6. Consider the language $L = \{a^n b^n \mid n \geq 1\}$ again. It is generated by the context-free grammar with productions

$$S \quad \to \quad ab \mid aSb$$

FLC can express the property "there is a maximal path whose label forms a word in L".

$$(\mu X.\langle a\rangle;\langle b\rangle \vee \langle a\rangle;X;\langle b\rangle);\Box;\mathtt{ff}$$

Notice the apparent similarity to the grammar above.

To illustrate the semantics of FLC formulas, we give the first few stages X^i of the fixed-point iteration for the subformula $\mu X.\langle a\rangle;\langle b\rangle \vee \langle a\rangle;X;\langle b\rangle$.

$$\begin{array}{rcl} X^0 & := & \lambda T.\varnothing \\ X^1 & := & \lambda T.[\![\langle a\rangle]\!]([\![\langle b\rangle]\!](T)) \cup [\![\langle a\rangle]\!](X^0([\![\langle b\rangle]\!](T))) = \lambda T.[\![\langle a\rangle]\!]([\![\langle b\rangle]\!](T)) \\ X^2 & := & \lambda T.[\![\langle a\rangle]\!]([\![\langle b\rangle]\!](T)) \cup [\![\langle a\rangle]\!](X^1([\![\langle b\rangle]\!](T))) \\ & = & \lambda T.[\![\langle a\rangle]\!]([\![\langle b\rangle]\!](T)) \cup [\![\langle a\rangle]\!]([\![\langle a\rangle]\!]([\![\langle b\rangle]\!]([\![\langle b\rangle]\!](T)))) \\ & \vdots & \end{array}$$

In general, X^i is the function taking any set T to the set of states from which there is a path to a state in T under any of the words $\{ab, aabb, \ldots, a^ib^i\}$. Hence, $[\![(\mu X.\langle a\rangle;\langle b\rangle \vee \langle a\rangle;X;\langle b\rangle);\Box;\mathtt{ff}]\!]$ takes any set T to the set of nodes from which a node without successors can be reached by some a^nb^n-path.

Example 3.7. FLC can, like MIC, axiomatize the tree of height (at least) ω upto bisimulation. Again, we first say that there is no infinite path utilizing the $\mathcal{L}_\mu$ formula $\mu X.\Box X$.

$$\varphi_{\text{fin}} \;:=\; \mu X.\Box;X$$

Then we need to say that there are paths of unbounded length.

$$\varphi_{\text{unb}} \;:=\; (\nu X.\tau \wedge X;\Diamond);\mathtt{tt}$$

Note that, by unfolding, this is equivalent to $\bigwedge_{n\in\mathbb{N}} \Diamond^n;\mathtt{tt}$.

The following then expresses that a transition system is bisimilar to a tree of height exactly ω.

$$\varphi_{\text{fin}} \wedge \varphi_{\text{unb}} \wedge \Box\overline{\varphi}_{\text{unb}}$$

where the latter is supposed to express the complement of the respective unboundedness property. It can be obtained straight-forwardly as $\overline{\varphi}_{\text{unb}} := (\mu X.\tau \vee X;\Box);\mathtt{ff}$.

Recall that MIC can express bisimilarity to a finite word but possibly not to an infinite one. In FLC it does not seem to be possible to express either of these, and the problem is not to do with (in-)finiteness. For FLC the difficulty is to speak about two different paths. Note that an LTS $\mathcal{T}$ with starting state s is not bisimilar to any linear model if, and only if, there are two different actions a and b and a natural number $n \geq 0$, s.t.

$$\mathcal{T}, s \models \underbrace{\Diamond\ldots\Diamond}_{n\text{times}}\langle a\rangle\mathtt{tt} \wedge \underbrace{\Diamond\ldots\Diamond}_{n\text{times}}\langle b\rangle\mathtt{tt}$$

The model checking games for FLC presented below will give an idea of why the existence of such an n cannot be expressed in FLC. Essentially, non-bisimilarity could be decided using two stacks of formulas which would be used by one player to build the two conjuncts while the other player then decides which formula to prove in a given model. The FLC model checking games, however, only provide a single stack.

Example 3.8. Consider the related but simpler property of bisimilarity to a balanced tree. While non-bisimilarity to a word can be characterized in meta-logic using quantification over three different sorts—there are actions a and b and a natural number n, s.t. there are paths of length n ending in a, resp. b—describing that an LTS looks like a balanced tree only needs two: there are $n, m \in \mathbb{N}$ s.t. $n < m$ and two paths, one of which is maximal and has length n, the other has length m.

Take the FLC formula

$$\big(\mu X.\tau \vee X; (\Diamond; \mathtt{tt} \wedge \Box)\big); \Box; \mathtt{ff}$$

Now note that for any FLC formula φ we have $\mathtt{tt}; \varphi \equiv \mathtt{tt}$. Let $\Phi := \mu X.\tau \vee X; (\Diamond; \mathtt{tt} \wedge \Box)$. Unfolding the formula above and rewriting it using some basic equalities yields:

$$\begin{aligned} \Phi; \Box\mathtt{ff} &\equiv \big(\tau \vee \Phi; (\Diamond\mathtt{tt} \wedge \Box)\big); \Box\mathtt{ff} \\ &\equiv \Box\mathtt{ff} \vee \Phi; (\Diamond\mathtt{tt} \wedge \Box; \Box\mathtt{ff}) \\ &\equiv \Box\mathtt{ff} \vee \big(\tau \vee \Phi; (\Diamond\mathtt{tt} \wedge \Box)\big); (\Diamond\mathtt{tt} \wedge \Box\Box\mathtt{ff}) \\ &\equiv \Box\mathtt{ff} \vee (\Diamond\mathtt{tt} \wedge \Box\Box\mathtt{ff}) \vee \Phi; (\Diamond\mathtt{tt} \wedge \Box\Diamond\mathtt{tt} \wedge \Box\Box\Box\mathtt{ff}) \\ &\equiv \ldots \end{aligned}$$

The i-th unfolding of this formula asserts that there is a path of length i, all paths of length less than i can be extended by at least one state, but there is no path of length $i + 1$. Hence, the union over all these approximations defines exactly the class of all balanced trees of finite height.

There is a straight-forward translation $\Phi_{\langle\cdot\rangle}$: CFG $\to$ FLC which assigns to each context-free grammar G an FLC formula $\Phi_{\langle G\rangle}$ s.t.

$$\mathcal{T}, s \models \Phi_{\langle G\rangle} \quad \text{iff} \quad \text{there is a } t \in \mathcal{S} \text{ and a } w \in L(G) \text{ s.t. } s \xrightarrow{w} t$$

where $L(G)$ denotes, as usual, the context-free language generated by G. Φ_G simply is the uniform translation of G which replaces nonterminals in the grammars with μ-quantified FLC variables, concatenation with sequential composition, and alternatives between rules with disjunctions [13].

However, there is no translation $\Phi_{[\cdot]} : \mathrm{CFG} \to \mathrm{FLC}$ s.t. for all LTS $\mathcal{T}$ with state set $\mathcal{S}$ and all $s \in \mathcal{S}$ we have

$$\mathcal{T}, s \models \Phi_{[G]} \quad \text{iff} \quad \text{for all } t \in \mathcal{S} \text{ and all } w \in \mathcal{A}^*\text{: if } s \xrightarrow{w} t \text{ then } w \in L(G)$$

Such a translation would contradict the decidability of FLC's model checking problem by a simple reduction from the universality problem for context-free languages.

Example 3.9. Finally, we consider the property of not doing more *returns* than *calls*, i.e. we want to specify a tree (or transition system) in which all paths, including non-maximal ones, are labeled by a word from the language $L = \{w \in \{a,b\}^* \mid \forall v \preceq w : |v|_b \leq |v|_a\}$ where $\preceq$ denotes the prefix relation on words, and $|v|_a$ stands for the number of occurrences of the letter a in v. This language is context-free, and for example generated by the grammar G as given by

$$S \;\to\; aTS \mid \varepsilon \qquad T \;\to\; b \mid aTT \mid \varepsilon$$

The specification would be completed if there was a translation $\Phi_{[\cdot]}$ as mentioned above. Even though this cannot exist, the desired property can still be specified in FLC. Note that the language $L(G)$ of all non-underflowing buffer runs is *deterministic context-free*, and its complement is generated by the grammar G' defined as

$$S \;\to\; bU \mid aTbU \qquad U \;\to\; \varepsilon \mid aU \mid bU \qquad T \;\to\; b \mid aTT$$

This can then be transformed into the FLC formula $\Phi_{\langle G' \rangle}$ and consecutively be complemented to obtain

$$\varphi \;:=\; [b];\mathtt{ff} \wedge [a];\big(\nu T.[b] \wedge [a];T;T\big);[b];\mathtt{ff}$$

which expresses lack of buffer underflows on all runs.

3.3 Complexity and expressive power

First of all, it is not hard to see that both MIC and FLC are genuine extensions of the modal μ-calculus w.r.t. expressive power.

Proposition 3.10 (Dawar-Grädel-Kreutzer-Müller-Olm, [4, 16])**.** $\mathcal{L}_\mu \lneq \mathrm{MIC}$, $\mathcal{L}_\mu \lneq \mathrm{FLC}$.

It is obvious that any $\mathcal{L}_\mu$-formula is equivalent to a formula in MIC—simply replace μ-operators by **ifp**- and ν- by **dfp**-operators. Furthermore, $\mathcal{L}_\mu$ translates into FLC almost as easily: simply replace every $\langle a \rangle\varphi$ by $\langle a \rangle;\varphi$, and every $[a]\varphi$ by $[a];\varphi$. The strictness of both inclusions immediately follows from the previous examples showing how to express certain non-regular properties in these logics. Related to this is also the loss of the finite model property compared to $\mathcal{L}_\mu$, as already shown in Ex. 3.3 and 3.7.

Proposition 3.11 (Dawar-Grädel-Kreutzer-Müller-Olm, [4, 16])**.** Both MIC and FLC do not have the finite model property.

The tree model property, however, can be proved by embedding MIC and FLC into infinitary modal logic. This is particularly simple for MIC where it only requires fixed-point elimination.

For every FLC formula φ we obtain a formula φ' of infinitary modal logic such that $\varphi \approx \varphi'$ by eliminating fixed points first, then followed by the elimination of sequential composition and the formula τ. This is possible because $\varphi \approx \varphi; \mathtt{tt}$, and can easily be done by successively pushing sequential composition inwards from the right.

Proposition 3.12 (Dawar-Grädel-Kreutzer-Lange-Stirling, [4, 14])**.** Both MIC and FLC are invariant under bisimulation and, hence, have the tree-model property.

Not much is known about the expressive power of each of these logics relative to other formalisms like Predicate Logic, or—when restricted to word models—formal grammars and automata. For MIC, it is known that it is not the bisimulation-invariant fragment of monadic inflationary fixed-point logic, which would have been the natural candidate as $\mathcal{L}_\mu$ is the bisimulation-invariant fragment of monadic least fixed-point logic. As to grammars and automata, FLC is slightly easier to compare in this respect because of the similarity between formulas and context-free grammars. Also, the characterisation of least and greatest fixed points by the Knaster-Tarski Theorem gives a straight-forward embedding of FLC into Third-Order Logic. To gain a good intuition about the expressive power of temporal logics, however, it is often useful to consider word or well-founded models.

Proposition 3.13 (Dawar-Grädel-Kreutzer-Lange, [4, 10])**.** When interpreted over word models only,

i) there is a language that is not context-free but definable in MIC.

ii) FLC is equi-expressive to alternating context-free languages.[3]

iii) every language in $\mathrm{DT{\small IME}}(\mathcal{O}(n))$ is definable in MIC.

[3] These are generated by *alternating context-free grammars* which enrich ordinary context-free grammars by two types of non-terminals: *existential* and *universal* ones. While the generation of sentential forms and, thus, words for existential non-terminals is the usual one, universal non-terminals derive a sentential form only if *all* (rather than any) of their productions derive it. There are various ways of defining a precise semantics that captures this idea. Alternating Context-Free Grammars as defined by the second author [10] are in fact the same as Conjunctive Grammars by Okhotin [17]. There are also presumably non-equivalent models like the grammars by Moriya [8].

iv) every language definable in MIC or FLC is in DSPACE($\mathcal{O}(n)$), i.e. deterministic context-sensitive.

As usual, great expressive power also comes at a price. One can show that arithmetic is expressible in MIC on trees of height ω, i.e. the tree unraveling of the ordinal ω. For this, a natural number $n \in \mathbb{N}$ is identified with the set of nodes of height at most n. Then, arithmetic on the height of nodes can be shown to be definable in MIC. By doing so, one can translate any first-order sentence ψ over the arithmetic $\mathcal{N} := (\mathbb{N}, <, +, \cdot)$ into a MIC-formula ψ^* such that $\mathcal{N} \models \psi$ if, and only if, ψ^* is satisfiable. Here, ψ^* enforces its models to be bisimilar to a tree of height ω and encodes the arithmetical sentence ψ on such trees. This immediately implies undecidability.

Satisfiability in FLC is undecidable as well. This was first shown by Müller-Olm using a reduction from the simulation equivalence problem for context-free processes [16]. An embedding of Propositional Dynamic Logic of Non-Regular Programs, however, yields a quantitatively similar result as the one for MIC.

Proposition 3.14 (Dawar-Grädel-Kreutzer-Lange-Somla, [4, 13])**.** The satisfiability problem for both MIC and FLC is undecidable. They are not even in the arithmetical hierarchy.

Concerning the model checking complexity, it is easily seen that a naïve evaluation of MIC-formulas by iteratively computing the stages of the fixed-point inductions leads to an algorithm that correctly checks whether a given MIC-formula φ is true in a given transition system $\mathcal{T}$ in time $\mathcal{O}(|\mathcal{T}|^{|\varphi|})$ and space $\mathcal{O}(|\mathcal{T}| \cdot |\varphi|)$. It is therefore in P whenever the formula is fixed. It is, however, PSPACE-hard already on a fixed 1-state transition system if the formula is part of the input.

FLC differs from MIC w.r.t. model checking. First of all, fixed-point approximations can be exponentially long in the size of the transition system [12]. FLC can even express problems which are hard for deterministic exponential time, namely Walukiewicz's Pushdown Game problem [21].

An upper bound of deterministic exponential time is not immediately seen. Note that naïve fixed-point iteration in the function space $2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}}$ would lead to a doubly exponential procedure. But remember that model checking in FLC means that the value of a function on a particular argument, namely $\mathcal{S}$, needs to be computed rather than the entire function itself. This observation leads—with the aid of stair parity games, see below—to a singly exponential model checking algorithm.

Proposition 3.15 (Dawar-Grädel-Kreutzer-Axelsson-Lange-Somla, [3, 4, 12])**.** The combined complexity of the model checking problem for *i*) MIC is PSPACE-complete,

ii) FLC is EXPTIME-complete.

Regarding the data complexity, we have the following result.

iii) For every fixed formula the model checking complexity of MIC is in P.

iv) There are fixed FLC formulas for which the model checking problem is EXPTIME-hard.

Regarding the expression complexity, we have the following results.

v) Model checking MIC on a fixed transition system is PSPACE-complete.

PSPACE-hardness of the expression complexity is obtained by a reduction from QBF, the problem to decide if a given quantified boolean formula is satisfiable. It can easily be reduced to the model checking problem of MIC on a trivial transition system consisting of one state only.

The only lower bound for the expression complexity of FLC that is currently known is P-hardness trivially inherited from $\mathcal{L}_\mu$ [6].

An interesting question for non-regular logics is decidability of the model checking problem over infinite state systems. The known results there are negative.

Proposition 3.16 (Müller-Olm-Lange-Stirling, [16, 14])**.** The model checking problem for FLC over the class of normed deterministic BPA processes is undecidable.

The proof uses the fact that characteristic formulas for simulation (equivalence) of BPA processes can easily be constructed in FLC. It is currently not known whether or not MIC has a decidable model checking problem over the class of context-free processes.

Finally, we can use the model checking complexity results to prove an inexpressibility theorem, and partially separate MIC and FLC in expressive power.

Theorem 3.17. FLC $\not\leq$ MIC.

Proof. Take an FLC formula φ whose set of models is EXPTIME-hard according to Proposition 3.15 (*iv*). Suppose FLC $\leq$ MIC. Then there would be a MIC formula φ' with the same set of models. However, according to Proposition 3.15 (*iii*), this set would also have to be in P, and we would have P = EXPTIME which is not the case. Q.E.D.

It is not yet known whether every MIC-definable property is also FLC definable or whether the two logics are incomparable w.r.t. expressive power. We suspect that the latter is the case. The difficulty in establishing this as a theorem though is the lack of machinery for showing inexpressibility in FLC.

4 Non-regular games

There are many ways of extending ordinary parity games. One option, which we shall consider first, is to introduce the concept of stacks to the games. Formally, these games are played on configuration graphs of pushdown processes. In this approach we increase the modeling power of the game arenas while keeping the traditional way of playing games, i.e. the two players push a token along paths in the game arena and the priorities of this path determine the winner.

A different approach is to stick to standard parity game arenas but change the way the games are played. This approach is taken in the concept of backtracking games, where a play no longer is a path through the arena but defines a complex subgraph.

4.1 Stair parity games

A *pushdown alphabet* $\mathcal{A}$ is a tuple $(\mathcal{A}_\mathrm{c}, \mathcal{A}_\mathrm{r}, \mathcal{A}_\mathrm{i})$ consisting of three disjoint finite alphabets, a finite set $\mathcal{A}_\mathrm{c}$ of *calls*, a finite set $\mathcal{A}_\mathrm{r}$ of *returns* and a finite set $\mathcal{A}_\mathrm{i}$ of internal states.

Definition 4.1. Let $\mathcal{A} := (\mathcal{A}_\mathrm{c}, \mathcal{A}_\mathrm{r}, \mathcal{A}_\mathrm{i})$ be a pushdown alphabet. A *visibly pushdown system* (VPS) over $(\mathcal{A}_\mathrm{c}, \mathcal{A}_\mathrm{r}, \mathcal{A}_\mathrm{i})$ is a tuple $\mathcal{B} = (Q, \mathcal{A}, \Gamma, \delta)$ where Q is a finite set of states, and Γ is a finite stack alphabet. We simply write $\Gamma_\perp$ for $\Gamma \cup \{\perp\}$ assuming that Γ itself does not contain the special stack bottom symbol $\perp$. Finally, $\delta = \delta_\mathrm{c} \cup \delta_\mathrm{r} \cup \delta_\mathrm{i}$ is the transition relation with

$$\begin{aligned} \delta_\mathrm{c} &\subseteq Q \times \mathcal{A}_\mathrm{c} \times Q \times \Gamma \\ \delta_\mathrm{r} &\subseteq Q \times \mathcal{A}_\mathrm{r} \times \Gamma_\perp \times Q \\ \delta_\mathrm{i} &\subseteq Q \times \mathcal{A}_\mathrm{i} \times Q \end{aligned}$$

A transition (q, a, q', γ), where $a \in \mathcal{A}_\mathrm{c}$, means that if the system is in the control state q and reads an a, it can change its state to q' and push the symbol γ onto the stack. Similarly, upon a transition (q, a, q', γ), where $a \in \mathcal{A}_\mathrm{r}$, it reads γ from the top of the stack (and pops it unless $\gamma = \perp$) and changes its state from q to q'. Transitions reading $a \in \mathcal{A}_\mathrm{i}$ are *internal* transitions that do not change the stack.

We now turn to defining stair parity games, which are parity games played on the configuration graph of visibly pushdown systems with a slightly modified winning condition.

Definition 4.2. A *stair parity game* (SPG) over a VPS $\mathcal{B}$ is a tuple $\mathcal{G}_{\mathcal{B}} = (V, v_0, Q_\exists, Q_\forall, E, \Omega)$ such that

- $V := Q \times \Gamma^*\{\bot\}$ is the set of nodes in this game,
- $v_0 \in V$ is a designated starting node,
- Q is partitioned into $Q_\exists$ and $Q_\forall$,
- $E \subseteq V \times V$ consists of edges $\big((q, \delta), (q', \delta')\big)$ s.t.
  - there is a $(q, a, q', \gamma) \in \delta_{\mathrm{c}}$ and $\delta' = \gamma\delta$, or
  - there is a $(q, a, \gamma, q') \in \delta_{\mathrm{r}}$ and $\delta = \gamma\delta'$, or
  - there is a $(q, a, q') \in \delta_{\mathrm{i}}$ and $\delta' = \delta$.
- $\Omega : Q \to \mathbb{N}$ assigns to each node a priority.

For simplicity we assume that SPGs always are total, i.e. every node has an outgoing edge. A play in such a game is, as usual, an infinite sequence of nodes. It is played starting in v_0, and continued by a choice along an outgoing edge of that player who owns the last visited node. Unlike the case of parity games, the winner is not determined by the least or greatest priority occurring infinitely often in a play. Instead, one only considers those nodes that form stairs, i.e. nodes with a stack that persists for the entire remainder of the play.

Definition 4.3. Let $\mathcal{G}_{\mathcal{B}} = (V, v_0, Q_\exists, Q_\forall, E, \Omega)$ be a SPG over a VPS $\mathcal{B}$, and let $\pi = v_0, v_1, v_2, \ldots$ be an infinite play of this game s.t. $v_i = (q_i, \delta_{\mathrm{i}})$ for all $i \in \mathbb{N}$.

Define $\mathrm{Steps}(\pi) = \{i \in \mathbb{N} : \forall j \geq i \ |\delta_j| \geq |\delta_{\mathrm{i}}|\}$ where $|\delta|$ denotes the length of the stack δ. Note that $|\mathrm{Steps}(\pi)| = \infty$ whenever π is infinite.

Player $\exists$ wins the play π if, and only if, $\max\{c :$ there are infinitely many q_i with $i \in \mathrm{Steps}(\pi)$ and $\Omega(q_i) = c\}$ is even. Otherwise, Player $\forall$ is the winner of π.

The *stair parity game problem* is: given a SPG $(V, v_0, Q_\exists, Q_\forall, E, \Omega)$, decide whether or not Player $\exists$ has a winning strategy from node v_0 in this game. It can be shown that such games are determined, and that this problem is decidable. In fact, a reduction to an exponentially large ordinary parity game yields a moderate upper complexity bound.

Theorem 4.4 (Löding-Madhusudhan-Serre, [15])**.** The stair parity game problem can be decided in EXPTIME.

$$\frac{s \vdash (\varphi_0 \vee \varphi_1) \ ; \ \delta}{s \vdash \varphi_i \ ; \ \delta} \ \exists i \in \{0,1\} \qquad \frac{s \vdash (\varphi_0 \wedge \varphi_1) \ ; \ \delta}{s \vdash \varphi_i \ ; \ \delta} \ \forall i \in \{0,1\}$$

$$\frac{s \vdash (\sigma X.\varphi) \ ; \ \delta}{s \vdash X \ ; \ \delta} \qquad \frac{s \vdash X \ ; \ \delta}{s \vdash \varphi \ ; \ \delta} \ \text{if } \mathrm{fp}(X) = \sigma X.\varphi \qquad \frac{s \vdash (\varphi;\psi) \ ; \ \delta}{s \vdash \varphi \ ; \ (\psi;\delta)}$$

$$\frac{s \vdash \tau \ ; \ (\varphi;\delta)}{s \vdash \varphi \ ; \ \delta} \qquad \frac{s \vdash \langle a\rangle \ ; \ (\varphi;\delta)}{t \vdash \varphi \ ; \ \delta} \ \exists s \xrightarrow{a} t \qquad \frac{s \vdash [a] \ ; \ (\varphi;\delta)}{t \vdash \varphi \ ; \ \delta} \ \forall s \xrightarrow{a} t$$

FIGURE 1. The rules of the FLC model checking games.

4.2 A game-theoretic characterization of FLC

Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} : a \in \mathcal{A}\}, L)$ be an LTS, $s_0 \in \mathcal{S}$ and Φ be a closed FLC formula. The model checking game $\mathcal{G}_{\mathcal{T}}(s_0, \Phi)$ is played between Players $\exists$ and Player $\forall$ in order to establish whether or not $\mathcal{T}, s_0 \models \Phi$ holds. The set of configurations is $\mathcal{C} := \mathcal{S} \times \mathrm{Sub}(\varphi) \times \mathrm{Sub}(\varphi)^*$. We usually write configurations in the form $s \vdash \varphi \ ; \ \delta$ where $\delta = \psi_1; \ldots; \psi_k$ acts as a stack of FLC formulas with its top on the left. The formula φ will in this case also be called the *principal formula* of this configuration.

The intuitive meaning of such a configuration is the following. Player $\exists$ wants to show that $s \in [\![\varphi;\delta]\!]^{\mathcal{T}}_{\rho}(\mathcal{S})$ holds under a ρ which interprets the free variables in $\varphi;\delta$ by suitable approximants.

The initial configuration is $s_0 \vdash \Phi \ ; \ \mathtt{tt}$— remember that $\Phi \approx \Phi; \mathtt{tt}$. The rules of the model checking game are shown in Figure 1.

The idea underlying these games is to defer the examination of ψ in a formula $\varphi;\psi$ and to first consider whether or not φ determines the winner already. This is in contrast to the Boolean binary constructs $\wedge$ and $\vee$ in which both operands have equal importance. However, this is not the case for the sequential composition operator. A natural choice would be to let Player $\exists$ provide a witness for the chop (a set of states for example) and then to let Player $\forall$ respond by choosing either of the composed subformulas. This is not sound though, as the following example shows.

Example 4.5. Let $\Phi = \nu X.\mu Y.X;Y$. The exact meaning of this rather simple formula is not too difficult to guess. It can also be computed using fixed-point iteration on an imaginary model with state set $\mathcal{S}$. Remember that $\top$ and $\bot$ are the top- and bottom-elements in the function lattice $2^{\mathcal{S}} \rightarrowtail 2^{\mathcal{S}}$. At the beginning, the ν-quantified X gets mapped to $\top$, and in the inner fixed-point iteration, Y gets mapped to $\bot$. We use the symbol $\circ$ to denote function composition semantically as opposed to the syntactical operator ";".

$$\begin{array}{lllllll}
X^0 & := \top & & & & & \\
 & & Y^{00} & := & \bot & & \\
 & & Y^{01} & := & X^0 \circ Y^{00} & = & \top \\
 & & Y^{02} & := & X^0 \circ Y^{01} & = & \top \;=\; Y^{01} \\
X^1 & := Y^{02} \;=\; \top & & & & & \\
 & & Y^{10} & := & \bot & & \\
 & & Y^{11} & := & X^1 \circ Y^{10} & = & \top \\
 & & Y^{12} & := & X^1 \circ Y^{11} & = & \top \;=\; Y^{11} \\
X^2 & := Y^{12} \;=\; \top \;=\; X^1 & & & & &
\end{array}$$

Hence, $\Phi \equiv \mathtt{tt}$. Now suppose that Player $\forall$ was given the opportunity to choose a subformula of a sequential composition. In this case he could enforce a play which traverses solely through the μ-quantified variable Y only. Hence, for such games we should have to abolish the correspondence between infinite unfoldings of fixed points and wins for either of the players known from parity games.

This example only explains why the games do a left-depth-first traversal through formulas w.r.t. sequential compositions. This does not mean though that parity winning conditions on these games provide a correct characterization of FLC's model checking problem. The next example shows that parity winning conditions indeed do not suffice.

Example 4.6. Consider the two-state LTS $\mathcal{T} = (\{s,t\}, \{\xrightarrow{a}, \xrightarrow{b}\}, L)$ with $L(s) = L(t) = \varnothing$, and $s \xrightarrow{a} t$, $t \xrightarrow{b} t$. We shall evaluate the formula $\Phi := \mu Y.\langle b\rangle \vee \langle a\rangle; \nu X.Y; X$ on $\mathcal{T}$. Its precise semantics can be computed using fixed-point iteration again.

$$\begin{array}{lllllll}
Y^0 & := \bot & & & & & \\
 & & X^{00} & := & \top & & \\
 & & X^{01} & := & Y^0 \circ X^{00} & = & \bot \circ \top \;=\; \bot \\
 & & X^{02} & := & Y^0 \circ X^{01} & = & \bot \circ \bot \;=\; \bot \;=\; X^{01} \\
Y^1 & := [\![\langle b\rangle]\!]^{\mathcal{T}} \sqcup ([\![\langle a\rangle]\!]^{\mathcal{T}} \circ X^{02}) \;=\; \lambda T.[\![\langle b\rangle]\!]^{\mathcal{T}}(T) \cup [\![\langle a\rangle]\!]^{\mathcal{T}}(\varnothing) & & & & & \\
 & = [\![\langle b\rangle]\!]^{\mathcal{T}}(T) & & & & & \\
 & & X^{10} & := & \top & & \\
 & & X^{11} & := & Y^1 \circ X^{10} & = & [\![\langle b\rangle]\!]^{\mathcal{T}} \circ \top \;=\; \lambda T.[\![\langle b\rangle]\!]^{\mathcal{T}}(\{s,t\})
\end{array}$$

$$
\begin{aligned}
&= \lambda T.\{t\} \\
X^{12} &:= Y^1 \circ X^{11} \;=\; [\![\langle b\rangle]\!]^{\mathcal{T}} \circ \lambda T.\{t\} \;=\; \lambda T.[\![\langle b\rangle]\!]^{\mathcal{T}}(\{t\}) \\
&= \lambda T.\{t\} \;=\; X^{11} \\
Y^2 &:= [\![\langle b\rangle]\!]^{\mathcal{T}} \sqcup ([\![\langle a\rangle]\!]^{\mathcal{T}} \circ X^{12}) \;=\; \lambda T.[\![\langle b\rangle]\!]^{\mathcal{T}}(T) \cup [\![\langle a\rangle]\!]^{\mathcal{T}}(\{t\}) \\
&= \lambda T.[\![\langle b\rangle]\!]^{\mathcal{T}}(T) \cup \{s\}
\end{aligned}
$$

Even though the fixed point is not found yet, we can deduce by monotonicity that $\mathcal{T}, s \models \Phi$ holds. Note that we shall have $\lambda T.[\![\langle b\rangle]\!]^{\mathcal{T}}(T) \cup \{s\} \sqsubseteq [\![\Phi]\!]^{\mathcal{T}}$ and therefore $s \in [\![\Phi]\!]^{\mathcal{T}}(\{s,t\})$.

On the other hand, consider the following infinite play of $\mathcal{G}_{\mathcal{T}}(s,\Phi)$ which Player $\exists$ can enforce. It is also not hard to see that all other plays he can enforce should lead to a win for Player $\forall$ immediately because they end in a configuration in which $\exists$ gets stuck with no transitions to chose.

$$
\begin{array}{ccrcl}
s & \vdash & \mu Y.\langle b\rangle \vee \langle a\rangle; \nu X.Y; X & ; & \mathtt{tt} \\
s & \vdash & Y & ; & \mathtt{tt} \\
s & \vdash & \langle b\rangle \vee \langle a\rangle; \nu X.Y; X & ; & \mathtt{tt} \\
s & \vdash & \langle a\rangle; \nu X.Y; X & ; & \mathtt{tt} \\
s & \vdash & \langle a\rangle & ; & (\nu X.Y; X); \mathtt{tt} \\
t & \vdash & \nu X.Y; X & ; & \mathtt{tt} \\
t & \vdash & X & ; & \mathtt{tt} \\
t & \vdash & Y; X & ; & \mathtt{tt} \\
t & \vdash & Y & ; & X; \mathtt{tt} \\
t & \vdash & \langle b\rangle \vee \langle a\rangle; \nu X.Y; X & ; & X; \mathtt{tt} \\
t & \vdash & \langle b\rangle & ; & X; \mathtt{tt} \\
t & \vdash & X & ; & \mathtt{tt} \\
 & & & \vdots &
\end{array}
$$

This play reaches a loop and can therefore be played ad infinitum. Note that both variables X and Y occur as principle formulas in configurations on this loop. Hence, if these games are equipped with an ordinary parity condition on principle formulas then the μ-quantified variable Y determines—as the outer one of the two— Player $\forall$ to be the winner of this play. But then he would have a winning strategy, and the games would not be correct.

The crucial difference between the occurrences of Y and X is that each Y does not stem from the unfolding of the Y above but from the unfolding of the inner X. Such a phenomenon does not occur in the parity model checking games for the modal μ-calculus.

The question that arises is how to recognize those variables that truly regenerate themselves when a simple comparison according to the outer-relation is not possible. The answer is provided by the stacks in those configurations that have variables as principle formulas. Note that between each two occurrences of X the stack does not decrease, but between two occurrences of Y it does. This shows that Y got "fulfilled" and the play continued with a formula that was on the stack when Y was principle—intuitively the left-depth-first search has terminated on this branch and follows a branch to the right of Y. This takes us back to the notion of $\text{Steps}(\pi)$ for a play π in a stair parity game.

Take the play π above. Then $\text{Steps}(\pi)$ consists of all positions whose stack contents persist for the rest of the game. Here they are $\{0, 1, 2, 3, 5, 6, 7, 11, \ldots\}$.

Definition 4.7. If $\pi = C_0, C_1, \ldots$ is an infinite play of $\mathcal{G}_\mathcal{T}(s_0, \Phi)$ and $C_i = s_i \vdash \varphi_i \ ; \ \delta_\text{i}$ then $\text{Steps}(\pi) = \{i \in \mathbb{N} : \forall j \geq i \ |\delta_j| \geq |\delta_\text{i}|\}$. Furthermore, let $\pi|_\text{st}$ denote the restriction of π to $\text{Steps}(\pi)$, i.e.

$$\pi|_\text{st} := C_{i_0}, C_{i_1}, C_{i_2}, \ldots \quad \text{iff} \quad \text{Steps}(\pi) = \{i_0, i_1, i_2, \ldots\}$$

with $i_j < i_{j'}$ iff $j < j'$.

This allows us to define the winning conditions of the FLC games in a way that correctly characterizes its model checking problem.

Definition 4.8. Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} : a \in \mathcal{A}\}, L)$ be an LTS, $s_0 \in \mathcal{S}$, Φ a closed FLC formula and $\pi = C_0, C_1, \ldots$ be a play of $\mathcal{G}_\mathcal{T}(s_0, \Phi)$ with $C_i = s_i \vdash \varphi_i \ ; \ \delta_\text{i}$ for all $i \in \mathbb{N}$. Player $\exists$ wins π if, and only if,

1. π is finite and ends in some C_n with
   a) $C_n = s_n \vdash q \ ; \ \delta$ and $q \in L(s)$,
   b) $C_n = s_n \vdash \neg q \ ; \ \delta$ and $q \notin L(s)$,
   c) $C_n = s_n \vdash [a] \ ; \ \delta$ and there is no $t \in \mathcal{S}$ s.t. $s \xrightarrow{a} t$;
2. π is infinite and the outermost variable occurring infinitely often as a principle formula in $\pi|_\text{st}$ is of type ν.

Player $\forall$ wins π if, and only if,

1. π is finite and ends in some C_n with
   a) $C_n = s_n \vdash q \ ; \ \delta$ and $q \notin L(s)$,
   b) $C_n = s_n \vdash \neg q \ ; \ \delta$ and $q \in L(s)$,
   c) $C_n = s_n \vdash \langle a \rangle \ ; \ \delta$ and there is no $t \in \mathcal{S}$ s.t. $s \xrightarrow{a} t$;
2. π is infinite and the outermost variable occurring infinitely often as a principle formula in $\pi|_\text{st}$ is of type μ.

It is then possible to show that each play has a unique winner, that the games are determined, etc.

Theorem 4.9 (Lange, [11]). Player $\exists$ has a winning strategy for the game $\mathcal{G}_{\mathcal{T}}(s, \Phi)$ if, and only if, $\mathcal{T}, s \models \Phi$.

As a consequence we obtain an upper bound on the complexity of FLC's model checking problem.

Corollary 4.10. The model checking problem for FLC can be decided in EXPTIME.

Proof. Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} : a \in \mathcal{A}\}, L)$ be an LTS and $\varphi_0 \in$ FLC. They induce a VPS $\mathcal{B}_{\mathcal{T},\varphi_0} = (Q, \mathcal{A}', \Gamma, \delta)$ with

- $Q = \mathcal{S} \times \mathrm{Sub}(\varphi_0)$,
- $\mathcal{A}'_{\mathrm{c}} = \{\mathsf{chop}\}$, $\mathcal{A}'_{\mathrm{r}} = \{\mathsf{tau}, \mathsf{mod}\}$, $\mathcal{A}'_{\mathrm{i}} = \{\mathsf{disj}, \mathsf{conj}, \mathsf{unf}\}$,
- $\Gamma = \mathrm{Sub}(\varphi_0)$,
- δ simply translates the rules of Figure 1 into a transition relation
  - $\delta_{\mathrm{c}} := \{((s, \varphi;\psi), \mathsf{chop}, (s, \varphi), \psi) : s \in \mathcal{S}, \varphi;\psi \in \mathrm{Sub}(\varphi_0)\}$;
  - $\delta_{\mathrm{r}} := \{((s, \tau), \mathsf{tau}, \varphi, (s, \varphi)) : s \in \mathcal{S}, \varphi \in \mathrm{Sub}(\varphi_0)\} \cup \{((s, \psi), \mathsf{mod}, \varphi, (t, \varphi)) : \psi \in \{\langle a\rangle, [a]\}, s \xrightarrow{a} t, \varphi \in \mathrm{Sub}(\varphi_0)\}$;
  - $\delta_{\mathrm{i}} := \{((s, \varphi_1 \vee \varphi_2), \mathsf{disj}, (s, \varphi_j)\big) : s \in \mathcal{S}, \varphi_1 \vee \varphi_2 \in \mathrm{Sub}(\varphi_0), j \in \{1, 2\}\} \cup \{((s, \varphi_1 \wedge \varphi_2), \mathsf{conj}, (s, \varphi_j)) : s \in \mathcal{S}, \varphi_1 \wedge \varphi_2 \in \mathrm{Sub}(\varphi_0), j \in \{1, 2\}\} \cup \{((s, \sigma X.\varphi), \mathsf{unf}, (s, X)) : s \in \mathcal{S}, \sigma X.\varphi \in \mathrm{Sub}(\varphi_0)\} \cup \{((s, X), \mathsf{unf}, (s, \varphi)) : s \in \mathcal{S}, X \in \mathrm{Sub}(\varphi_0), \mathrm{fp}X = \sigma X.\varphi\}$

A stair parity game is then obtained by simply making states of the form $(s, \varphi_0 \vee \varphi_1)$ choices of Player $\exists$ etc., and by assigning priorities to nodes $\big((s, \varphi), \delta\big)$ only depending on the principal formula φ s.t. all formulas other than variables have priority 0, μ-bound, resp. ν-bound variables have odd, resp. even priorities, and outer variables have greater priorities than inner ones. Correctness of this translation is given by the fact that the winning conditions of the FLC model checking games can easily be transferred into stair parity conditions by artificially prolonging finite plays ad infinitum. The complexity bound then follows from Theorem 4.4. Q.E.D.

These games do not only provide a local model checking algorithm for FLC. They can also be used to show that the fixed-point alternation hierarchy in FLC is strict [11]. The proof proceeds along the same lines as Arnold's proof for the alternation hierarchy in the modal μ-calculus [2] by constructing hard formulas (that define the winning positions for Player $\exists$ in such a game) and by using Banach's fixed-point theorem.

4.3 Model-checking games for the Modal Iteration Calculus

Stair Parity Games provide an elegant framework of model checking games for logics such as CaRet and FLC. We give further evidence for the significance of this concept in relation to fixed-point logics beyond the modal μ-calculus by showing that model checking games for MIC can also be phrased in this context. However, the games we present here only work for finite transition systems. The reason for this will become clear later in the section.

To simplify notation, we shall only explain the games for MIC-formulas without simultaneous inductions. Using similar ideas one can extend the games to cover simultaneous fixed points also.

Suppose first that we are given a transition system $\mathcal{T}$ and a formula $\varphi := \mathbf{ifp}X.\psi$, where $\psi \in$ ML is a modal logic formula in negation normal form. If Player $\exists$ wants to show that φ holds true at a node s in $\mathcal{T}$, he has to prove that there is a stage $n \in \mathbb{N}$ so that $s \in X^n$. Here, choosing n out of the natural numbers is enough as the fixed point in a finite transition system is always reached after a finite number of steps. In other words, he chooses an $n \in \mathbb{N}$ and then has to show that the n-fold unraveling[4] ψ^n of ψ holds true at s.

This idea is modeled in a stair parity game as follows. To choose the stage $n \in \mathbb{N}$, we give Player $\exists$ the option to push as many (finitely many) symbols X onto the stack as he wishes. This done, the two players continue by playing the standard modal logic game on the ML-formula ψ, with the modification that each time the game reaches the proposition X, one symbol X is popped from the stack and the game continues at ψ again. If the stack is empty, then Player $\exists$ has lost as he has failed to show that the starting state s satisfies ψ^n. However, there is one problem we need to solve. As φ is a MIC formula, the fixed-point variable X may be used negatively, i.e. the play on ψ may reach a literal $\neg X$. In this case, we again pop one symbol X from the stack, but then the game proceeds to the negation $\neg\psi$. To keep track of whether we are currently playing in ψ or the negation thereof, we rename the fixed-point variable X in $\neg\psi$ to X^c. If the play reaches X^c and there are no more symbols X left on the stack, then Player $\exists$ wins. Otherwise, one symbol is popped and the play continues at $\neg\psi$ again. If, however, a literal $\neg X^c$ is reached, then one symbol X is popped and the game proceeds back to the original formula ψ.

It should be clear now that Player $\exists$ can win this game on a formula $\varphi := \mathbf{ifp}X.\psi$ and a transition system $\mathcal{T}$ with initial state s if, and only if, $s \in [\![\varphi]\!]^{\mathcal{T}}$.

To extend this idea to formulas containing nested fixed points, we have

[4] The n-fold unravelling ψ^n of ψ is defined as follows: $\psi^0 := \mathtt{ff}$ and ψ^{n+1} is obtained from ψ by replacing each occurrence of X by ψ^n. It is easily seen that $s \in [\![\psi^n]\!]^{\mathcal{T}}$ if, and only if, the state s occurs in stage X^n of the induction on ψ in $\mathcal{T}$.

to modify the game slightly. Suppose a variable Y is bound by a fixed-point operator $\mathbf{dfp}\, Y.\psi$ inside the formula $\mathbf{ifp}\, X.\vartheta$ which binds an outer fixed-point variable X. When the game reaches $\mathbf{ifp}\, X.\vartheta$, Player $\exists$ pushes as many symbols X onto the stack as he likes. The game continues inside ϑ and reaches the formula $\mathbf{dfp}\, Y.\psi$ at which Player $\forall$ can push symbols Y onto the stack. Now, when the game reaches an atom X, then before we can regenerate the formula ϑ and pop one symbol X from the stack, we have to pop all Ys first. Other than that, the rules of the game are as described above.

To present this idea in more detail, let us first fix some notation. Let $\varphi \in \mathrm{MIC}$ be a formula in negation normal form and let $X_1, \dots, X_k$ be the fixed-point variables occurring in it. W.l.o.g. we assume that no fixed-point variable is bound twice in φ. Hence, with each X_i we can associate the unique formula ϑ_i such that X_i is bound in φ by $\mathbf{fp}\ X_i.\vartheta_i$, where $\mathbf{fp}$ is either $\mathbf{ifp}$ or $\mathbf{dfp}$. We also assume that the X_i are numbered such that if $i < j$ then ϑ_i is not a subformula of ϑ_j.

Let φ' be the formula obtained from $\neg\varphi$ by first renaming every fixed-point variable X_i in φ to X_i^c and then bringing the formula into negation normal form. Let $\Phi := \mathrm{Sub}(\varphi) \cup \mathrm{Sub}(\varphi')$.

Let $\mathcal{T} := (\mathcal{S}, \{\xrightarrow{a} : a \in \mathcal{A}\}, L)$ be a finite transition system. The formula φ and the system $\mathcal{T}$ induce a visibly pushdown system $\mathcal{B}_{\mathcal{T},\varphi} := (Q, \mathcal{A}', \Gamma, \delta)$ as follows. The stack alphabet is $\Gamma := \{X_1, \dots, X_k\}$.

For each variable X_i or X_i^c we use a gadget $\mathsf{clear}(X_i)$ that pops all symbols X_j from the stack with $j > i$ until the top of the stack contains a symbol X_j with $j \leq i$. As the gadget is deterministic, we can arbitrarily assign the nodes in it to either player. To simplify the presentation, we shall treat these gadgets as black boxes, i.e. as single nodes in the game graph.

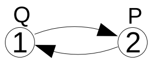
Now, Q contains all pairs $\mathcal{S} \times \Phi$ and the nodes of the gadgets $\mathsf{clear}(X_i)$ and $\mathsf{clear}(X_i^c)$ for $1 \leq i \leq k$. (Recall that $\Phi := \mathrm{Sub}(\varphi) \cup \mathrm{Sub}(\varphi')$.) We let $\mathcal{A}' := \mathcal{A}_\mathrm{c} \cup \mathcal{A}_\mathrm{r} \cup \mathcal{A}_\mathrm{i}$ where $\mathcal{A}_\mathrm{c} := \{\mathsf{push}\}$, $\mathcal{A}_\mathrm{r} := \{\mathsf{pop}\}$, and $\mathcal{A}_\mathrm{i} := \{\mathsf{int}\}$.

$$
\begin{aligned}
\delta_\mathrm{c} &:= \big\{\big((s, \mathbf{ifp}X.\vartheta), \mathsf{push}, (s, \mathbf{ifp}X.\vartheta), X\big) : s \in \mathcal{S}, \mathbf{ifp}X.\vartheta \in \Phi\big\} \\
&\cup \big\{\big((s, \mathbf{dfp}X.\vartheta), \mathsf{push}, (s, \mathbf{ifp}X.\vartheta), X\big) : s \in \mathcal{S}, \mathbf{dfp}X.\vartheta \in \Phi\big\} \\
\delta_\mathrm{r} &:= \big\{\big((s, \mathsf{clear}(X_i)), \mathsf{pop}, X_i, (s, \vartheta_i)\big) : 1 \leq i \leq k, s \in \mathcal{S}\big\} \\
&\cup \big\{\big((s, \mathsf{clear}(X_i^c)), \mathsf{pop}, X_i, (s, \vartheta_i^c)\big) : 1 \leq i \leq k, s \in \mathcal{S}\big\} \\
\delta_\mathrm{i} &:= \big\{\big((s, \mathbf{ifp}X.\vartheta), \mathsf{int}, (s, \vartheta)\big) : s \in \mathcal{S}, \mathbf{ifp}X.\vartheta \in \Phi\big\} \\
&\cup \big\{\big((s, \mathbf{dfp}X.\vartheta), \mathsf{int}, (s, \vartheta)\big) : s \in \mathcal{S}, \mathbf{dfp}X.\vartheta \in \Phi\big\} \\
&\cup \big\{\big((s, \varphi_1 \vee \varphi_2), \mathsf{int}, (s, \varphi_i)\big) : s \in \mathcal{S}, \varphi_1 \vee \varphi_2 \in \Phi, i \in \{1,2\}\big\} \\
&\cup \big\{\big((s, \varphi_1 \wedge \varphi_2), \mathsf{int}, (s, \varphi_i)\big) : s \in \mathcal{S}, \varphi_1 \vee \varphi_2 \in \Phi, i \in \{1,2\}\big\} \\
&\cup \big\{\big((s, \langle a\rangle\psi), \mathsf{int}, (t, \psi)\big) : s \in \mathcal{S}, t \in \mathcal{S},\ s \xrightarrow{a} t, \langle a\rangle\psi \in \Phi\big\}
\end{aligned}
$$

$$\cup\ \big\{\big((s,[a]\psi),\mathsf{int},(t,\psi)\big) : s\in\mathcal{S}, t\in\mathcal{S},\ s\xrightarrow{a}t, [a]\psi\in\Phi\big\}$$
$$\cup\ \big\{\big((s,X_i),\mathsf{int},\mathsf{clear}(X_i)\big) : s\in\mathcal{S}, 1\le i\le k\big\}$$
$$\cup\ \big\{\big((s,X_i^c),\mathsf{int},\mathsf{clear}(X_i^c)\big) : s\in\mathcal{S}, 1\le i\le k\big\}$$
$$\cup\ \big\{\big((s,\neg X_i),\mathsf{int},\mathsf{clear}(X_i^c)\big) : s\in\mathcal{S}, 1\le i\le k\big\}$$
$$\cup\ \big\{\big((s,\neg X_i^c),\mathsf{int},\mathsf{clear}(X_i)\big) : s\in\mathcal{S}, 1\le i\le k\big\}$$

To turn $\mathcal{B}_{\mathcal{T},\varphi}$ into a visibly pushdown game, we need to assign priorities and the nodes where each of the players moves. Player $\exists$ moves at disjunctions, nodes $(s,\langle a\rangle\psi)$, $(s,\mathbf{ifp}X_i.\vartheta_i)$, (s,X_i), $(s,\neg X_i^c)$, $(s,\mathsf{clear}(X_i))$, and nodes (s,q), $q\in\mathcal{P}$, if $q\notin L(s)$. At all other nodes Player $\forall$ moves. Finally, nodes $(s,\mathbf{ifp}X_i.\vartheta_i)$ are assigned the priority 1 and nodes $(s,\mathbf{dfp}X_i.\vartheta_i)$ are assigned the priority 0. Note that the priority assignment only needs to ensure that no player can loop forever on a fixed-point formula $\mathbf{fp}X_i.\vartheta_i$ pushing infinitely many variables onto the stack. As there are no infinite plays unless one of the players keeps pushing symbols onto the stack forever, the priorities do not influence the winner of "interesting" plays.

Example 4.11. We illustrate the construction by an example. Let $\mathcal{T}$ be the system



and let $\varphi := \mathbf{ifp}\ X.\big(p\vee\mathbf{ifp}\ Y.(q\wedge\Diamond X)\big)$. The corresponding game graph is depicted in Figure 2, where $\varphi_2 := (p\vee\mathbf{ifp}\ Y.(q\wedge\Diamond X))$, $\varphi_3 := \mathbf{ifp}\ Y.(q\wedge\Diamond X)$, and $\varphi_4 := q\wedge\Diamond X$ are the non-trivial sub-formulas of φ. To simplify the presentation, we have dropped the labels int and annotated the push and pop labels by the variable being pushed onto the stack. Note that there are no $\mathsf{pop}\,Y$ or $\mathsf{clear}(Y)$ labels, as the variable Y does not occur as an atom in the formula.

Clearly, Player $\exists$ wins the game from position $(1,\varphi)$ by first using the push transition to push one variable X onto the stack and then continue to $(1,\varphi_2)$. In this way, the play will either terminate in $(1,q)$ or continue along the node $(2,X)$ to $(2,\mathsf{clear}(X))$ and then along the pop-edge, where the symbol X will be popped from the stack, to $(2,\varphi)$, and finally to $(2,p)$. In both cases Player $\forall$ loses. Note, however, that Player $\exists$ cannot win without initially pushing X onto the stack, as the play will then terminate at the node $(2,\mathsf{clear}(X))$ with the pop-edge no longer available. This corresponds to the state 1 being in the stage X^2 but not in the stage X^1. (By pushing X once onto the stack, Player $\exists$ enforces the play to go through the inner formula φ_2 twice, corresponding to the stage X^2.)

The following theorem can be proved along the same lines as the corresponding proof for backtracking games in [5].

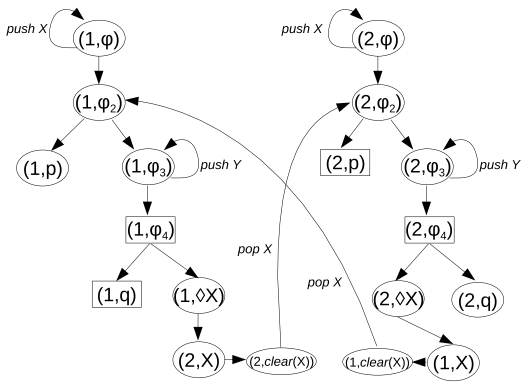


FIGURE 2. Visibly Pushdown System for Ex. 4.11.

Theorem 4.12. For every $\varphi \in$ MIC (without simultaneous fixed points) and finite transition system $\mathcal{T}$, Player $\exists$ has a winning strategy from a node (s, φ) in the visibly pushdown game $\mathcal{B}_{\mathcal{T},\varphi}$ if, and only if, $\mathcal{T}, s \models \varphi$.

Clearly, winning regions of general visibly pushdown games are not definable in MIC (as computing them is EXPTIME-hard), presumably not even if we restrict attention to a fixed number of priorities. However, the pushdown games constructed above have a rather simple structure. They only have two priorities but, even more important, the push transitions are local, i.e. for each fixed-point operator in φ there is one node which has a self-loop pushing a variable onto the stack. Therefore, there is hope that we can identify a suitable fragment of visibly pushdown games containing the games arising from MIC-formulas and whose winning regions can be defined in MIC, i.e. the winner of games from this fragment are MIC-definable in the same way as the winner of parity games can be defined in $\mathcal{L}_\mu$.

We illustrate this by considering games arising from formulas $\mathbf{ifp}X.\psi$ where $\psi \in$ ML. Such a game has a starting node from which a path labeled by push emerges. To each node v on this path with distance n to the root there is a copy of the game ψ^n attached to it, where ψ^n is the n-fold unraveling of ψ w.r.t. X. Hence, to define the winner of such games we only need a formula that checks whether on this push-path emerging from the root there is a node such that Player $\exists$ wins the modal logic game attached to it. The latter is clearly MIC-definable, so the whole formula is easily seen to be definable in MIC.

It is conceivable that a similar construction using nested fixed points works for games arising from MIC-formulas with nested fixed points. However, a formal proof of this results is beyond the scope of this survey and is left for future work.

4.4 Backtracking games

We now turn to a different type of non-regular games, the so-called *backtracking games.* The motivation for backtracking games comes from properties such as *the tree is balanced* as shown to be expressible in MIC and FLC. To verify such properties in a game-theoretical setting, the game needs to be able to inspect all subtrees rooted at successors of the root of a finite tree. However, *linear* games such as parity games that construct an infinite path through a game arena can only visit one subtree, unless we introduce back-edges towards the root. This motivates a game model where a play is no longer an infinite path but a more complex subgraph. Backtracking games were originally introduced as model checking games for inflationary fixed-point logics such as MIC and the general inflationary fixed-point logic IFP (see [5]). We first give an informal description.

Backtracking games are essentially parity games with the addition that, under certain conditions, players can jump back to an earlier position in the play. This kind of move is called backtracking.

A backtracking move from position v to an earlier position u is only possible if v belongs to a given set B of backtrack positions, if u and v have the same priority $\Omega(v)$ and if no position of higher priority has occurred between u and v. With such a move, the player who backtracks not only resets the play back to u, he also commits himself to a backtracking distance d, which is the number of positions of priority $\Omega(v)$ that have been seen between u and v. After this move, the play ends when d further positions of priority $\Omega(v)$ have been seen, unless this priority is "released" by a higher priority.

For finite plays we have the winning condition that a player wins if his opponent cannot move. For infinite plays, the winner is determined according to the parity condition, i.e. Player $\exists$ wins a play π if the highest priority seen infinitely often in π is even, otherwise Player $\forall$ wins.

Definition 4.13. The arena $\mathcal{G} := (V, E, V_\exists, V_\forall, B, \Omega)$ of a backtracking game is a directed graph (V, E), with a partition $V = V_\exists \cup V_\forall$ into positions of Player $\exists$ and positions of Player $\forall$, a subset $B \subseteq V$ of backtrack positions and a map $\Omega : V \to \{0, \dots, k-1\}$ that assigns to each node a priority.

Proposition 4.14 (Dawar-Grädel-Kreutzer, [5])**.** The following basic properties about backtracking games are known.

1. Backtracking games are determined, i.e. in every backtracking game, one of the two players has a winning strategy

2. Backtracking games in general do not admit finite memory strategies.

3. Deciding the winner of a backtracking game even with only two priorities is hard for NP and co-NP.

4. Deciding the winner of a backtracking game in general is PSPACE-hard.

However, no upper bound for the complexity of backtracking games is known.

Backtracking games can be used as model checking games for inflationary fixed-point logics, e.g. for every MIC-formula φ and every transition system $\mathcal{T}$ one can construct in polynomial time a backtracking game that is won by Player $\exists$ if, and only if, $\mathcal{T} \models \varphi$. Here, the backtracking distance plays the role of the stack being used to determine a stage of the fixed-point induction containing the current state of the transition system. The rule that higher priorities reset the distance for all lower priorities corresponds to the usual idea that regenerating an outer fixed point restarts the induction on the inner fixed points.

Unlike the stair parity games we constructed in Section 4.3, it seems unlikely that the winner of backtracking games is definable in MIC, even for the very simple fragment of backtracking games that suffice for a game-theoretical framework for MIC model checking. The reason is that while in a pushdown game, the possible stack contents are represented in the game graph explicitly, the backtracking distance is an "external" concept and counting the distance must be done in the logic itself. Therefore it seems unlikely that MIC suffices for this. In [5], it was shown, however, that the winner of a restricted class of backtracking games can be defined in inflationary fixed-point logic.

5 Outlook

Clearly, MIC and FLC are not the only (modal) fixed-point logics that extend the modal μ-calculus semantically. Another modal fixed-point logic of high expressivity is *Higher-Order Fixed-Point Logic* (HFL) [20]. It incorporates into $\mathcal{L}_\mu$ a simply typed λ-calculus. Its ground type is that of a *predicate* and its only type constructor is the function arrow. Syntactically, HFL extends $\mathcal{L}_\mu$ by function abstraction $(\lambda X.\varphi)$ and application $(\varphi\ \psi)$.

Not surprisingly, HFL subsumes FLC. In fact, every (sub-)formula in FLC is, semantically, a predicate transformer, i.e. an object of a function type. This way, FLC is embedded into a very low level of HFL, namely HFL1—the *First-Order Fragment of HFL*. Here, first order does not refer to predicate logic but to the degree of function typing that is allowed in subformulas of that logic. HFL0, the fragment of HFL restricted to formulas in which every subformula is a predicate, is exactly $\mathcal{L}_\mu$—syntactically already.

The type level hierarchy in HFL is strict, and it comes with increasing model checking complexity: it is k-EXPTIME-complete for level k of that hierarchy [3], and this holds already for the data complexity of each fragment. Consequently, model checking full HFL is non-elementary.

HFL, and in particular HFL1, is also interesting as a specification language for non-regular properties. It can, for instance, define assume-guarantee properties [20]. Furthermore, it can define structural properties that we are unable to express in FLC or MIC like that of being bisimilar to a (possibly infinite) word model. Even though we have not formally defined HFL1, we can present the formula for this property because it is very neat, and it can be read with a little bit of understanding of functional programming.

$$\neg\Big(\ \big(\ \mu X^{\mathrm{Pr}\to\mathrm{Pr}\to\mathrm{Pr}}.\lambda A^{\mathrm{Pr}}.\lambda B^{\mathrm{Pr}}.(A \wedge B) \vee (X\ \Diamond A\ \Diamond B)\ \big)\ \langle a\rangle\mathtt{tt}\ \langle b\rangle\mathtt{tt}\ \Big)$$

The superscripts are type annotations. The least fixed-point formula can be seen as a recursively defined function that takes two predicates and checks whether or not their conjunction holds. If not, it calls itself recursively with the two arguments preceeded by $\Diamond$-operators. Applied to the two initial arguments, it checks successively, whether there are two paths of length $1, 2, \ldots$ ending in an a-, resp. b-transition.

We have not included a thorough presentation of HFL (or just HFL1) here, mainly because there is no interesting known game-theoretic characterization of its model checking problem. It can be solved by a reduction to a reachability game using fixed-point elimination [3], but it is not known whether or not there is an extension of parity games to capture this. The example above suggests that stair parity games do not suffice since two stacks would be needed to contain the $\Diamond$-operators for the two different paths.

We conclude with a positive remark: while FLC is trivially embeddable into HFL1, and MIC and FLC seem incomparable, it is reasonable to ask whether HFL1 is a superlogic of both of them. On finite models, MIC can indeed be embedded into HFL. This is because the *computation* of an inflationary fixed point can be carried out by a function of first-order type. However, since this is modeled iteratively, this translation fails in stages beyond ω. Hence, it may not work on infinite models. It remains to be seen whether MIC can be embedded into HFL over arbitrary models.

References

[1] R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In K. Jensen and A. Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 467–481. Springer, 2004.

[2] A. Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. *ITA*, 33(4/5):329–340, 1999.

[3] R. Axelsson, M. Lange, and R. Somla. The complexity of model checking higher-order fixpoint logic. *Log. Methods Comput. Sci.*, 3(2):2:7, 33 pp. (electronic), 2007.

[4] A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logic. *ACM Trans. Comput. Log.*, 5(2):282–315, 2004.

[5] A. Dawar, E. Grädel, and S. Kreutzer. Backtracking games and inflationary fixed points. *Theor. Comput. Sci.*, 350(2-3):174–187, 2006.

[6] S. Dziembowski, M. Jurdziński, and D. Niwiński. On the expression complexity of the modal μ-calculus model checking. Unpublished manuscript, 1996.

[7] E. A. Emerson. Uniform inevitability is tree automaton ineffable. *Inf. Process. Lett.*, 24(2):77–79, 1987.

[8] O. H. Ibarra, T. Jiang, and H. Wang. A characterization of exponential-time languages by alternating context-free grammars. *Theor. Comput. Sci.*, 99(2):301–313, 1992.

[9] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In U. Montanari and V. Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996.

[10] M. Lange. Alternating context-free languages and linear time mu-calculus with sequential composition. *Electr. Notes Theor. Comput. Sci.*, 68(2), 2002.

[11] M. Lange. The alternation hierarchy in fixpoint logic with chop is strict too. *Inf. Comput.*, 204(9):1346–1367, 2006.

[12] M. Lange. Three notes on the complexity of model checking fixpoint logic with chop. *Theoret. Informatics Appl.*, 41:177–190, 2007.

[13] M. Lange and R. Somla. Propositional dynamic logic of context-free programs and fixpoint logic with chop. *Inf. Process. Lett.*, 100(2):72–75, 2006.

[14] M. Lange and C. Stirling. Model checking fixed point logic with chop. In M. Nielsen and U. Engberg, editors, *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 250–263. Springer, 2002.

[15] C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In K. Lodaya and M. Mahajan, editors, *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 408–420. Springer, 2004.

[16] M. Müller-Olm. A modal fixpoint logic with chop. In C. Meinel and S. Tison, editors, *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 510–520. Springer, 1999.

[17] A. Okhotin. Boolean grammars. *Inf. Comput.*, 194(1):19–48, 2004.

[18] C. Stirling. Lokal model checking games. In I. Lee and S. A. Smolka, editors, *CONCUR*, volume 962 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 1995.

[19] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–309, 1955.

[20] M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In P. Gardner and N. Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 512–528. Springer, 2004.

[21] I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

The universal automaton

Sylvain Lombardy[1]
Jacques Sakarovitch[2]

[1] Institut Gaspard Monge
Université de Paris-Est
5, boulevard Descartes
77454 Champs-sur-Marne, France
Sylvain.Lombardy@univ-mlv.fr

[2] École Nationale Supérieure des Télécommunications / C.N.R.S.
46, rue Barrault
75634 Paris Cedex 13, France
Jacques.Sakarovitch@enst.fr

Abstract

This paper is a survey on the universal automaton, which is an automaton canonically associated with every language. In the last forty years, many objects have been defined or studied, that are indeed closely related to the universal automaton.

We first show that every automaton that accepts a given language has a morphic image which is a subautomaton of the universal automaton of this language. This property justifies the name "universal" that we have coined for this automaton. The universal automaton of a regular language is finite and can be effectively computed in the syntactic monoid or, more efficiently, from the minimal automaton of the language. We describe the construction that leads to tight bounds on the size of the universal automaton. Another outcome of the effective construction of the universal automaton is the computation of a minimal NFA accepting a given language, or approximations of such a minimal NFA. From another point of view, the universal automaton of a language is based on the factorisations of this language, and is thus involved in the problems of factorisations and approximations of languages. Last, but not least, we show how the universal automaton gives an elegant solution to the star height problem for some classes of languages (pure-group or reversible languages).

With every language is canonically associated an automaton, called *the universal automaton of the language*, which is finite whenever the language is regular. It is large, it is complex, it is complicated to compute, but it contains, hopefully, many interesting informations on the language. In the last forty years, it has been described a number of times, more or less explicitly, more or less approximately, in relation with one or another property of the language. This is what we review here systematically.

1 A brief history of the universal automaton

The origin of the universal automaton is not completely clear. A well-publicized note [1] credits Christian Carrez of what seems to be the first definition of the universal automaton in a report that remained unpublished [2]. The problem at stake was the computation of the, or of *a*, NFA with minimal number of states that recognizes a given regular language L. And Carrez's report states the existence of an automaton $\mathcal{U}_L$, very much in the way we do in Section 2, with the property that it contains a morphic image of any automaton which recognizes L, and thus a copy of any minimal NFA which recognizes L.

At about the same time, Kameda and Weiner tackled the same problem and, *without stating the existence of* $\mathcal{U}_L$, described a construction for a NFA recognizing L with minimal number of states [14], a construction which we recognize now as being similar to the construction of $\mathcal{U}_L$ we propose in Section 4.

Soon afterwards, in another context, and with no connexion of any kind with the previous problem (*cf.* Section 6) Conway proposed the definition of what can be seen also as an automaton attached to L and which is again equal to $\mathcal{U}_L$ [5] (*cf.* Section 3.1).

Among other work related to $\mathcal{U}_L$, but without reference to the previous one, let us quote [6] and [22]. Eventually, we got interested in the universal automaton as we discovered it contains other informations on the languages that those studied before (see Section 7.1) and we made the connexion between the different instances [21, 20].

2 Creation of the universal automaton

No wonder, we first fix some notations. If X is a set, $\mathfrak{P}(X)$ denote the *power set* of X, *i.e.* the set of subsets of X. We denote by A^* the free monoid generated by a set A. Elements of A^* are *words*, the identity of A^* is the *empty word*, written 1_{A^*}. The product in A^* is denoted by concatenation and is extended by additivity to $\mathfrak{P}(A^*)$: $XY = \{uv \mid u \in X, v \in Y\}$.

An automaton $\mathcal{A}$ is a 5-tuple $\mathcal{A} = \langle Q, A, E, I, T \rangle$, where Q is a finite set of states, A is a finite set of letters, E, the set of transitions, is a subset of $Q \times A \times Q$, and I (*resp.* T), the set of initial (*resp.* terminal) states, is a subset of Q. Such an automaton $\mathcal{A}$ defines an *action*[1] $\triangleright$ of A^* on $\mathfrak{P}(Q)$,

[1] Normally, we would have denoted the action by a simple $\cdot$; but later, in Section 5, we shall need to consider an action on the right *and* an action on the left, hence a lateralized symbol which makes the reading easier. Moreover, when necessary, *i.e.* when several automata are considered at the same time, we shall even specify as a

by setting first for all $p \in Q$ and all $a \in A$

$$p \triangleright a = \{q \in Q \mid (p,a,q) \in E\},$$

and then by additivity and the definition of an action for all $X \in \mathfrak{P}(Q)$

$$X \triangleright 1_{A^*} = X\,, \quad X \triangleright a = \bigcup_{p \in X} p \triangleright a\,, \quad X \triangleright wa = (X \triangleright w) \triangleright a.$$

The behaviour $|\mathcal{A}|$ (or the accepted language) of an automaton $\mathcal{A} = \langle Q, A, E, I, T\rangle$ is the set of words that label a path from an initial state to a terminal state, *i.e.*

$$|\mathcal{A}| = \{w \in A^* \mid \exists i \in I, \quad t \in T \quad i \xrightarrow[\mathcal{A}]{w} t\} = \{w \in A^* \mid I \triangleright w \cap T \neq \varnothing\}.$$

A subset of A^* is called a *language* and a language is *regular* if it is the behaviour of some finite automaton.

Let $\mathcal{A} = \langle Q, A, E, I, T\rangle$ be an automaton over A^*. For each state p of $\mathcal{A}$, the *past of* p is the set of labels of computations which go from an initial state of $\mathcal{A}$ to p, and we write it $\mathsf{Past}_{\mathcal{A}}(p)$; *i.e.*

$$\mathsf{Past}_{\mathcal{A}}(p) = \{w \in A^* \mid \exists i \in I \quad i \xrightarrow[\mathcal{A}]{w} p\} = \{w \in A^* \mid p \in I \triangleright w\}.$$

Dually, the *future of* p is the set of labels of computations that go from p to a final state of $\mathcal{A}$ and we write it $\mathsf{Fut}_{\mathcal{A}}(p)$, *i.e.*:

$$\mathsf{Fut}_{\mathcal{A}}(p) = \{w \in A^* \mid \exists t \in T \quad p \xrightarrow[\mathcal{A}]{w} t\} = \{w \in A^* \mid p \triangleright w \cap T \neq \varnothing\}.$$

Likewise, for each pair of states (p,q) of $\mathcal{A}$, the transitional language of (p,q) is the set of labels of computations that go from p to q and we write it $\mathsf{Trans}_{\mathcal{A}}(p,q)$, *i.e.*:

$$\mathsf{Trans}_{\mathcal{A}}(p,q) = \{w \in A^* \mid p \xrightarrow[\mathcal{A}]{w} q\} = \{w \in A^* \mid q \in p \triangleright w\}.$$

For each q in Q, we clearly have

$$[\mathsf{Past}_{\mathcal{A}}(q)]\,[\mathsf{Fut}_{\mathcal{A}}(q)] \subseteq |\mathcal{A}|. \qquad (*)$$

Thus, in every automaton, each state induces a set of ‘factorisations’ — which is the name we give to equations of the type $(*)$ — of the language it recognizes. The starting point of the construction is to prove the converse of this observation, namely that we can construct from the set of factorisations of a language L of A^* an automaton which accepts L.

subscript the automaton that defines the action in action: $p \underset{\mathcal{A}}{\triangleright} a$.

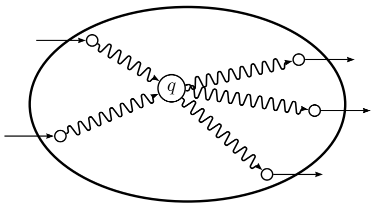


FIGURE 1. Representation of the past and future of q in $\mathcal{A}$

2.1 Factorisations of a language

In the rest of this paper, L is a language of A^*. We call *subfactorisation* of L a pair (X, Y) of languages of A^* such that $XY \subseteq L$ and *factorisation* a subfactorisation (X, Y) which is maximal for the inclusion, that is, if $X \subseteq X'$, $Y \subseteq Y'$ and $X'Y' \subseteq L$ then $X = X'$ and $Y = Y'$. We write $\mathcal{F}_L$ for the set of factorisations of L. If (X, Y) is in $\mathcal{F}_L$ then X is called a *left factor* and Y a *right factor* of L. The maximality condition on factorisations already implies that the left and right factors are in a 1-1 correspondence. The notion of quotient allows to be even more precise.

The *left quotient* (*resp.* the *right quotient*) of L by a word v is the language[2] $v^{-1}L = \{w \in A^* \mid vw \in L\}$ (*resp.* the language $Lv^{-1} = \{u \in A^* \mid uv \in L\}$).

If $WZ \subseteq L$ then Z is contained in $Y = \bigcap_{w \in W} w^{-1}L$ and thus Y is maximum such that $WY \subseteq L$. From which a series of properties are easily derived, that are worth stating for further usage, with, or without, explicit reference.

Proposition 2.1.

(i) For every (X, Y) in $\mathcal{F}_L$,

$$Y = \bigcap_{x \in X} x^{-1}L \quad \text{and} \quad X = \bigcap_{y \in Y} Ly^{-1}.$$

(ii) Conversely, any intersection of left quotients is a right factor, and any intersection of right quotients is a left factor.

(iii) If W and Z are such that $WZ \subseteq L$, then there exists (at least) one factorisation (X, Y) of L such that $W \subseteq X$ and $Z \subseteq Y$

(iv) The property '(X, Y) is a factorisation of L' induces a bijection between the left and right factors of L.

[2] Sometimes called *residual* of L.

Corollary 2.2. A language is regular if, and only if, it has a finite number of factorisations.

Remark 2.3. We write L^{t} for the *transpose* of L, that is, the set of mirror image of words in L. If (X, Y) is a factorisation of L, $(Y^{\mathrm{t}}, X^{\mathrm{t}})$ is a factorisation of L^{t}. By *duality*, we unterstand the change from L to L^{t}.

2.2 Universal automaton of a language

The definition of factorisations of a language allows in turn to set up the definition we are aiming at.

Definition 2.4. The *universal automaton* $\mathcal{U}_L$ of L is defined as $\mathcal{U}_L = \langle \mathcal{F}_L, A, E^L, I^L, T^L \rangle$, where:

$$I^L = \{(X,Y) \in \mathcal{F}_L \mid 1_{A^*} \in X\}\,, \quad T^L = \{(X,Y) \in \mathcal{F}_L \mid 1_{A^*} \in Y\},$$
$$E^L = \{\big((X,Y), a, (X',Y')\big) \in \mathcal{F}_L \times A \times \mathcal{F}_L \mid XaY' \subseteq L\}.$$

From the maximality of the factorisations follows:

$$(X,Y) \in I^L \iff Y \subseteq L\,, \quad (X,Y) \in T^L \iff X \subseteq L, \tag{1.1}$$
$$\big((X,Y), a, (X',Y')\big) \in E^L \iff Xa \subseteq X' \iff aY' \subseteq Y. \tag{1.2}$$

The description of computations in the universal automaton is then a generalisation of the above equation.

Lemma 2.5. For all (X, Y) and (X', Y') in $\mathcal{F}_L$ and for every w in A^+, it holds:

$$(X,Y) \xrightarrow[\mathcal{U}_L]{w} (X',Y') \iff XwY' \subseteq L \iff Xw \subseteq X' \iff wY' \subseteq Y.$$

Proof. By induction on $|w|$. The property holds true for $|w| = 1$, by definition of E^L and by (1.2).

Suppose that $(X,Y) \xrightarrow[\mathcal{U}_L]{aw} (X',Y')$; there exists then (X'',Y'') in $\mathcal{F}_L$ such that $(X,Y) \xrightarrow[\mathcal{U}_L]{a} (X'',Y'')$ and $(X'',Y'') \xrightarrow[\mathcal{U}_L]{w} (X',Y')$. We thus have $Xa \subseteq X''$ and $X''w \subseteq X'$, hence $XawY' \subseteq L$.

Conversely, $XawY' = [Xa]\,[wY'] \subseteq L$ implies that there exists (X'', Y'') in $\mathcal{F}_L$ such that $Xa \subseteq X''$ and $wY' \subseteq Y''$, thus $XaY'' \subseteq L$ and $X''wY' \subseteq L$, which, by induction hypothesis, gives $(X,Y) \xrightarrow[\mathcal{U}_L]{aw} (X',Y')$. Q.E.D.

A fundamental property of the universal automaton is given by the following.

Proposition 2.6. If (X, Y) is a factorisation of L, it then holds:

$$\mathsf{Past}_{\mathcal{U}_L}((X,Y)) = X \quad \text{and} \quad \mathsf{Fut}_{\mathcal{U}_L}((X,Y)) = Y.$$

Proof. The definition of T^L itself states that $\mathsf{Fut}_{\mathcal{U}_L}((X,Y))$ contains 1_{A^*} if, and only if, Y contains 1_{A^*}. Let w be a non empty word in $\mathsf{Fut}_{\mathcal{U}_L}((X,Y))$, that is, $(X,Y) \xrightarrow[\mathcal{U}_L]{w} (X',Y')$ with 1_{A^*} in Y'. By Lemma 2.5, $XwY' \subseteq L$; as 1_{A^*} is in Y', $Xw \subseteq L$ and w is in Y by maximality of Y. Therefore $\mathsf{Fut}_{\mathcal{U}_L}((X,Y)) \subseteq Y$.

Conversely, if (X,Y) is in $\mathcal{F}_L$, $XY = [XY]\,[1_{A^*}] \subseteq L$ and there exists a right factor Y', containing 1_{A^*}, such that $XYY' \subseteq L$. By Lemma 2.5 again, $Y \subseteq \mathsf{Fut}_{\mathcal{U}_L}((X,Y))$.

The other equality is obtained by duality. Q.E.D.

As $1_{A^*}L = L1_{A^*} = L$, L is both a right and a left factor, to which correspond the left factor X_s and the right factor Y_e. We call (X_s, L) the *starting factorisation*, and (L, Y_e) the *ending factorisation.*[3]

Corollary 2.7. $\mathcal{U}_L$ recognises L.

Proof. For any factorisation (X,Y) in I^L, $Y \subseteq L$ since $1_{A^*} \in X$. Then $|\mathcal{U}_L| = \bigcup_{(X,Y)\in I^L} \mathsf{Fut}_{\mathcal{U}_L}((X,Y)) = \bigcup_{(X,Y)\in I^L} Y$ is contained in L. Since $(X_s, L) \in I^L$ then $|\mathcal{U}_L| = L$. Q.E.D.

The universal automaton is canonically associated with L, like the *minimal deterministic automaton* or the *minimal co-deterministic automaton*; unlike them, it is not lateralised, that is, not oriented from left to right nor from right to left. It is a restatement of Corollary 2.2 that L is regular if, and only if, $\mathcal{U}_L$ is finite. And the universal automaton $\mathcal{U}_{L^{\mathsf{t}}}$ of L^{t} is the transpose automaton of $\mathcal{U}_L$.

Example 2.8.

(i) Let $L_1 = A^*abA^*$. The set $\mathcal{F}_{L_1} = \{u,v,w\}$ is easily computed: $u = (A^*, A^*abA^*)$, $v = (A^*aA^*, A^*bA^*)$ and $w = (A^*abA^*, A^*)$. Figure 2 shows $\mathcal{U}_{L_1}$.

(ii) Figure 2 also shows the universal automaton of $L_2 = aA^*$. This example allows us to see that a universal automaton is not necessarily trim: a factorisation $(\varnothing, A^*)$ (*resp.* $(A^*, \varnothing)$), if it exists, corresponds to a non-accessible (*resp.* a non-co-accessible) state.

2.3 Universality of the universal automaton

We begin with the definition of *morphism of automata* and some related notions that will be central to our purpose.

[3] Conway, who did not define the universal automaton as such, called them *initial* and *final* factorisation respectively, an option that is not open to us.

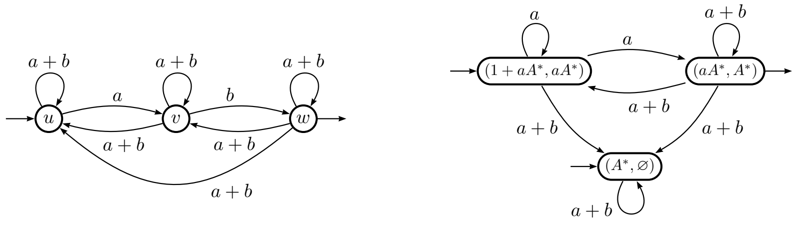


FIGURE 2. The universal automaton of L_1 (left) and of L_2 (right)

2.3.1 Morphisms, quotients and minimal automata

In the sequel, $\mathcal{A} = \langle Q, A, E, I, T\rangle$ and $\mathcal{B} = \langle R, A, F, J, U\rangle$ are two automata over A^*.

Definition 2.9. A map φ from Q into R is a *morphism of automata*, and we write $\varphi\colon \mathcal{A} \to \mathcal{B}$ if, and only if,

$$\varphi(I) \subseteq J, \quad \varphi(T) \subseteq U, \quad \text{and } \varphi(E) = \{(\varphi(p), a, \varphi(q)) \mid (p, a, q) \in E\} \subseteq F\,.$$

The morphism φ is *surjective* if $\mathcal{B} = \langle \varphi(Q), A, \varphi(E), \varphi(I), \varphi(T)\rangle$ (we also say that $\mathcal{B}$ is a *morphic image* of $\mathcal{A}$).

If $\varphi\colon \mathcal{A} \to \mathcal{B}$ is a morphism, the image of a computation in $\mathcal{A}$ is a computation in $\mathcal{B}$, with the same label, which directly implies the following.

Proposition 2.10. Let φ be a morphism from $\mathcal{A}$ into $\mathcal{B}$. Then, for every state p of $\mathcal{A}$,

$$\mathsf{Past}_{\mathcal{A}}(p) \subseteq \mathsf{Past}_{\mathcal{B}}(\varphi(p))\,, \quad \mathsf{Fut}_{\mathcal{A}}(p) \subseteq \mathsf{Fut}_{\mathcal{B}}(\varphi(p)), \tag{1.3}$$

and then

$$|\mathcal{A}| \subseteq |\mathcal{B}|. \tag{1.4}$$

The notion of morphism is not lateralised and if $\varphi\colon \mathcal{A} \to \mathcal{B}$ is a morphism then so is $\varphi\colon \mathcal{A}^{\mathsf{t}} \to \mathcal{B}^{\mathsf{t}}$. If $\varphi\colon \mathcal{A} \to \mathcal{B}$ is a surjective morphism and if moreover $|\mathcal{A}| = |\mathcal{B}|$, then any two states p and q of $\mathcal{A}$ such that $\varphi(p) = \varphi(q)$ are said to be *mergible* (in $\mathcal{A}$).

Proposition 2.11. The universal automaton $\mathcal{U}_L$ has no mergible states.

Proof. Suppose, by way of contradiction, that $\varphi\colon \mathcal{U}_L \to \mathcal{C}$ is a surjective morphism and that $|\mathcal{C}| = L$.

If $\varphi((X, Y)) = \varphi((X', Y')) = s$ the combination of Proposition 2.6 and Proposition 2.10 yields $X \cup X' \subseteq \mathsf{Past}_{\mathcal{C}}(s)$ and $Y \cup Y' \subseteq \mathsf{Fut}_{\mathcal{C}}(s)$ from which follows $(X \cup X')(Y \cup Y') \subseteq \mathsf{Past}_{\mathcal{C}}(s)\mathsf{Fut}_{\mathcal{C}}(s) \subseteq L$, impossible by the maximality of factorisations. Q.E.D.

Definition 2.12. A morphism $\varphi\colon \mathcal{A} \to \mathcal{B}$ is *Out-surjective* if

(i) for every (r, a, s) in F and every p such that $\varphi(p) = r$ there exists q such that $\varphi(q) = s$ and (p, a, q) in E;

(ii) for every p in Q, if $\varphi(p)$ is in U then p is in T.

The notion of Out-surjectivity is lateralised and $\varphi\colon \mathcal{A} \to \mathcal{B}$ is said to be *In-surjective* if $\varphi\colon \mathcal{A}^{\mathrm{t}} \to \mathcal{B}^{\mathrm{t}}$ is Out-surjective. If $\varphi\colon \mathcal{A} \to \mathcal{B}$ is both surjective and Out-surjective (*resp.* and In-surjective) $\mathcal{B}$ — and φ — is called a *quotient* (*resp.* a *co-quotient*) of $\mathcal{A}$. An easy proof by induction on the length of the computations establishes the following.

Proposition 2.13. If the automaton $\mathcal{B}$ is a quotient (*resp.* a co-quotient) of the automaton $\mathcal{A}$ then $|\mathcal{A}| = |\mathcal{B}|$.

We thus have three distinct notions of maps for automata: morphism, quotient, and co-quotient, that lead to three distinct notions of *minimality.* The minimal quotient of a (non deterministic) automaton $\mathcal{A}$ exists and is unique, canonically associated with $\mathcal{A}$ — not with $|\mathcal{A}|$ unless $\mathcal{A}$ is deterministic —, defined by a generalisation of the so-called Nerode equivalence, and computed, if necessary, by a kind of Moore algorithm. The same is true of co-quotient, up to a transposition. The notion of minimality with respect to morphism is slightly more tricky and unicity is lost.

Definition 2.14. Let $\mathcal{A}$ be an automaton over A^* that accepts a language L. We say that $\mathcal{A}$ is *m-minimal* if the following two properties hold:

(i) every proper subautomaton of the *trim* part of $\mathcal{A}$ accepts a language that is strictly contained in L;

(ii) every proper morphic image of $\mathcal{A}$ accepts a language that contains strictly L.

In other words, an automaton is m-minimal if every state is necessary –unless it is a sink or a co-sink– and no two states are mergible.

We have decided to coin that new term ‘m-minimal’ for there are too many ‘minimal’ around. A minimal quotient is not necessarily m-minimal and the sentence ‘*A minimal quotient is not necessarily minimal*’ sounds definitively too awkward. Of course, neither a minimal quotient, nor a m-minimal automaton have a minimal number of states for accepting the same language. Some consistency is given by the following.

Proposition 2.15. The *minimal automaton* of a language L (which is the minimal quotient of any *deterministic* automaton that recognises L) is m-minimal.

Proof. Every state p of the minimal automaton of L is characterised by its future which is equal to $u^{-1}L$, for any u in its past. If p and q are two distinct states there is one, say p, whose future contains a word w which is not in the future of q. For any v in the past of q, w does not belong to $v^{-1}L$, that is, vw does not belong to L and still would be accepted in any morphic image where p and q were merged. Q.E.D.

2.3.2 Morphisms into the universal automaton

The following property of the universal automaton is the one that has been appealing to most people. We call it 'universality property' and the universal automaton gets its name from it.

Theorem 2.16. If $\mathcal{A}$ is an automaton that recognises any subset K of L, then there exists a morphism from $\mathcal{A}$ into $\mathcal{U}_L$.

This result is established via the definition of a map from $\mathcal{A}$ into $\mathcal{U}_L$, canonically associated with $\mathcal{A}$, and which is then shown to be a morphism.

Definition 2.17. Let $\mathcal{A} = \langle Q, A, E, I, T\rangle$ be an automaton that recognises a subset K of L. The *(left) canonical map* $\varphi\colon Q \to \mathcal{F}_L$ is defined by $\varphi(p) = (X_p, Y_p)$ with

$$Y_p = \{v \in A^* \mid \mathsf{Past}_{\mathcal{A}}(p)v \subseteq L\} = \bigcap_{u \in \mathsf{Past}_{\mathcal{A}}(p)} u^{-1}L. \tag{1.5}$$

In other words, φ is defined by associating with every state p of $\mathcal{A}$ the factorisation of L with the largest possible right factor that is compatible with the *past* of p in $\mathcal{A}$.

Proof of Theorem 2.16. Let p in Q and $\varphi(p) = (X_p, Y_p)$. It follows directly from the definition that $\mathsf{Past}_{\mathcal{A}}(p) \subseteq X_p$ and $\mathsf{Fut}_{\mathcal{A}}(p) \subseteq Y_p$ from which we deduce that $\varphi(I) \subseteq I^{L}$ and $\varphi(T) \subseteq T^{L}$.

Moreover, (p, a, q) in E implies $\mathsf{Past}_{\mathcal{A}}(p)a \subseteq \mathsf{Past}_{\mathcal{A}}(q)$ from which one deduces $\mathsf{Past}_{\mathcal{A}}(p)aY_q \subseteq \mathsf{Past}_{\mathcal{A}}(q)Y_q \subseteq L$ hence $aY_q \subseteq Y_p$ and by (1.2), φ is a morphism, that will be called *(left) canonical morphism* (from $\mathcal{A}$ to $\mathcal{U}_L$). Q.E.D.

If we apply Theorem 2.16 to a m-minimal automaton $\mathcal{A}$ accepting L we get a morphism from $\mathcal{A}$ into $\mathcal{U}_L$ that has to be injective since $\mathcal{A}$ is m-minimal. We have thus proved (see an example at Figure 3):

Corollary 2.18. Every m-minimal automaton accepting L is a subautomaton of $\mathcal{U}_L$.

On the other hand, an automaton $\mathcal{A}$ accepting L and that has stricly more states than $\mathcal{U}_L$ is sent into $\mathcal{U}_L$ by a morphism which is necessarily non injective. We have thus proved:

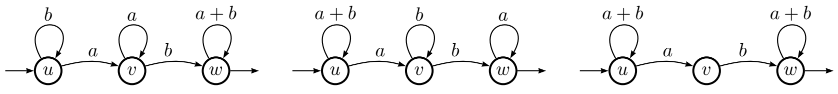


FIGURE 3. Three m-minimal subautomata of $\mathcal{U}_{L_1}$

Corollary 2.19. The universal automaton $\mathcal{U}_L$ is the largest automaton recognizing L without merging states.

Proposition 2.20. The universal automaton $\mathcal{U}_L$ is minimal for the universality property.

Proof. Suppose $\mathcal{C}$ has the universality property (with respect to L). As $\mathcal{U}_L$ accepts L, there should be a morphism from $\mathcal{U}_L$ into $\mathcal{C}$; as $\mathcal{U}_L$ has no merging states, this morphism should be injective: $\mathcal{C}$ has at least as many states as $\mathcal{U}_L$. Q.E.D.

3 Exploration of the universal automaton

The universal automaton we have just defined may be seen in different ways, from different perspective, bringing to light other characteristics and properties of this unique and canonical object. We consider here three of them. The first one is Conway's method, that yields the 'fatest' version of the universal automaton. The second one follows a universal algebra track that eventually makes easy and natural a geometric description of factorisations that was presented by Courcelle, Niwinski and Podelski ([6]). The third one, due to Lombardy [17] produces the most 'emaciated' version, an automaton where only the minimal information is kept and where an interesting and hidden structure is thus discovered, especially in the case of pure group languages.

3.1 The factor matrix

We keep the previous notation: $\mathcal{U}_L = \langle \mathcal{F}_L, A, E^L, I^L, T^L \rangle$ is the universal automaton of the language L of A^*. Automata are matrices (and vectors); this is the way we look at them in this section. As we are interested in matrices (and vectors) of dimension $\mathcal{F}_L$, we use throughout the section the following notation: if M is a square matrix of dimension $\mathcal{F}_L$ and for brevity, we write $M_{X,Y'}$ instead of $M_{(X,Y),(X',Y')}$ for the entry at row (X,Y) and column (X',Y'), for all (X,Y) and (X',Y') in $\mathcal{F}_L$. For a row-vector (*resp.* a column-vector) V we write V_Y (*resp.* V_X) instead of $V_{(X,Y)}$. Proposition 2.1 (iv) legitimates this shorthand.

A first example is E^L itself, viewed as a matrix with entries in $\mathfrak{P}(A^*)$ (indeed in $\mathfrak{P}(A)$):

$$E^L_{X,Y'} = \{a \in A \mid XaY' \subseteq L\}$$

for all factorisations (X,Y) and (X',Y') in $\mathcal{F}_L$. On the other hand, the left factors are naturally ordered by inclusion, an order that carries over on $\mathcal{F}_L$:

$$(X,Y) \leqslant (X',Y') \iff X \subseteq X' \iff Y' \subseteq Y.$$

As any relation on $\mathcal{F}_L$, this order is described by a Boolean matrix C^L:

$$\forall (X,Y),(X',Y') \in \mathcal{F}_L \qquad C^L_{X,Y'} = 1 \iff X \subseteq X' \iff XY' \subseteq L;$$

and since C^L is the matrix of a reflexive and transitive relation, it holds:

$$(C^L)^* = C^L. \tag{1.6}$$

The characterisation of E^L by Equation (1.2) yields that $C^L_{X,Y'} = 1$ implies, for all (X'',Y'') in $\mathcal{F}_L$, $E^L_{X',Y''} \subseteq E^L_{X,Y''}$ and $E^L_{X'',Y} \subseteq E^L_{X'',Y'}$, which means:

$$C^L \cdot E^L = E^L \cdot C^L = E^L. \tag{1.7}$$

Definition 3.1. The *factor matrix* of a language L is the matrix F^L of dimension $\mathcal{F}_L$ with entries in $\mathfrak{P}(A^*)$ defined by:

$$F^L_{X,Y'} = \{w \in A^* \mid XwY' \subseteq L\}$$

for all factorisations (X,Y) and (X',Y') in $\mathcal{F}_L$. Every entry of F^L is called *a factor of* L.

By definition, $F^L_{X,Y'}$ is the maximal Z such that $XZY' \subseteq L$. By definition[4] also, $F^L \cap \{1_{A^*}\} = C^L$ and $F^L \cap \{A\} = E^L$. Lemma 2.5 states exactly that

$$F^L \cap \{A^+\} = (E^L)^+ \quad \text{and thus} \quad F^L = C^L + (E^L)^+ = C^L + (E^L)^*.$$

Classical formulas for the star of a sum, together with (1.6) and (1.7) yields:

Proposition 3.2. $F^L = (C^L + E^L)^*$.

From which one deduces:

Corollary 3.3. $F^L = (F^L)^*$.

A direct consequence of which is:

$$\forall (X,Y),(X',Y'),(X'',Y'') \in \mathcal{F}_L \qquad F^L_{X,Y'} F^L_{X',Y''} \subseteq F^L_{X,Y''}. \tag{1.8}$$

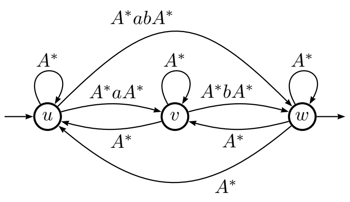
Conversely, we have:

[4] It should be obvious that $F^L \cap K$ is the matrix of dimension $\mathcal{F}_L$ obtained by taking the intersection of every entry of F^L with K.

Lemma 3.4. If $W, Z \subseteq A^*$ and (X, Y), (X', Y') in $\mathcal{F}_L$ are such that $WZ \subseteq F^L_{X,Y'}$ then there exists (X'', Y'') in $\mathcal{F}_L$ such that $W \subseteq F^L_{X,Y''}$ and $Z \subseteq F^L_{X'',Y'}$.

Proof. If $WZ \subseteq F^L_{X,Y'}$ then $XWZY' \subseteq L$ and there exists a factorisation (X'', Y'') that dominates the subfactorisation (XW, ZY') of L. The inclusions $XW \subseteq X''$ and $ZY' \subseteq Y''$ yield the conclusion. Q.E.D.

A matrix, together with initial and final vectors, is an automaton and one can see $\langle \mathcal{F}_L, A, F^L, I^L, T^L \rangle$ as a generalised automaton where the transitions are labelled by the factors of L, instead of by letters. Figure 4 shows the factor matrix of the languages L_1 and L_2 of Example 2.8 in this way.



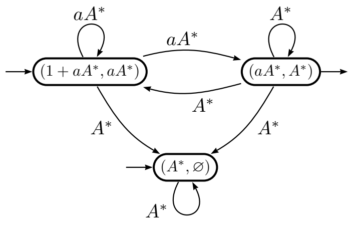


FIGURE 4. The factor matrix of L_1 (left) and of L_2 (right)

The starting and ending factorisations play a special role in the factor matrix. Since $X_s L = L Y_e = L$, we have $X_s L Y_e = L$ where L is obviously maximal: $F^L_{X_s,Y_e} = L$.

For every (X, Y) in $\mathcal{F}_L$, $F^L_{X_s,Y}$ is maximal in $X_s F^L_{X_s,Y} Y \subseteq L$ thus in the factorisation $(F^L_{X_s,Y}, Y)$ of $Y_s = L$, hence $F^L_{X_s,Y} = X$ and dually $F^L_{X,Y_e} = Y$.

3.2 The syntactic nature of the universal automaton

All that has been done so far for *languages*, that is, subsets of a free monoid, could have easily been done as well for subsets in any monoid: the freeness of the base monoid A^* was not involved, at the very most the generators of A^* were considered but this also could have been bypassed, especially with the help of the factor matrix. If M is a monoid and K a subset of M, a *subfactorisation* of K is a pair (X, Y) of subsets of M such that $XY \subseteq K$ and a *factorisation* is a subfactorisation (X, Y) that is maximal for the inclusion, that is, if $X \subseteq X'$, $Y \subseteq Y'$ and $X'Y' \subseteq K$ then $X = X'$ and $Y = Y'$. We write $\mathcal{F}_K$ for the set of factorisations of K. If (X, Y) is in $\mathcal{F}_K$ then X is called a *left factor* and Y a *right factor* of K. And so on.

On the other hand, the study of regular languages relies heavily on the notion of morphisms (of monoids) and that of syntactic monoid (of a language). A language L of A^* is said to be *recognised by a morphism* α,

$\alpha\colon A^* \to N$, if $\alpha^{-1}(\alpha(L)) = L$ or, which is the same, if L is a union of classes for the map equivalence of α, that is a congruence of A^*. The same could be said of a subset K, replacing L, of a monoid M, replacing A^*. The quotient of A^* by the coarsest congruence that saturates L is the *syntactic monoid* of L, denoted $\mathrm{Synt}(L)$. A language of A^*, a subset of a monoid M, is said to be *recognisable* if it is recognised by a morphism into a *finite* monoid, or, which is the same, if *its syntactic monoid is finite.*

We like to say that a property is '*syntactic*' if true for a language L, or a subset K, recognised by a *surjective* morphism α, it is true for $\alpha(L)$ or $\alpha(K)$. The factorisations, the universal automaton, are 'syntactic objects', as shown by the following.

Proposition 3.5. Let L be a language of A^*, recognised by a *surjective* morphism α.

(i) Any factor of L is recognised by α.

(ii) If (X, Y) is a factorisation of L, $(\alpha(X), \alpha(Y))$ is a factorisation of $\alpha(L)$.

(iii) α establishes a bijection between the factorisations of L and those of $\alpha(L)$.

Proof. Let (X, Y) be a factorisation of L: $XY \subseteq L$. Then $\alpha(X)\alpha(Y) \subseteq \alpha(L)$ and $(\alpha(X), \alpha(Y))$ is a *sub*factorisation of $\alpha(L)$ which we suppose dominated by a factorisation (U, V). From $\alpha(X) \subseteq U$ and $\alpha(Y) \subseteq V$ we deduce $X \subseteq \alpha^{-1}(\alpha(X)) \subseteq \alpha^{-1}(U)$ and $Y \subseteq \alpha^{-1}(\alpha(Y)) \subseteq \alpha^{-1}(V)$ and $\alpha^{-1}(U)\alpha^{-1}(V) \subseteq \alpha^{-1}(\alpha(L)) = L$. Since (X, Y) is a factorisation, $X = \alpha^{-1}(U)$ and $Y = \alpha^{-1}(V)$.

This demonstrates at the same time that, (i) $X = \alpha^{-1}(\alpha(X))$ and $Y = \alpha^{-1}(\alpha(Y))$, and (ii) $\alpha(X) = U$ and $\alpha(Y) = V$: $(\alpha(X), \alpha(Y))$ is a factorisation of $\alpha(L)$.

For the same reason, $\alpha^{-1}(\alpha(F^L_{X,Y})) = F^L_{X,Y}$ for all factorisations (X, Y) and (X', Y') of L.

Conversely, let (U, V) be a factorisation of $\alpha(L)$; then $(\alpha^{-1}(U), \alpha^{-1}(V))$ is a *sub*factorisation of $\alpha^{-1}(\alpha(L)) = L$ which we suppose dominated by a factorisation (X, Y). Since (U, V) is a factorisation, neither $U \subseteq \alpha(X)$ or $V \subseteq \alpha(Y)$ may be strict inclusion and $(\alpha^{-1}(U), \alpha^{-1}(V))$ is a factorisation. Q.E.D.

Example 3.6. The syntactic monoid of L_1 is $M_1 = \{1_{M_1}, x, y, t, z\}$ defined by the relations $xx = x$, $yy = y$, $yx = t$, and $xy = xt = ty = z$. The syntactic morphism $\alpha\colon A^* \to M_1$ sends a onto x and b onto y. Then $\alpha(L_1) = z$ and the factorisations of z in M_1 are $(\{z\}, M_1)$, $(M_1, \{z\})$, and $(\{x, t, z\}, \{y, t, z\})$.

Let M be any monoid and let $\psi_M \colon M \times M \to M$ be the map defined by $\psi_M((u,v)) = uv$. (This map is not a morphism unless M is commutative.) It can be seen as the *multiplication table* of M: in a matrix T of size $M \times M$, each element m appears as the entry (u,v) of T, for all (u,v) in $\psi_M^{-1}(m)$.

A factorisation of a subset K of M appears as a maximal rectangle in the subset $\psi_M^{-1}(K)$ and this point of view, possible in the general case, is without doubt the simplest *when M is finite.*

Example 3.7. The table of the monoid M_1, cleverly laid out (we have inverted the order of the elements x and y by row and column) is shown in Figure 5. The factorisations of the subset $\{z\}$ are made clearly visible with rectangles. The figure also shows *the factor matrix* of $\{z\}$, under the form of an automaton labelled with subsets.

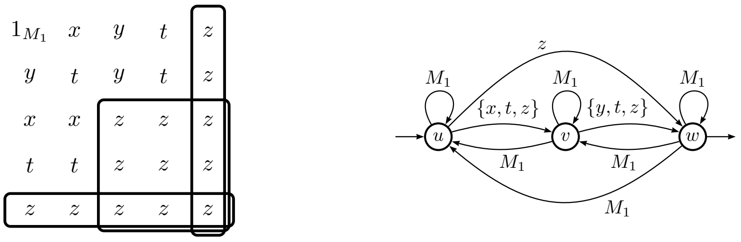


FIGURE 5. Factorisations and factor matrix of $\{z\}$ in M_1

Example 3.8. We consider the (additive) monoid $\mathbb{Z}/3\mathbb{Z}$. Figure 6 shows the factorisations of the subset $\{1, 2\}$ and its universal automaton. The states are labelled by the left factor of the corresponding factorisation, and the label of the transitions is always the generator 1. This automaton is thus (up the addition of the transitions having the label b) the universal automaton of the language $L_3 = \{w \in \{a,b\}^* \mid |w|_a \not\equiv |w|_b \mod 3\}$ since $\mathrm{Synt}(L_3) = \mathbb{Z}/3\mathbb{Z}$ and the image of L_3 there is $\{1, 2\}$.

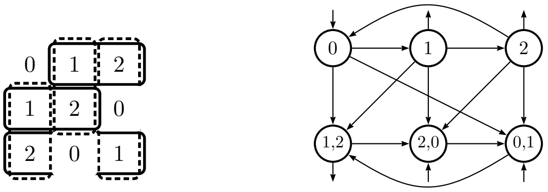


FIGURE 6. Factorisations and universal automaton of $\{1, 2\}$ in $\mathbb{Z}/3\mathbb{Z}$

Proposition 3.5 holds for a recognisable subset K of any monoid M. This implies that such a subset is accepted by an automaton with a finite number of states, whose transition matrix is the factor matrix F^L. To make this automaton really finite, the monoid is required to be generated by a finite set G, and the transitions of the universal automaton of K are then given by $F^L \cap G$. This automaton accepts K, and more precisely, for every element x of K, for every factorisation $x = x_1 \ldots x_n$ of x over G, the sequence $(x_1, \ldots, x_n)$ is the label of (at least) one computation of the universal automaton. This is the reason why the universal automaton relates to recognisable subsets and not to rational subsets.

Actually, a subset K of a monoid is rational if and only if there exists a finite automaton such that for every element of K, there is at least a factorisation of this element that labels a computation, whereas a subset K of a finitely generated monoid is recognisable if and only if there exists a finite automaton such that for every element of K, *every* factorisation of this element (*w.r.t.* the generators) is accepted. [5]

3.3 The écorché of the universal automaton

The order on the factorisations of L considered above (and induced by the inclusion order on the left factors) can be used to give a simplified description of $\mathcal{U}_L$. Indeed, if $(X,Y) \xrightarrow[\mathcal{U}_L]{a} (X',Y')$ then,

$$\forall (X_1,Y_1) \in \mathcal{F}_L \qquad (X_1,Y_1) \leqslant (X,Y) \Longrightarrow (X_1,Y_1) \xrightarrow[\mathcal{U}_L]{a} (X',Y'),$$
$$\forall (X_2,Y_2) \in \mathcal{F}_L \qquad (X',Y') \leqslant (X_2,Y_2) \Longrightarrow (X,Y) \xrightarrow[\mathcal{U}_L]{a} (X_2,Y_2).$$

Moreover, if (X,Y) is initial, any larger factorisation is initial and, dually, if it is final, any smaller factorisation is final. The order on factorisations is described by the matrix C^L and what we have just observed is a rewording of (1.7): $C^L \cdot E^L = E^L \cdot C^L = E^L$ and of $I^L = C^L \cdot I^L$ and $T^L = T^L \cdot C^L$.

A solution of $XaY' \subseteq L$ is *maximal* if

$$X_1 a Y_2 \subseteq L \quad \text{and} \quad X \subseteq X_1 \,, \quad Y' \subseteq Y_2 \quad \Longrightarrow \quad X = X_1 \quad \text{and} \quad Y' = Y_2$$

for all factorisations (X_1,Y_1) and (X_2,Y_2) in $\mathcal{F}_L$. That is, (X,Y) is as large as possible, and (X',Y') as small as possible such that $(X,Y) \xrightarrow[\mathcal{U}_L]{a} (X',Y')$.

We then define the matrix H^L of dimension $\mathcal{F}_L$ and with entries in $\mathfrak{P}(A)$ by

$$a \in H^L_{X,Y'} \quad \Longrightarrow \quad XaY' \subseteq L \quad \text{is maximal.}$$

[5] By virtue of Kleene Theorem, in the free monoid, recognisable subsets and rational subsets are the same: they are regular languages.

On the other hand, we note that the *starting factorisation* (X_s, L) is the smallest factorisation that is initial in $\mathcal{U}_L$ and, dually, the *ending factorisation* (L, Y_e) is the largest factorisation that is final. All these observations amount to the following.

Proposition 3.9.

(i) H^L is the minimal matrix such that $E^L = C^L \cdot H^L \cdot C^L$;

(ii) I^L is the X_sth row of C^L;

(iii) T^L is the Y_eth column of C^L.

Further economy in the description consists in considering the “maximal” solutions of $XY' \subseteq L$ that are not in $\mathcal{F}_L$ and in defining the Boolean matrix D^L by:

$$D^L_{X,Y'} = 1 \quad \Longleftrightarrow \quad (X,Y) \in \max\{(X'', Y'') \in \mathcal{F}_L \mid X \subset X'\}.$$

That is, $D^L_{X,Y'}$ is the matrix of the Hasse diagram of the order on factorisations.

This definition directly yields

Proposition 3.10. D^L is the minimal matrix such that $C^L = (D^L)^*$.

Definition 3.11. We call *écorché* of $\mathcal{U}_L$ the automaton:
$\mathcal{E}_L = \langle \mathcal{F}_L, A, D^L \cup H^L, \{(X_s, L)\}, \{(L, Y_e)\} \rangle$.

The automaton $\mathcal{U}_L$ is then obtained from $\mathcal{E}_L$ by backward *and* forward closure of the spontaneous transitions. In the sequel, we rather draw écorchés instead of universal automata, because they have less transitions and it is often easier to understand the structure of the universal automaton on the écorché.

Example 3.12. The factorisations of $L_1 = A^*abA^*$ are totally ordered

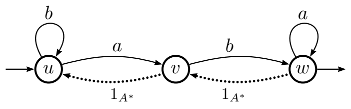
$$u = (A^*, L_1) \geqslant v = (A^*aA^*, A^*bA^*) \geqslant w = (L_1, A^*),$$

and so are the factorisations of $L_2 = aA^*$:

$$(aA^*, A^*) \leqslant (1 + aA^*, aA^*) \leqslant (A^*, \varnothing).$$

Figure 7 shows the écorché of the universal automata of these two languages.

In the case of pure-group languages, that is, languages whose syntactic monoid is a group, the écorché of the universal automaton has a very special form. The states of the strongly connected components are the pairwise uncomparable factorisations. The non spontaneous transitions, that is,



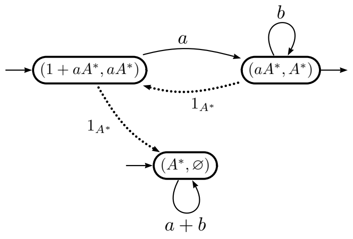


FIGURE 7. The écorché of $\mathcal{U}_{L_1}$ (left) and $\mathcal{U}_{L_2}$ (right)

the transitions described by the matrix H^L, are all the transitions in these strongly connected components whereas the spontaneous transitions put an order on the strongly connected components and the écorché is thus decomposed into levels. Figure 8 shows the écorché of $\mathcal{U}_{L_3}$. A more complicated écorché for a pure group language is shown at Figure 15, where the levels appear even more clearly. We shall characterise them at Subsection 7.2.

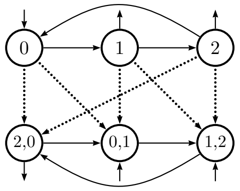


FIGURE 8. Ecorché of the universal automaton of $\{1, 2\}$ in $\mathbb{Z}/3\mathbb{Z}$

4 Construction of the universal automaton

The universal automaton has been defined, and then described. From what we have already seen, it follows immediately that the universal automaton of a regular language is effectively computable. We present now an algorithm [18], somehow optimal, which performs the task. From this construction of $\mathcal{U}_L$ we then derive an effective description of the *(left) canonical morphism* from any automaton $\mathcal{B}$ which accepts L into $\mathcal{U}_L$. An example of a method for finding a small NFA accepting a given language is described in the last subsection.

4.1 Computation of the factorisations

As above, let L be a language of A^*. The key for the construction of $\mathcal{U}_L$ is the computation of the factorisations of L. From Proposition 2.1, every right factor of a language is an intersection of left quotients of this language. As the quotients of the languages are the futures of the states of any deterministic automaton $\mathcal{A}$ that accepts the language, for every factorization (X_i, Y_i) of the language L, there exists a subset P of states of $\mathcal{A}$ such that $Y_i = \bigcap_{p \in P} \mathsf{Fut}_{\mathcal{A}}(p)$. But this subset may be not unique. The set of subsets P such that the intersection of the futures of states in P is equal to Y_i is closed under union, thus there exists a unique maximal P_i such that $Y_i = \bigcap_{p \in P_i} \mathsf{Fut}_{\mathcal{A}}(p)$. To get an efficient representation of factorisations, we have to compute these maximal subsets corresponding to factorisations.

Let $\mathcal{A} = \langle Q, A, \delta, i, T \rangle$ be a *complete accessible deterministic* automaton that accepts L. Let $\mathcal{Q}_{\mathcal{A}}$ be the set of states of $\mathcal{A}_{\mathsf{cod}}$, the co-determinisation of $\mathcal{A}$; $\mathcal{Q}_{\mathcal{A}}$ is a subset of $\mathfrak{P}(Q)$, *i.e.* an element of $\mathfrak{P}(\mathfrak{P}(Q))$. We denote by $\mathcal{I}_{\mathcal{A}}$ the closure under intersection of $\mathcal{Q}_{\mathcal{A}}$. Notice that $\mathcal{I}_{\mathcal{A}}$ always contains Q itself (as the intersection of an *empty* set of elements of $\mathcal{Q}_{\mathcal{A}}$).

Theorem 4.1. The mapping $\psi_{\mathcal{A}}$ from $\mathcal{I}_{\mathcal{A}}$ into $\mathcal{F}_L$ defined by:

$$\begin{aligned} \psi_{\mathcal{A}} \colon \mathcal{I}_{\mathcal{A}} &\longrightarrow \mathcal{F}_L \\ P &\longmapsto (X, Y), \text{ with } Y = \bigcap_{p \in P} \mathsf{Fut}_{\mathcal{A}}(p) \end{aligned}$$

is a bijection.

Proof. As every intersection of left quotient is a right factor of the language, this mapping is well defined. In order to prove that this is a bijection, we prove that

$$\begin{aligned} \mathcal{F}_L &\longrightarrow \mathfrak{P}(Q) \\ (X, Y) &\longmapsto \{p \mid Y \subseteq \mathsf{Fut}_{\mathcal{A}}(p)\} \end{aligned}$$

is a mapping from $\mathcal{F}_L$ onto $\mathcal{I}_{\mathcal{A}}$. Let (X, Y) be a factorisation and $P = \{p \mid Y \subseteq \mathsf{Fut}_{\mathcal{A}}(p)\}$. Let $\mathcal{A}_{\mathsf{cod}} = \langle \mathcal{Q}_{\mathcal{A}}, A, H, J, t \rangle$ be the co-determinisation of $\mathcal{A}$ and let R be the set of states of $\mathcal{A}_{\mathsf{cod}}$ that contain P.

By construction of the co-determinisation, for every state s in $\mathcal{A}_{\mathsf{cod}}$ and for every word u in $\mathsf{Fut}_{\mathcal{A}_{\mathsf{cod}}}(s)$, it holds: $s = \{p \mid u \in \mathsf{Fut}_{\mathcal{A}}(p)\}$. Hence R is the set of states of $\mathcal{A}_{\mathsf{cod}}$ whose future has a non empty intersection with Y. Moreover, $Y = \bigcup_{s \in R} \mathsf{Fut}_{\mathcal{A}_{\mathsf{cod}}}(s)$. Hence, a state p of $\mathcal{A}$ belongs to every state of R if and only if its future contains Y. Thus, $P = \bigcap_{s \in R} s \in \mathcal{I}_{\mathcal{A}}$. Q.E.D.

Remark 4.2. This construction can take any deterministic automaton as input and gives the same result. Indeed, when a deterministic automaton is co-determinised, states that are Nerode-equivalent (*i.e.* that would be merged by a minimisation algorithm) appear exactly in the same states of the co-determinisation. They become indissociable and the set $\mathcal{I}_{\mathcal{A}}$ actually does not depend on the input, but only on the language L.

Remark 4.3. The order on factorisations is realised on $\mathcal{I}_\mathcal{A}$ by the inclusion order.

Proposition 4.4. Let $\mathcal{A} = \langle Q, A, \delta, i, T\rangle$ be a complete deterministic automaton. Let P in $\mathcal{I}_\mathcal{A}$ and $(X, Y) = \psi_\mathcal{A}(P)$. Then

$$X = \bigcup_{p\in P} \mathsf{Past}_\mathcal{A}(p) \quad \text{and} \quad P = i \triangleright X.$$

Proof. Let (X, Y) be a factorisation; $X = \{u \mid uY \subseteq L\} = \{u \mid Y \subseteq u^{-1}L\}$. For every word u in X, let $p = i \triangleright u$; as $\mathcal{A}$ is deterministic, $u^{-1}L = \mathsf{Fut}_\mathcal{A}(p)$. Hence, p is in P; therefore, $X \subseteq \bigcup_{p\in P} \mathsf{Past}_\mathcal{A}(p)$. Conversely, let v be a word in the past of some state p in P. It holds $v\mathsf{Fut}_\mathcal{A}(p) \subseteq L$ and $Y \subseteq \mathsf{Fut}_\mathcal{A}(p)$, hence v is in X. Q.E.D.

We have thus characterized the factorisations of the language, that is the states of the universal automaton. We can now give a construction for the universal automaton.

Proposition 4.5. Let $\mathcal{A} = \langle Q, A, \delta, i, T\rangle$ be a complete deterministic automaton that accepts L. The automaton $\langle \mathcal{I}_\mathcal{A}, A, D, J, U\rangle$ defined by:

$$D = \{(P, a, S) \in \mathcal{I}_\mathcal{A} \times A \times \mathcal{I}_\mathcal{A} \mid P \triangleright a \subseteq S\}\,, \tag{1.9}$$

$$J = \{P \in \mathcal{I}_\mathcal{A} \mid i \in P\}\,, \quad U = \{P \in \mathcal{I}_\mathcal{A} \mid P \subseteq T\}, \tag{1.10}$$

is isomorphic to the universal automaton of L: $\mathcal{U}_L = \langle \mathcal{F}_L, A, E^L, I^L, T^L\rangle$.

Proof. Theorem 4.1 defines a bijection from $\mathcal{I}_\mathcal{A}$ onto $\mathcal{F}_L$. We have to check that the definitions of D, J and U correspond to E^L, I^L and T^L of Definition 2.4. Let (X_P, Y_P) and (X_S, Y_S) the factorisations corresponding to P and S:

$$Y_P = \bigcap_{p\in P} \mathsf{Fut}_\mathcal{A}(p), \qquad Y_S = \bigcap_{p\in S} \mathsf{Fut}_\mathcal{A}(p).$$

We have

$$\begin{aligned} P \triangleright a \subseteq S \quad &\Longleftrightarrow \quad Y_S \subseteq \bigcap_{p\in P\triangleright a} \mathsf{Fut}_\mathcal{A}(p) \\ &\Longleftrightarrow \quad aY_S \subseteq \bigcap_{p\in P} \mathsf{Fut}_\mathcal{A}(p) = Y_P \quad \Longleftrightarrow \quad X_P a Y_S \subseteq L. \end{aligned}$$

$$\begin{aligned} \psi_\mathcal{A}(J) =& \{(X, Y) \in \mathcal{F}_L \mid Y \subseteq L\} \\ =& \{(X, Y) \in \mathcal{F}_L \mid 1_{A^*} \in X\}. \end{aligned}$$

$$\psi_{\mathcal{A}}(U) = \{(X,Y) \in \mathcal{F}_L \mid 1_{A^*} \in Y\}.$$

Q.E.D.

Remark 4.6. Once $\mathcal{I}_{\mathcal{A}}$ is computed, the construction of $\mathcal{U}_L$ goes as follow: $\mathcal{I}_{\mathcal{A}}$ is the set of states; for every P in $\mathcal{I}_{\mathcal{A}}$, if i is in P, make P initial, if P is a subset of T, make P final; for every letter a, compute $P \triangleright a$, and for every R in $\mathcal{I}_{\mathcal{A}}$ that contains $P \triangleright a$, add a transition (P, a, R).

4.2 Computation of the canonical morphism

If the universal automaton is computed from a complete deterministic accessible automaton $\mathcal{A}$, the left canonical morphism from any equivalent automaton $\mathcal{B}$ into the universal automaton can be computed in polynomial time.

Let $\mathcal{P}$ be the accessible part of the product of $\mathcal{A}$ by $\mathcal{B}$. Every state of $\mathcal{P}$ is a pair (p,q) of a state of $\mathcal{A}$ and a state of $\mathcal{B}$. Let R_q be the set of states p of $\mathcal{A}$ such that (p,q) is a state of $\mathcal{P}$. We define an application $\varphi_{\mathcal{B}}$ from the states of $\mathcal{B}$ into $\mathcal{I}_{\mathcal{A}}$: $\varphi_{\mathcal{B}}(q)$ is the smallest element of $\mathcal{I}_{\mathcal{A}}$ which contains R_q.

Proposition 4.7. The morphism from $\mathcal{B}$ into $\mathcal{U}_L$ induced by $\varphi_{\mathcal{B}}$ is the left canonical morphism.

Proof. Let r be a state of $\mathcal{B}$. It holds $R_q = i \underset{\mathcal{A}}{\triangleright} \mathsf{Past}_{\mathcal{B}}(q)$. Let $Y = \bigcap_{p \in R_q} \mathsf{Fut}_{\mathcal{A}}(p)$. As $\mathcal{A}$ is deterministic, the futures of its states are quotients and thus Y is a right factor. We show that this is the largest right factor such that $[\mathsf{Past}_{\mathcal{B}}(q)]\,[Y] \subseteq L$.

$$\mathsf{Past}_{\mathcal{B}}(q) = \bigcup_{p \in R_q} \mathsf{Past}_{\mathcal{P}}((p,q)) \subseteq \bigcup_{p \in R_q} \mathsf{Past}_{\mathcal{A}}(p)$$

As $\left[\bigcup_{p \in R_q} \mathsf{Past}_{\mathcal{A}}(p)\right] \left[\bigcap_{p \in R_q} \mathsf{Fut}_{\mathcal{A}}(p)\right] \subseteq L$, $[\mathsf{Past}_{\mathcal{B}}(q)]\,[Y] \subseteq L$. Let v be a word which is not in Y. There exists a state p in R_q such that v is not in $\mathsf{Fut}_{\mathcal{A}}(p)$ and there exists a word u in $\mathsf{Past}_{\mathcal{B}}(q)$ such that $p = i \underset{\mathcal{A}}{\triangleright} u$. We have $v \notin u^{-1}L$ and $uv \notin L$. This proves that Y is maximal.

We show now that Y is the right factor of $\psi_{\mathcal{A}}(\varphi_{\mathcal{B}}(q))$. As $\varphi_{\mathcal{B}}(q)$ is the smallest element of $\mathcal{I}_{\mathcal{A}}$ which contains R_q, they correspond to the same right factor, *i.e.* $Y = \bigcap_{p \in \varphi_{\mathcal{B}}(q)} \mathsf{Fut}_{\mathcal{A}}(p)$. Q.E.D.

4.3 Searching for NFA of minimal size

It is known that the computation of a NFA with minimal size from the minimal automaton of a language is a PSPACE-complete problem [13]. However, the universal automaton is a good framework to explain exact algorithms or to describe heuristics that give approximate solutions.

First, the universal automaton of a language contains any equivalent NFA with minimal size, since the canonical morphism from this NFA into the universal automaton is injective.

Then an exact algorithm would consist in enumerating all subautomata of the universal automaton (starting with the smallest) and testing if they accept every word of the language.

This fact is the base of many heuristics. There exist several conditions on subautomata of the universal automaton built as in Proposition 4.5. Each of these conditions is either necessary or sufficient for the subautomaton accepts the language. In [25], Polák has made a comparison between a large set of these conditions. They all give tractable algorithms that compute NFA accepting the language, hopefully small, but not necessarily of minimal size.

The first authors that give such a condition are Kameda and Weiner in [14]. They build a table, whose rows are indexed by the states of the minimal automaton and the columns by the states of its co-determinisation, and read factorisations in this table. They define a property of *cover*, that guarantees that a set of factorisations corresponds to an automaton (actually a subautomaton of the universal automaton, even if they do not define it), that accepts the language.

Along the same line of work, Matz and Potthoff [22] have defined another automaton, which they call *fundamental automaton* and which contains, strictly in some cases, the universal automaton. They then give a condition that guarantees that a subautomaton of the fundamental automaton accepts the language. We present here a condition that is inspired by this one and which is an example of a heuristic that search for small NFA.

Proposition 4.8. Let $\mathcal{A} = \langle Q, A, \delta, i, T \rangle$ be a deterministic complete automaton and let $\mathcal{U}_L$ be the universal automaton built from $\mathcal{A}$. Let R be a subset of $\mathcal{I}_\mathcal{A}$ such that:

(i) $\bigcup_{P \in R, P \subseteq T} P = T$;

(ii) for every P in R, for every letter a, for every q in Q such that $q \triangleright a = p$, there exists S in R, such that q is in S and $S \triangleright a \subseteq P$.

Then the subautomaton of $\mathcal{U}_L$ with set of states R accepts the language.

Proof. Let u be a word of L. Let $p_0 = i, p_1, \ldots, p_k$ the states of the computation in $\mathcal{A}$ labeled by u. The state p_k is final, hence there exists P_k final in R that contains p_k. If there exists P_i that contains p_i, as $p_i = p_{i-1} \triangleright u_i$, there exists P_{i-1} in R that contains p_{i-1} such that there is a transition from P_{i-1} to P_i labeled by a. Hence, by induction, the word u is accepted by the subautomaton. Q.E.D.

5 Size of the universal automaton

It follows from the construction of the universal automaton that $\mathcal{U}_L$ has at most 2^n states if the size of the minimal deterministic automaton is n. The computation for the language $\{w \mid |w| \neq 0 \mod n\}$ shows that this bound is tight (*cf.* Section 7.2 below and also [9]).

As the determinisation of a non deterministic n-state automaton may give at most a 2^n-state automaton, we immediately get a 2^{2^n} upper bound for the size of the universal automaton with respect to the minimal non deterministic automaton that accepts the language.

This bound is not tight, for the worst cases in determinisation and in the construction of the universal automaton cannot occur in a row. We give here the proof that the tight bound is given by the Dedekind numbers and also that, in the case of a unary alphabet, the conjunction of worst cases may occur, but the determinisation does not yield a 2^n blow-up then.

5.1 Bounds for the universal automaton

The nth Dedekind number $D(n)$ is defined as the number of monotonous Boolean functions with n variables. Since such a function is characterised by a Boolean expression in disjunctive normal form whose clauses are pairwise uncomparable, it is also the number of antichains of $\mathfrak{P}([1;n])$ (ordered by inclusion).

Theorem 5.1 (Lombardy, [19])**.** Let $\mathcal{A}$ be an NFA with n states that accepts a language L. Then:

(i) $\mathcal{U}_L$ has at most $D(n)$ states;

(ii) the trim part of $\mathcal{U}_L$ has at most $D(n) - 2$ states.

For every integer n, there exist automata with n states for which these bounds are reached.

Remark 5.2. There is no closed form expression for $D(n)$, and its exact value is only known for n smaller than 9 (*cf.* [28]). However, Korshunov [16] has given an approximate expression of $D(n)$. For instance, if n is even,

$$D(n) \sim 2^{\binom{n}{n/2}} \exp\left(\binom{n}{n/2-1}\left(2^{-n/2} + n^2 2^{-n-5} - n 2^{-n-4}\right)\right).$$

Figure 9 gives a visual comparison between $D(n)$ and the double exponential function $n \mapsto 2^{2^n}$.

Definition 5.3. Let O be an ordered set. An *upset* $\mathcal{V}$ of O is an upperly closed subset of O:

$$\forall x \in \mathcal{V},\ \forall y \in O,\ x \leqslant y \Longrightarrow y \in \mathcal{V}. \qquad (1.11)$$

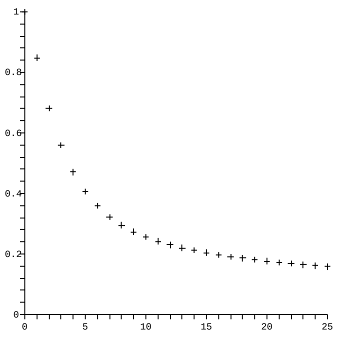


FIGURE 9. The graph of $\frac{\log_2 D(n)}{2^n}$

Notice that an upset may be empty and may also be equal to O itself. If Q is a set, $\mathfrak{P}(Q)$ or every subset of $\mathfrak{P}(Q)$ is naturally ordered by inclusion. Upsets of $\mathfrak{P}(Q)$ are naturally in bijection with *antichains* by taking their *minimal* elements.

We now use the construction given in the previous section. As we start with a non deterministic automaton, we first determinize it to obtain an automaton $\mathcal{D}$ that is used to build the universal automaton.

Proposition 5.4. Let $\mathcal{A} = \langle Q, A, E, I, T\rangle$ be a non deterministic automaton. Let $\mathcal{D} = \langle R, A, F, \{I\}, U\rangle$ be the determinisation of $\mathcal{A}$ and $\mathcal{C} = \langle S, A, G, K, \{U\}\rangle$ the co-determinisation of $\mathcal{D}$. Every element of S is an upset of R.

Proof. Let X and Y be two states of $\mathcal{D}$ such that $X \subseteq Y$. It holds $\mathsf{Fut}_{\mathcal{D}}(X) = \bigcup_{p\in X} \mathsf{Fut}_{\mathcal{A}}(p) \subseteq \bigcup_{p\in Y} \mathsf{Fut}_{\mathcal{A}}(p) = \mathsf{Fut}_{\mathcal{D}}(Y)$. Let P be a state of $\mathcal{C}$ which contains X. For every v in $\mathsf{Fut}_{\mathcal{C}}(P)$, $P = v \underset{\mathcal{D}}{\lhd} U$. As X is in P, v is in $\mathsf{Fut}_{\mathcal{D}}(X)$, thus in $\mathsf{Fut}_{\mathcal{D}}(Y)$. Hence, Y is in P. Q.E.D.

Proposition 5.5. Let $\mathcal{A} = \langle Q, A, E, I, T\rangle$ be a non deterministic automaton that recognizes a language L. The universal automaton of L has at most $D(\mathsf{card}(Q))$ states, where $D(n)$ it the nth Dedekind number.

Proof. Let $n = \mathsf{card}(Q)$ and let $\mathcal{D}$ be the determinisation of $\mathcal{A}$. As the intersection of two upsets is an upset, the elements of $\mathcal{I}_{\mathcal{D}}$ are upsets of $\mathfrak{P}(Q)$, and $D(n)$ is equal to the number of upsets of $\mathfrak{P}(Q)$. Q.E.D.

Proposition 5.6. Let $\mathcal{A} = \langle Q, A, E, I, T\rangle$ be an NFA that recognizes a language L. The number of states of the trim universal automaton of L is bounded by $D(\mathsf{card}(Q)) - 2$.

Proof. Actually, if a state corresponds to the empty upset, it has an empty past and it is therefore not accessible. Likewise, if a state corresponds to the upset $\{\varnothing\}$, it has an empty future and it is therefore not co-accessible.

Q.E.D.

The first part of Theorem 5.1 is thus established.

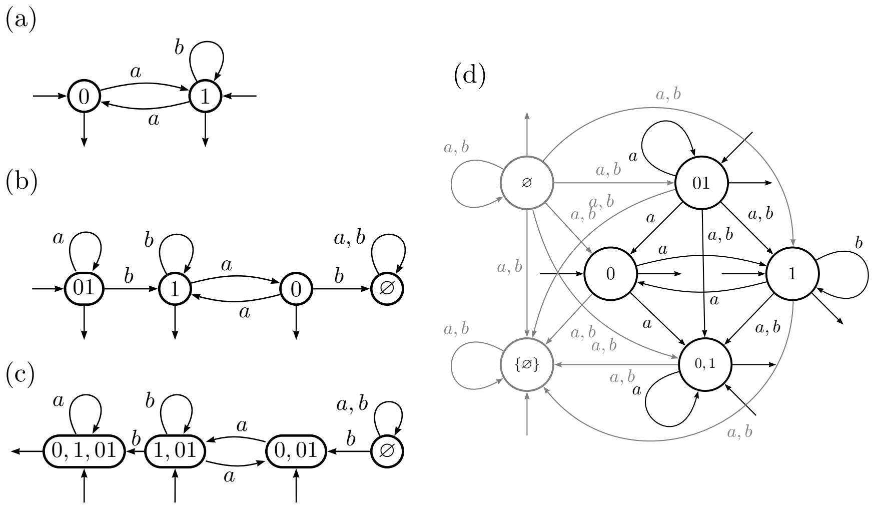


FIGURE 10. The construction of the universal automaton from $\mathcal{Z}_2$

Example 5.7. We give here an example for the construction of the universal automaton. Let $\mathcal{Z}_2$ be the automaton of Figure 10(a). Let $\mathcal{D}_2$ be the determinized automaton of $\mathcal{Z}_2$, drawn on Figure 10(b). Each of its states is a subset of the set of states of $\mathcal{Z}_2$. We denote this set by a word whose letters are the elements of the state: the word 01 stands for the set $\{0, 1\}$. The states of the universal automaton (Figure 10(c)) are upsets of the power set of states of $\mathcal{Z}_2$. We denote an upset by the setr of its minimal elements. For instance 0, 1 means $\{\{1\}, \{2\}, \{0, 1\}\}$. Notice that $\varnothing$ is the empty upset, whereas $\{\varnothing\}$ is the upset with $\varnothing$ as minimal element, *i.e.* the power set itself. The non accessible part of the universal automaton is drawn in gray. The automaton $\mathcal{Z}_2$ is an example of the worst case in the construction of the universal automaton. Indeed, $D(2) = 6$.

Likewise, $D(3) = 20$ and we give a three-state automaton which recognizes a language whose universal automaton has twenty states: the automaton $\mathcal{Z}_3$ shown on Figure 11(a). As the number of transitions of the universal automaton is to high to allow to draw them all, the more compact representation given by the écorché is drawn on Figure 11(c).

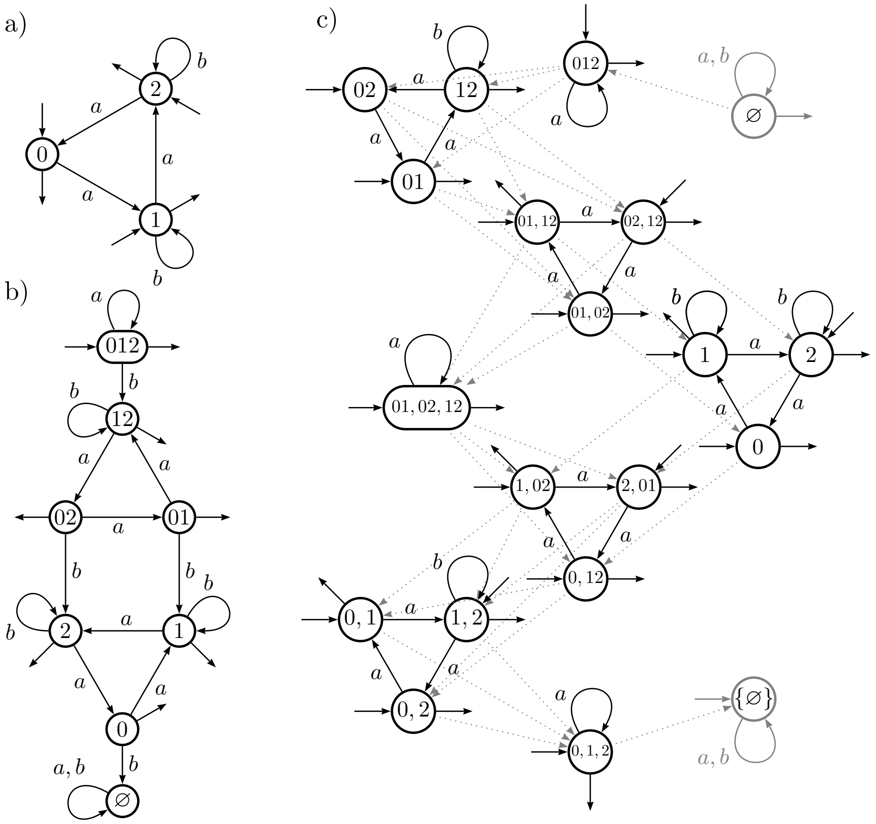


FIGURE 11. The construction of the universal automaton from $\mathcal{Z}_3$

In the following section, we generalise this example to show that, for every n, there exists a n-state NFA that accepts a language whose universal automaton has $D(n)$ states.

5.2 Reaching the bounds of the universal automata

As announced, we introduce first a notation for the dual action induced by an automaton.

Definition 5.8. Let $\mathcal{A} = \langle Q, A, E, I, T\rangle$ be an automaton. The *set of predecessors* of a state p of $\mathcal{A}$ by a letter a is $a \underset{\mathcal{A}}{\triangleleft} p = \{q \in Q \mid (q,a,p) \in E\}$, denoted $a \triangleleft p$ if there is no ambiguity. This defines a *left action* of A^* on $\mathfrak{P}(Q)$: for every letter a, every word w, and every subset X of Q, we have:

$$a \underset{\mathcal{A}}{\triangleleft} X = \bigcup_{p\in X} a \underset{\mathcal{A}}{\triangleleft} p\,, \quad 1_{A^*} \underset{\mathcal{A}}{\triangleleft} X = X\,, \quad aw \underset{\mathcal{A}}{\triangleleft} X = a \underset{\mathcal{A}}{\triangleleft} (w \underset{\mathcal{A}}{\triangleleft} X).$$

Obviously, q is in $p \underset{\mathcal{A}}{\triangleright} w$ if and only if p is in $w \underset{\mathcal{A}}{\triangleleft} q$. In the sequel, for every positive integer n, $\mathcal{Z}_n = \langle Q, A, E, I, T\rangle$ is the automaton defined by:

$$\begin{gathered} Q = \mathbb{Z}/n\mathbb{Z}; \qquad A = \{a, b\}; \qquad I = T = Q; \\ E = \{(p, a, p+1) \mid p \in Q\} \cup \{(p, b, p) \mid p \in Q \setminus \{0\}\}. \end{gathered} \tag{1.12}$$

In the sequel, if X is a subset of Q, *i.e.* a subset of $\mathbb{Z}/n\mathbb{Z}$, for every integer k, we denote $X + k = \{x + k \mid x \in X\}$.

Lemma 5.9. Let n be a positive integer. The determinisation of $\mathcal{Z}_n$ is $\mathcal{D}_n = \langle \mathfrak{P}(Q), A, F, \{Q\}, \mathfrak{P}(Q)\setminus\{\varnothing\}\rangle$, with:

$$F = \{(X, a, X+1), (X, b, X\setminus\{0\}) \mid X \subseteq Q\}. \tag{1.13}$$

Proof. As every state of $\mathcal{A}$ is initial, the initial state of $\mathcal{D}$ is Q. As every state of $\mathcal{A}$ is terminal, every state of $\mathcal{D}$ different from $\varnothing$ is terminal.
If X is a subset of Q, $X \underset{\mathcal{A}}{\triangleright} a = \bigcup_{p\in X} p \underset{\mathcal{A}}{\triangleright} a = \bigcup_{p\in X} p + 1 = X + 1$; likewise, $X \underset{\mathcal{A}}{\triangleright} b = \bigcup_{p\in X, p\neq 0} p = X\setminus\{0\}$. This gives the set of transitions F of $\mathcal{D}$.
We show that every element of $\mathfrak{P}(Q)$ is an accessible state by induction on the number of elements. The set Q itself is the initial state of $\mathcal{D}$. Let assume that X is an accessible state. Let x be an element of X, we show that $X\setminus\{x\}$ is accessible. Actually, $X \triangleright a^{n-x}ba^x = (X - x) \triangleright ba^x = ((X - x)\setminus\{0\}) \triangleright a^x = ((X - x)\setminus\{0\}) + x = X\setminus\{x\}$. Therefore, every element of $\mathfrak{P}(Q)$ is accessible. Q.E.D.

For every subset X of Q, we denote $\underline{X} = \{Y \mid Y \subseteq X\}$, and $(\underline{X})^c = \mathfrak{P}(Q)\setminus\underline{X}$; we can notice that $(\underline{X})^c$ is an upset of $\mathfrak{P}(Q)$.

Lemma 5.10. Let n be a positive integer. The co-determinisation of $\mathcal{D}_n$ is $\mathcal{C}_n = \langle S, A, G, K, V\rangle$, with:

$$\begin{gathered} S = \{(\underline{X})^c \mid X \in \mathfrak{P}(Q)\}; \qquad K = S\setminus\{\varnothing\}; \qquad V = \{(\underline{\varnothing})^c\} \\ G = \{(\underline{X})^c, a, (\underline{Y})^c \mid (X, a, Y) \in F\} \cup \{\left(\underline{X \cup \{0\}}\right)^c, b, (\underline{X})^c \mid X \subseteq Q\}. \end{gathered} \tag{1.14}$$

Proof. As any state X of $\mathcal{D}_n$ different from $\varnothing$ is final, the state $t = (\underline{\varnothing})^c$ is the final state of $\mathcal{D}_n$. First, we show by induction on the word w that any state $P = w \triangleleft t$ is in S. This is obviously true if w is the empty word: $P = t$. If $P = (\underline{X})^c$ is in S, so its predecessors are:

$$\begin{aligned} a \underset{\mathcal{C}}{\triangleleft} (\underline{X})^c =& \{a \underset{\mathcal{D}}{\triangleleft} Y \mid Y \not\subseteq X\} = \{Y \mid Y \not\subseteq a \underset{\mathcal{D}}{\triangleleft} X\} = \left(\underline{a \underset{\mathcal{D}}{\triangleleft} X}\right)^c \\ =& (\underline{X-1})^c; \end{aligned} \tag{1.15}$$

$$\begin{aligned} b \underset{\mathcal{C}}{\triangleleft} (\underline{X})^c =& \{b \underset{\mathcal{D}}{\triangleleft} Y \mid Y \not\subseteq X\} \\ =& \{Y \mid Y \not\subseteq X, 0 \notin Y\} \cup \{Y \cup \{0\} \mid Y \not\subseteq X, 0 \notin Y\} \\ =& \{Y \mid Y \not\subseteq X \cup \{0\}, 0 \notin Y\} \cup \{Y' \mid Y' \not\subseteq X \cup \{0\}, 0 \in Y'\} \\ =& \{Y \mid Y \not\subseteq X \cup \{0\}\} = \left(\underline{X \cup \{0\}}\right)^c . \end{aligned} \tag{1.16}$$

We show that every element $P = (\underline{X})^c$ of S is co-accessible from t.If $X = \varnothing$, then $P = t$. If $P = (\underline{X})^c$ is co-accessible, for any x in Q, $P' = \left(\underline{X \cup \{x\}}\right)^c$ is co-accessible too:

$$a^{n-x}ba^x \underset{\mathcal{C}}{\triangleleft} P = a^{n-x}b \underset{\mathcal{C}}{\triangleleft} (\underline{X-x})^c = a^{n-x} \underset{\mathcal{C}}{\triangleleft} \left(\underline{(X-x) \cup \{0\}}\right)^c = \left(\underline{X \cup \{x\}}\right)^c . \tag{1.17}$$

Therefore the set of states of $\mathcal{C}_n$ is exactly S. Q.E.D.

Lemma 5.11. Let Q be a finite set. The intersection closure of $\{(\underline{X})^c \mid X \in \mathfrak{P}(Q)\}$ is exactly the set of upsets of $\mathfrak{P}(Q)$.

Proof. Let $\mathcal{U}$ be an upset of $\mathfrak{P}(Q)$. For every Y in $\mathcal{U}$, for every X not in $\mathcal{U}$, $Y \not\subseteq X$. Hence, Y is in $(\underline{X})^c$ and $\mathcal{U}$ is a subset of $(\underline{X})^c$. Thus, as X is not in $(\underline{X})^c$, it comes $\mathcal{U} = \bigcap_{X \notin \mathcal{U}} (\underline{X})^c$. Q.E.D.

With this last lemma, the proof of Theorem 5.1 is now complete.

In the case of a one-letter alphabet, the determinisation algorithm is known not to be exponential. Indeed, if $\mathcal{A}$ is a one-letter NFA with n states, the determinisation of $\mathcal{A}$ (and the minimal automaton of the accepted language) has at most $G(n)$ states (*cf.* [3]), where $G(n)$ is the Landau function of n, that is, the maximal least common multiple of a set of integers with sum equal to n. We show that in this case, the obvious upper bound $2^{G(n)}$ for the size of the universal automaton is tight.

Proposition 5.12. For every integer n, there exist automata with n states over a one-letter alphabet such that, if L is the accepted language:

(i) $\mathcal{U}_L$ has $2^{G(n)}$ states;

(ii) the trim part of $\mathcal{U}_L$ has $2^{G(n)} - 2$ states.

Proof. There exist an integer r and r numbers $k_1,..,k_r$ such that $k_1+..+k_r = n$ and $\mathsf{lcm}(k_1,k_2,...,k_r) = G(n)$. Let Q be the disjoint union of all the $(Q_i = \mathbb{Z}/k_i\mathbb{Z})$ for i in $[1;r]$ and let $\mathcal{Y}_n = \langle Q, \{a\}, E, I, T\rangle$ be the automaton defined by:

$$I = \{0 \in Q_i \mid i \in [1;r]\}, \quad T = Q\smallsetminus I, \quad E = \{(p,a,p+1) \mid \exists i, p \in Q_i\}.$$

The determinisation of $\mathcal{Y}_n$ is the automaton $\mathcal{D}_n = \langle R, \{a\}, F, J, U\rangle$, with $R = \mathbb{Z}/G(n)\mathbb{Z}$, $J = \{0\}$, $U = R\smallsetminus J$ and $F = \{(p,a,p+1) \mid p \in R\}$.

The states of the co-determinisation of $\mathcal{D}_n$ are all the subset of R with $\mathsf{card}(R)-1$ elements. The intersection closure of this set of states is equal to $\mathfrak{P}(R)$. Hence, the universal automaton of the language recognized by $\mathcal{Y}_n$ has $2^{G(n)}$ states and the trim universal automaton has $2^{G(n)}-2$ states. Q.E.D.

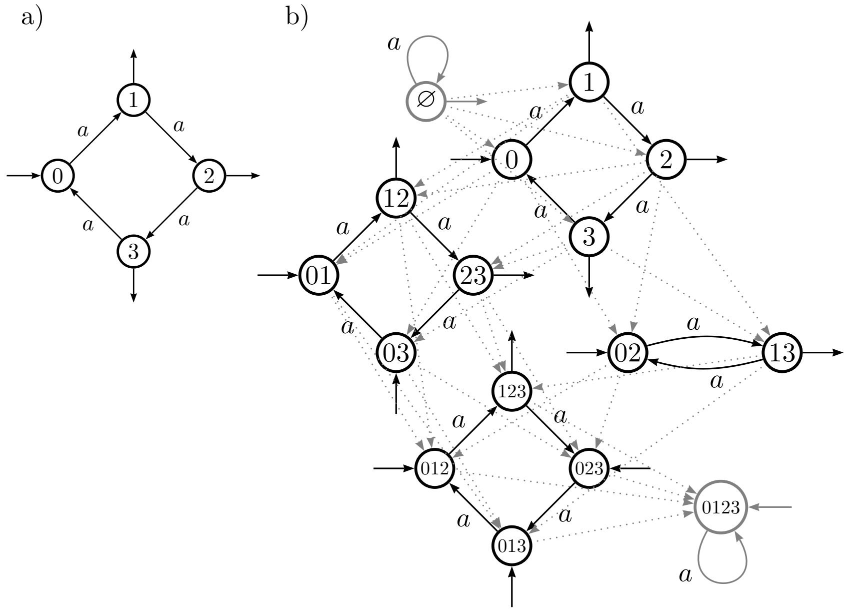


FIGURE 12. The automaton $\mathcal{Y}_4$ and its universal automaton

Remark 5.13. Starting from a one-letter DFA with n states ($n > 1$), it is not possible to obtain a trim universal automaton with $2^n - 1$ states. The state corresponding to the empty set in the construction of Theorem 4.1 cannot be accessible. If the full set corresponds to a co-accessible state, it means that every state of the DFA is final, thus every word is accepted and the universal automaton has one state, or, if the DFA is not complete, the language is a finite prefix language and the universal automaton has n states. Therefore, the trim universal automaton has at most $2^n - 2$ states.

Example 5.14. Let $\mathcal{Y}_4$ be the automaton of Figure 12 a). It is equal to $\mathcal{D}_4$ (actually, $G(4) = 4$). The universal automaton, drawn on Figure 12 b), has $2^4 = 16$ states, including a non accessible state and a non co-accessible state.

6 Equations in the universal automaton

John H. Conway who gave, in his own language and terminology [5], another definition of the universal automaton, was not at all interested in the computation of small NFA's for a regular language. He used the *factor matrix* of a language to solve two dual classes of problems. First, in the *approximation problem*, are given on one hand a language L in A^* and on the other hand a family $\mathcal{K} = \{K_1, \ldots, K_n\}$, all in A^*. The latter determines a *substitution* σ from X^* into A^*, with $X = \{x_1, \ldots, x_n\}$ and $\sigma(x_i) = K_i$. The construction of the universal automaton of L allows us to show that the set W of words w in X^* such that $\sigma(w)$ is contained in L is regular when L is regular (and without any hypohesis on the K_i's).

The dual problem is the one of (in)equations. The regular languages L in A^* and K in X^* being given, the universal automaton of L allows the effective computation of all maximal n-tuples of languages $\{H_1, \ldots, H_n\}$ such that $\sigma(K)$ is contained in L.

6.1 The approximation problem

The construction of the automaton $\mathcal{U}_L$ can be seen as a special case of an approximation problem: the reasoning that proves that $\mathcal{U}_L$ accepts L can be generalised to other families of subsets than the generating set of A^*, with remarkable results.

Let L be a language of A^* and $\mathcal{K} = \{K_1, \ldots, K_n\}$ a family of n languages of A^*. We set $X = \{x_1, x_2, \ldots, x_n\}$ an n-letter alphabet and $\sigma\colon B^* \to A^*$ the *substitution* defined by

$$\forall i \in [1; n] \qquad \sigma(x_i) = K_i .$$

The sole consideration of the syntactic morphism allows us to show that the language W over B^*,

$$W = \{f \in B^* \mid \sigma(f) \subseteq L\} ,$$

is recognisable if L is recognisable and without any assumption on the K_i's — a corollary of a result in [26], see [27]. But here we prove the result and give it a more precise interpretation using the universal automaton.

To simplify the statements, with K_i and hence σ being fixed, we write $\breve{\sigma}$ for the map from $\mathfrak{P}(A^*)$ to $\mathfrak{P}(B^*)$ defined by

$$\forall L \in \mathfrak{P}(A^*) \qquad \breve{\sigma}(L) = \{f \in B^* \mid \sigma(f) \subseteq L\}\,, \tag{1.18}$$

that is, $\breve{\sigma}(L)$ is equal to the language W defined above. The map $\breve{\sigma}$ acts as the *inverse* of the substitution σ but retains only those words whose image under σ is *contained* in L. In other words, $\sigma(\breve{\sigma}(L))$ is the *best possible approximation* (by default) to L as a *sum of products* of languages K_i and $\breve{\sigma}(L)$ describes how this approximation is constructed.

Let L be a language of A^*, $\mathcal{U}_L$ its universal automaton and F^L its factor matrix. We write $\mathcal{S}_L^{\mathcal{K}}$ for the automaton over X^* obtained from $\mathcal{U}_L$ by replacing each label $E_{X,Y'}^L$ by the set of letters in X whose image under σ is contained in $F_{X,Y'}^L$:

$$\forall (X,Y),(X',Y') \in \mathcal{F}_L \qquad (X,Y) \xrightarrow[\mathcal{S}_L^{\mathcal{K}}]{x} (X',Y') \Longleftrightarrow \sigma(x) \subseteq F_{X,Y'}^L.$$

Theorem 6.1 (Conway [5]). $\qquad \breve{\sigma}(L) = |\mathcal{S}_L^{\mathcal{K}}|$.

Proof. The proof goes by induction on the length of f and amounts to establish that, for all (X,Y), (X',Y') in $\mathcal{F}_L$, and all f in B^*, it holds:

$$(X,Y) \xrightarrow[\mathcal{S}_L^{\mathcal{K}}]{f} (X',Y') \quad \Longleftrightarrow \quad \sigma(f) \subseteq F_{X,Y'}^L.$$

For $|f| = 1$, this is exactly the definition of $\mathcal{S}_L^{\mathcal{K}}$.

Suppose then that we have $(X,Y) \xrightarrow[\mathcal{S}_L^{\mathcal{K}}]{xf} (X',Y')$; then there exists (X'',Y'') in $\mathcal{F}_L$ such that $(X,Y) \xrightarrow[\mathcal{S}_L^{\mathcal{K}}]{x} (X'',Y'')$ and $(X'',Y'') \xrightarrow[\mathcal{S}_L^{\mathcal{K}}]{f} (X',Y')$. We thus have $\sigma(x) \subseteq F_{X,Y''}^L$ by definition of $\mathcal{S}_L^{\mathcal{K}}$ and $\sigma(f) \subseteq F_{X'',Y'}^L$ by induction hypothesis. Then, by Equation (1.8),

$$\sigma(xf) \subseteq F_{X,Y''}^L F_{X'',Y'}^L \subseteq F_{X,Y'}^L.$$

Conversely, suppose that $\sigma(xf) = \sigma(x)\sigma(f) \subseteq F_{X,Y'}^L$. By Lemma 3.4, there exists (X'',Y'') in $\mathcal{F}_L$ such that $\sigma(x) \subseteq F_{X,Y''}^L$ and $\sigma(f) \subseteq F_{X'',Y'}^L$. This, in turn, by definition of $\mathcal{S}_L^{\mathcal{K}}$ and by induction hypothesis, implies $(X,Y) \xrightarrow[\mathcal{S}_L^{\mathcal{K}}]{xf} (X',Y')$. Q.E.D.

As announced, a mere consequence of Theorem 6.1 is that if L is regular, $\mathcal{U}_L$ has a finite number of states and $\breve{\sigma}(L)$ is regular. The definition of $\mathcal{S}_L^{\mathcal{K}}$ is itself a procedure for computing the 'best approximation' to L, on condition that we know how to compute effectively the factors of L and the inclusion of K_i in these factors. These conditions are fulfilled in particular when considering the rational sets of a free monoid. We then deduce:

Corollary 6.2. Given a regular language L and a *finite* family $\mathcal{K}$ of regular languages over A^*, we can decide whether L belongs to $\mathsf{Rat}\mathcal{K}$, the rational closure of $\mathcal{K}$.

Proof. We compute the best approximation to L by the n languages of the family $\mathcal{K}$ and then decide whether this approximation is equal to L. Q.E.D.

The elegance of this proof, and the efficiency of the computations it entails is to be compared with those of the proofs given subsequently for the same result (*e.g.* [10]).

6.2 Solutions of pure language equations

The problem of approximation is susceptible to a 'dual' approach.[6] The (recognisable) subset L of A^* having been fixed, instead of choosing the subsets K_i, that is the substitution $\sigma\colon B^* \to A^*$, and trying to compute the language $\breve{\sigma}(L)$ over B^*, we can choose a language W (not even necessarily regular) over a free monoid B^* and seek a substitution $\sigma\colon B^* \to A^*$ such that $\sigma(W) \subseteq L$, which will be called a *sub-solution* of the *problem* (L, W). The sub-solutions are naturally (and partially) ordered by inclusion of the images of the letters of B, and the interesting sub-solutions are the maximal ones.

Theorem 6.3 (Conway, [5])**.** Let L be a subset of A^*, W a language of B^* and $\sigma\colon B^* \to A^*$ a maximal sub-solution of the problem (L, W). Then for each x in B, $\sigma(x)$ is an intersection of factors of L.

Proof. Let $f = x_1 x_2 \ldots x_n$ be a word of W. If σ is a solution of (L, W), $\sigma(x_1)\sigma(x_2)\ldots\sigma(x_n) \subseteq L$. By Lemma 3.4, and an induction argument, there exist $(X_0, Y_0), (X_1, Y_1), \ldots, (X_n, Y_n)$ in $\mathcal{F}_L$ such that

$$\sigma(x_i) \subseteq F^L_{X_{i-1}, Y_i}$$

for each i in $[1; n]$. As these inclusions are verified for each f in W, each $\sigma(x_i)$ is contained in an intersection of factors and such an intersection is a maximal component in a sub-solution of the problem. Q.E.D.

Corollary 6.4 (Conway, [5])**.** If L is regular, then the maximal sub-solutions of the problem (L, W) are k-tuples ($k = \mathrm{Card}(B)$) of regular subsets of A^*. If in addition W is regular, we can effectively compute all the maximal sub-solutions of the problem (L, W).

Proof. If L is regular, there is only a finite number of factors that are all regular and their intersections are finite in number and regular. There is only a finite number of k-tuples of intersections among which all the maximal

[6] This is not the left-right duality of automata, but rather a vector–linear form duality.

sub-solutions are found. If W is regular we can effectively find all the k-tuples which are sub-solutions and keep only the maximal ones. Q.E.D.

Example 6.5. A regular language L of A^* being chosen, let us find all the subsets U such that $U^2 \subseteq L$ and U is maximal for this property (*i.e.* find the maximal sub-solutions of the problem (L, x^2)). If $U^2 \subseteq L$, (U, U) is a subfactorisation of L, it is dominated by (at least) one factorisation (X, Y), and $U \subseteq X \cap Y$. The maximal sub-solutions are thus among the $X \cap Y$ when (X, Y) varies over $\mathcal{F}_L$.

7 Stars in the universal automaton

Last but not least, the universal automaton contains informations on the *star height* of the language if it is a regular one, may be not always but certainly for some subfamilies of regular languages — and this was what motivated first the interest of the authors in this construction.

The computation of the star height problem is a hard question that was stated by Eggan [8] in 1963. It was positively solved in 1988 by Hashiguchi [11] and Kirsten gave more recently a particulary elegant proof for its decidability [15]. The results we present here do not give the solution of the star height problem for any regular language, but in the cases where they can be applied, they give more precise informations on the form of the result than the other works.

7.1 Star height and loop complexity

The *star height* of a regular expression E, denoted by $\mathsf{h}(\mathsf{E})$, is defined recursively by:

$$\begin{array}{ll} \text{if } \mathsf{E} = 0,\ \mathsf{E} = 1 \text{ or } \mathsf{E} = a \in A, & \mathsf{h}(\mathsf{E}) = 0\,, \\ \text{if } \mathsf{E} = \mathsf{E}' + \mathsf{E}'' \text{ or } \mathsf{E} = \mathsf{E}' \cdot \mathsf{E}'', & \mathsf{h}(\mathsf{E}) = \max(\mathsf{h}(\mathsf{E}'), \mathsf{h}(\mathsf{E}''))\,, \\ \text{if } \mathsf{E} = \mathsf{F}^*, & \mathsf{h}(\mathsf{E}) = 1 + \mathsf{h}(\mathsf{F})\,. \end{array}$$

Example 7.1. The expressions $(a + 1)(a^2 + b)^*a + 1$ and $(b^*a + 1)(ab^*a)^*$ have star height 1 and 2 respectively. As they both denote the same language accepted by the automaton $\mathcal{A}_2$ shown at Figure 13, two equivalent expressions may have different star heights.

Definition 7.2. The star height of a regular language L of A^*, which we note as $\mathsf{h}(L)$, is the *minimum* of the star height of the expressions that denote the language L:

$$\mathsf{h}(L) = \min\{\mathsf{h}(\mathsf{E}) \mid \mathsf{E} \in \mathsf{RatE}A^* \quad |\mathsf{E}| = L\}\,.$$

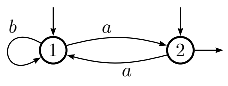


FIGURE 13. The automaton $\mathcal{A}_2$

The star height induces a hierarchy on regular languages. We shall give examples for the fact (see Corollary 7.12):

Fact 7.3. There exist regular languages of arbitrary large star height.

The star height of an expression reflects also a structural property of an automaton which corresponds to that expression (more precisely, of the underlying graph of an automaton). In order to state it, we first define the notion of *a ball* of a graph: a ball in a graph is a strongly connected component that contains at least one arc.

Definition 7.4. The *loop complexity*[7] of a graph $\mathcal{G}$ is the integer $\mathsf{lc}(\mathcal{G})$ recursively defined by:

$$\begin{aligned} &\mathsf{lc}(\mathcal{G}) = 0 && \text{if } \mathcal{G} \text{ contains no ball (in particular, if } \mathcal{G} \text{ is empty);}\\ &\mathsf{lc}(\mathcal{G}) = \max\{\mathsf{lc}(\mathcal{P}) \mid \mathcal{P} \text{ ball of } \mathcal{G}\} && \text{if } \mathcal{G} \text{ is not a ball itself;}\\ &\mathsf{lc}(\mathcal{G}) = 1 + \min\{\mathsf{lc}(\mathcal{G} \setminus \{s\}) \mid s \text{ vertex of } \mathcal{G}\} && \text{if } \mathcal{G} \text{ is a ball.} \end{aligned}$$

As Eggan showed, star height and loop complexity are the two faces of the same notion:

Theorem 7.5 (Eggan [8])**.** The star height of a language L is equal to the minimal loop complexity of an automaton that accepts L.

More precisely, from every automaton with loop complexity n, an expression with star height n can be computed, and *vice-versa.* Theorem 7.5 allows to deal with automata instead of expressions, and to look for automata of minimal loop complexity instead of expressions of minimal star height. A reason why star height, or loop complexity is not an easy parameter to compute is given by the following fact, for which we give an example below (see Example 7.13).

Fact 7.6. The minimal automaton is not always of minimal loop complexity (for the language it recognises).

[7] Eggan [8] as well as Cohen [4] and Hashiguchi [12] call it ‘cycle rank’, Büchi calls it ‘feedback complexity’. McNaughton [23] calls loop complexity of a language the minimum cycle rank of an automaton that accepts the language. We have taken this terminology and made it parallel to star height, for ‘rank’ is a word of already many different meanings.

The following structural result gives a criterium to bound the loop complexity of an automaton.

Definition 7.7. Let $\mathcal{A}$ and $\mathcal{B}$ be two automata and let φ be a surjective morphism from $\mathcal{A}$ onto $\mathcal{B}$. The morphism φ is *conformal* if every path in $\mathcal{B}$ is the image of a path in $\mathcal{A}$.

Theorem 7.8 (McNaughton, [23])**.** If $\varphi\colon \mathcal{B} \to \mathcal{A}$ is a conformal morphism, the loop complexity of $\mathcal{B}$ is larger than or equal to that of $\mathcal{A}$: that is, $\mathsf{lc}(\mathcal{B}) \geqslant \mathsf{lc}(\mathcal{A})$.

We first show a lemma:

Lemma 7.9. Let $\varphi\colon \mathcal{B} \to \mathcal{A}$ be a conformal morphism. For every ball $\mathcal{P}$ in $\mathcal{A}$, there exists a ball $\mathcal{Q}$ in $\mathcal{B}$ such that the restriction of φ to $\mathcal{Q}$ is a conformal morphism from $\mathcal{Q}$ to $\mathcal{P}$.

Proof. This lemma (like the theorem) is in fact a proposition about graphs, but we shall use automata-theoretic notions to simplify the proof. We assume, possibly by changing them all, that each transition of $\mathcal{A}$ bears a distinct label, and that each state of $\mathcal{A}$ is both initial and final; this may change the language accepted by $\mathcal{A}$ but has no effect on its loop complexity. The words of the language recognised by $\mathcal{A}$ (*resp.* by a subautomaton $\mathcal{P}$ of $\mathcal{A}$) describe paths in the *graph* $\mathcal{A}$ (*resp.* in the sub-graph $\mathcal{P}$). The transitions of $\mathcal{B}$ are labeled in such a way that φ is an automata morphism and each state of $\mathcal{B}$ is both initial and final.

Let $\mathcal{P}$ be a ball in $\mathcal{A}$ and $\mathcal{R} = \mathcal{P}\varphi^{-1}$. Set $n = \|\mathcal{R}\|$ and $m = \|\mathcal{P}\|$ to be the number of states of $\mathcal{R}$ and $\mathcal{P}$ respectively and consider a circuit (hence a word) w which visits all the paths in $\mathcal{P}$ of length less than 2^{n+m}. The circuit w^n is a path in $\mathcal{P}$ which can be lifted to a path in $\mathcal{R}$ (since φ is conformal). By the proof of the block star lemma, a factor w^k is the label of a *circuit* in $\mathcal{R}$; let $\mathcal{Q}$ be the ball in $\mathcal{R}$, and hence in $\mathcal{B}$, that contains this circuit. By construction, $\mathcal{Q}$ recognises all words of length less than 2^{n+m} of the language recognised by $\mathcal{P}$, hence $\mathcal{Q}$ is equivalent to $\mathcal{P}$, hence all the paths in $\mathcal{P}$ become paths in $\mathcal{Q}$: thus, φ is conformal from $\mathcal{Q}$ to $\mathcal{P}$. Q.E.D.

Proof of Theorem 7.8. Suppose that the property is false, and proceed by *reductio ad absurdum.* Among the automata which are sent by a conformal morphism to an automaton of strictly greater complexity, let $\mathcal{B}$ be an automaton of minimal loop complexity d, and let $\mathcal{A}$, of complexity c, be the image of $\mathcal{B}$ under a conformal morphism: thus, $c > d$.

If $d = 0$, the length of the paths in $\mathcal{B}$ is bounded and it is impossible for φ to be conformal, hence $d > 0$.

By definition, there is a ball $\mathcal{P}$ in $\mathcal{A}$ of complexity c and, by Lemma 7.9, a ball $\mathcal{Q}$ in $\mathcal{B}$ whose image under φ is $\mathcal{P}$. This ball is of complexity at most d

but also, by the minimality of d, at least d. There exists a state q in $\mathcal{Q}$ such that

$$\mathsf{lc}(\mathcal{Q}\smallsetminus\{q\}) = d-1. \tag{1.19}$$

Let $p = q\varphi$, $\mathcal{P}' = \mathcal{P}\smallsetminus\{p\}$ and $\mathcal{Q}' = \mathcal{Q}\smallsetminus\{p\varphi^{-1}\}$; we have $\mathsf{lc}(\mathcal{Q}') \leqslant \mathsf{lc}(\mathcal{Q}\smallsetminus\{q\}) = d-1$ and $\mathsf{lc}(\mathcal{P}') \geqslant c-1 > d-1$.

Every path in $\mathcal{P}'$ is a path in $\mathcal{P}$ which does not visit p, hence the image of a path in $\mathcal{Q}$ which does not go through any of the vertices of $p\varphi^{-1}$; that is, the image of a path in $\mathcal{Q}'$: thus, φ is a conformal morphism from $\mathcal{Q}'$ to $\mathcal{P}'$, which contradicts the assumed minimality of d. Q.E.D.

7.2 Star height of group languages

The star height of a group language can be computed within the universal automaton. The simplest instance of this fact is the following statement which provides a new, easier, and clearer presentation of McNaughton's proof of computability of the star height of pure group languages.

Theorem 7.10 (Lombardy-Sakarovitch, [21])**.** The universal automaton of a regular group language L contains a subautomaton of minimal loop complexity that recognises L.

Since the universal automaton of a regular language is finite, we can enumerate its subautomata, keeping those that recognise the language, and from among them find those of minimal loop complexity. We therefore have:

Corollary 7.11 (McNaughton, [23])**.** The star height of a regular group language is computable.

Furthermore, the same theorem allows us to establish directly, a result whose original proof relied on a highly subtle combinatorial method. Let W_q be the language defined by:

$$W_q = \{w \in \{a,b\}^* \mid |w|_a \equiv |w|_b \mod 2^q\}.$$

Corollary 7.12 (Dejean-Schützenberger, [7])**.** $\mathsf{lc}(W_q) = q$.

In this case indeed the universal automaton is isomorphic to the minimal automaton, which has thus the minimal loop complexity (see below).

Example 7.13. Let H_2 and H_3 be the languages over $A^* = \{a,b\}^*$ consisting of words whose number of a's is *congruent* to the number of b's *plus* 1 modulo 2 and 3 respectively and H_6 their union:

$$H_2 = \{f \mid |f|_a - |f|_b \equiv 1 \mod 2\}\,, \quad H_3 = \{f \mid |f|_a - |f|_b \equiv 1 \mod 3\}$$
$$\text{and} \qquad H_6 = \{f \mid |f|_a - |f|_b \equiv 1, 3, 4 \text{ or } 5 \mod 6\}.$$

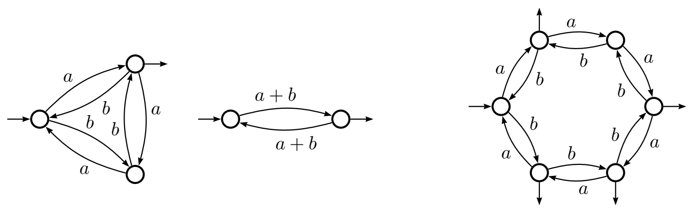


FIGURE 14. An automaton of minimal loop complexity (left) which is not the minimal automaton (right) for H_6

The minimal automaton of H_6 is the 'double ring' of length 6 whose loop complexity is 3. The minimal automata of H_2 and H_3 have complexity 1 and 2, hence the star height of H_6 is at most 2 (*cf.* Figure 14).

Figure 15 shows the écorché of the universal automaton of H_6. We see, all the better for its grey background, a subautomaton of this universal automaton which recognises H_6, with a minimal complexity. This subautomaton is equal to the union of the minimal automata of H_2 and H_3 seen above, and this is not a coincidence.

Let $\mathcal{B}$ be an automaton of minimal loop complexity which recognises L and $\varphi\colon \mathcal{B} \to \mathcal{U}_L$ a morphism from $\mathcal{B}$ to the universal automaton of L. If φ is a *conformal* morphism from $\mathcal{B}$ to its image $\varphi(\mathcal{B})$ in $\mathcal{U}_L$, this subautomaton of $\mathcal{U}_L$ is of lesser or equal complexity to that of $\mathcal{B}$ by Theorem 7.8 and the property is proved. However, in the general case φ is not conformal. The proof comes down to showing that nonetheless φ is conformal on some subautomata of $\mathcal{B}$ (on some balls) which are crucial for the complexity. We start by proving some properties of the structure of the universal automaton of a group language.

7.2.1 The universal automaton of a group language

In what follows, $L \subseteq A^*$ is a group language, $\alpha\colon A^* \to G$ is the syntactic morphism, $P = \alpha(L)$ and $\mathcal{A}_L = \langle G, A, \delta, 1_G, P\rangle$ is a complete accessible deterministic automaton that recognises L. For w in A^* and g in G we therefore write $g \triangleright w$ for $g\alpha(w)$, multiplication in G.

As we have seen (Subsection 3.2), the universal automaton $\mathcal{U}_L$ of L, is obtained by considering the factorisations (X, Y) of P in G and that if

$$(X_1, Y_1) \xrightarrow[\mathcal{U}_L]{a} (X_2, Y_2)$$

is a transition of $\mathcal{U}_L$, then $X_1(a\alpha)Y_2 \subseteq P$ and hence

$$X_1 \triangleright a \subseteq X_2 \quad \text{and} \quad \alpha(a)^{-1}Y_2 \subseteq Y_1.$$

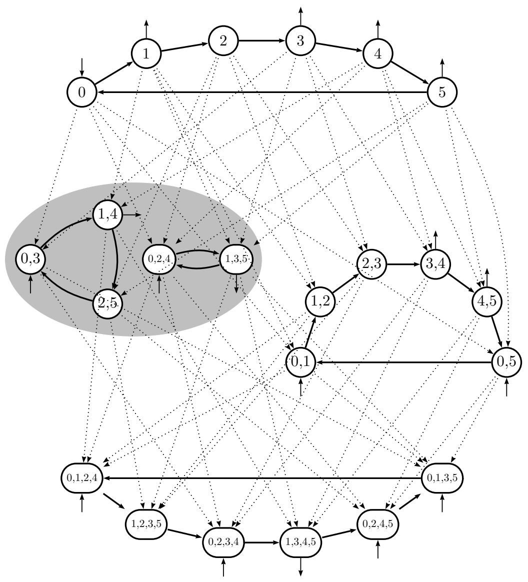


FIGURE 15. The écorché of the universal automaton of H_6 (without the sink and co-sink states). The bold arrows represent a double transition, one labeled a in the direction of the arrow and one labeled b in the opposite direction; the dotted arrows represent the spontaneous transitions.

Lemma 7.14. The balls of $\mathcal{U}_L$ are deterministic and complete.

Proof. Let (X_1, Y_1) and (X_2, Y_2) be two states of $\mathcal{U}_L$ belonging to the same ball. There exists u and v in A^* such that $X_1 \triangleright u \subseteq X_2$ and $X_2 \triangleright v \subseteq X_1$. As G is a group, the action of every element is injective and $\|X_1\| \leqslant \|X_2\| \leqslant \|X_1\|$ hence $\|X_1\| = \|X_2\|$ and $X_1 \triangleright u = X_2$. That is, X_2 is *uniquely determined* by X_1 and u: the ball is deterministic.

Furthermore, if (X, Y) is a factorisation of P, then $(X(u\alpha), (u\alpha)^{-1}Y)$ is also a factorisation of P, for all u in A^*, and there exists a transition labeled u from the first to the second. For all u, there exists v such that $(uv)\alpha = 1_G$, and hence a transition labeled v from $(X(u\alpha), (u\alpha)^{-1}Y)$ to (X, Y). Thus, $(X(u\alpha), (u\alpha)^{-1}Y)$ belongs to the same ball as (X, Y) and the ball is complete. Q.E.D.

A direct consequence of Lemma 7.14 is the following.

Corollary 7.15. Let L be a group language whose image in its syntactic monoid is reduced to one element. Then $\mathcal{U}_L$ is isomorphic with the minimal automaton of L whose loop complexity is thus minimal.

7.2.2 Proof of Theorem 7.10

Lemma 7.16. For every integer k, there exists a word w_k in A^* whose image in G is 1_G and such that every computation of length k of every ball C in $\mathcal{U}_L$ is contained in every computation of C labeled w_k.

Proof. Every word whose image in G is 1_G labels a circuit in every ball of $\mathcal{U}_L$ and for every source vertex. For each ball, and each vertex of this ball, we construct a circuit which visits every computation of length k of this ball. The product of the labels of all these circuits is a word w_k that answers the question. Q.E.D.

We now turn to the proof of the theorem itself.

Proof of Theorem 7.10. The automaton $\mathcal{B}$, an automaton of minimal loop complexity which recognises L, has n states. Let g be in P, a final state of $\mathcal{A}_L$, and u_g be a word in A^* that is sent to g by α. For every integer k, the word $(w_k)^n u_g$ is in L and is hence accepted by $\mathcal{B}$. The Block Star Lemma, applied to the factors w_k, ensures that there exists a state p_k of $\mathcal{B}$ such that there exists a circuit with source p_k labeled by a certain power (w_k^l). Let $\mathcal{D}_k$ be the ball in $\mathcal{B}$ which contains p_k, and hence this circuit. We thus obtain an infinite sequence of balls $\mathcal{D}_k$ in which at least one ball $\mathcal{D}$ in $\mathcal{B}$ appears infinitely often.

Let $\mathcal{C}$ be the ball in $\mathcal{U}_L$ which contains the image of $\mathcal{D}$ under the morphism $\varphi\colon \mathcal{B} \to \mathcal{U}_L$. For every path c in $\mathcal{C}$, there exists a k greater than the length of c, an integer l and a state p of $\mathcal{D}$ such that there exists a loop in $\mathcal{D}$ with source p labeled $(w_k)^l$. This same word $(w_k)^l$ labels a loop in $\mathcal{C}$ which contains all the computations of length less than or equal to k; it thus contains c in particular. That is, c is the image of a computation of $\mathcal{D}$, hence on one hand, $\mathcal{C}$ is the image of $\mathcal{D}$ under φ and on the other, the restriction of φ to $\mathcal{D}$ is *conformal*. By Theorem 7.8, $\mathsf{lc}(\mathcal{D}) \geqslant \mathsf{lc}(\mathcal{C})$.

Let (X, Y) be the factorisation, which is the image of p under φ (the state p that was defined just above). Since $(w_k)^{l'}$ is in $\mathsf{Past}_{\mathcal{B}}(p)$, 1_G is in $\mathsf{Past}_{\mathcal{U}_L}((X, Y))$ and hence 1_G is in X; that is, (X, Y) is an initial state of $\mathcal{U}_L$. Likewise, $(w_k)^{l''} u_g$ is in $\mathsf{Fut}_{\mathcal{B}}(p)$ and g is in Y. Every word u of A^* such that $u\alpha = g$ labels a computation of $\mathcal{C}$ with source (X, Y) and destination $(Xg, g^{-1}Y)$, a final state of $\mathcal{U}_L$, since $1_G \in g^{-1}Y$. Hence u is accepted by $\mathcal{C}$.

We can repeat this construction for each g in P and finally obtain a set of balls of $\mathcal{U}_L$ that recognise all of L and each of which has complexity less than or equal to at least one ball in $\mathcal{B}$. The complexity of the set is at most equal to that of $\mathcal{B}$, which was assumed to be minimal. Q.E.D.

7.3 Star height of reversible languages

The method of the proof of Theorem 7.10 can be both deepened and generalised in order to settle the question of star height for a larger class of languages.

Definition 7.17. An automaton $\mathcal{A}$ is *reversible* if the letters induce partial bijections on the set of states, that is, if for every state p and every letter a, $\mathsf{card}(p \triangleright a) \leqslant 1$ and $\mathsf{card}(a \triangleleft p) \leqslant 1$.

A language is *reversible* if it is recognised by a reversible automaton.

Remark 7.18. A reversible automaton may be not deterministic, nor co-deterministic, for the definition puts no restriction on the number of initial or final sates.

The minimal automaton of a reversible language may be not reversible. Nevertheless, given an automaton, it can be decided (in polynomial time) whether the language it accepts is reversible or not (see [24]). It is to be stressed that this decision procedure *does not* yield a reversible automaton for a regular language that is determined to be reversible but only the information that such a reversible automaton exists.

Theorem 7.19 (Lombardy-Sakarovitch, [20])**.** The universal automaton of a reversible language contains an equivalent subautomaton of minimal loop complexity.

The subautomaton quoted in this result is not necessarily reversible, but it is 'not far' of being so. We then introduce a weaker notion for automata, that will not change the class of accepted languages and that will be useful for both the statement and the proof of the result.

Definition 7.20. An automaton $\mathcal{A}$ is *quasi-reversible* if for every state p and every letter a the following holds:

(i) if $\mathsf{card}(p \triangleright a) > 1$, none of the states in $p \triangleright a$ is in the same ball as p;

(ii) if $\mathsf{card}(a \triangleleft p) > 1$, none of the states is $a \triangleleft p$ is in the same ball as p.

Quasi-reversible automata will be analysed by means of the following decomposition.

Definition 7.21. Let $\mathcal{A}$ be an automaton. A subautomaton $\mathcal{B}$ of $\mathcal{A}$ is *a $\mathcal{A}$-constituent* if the following holds:

(i) any ball of $\mathcal{A}$ is either contained in, or disjoint from, $\mathcal{B}$;

(ii) there is at most one incoming transition to, and one outgoing transition from, every ball of $\mathcal{B}$;

(iii) $\mathcal{B}$ has one initial state and one final state.

It follows from the definition that every finite automaton $\mathcal{A}$ has a finite (but exponential) number of $\mathcal{A}$-constituents and that any $\mathcal{A}$-constituent of a quasi-reversible automaton $\mathcal{A}$ is a reversible automaton. It then holds:

Proposition 7.22. The language accepted by a quasi-reversible automaton is reversible.

We can now give the main result of this section its true form.

Theorem 7.23 (Lombardy, [17])**.** The universal automaton of a reversible language contains an equivalent quasi-reversible subautomaton of minimal loop complexity.

The overall scheme of the proof is illustrated by the figure below.

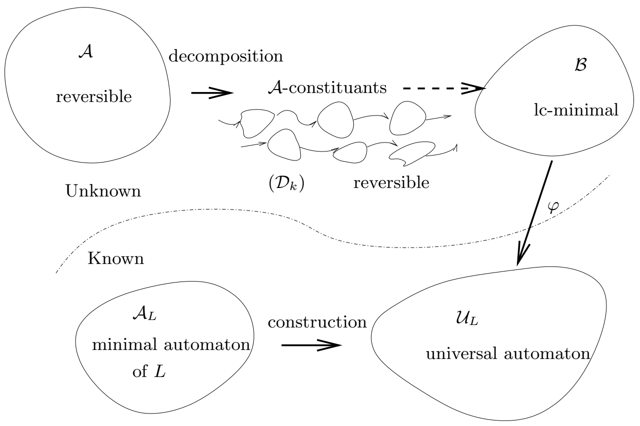


FIGURE 16. The construction underlying the proof of Theorem 7.23

Let L be a reversible language. We know that there exists an unknown automaton $\mathcal{A}$ that recognizes this language and there exists an unknown automaton $\mathcal{B}$ that recognizes this language with a minimal loop complexity. On the other side, we can build the minimal automaton $\mathcal{A}_L$ of the language and the universal automaton $\mathcal{U}_L$. We know that there exists a morphism φ from $\mathcal{B}$ into $\mathcal{U}_L$. Notice that the image of $\mathcal{B}$ by φ may have a loop complexity greater than the loop complexity of $\mathcal{B}$.

Thanks to the reversible automaton $\mathcal{A}$, we decompose L into a union of sub-languages, and we prove that the images of the computations in $\mathcal{B}$ labeled by these sub-languages give a subautomaton of $\mathcal{U}_L$ which is both quasi-reversible and with minimal loop complexity.

To prove the theorem, we must give first a more precise description of the structure of the universal automaton.

7.3.1 The universal automaton of a reversible language

To handle the particular structure of the universal automaton of a reversible language, we consider the construction of the universal automaton from a reversible automaton $\mathcal{A}$ with set of states Q. From Proposition 5.4, every state of the universal automaton is an upset of $\mathfrak{P}(Q)$.

Every upset is characterized by the anti-chain of its minimal elements. The *shape* of an upset R of $\mathfrak{P}(Q)$ is a $|Q|+1$-uplet $s(R)$ of integers such that, for every $k \in [0;|Q|]$, $s(R)_k$ is the number of subsets of Q with cardinal k among minimal elements of R. We define a lexicographic order on shapes:

$$\begin{aligned}&s(R) < s(R') \quad \Longleftrightarrow \\ &\exists k \in [0;|Q|]\,,\ \forall l \in [0;k-1] \qquad s(R)_l = s(R')_l \quad \text{and} \quad s(R)_k < s(R')_k.\end{aligned}$$

Proposition 7.24. If there is a path in the universal automaton from a state with index R to a state with index R', then $s(R) \leqslant s(R')$

Proof. Let w be the label of the path. The state R' contains $\{X \triangleright w \mid X \in R\}$. For every minimal element X in R, either $X \triangleright w$ has the same cardinal as X, or it has a smaller cardinal.

If there is some X such that $|X \triangleright w| < |X|$, thanks to the reversibility of $\mathcal{A}$ there is no X' such that $X' \triangleright w = X \triangleright w$ and $|X'| = |X' \triangleright w|$, hence, $s(R) \leqslant s(R')$. Otherwise, let M be the set of minimal elements of R; the set $\{X \triangleright w \mid X \in M\}$ is a subset of the set of minimal elements of R' and $s(R) \leqslant s(R')$. Q.E.D.

Proposition 7.25. The balls of the universal automaton of a reversible language are reversible.

Proof. Let R and R' be two such states. Let u be a word that labels a path from R to R' and let v be a word that labels a path from R' to R. By Proposition 7.24, two states that belong to the same ball have the same shape. In this case, if M the set of minimal elements of R, for every X in M, $Y = X \triangleright u$ is a minimal element of R' and $|Y| = |X|$. Thanks to the reversibility, the mapping from M into the minimal elements M' of R' is injective. Likewise, there is an injective mapping from M' into M. Therefore, the word u induces a bijection between M and M'; as these minimal elements characterize states, the balls are reversible. Q.E.D.

Corollary 7.26 (Cohen, [4])**.** If L is a reversible language recognised by a reversible minimal automaton with only one final state, then the minimal automaton has a minimal loop complexity.

Actually, in this case, the universal automaton is the minimal automaton itself and the only subautomaton that accepts the langugae is the complete universal automaton.

7.3.2 Proof of Theorem 7.23

We begin with a series of definitions and notation that allow us to describe a decomposition of a language according to an automaton that accepts it and to state a property of the constituents of that decomposition. This is indeed an adaptation of a method devised by Hashiguchi in [12].

Definition 7.27. Let $\mathcal{A}$ be a reversible automaton that accepts L.

(i) We say that a word w is *an idempotent for* $\mathcal{A}$ if, for every state p, $p \triangleright w = p$ or $p \triangleright w = \varnothing$.

(ii) Let $\mathcal{C}$ be a trim $\mathcal{A}$-constituent with m balls. The *marker sequence* of $\mathcal{C}$ is the $2m$-uple $(p_1, q_1, \ldots, p_m, q_m)$ such that p_i (*resp.* q_i) is the first (*resp.* last) state of the ith ball crossed by any computation.

(iii) A $\mathcal{A}$-constituent with marker sequence $(p_1, q_1, \ldots, p_m, q_m)$ accepts the language $v_0 H_1 v_1 H_2 \ldots v_{m-1} H_m v_m$, where H_i is the language of labels of paths from p_i to q_i.

(iv) We denote by W_i the set of idempotents for $\mathcal{A}$ that label a circuit around p_i.

$$\begin{tikzcd} i \arrow[r, "v_0"] & p_1 \arrow[r, "u_1"] & q_1 \arrow[r, "v_1"] & p_2 \arrow[r, "u_2"] & q_2 \arrow[r] & q_{m-1} \arrow[r, "v_{m-1}"] & p_m \arrow[r, "u_m"] & q_m \arrow[r, "v_m"] & t \end{tikzcd}$$

FIGURE 17. A marker sequence

Lemma 7.28. Let $\mathcal{A}$ be a reversible automaton and $\mathcal{C}$ a trim $\mathcal{A}$-constituent with m balls. Let $\mathcal{B}$ be any automaton equivalent to $\mathcal{A}$.

Then, there exist m states r_1, r_2,…,r_m in $\mathcal{B}$ such that, with the above notation, the following holds:

$$v_0 \, W_{p_1} \cap \mathsf{Past}_{\mathcal{B}}(r_1) \neq \varnothing, \qquad H_m \, v_m \cap \mathsf{Fut}_{\mathcal{B}}(r_m) \neq \varnothing,$$
$$\text{and,} \quad \forall i \in [1; m-1] \qquad (H_i \, v_i \, W_{p_i}) \cap \mathsf{Trans}_{\mathcal{B}}(r_i, r_i + 1) \neq \varnothing.$$

For every i in $[1;m]$, for every circuit around p_i labeled by a word v, there exists a circuit around r_i, labeled by a word $u\,v\,w$, where u is in W_{p_i} and w in A^*.

Proof. There exists an integer k such that, for every word $v \in A^*$, the image of v^k is an idempotent for $\mathcal{A}$. Let n be the number of states of $\mathcal{B}$. Let l be an integer that will silently index the sets we define now. For every $i \in [1;m]$, let C_i be the set of words of length smaller than l that label a circuit around p_i in $\mathcal{A}$. Let w_i be the product of all kth power of words in C_i:

$$w_i = \prod_{v \in C_i} v^k \,.$$

For every $v_0 u_1 v_1 \cdots v_m$ in the $\mathcal{A}$-constituent,

$$w = v_0 (w_1)^n u_1 v_1 (w_2)^n u_2 ... (w_m)^n u_m v_m$$

is in the $\mathcal{A}$-constituent as well. Hence, there is a successful computation labeled by w in $\mathcal{B}$. As $\mathcal{B}$ has only n states, this path contains, for every i, a loop labeled by a power of w_i around a state r_i of $\mathcal{B}$. The m-tuple $r^{(l)} = (r_1, r_2, \ldots, r_m)$ verifies i) and ii) for y shorter than l. If we consider the infinite sequence $r^{(1)}, r^{(2)}, \ldots$, we can find an m-tuple that occurs infinitly often and that verifies the lemma. Q.E.D.

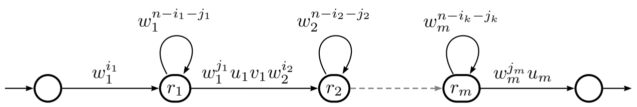


FIGURE 18. A witness word for a $\mathcal{A}$-constituent.

We can now proceed to the proof of Theorem 7.23. We consider a set C of $\mathcal{A}$-constituents such that every element of C accepts at least one word that is not accepted by the other elements of C and such that the union of elements of C is equivalent to $\mathcal{A}$.

Let $\mathcal{D}$ be an element of C and let $p_1, q_1, p_2, \ldots, q_m$ be the marker sequence of $\mathcal{D}$ and let $u = v_0 u_1 v_1 \ldots v_k$ be a word accepted only by $\mathcal{D}$ in C, with v_i labelling a path from q_{i-1} to p_i and u_i a path from p_i to q_i. Let $r_1, r_2, \ldots, r_m$ be the states of $\mathcal{B}$ defined in Lemma 7.28 w.r.t the $\mathcal{A}$-constituent $\mathcal{D}$ and $w_1, \ldots, w_m$ be the idempotents defined in the proof of the lemma. Let φ be a morphism from $\mathcal{B}$ into the universal automaton.

We deal with the strongly connected component of r_i, for $i \in [1;m]$. Let $s_i = \varphi(r_i)$ and $\mathcal{P}_i$ be the ball of $\mathcal{U}_L$ containing s_i. There exist integers $h_1, \ldots, h_m$ (*resp.* $l_1, \ldots, l_m$) such that the word $x = v_0 w_1^{h_1+l_1} u_1 v_1 \ldots w_i^{h_i}$ (*resp.* $y = w_i^{l_i} u_i v_i \ldots w_k^{h_k+l_k} v_k u_k$) is in the past (*resp.* the future) of r_i and thus of s_i.

(i) The morphism φ is conformal on $\mathcal{P}_i$. Let $\mathcal{C}$ be a path of $\mathcal{P}_i$. We can assume, up to make it longer, that this is a circuit around s_i and, up to take it several times, that it is labeled by an idempotent for $\mathcal{A}$: z. The word xzy is in L; every $\mathcal{A}$-constituent that accepts this word accepts also xy, therefore xzy is accepted only by $\mathcal{D}$ in C. As $\mathcal{D}$ is reversible, x labels a path from the initial state to p_i, y a path from p_i to the final state and z a circuit around p_i. Therefore, from Lemma 7.28, there exist a word w idempotent for $\mathcal{A}$ and a word v such that wzv labels a circuit around r_i. The image of this circuit is a circuit around s_i. As w is an idempotent for $\mathcal{A}$, it is an idempotent in the syntactic monoid; hence for every k, $w^k zv$ labels a circuit, if k is large enough, this circuit contains a sub-circuit labeled by a power of w, as the ball is reversible, this power labels a circuit around s_i, and as w is an idempotent, it labels itself a circuit around s_i. As balls are deterministic, the circuit $\mathcal{C}$ around s_i is the image of the part of the circuit around r_i labeled by z. Thus the morphism φ is conformal onto $\mathcal{P}_i$ which have a loop complexity not greater than the loop complexity of $\mathcal{B}$.

(ii) The images of the words linking balls in $\mathcal{B}$ contain no circuit. The word $w_i^{l_i} u_i v_i w_{i+1}^{h_{i+1}}$ is in $\mathsf{Trans}_{\mathcal{B}}(r_i, r_{i+1})$ thus in $\mathsf{Trans}_{\mathcal{U}_L}(s_i, s_{i+1})$. Let $s_i = (X_i, Y_i)$ and let $t_i = (X'_i, Y'_i)$ be the state in $\mathcal{P}_i$ such that u_i is in $\mathsf{Trans}_{\mathcal{U}_L}(s_i, t_i)$. By definition of the universal automaton:

$$X_i w_i^{l_i} u_i v_i w_{i+1}^{h_{i+1}} Y_{i+1} \subseteq L.$$

The words w_i and w_{i+1} are idempotents and L is reversible, therefore it holds $X_i u_i v_i Y_{i+1} \subseteq L$, and X'_i is the smallest left factor that contains $X_i u_i$, hence $X'_i v_i Y_{i+1} \subseteq L$ Thus there exists a path labeld by v_i from t_i to s_{i+1}. This holds for every i in $[1; k-1]$. We prove the same way, that there exists a path from an initial state to s_1 labeled by v_0 and a path from s_m to a final state labeled by v_m.

If one of the intern states of one of these paths labeled by v_i belongs to a ball, the word v_i can be factorised into $x_i y_i$ and there exists an idempotent w for $\mathcal{A}$ such that

$$v_0 u_1 v_1 \ldots u_i x_i w y_i u_{i+1} v_{i+1} \ldots u_k v_k \in L.$$

It can only be accepted by $\mathcal{D}$, which would imply the existence of a circuit between q_i and p_{i+1}.

(iii) The subautomaton obtained in $\mathcal{U}_L$ accepts every word accepted by $\mathcal{D}$. Such a word can be factorised into $v_0u'_1v_1su'_kv_k$, with $p_i \rhd u'_i = q_i$. There exists a word w_i such that u'_iw_i is an idempotent and both u_iw_i and u'_iw_i label circuits around p_i. As above, these words label circuits around s_i and, as the ball is co-deterministic, the path from s_i labeled by u'_i ends in the same state as the one labeled by u_i, *i.e.* t_i.

(iv) This subautomaton is reversible. The balls of the universal automaton are reversible. Between every ball, there is only one path in the automaton, by construction. If there exists a letter a that labels two incoming transitions of s_i, this letter is the last one in v_i and there exists a circuit around p_i with a as last letter, which is a constradiction with the reversibility of $\mathcal{D}$. Hence, this subautomaton is co-deterministic; likewise, it is deterministic.

(v) Conclusion of the proof. For every constituent of $\mathcal{A}$, we prove that there exists a subautomaton of the universal automaton, with a loop complexity not greater than the star height of the language, and that accepts every word accepted by the constituent. The superposition of all these subautomata of the universal automaton gives a subautomaton of the universal automaton that recoginzes the language. More, every ball intersected by one of these subautomata is entirely included in the subautomaton, hence, the loop complexity of the superposition is not greater than the maximal loop complexity of the superposed automata. Therefore the superposition is a subautomaton of the universal automaton that have a minimal loop complexity for the language.

Moreover, as every superposed automaton is reversible, the superposition is a quasi-reversible automaton. That proves that, for every reversible language, there exists a quasi-reversible automaton, with minimal loop complexity, and that is a subautomaton of the universal automaton.

8 Conclusion

The aim of this paper is to show the soundness of the notion of universal automaton and its various applications. Its large size leads to algorithms with poor complexity, but it is a good theoretical framework to state different kinds of problems on regular languages.

We end this survey with an open question about star height. We have said that, roughly speaking, the universal automaton of a language contains every automaton that accepts this language. This is true up to morphic image, but morphisms do not preserve loop complexity. This is the reason why in the general case, we do not know how to prove the following extension of Theorems 7.10 and 7.19: *The universal automaton of a regular language contains a subautomaton with a minimal loop complexity for this language.* The universal automaton has not revealed all its secrets.

References

[1] A. Arnold, A. Dicky, and M. Nivat. A note about minimal non-deterministic automata. *Bulletin of the EATCS*, 47:166–169, 1992.

[2] C. Carrez. On the minimalization of non-deterministic automaton. Technical report, Computing Laboratory of the Science Faculty of Lille University, 1970.

[3] M. Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(3):149–158, 1986.

[4] R. S. Cohen. Star height of certain families of regular events. *J. Comput. Syst. Sci.*, 4(3):281–297, 1970.

[5] J. H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, London, 1971.

[6] B. Courcelle, D. Niwiński, and A. Podelski. A geometrical view of the determinization and minimization of finite-state automata. *Mathematical Systems Theory*, 24(2):117–146, 1991.

[7] F. Dejean and M. P. Schützenberger. On a question of eggan. *Information and Control*, 9(1):23–25, 1966.

[8] L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10:385–397, 1963.

[9] I. Grunsky, O. Kurganskyy, and I. Potapov. On a maximal NFA without mergible states. In D. Grigoriev, J. Harrison, and E. A. Hirsch, editors, *CSR*, volume 3967 of *Lecture Notes in Computer Science*, pages 202–210. Springer, 2006.

[10] K. Hashiguchi. Representation theorems on regular languages. *J. Comput. Syst. Sci.*, 27(1):101–115, 1983.

[11] K. Hashiguchi. Algorithms for determining relative star height and star height. *Inf. Comput.*, 78(2):124–169, 1988.

[12] K. Hashiguchi and N. Honda. The star height of reset-free events and strictly locally testable events. *Information and Control*, 40(3):267–284, 1979.

[13] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.

[14] T. Kameda and P. Weiner. On the state minimization of nondeterministic finite automata. *IEEE Trans. Computers*, C-19(7):617–627, 1970.

[15] D. Kirsten. Distance desert automata and the star height problem. *Theor. Inform. Appl.*, 39(3):455–509, 2005.

[16] A. D. Korshunov. The number of monotone Boolean functions. *Problemy Kibernet.*, 38:5–108, 272, 1981.

[17] S. Lombardy. *Approche structurelle de quelques problèmes de la théorie des automates.* PhD thesis, ENST, Paris, 2001.

[18] S. Lombardy. On the construction of reversible automata for reversible languages. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 170–182. Springer, 2002.

[19] S. Lombardy. On the size of the universal automaton of a regular language. In W. Thomas and P. Weil, editors, *STACS*, volume 4393 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2007.

[20] S. Lombardy and J. Sakarovitch. Star height of reversible languages and universal automata. In *LATIN 2002: Theoretical informatics (Cancun)*, volume 2286 of *Lecture Notes in Comput. Sci.*, pages 76–90, Berlin, 2002. Springer.

[21] S. Lombardy and J. Sakarovitch. On the star height of rational languages: a new presentation for two old results. In *Words, languages & combinatorics, III (Kyoto, 2000)*, pages 266–285. World Sci. Publ., River Edge, NJ, 2003.

[22] O. Matz and A. Potthoff. Computing small finite nondeterministic automata. In *Proc. of the Workshop on Tools and Algorithms for Construction and Analysis of Systems*, BRICS Note Series, pages 74–88, Aarhus, 1995. BRICS.

[23] R. McNaughton. The loop complexity of pure-group events. *Information and Control*, 11:167–176, 1967.

[24] J.-E. Pin. On reversible automata. In I. Simon, editor, *LATIN*, volume 583 of *Lecture Notes in Computer Science*, pages 401–416. Springer, 1992.

[25] L. Polák. Minimalizations of NFA using the universal automaton. *Int. J. Found. Comput. Sci.*, 16(5):999–1010, 2005.

[26] C. Reutenauer. Sur les variétés de langages et de monoídes. In K. Weihrauch, editor, *Theoretical Computer Science*, volume 67 of *Lecture Notes in Computer Science*, pages 260–265. Springer, 1979.

[27] J. Sakarovitch. *Eléments de théorie des automates.* Vuibert, Paris, 2003. In French, English translation: *Elements of Automata Theory*, Cambridge University Press, to appear.

[28] D. Wiedemann. A computation of the eighth Dedekind number. *Order*, 8(1):5–6, 1991.

Deterministic top-down tree automata: past, present, and future

Wim Martens[1]
Frank Neven[2]
Thomas Schwentick[1]

[1] Lehrstuhl Informatik I
Universität Dortmund
44221 Dortmund, Germany
{Wim.Martens,Thomas.Schwentick}@udo.edu

[2] Departement Wiskunde, Natuurkunde en Informatica
Universiteit Hasselt
Agoralaan
3590 Diepenbeek, Belgium
frank.neven@uhasselt.be

Abstract

In strong contrast to their non-deterministic counterparts, deterministic top-down tree automata received little attention in the scientific literature. The aim of this article is to survey recent and less recent results and stipulate new research directions for top-down deterministic tree automata motivated by the advent of the XML data exchange format. In particular, we survey different ranked and unranked top-down tree automata models and discuss expressiveness, closure properties and the complexity of static analysis problems.

1 Introduction

The goal of this article is to survey some results concerning deterministic top-down tree automata motivated by purely formal language theoretic reasons (past) and by the advent of the data exchange format XML (present). Finally, we outline some new research directions (future).

The Past. Regular tree languages have been studied in depth ever since their introduction in the late sixties [10]. Just as for regular string languages, regular tree languages form a robust class admitting many closure properties and many equivalent formulations, the most prominent one in the form of tree automata. A striking difference with the string case where left-to-right equals right-to-left processing, is that top-down is no longer equivalent to bottom-up. In particular, top-down deterministic tree automata are strictly less expressive than their bottom-up counterparts and consequently form a strict subclass of the regular tree languages. Furthermore, deterministic top-down tree automata do not enjoy many of the important closure properties. For instance, they are neither closed under union nor under complement.

Several variants of deterministic top-down tree automata models have been introduced of which the one defined in [10, 7] is considered to be the standard one: the states assigned to the children of a node depend solely on the label and the state at the current node. We refer to these automata as 'blind' because they cannot see the label of the children when assigning states to them. A natural extension would therefore be to make automata 'sensing' by allowing them to see those labels. The latter model is more expressive than the former and both can be characterized by closure under a subtree exchange property. Using the latter property it becomes very easy to show that the models are neither closed under union nor under complement. The l-r-determinism for top-dowm tree automata introduced by Nivat and Podelski [17] and defining the homogeneous tree languages is strictly more expressive than blind automata and incomparable to sensing ones. Both blind and sensing tree automata allow for tractable static analysis: emptiness, containment and minimization are in PTIME.

The Present. XML, which stands for the eXtensible Markup Language, is a standard defined by W3C [4] for data exchange over the internet. From an abstract viewpoint, XML data or XML documents can be represented by finite labeled unranked trees where unranked means that there is no a priori bound on the number of child nodes a node can have. In a data exchange scenario not every XML document is allowed and the structure of XML documents is usually restricted to adhere to a specified schema. Many schema languages for XML exist of which the most prominent ones are DTD [4], XML Schema [20], and Relax NG [6]. In formal language theoretic terms, every schema defines an unranked tree language. This XML setting motivated Brüggemann-Klein, Murata, and Wood to develop a theory of unranked tree automata, an endeavor already initiated in the late sixties by Thatcher [21]. For deterministic top-down unranked tree automata there is again the difference between the blind and the sensing variant. Furthermore, as nodes can have arbitrarily many children it is natural to consider two variants of sensing automata. The first variant is an online one: given the state and the label of its parent, the state of a child only depends on its label and the labels of its left-siblings. The variant is called online as child states are assigned when processing the child string in one pass from left to right. In contrast, the offline variant first reads the complete child string and only then assigns states to all children. All three models can again be characterized in terms of closure under specific forms of subtree exchange. These properties can be used to show that blind, online, and offline sensing are increasingly more expressive and that neither of the models is closed under union and complement. Interestingly, online sensing top-down tree automata suffice to express all DTDs and XML Schema Definitions. Furthermore, they correspond precisely to the unranked regular tree languages admitting one-pass preorder typing [14]. In this context, typing means the assignment of the correct state to each node. So, online sensing deterministic top-down tree automata capture precisely the schemas which can be validated and typed in a one-pass fashion. A difference with the binary case is that minimization is NP-complete for offline sensing top-down automata, while it is in PTIME for online sensing top-down automata. Minimization for blind automata is in NP but the precise complexity is unknown.

The Future. From a theoretical point of view, there is a schema language superior to XML Schema: Relax NG is more expressiveness than XML Schema and it is closed under the Boolean operations. Nevertheless, XML Schema is the language endorsed by W3C and therefore supported by the major database vendors. It constitutes deterministic top-down processing as its basic validation mechanism. As mentioned before, XML Schema lacks the most basic closure properties. From the viewpoint of model management [1] or schema integration, especially the inability to express the union of two schemas is a serious defect. From a formal language theory perspective, Jurvanen, Potthof, and Thomas proposed regular frontier checks as a general extension of deterministic top-down automata [12]. In particular, the acceptance condition is determined by a regular string language F over states added to the model. A tree is then accepted when the string formed by the states assigned to the frontier of the tree is in F. Although this formalism is expressive enough to define union and complement it is less convenient as an addition for a schema language. It would therefore be interesting to come up with a convenient top-down deterministic model closed under the Boolean operations. We discuss this and other future directions like optimization and automatic inference problems in the Conclusions.

Outline. The article is further organized as follows. In Section 2, we introduce the necessary notation. In Section 3 and 4, we discuss ranked and unranked deterministic top-down models, respectively. Finally, in Section 5, we consider regular frontier checks.

2 Preliminaries

2.1 An abstract notation for automata

We first explain the generic automata notation that we shall use throughout the paper. For a finite set S, we denote by $|S|$ its number of elements. By Σ we always denote a finite alphabet. We consider different types of data structures built from Σ like strings, binary trees, or unranked trees. We write D_Σ for the set of all data structures of the given type that can be built from Σ. For every $d \in D_\Sigma$, we shall define a set $\mathrm{Nodes}(d)$, a designated element $\mathrm{root}(d) \in \mathrm{Nodes}(d)$, and a designated set $\mathrm{Frontier}(d) \subseteq \mathrm{Nodes}(d)$. Here, $\mathrm{root}(d)$ will be the root of a tree or the first symbol of a string; $\mathrm{Frontier}(d)$ will be the set of leaves in a tree or the last symbol of a string.

To address automata in a uniform way for the different data structures, we first define them in abstract terms to instantiate them later operating on strings, trees, and unranked trees.

Definition 2.1. A *finite automaton* over Σ is a tuple

$$A = (\mathrm{States}(A), \mathrm{Alphabet}(A), \mathrm{Rules}(A), \mathrm{Init}(A), \mathrm{Final}(A)),$$

where $\mathrm{States}(A)$ is a finite set of states, $\mathrm{Alphabet}(A) = \Sigma$ is the finite alphabet, $\mathrm{Rules}(A)$ is a finite set of transition rules, $\mathrm{Init}(A) \subseteq \mathrm{States}(A)$ is the set of initial states, and $\mathrm{Final}(A) \subseteq \mathrm{States}(A)$ is the set of final states.

The size of A, denoted by $|A|$, is a natural number, which by default will be the number of states of A unless explicitly stated otherwise. A run of an automaton A on a data structure $d \in D_{\mathrm{Alphabet}(A)}$ will always be defined as some function of type $r : \mathrm{Nodes}(d) \to \mathrm{States}(A)$. For each kind of automaton, we shall define when a run is *accepting*. Then, the language $L(A)$ of an automaton is the set of data structures d that permit an accepting run.We call a finite automaton *unambiguous* if, for every d, there exists at most one accepting run of A on d.

We consider the following static analysis problems:

- **Emptiness:** Given a finite automaton A, is $L(A) = \varnothing$?
- **Containment:** Given two finite automata A and B, is $L(A) \subseteq L(B)$?
- **Minimization:** Given a finite automaton A and integer k, does there exist an automaton B (of the same class as A) such that $L(A) = L(B)$ and $|B| \leq k$?

In the remainder of the paper, we shall use the letters $a, b, c, \ldots$ to range over alphabet symbols and we shall use $p, q, \ldots$ to range over states.

2.2 Strings and trees

By $\mathbb{N}_0$ we denote the set of nonnegative integers and by $\mathbb{N}$ the set of positive integers. We call $a \in \Sigma$ a Σ-*symbol*. A Σ-*string* (or simply *string*) $w \in \Sigma^*$ is a finite sequence $a_1 \cdots a_n$ of Σ-symbols. We denote the empty string by ε.

The set of *positions*, or *nodes*, of a Σ-string w is $\mathrm{Nodes}(w) = \{1, \ldots, n\}$. The *root* of w is $\mathrm{root}(w) = 1$ and the *frontier* of w is $\mathrm{Frontier}(w) = \{n\}$. The *length* of w, denoted by $|w|$, is n. The label a_i of node i in w is denoted by $\mathrm{lab}^w(i)$.

A *tree domain* N is a non-empty, prefix-closed subset of $\mathbb{N}^*$ satisfying the following condition: if $ui \in N$ for $u \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $uj \in N$ for all j with $1 \leq j \leq i$. An *unranked* Σ*-tree* t (which we simply call tree in the following) is a mapping $t : \text{Nodes}(t) \to \Sigma$ where $\text{Nodes}(t)$ is a finite tree domain. The elements of $\text{Nodes}(t)$ are called the *nodes* of t. For $u \in \text{Nodes}(t)$, we call nodes of the form $ui \in \text{Nodes}(t)$ with $i \in \mathbb{N}$ the *children* of u (where ui is the ith child). The *root* of a tree is $\text{root}(t) = \varepsilon$ and the *frontier* of a tree is its set of nodes with no children, that is, $\text{Frontier}(t) = \{u \mid u1 \notin \text{Nodes}(t)\}$. For a tree t and a node $u \in \text{Nodes}(t)$, we denote the label $t(u)$ by $\text{lab}^t(u)$. If the root of t is labeled by a, that is, $\text{lab}^t(\varepsilon) = a$, and if the root has k children at which the subtrees $t_1, \ldots, t_k$ are rooted from left to right, then we denote this by $t = a(t_1 \cdots t_k)$. In the sequel, we adopt the following convention: when we write a tree as $a(t_1 \cdots t_n)$, we tacitly assume that all t_i's are trees. The *depth* of a node $i_1 \cdots i_n \in \mathbb{N}^*$ in a tree is $n + 1$. The *depth* of a tree is the maximum of the depths of its nodes. We denote the set of unranked Σ-trees by $\mathcal{T}_\Sigma$. By $\text{subtree}^t(u)$ we denote the *subtree of* t *rooted at* u. For two Σ-trees t_1 and t_2, and a node $u \in \text{Nodes}(t_1)$, we denote by $t_1[u \leftarrow t_2]$ the tree obtained from t_1 by replacing its subtree rooted at u by t_2. A *tree language* is a set of trees.

A *binary alphabet* or *binary signature* is a pair $(\Sigma, \text{rank}_\Sigma)$, where rank_Σ is a function from Σ to $\{0, 2\}$. The set of *binary* Σ*-trees* is the set of Σ-trees inductively defined as follows. When $\text{rank}_\Sigma(a) = 0$, then a is a binary Σ-tree. When $\text{rank}_\Sigma(a) = 2$ and t_1, t_2 are binary Σ-trees, then $a(t_1 t_2)$ is a binary Σ-tree.

2.3 Finite string automata

We instantiate our abstract notion of finite automata over strings:

Definition 2.2. A *finite string automaton (FSA)* over Σ is a finite automaton over Σ where $\text{Rules}(A)$ is a finite set of rules of the form $q_1 \xrightarrow{a} q_2$ with $q_1, q_2 \in \text{States}(A)$ and $a \in \text{Alphabet}(A)$.

A *run* of A on a string $w \in \text{Alphabet}(A)^*$ is a mapping $r : \text{Nodes}(w) \to \text{States}(A)$ such that

(i) there exists $q_0 \in \text{Init}(A)$ with $q_0 \xrightarrow{a} r(1)$ in $\text{Rules}(A)$ for $\text{lab}^w(1) = a$; and,

(ii) for every $i = 1, \ldots, |w| - 1$, it holds that $r(i) \xrightarrow{a} r(i + 1)$ in $\text{Rules}(A)$ where $\text{lab}^w(i + 1) = a$.

A run r is *accepting* if $r(|w|) \in \text{Final}(A)$. An FSA A is *deterministic* if it satisfies the following two conditions, implying that no string permits more than one run by A:

(i) $\mathrm{Init}(A)$ is a singleton; and,

(ii) for every $q_1 \in \mathrm{States}(A)$ and $a \in \mathrm{Alphabet}(A)$, there exists at most one rule $q_2 \in \mathrm{States}(A)$ such that $q_1 \xrightarrow{a} q_2$ is in $\mathrm{Rules}(A)$.

We denote by DFSA be the class of deterministic finite string automata.

2.4 Exchange properties for tree languages

We define several of the exchange properties for tree languages that we use in the following sections to characterize the expressive power of tree automata.

2.4.1 Path-closed languages

A well-known characterization of tree languages recognizable by a class of top-down deterministic tree automata is the one of *path closed* languages by Virágh [23]. The *path language of a tree* t, denoted $\mathrm{Path}(t)$, is the set of strings

$$\mathrm{lab}(\varepsilon) i_1 \mathrm{lab}(i_1) \quad \cdots \quad i_n \mathrm{lab}(i_1 \cdots i_n),$$

for nodes $i_1, i_1 i_2, \ldots i_1 \cdots i_n$ in $\mathrm{Nodes}(t)$.[1] The *path langauge of a tree language* L, denoted $\mathrm{Path}(L)$, then is the union of the path languages of its trees, that is, $\mathrm{Path}(L) = \bigcup_{t \in L} \mathrm{Path}(t)$. The *path closure* of a tree language L is defined as $\text{P-Closure}(L) = \{t \mid \mathrm{Path}(t) \subseteq \mathrm{Path}(L)\}$. Finally, a tree language L is *path-closed* when $\text{P-Closure}(L) \subseteq \mathrm{Path}(L)$.

Nivat and Podelski argued that path-closed languages can also be characterized using the following subtree exchange property [17].[2] A regular tree language L is path-closed if and only if, for every $t \in L$ and every node $u \in \mathrm{Nodes}(t)$,

$$\text{if } t[u \leftarrow a(t_1, \ldots, t_n)] \in L \text{ and } t[u \leftarrow a(s_1, \ldots, s_n)] \in L, \text{ then}$$
$$t[u \leftarrow a(t_1, \ldots, s_i, \ldots, t_n)] \in L \text{ for each } i = 1, \ldots, n.$$

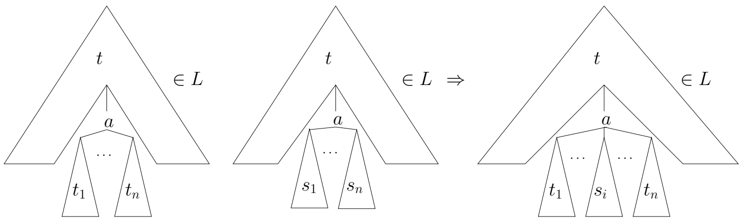
This subtree exchange closure for path-closed languages is illustrated in Figure 1. In the remainder of the article, when we say that a language is path-closed, we shall always refer to this closure under the just mentioned exchange property.

2.4.2 Guarded subtree exchange

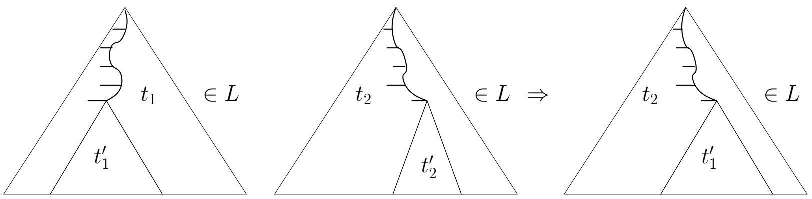
For a node $v = uk$ in a tree t with $k \in \mathbb{N}$, we denote by $\text{l-sib-str}^t(v)$ the string formed by the label of the v and the labels of its left siblings, that is, $\mathrm{lab}^t(u1) \cdots \mathrm{lab}^t(uk)$. By $\text{r-sib-str}^t(v)$ we denote the string formed by v and its right siblings, that is, $\mathrm{lab}^t(uk) \cdots \mathrm{lab}^t(un)$, if u has n children.

[1] We tacitly assume here that $\Sigma \cap \mathbb{N} = \varnothing$.

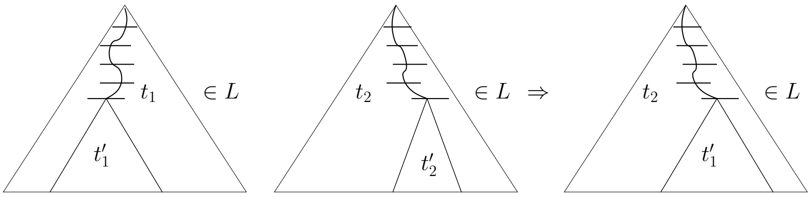
[2] Actually, Nivat and Podelski only considered path-closedness on ranked trees, but it is easy to see that the properties are also equivalent on unranked trees.



(a) Path-closed.



(b) Ancestor-left-sibling-closed.



(c) Spine-closed.

FIGURE 1. Various kinds of subtree exchange properties for tree languages.

We define $\text{l-sib-str}^t(\varepsilon) = \text{r-sib-str}^t(\varepsilon) = \text{lab}^t(\varepsilon)$. Let $v = i_1 i_2 \cdots i_\ell$ with $i_1, i_2, \ldots, i_\ell \in \mathbb{N}$. Let $\#$ and $\triangledown$ be two symbols not in Σ. By $\text{anc-l-sib-str}^t(v)$ we denote the ancestor-left-sibling-string

$$\text{l-sib-str}^t(\varepsilon)\#\text{l-sib-str}^t(i_1)\#\cdots\#\text{l-sib-str}^t(i_1 i_2 \cdots i_\ell),$$

formed by concatenating the left-sibling-strings of all ancestors of v, starting from the root. By $\text{spine}^t(v)$ we denote the ancestor-sibling-string

$$\text{l-sib-str}^t(\varepsilon)\triangledown\text{r-sib-str}^t(\varepsilon)\#\text{l-sib-str}^t(i_1)\triangledown\text{r-sib-str}^t(i_1)\#\cdots$$
$$\cdots\#\text{l-sib-str}^t(i_1 i_2 \cdots i_\ell)\triangledown\text{r-sib-str}^t(i_1 i_2 \cdots i_\ell)$$

formed by concatenating the left-sibling-strings and right-sibling strings of all ancestors of v, starting from the root.

We say that a tree language L is *ancestor-left-sibling-closed*[3] if whenever

[3] This property was called "closure under ancestor-sibling-guarded subtree exchange" in [14].

for two trees $t_1, t_2 \in L$ with nodes $u_1 \in \text{Nodes}(t_1)$ and $u_2 \in \text{Nodes}(t_2)$, $\text{anc-l-sib-str}^{t_1}(u_1) = \text{anc-l-sib-str}^{t_2}(u_2)$ implies $t_1[u_1 \leftarrow \text{subtree}^{t_2}(u_2)] \in L$. We say that L is *spine-closed* if $\text{spine}^{t_1}(u_1) = \text{spine}^{t_2}(u_2)$ implies $t_1[u_1 \leftarrow \text{subtree}^{t_2}(u_2)] \in L$. The latter notions are illustrated in Figure 1.

3 Top-down automata on binary trees

As we consider in this section automata over binary trees, we take Σ as a binary alphabet. We define two flavors of top-down determinism. The first is the traditional one, such as defined, for example, by Gecseg and Steinby [9] and in the on-line textbook TATA [7]. In brief, the label of the current symbol and the current state uniquely determine the states assigned to the children of the current symbol (Definition 3.1). The second notion of top-down determinism is slightly more expressive. Here, the states assigned to the children of the current node are determined by the current node's label, the state assigned to the current node, *and the labels of the children* (Definition 3.2). The latter notion of top-down determinism is reminiscent to the notion of "l-r-determinism" studied by Nivat and Podelski [17], and similar notions of top-down determinism on unranked trees have been studied by Cristau, Löding, and Thomas [8] and by Martens [13]. We refer to the first kind of automata as blind and to the second as sensing.

Definition 3.1. A *blind top-down finite tree automaton (BTA)* is a finite automaton A such that $\text{Rules}(A)$ is a set of rules

$$(q, a) \rightarrow (q_1, q_2) \text{ or } (q, a) \rightarrow \varepsilon.$$

A *run* of A on a binary Σ-tree t is a mapping $r : \text{Nodes}(t) \rightarrow \text{States}(A)$ such that

(i) $r(\varepsilon) \in \text{Init}(A)$;

(ii) for each leaf node u with label a, $(r(u), a) \rightarrow \varepsilon$ is in $\text{Rules}(A)$; and

(iii) for each non-leaf node u with label a, $(r(u), a) \rightarrow (r(u1), r(u2))$ is in $\text{Rules}(A)$.

If a run exists, it is *accepting*. We say that a BTA is *(top-down) deterministic* if $\text{Init}(A)$ is a singleton and no two of its rules have the same left-hand sides.

Definition 3.2. A *sensing top-down finite tree automaton (STA)* is a finite automaton A such that $\text{Rules}(A)$ is a set rules of the form

$$a \rightarrow q \quad \text{or} \quad q(a_1, a_2) \rightarrow (q_1, q_2).$$

For an STA A, we have that $\text{Init}(A) = \{q \mid a \rightarrow q \in \text{Rules}(A)\}$. A *run* of A on a binary Σ-tree t is a mapping $r : \text{Nodes}(t) \rightarrow \text{States}(A)$ such that

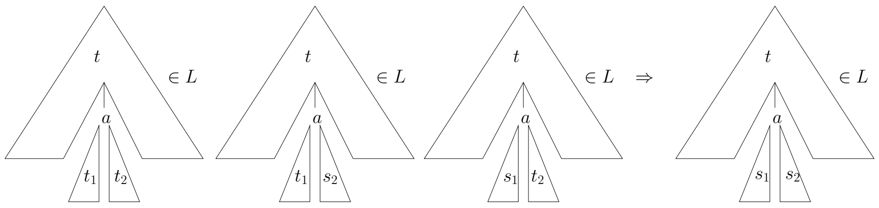


FIGURE 2. Closure property for homogeneous tree languages.

(i) if $r(\varepsilon) = q$ and $\text{lab}(\varepsilon) = a$ then there is a rule $a \to q \in \text{Rules}(A)$, and

(ii) for each non-frontier node u, if $r(u) = q$, $\text{lab}(u1) = a_1$, and $\text{lab}(u2) = a_2$, then there is a rule $q(a_1, a_2) \to (r(u1), r(u2))$ in $\text{Rules}(A)$.

The run is *accepting* if, for each leaf node u, $r(u) \in \text{Final}(A)$. We say that an STA is *deterministic* if no two of its rules have the same left-hand sides.

3.1 Relative expressive power

It is well-known that top-down automata cannot recognize all regular tree languages. In this section, we compare several forms of top-down determinism that have been investigated with respect to their expressive power.

3.1.1 Homogeneous languages

Nivat and Podelski defined a notion of top-down determinism that they called *l-r-determinism.* This form of determinism will not be treated very deeply in this article, as it does not correspond to the order in which one would like to process trees in an XML context. We use their characterization in terms of closure under subtree exchange to formally argue this. Nivat and Podelski define a BTA A to be l-r-deterministic if whenever $(q, a) \to (q_1, q_2)$ and $(q, a) \to (q_1', q_2')$ is in $\text{Rules}(A)$ then

- $q_1 \neq q_1'$ implies that $L(A[q_2]) \cup L(A[q_2']) = \varnothing$ and
- $q_2 \neq q_2'$ implies that $L(A[q_1]) \cup L(A[q_1']) = \varnothing$.

Here, for $q = q_1, q_2, q_1', q_2'$, $A[q]$ denotes automaton A in which $\text{Init}(A) = \{q\}$. We shall, however, focus on a characterization of the languages accepted by l-r-deterministic tree automata which is, for our purpose, more workable.

A regular tree language L is *homogeneous* if, whenever $t[u \leftarrow a(t_1, t_2)] \in L$, $t[u \leftarrow a(s_1, t_2)] \in L$, and $t[u \leftarrow a(t_1, s_2)] \in L$, then also $t[u \leftarrow a(s_1, s_2)] \in L$. This closure under subtree exchange is illustrated in Figure 2.

3.1.2 The characterization

We characterize the expressiveness of the tree automata models by the closure properties introduced in Section 2.4.

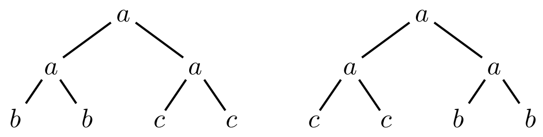


FIGURE 3. A homogeneous language that is not spine-closed.

Theorem 3.3 (Characterization Theorem). A regular tree language L is recognizable by

(1) a deterministic BTA if and only if L is path-closed.

(2) an l-r-deterministic tree automaton if and only if L is homogeneous.

(3) a deterministic STA if and only if L is spine-closed.

Theorem 3.3(1) is known from, e.g., Virágh [23] and from Gecseg and Steinby [10]. Theorem 3.3(2) is Theorem 2 in the work by Nivat and Podelski [17]. Finally, Theorem 3.3(3) is proved by Cristau, Löding, and Thomas [8] and by Martens [13] for more general unranked tree automata with this form of top-down determinism. It should be noted that Cristau et al. did not explicitly use a subtree exchange property for spine-closedness but an equivalent closure property that considers the spine language of a tree (as in the original definition of path-closedness).

Corollary 3.4.

(1) l-r-deterministic tree automata are strictly more expressive than deterministic BTAs.

(2) Deterministic STAs are strictly more expressive than deterministic BTAs.

(3) Deterministic STAs and l-r-deterministic tree automata are incomparable w.r.t. expressive power.

Proof. (1) It is easy to see that every path-closed language is homogeneous. Furthermore, the language $\{a(b,b), a(c,c)\}$ is homogeneous but not path-closed.
(2) It is easy to see that every path-closed language is also spine-closed. Furthermore, the language $\{a(b,b), a(c,c)\}$ is spine-closed but is not path-closed.
(3) The language $\{a\big(a(b,b), a(c,c)\big), a\big(a(c,c), a(b,b)\big)\}$ is homogeneous but not spine-closed (see also Figure 3). The language $\{a(b,b), a(b,c), a(c,b)\}$ is spine-closed but not homogeneous. Q.E.D.

3.1.3 L-R-determinism versus top-down state assignment

Figure 3 depicts a finite language L which is homogeneous but not spine-closed. So, L can be recognized by an l-r-deterministic tree automaton but not by a deterministic STA.

One easily obtains infinite languages with this property. Indeed, let L_b and L_c be the set of trees in which every internal node is labeled a and every leaf is labeled b and c, respectively. The language L_{bc} now consists of all trees $a(t_b, t_c)$ and $a(t_c, t_b)$ for which $t_b \in L_b$ and $t_c \in L_c$. Clearly, L_{bc} is homogeneous.

We now want to argue informally that, for any tree automaton A recognizing L_{bc}, the state that A assigns to each of the two children of the root in an accepting run cannot be determined without looking *arbitrarily deep* into at least one subtree of the root. In other words, this means that there is at least one child u of the root such that A needs to investigate the subtree rooted at u before assigning a state to u. This is something what is not commonly associated with "top-down determinism".

Let A be a tree automaton that recognizes the language L_{bc}. Let n be an arbitrarily large natural number and let $a(t_b, t_c)$ be a tree in L_{bc} such that every path from root to leaf in t_b and t_c has length at least $n + 1$. This way, t_b and t_c are identical up to depth n. Towards a contradiction, suppose that A does not investigate t_b or t_c arbitrarily deep, i.e., not up to depth n, before assigning a state to the root of t_b (the argument for t_c is the same). More formally, assume that the state A assigns to the root of t_b is functionally determined by the structure of t_b and t_c up to depth at most $n - 1$. Let r_1 be an accepting run of A on $a(t_b, t_c)$ and let r_2 be an accepting run of A on $a(t_c, t_b)$. As A does not investigate t_b or t_c arbitrarily deep, r_1 assigns the sames state to the root of t_b in $a(t_b, t_c)$ as r_2 assigns to the root of t_c in $a(t_c, t_b)$. As A is a tree automaton, it is now easy to see that $a(t_c, t_c)$ is also in $L(A)$, with the accepting run that behaves as r_2 on the left copy of t_c and as r_1 on the right copy of t_c. This contradicts that A accepts L_{bc}.

Therefore, our focus in the remainder of the article will be on deterministic BTAs and deterministic STAs, rather than l-r-deterministic tree automata.

3.2 Closure properties

The characterization theorem can easily be used to show that deterministic top-down tree automata are not closed under complement and union.

Theorem 3.5.

(1) Deterministic BTAs and deterministic STAs are closed under intersection.

(2) Deterministic BTAs and deterministic STAs are not closed under complement or union.

Proof. (1) This follows immediately from the standard product construction for tree automata. One merely has to observe that the intersection construction preserves the determinism constraint for BTAs and STAs.
(2) These results can be proved quite directly from the characterizations in Theorem 3.3. Indeed, let L_b (resp., L_c) be the tree language over alphabet $\{a, b, c\}$ in which every internal node (i.e., with two children) is labeled a and every leaf is labeled b (resp., c). The languages L_b and L_c are easily seen to be recognizable by deterministic BTAs.

On the other hand, the union $L_b \cup L_c$, the set of all trees in which every internal node is labeled a and either all leaves are labeled b or all leaves are labeled c is *not* spine-closed. Hence, $L_b \cup L_c$ is not recognizable by a deterministic STA, which means that deterministic BTAs and deterministic STAs are not closed under union. From closure under intersection and non-closure under union we can readily conclude non-closure under complement.

Q.E.D.

3.3 Static analysis

In this section, we shall prove the following theorem:

Theorem 3.6.

(1) Emptiness is in PTIME for BTAs and STAs.

(2) Containment is in PTIME for deterministic BTAs and deterministic STAs.

(3) Minimization is in PTIME for deterministic BTAs and deterministic STAs.

Proof. (1) It is well-known that emptiness is in PTIME for (non-deterministic bottom-up) tree automata in general [7]. Therefore, emptiness is also in PTIME for deterministic BTAs and deterministic STAs.
(2) It is easy to see that deterministic BTAs and deterministic STAs and intersections thereof are in fact unambiguous tree automata. The result now follows from the work by Seidl, who proved that equivalence of unambiguous tree automata is in PTIME [18].
(3) For deterministic BTAs, this follows from the work by Gecseg and Steinby [9]. Although their work does not explicitly concern complexity, they prove that minimization for deterministic BTAs can be polynomially reduced to equivalence/containment for deterministic BTAs. As containment for deterministic BTAs is in PTIME by part (2), we also have that minimization is in PTIME.

(1) *Reduce* A, that is,

  (a) remove all states q from A for which $L(A[q]) = \varnothing$; and then

  (b) remove all states q from A which are not reachable from $\text{Init}(A)$.

(2) Test, for each $p \neq q$ in $\text{States}(A)$, whether $L(A[p]) = L(A[q])$.
If $L(A[p]) = L(A[q])$, then

  (a) replace all occurrences of p in the definition of A by q and

  (b) remove p from A.

FIGURE 4. The Minimization Algorithm.

To explain their algorithm, we start by discussing a minor optimization matter for tree automata. For an automaton A and $q \in \text{States}(A)$ we denote by $A[q]$ the language accepted by A when $\text{Init}(A) = \{q\}$.[4] We say that q is *reachable* in A if one of the following holds:

- $q \in \text{Init}(A)$ or
- p is reachable and there is a rule of the form $(p, a) \to (q_1, q_2)$ or $p(a_1, a_2) \to (q_1, q_2)$ in $\text{Rules}(A)$, where $q = q_1$ or $q = q_2$.

We now say that A is *reduced* if, every state q is *useful*, that is, q is reachable and $L(A[q]) \neq \varnothing$. Algorithmically, one would convert a tree automaton into a reduced tree automaton by first removing all the states q for which $L(A[q]) = \varnothing$ and then removing all the states that are not reachable. The order in which these two steps are performed is important, as the other order does not necessarily produce a reduced automaton.

The following observation states that a state is useful if and only if it can be used in some accepting run of the automaton.

Observation 3.7. Let A be a tree automaton and $q \in \text{States}(A)$. Then, q is useful if and only if there exists a tree $t \in L(A)$, an accepting run r of A on t, and a node $u \in \text{Nodes}(t)$ such that $r(u) = q$.

The algorithm of Gecseg and Steinby is now informally presented in Figure 4.

Interestingly, for deterministic STAs, it seems that one can likewise use the algorithm of Figure 4 for minimization. It only has to be shown that,

[4] If A is an STA, we require in addition that every rule $a \to p$ is replaced by $a \to q$.

given a deterministic STA, the algorithm returns a minimal deterministic STA. Thereto, let $A_{\min}$ be the automaton obtained by applying the above minimization algorithm on a deterministic STA A. Formally, we need to prove that

(a) $A_{\min}$ is a deterministic STA;

(b) $L(A_{\min}) = L(A)$; and that

(c) the number of states of $A_{\min}$ is indeed minimal.

To show (a), observe that, in step (1) of the algorithm, we only remove states. Hence, no non-determinism is introduced in step (1). In step (2), non-determinism can be introduced by overwriting occurrences of p with q. However, the following observation, which is easy to show by contraposition, proves that this non-determinism is removed further on in the algorithm.

Observation 3.8. Let p and q be two states such that $L(A[p]) = L(A[q])$ and let $p(a_1, a_2) \to (p_1, p_2)$ and $q(a_1, a_2) \to (q_1, q_2)$ be two transition rules of A. Then $L(A[p_1]) = L(A[q_1])$ and $L(A[p_2]) = L(A[q_2])$.

To show (b), observe that, in step (1), we only remove states that cannot be used in a successful run of A (Observation 3.7). Hence, this does not alter the language accepted by A. In step (2), we replace states p in A with states q that define the same language. The following observation is easy to prove:

Observation 3.9. Let p and q be two states such that $L(A[p]) = L(A[q])$. Let A' be obtained from A by replacing all occurrences of p in the definition of A by q, and by removing q. Then $L(A) = L(A')$.

It remains to show (c), which is a bit more involved. First, we introduce the following concept. We say that a finite tree automaton A over Σ *has spine-based runs* if there is a (partial) function

$$f : (\Sigma \cup \{\#, \triangledown\})^* \to \text{States}(A)$$

such that, for each tree $t \in L(A)$, for each node $v \in \text{Nodes}(t)$, and for each accepting run r of A on t, we have that

$$r(v) = f(\text{spine}^t(v)).$$

Observation 3.10. Every deterministic STA has spine-based runs.

Proof. Let A be a deterministic STA. We assume w.l.o.g. that A is reduced. We define the function $f : (\Sigma \cup \{\#, \triangledown\})^* \to \text{States}(A)$ inductively as follows: for each $a \in \Sigma$, $f(a \triangledown a) = q$, for the unique q such that $a \to q$ is a rule in A.

Further, for every string $w_0\#w_1a\triangledown aw_2$ with $w_0 \in (\Sigma \cup \{\#, \triangledown\})^*$, $w_1, w_2 \in \Sigma \cup \{\varepsilon\}$, and $a \in \Sigma$, we define $f(w_0\#w_1a\triangledown aw_2) = q$ where $f(w_0) = p$ and q is the unique state such that the following holds. If $w_1 = \varepsilon$, q is the unique state such that $p(a, w_2) \to (q, q') \in \text{Rules}(A)$, and if $w_2 = \varepsilon$, then q is the unique state such that $p(w_1, a) \to (q', q) \in \text{Rules}(A)$. As A is a reduced deterministic STA, f is well-defined and induces a spine-based run. Q.E.D. (Observation 3.10)

Observation 3.11. Let A_1 and A_2 be equivalent deterministic STAs and let $t \in L(A_1) = L(A_2)$. Let r_1 and r_2 be the unique runs of A_1 and A_2 on t, respectively, and let u be a node in t. Then $L(A_1[r_1(u)]) = L(A_2[r_2(u)])$.

Proof. Let p and q be $r_1(u)$ and $r_2(u)$, respectively. If $|L(A_1[p])| = |L(A_2[q])| = 1$, the proof is trivial. We show that $L(A_1[p]) \subseteq L(A_2[q])$. The other inclusion follows by symmetry.

Towards a contradiction, assume that there exists a tree $t_0 \in L(A_1[p]) - L(A_2[q])$. As A_1 is reduced, there exists a tree T_0 in $L(A_1)$, such that

- t_0 is a subtree of T_0 at some node v; and,
- $r'_1(v) = p$, where r'_1 is the unique run of A_1 on T_0.

As $r_1(u) = p = r'_1(v)$, the tree $t_3 = t[u \leftarrow t_0]$ is also in $L(A_1)$. As A_1 and A_2 are equivalent, t_3 is also in $L(A_2)$. Notice that u has the same spine in t and in $t_3 = t[u \leftarrow t_0]$. By Observation 3.10, A_2 has spine-based runs, which implies that $r'_2(u) = q$ for the unique run r'_2 of A_2 on t_3. Therefore, $t_0 \in L(A_2[q])$, which leads to the desired contradiction. Q.E.D. (Observation 3.11)

The next observation states that every equivalent minimal deterministic STA is equally large as $A_{\min}$.

Observation 3.12. If A_0 is a minimal deterministic STA for $L(A_{\min})$, then $|A_0| = |A_{\min}|$.

Proof. As A_0 is minimal, we know that A_0 is reduced and that $|A_0| \leq |A_{\min}|$. As $A_{\min}$ is the output of the minimization algorithm, $A_{\min}$ is reduced as well.

We only have to prove that $|A_{\min}| \leq |A_0|$. Towards a contradiction, assume that $|\text{States}(A_{\min}))| > |\text{States}(A_0)|$. For every state $q \in \text{States}(A_{\min})$, let $t^q_{\min} \in L(A_{\min})$ be a tree and $u^q_{\min} \in \text{Nodes}(t^q_{\min})$ such that $r^q_{\min}(u^q_{\min}) = q$ for the unique accepting run $r^q_{\min}$ of $A_{\min}$ on $t^q_{\min}$. Moreover, let, for every such $t^q_{\min}$, r^q_0 be the unique accepting run r^q_0 of A_0 on $t^q_{\min}$.

According to the Pigeon Hole Principle, there exist two states $p \neq q \in \text{States}(A_{\min})$ such that $r^p_0(u^p_{\min}) = r^q_0(u^q_{\min}) = p_0$, for some $p_0 \in \text{States}(A_0)$. From Observation 3.11, it now follows that $L(A_{\min}[p]) = L(A_0[p_0]) =$

$L(A_{\min}[q])$. This contradicts that $A_{\min}$ is the output of the minimization algorithm, as there still exist two states for which step (2) must be performed. Q.E.D. (Observation 3.12)

This concludes the proof of the theorem. Q.E.D. (Theorem 3.6)

4 Top-down automata on unranked trees

The definition of unranked tree automata dates back to the work of Thatcher [21]. Unranked tree automata use $\mathcal{T}_\Sigma$ (that is, unranked Σ-trees) as their data structure. For convenience, we sometimes abbreviate "unranked tree automaton" by UTA in this section. We start by defining blind top-down deterministic unranked tree automata, which generalize the determinism in BTAs to unranked trees. Blind top-down deterministic unranked automata are, e.g., defined in [5] under the name of *top-down deterministic* automata.

Definition 4.1. A *blind top-down deterministic unranked tree automaton (BUTA)* over Σ is a finite automaton A over Σ in which Rules(A) is a set of rules of the form

$$a \to p \qquad \text{or} \qquad (q, a) \to B$$

such that $\text{Init}(A) = \{p \mid a \to p \in \text{Rules}(A)\}$ is a singleton and B is a deterministic FSA over States(A) with the property that, for each $i \in \mathbb{N}$, $L(B)$ contains at most one string of length i. Furthermore, for each $q \in$ States(A) and $a \in$ Alphabet(A), Rules(A) contains at most one rule of the form $(q, a) \to B$.

A *run* of A on a tree t is a labeling $r : \text{Nodes}(t) \to \text{States}(A)$ such that

- if $\text{lab}(\varepsilon) = a$ and $r(\varepsilon) = q$ then $a \to q \in \text{Rules}(A)$ and,
- for every node $u \in \text{Nodes}(t)$ such that $\text{lab}(u) = a$, $r(u) = q$, and u has n children, there is a rule $(q, a) \to B$ such that B accepts $r(u1) \cdots r(un)$.

Notice that, in the second bullet, the criterion that u is a leaf reduces to $\varepsilon \in L(B)$. Therefore, each run that satisfies the above conditions is *accepting.*

Notice that the regular languages defined by the above Bs are very restricted. Indeed, as pointed out in [16], Shallit [19] has shown that such regular languages are finite unions of regular expressions of the form xy^*z where $x, y, z \in \Sigma^*$.

Just as in the ranked case, blind top-down determinism is the most widely accepted form of top-down determinism. However, in a context such as XML, blind top-down determinism is not very useful as its expressiveness is very limited. We therefore also investigate 'sensing' extensions that can read labels of child nodes before assigning them states.

The following definition is the generalization of determinism for STAs. In a similar effort to generalize determinism for STAs to unranked trees, Cristau et al. [8] and Martens [13] define models with the same expressive power as this one.

Definition 4.2. An *offline sensing top-down deterministic unranked tree automaton (offline SUTA)* is a finite automaton A in which Rules(A) is a set of rules of the form

$$a \rightarrow p \qquad \text{or} \qquad q \rightarrow B_q,$$

where the automata B_q are FSAs over Σ and use the states of A as their state set. That is, $\text{States}(B_q) = \text{States}(A)$. Furthermore, all the B_q have same the final states and the same transition rules, that is, for all $q_1, q_2 \in \text{States}(A)$, $\text{Final}(B_{q_1}) = \text{Final}((B_{q_2})$ and $\text{Rules}(B_{q_1}) = \text{Rules}(B_{q_2})$. In short, the only difference between the automata B_q is their choice in initial states.[5] Furthermore,

- for each $a \in \text{Alphabet}(A)$ there is at most one rule of the form $a \rightarrow p$,
- for each $q \in \text{States}(A)$, there is at most one rule $q \rightarrow B_q$, and
- for each rule $q \rightarrow B_q$, B_q is an *unambiguous FSA*.

We define Init(A) to be $\{p \mid a \rightarrow p \in \text{Rules}(A)\}$ and we require that $\text{Init}(A) \subseteq \text{Final}(B_q)$, for each state q.

A *run* r of A on a tree t is a labeling $r : \text{Nodes}(t) \rightarrow \text{States}(A)$ such that

- if $\text{lab}(\varepsilon) = a$ and $r(\varepsilon) = q$ then $a \rightarrow q \in \text{Rules}(A)$ and,
- for every node $u \in \text{Nodes}(t)$ such that $\text{lab}(u) = a$, $r(u) = q$, and u has n children, there is a rule $q \rightarrow B_q$ such that $r(u1) \cdots r(un)$ is an accepting run of B_q on $\text{lab}(u1) \cdots \text{lab}(un)$.

As with BUTAs, the criterion that u is a leaf reduces to $\varepsilon \in L(B)$ in the second bullet. Therefore, each run that satisfies the above conditions is *accepting.*

The restriction to unambiguous FSAs actually ensures that the complete child string can be read prior to the assignment of states. We note that the above mentioned work [8, 13], where "sensing top-down determinism" is

[5] A similar sharing of states is used in *stepwise tree automata*, which were used for defining a clean notion of *bottom-up determinism* for unranked tree automata [15].

simply called "top-down determinism", employs slightly more involved but equivalent definitions in terms of expressive power.

In Section 4.2, we shall see that, in contrast to the ranked case, offline sensing top-down determinism is in fact too powerful for efficient static analysis. In particular, minimization will turn out to be NP-hard for offline sensing deterministic automata. We therefore discuss *online sensing*, an intermediate form of top-down determinism which is also known under the name of *restrained competition* for *extended DTDs*.[6] This restriction will turn out to be more expressive than blind top-down determinism, while retaining the desirable complexities for static analysis.

Definition 4.3. An *online sensing top-down deterministic unranked tree automaton (online SUTA)* is an offline SUTA with the difference that, for each rule $q \to B_q$, B_q is a *deterministic FSA*.

The restriction to deterministic FSAs ensures that states have to be assigned to child nodes when processing them from left to right.

4.1 Relative expressive power

Again, we characterize the expressiveness of the formalisms in terms of subtree exchange properties.

Theorem 4.4. An (unranked) regular tree language L is recognizable by

1. a BUTA if and only if L is path-closed.
2. an online SUTA if and only if L is ancestor-sibling-closed.
3. an offline SUTA if and only if L is spine-closed.

The proof of Theorem 4.4(1) is analogous to the ranked case. Theorem 4.4(2) and Theorem 4.4(3) are proved by Martens et al. [13, 14].

The next corollary then immediately follows:

Corollary 4.5.

1. BUTAs are strictly less expressive than online SUTAs.
2. Online SUTAs are strictly less expressive than offline SUTAs.

[6] Extended DTDs or EDTDs are a grammar-based alternative to tree automata which have been investigated in the context of XML schema languages [13, 14].

4.2 Static analysis

Theorem 4.6.

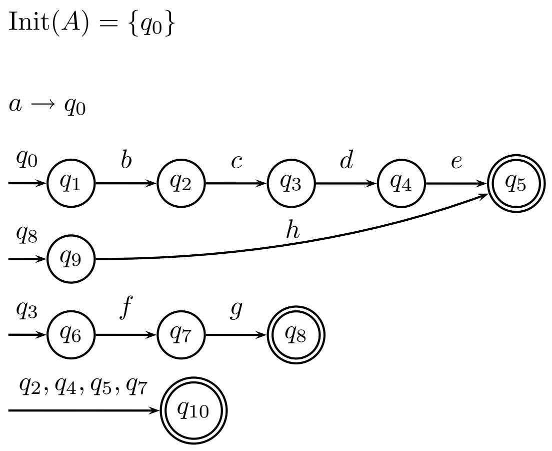
1. Emptiness is in PTIME for BUTAs, online SUTAs and offline SUTAs.
2. Containment is in PTIME for BUTAs, online SUTAs and offline SUTAs.
3. Minimization is in PTIME for online SUTAs.
4. Minimization is NP-complete for offline SUTAs.

Proof. (1) This follows from the result that emptiness is in PTIME for non-deterministic unranked tree automata. (See, e.g., [13].)
(2) This follows from PTIME containment for *unambiguous* (ranked) tree automata [18]. For example, when translating an offline SUTA to a ranked tree automaton through the well-known *first-child next-sibling encoding*, one obtains an unambiguous ranked tree automaton. Containment of the unranked tree automata can then be decided by testing containment for the unambiguous ranked automata.
(3) We can reduce to Theorem 3.6(3) by means of the unranked-versus-ranked encoding enc and decoding dec illustrated in Figure 5. We explain intuitively how a run of an online SUTA A for L translates to a run of a deterministic STA enc(A) for enc(L). We assume w.l.o.g. that A is reduced. Assignment of initial states to the root of the trees is the same for both automata. Furthermore, the transition rules translate as follows. For each $q \in \text{States}(A)$ and $a \in \text{Alphabet}(A)$, Rules(enc(A)) contains

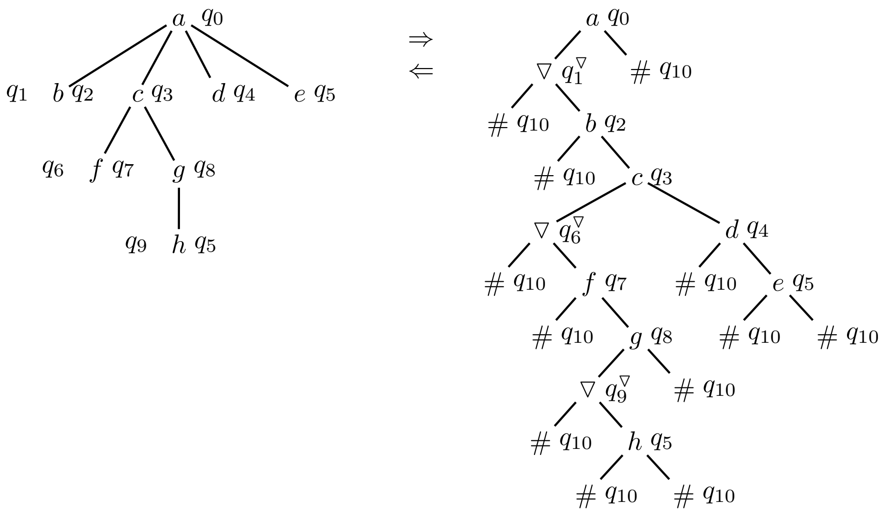
- $a \to q$ if $a \to q$ in Rules(A);
- $q(\nabla, \#) \to (p^\nabla, q_{\text{leaf}})$ if $\text{Init}(B_q) = \{p\}$ and $q \in \text{Final}(B_q)$;
- $q(\nabla, a) \to (p^\nabla, q')$ if $\text{Init}(B_q) = \{p\}$ and $q \xrightarrow{a} q' \in \text{Rules}(B_q)$;
- $q(\#, a) \to (q_{\text{leaf}}, q')$ if B_q accepts ε and $q \xrightarrow{a} q' \in \text{Rules}(B_q)$;
- $q^\nabla(\#, a) \to (q_{\text{leaf}}, q')$ if $q \xrightarrow{a} q' \in \text{Rules}(B_q)$; and
- $q(\#, \#) \to (q_{\text{leaf}}, q_{\text{leaf}})$ if B_q accepts ε and q is a final state in B_q.

Here, q_{leaf} is a new state not occurring in States(A). The states q^∇ are copies of states q in A that can only be assigned to the ∇-labeled nodes in the encoding. The encoded automaton always assigns q_{leaf} to leaf symbols. Hence, $\text{Final}(\text{enc}(A)) = q_{\text{leaf}}$. Figure 5 illustrates an automaton A, an accepting run of A on a tree t, and an accepting run of enc(A) on enc(t).

It is easy to see that this encoding preserves determinism. The decoding, however, would not preserve determinism in general, as the initial


$\text{Init}(A) = \{q_0\}$
$a \to q_0$
q_0 q_1 b q_2 c q_3 d q_4 e q_5
q_8 q_9 h
q_3 q_6 f q_7 g q_8
q_2, q_4, q_5, q_7 q_{10}

(a) Automaton A accepting the tree in Figure 5(b).


$a\ q_0$
$q_1\ b\ q_2$ $c\ q_3$ $d\ q_4$ $e\ q_5$
$q_6\ f\ q_7$ $g\ q_8$
$q_9\ h\ q_5$
$\Rightarrow$
$\Leftarrow$
$\triangledown\ q_1^\triangledown$ $\#\ q_{10}$
$b\ q_2$
$c\ q_3$
$\triangledown\ q_6^\triangledown$ $d\ q_4$
$f\ q_7$ $e\ q_5$
$g\ q_8$
$\triangledown\ q_9^\triangledown$
$h\ q_5$

(b) An unranked tree and its ranked encoding.

Figure 5. Encoding of unranked to binary trees (and back) that links deterministic STAs to online SUTAs. Letters $a, \dots, h$ represent alphabet symbols and $\{q_0, \dots, q_{10}\}$ represent states of an accepting run.

states $\text{Init}(B_q)$ might not be unique. It can be shown, however, that if L is ancestor-sibling closed, the decoding of a minimal deterministic STA for $\text{enc}(L)$ is always deterministic.

In order to give the relation between the minimal sizes of A and $\text{enc}(A)$, we need a few parameters. We call a state q of A a *sink state*, when no rules of the form $q \to B$ occur in A and no B has a rule $q \xrightarrow{a} q'$ for some a. For example, the state q_{10} in Figure 5 is such a sink state. We define $\text{sink}(A) = 0$ if A has such a sink state and $\text{sink}(A) = 1$ otherwise. Furthermore, let $\text{trans-init}(A)$ be the number of states p such that $\{p\} = \text{Init}(B_q)$ for some q and p has an incoming transition.

Observation 4.7. There exists an online SUTA of size k for $L(A)$ if and only if there exists a deterministic STA of size $k + \text{sink}(A) + \text{trans-init}(A)$ for $L(\text{enc}(A))$.

The reasons for the difference in sizes concerning the sink state and the trans-init states are as follows. If A contains a sink state q, then $\text{enc}(A)$ could use this sink state instead of q_{leaf} to label all the #-leaves in the encoding. Furthermore, in the encoding, each ∇ node is labeled by a copy q^{∇} of a state q, which introduces extra states for $\text{enc}(A)$. However, if q contains an incoming transition in A (and A is reduced), then both q and q^{∇} appear in the minimal automaton for $L(\text{enc}(A))$.
(4) We first argue that minimization for offline SUTAs is in NP. To this end, observe that, given an offline SUTA A and an integer k, an NP algorithm can guess an offline SUTA B of size at most k and test in PTIME (according to Theorem 4.6(2)) whether A and B define the same language.

For the NP lower bound, we reduce from the minimization problem for unambiguous FSAs, which is shown to be NP-complete by Jiang and Ravikumar [11]. Observe that, in the proof of Jiang and Ravikumar [11, Theorem 3.1], it is shown that minimization is already NP-hard for unambiguous FSAs that *only accept strings of length two.* As FSAs that only accept strings of length two have a *sink state*, i.e., a state with no outgoing transitions, this simplifies our reduction.

Thereto, let U be an unambiguous FSA that only accepts strings of length two and let k be an integer. We construct an offline SUTA A and an integer ℓ such that there exists an equivalent unambiguous FSA for $L(U)$ of size at most k if and only if there exists an offline SUTA for $L(A)$ of size at most ℓ.

Let r be a symbol not occurring in $\text{Alphabet}(U)$. Intuitively, A will accept the trees $r(w)$ such that $w \in L(U)$. We define $\text{States}(A) = \text{States}(U) \uplus \{q_0\}$, $\text{Alphabet}(A) = \text{Alphabet}(U) \uplus \{r\}$, and the rules of A are defined as

- $r \to q_0$,
- $(q_0, r) \to U$, and

- $(q, a) \to E$, for every $q \in \text{States}(U)$ and $a \in \text{Alphabet}(U)$,

where E is the UFA with $\text{States}(E) = \{q_f\}$ and $L(E) = \{\varepsilon\}$. Here, q_f is a state in $\text{Final}(U)$ which is reachable in U from an initial state of U. Finally, $\ell = k + 1$.

We need to argue that the reduction is correct. It is easy to see that A accepts $\{r(w) \mid w \in L(U)\}$.

We need to prove that there is an unambiguous FSA for $L(U)$ of size at most k if and only if there is an offline SUTA for $L(A)$ of size at most ℓ. From left to right, let U' be an unambiguous FSA of size at most k for $L(U)$. Then, A', constructed from U' in the same way as A is constructed from U is an offline SUTA for $L(A)$ of size at most ℓ. From right to left, let A' be an offline SUTA for $L(A)$ of size at most ℓ. W.l.o.g., we can assume that A' is reduced. As A' is an offline SUTA, A' has a unique state q_0 which is used in the rule $r \to q_0$. Now consider the transition rule of q_0, i.e., $q_0 \to U''$ in $\text{Rules}(A)$. Clearly, U'' accepts $L(U)$. As A' only accepts trees of depth two, we have that q_0 has no incoming or outgoing transitions in the definition of U''. (Otherwise, as A' is reduced, trees can be constructed that are also in $L(A')$ and have depth larger than two, contradicting that $L(A') = L(A)$.) Therefore, the unambiguous FSA U', obtained from U'' by removing state q_0 also recognizes $L(U)$ and has size at most k. Q.E.D.

A similar result as Theorem 4.6(3) was also proved in the context of extended DTDs in [15]. To the best of our knowledge, the precise complexity of minimization for BUTAs is still unknown. It is in NP, as testing equivalence between BUTAs is in PTIME.

4.3 Closure properties

The same closure properties hold for the deterministic unranked tree automata as for the ranked tree automata we defined. The witness languages for non-closure are analogous to the ones in Section 3.2.

5 Regular frontier checks

In this section, we revisit the notion of regular frontier checks as a theoretical tool to close top-down deterministic languages under Boolean operations. We apply regular frontier checks to *unranked* automata.

To this end, we assume that the *frontier* of a tree is no longer an unordered set, but ordered from left to right. That is, we assume the lexicographical ordering $<$ on $\text{Frontier}(t)$.

Definition 5.1. A *top-down deterministic unranked tree automaton with regular frontier check (FC-UTA)* over alphabet Σ is a (blind, online sensing, or offline sensing) top-down deterministic unranked tree automaton A over alphabet Σ, together with a regular language F over alphabet $\Sigma \times \text{States}(A)$.

A *run* of A on a tree t is defined precisely the same for blind, online sensing, or offline sensing unranked automata, respectively. A run r is *accepting* if $(\text{lab}(u_1), r(u_1)) \cdots (\text{lab}(u_n), r(u_n)) \in F$, where $\text{Frontier}(t) = \{u_1, \ldots, u_n\}$ with $u_1 < \cdots < u_n$.

On ranked trees, top-down tree automata with frontier checks are known to be closed under union, intersection, and complement [12]. On unranked trees, these results can be obtained analogously. Moreover, in order to obtain this closure, one does not even need arbitrary regular languages. Indeed, it is sufficient to consider locally threshold testable languages [22] with diameter $k = 1$.

Hence, FC-UTAs could be one candidate for closing schema languages for XML under the Boolean operations, thereby resolving the issues in model management or schema integration.

6 Conclusions and discussion

We presented an overview of top-down determinism in ranked and unranked tree automata, and explored several connections between them. As many connections were to be expected, we start the conclusions with a discrepancy. This discrepancy is observed between the (ranked) deterministic sensing tree automata (STAs) and the (unranked) deterministic offline sensing tree automata (offline SUTAs). Although they are closely related — they have, e.g., the same expressive power on binary trees and their way of assigning states to nodes in a top-down fashion is quite similar — we have shown that *optimization*, i.e., state minimization, is easy for one class but hard for the other.[7] Indeed, whereas state minimization is in PTIME for STAs, it is NP-complete for offline SUTAs. When inspecting the NP-hardness proof, the difference becomes even more striking: it already holds for offline SUTAs recognizing binary trees.

It thus follows that the determinism in offline SUTAs is actually not a very suitable notion for "top-down determinism" on unranked trees. Similarly as has been argued for the "standard" notion of *bottom-up determinism* on unranked trees [15], determinism in offline SUTAs corresponds more closely to *unambiguousness* rather than true determinism.[8]

On the positive side, the determinism in *online SUTAs* seems to be more suitable. Online SUTAs have been investigated in the context of XML schema languages under the name of *restrained competition EDTDs* and are already attributed to have desirable static analysis properties, while being more expressive than the core of XML Schema [14]. It is even decidable

[7] If PTIME $\neq$ NP.

[8] Of course, this is because our definition of determinism in offline SUTAs use unambiguous automata. However, we feel that similar problems will arise when investigating minimization for the equally expressive models presented in [8, 13].

(EXPTIME-complete) for a bottom-up (non)-deterministic unranked tree automaton, whether there exists an equivalent deterministic online SUTA. The latter is referred to as the simplification problem.

In conclusion, only the determinism notion in online SUTAs is known to be truly top-down deterministic on unranked trees. Determinism in BUTAs, as defined by Brüggemann-Klein et al. [5] as the straightforward extension of the "standard" top-down determinism for ranked trees [7], is a bit different. In spite of the close connection to the well-behaved top-down determinism on ranked trees, minimizing deterministic BUTAs is not completely trivial and the precise complexity is still unknown. From an XML point of view, however, this notion of determinism might be less interesting. It assigns states to nodes, only based on the number of their siblings, which makes them rather poor in expressive power. When one would, for instance, want to allow an automaton to *read the label of a node* before assigning it a state, which seems to be the case in XML schema languages for example, the determinism in online SUTAs would be the obvious candidate.

With respect to future research several natural directions emerge:

1. Top-down determinism and closure properties. As previously mentioned, the lack of closure under union is quite unnatural for an XML schema language. This leads to the following natural questions: (1) What are the possible additions to the deterministic top-down automaton model that closes them under the Boolean operations?; (2) What is the best way to approximate a Boolean combination of deterministic top-down tree automata?; and, (3) What are the properties of the class consisting of the Boolean closure of deterministic top-down tree automata (BC-TA)?

2. Optimization problems. Minimization is of course a very important problem. Can FC-UTAs or BC-TAs be efficiently minimized? Furthermore, what is the complexity of the simplification problem (as defined above) for the various models?

3. In practice not many XML schemas are available and some of those are syntactically incorrect, which leads to the problem of automatically inferring them from a set of XML documents. As the latter reduces to learning in the limit from positive data of deterministic top-down tree automata, it would be interesting to pinpoint classes which can be learned in this manner. Bex et al. addressed the problem of inferring subclasses of DTDs and XSDs [2, 3].

References

[1] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In C. Y. Chan, B. C. Ooi, and A. Zhou, editors, *SIGMOD Conference*, pages 1–12. ACM, 2007.

[2] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise dtds from XML data. In U. Dayal, K.-Y. Whang, D. B. Lomet, G. Alonso, G. M. Lohman, M. L. Kersten, S. K. Cha, and Y.-K. Kim, editors, *VLDB*, pages 115–126. ACM, 2006.

[3] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML schema definitions from XML data. In C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, editors, *VLDB*, pages 998–1009. ACM, 2007.

[4] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible markup language (XML). *World Wide Web Journal*, 2(4):27–66, 1997.

[5] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1, April 3, 2001. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.

[6] J. Clark and M. Murata. Relax NG specification. Technical report, OASIS, 2001. http://relaxng.org/spec-20011203.html.

[7] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2001. http://www.grappa.univ-lille3.fr/tata.

[8] J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In M. Liskiewicz and R. Reischuk, editors, *FCT*, volume 3623 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2005.

[9] F. Gécseg and M. Steinby. Minimal ascending tree automata. *Acta Cybern.*, 4:37–44, 1980.

[10] F. Gécseg and M. Steinby. *Tree automata.* Akadémiai Kiadó (Publishing House of the Hungarian Academy of Sciences), Budapest, 1984.

[11] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.

[12] E. Jurvanen, A. Potthoff, and W. Thomas. Tree languages recognizable by regular frontier check. In *Developments in Language Theory*, pages 3–17, 1993.

[13] W. Martens. *Static Analysis of XML Transformation- and Schema Languages*. PhD thesis, Hasselt University, 2006.

[14] W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML schema. *ACM Trans. Database Syst.*, 31(3):770–813, 2006.

[15] W. Martens and J. Niehren. On the minimization of XML schemas and tree automata for unranked trees. *J. Comput. Syst. Sci.*, 73(4):550–583, 2007.

[16] F. Neven and T. Schwentick. Query automata over finite trees. *Theor. Comput. Sci.*, 275(1-2):633–674, 2002.

[17] M. Nivat and A. Podelski. Minimal ascending and descending tree automata. *SIAM J. Comput.*, 26(1):39–58, 1997.

[18] H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, 19(3):424–437, 1990.

[19] J. Shallit. Numeration systems, linear recurrences, and regular sets (extended abstract). In W. Kuich, editor, *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 1992.

[20] C. Sperberg-McQueen and H. Thompson. XML Schema. Technical report, World Wide Web Consortium, 2007. http://www.w3.org/XML/Schema.

[21] J. W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *J. Comput. Syst. Sci.*, 1(4):317–322, 1967.

[22] W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25(3):360–376, 1982.

[23] J. Virágh. Deterministic ascending tree automata I. *Acta Cybern.*, 5:33–42, 1981.

Expressive power of monadic logics on words, trees, pictures, and graphs[*]

Oliver Matz[1]
Nicole Schweikardt[2]

[1] Institut für Informatik
Christian-Albrechts-Universität zu Kiel
Christian-Albrechts-Platz 4
24118 Kiel, Germany
matz@ti.informatik.uni-kiel.de

[2] Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
schweika@informatik.hu-berlin.de

Abstract

We give a survey of the expressive power of various monadic logics on specific classes of finite labeled graphs, including words, trees, and pictures. Among the logics we consider, there are monadic second-order logic and its existential fragment, the modal mu-calculus, and monadic least fixed-point logic. We focus on nesting-depth and quantifier alternation as a complexity measure of these logics.

1 Introduction

There is a close relationship between (generalized) automata theory and the expressive power of certain monadic logics. Already in 1960, Büchi and Elgot proved that a word-language is recognizable by a finite automaton if, and only if, it can be characterized by a monadic second-order formula. Since then, various analogous results, e.g., for labeled trees rather than words, and also for more general classes of labeled graphs, have been obtained. Alluding to the notion of "descriptive complexity theory", in his survey article [39] for the *Handbook of Formal Languages*, Wolfgang Thomas called the branch of research that investigates the relationship between generalized finite automata and monadic logics a "descriptive theory of recognizability".

[*] Both authors wish Wolfgang Thomas all the best for this jubilee. Matz would like to express his gratitude for Wolfgang Thomas' careful supervision during the preparation of Matz' Ph.D. thesis. Furthermore, we should like to thank the anonymous referee for the detailed remarks.

The present paper's aim is to give a survey of the expressive power of various monadic logics (including monadic second-order logic and its existential fragment, the modal mu-calculus, and monadic least fixed-point logic), on specific classes of finite labeled graphs. In particular, we give details on the following topics:

It is known that on finite words and labeled trees, all the above mentioned monadic logics have the same expressive power and can characterize exactly the languages that are recognizable by a suitable notion of finite automata. Moreover, already one single existential set quantifier suffices to obtain the expressive power of existential monadic second-order logic on words, trees, and pictures (i.e., two-dimensional words or, equivalently, labeled grid-graphs). This goes back to a paper by Wolfgang Thomas [38], in which he showed that a single existential set quantifier suffices for words. From the proof, one can also infer an elegant proof which shows that finite automata can be simulated by monadic least fixed-point logic. Wolfgang Thomas' Ph.D. students Potthoff [32] and Matz [25] obtained according results for trees and pictures, respectively. On the other hand, when going slightly beyond the class of pictures, it is known from work by Otto [31] that within existential monadic second-order logic, *more* set quantifiers lead to strictly more expressive power.

While on words and labeled trees, existential monadic second-order logic has the same expressive power as full monadic second-order logic, the situation is different for the class of pictures. From work by Giammarresi, Restivo, Seibert, and Thomas [17] it is known that existential monadic second-order logic can define exactly the *recognizable* picture languages, which are characterized by a suitably adapted automaton model, the *tiling-systems*. But full monadic second-order logic on pictures has considerably more expressive power and, in fact, precisely corresponds to the linear time hierarchy (i.e., the linear time analogue of Stockmeyer's polynomial time hierarchy). Similarly, building on results by Schweikardt [36], one obtains that the picture languages definable in monadic least fixed point logic can encode at least all problems that belong to Grandjean's deterministic linear time complexity class DLIN [18]. Furthermore, unless P = NP, the expressiveness of monadic least fixed point logic on pictures is strictly weaker than that of monadic second-order logic.

Also some aspects concerning the fine structure of monadic second-order logic over pictures and graphs are understood quite well by now: Matz, Schweikardt, and Thomas [28, 35, 27] showed that the monadic second-order quantifier alternation hierarchy is strict, i.e., that formulas in prenex normal form having a prefix of $k{+}1$ alternations of set quantifiers can describe strictly more picture languages (or, in general, graph properties) than formulas with only k quantifier alternations. Note, however, that this result does not have implications concerning the strictness of the linear time hierarchy (or the polynomial time hierarchy) as the levels of the monadic second-order quantifier alternation hierarchy do not correspond to the levels of the linear time hierarchy.

When considering the modal mu-calculus instead of monadic second-order logic on finite labeled graphs, an according hierarchy based on the alternation of least and greatest fixed point operators was proved independently by Bradfield [4] and Lenzi [22], see also Arnold [2] for an elegant proof. (The hierarchies proved in [4, 22, 2] are about general structures that are not necessarily finite; via the mu-calculus' *finite model property* (cf., e.g., [3]), however, they can be directly transferred to the class of finite labeled graphs.) Up to date, it still is an open question whether an analogous hierarchy can be proved for monadic least fixed point logic.

The rest of this paper is structured as follows: In Section 2 we fix the necessary notation concerning the logics and the structures that are considered in this paper. Section 3 concentrates on the relations between (finite-state) recognizability of word languages, tree languages, and picture languages and their definability in various monadic logics. In Section 4, we go beyond recognizability and study nesting-depth and quantifier alternation as a complexity measure of logics.

2 Logics and structures considered in this paper

This section fixes some basic notations and conventions used throughout the remainder of the paper.

2.1 Structures

All structures considered in this paper are finite and can be viewed as particular kinds of labeled graphs. Namely, we consider labeled trees, words, and pictures (i.e., two-dimensional words).

Let us fix a finite alphabet Σ, whose elements serve as letters at positions in a word or a picture or as labels for nodes in a graph or a tree. For this exposition it is convenient (and no essential loss of generality) to assume that Σ is of the form $\{0,1\}^t$ for some $t \geqslant 0$ (for $t = 0$, the alphabet Σ is a singleton).

A *word (over* Σ*)* is a finite sequence of elements in Σ. A *word language* is a set of words. In order to use logic formulas to define word languages we consider the signature $\{\mathrm{Succ}, B_1, .., B_t\}$, where Succ is a binary relation symbol and $B_1, .., B_t$ are unary relation symbols. We identify a word $w = w_1 \cdots w_n$ over Σ with the structure of signature $\{\mathrm{Succ}, B_1, .., B_t\}$ whose universe is the set $[n] := \{1, .., n\}$ of positions in the word, and where Succ is interpreted by the natural successor relation on $[n]$ and, for every $i \in \{1, .., t\}$, the relation symbol B_i is interpreted by the set of all positions at which the word carries a letter $(\sigma_1, .., \sigma_t) \in \Sigma = \{0,1\}^t$ with $\sigma_i = 1$.

Pictures are two-dimensional analogues of words, i.e., a *picture (over Σ)* is a two-dimensional (rectangular) array over Σ. A *picture language* is a set of pictures. Like for words, it is straightforward to associate, with every picture, a model over a specific signature, this time with two binary relations Succ_h and Succ_v for the horizontal and the vertical successor relation, respectively.

For convenience, all *trees* considered in this paper will be ordered and binary, i.e., every node is either a leaf or has two children. Each node of a *labeled tree* (over Σ) is labeled by an element in Σ. A *tree language* is a set of labeled trees. Similarly as words and pictures, also trees can be identified in a straightforward way by structures over the signature $\{\mathrm{Succ}_1, \mathrm{Succ}_2, B_1, .., B_t\}$, where the binary relations Succ_1 and Succ_2 are used for the edges from a node to its first child and to its second child, respectively.

2.2 Logics

We assume that the reader is familiar with first-order logic (FO), monadic second-order logic (MSO), least fixed-point logic (LFP), and the modal mu-calculus. We write MLFP for *monadic* least fixed-point logic, i.e., the fragment of LFP where only *monadic* second-order variables are allowed. It is straightforward to see that monadic least fixed-points can be defined in MSO, and thus the expressive power of MLFP lies between the expressive power of FO and the expressive power of MSO. Some focus of the present paper will also be on *existential monadic second-order logic* (EMSO), which consists of all MSO-formulas of the form

$$\exists X_1 \cdots \exists X_\ell \ \varphi,$$

where φ is first-order, $\ell \geqslant 0$, and $X_1, .., X_\ell$ are set variables (i.e., monadic second-order variables). Further, we shall write 1-EMSO for the fragment of EMSO where just a single set variable is available.

If φ is a sentence (over a suitable signature and a certain logic), the (word, picture, or tree) language *defined* by φ is the set of all words (or pictures or trees) whose associated word (or picture or tree) models make φ true.

3 Monadic logics and recognizability

This section concentrates on the relations between recognizability of word languages, tree languages, and picture languages and their definability in various monadic logics. Here, "recognizability" refers to non-deterministic finite automata or suitable adaptations thereof.

We shall first quickly review the well-known results on words and trees which, basically, state that all the monadic logics mentioned in Section 2 have the same expressive power, namely of defining exactly the *regular* word languages and tree languages.

Afterwards, we shall move over to the case of pictures, where things turn out to be much more subtle, since the various monadic logics differ with respect to their power of defining picture languages.

3.1 Monadic logics and recognizability of words and trees

The class of regular (or, recognizable) word languages plays a central role in the theory of formal languages. One reason for this is the large variety of its conceptually different characterizations, for example by means of monoids, grammars, automata, closure properties, and logics. Concerning the subject of this paper, let us focus on the following two: non-deterministic finite automata (NFA) and monadic second-order logic.

Theorem 3.1 (Büchi-Elgot, [6, 12])**.** A word language is regular if, and only if, it can be defined by an MSO-sentence.

Since we shall come back to this later (in the context of pictures instead of words), let us briefly point out the essential steps in the well-known proof of the above theorem.

Proof (sketch). One direction is simple to prove: Given a non-deterministic finite automaton $\mathfrak{A}$, we have to construct a monadic second-order sentence that asserts for a given word (model) that there exists an accepting run. The existence of such a run can be expressed by a formula of the form

$$\exists X_1 \cdots \exists X_\ell \; \varphi(X_1, .., X_\ell),$$

where an assignment to the set variables encodes an assignment of $\mathfrak{A}$'s states to positions in the word, and φ asserts that for any two consecutive positions, this assignment is compatible with the automaton's transition relation, the initial state and the final states. We observe that the resulting formula is in the existential fragment EMSO of monadic second-order logic.

The other direction is more intricate. Typically, it is done as follows: Given an MSO-sentence φ, we may pass to a similar sentence φ' in prenex normal form, where all first-order quantifiers are eliminated and special, new predicates *singleton*(X) are used instead, which assert for a set X that it has just one element. An NFA can be constructed by induction on the construction of such formulas. In this induction, one exploits that the class of regular word languages is closed under union, complementation, and projection, to handle disjunction, negation, and existential MSO quantification, respectively. Q.E.D.

The above proof, in particular, leads to:

Corollary 3.2. Over the class of words, every MSO-sentence is equivalent to an EMSO-sentence.

Even more, it is known that already a single existentially quantified set variable suffices:

Theorem 3.3 (Thomas, [38]). Over the class of words, every MSO-sentence is equivalent to a 1-EMSO-sentence.

Proof (sketch). The proof relies on the following simple and elegant idea: Given a deterministic finite automaton $\mathfrak{A}$ with r states and, w.l.o.g., state space $\{1, \ldots, r\}$, each state i can be represented by the bit-string 01^i0^{r-i} of length $r' := r + 1$. If w is an input word, we can subdivide w into sub-words such that each of these sub-words has length r', except for the last one, whose length is between r' and $2r' - 1$. Each of these sub-words can be decorated by the bit-string that represents $\mathfrak{A}$'s state when entering the first position of the sub-word. Such bit-strings, in turn, can of course be represented by an assignment to a single set variable, e.g., by assuming that the set consists of exactly those positions where the bit-string carries the letter 1.

Now, it is easy to construct a 1-EMSO-sentence of the form $\exists X\, \varphi(X)$, where φ is first-order and expresses that the bit-string represented by X encodes the list of states assumed by $\mathfrak{A}$ at the beginnings of the sub-words. For constructing φ, note that (1) each sub-word has constant length $< 2r'$, (2) the leftmost positions of the sub-words can be identified from the fact that they do not belong to X but their successors do, and (3) the steps that $\mathfrak{A}$ performs while reading the sub-word can be simulated by a first-order formula. This way, φ can check that the list of states represented by X is consistent with $\mathfrak{A}$'s transition relation and represents an accepting run of $\mathfrak{A}$. Q.E.D.

A closer look at this proof sketch shows that a similar set X can also be defined as a monadic least fixed-point of a suitable first-order formula: This time, sub-words of length $r' := 1 + 2r$ are considered, and each state $i \in \{1, .., r\}$ is represented by the bit-string $10^{i-1}10^{2r-i}$. Note that r' is chosen in such a way that the distance between two consecutive positions carrying the letter 1 tells us, which of the two positions marks the beginning of a sub-block and which of the two positions marks a state of the automaton. Using this, one obtains that every regular word language can be described by an MLFP-sentence which uses just a single monadic least fixed point operator (see Potthoff [32] for details). In a similar way, one can also prove that the modal mu-calculus can describe exactly the regular word languages. In summary, we thus have the following situation:

Theorem 3.4. On the class of words, MSO, EMSO, 1-EMSO, MLFP, and the modal mu-calculus have the same expressive power and can describe exactly the regular word languages.

The same result holds true for the class of labeled trees (cf. [37, 10, 32, 20]).

If we leave the classes of words and labeled trees and pass over to pictures, this is not the case any more. We shall give details on this in the next subsection.

3.2 EMSO-definability and recognizability of pictures

In [16], Giammarresi and Restivo suggested a natural adaptation of NFA to picture languages: the so-called *tiling-systems*.

Definition 3.5. A tiling-system is a quadruple $(\Sigma, \Gamma, \Delta, \pi)$, where Σ and Γ are finite alphabets, $\pi : \Gamma \to \Sigma$ is an *alphabet projection*, and Δ is a set of $2{\times}2$-pictures over alphabet $\Gamma \cup \{\#\}$, where $\#$ is a fresh boundary symbol. The mapping π is lifted to pictures in the obvious way.

A picture p over Σ is *accepted* by such a tiling-system iff there is a picture r over Γ such that $\pi(r) = p$ and Δ contains all $2{\times}2$-sub-blocks of the picture that results by surrounding r with the boundary symbol $\#$. The picture language *recognized* by some tiling-system T is the set of pictures accepted by T.

Example 3.6. Consider the tiling-system $T = (\{a\}, \{0,1\}, \Delta, \pi)$, where $\pi(0) = \pi(1) = a$, and where Δ is the set of 2×2-subblocks of

$$\begin{array}{cccccccccc}
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & \# \\
\# & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \# \\
\# & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \#
\end{array}$$

Then T recognizes the set of all pictures p over $\{a\}$ for which there exists $m \geqslant 1$ such that p has size $m \times 2^m$. Intuitively, T establishes a mechanism of binary counting the columns.

More examples of recognizable picture languages can be found in Giammarresi and Restivo's article in the present book.

Unlike the regular word languages, which are pretty simple to understand, the recognizable picture languages can be very complex, both from an intuitive and from a computational point of view. For example, in [33, 34], Reinhard has found examples of picture languages whose proofs of recognizability are very difficult and which disproved previous conjectures by Matz, e.g. [24]. Still, examples near the borderline between recognizable and non-recognizable picture languages are subject of current research, see [7].

It is known that the class of recognizable picture languages is closed under union, intersection, row- and column-concatenation, and row- and column-Kleene-star [16], but we have:

Theorem 3.7 (Giammarresi-Restivo-Seibert-Thomas [17])**.** If the alphabet has at least two symbols, the class of recognizable picture languages is not closed under complement.

A witness for the above theorem is given by:

Example 3.8 (Giammarresi-Restivo-Seibert-Thomas, [17])**.** Let L be the set of pictures over $\{0, 1\}$ that result from the concatenation of two identical pictures of quadratic shape. Then L is not recognizable, but its complement is.

The statement of Theorem 3.7 is true also for singleton alphabets, see Theorem 4.12 below.

MSO logic, of course, is closed under negation, so Theorem 3.7 immediately implies that the statement of Theorem 3.1 is not true when replacing the terms "word language" and "regular" with "picture language" and "recognizable". However, it is known that the existential fragment of monadic second-order logic, EMSO, has exactly the right power for expressing recognizable picture languages:

Theorem 3.9 (Giammarresi-Restivo-Seibert-Thomas, [17])**.** A picture language is recognizable if, and only if, it can be defined by an EMSO-sentence.

The "easy" direction in the proof is to show that recognizability by a tiling-system can be described by an EMSO-formula. This case can be handled in a similar way as in the proof of Theorem 3.1.

The other direction, however, cannot be handled in a similar way as in that proof because the initial replacement of first-order quantifiers by set quantifiers would force us to deal with the negation during the induction, but the class of recognizable picture languages is not closed under complement. Thus, one essential step in the proof given in [17] is a specific treatment of the first-order quantifiers with Ehrenfeucht-Fraïssé games. This yields, as a side-product, also the characterization of the *first-order* definable picture languages as the *locally threshold testable* ones, analogously to the one-dimensional case.

The characterization of the EMSO-definable picture languages given in Theorem 3.9 opened the door to several combinatorial arguments that allow to show that certain picture languages are not EMSO-definable, see for example [15, 24]. This was the basis for the original proof of the strictness of the monadic second-order quantifier alternation hierarchy [28], see Section 4.3 below.

Similarly as for words it is known that for defining recognizable picture languages, already a single existentially quantified set variable suffices:

Theorem 3.10 (Matz, [25])**.** Over the class of pictures, every EMSO-sentence is equivalent to a 1-EMSO-sentence.

The proof is by an adaptation of the proof of Theorem 3.3 to the two-dimensional case: A tiling-system plays the role of the finite automaton $\mathfrak{A}$, with the minor technical inconvenience that tiling-systems are inherently non-deterministic. However, the determinism of the automaton $\mathfrak{A}$ in the proof of Theorem 3.3 is not essential.

Let us mention that by results of Otto [31] it is known that when going slightly beyond the class of pictures, EMSO does not collapse to 1-EMSO but, quite to the contrary, there is a strict hierarchy within EMSO with respect to the number of existentially quantified set variables. To precisely state Otto's result, let us write k-EMSO for the fragment of EMSO where k set variables are available. Instead of pictures, Otto considers particular structures over a signature which consists of two binary relation symbols R and C. These *Otto-grids* are structures whose universe forms a rectangular array and where R and C are interpreted by the relations stating that two vertices belong to the same row, respectively, the same column of the array.

Theorem 3.11 (Otto, [31])**.** For every $k \geqslant 0$ there is a $(k{+}1)$-EMSO-sentence that is not equivalent (over the class of Otto-grids) to any k-EMSO-sentence.

The proof is by showing that the set L_k of all Otto-grids with the property that the number of columns is $\leqslant 2^{(k+1)\cdot\text{number of rows}}$ is definable in $(k{+}1)$-EMSO but not in k-EMSO (for the latter, an Ehrenfeucht-Fraïssé game argument is used).

To close the subsection on recognizable picture languages, let us have a quick look at the *computational complexity* of standard decision problems concerning recognizable picture languages.

Proposition 3.12 (Giammarresi-Restivo, [16])**.** The *emptiness* problem for tiling-systems is undecidable.

Proof (sketch). We sketch a reduction of the emptiness problem for Turing machines. Let $\mathfrak{A}$ be a Turing machine. It is straightforward to encode a configuration of $\mathfrak{A}$ by a finite word over a fixed alphabet Σ. Each step in a computation of $\mathfrak{A}$ corresponds to a local modification of that code. Every finite—and hence every accepting—run R of $\mathfrak{A}$ can be encoded by a picture p over Σ, where p contains, in each row i, the code of the i-th configuration of R (possibly padded with blank symbols).

Now it is easy to effectively construct a tiling-system T that accepts all pictures that encode an accepting run. Then the language recognized by T is non-empty iff $\mathfrak{A}$ has an accepting run. Q.E.D.

Furthermore, the *membership* problem for tiling-systems is NP-complete:

Proposition 3.13 (Schweikardt, [35]). (a) The following problem belongs to NP: Given a tiling-system T and a picture p, does T accept p?

(b) There exists a tiling-system T such that the following problem is NP-complete: Given a picture p, does T accept p?

Proof (sketch). The proof of (a) is straightforward. (b) is obtained by coding the (NP-complete) problem of satisfiability of propositional formulas in conjunctive normal form into an EMSO-definable picture language. To this end, each propositional formula α is represented by a picture which has a row for each variable and a column for each clause of α, such that the entry in row i and column j of the picture is labeled by the letter P (resp. N, resp. $\ominus$) if the i-th propositional variable occurs unnegated (resp. negated, resp. not at all) in the j-th clause of α. A *truth assignment* to the variables of α is represented by a set X of positions in the picture which, for each row, contains either none or all positions of that row. I.e., if the i-th propositional variable is assigned the value *true* (resp., *false*), then X contains *all* (resp. *none*) of the positions in the i-th row. It is not difficult to find an EMSO-formula ψ which expresses that there exists such a set X which encodes a satisfying assignment for α. Altogether, this gives us a reduction from the NP-complete satisfiability problem to the problem of deciding whether an input picture belongs to the picture language defined by ψ. Q.E.D.

It is current research interest to determine computationally feasible subclasses of recognizable picture languages, see e.g. the article of Giammarresi and Restivo in the present book.

3.3 Picture languages definable in MSO and MLFP

From the previous subsection we know that the EMSO-definable picture languages coincide with the 1-EMSO-definable and the recognizable picture languages. Furthermore, recall that Example 3.8 exposes a picture language that is not definable in EMSO. It is not difficult to see that this language is definable in MSO as well as in MLFP. The present subsection aims at a deeper understanding of the MSO-definable and the MLFP-definable picture languages.

Let us first concentrate on the MSO-definable picture languages. It is easy to see that the *membership* problem for each MSO-definable picture language belongs to LINH, i.e., the *linear time hierarchy* (cf., e.g. [11]), which is the linear time analogue to Stockmeyer's polynomial time hierarchy. On the other hand, it is not difficult to see that, in fact, the MSO-definable picture languages precisely correspond to the linear time hierarchy, since every decision problem that belongs to LINH can be encoded by an MSO-definable picture language. This can be obtained as follows: From [29] we know that LINH is the class of all word languages that can be defined in MSO(Bit), i.e., in monadic second-order logic on words where in addition to the successor relation, also the *Bit predicate* on the set of positions in the word is available (the Bit predicate is the set of all tuples (i, j) such that the i-th bit in the binary representation of the natural number j is 1). The basic idea now is to represent a word of length n by a picture as follows: Let ℓ be the largest integer such that $n \geqslant \ell \cdot 2^{\ell}$, cut the word into sub-words of length 2^{ℓ}, and arrange the consecutive sub-words into consecutive rows of the resulting picture (if necessary, pad the last row with dummy entries to obtain a rectangular picture). Of course, the successor relation Succ of the original word can easily be simulated by an MSO-formula over the corresponding picture. Furthermore, it is a not too difficult exercise to also construct an MSO-formula over the picture which simulates the Bit predicate of the original word (*hint:* use an existentially quantified unary relation to encode a "column-numbering" which writes the binary representations of the numbers $0, 1, 2, \ldots 2^{\ell}-1$ into the consecutive columns of the picture). It then is not difficult to see that every MSO(Bit)-definable set of strings is represented by an MSO-definable set of pictures. In this sense, the MSO-definable picture languages can encode all problems that belong to the linear time hierarchy.

Let us now concentrate on the MLFP-definable picture languages. Of course, for each picture language defined by a fixed MLFP-sentence, the membership problem belongs to P. Together with Proposition 3.13 and the fact that the expressive power of MLFP lies between FO and MSO, this implies the following:

Fact 3.14. Unless P = NP, MLFP is strictly less expressive on the class of pictures than MSO.

On the other hand, MLFP is still quite expressive as it can define picture languages corresponding to every problem in the deterministic linear time complexity class DLIN introduced by Grandjean in [18]. The class DLIN is based on linear time random access machines. In a series of papers, Grandjean made a convincing point that DLIN might be viewed as "the" adequate mathematical formalization of linear time complexity. For example, DLIN contains all problems in DTIME(n), i.e., all problems solvable by deterministic linear time multi-tape Turing machines; but DLIN also contains problems such as the *sorting* problem, which are conjectured not to belong to DTIME(n).

In a similar way as described above for MSO and LINH, one obtains that every problem in DLIN can be encoded by an MLFP-definable picture language—instead of using the characterization of LINH as the MSO(Bit)-definable word languages, one now just has to use a result from [36] stating that every word language which belongs to DLIN can be defined by an MLFP(Bit)-sentence.

4 Alternation hierarchies

In descriptive complexity theory it is a general task to classify properties by the complexity a formula must have to describe this property. But what is the suitable measure for the complexity of a formula? A typical approach is to measure the complexity by the nesting depth of the "most powerful ingredient" of the logic under consideration.

For example, a measurement for the complexity of a first-order formula is the nesting depth of first-order quantifiers, neglecting the complexity introduced by boolean combinations. Another example is the modal mu-calculus, where it is the nesting depth of fixpoint iterations that is the natural means to measure the complexity of a formula. MSO is a third example, where the nesting depth of the most powerful quantifications (in this case, the monadic ones) establishes a measure of formula complexity.

In Section 3 we have already considered the nesting depth of set quantifiers as a complexity measure of MSO-formulas and have seen (Theorem 3.3) that the corresponding hierarchy collapses for the classes of words and of trees whereas it is infinite for Otto-grids (Theorem 3.11).

However, for many logics and classes of structures, the complexity measurement obtained by simply counting syntactic nesting of single quantifiers is (1) not sufficiently robust, and (2) does not result in the natural parameters for the computational complexity, e.g. of the model checking or the satisfiability problem of formulas.

To illustrate the first reason, consider two 1-EMSO-sentences on the class of finite structures. Their conjunction is in 2-EMSO but, unlike their disjunction, in general not in 1-EMSO, so that the class of 1-EMSO-definable properties is not necessarily closed under intersection.

To illustrate the second reason, let us consider MSO over words. A good approach for solving the model checking problem relies on the well-known construction of an NFA for a given MSO-formula (see Theorem 3.1 and its proof sketch). The constructions for conjunction, disjunction, and existential quantification can be done directly on NFA and result in no essential increase of the number of states. However, the construction for the negation of a formula requires a deterministic automaton and therefore the famous powerset construction, which results in an exponential state blow-up. Thus it is the *alternation* of existential quantifications and negation (or, equivalently: the alternation of existential and universal quantifications) that is significant for the increase of the state set size and therefore for the computational complexity of the model checking problem.

4.1 First-order alternation

As motivated above, one passes to a coarser view of "nesting" by considering a block of only existential (or only universal) quantifiers as one single, "vectorial" quantifier. This vectorial approach is the basis for the *first-order quantifier alternation hierarchy.* For example, a property of finite labeled graphs is in the third level of that hierarchy iff it can be defined by a first-order formula that has a prenex normal form with a quantifier prefix of type

$$\exists^* \forall^* \exists^*,$$

i.e., a quantifier prefix with three blocks of first-order quantifications, starting with an existential one, and the following kernel formula is quantifier-free. Level k of the first-order quantifier alternation hierarchy is usually denoted Σ^0_k, its "complement" Π^0_k (i.e., Π^0_k is the set of all graph properties that can be defined by a first-order formula in prenex normal form that has a quantifier prefix with k blocks of first-order quantifications, starting with a universal one).

Theorem 4.1 (Chandra-Harel-Thomas, [8, 38])**.** The first-order quantifier alternation hierarchy is strict over the class of finite labeled graphs, i.e., for every $k \geqslant 0$, $\Sigma^0_k \subsetneq \Sigma^0_{k+1}$. Furthermore, for every $k \geqslant 1$, $\Sigma^0_k \neq \Pi^0_k$.

Chandra and Harel's proof in [8] explicitly provides, for each $k \geqslant 0$, a property of finite labeled directed graphs that belongs to Σ^0_{k+1} but not to Σ^0_k. They consider graphs that are equipped with a distinguished "start node" and a subset of nodes called "winning positions". With each such graph, they associate a 2-player game in which a token is moved along the edges of the graph. At the beginning, the token is placed on the "start node". The players take turns, starting with player 1, and in each move one of the players moves the token along an edge of the graph. After $k{+}1$ such moves, player 1 has *won* the game, if the token lies on a "winning position". It is now easy to find a Σ^0_{k+1}-sentence which expresses that player 1 has a winning strategy for $k{+}1$ moves; and by an Ehrenfeucht-Fraïssé game argument it can be shown that this cannot be expressed by any Σ^0_k-sentence.

A different proof of the strictness of the first-order quantifier alternation hierarchy is given in [38], where Wolfgang Thomas considers first-order formulas over word models with a different signature than in the present paper, namely with the ordering $<$ instead of the successor relation on the word positions. He shows that the first-order quantifier alternation hierarchy over that signature corresponds to the *dot-depth alternation hierarchy*, which is shown to be strict in [9].

However, for words, trees, and pictures (over the signatures introduced in Section 2.1, i.e., without ordering but with successor relation(s)), the first-order quantifier alternation hierarchy collapses to boolean combinations of its first level. This is a consequence of the characterization of first-order definable properties of words, trees, and pictures by *local threshold testability*, cf., e.g., the survey [39] and the article [17].

4.2 Fixpoint alternation in the mu-calculus

Niwiński [30] introduced vectorial fixpoints to result in a sufficiently coarse and robust definition for the modal mu-calculus fixpoint alternation hierarchy which relies on the number of alternations of least and greatest fixed point quantifiers—see [4] for a detailed discussion of that subject.

Theorem 4.2 (Bradfield, [4]). The modal mu-calculus alternation hierarchy is strict over the class of finite labeled graphs, i.e., for every $k \geqslant 0$, there is a property of finite labeled graphs that is definable in level $k{+}1$ of the Niwiński alternation hierarchy of the modal mu-calculus, but not in level k.

In [22], Lenzi proved a corresponding but slightly weaker result referring to a different variant of fixpoint alternation, the Emerson-Lei hierarchy. An elegant proof of Bradfield's and Lenzi's hierarchy was given by Arnold in [2]. Let us mention that the hierarchies proved in [4, 22, 2] are about general structures that are not necessarily finite; via the mu-calculus' *finite model property* (cf., e.g., [3]), however, they can be directly transferred to the class of finite labeled graphs.

On the other hand, when considering the class of finite words (instead of the class of finite labeled graphs), the modal mu-calculus alternation hierarchy is known to collapse (this can be proved in a similar way as discussed in the paragraph before Theorem 3.4). More details on the collapse of the modal mu-calculus hierarchy on particular classes of structures can be found in [23, 40, 21].

It is a challenging future task to settle the following question:

Question 4.3. Does a similar result as Theorem 4.2 hold for monadic least fixed point logic MLFP instead of the modal mu-calculus? I.e., is there a strict hierarchy within MLFP that is based on the number of alternations of least and greatest fixed point quantifiers?

4.3 Monadic second-order logic

Let us now consider monadic second-order logic MSO. In that logic, the most powerful ingredient is the set quantification. The quantifier structure of an MSO-formula in prenex normal form can be represented by a word over the four-element alphabet $\{\exists, \forall, {\scriptstyle\exists}, {\scriptstyle\forall}\}$, where $\exists, \forall$ represent set quantifiers, and ${\scriptstyle\exists}, {\scriptstyle\forall}$ represent first-order quantifiers. In the following, we use regular expressions over that alphabet to describe quantifier prefixes of formulas in prenex normal form.

Every MSO-formula is equivalent (over the class of all structures) to an MSO-formula whose quantifier prefix is of type

$$\{\exists, \forall\}^* \{{\scriptstyle\exists}, {\scriptstyle\forall}\}^*.$$

A transformation of a given MSO-formula ψ into the above form can be done in three steps: Firstly, replace every sub-formula of the form $\exists x\, \varphi(x)$ with an equivalent formula of the form $\exists X\ (\mathit{singleton}(X) \wedge \varphi'(X))$, where $\mathit{singleton}(X)$ is an auxiliary first-order formula asserting that X is a singleton, and where φ' results from φ by replacing every atomic formula $\alpha(x_1, .., x_n)$ with a suitable auxiliary first-order formula $\alpha'(X_1, .., X_n)$. Note that the resulting formula ψ' contains first-order quantifiers only within the new auxiliary formulas $\mathit{singleton}(X)$ and $\alpha'(X_1, .., X_n)$. Secondly, transform ψ' into prenex normal form, treating the auxiliary formulas like atoms. Now, viewing the auxiliary formulas again as first-order formulas, the resulting MSO-formula ψ'' obviously consists of a quantifier prefix of set quantifiers that is followed by a first-order formula. By transforming the first-order part of this formula into prenex normal form, one then obtains an MSO-formula in prenex normal form whose quantifier prefix is of type $\{\exists, \forall\}^* \{{\scriptstyle\exists}, {\scriptstyle\forall}\}^*$.

4.3.1 The MSO Quantifier Alternation Hierarchy

The definition of the *monadic second-order quantifier alternation hierarchy* (or "MSO alternation hierarchy" for short) is based on the above representation. For each $k \geqslant 0$, level k of this hierarchy consists of those properties (of, say, finite labeled graphs) that can be defined by an MSO-formula in prenex normal form where the set quantifiers are grouped into k blocks, existential and universal in alternation, starting with an existential one. While most parts of Section 3 are devoted to EMSO, the first level of this hierarchy, we consider the higher levels now. For example, a property is in level three of that hierarchy iff it can be defined by a formula in prenex normal form of type

$$\exists^* \forall^* \exists^* \{{\scriptstyle\exists}, {\scriptstyle\forall}\}^*.$$

i.e., one that starts with three blocks of set quantifiers, the first one being existential, and continues with a first-order kernel formula.

Let us denote level k of the MSO quantifier alternation hierarchy by mon-Σ^1_k, its "complement" by mon-Π^1_k (i.e., mon-Π^1_k consists of all graph properties whose complement belongs to mon-Σ^1_k), and their intersection by mon-Δ^1_k. Furthermore, we write BC(mon-Σ^1_k) to denote the class of all properties that can be defined by a boolean combination of sentences suitable for mon-Σ^1_k. (Thus BC(mon-Σ^1_k) is the smallest superclass of mon-Σ^1_k that is closed under union and complement.)

By slightly abusing notation, we shall sometimes also speak of mon-Σ^1_k *formulas* to address the particular kind of formulas suitable for defining properties that belong to mon-Σ^1_k.

Fagin has shown that *connectivity* of finite graphs is (analogously to Example 3.8) definable by a sentence in prenex normal form of type $\forall^*\{{\scriptstyle\exists},{\scriptstyle\forall}\}^*$, but not by one of type $\exists^*\{{\scriptstyle\exists},{\scriptstyle\forall}\}^*$. This leads to the following result:

Theorem 4.4 (Fagin, [13])**.** mon-$\Sigma^1_1 \neq$ mon-Π^1_1 and thus, in particular, mon-$\Sigma^1_1 \subsetneq$ mon-Σ^1_2.

Fagin raised the question whether the MSO quantifier alternation hierarchy collapses on some higher level. The question has been answered negatively in [28]. Refining that proof, [35, 27] shows that a witness for the separation of level $k{+}1$ from level k is the set of all pictures of size $m \times f(m)$ for a specific ($k{+}1$)-fold exponential function: this picture language is definable by a sentence with $k{+}1$ alternations of set quantifiers, but not by one with just k alternations of set quantifiers. The same witness even separates mon-Δ^1_{k+1} from BC(mon-Σ^1_k). Using standard techniques, the results can be transported to the class of graphs. We thus obtain

Theorem 4.5 (Matz-Schweikardt-Thomas, [27])**.** For every $k \geqslant 0$, mon-$\Sigma^1_k \subsetneq$ mon-Σ^1_{k+1}. Moreover, there even exists a picture language over a singleton alphabet that belongs to mon-Δ^1_{k+1} but not to BC(mon-Σ^1_k).

However, the proof of this theorem has also exhibited the following: it is *not* the alternation of set quantifiers that gives the expressive power needed to leave a fixed level of that hierarchy—it is the nesting of *first-order* quantifiers, followed by one single block of set quantifiers. For example, there is an MSO-sentence with quantifier prefix of type

$${\scriptstyle\forall}^*{\scriptstyle\exists}^*\forall^*\{{\scriptstyle\exists},{\scriptstyle\forall}\}^*,$$

that is not equivalent to any sentence with quantifier prefix of type

$$\exists^*\forall^*\exists^*\{{\scriptstyle\exists},{\scriptstyle\forall}\}^*$$

(and likewise for values larger than three).

How is this possible? The definition of the MSO quantifier alternation hierarchy allows to neglect first-order quantifications inside the kernel formula, but it does not allow to neglect first-order quantifications completely. This is so because first-order quantifications do not factor through *monadic* second-order quantifications, unlike for the full second-order logic, in which quantification is available over relations of arbitrary arity. We shall take a closer look at this phenomenon in the following paragraph.

4.3.2 The Closed MSO Hierarchy

As motivated above, the value of the strictness of the MSO quantifier alternation hierarchy would be much higher if first-order quantification was, by definition, neglectable. This point was made by Ajtai, Fagin, and Stockmeyer in [1]. In that paper, the authors suggest the *closed MSO alternation hierarchy*, which is coarser and more robust than the ordinary MSO alternation hierarchy because it allows to intersperse first-order quantifiers "for free" between set quantifiers. For example, a property is in level three of that hierarchy iff can be defined by an MSO-formula which has a prenex normal form of type

$$\{\exists, {\scriptstyle\exists}, {\scriptstyle\forall}\}^* \, \{\forall, {\scriptstyle\exists}, {\scriptstyle\forall}\}^* \, \{\exists, {\scriptstyle\exists}, {\scriptstyle\forall}\}^* \, \{{\scriptstyle\exists}, {\scriptstyle\forall}\}^* \, .$$

As noted in [1], the strictness of the *closed MSO alternation hierarchy* would be implied by the conjectured strictness of the polynomial time hierarchy, because each level of the latter is closed under first-order quantification and each level of the MSO alternation hierarchy contains a complete problem for the polynomial time hierarchy. The following is a challenging future task:

Task 4.6. Show, without relying on complexity theoretic assumptions, that the closed MSO alternation hierarchy is strict.

4.3.3 The First-Order Closure

As pointed out above, it is desirable to understand more about the role of first-order quantifications in the context of monadic second-order quantifier alternation. Let us mention two approaches that have been made to achieve progress in this area. Both deal with the *first-order closure* of some subclass $\mathcal{L}$ of MSO, meaning the smallest superset of $\mathcal{L}$ that is closed under first-order quantification and boolean combinations.

In [19], the authors develop a technique to infer new separation results dealing with the first-order closure. Specifically, they show the following:

Theorem 4.7 (Janin-Marcinkowski, [19])**.** Let $V, W \subseteq \{\exists, \forall, {\scriptstyle\exists}, {\scriptstyle\forall}\}^*$. Let S be a graph property definable by a prenex normal form of type V but not by one of type W, then there is another property definable by a prenex normal form of type $\exists \, {\scriptstyle\forall\forall} \, V$ but not by one of type $\{{\scriptstyle\exists}, {\scriptstyle\forall}\}^* \, W$.

This technique works for the class of graphs, but it does not work for the classes of words, trees, or pictures. The authors of [19] apply it to show the following corollary (previously shown directly in [1]).

Corollary 4.8. There exists a graph property definable by a prenex normal form of type $\boldsymbol{\exists}^*\{\exists,\forall\}^*\boldsymbol{\exists}^*\{\exists,\forall\}^*$ but not with one of type $\{\exists,\forall\}^*\boldsymbol{\exists}^*\{\exists,\forall\}^*$.

Apart from this, not many separation results are known by now. In fact, to our best knowledge, even the following remains open:

Question 4.9. Is every MSO-formula equivalent to one of the form

$$\boldsymbol{\exists}^*\{\exists,\forall\}^*\boldsymbol{\exists}^*\{\exists,\forall\}^* \quad ?$$

For the class of pictures, [26] contains another preliminary step towards understanding the expressive power of the first-order closure of logics. In that paper, the *MSO alternation hierarchy with first-order closure* is considered. A property belongs to level k of that hierarchy iff it is definable in the first-order closure of the set of mon-Σ^1_k formulas.

Theorem 4.10 (Matz, [26]). The MSO alternation hierarchy with first-order closure is strict.

The proof shows, for example, that there is a prenex normal form of type $\forall^*\boldsymbol{\exists}^*\forall^*\exists^*\boldsymbol{\forall}^*\{\exists,\forall\}^*$ that is not equivalent to a prenex normal form of type $\{\exists,\forall\}^*\boldsymbol{\exists}^*\boldsymbol{\forall}^*\boldsymbol{\exists}^*\{\exists,\forall\}^*$. That means, to exceed some level of the MSO alternation hierarchy with first-order closure, only *two* blocks of set quantifiers are needed.

4.4 Labels and complement

Let us review the mentioned results and see what they imply concerning the question whether the levels of the MSO quantifier alternation hierarchy are closed under complement. Theorem 4.5 considers the class of picture languages over a singleton alphabet and shows that, for every k, there is a picture language that belongs to level $k{+}1$, but not to level k of the MSO alternation hierarchy. This implies

Corollary 4.11. For every $k \geqslant 1$ there exists a $t \geqslant 0$ such that there is a picture language over alphabet $\Sigma := \{0,1\}^t$ which belongs to mon-Σ^1_k but not to mon-Π^1_k.

By standard encoding techniques it can be deduced that $t = 1$ suffices. In other words, if the alphabet Σ is fixed and of size $\geqslant 2$, then all separation results of Figure 4.4 hold. Even more, the above is true also for a singleton alphabet, so Theorem 3.7 can be generalized to:

Theorem 4.12 (Matz, [26]). For every $k \geqslant 1$ there is a picture language over a singleton alphabet which belongs to mon-Σ^1_k but not to mon-Π^1_k.

$$\begin{tikzcd}
\text{mon-}\Sigma^1_{k+1} \arrow[dr, no head] & \neq & \text{mon-}\Pi^1_{k+1} \arrow[dl, no head] \\
& \text{mon-}\Delta^1_{k+1} \arrow[d, no head] & \\
& \mathrm{BC}(\text{mon-}\Sigma^1_k) \arrow[dl, no head] \arrow[dr, no head] & \\
\text{mon-}\Sigma^1_k & \neq & \text{mon-}\Pi^1_k
\end{tikzcd}$$

FIGURE 1. The MSO quantifier alternation hierarchy

A picture language which witnesses the difference between mon-Σ^1_k and mon-Π^1_k is the set of all pictures of size $m \times n$ for which n is not a multiple of $f(m)$, where f is a specific $(k+1)$-fold exponential function.

Again, the witness sentence actually makes little use of set quantifiers. For example, if $k = 5$, it is of the form

$$\exists^* {\scriptstyle\forall}^* {\scriptstyle\exists}^* {\scriptstyle\forall}^* \exists^* \{{\scriptstyle\exists}, {\scriptstyle\forall}\}^*.$$

References

[1] M. Ajtai, R. Fagin, and L. J. Stockmeyer. The closure of monadic np. *J. Comput. Syst. Sci.*, 60(3):660–716, 2000.

[2] A. Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. *ITA*, 33(4/5):329–340, 1999.

[3] J. Bradfield and C. Stirling. Modal logics and mu-calculi: an introduction. In *Handbook of process algebra*, pages 293–330. North-Holland, Amsterdam, 2001.

[4] J. C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theor. Comput. Sci.*, 195(2):133–153, 1998.

[5] L. Brim, J. Gruska, and J. Zlatuska, editors. *Mathematical Foundations of Computer Science 1998, 23rd International Symposium, MFCS'98, Brno, Czech Republic, August 24-28, 1998, Proceedings*, volume 1450 of *Lecture Notes in Computer Science*. Springer, 1998.

[6] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

[7] G. Castiglione and R. Vaglica. Recognizable picture languages and polyominoes. In *Proceedings of the 2nd International Conference on Algebraic Informatics, Thessaloniki, May 21–25, 2007*, Thessaloniki, Greece, 2007. Aristotle University of Thessaloniki Department of Mathematics. To appear.

[8] A. K. Chandra and D. Harel. Structure and complexity of relational queries. *J. Comput. Syst. Sci.*, 25(1):99–128, 1982.

[9] R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *J. Comput. Syst. Sci.*, 5(1):1–16, 1971.

[10] J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.

[11] A. Durand and M. More. Nonerasing, counting, and majority over the linear time hierarchy. *Inf. Comput.*, 174(2):132–142, 2002.

[12] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.

[13] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of computation (Proc. SIAM-AMS Sympos. Appl. Math., New York, 1973)*, pages 43–73. SIAM–AMS Proc., Vol. VII, Providence, R.I., 1974. Amer. Math. Soc.

[14] A. Ferreira and H. Reichel, editors. *STACS 2001, 18th Annual Symposium on Theoretical Aspects of Computer Science, Dresden, Germany, February 15-17, 2001, Proceedings*, volume 2010 of *Lecture Notes in Computer Science*. Springer, 2001.

[15] D. Giammarresi. Two-dimensional languages and recognizable functions. In *Developments in Language Theory*, pages 290–301, 1993.

[16] D. Giammarresi and A. Restivo. Recognizable picture languages. *IJPRAI*, 6(2&3):241–256, 1992.

[17] D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Inf. Comput.*, 125(1):32–45, 1996.

[18] E. Grandjean. Sorting, linear time and the satisfiability problem. *Ann. Math. Artif. Intell.*, 16:183–236, 1996.

[19] D. Janin and J. Marcinkowski. A toolkit for first order extensions of monadic games. In Ferreira and Reichel [14], pages 353–364.

[20] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In U. Montanari and V. Sassone, editors, *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996.

[21] O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *STOC*, pages 224–233, 1998.

[22] G. Lenzi. A hierarchy theorem for the μ-calculus. In F. M. auf der Heide and B. Monien, editors, *ICALP*, volume 1099 of *Lecture Notes in Computer Science*, pages 87–97. Springer, 1996.

[23] R. Mateescu. Local model-checking of modal mu-calculus on acyclic labeled transition systems. In J.-P. Katoen and P. Stevens, editors, *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 281–295. Springer, 2002.

[24] O. Matz. On piecewise testable, starfree, and recognizable picture languages. In *Foundations of software science and computation structures (Lisbon, 1998)*, volume 1378 of *Lecture Notes in Comput. Sci.*, pages 203–210. Springer, Berlin, 1998.

[25] O. Matz. One quantifier will do in existential monadic second-order logic over pictures. In Brim et al. [5], pages 751–759.

[26] O. Matz. Dot-depth, monadic quantifier alternation, and first-order closure over grids and pictures. *Theor. Comput. Sci.*, 270(1-2):1–70, 2002.

[27] O. Matz, N. Schweikardt, and W. Thomas. The monadic quantifier alternation hierarchy over grids and graphs. *Inf. Comput.*, 179(2):356–383, 2002.

[28] O. Matz and W. Thomas. The monadic quantifier alternation hierarchy over graphs is infinite. In *LICS*, pages 236–244, 1997.

[29] M. More and F. Olive. Rudimentary languages and second order logic. *Math. Log. Q.*, 43:419–426, 1997.

[30] D. Niwinski. On fixed-point clones (extended abstract). In L. Kott, editor, *ICALP*, volume 226 of *Lecture Notes in Computer Science*, pages 464–473. Springer, 1986.

[31] M. Otto. An note on the number of monadic quantifiers in monadic Σ^1_1. *Inf. Process. Lett.*, 53(6):337–339, 1995.

[32] A. Potthoff. *Logische Klassifizierung regulärer Baumsprachen.* PhD thesis, Christian-Albrechts-Universität zu Kiel, Germany, 1994.

[33] K. Reinhardt. On some recognizable picture-languages. In Brim et al. [5], pages 760–770.

[34] K. Reinhardt. The #a = #b pictures are recognizable. In Ferreira and Reichel [14], pages 527–538.

[35] N. Schweikardt. The monadic quantifier alternation hierarchy over grids and pictures. In M. Nielsen and W. Thomas, editors, *CSL*, volume 1414 of *Lecture Notes in Computer Science*, pages 441–460. Springer, 1997.

[36] N. Schweikardt. On the expressive power of monadic least fixed point logic. *Theor. Comput. Sci.*, 350(2-3):325–344, 2006.

[37] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[38] W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25(3):360–376, 1982.

[39] W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume III, Beyond Words, pages 389–455. Springer, Berlin, 1997.

[40] I. Walukiewicz. Notes on the propositional μ-calculus: Completeness and related results. Technical Report NS-95-1, BRICS, Department of Computer Science, University of Aarhus, Denmark, 1995.

Structured strategies in games on graphs

R. Ramanujam
Sunil Simon

The Institute of Mathematical Sciences
Central Institutes of Technology (C. I. T.) Campus, Taramani
Chennai 600 113, India
{jam,sunils}@imsc.res.in

Abstract

We study two-player non-zero sum games of perfect information in infinite games on graphs. We suggest that in such games, it is useful to study structurally specified strategies, so that we can reason about how a player's strategy may depend on assumptions about the opponent's strategy. In such a setting, we show that best response computation can be carried out in games with Muller objectives. We discuss a simple modal logic in which we can reason about how a player can ensure an outcome by following a specific strategy.

1 Summary

We discuss strategies in non-zero sum games of perfect information on graphs. The study of non-zero sum games on graphs is motivated by the advent of computational tasks on the world-wide web and related security requirements which have thrown up many interesting areas of interaction between game theory and computer science. For example, signing contracts on the web requires interaction between principals who do not know each other and typically distrust each other. Protocols of this kind which involve *selfish agents* can be easily viewed as strategic games of imperfect information. These are complex interactive processes which critically involve players reasoning about each others' strategies to decide on how to act. In the case of interacting web services, these games involve infinite plays as well. Developing a game theoretic computational study of such interactions is an interesting challenge. Admittedly, these are games of partial information, but a theoretical analysis is interesting even in the more restricted case of perfect information.

On one hand, zero sum games on graphs have been extensively studied in logic and automata theory [5], and on the other, a rich theory of non-zero sum *matrix form* games has been developed by game theorists [8]. We call graph games large, to indicate that plays consist of (long) sequences of moves, whereas matrix form games are termed small, in the sense that a play is typically one simultaneous move. We can have matrix form presentations for sequential plays as well, but not very usefully for analysis.

While one talks of winning strategies in win / loss games, when players have overlapping objectives, we consider the best response each player can offer to moves of other players. In a small game which consists of both players deciding on a move simultaneously, it is best analyzed by considering pairs of moves. When we have a pair (a, b) such that a is player 1's best response to player 2 deciding on b, as well as the other way about, they constitute a Nash equilibrium: there is no incentive for rational players to unilaterally deviate from such a decision. Thus equilibrium concepts predict rational play, and games are so designed that equilibrium behaviour achieves desired outcomes. Nash's theorem asserts the existence of equilibria in the space of randomized strategies and game theory offers similar theorems for related notions of equilibria.

Equating equilibria with rational play rests on the following analysis: at a game position a rational player would choose the best response to the opponent's strategy which (by assumption of rationality of the opponent) must be his best possible choice of move. Thus, the reasoning critically involves players reasoning about other players' strategies. When strategies consist of picking one move out of a set of possible moves, such as in small games, this is clear. When strategies use the current history of play to make a local move when the eventual outcome is not as yet determined, the situation is much less clear.

A strategy is a function from the set of partial plays to moves: it advises a player at a game position on the choice she can make. In a large game, this amounts to a complete specification of behaviour in all possible game situations. But then in such a game, one player's knowledge of the strategies employed by the other is necessarily partial. Rational play requires much finer analysis since strategies have structure that depends on the player's observations of game positions, history of play and the opponent's apparent strategies.

Such study of structure in strategies is relevant even in finite, determined, but large, zero-sum games. A classic example of such a game is the game of chess. Zermello showed in [14] that chess is determined, i.e. from every game position, either there exists a (pure) strategy for one of the two players (white or black) guaranteeing that she will win or each one of the two players has a strategy guaranteeing at least a draw. However, given any game position, we do not know which of the three alternatives is the correct one. For games like Hex, it is known that the first player can force a win [3] but nonetheless a winning strategy is not known. Again, in such situations, rather than be content with reasoning *about games* using the functional notion of strategies, one needs to reason *about strategies* themselves. For instance, most of the chess playing programs use heuristics which are basically partially specified strategies. A library of such specifications is developed and during the course of play, the actual strategy is built up by composing various partial strategies.

Thus we are led to the idea of strategies specified in a syntax, and composed structurally, with a player's strategies built up using assumptions about another. The notion of strategy composition is inspired by an analogous notion of *game composition* proposed by Rohit Parikh [9] who initiated the study of game structure using algebraic properties.

In this paper, we suggest that standard automata theoretic techniques can be employed to usefully specify and analyze partial strategies in non-zero games on graphs. We propose a syntactic framework for strategies in which best response can be algorithmically determined, and a simple modal logic in which we can reason about such strategies. This proposal is intended more as an illustration of such analysis; ideally, we need a "programming language" for strategies, whose structure should be determined empirically by how well they describe interesting heuristics employed in many classes of games that arise in applications mentioned above.

Related work

Automata theoretic analyses of two-player zero-sum infinite games of perfect information [5] have led to interesting applications in the design and verification of reactive systems and in control synthesis. We use this technical machinery, but in the non-zero sum context.

As remarked earlier, the logical structure we study is inspired by propositional game logic [9]. Pauly [10] has built on this to provide interesting relationships between programs and games, and to describe coalitions to achieve desired goals. Bonanno [2] suggested obtaining game theoretic solution concepts as characteristic formulas in modal logic. van Benthem [12] uses dynamic logic to describe games as well as (atomic) strategies.

On the other hand, the work on Alternating Temporal Logic [1] considers selective quantification over paths that are possible outcomes of games in which players and an environment alternate moves. Here, we talk of the existence of a strategy for a coalition of players to force an outcome. [4] draws parallels between these two lines of work, that of Pauly's coalition logics and alternating temporal logic. In the work of [6] and [13], van der Hoek and co-authors develop logics for strategic reasoning and equilibrium concepts.

The underlying reasoning, whether explicitly described (as in game logics) or implicit (as in automata theoretic studies) is carried out in a logic of games and the reasoning is about *existence* of strategies, rather than about strategies themselves. For instance, the existence of an appropriate strategy in sub-games is used to argue the existence of one in the given game. Moreover, most of the techniques involve win / lose games. Thus our departure consists in considering non-zero sum games and (hence) structured partial strategies.

In [11], we presented an axiomatization of the logic we discuss here. In this paper, the emphasis is more on showing how standard automata theoretic techniques can be employed to solve the associated algorithmic questions.

2 Games and strategies

We begin with a description of the game arena. We use the graphical model for extensive form turn-based games, where at most one player gets to move at each game position.

Game arena

Let $N = \{1, 2\}$ be the set of players and $\Sigma = \{a_1, a_2, \ldots, a_m\}$ be a finite set of action symbols, which represent moves of players.

A **game arena** is a finite graph $\mathcal{G} = (W^1, W^2, \longrightarrow, w_0)$ where W^i is the set of *game positions* of player i for $i \in \{1, 2\}$. Let $W = W^1 \cup W^2$. The transition function $\longrightarrow: (W \times \Sigma) \to W$ is a partial function also called the move function and w_0 is the initial node of the game. Let $\bar{\imath} = 2$ when $i = 1$ and $\bar{\imath} = 1$ when $i = 2$.

Let the set of successors of $w \in W$ be defined as $\overrightarrow{w} = \{w' \in W \mid w \xrightarrow{a} w'$ for some $a \in \Sigma\}$. We assume that for all game positions w, $\overrightarrow{w} \neq \varnothing$.

In an arena, the play of a game can be viewed as placing a token on w_0. If player i owns the game position w_0 (i.e $w_0 \in W^i$), then she picks an action 'a' which is enabled for her at w_0 and moves the token to w' where $w_0 \xrightarrow{a} w'$. The game then continues from w'. Formally, a play in $\mathcal{G}$ is an infinite path $\rho : w_0 a_0 w_1 a_1 \cdots$ where $\forall j : w_j \xrightarrow{a_j} w_{j+1}$. Let Plays denote the set of all plays in the arena.

Games and winning conditions

Let $\mathcal{G}$ be an arena as defined above. The arena merely defines the rules about how the game progresses and terminates. More interesting are the winning conditions of the players, which specify the game outcomes. Since we consider non-zero sum games, players' objectives need not be strictly conflicting, and each player has a preference relation inducing an ordering over the set of valid plays. The game is specified by presenting the game arena along with the preference relation for each player. Let $\preceq^i \subseteq (\text{Plays} \times \text{Plays})$ be a complete, reflexive, transitive binary relation denoting the preference relation of player i for $i \in \{1, 2\}$. Then the game G is given as, $G = (\mathcal{G}, \{\preceq^i\}_{i \in \{1,2\}})$.

In general, the preference relation need not have a finite presentation, and we restrict our attention to *finite state* preferences. (This is because in the applications we have in mind, as in network games, desired or preferred plays are easily expressed as formulas of temporal logics.) Thus, the preferences of players are presented as finite state evaluation automata, with Muller acceptance conditions.

Let $\mathcal{M} = (R, \Delta, r_0)$ be a deterministic automaton with finite set of states R, initial state $r_0 \in R$ and transition function $\Delta : R \times W \times \Sigma \to R$. The evaluation automaton is given by: $\mathcal{E} = (\mathcal{M}, \{\lhd^i\}_{i \in \{1,2\}})$ where $\lhd^i \subseteq (\mathcal{F} \times \mathcal{F})$ is a total order over $\mathcal{F} = 2^R \setminus \varnothing$ for $i \in \{1, 2\}$.

A run of $\mathcal{E}$ on a play $\rho : s_0 a_0 \cdots \in$ Plays is a sequence of states $\varphi : r_0 r_1 \cdots$ such that $\forall i : 0 \leq i < n$, we have $r_{i+1} = \Delta(r_i, s_i, a_i)$. Let $\inf(\varphi)$ denote the set of states occurring infinitely often in φ. The evaluation automaton $\mathcal{E}$ induces a preference ordering on Plays in the following manner. Let $\rho : s_0 a_0 s_1 \cdots$ and $\rho' : s_0 a'_0 s'_1 \cdots$ be two plays. Let the run of $\mathcal{E}$ on ρ and ρ' be $\varphi : r_0 r_1 \cdots r_n$ and $\varphi' : r_0 r'_1 \cdots r'_n$ respectively. For $i \in \{1, 2\}$, we have $\rho \preceq^i \rho'$ iff $\inf(\varphi) \lhd^i \inf(\varphi')$. A game is presented as $G = (\mathcal{G}, \mathcal{E})$.

We shall also be interested in *binary* evaluation automata which specify least outcomes for player i. Such a automaton is given by $\mathcal{E}^i_F$, where $F \in 2^R$: for every $F' \in 2^R$, if $F \lhd^i F'$, it is taken to be "winning" for player i, and every $F'' \neq F$ such that $F'' \lhd^i F$ is taken to be "losing". Such an automaton checks if i can ensure an outcome which is at least as preferred as F. Note that the terminology of win / loss is only to indicate a binary preference for player i, and applies even in the context of non-zero sum games.

Thus we have game arenas, with players' preference on plays. We now discuss strategies of players.

Strategies

Let $\mathcal{G}_T$ denote the tree unfolding of the arena $\mathcal{G}$. We use s, s' to denote the nodes in $\mathcal{G}_T$. A strategy for player 1, $\mu = (W_\mu, \longrightarrow_\mu, s_0)$ is a maximal connected subtree of $\mathcal{G}_T$ where for each player 1 node, there is a unique outgoing edge and for the other player every move is included. That is, for $s \in W_\mu$ the edge relation satisfies the following property:

- if $s \in W^1_\mu$ then there exists a unique $a \in \Sigma$ such that $s \xrightarrow{a}_\mu s'$, where we have $s \xrightarrow{a}_T s'$.
- if $s \in W^2_\mu$, then for each s' such that $s \xrightarrow{a}_T s'$, we have $s \xrightarrow{a}_\mu s'$.

Let Ω^i denote the set of all strategies of Player i in $\mathcal{G}$, for $i = 1, 2$. We shall use μ to denote a strategy of player 1 and τ a strategy of player 2. A strategy profile $\langle \mu, \tau \rangle$ defines a unique path ρ^τ_μ in the arena $\mathcal{G}$.

In games with overlapping objectives, the common solution concept employed is that of an equilibrium strategy profile [7]. A profile of strategies, one for each player, is said to be in equilibrium if no player gains by unilaterally deviating from his strategy. The notion of equilibrium can be formally defined as follows. Let μ denote a strategy of player 1 and τ denote a strategy of player 2.

- μ is the best response for τ iff $\forall \mu' \in \Omega^1$, $\rho^{\tau}_{\mu'} \preceq^1 \rho^{\tau}_{\mu}$.
- τ is the best response for μ iff $\forall \tau' \in \Omega_2$, $\rho^{\tau'}_{\mu} \preceq^2 \rho^{\tau}_{\mu}$.
- $\langle \mu, \tau \rangle$ is a Nash equilibrium iff μ is the best response for τ and τ is the best response for μ.

The natural questions that are of interest include:

- Given a strategy τ of player 2, what is the best response for player 1?
- Given a strategy profile $\langle \mu, \tau \rangle$, is it a Nash equilibrium?
- Does the game possess a Nash equilibrium?

Clearly, if we can answer the first question, we can answer the second as well. In any case, to study these questions algorithmically, we need to be able to present the preferences of players and their strategies in a finite fashion. We have evaluation automata presenting preferences; we now proceed to a syntax for strategies.

3 Strategy specification

We conceive of strategies as being built up from atomic ones using some grammar. The atomic case specifies, for a player, what conditions she tests for before making a move. We can associate with the game graph a set of observables for each player. One elegant method then, is to state the conditions to be checked as a past time formula of a simple tense logic over the observables. The structured strategy specifications are then built from atomic ones using connectives. We crucially use an implication of the form: "if the opponent is apparently playing a strategy π then play σ".

Below, for any countable set X, let Past(X) be sets of formulas given by the following syntax:

$$\psi \in \mathrm{Past}(X) := x \in X \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \Diamond\!\!\!\!-\psi.$$

Syntax

Let $P^i = \{p^i_0, p^i_1, \ldots\}$ be a countable set of observables for $i \in \{1,2\}$ and let $P = P^1 \cup P^2$. The syntax of strategy specifications is then given by:

$$\mathrm{Strat}^i(P^i) := \mathsf{null} \mid [\psi \mapsto a]^i \mid \sigma_1 + \sigma_2 \mid \sigma_1 \cdot \sigma_2 \mid \pi \Rightarrow \sigma_1$$

where $\pi \in \mathrm{Strat}^{\bar{\imath}}(P^1 \cap P^2)$ and $\psi \in \mathrm{Past}(P^i)$.

Semantics

Given any sequence $\xi = t_0 t_1 \cdots t_m$, $V : \{t_0, \cdots, t_m\} \to 2^X$, and k such that $0 \leq k \leq m$, the truth of a past formula $\psi \in \mathrm{Past}(X)$ at k, denoted $\xi, k \models \psi$ can be defined as follows:

- $\xi, k \models p$ iff $p \in V(s_k)$.
- $\xi, k \models \neg\psi$ iff $\xi, k \not\models \psi$.
- $\xi, k \models \psi_1 \vee \psi_2$ iff $\xi, k \models \psi_1$ or $\xi, k \models \psi_2$.
- $\xi, k \models \Diamond\!\!\!-\psi$ iff there exists a $j : 0 \leq j \leq k$ such that $\xi, j \models \psi$.

We consider the game arena $\mathcal{G}$ along with a valuation function for the observables $V : W \to 2^P$. We assume the presence of two special propositions τ_i for each $i \in \{1, 2\}$ which specify at a game position, which player's turn it is to move, i.e. $\tau_i \in V(w)$ iff w is a player i game position. Given a strategy μ of player i and a node $s \in \mu$, let $\rho_s : s_0 a_0 s_1 \cdots s_m = s$ be the unique path in μ from the root node to s. For a strategy specification $\sigma \in \mathrm{Strat}^i(P^i)$, we define when μ conforms to σ (denoted $\mu \models_i \sigma$) as follows:

- $\mu \models_i \sigma$ iff for all player i nodes $s \in \mu$, we have $\rho_s, s \models_i \sigma$.

where we define $\rho_s, s_j \models_i \sigma$ for any player i node s_j in ρ_s as,

- $\rho_s, s_j \models_i \mathsf{null}$ for all ρ_s, s_j.
- $\rho_s, s_j \models_i [\psi \mapsto a]^i$ iff $\rho_s, s_j \models \psi$ implies $\mathrm{out}_{\rho_s}(s_j) = a$.
- $\rho_s, s_j \models_i \sigma_1 + \sigma_2$ iff $\rho_s, s_j \models_i \sigma_1$ or $\rho_s, s_j \models_i \sigma_2$.
- $\rho_s, s_j \models_i \sigma_1 \cdot \sigma_2$ iff $\rho_s, s_j \models_i \sigma_1$ and $\rho_s, s_j \models_i \sigma_2$.
- $\rho_s, s_j \models_i \pi \Rightarrow \sigma_1$ iff for all player $\bar{\imath}$ nodes $s_k \in \rho_s$ such that $k \leq j$, if $\rho_s, s_k \models_{\bar{\imath}} \pi$ then $\rho_s, s_j \models_i \sigma_1$.

Above, $\pi \in \mathrm{Strat}^{\bar{\imath}}(P^1 \cap P^2)$, $\psi \in \mathrm{Past}(P^i)$, and for all $i : 0 \leq i < m$, $\mathrm{out}_{\rho_s}(s_i) = a_i$ and $\mathrm{out}_{\rho_s}(s)$ is the unique outgoing edge in μ at s.

Remarks

Note that we do not have negation in specifications. One reason is that they are partial, and hence the semantics is not immediate. If we were to consider a specification of the form $\overline{\pi} \Rightarrow \sigma$, we could interpret this as: if player has seen that opponent has violated π in the past, then play σ. This seems rather unnatural, and hence, for the present, we are content to leave negation aside. Note that we do have negation in tests in atomic specifications, and later we shall embed these specifications into a modal logic (with negation on formulas).

When we consider repeated or multi-stage games, we have strategy *switching*, whereby players receive payoffs at specified points, and depending on the outcomes, decide on what new strategies to adopt later. Then it makes sense to include specifications whereby a player conforms to a strategy until some observable change, and then switches to another strategy. In this context, we have (a form of) sequential composition as well as iteration. However, operators are best added after a systematic study of their algebraic properties. We stick to a simple presentation here since our main aim is only to describe the framework. As we shall see below, any set of specifications that allows effective automaton construction will do.

Clearly, each strategy specification defines a set of strategies. We now show that it is a *regular* set, recognizable by a finite state device. In the spirit of prescriptive game theory, we call them advice automata.

Advice Automata

For a game graph $\mathcal{G}$, a nondeterministic advice automaton for player i is a tuple $\mathcal{A} = (Q, \delta, o, I)$ where Q is the set of states, $I \subseteq Q$ is the set of initial states, $\delta : Q \times W \times \Sigma \to 2^Q$ is the transition relation, and $o : Q \times W^i \to \Sigma$, is the output or advice function.

The language accepted by the automaton is a set of strategies of player i. Given a strategy $\mu = (W_\mu, \longrightarrow_\mu, s_0)$ of player i, a run of $\mathcal{A}$ on μ is a Q labelled tree $T = (W_\mu, \longrightarrow_\mu, \lambda)$, where λ maps each tree node to a state in Q as follows: $\lambda(s_0) \in I$, and for any s_k where $s_k \xrightarrow{a}_\mu s'_k$, we have $\lambda(s'_k) \in \delta(\lambda(s_k), s_k, a_k)$.

A Q-labelled tree T is accepted by $\mathcal{A}$ if for every tree node $s \in W^i_\mu$, if $s \xrightarrow{a}_T s'$ then $o(\lambda(s)) = a$. A strategy μ is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on μ.

It is easy to see that any bounded memory strategy can be represented using a *deterministic* advice automaton. In such a framework we can ask, given a bounded memory strategy for player 2 represented by a *deterministic* strategy automaton B, can we compute the best response for player 1?

Proposition 3.1. Given a game $G = (\mathcal{G}, \mathcal{E})$ and a deterministic advice automaton B for player 2, the best response for player 1 can be effectively computed.

The proposition is proved easily. For each $F \in 2^R$, we can construct a nondeterministic automaton A_F which explores paths of $\mathcal{G}$ as follows. It consults B to pick player 2's moves and simply guesses 1's moves. It runs the binary evaluation automaton $\mathcal{E}^1_F$ for player 1 in parallel and checks if the run is winning for player 1. Now, we can enumerate the $F \in 2^R$ in such a way that those higher in $\lhd^1$ appear earlier in the enumeration. We try automata A_F in this order.

Therefore, given an strategy profile presented as advice automaton for each of the players, we can also check if a strategy profile constitutes a Nash equilibrium. However, we are interested in strategy specifications which are partial and hence constitute nondeterministic advice automata. The following lemma relates structured strategy specifications to advice automata.

Lemma 3.2. Given a player $i \in \{1, 2\}$ and a strategy specification σ, we can construct an advice automaton $\mathcal{A}_\sigma$ such that $\mu \in \text{Lang}(\mathcal{A}_\sigma)$ iff $\mu \models_i \sigma$.

Proof. The construction of automata is inductive, on the structure of specifications. Note that the strategy is implemented principally by the output function of the advice automaton.

For a strategy specification σ, let $SF(\sigma)$ denote the subformula closure of σ and $SF_\psi(\sigma)$ denote the Past subformulas in σ. Call $R \subseteq SF_\psi(\sigma)$ an atom if it is propositionally consistent and complete: that is, for every $\neg\gamma \in SF_\psi(\sigma)$, $\neg\gamma \in R$ iff $\gamma \notin R$, and for every $\gamma_1 \vee \gamma_2 \in SF_\psi(\sigma)$, $\gamma_1 \vee \gamma_2 \in R$ iff $\gamma_1 \in R$ or $\gamma_2 \in R$.

Let $\mathcal{AT}_\sigma$ denote the set of atoms. Let $C_0 = \{C \in \mathcal{AT}_\sigma |$ there does not exist any $\diamondminus\psi \in C\}$. For $C, D \in \mathcal{AT}_\sigma$,define $C \longrightarrow D$ iff for all $\diamondminus\psi \in SF_\psi(\sigma)$, the following conditions hold.

- $\psi \in C \Rightarrow \diamondminus\psi \in D$
- $\diamondminus\psi \in D \Rightarrow \psi \in C$ or $\diamondminus\psi \in C$.

We proceed by induction on the structure of σ. We construct automata for atomic strategies and compose them for complex strategies.
($\sigma \equiv [\psi \mapsto a]$): The automaton works as follows. Its states keep track of past formulas satisfied along a play as game positions are traversed and that the valuation respects the constraints generated for satisfying ψ. The automaton also guesses a move at every step and checks that this is indeed a when ψ holds; in such a case this is the output of the automaton. Formally:
$\mathcal{A}_\sigma = (Q_\sigma, \delta_\sigma, o_\sigma, I_\sigma)$, where

- $Q_\sigma = \mathcal{AT}_\sigma \times \Sigma$.
- $I_\sigma = \{(C, x) | C \in C_0, V(s_0) = C \cap P_\sigma, x \in \Sigma\}$.
- For a transition $s \xrightarrow{a} s'$ in $\mathcal{G}$, we have:
$\delta_\sigma((C, x), s, a) = \{(C', y) | C \longrightarrow C', V(s') = C' \cap P_\sigma, y \in \Sigma\}$.
- $o((C, x), s) = \begin{cases} a & \text{if } \psi \in C \\ x & \text{otherwise} \end{cases}$

We now prove the assertion in the lemma that $\mu \in \text{Lang}(\mathcal{A}_\sigma)$ iff $\mu \models_i \sigma$.

($\Rightarrow$). Suppose $\mu \in \text{Lang}(\mathcal{A}_\sigma)$. Let $T = (W^1_\mu, W^2_\mu, \longrightarrow_T, \lambda)$ be the Q-labelled tree accepted by $\mathcal{A}_\sigma$. We need to show that for all $s \in W_\mu$, we have $\rho_s, s \models \psi$ implies $\text{out}(s) = a$.

The following claim, easily proved by structural induction on the structure of ψ, using the definition of $\longrightarrow$ on atoms, asserts that the states of the automaton check the past requirements correctly. Below we use the notation $\psi \in (C, x)$ to mean $\psi \in C$.

Claim 3.3. For all $s \in W_\mu$, for all $\psi' \in SF_\psi(\sigma)$, $\psi' \in \lambda(s)$ iff $\rho_s, s \models \psi'$.

Assume the claim and consider any $s \in W_\mu$. From claim 3.3, we have $\rho_s, s \models \psi$ implies $\psi \in \lambda(s)$. By the definition of o, we have $o(\lambda(s), s) = a$.

($\Leftarrow$). Suppose $\mu \models_1 [\psi \mapsto a]$. From the semantics, we have $\forall s \in W^1_\mu, \rho_s, s \models \psi$ implies $\text{out}(s) = a$. We need to show that there exists a Q-labelled tree accepted by $\mathcal{A}_\sigma$. For any s let the Q-labelling be defined as follows. Fix $x_0 \in \Sigma$.

- For $s \in W^1_\mu$, let $\lambda(s) = (\{\psi' \in SF_\psi(\sigma) | \rho_s, s \models \psi'\}, \text{out(s)})$.

- For $s \in W^2_\mu$, let $\lambda(s) = (\{\psi' \in SF_\psi(\sigma) | \rho_s, s \models \psi'\}, x_0)$.

It is easy to check that $\lambda(s)$ constitutes an atom and the transition relation is respected. By the definition of o, we get that it is accepting.
($\sigma \equiv \sigma_1 \cdot \sigma_2$): By induction hypothesis there exist $\mathcal{A}_{\sigma_1} = (Q_{\sigma_1}, \delta_{\sigma_1}, o_{\sigma_1}, I_{\sigma_1})$ and $\mathcal{A}_{\sigma_2} = (Q_{\sigma_2}, \delta_{\sigma_2}, o_{\sigma_2}, I_{\sigma_2})$ which accept all strategies satisfying σ_1 and σ_2 respectively. To obtain an automaton which accepts all strategies which satisfy $\sigma_1 \cdot \sigma_2$ we just need to take the product of $\mathcal{A}_{\sigma_1}$ and $\mathcal{A}_{\sigma_2}$.
($\sigma \equiv \sigma_1 + \sigma_2$): We take $\mathcal{A}_\sigma$ to be the disjoint union of $\mathcal{A}_{\sigma_1}$ and $\mathcal{A}_{\sigma_2}$. Since the automaton is nondeterministic with multiple initial states, we retain the initial states of both $\mathcal{A}_{\sigma_1}$ and $\mathcal{A}_{\sigma_2}$. If a run starts in an initial state of $\mathcal{A}_{\sigma_1}$ then it will never cross over into the state space of $\mathcal{A}_{\sigma_2}$ and vice versa.
($\sigma \equiv \pi \Rightarrow \sigma'$): By induction hypothesis we have $\mathcal{A}_\pi = (Q_\pi, \delta_\pi, o_\pi, I_\pi)$ which accepts all player 2 strategies satisfying π and $\mathcal{A}_{\sigma'} = (Q_{\sigma'}, \delta_{\sigma'}, o_{\sigma'}, I_{\sigma'})$ which accepts all player 1 strategies satisfying σ'.

The automaton $\mathcal{A}_\sigma$ has the product states of $\mathcal{A}_\pi$ and $\mathcal{A}_{\sigma'}$ as its states along with a special state q_{free}. The automaton keeps simulating both $\mathcal{A}_\pi$, $\mathcal{A}_{\sigma'}$ and keeps checking if the path violates the advice given by $\mathcal{A}_\pi$, if so it moves into state q_{free} from which point onwards it is "free" to produce any advice. Till π is violated, it is forced to follow the transitions of $\mathcal{A}_{\sigma'}$.

Define $\mathcal{A}_\sigma = (Q, \delta, o, I)$ where $Q = (Q_\pi \times Q_{\sigma'}) \cup (q_{\text{free}} \times \Sigma)$. The transition function is given as follows:

- For $s \in W^1_\mu$, we have $\delta((q_\pi, q_{\sigma'}), s, a) = \{(q_1, q_2) | q_1 \in \delta_\pi(q_\pi, s, a)$ and $q_2 \in \delta_{\sigma'}(q_{\sigma'}, s, a)\}$.
- For $s \in W^2_\mu$, we have:
  - If $o_\pi(q_\pi, s) \neq a$, then $\delta((q_\pi, q_{\sigma'}), s, a) = \{(q_{\text{free}}, a) | a \in \Sigma\}$.
  - If $o_\pi(q_\pi, s) = a$, then $\delta((q_\pi, q_{\sigma'}), s, a) = \{(q_1, q_2) | q_1 \in \delta_\pi(q_\pi, s, a)$ and $q_2 \in \delta_{\sigma'}(q_{\sigma'}, s, a)\}$.
- $\delta((q_{\text{free}}, x), s, a) = \{(q_{\text{free}}, a) | a \in \Sigma\}$

The output function is defined as follows: For $s \in W^1_\mu$, $o((q_\pi, q_{\sigma'}), s) = o_{\sigma'}(q_{\sigma'}, s)$ and $o((q_{\text{free}}, x), s) = x$.

The automaton keeps simulating both $\mathcal{A}_\pi$, $\mathcal{A}_{\sigma'}$ and keeps checking if the path violates π. If so it moves into state q_{free} from which point onwards it is not constrained to follow σ'. Q.E.D.

4 Best response

Since a strategy specification denotes a set of strategies satisfying certain propeties, notions like strategy comparison and best response with respect to strategy specifications need to be redefined.

Given a game arena $G = (\mathcal{G}, \mathcal{E})$ and a strategy specification π for player $\bar{\imath}$, we can have different notions as to when a specification for player i is "better" than another.

- $\text{Better}_1(\sigma, \sigma')$: if there is an $F \in 2^R$, then there is a μ' with $\mu' \models_i \sigma'$ such that for all τ with $\tau \models_{\bar{\imath}} \pi$, $\rho^\tau_{\mu'}$ is winning with respect to $\mathcal{E}^i_F$ then there is μ with $\mu \models_i \sigma$ such that for all τ with $\tau \models_{\bar{\imath}} \pi$, ρ^τ_μ is winning with respect to $\mathcal{E}^i_F$.
  The predicate $\text{Better}_1(\sigma, \sigma')$ says that, for some (binary) outcome F, if there is a strategy conforming to the specification σ' which ensures winning $\mathcal{E}^i_F$ then there also exists a strategy conforming to σ which ensures winning $\mathcal{E}^i_F$ as well.
- $\text{Better}_2(\sigma, \sigma')$: if there is $F \in 2^R$ such that for all μ' with $\mu' \models_i \sigma'$, for all τ with $\tau \models_{\bar{\imath}} \pi$, $\rho^\tau_{\mu'}$ is winning with respect to $\mathcal{E}^i_F$ then for all μ with $\mu \models_i \sigma$, for all τ with $\tau \models_{\bar{\imath}} \pi$, ρ^τ_μ is winning with respect to $\mathcal{E}^i_F$.
  This notion is best understood contrapositively: for some (binary) outcome F, whenever there is a strategy conforming to σ which is not winning for $\mathcal{E}^i_F$, there also exists a strategy conforming to σ' which is not winning for $\mathcal{E}^i_F$. This can be thought of as a soundness condition. A risk averse player might prefer this way of comparison.

To algorithmically compare strategies, we first need to be able to decide the following questions. Let σ and π be strategy specifications for player i and player $\bar{\imath}$ and $\mathcal{E}^i_F$ a binary evaluation automaton for player i.

- Does player i have a strategy conforming to σ which ensures a valid play which is winning for i with respect to $\mathcal{E}^i_F$, as long as player $\bar{\imath}$ is playing a strategy conforming to π (abbreviated as $\exists\sigma, \forall\pi : \mathcal{E}^i_F$)?
- Is it the case that for all strategies of player i conforming to σ, as long as player $\bar{\imath}$ is playing a strategy conforming to π, the result will be a valid play which is winning for i with respect to $\mathcal{E}^i_F$ (abbreviated as $\forall\sigma, \forall\pi : \mathcal{E}^i_F$)?

We call this the *verification* question. The *synthesis* question is given π and $\mathcal{E}^i_F$ to construct a specification σ such that $\exists\sigma, \forall\pi : \mathcal{E}^i_F$ holds.

Once we can show that the verification question is decidable and synthesis possible, the game theoretic questions of interest include: For a game $G = (\mathcal{G}, \mathcal{E})$,

- Given strategy specifications σ and π, check if σ is a best response to π.
- Given a strategy specification profile $\langle\sigma, \pi\rangle$, check if it is a Nash equilibrium.
- Given a strategy specification π for player $\bar{\imath}$ and $F \in \mathcal{F}$, synthesize (if possible) a specification σ for i such that $\exists\sigma, \forall\pi : \mathcal{E}^i_F$ holds.
- Given a strategy specification π for $\bar{\imath}$, synthesize a specification σ such that σ is the best response to π.

The main theorem of the paper is the following assertion.

Theorem 4.1. Given a game $G = (\mathcal{G}, \mathcal{E})$ and a strategy specification π for player $\bar{\imath}$,

1. The verification problem of checking whether for a player i strategy specification σ and a binary evaluation automaton $\mathcal{E}^i_F$, if $\exists\sigma, \forall\pi : \mathcal{E}^i_F$ and $\forall\sigma, \forall\pi : \mathcal{E}^i_F$ holds in $\mathcal{G}$ is decidable.
2. For a binary evaluation automaton $\mathcal{E}^i_F$, it is possible to synthesize (when one exists), a deterministic advice automaton $\mathcal{A}_i$ such that $\mathcal{A}_i, \forall\pi : \mathcal{E}^i_F$ holds.
3. For a specification σ, checking if σ is the best response to π is decidable.

4. It is possible to synthesize a deterministic advice automaton $\mathcal{A}_i$ such that $\mathcal{A}_i$ is the best response to π.

Proof. Without loss of generality we assume $i = 1$, $\bar{\imath} = 2$ and σ, π to be the strategy specification for player 1 and 2 respectively.

For an advice automaton $\mathcal{A}_i = (Q_i, \delta_i, I_i, o_i)$, we define the restriction of $\mathcal{G}$ with respect to $\mathcal{A}_i$ to be $\mathcal{G} \upharpoonright A_i = (U, \longrightarrow_i, S_i)$ where $U = W \times Q_i$ and $S_i = \{s_0\} \times I_i$. In U, the nodes are partitioned in the obvious way. i.e. $u = (s, q) \in U^i$ iff $s \in W^i$. The transition relation $\longrightarrow_i$: $U \times \Sigma \to U$ is defined as,

- $(s, q) \xrightarrow{a}_i (s', q')$ iff $s \xrightarrow{a} s'$, $q' \in \delta_i(q, s, a)$ and ($s \in W^i$ implies $o_i(q, s) = a$).

For a node $u = (s, q) \in U$, let enabled$(u) = \{a | \exists (s', q') \in U$ with $(s, q) \xrightarrow{a} (s', q')\}$. Note that for all $u \in U^i$, $|\text{enabled}(u)| = 1$

$\mathcal{G} \upharpoonright \mathcal{A}_\pi$ is the arena restricted with π. i.e. all strategies of player 2 in $\mathcal{G} \upharpoonright \mathcal{A}_\pi$ conform to π. The game arena $\mathcal{G} \upharpoonright \mathcal{A}_\pi$ is no longer deterministic. However, for any player 2 node in $\mathcal{G} \upharpoonright \mathcal{A}_\pi$ there is exactly one action enabled (i.e. $\Big|\{a \in \Sigma \mid \exists\, u' \text{ with } u \xrightarrow{a} u'\}\Big| = 1$).

(1): To check if $\exists\sigma, \forall\pi : \mathcal{E}_F^i$ holds, we build a non-deterministic tree automaton $\mathcal{T}$ which runs on $\mathcal{G} \upharpoonright \mathcal{A}_\pi$. For a 1 node, it guesses an action "a" which conforms to σ and branches out on all a edges. For a 2 node, there is only one action enabled in $\mathcal{G} \upharpoonright \mathcal{A}_\pi$, call the action b. The automaton branches out on all b labelled edges. $\mathcal{T}$ runs $\mathcal{E}_F^1$ in parallel to verify that all plays thus constructed are winning for 1 with respect to $\mathcal{E}_F^1$. If $\mathcal{T}$ has an accepting run, then $\exists\sigma, \forall\pi : \mathcal{E}_F^i$ holds in $\mathcal{G}$. The details are as follows.

Consider $\exists\sigma, \forall\pi : \mathcal{E}_F^i$ in $\mathcal{G}$. According to the proof of Lemma 3.2, construct the advice automaton $\mathcal{A}_\sigma = (Q_\sigma, \delta_\sigma, I_\sigma, o_\sigma)$ and $\mathcal{A}_\pi = (Q_\pi, \delta_\pi, I_\pi, o_\pi)$. Let $\mathcal{E}_F^i = (\mathcal{M}, \{\lhd^i\}_{i \in \{1,2\}})$ with $\mathcal{M} = (R, \Delta, r_0)$.

Let $\mathcal{G}' = \mathcal{G} \upharpoonright \mathcal{A}_\pi = (U, \longrightarrow_\pi, S_\pi)$. Its easy to see that all player 2 strategies in $\mathcal{G}'$ is accepted by $\mathcal{A}_\pi$. Therefore we have $\exists\sigma, \forall\pi : \mathcal{E}_F^i$ holds in $\mathcal{G}$ iff there is a strategy μ accepted by $\mathcal{A}_\sigma$ such that for each strategy τ of 2 in $\mathcal{G} \upharpoonright \mathcal{A}_\pi$, the resulting path is winning for 1 with respect to $\mathcal{E}_F^i$. We give a nondeterministic top down tree automaton $\mathcal{T}$, which checks this property. Since S_π in general has more than one element, we add a new position called root and for all $u \in S_\pi$ add edges labelled with ε between root and u.

Formally, the tree automaton $\mathcal{T} = (Q, \delta, I)$ where $Q = (Q_\sigma \times R) \cup \{q_{\text{root}}\}$ and $I = q_{\text{root}}$. For $\mathcal{T}$ in a state q, reading node u, $\delta(q, u) = \langle (q_1, a, 1), (q_2, a, 2) \rangle$ means the automaton will branch out into two copies, on the first a successor it goes into state q_1 and the second it goes into state q_2. For a node $u = (s, q_\pi)$, let $\overrightarrow{u} \lceil a$ have k elements and let the successors be ordered in some way. The transition relation is defined as follows:

- If $u \in U^1$, then

$$\delta((q,r),u) = \{\langle((q',r'),a,1),\ldots,((q',r'),a,k)\rangle \mid \\ o_\sigma(q,s)=a, q' \in \delta_\sigma(q,s,a) \text{ and } r' = \Delta(r,s,a)\}$$

- If $u \in U^2$, then

$$\delta((q,r'),u) = \{\langle((q',r'),a,1),\ldots,((q',r'),a,k)\rangle \mid \\ q' \in \delta_\sigma(q,s,a) \text{ and } r' = \Delta(r,s,a)\}.$$

- If $u = \text{root}$, then

$$\delta(q_{\text{root}},u) = \{\langle((q_0,r_0),\varepsilon,1),\ldots,((q_0,r_0),\varepsilon,k)\rangle \mid q_0 \in I_\sigma\}.$$

To check if $\forall\sigma,\forall\pi : \mathcal{E}^i_F$ holds, it suffices to check if all plays in $(\mathcal{G}\upharpoonright\mathcal{A}_\pi)\upharpoonright\mathcal{A}_\sigma$ is winning for 1 with respect to $\mathcal{E}^1_F$. This can be done easily.

(2): We want a deterministic advice automaton $\mathcal{A}_1$ which ensures that for all strategies of 2 conforming to π the play is "winning" for player 1. We construct a tree automaton $\mathcal{T}$ which mimics the subset construction to synthesize $\mathcal{A}_1$. The states of $\mathcal{T}$ are the subsets of states of $\mathcal{A}_\pi$. At game positions of player 1, it guesses a move and for every player 2 game position, it branches out on all the action choices of $\mathcal{A}_\pi$ where for each move the resulting new state is the subset of states given by the nondeterministic transition relation of $\mathcal{A}_\pi$. $\mathcal{T}$ runs $\mathcal{E}^1_F$ in parallel and checks if all paths constitutes a valid play and that the play is winning for 1 with respect to $\mathcal{E}^1_F$. If there is an accepting run for $\mathcal{T}$, then constructing $\mathcal{A}_1$ is easy. The state space of $\mathcal{A}_1$ is the set of all subsets of the states of $\mathcal{A}_\pi$. The transition relation is derived from the usual subset construction performed by $\mathcal{T}$. The output function basically follows the accepting run of $\mathcal{T}$.

Let $\mathcal{A}_\pi = (Q_\pi, \delta_\pi, I_\pi, o_\pi)$ be the advice automaton corresponding to the strategy specification π. Let $\mathcal{B} = (Q_b, \delta_b, I_b, G)$. We extend the transition relation δ_π as follows. For a set $X \subseteq Q_\pi$, $\delta_\pi(X,s,a) = \cup_{q\in X}\delta_\pi(q,s,a)$. Let $\mathcal{T} = (Q,\delta,q_0)$ be the tree automaton where $Q = 2^{Q_\pi} \times R$ and the initial state $q_0 = I_\pi \times \{r_0\}$ is the set of all initial states of $\mathcal{A}_\pi$. For a tree automaton in state q reading node s of the tree, $\delta(q,s) = \langle(q_1,a),(q_2,b)\rangle$ means that the automaton will branch out into two copies , on the a labelled outgoing edge of s it goes into state q_1 and on the b labelled outgoing edge, it goes into state q_2.

For game position s, and an automaton state $q = (\{q^1_\pi,\ldots,q^k_\pi\},r)$, the transition relation is defined as follows:

- if $s \in W^1$: $\delta(q,s) =$

$$\{\langle((p,r'),a)\rangle | \exists s \xrightarrow{a} s' \text{ in } \mathcal{G}, p = \delta_\pi(q,s,a) \text{ and } r' = \Delta(r,s,a)\}.$$

- if $s \in W^2$: Let $\{a_1, \ldots, a_k\} = \{o_\pi(q_\pi^1), \ldots, o_\pi(q_\pi^k)\}$.

$$\delta(q,s) = \{\langle((p_1,r_1),a_1),\ldots,((p_k,r_k),a_k)\rangle \mid \\ p_i = \delta_\pi(q,s,a_i) \text{ and } r_i = \Delta(r,s,a_i)\}.$$

If $\mathcal{T}$ has a successful run on $\mathcal{G}$, then let T_π be the run tree with λ being the labelling function from game positions to Q. We build the advice automaton for 1 from this tree. The advice automaton $\mathcal{A}_1 = (q_1, \delta_1, q_1^0, o_1)$ where $Q_1 = 2_\pi^Q$, $q_1^0 = I_\pi$, $\delta_1(q_1, s, a) = q'$ if in T_π we have $s \xrightarrow{a} s'$ where $\lambda(s) = (q,r)$ and $\lambda(s') = (q', r')$. By definition of the transition function of $\mathcal{T}$, δ_1 is deterministic. The output function o_1, for each of the 1 nodes is dictated by the guess made by $\mathcal{T}$ on the successful run T_π.

(3): Given σ and π to check if σ is the best response to π, we use the tree automaton construction in (1) with a slight modification.

We enumerate the elements of 2^R in such a way that those higher in $\lhd^1$ appear earlier in the enumeration. For each F, we construct a tree automaton as in (1), the only difference being that the guesses made by $\mathcal{T}$ at player 1 game positions are not restricted by σ. $\mathcal{T}$ runs $\mathcal{E}_F^1$ in parallel to check if player 1 can ensure F for all choices of 2 which conform to π. Since the evaluation automaton is "complete", the play eventually settles down in one of $F' \in 2^R$. Therefore, as we try elements of 2^R in order, the tree automaton succeeds for some $\mathcal{E}_{F'}^1$. This gives us the "best" outcome which player 1 can guarantee. We then check if $\exists\sigma, \forall\pi : \mathcal{E}_{F'}^1$ holds in $\mathcal{G}$. If it does then $\mathcal{A}_\sigma$ is a best response to $\mathcal{A}_\pi$.

This also implies that we can check whether a strategy profile (presented as advice automata) constitutes a Nash equilibrium.

(4) is similar to (3). We enumerate 2^R and find the "best" outcome that can be achieved and using the synthesis procedure, synthesize an advice automaton for this outcome. Q.E.D.

5 A strategy logic

We now discuss how we may reason about structured strategies in a formal logic. Formulas of the logic (also referred to as *game formulas*) are built up using structured strategy specifications (as defined in section 3). Game formulas describe the game arena in a standard modal logic, and in addition specify the result of a player following a particular strategy at a game position, to choose a specific move a. Using these formulas one can specify how a strategy helps to eventually *win* (ensure) an outcome β.

Syntax

Let $P^i = \{p_0^i, p_1^i, \ldots\}$ be a countable set of proposition symbols where $\tau_i \in P_i$, for $i \in \{1,2\}$. Let $P = P^1 \cup P^2$. τ_1 and τ_2 are intended to specify, at a game position, which player's turn it is to move. Further, the logic is parametrized by the finite alphabet set $\Sigma = \{a_1, a_2, \ldots, a_m\}$ of players' moves and we only consider game arenas over Σ.

The syntax of the logic is given by:

$$\Pi := p \in P \mid \neg\alpha \mid \alpha_1 \vee \alpha_2 \mid \langle a\rangle\alpha \mid \diamondminus\alpha \mid (\sigma)_i : c \mid \sigma \leadsto_i \beta$$

where $c \in \Sigma$, $\sigma \in \mathrm{Strat}^i(P^i)$, $\beta \in \mathrm{Past}(P^i)$. The derived connectives $\wedge$, $\supset$ and $[a]\alpha$ are defined as usual. Let $\boxminus\alpha = \neg\diamondminus\neg\alpha$, $\langle X\rangle\alpha = \bigvee_{a\in\Sigma}\langle a\rangle\alpha$ and $[N]\alpha = \neg\langle X\rangle\neg\alpha$.

The formula $(\sigma)_i : c$ asserts, at any game position, that the strategy specification σ for player i suggests that the move c can be played at that position. The formula $\sigma \leadsto_i \beta$ says that from this position, following the strategy σ for player i ensures the outcome β. These two modalities constitute the main constructs of our logic.

Semantics

The models for the logic are extensive form game trees along with a valuation function. A model $M = (\mathrm{T}, V)$ where $\mathrm{T} = (S, \longrightarrow, s_0)$ is a game tree obtained by the unfolding of the arena $\mathcal{G}$, and $V : S \to 2^P$ is the valuation function.

Given a game tree T and a node s in it, let $\rho^s_{s_0} : s_0 \xRightarrow{a_1} s_1 \cdots \xRightarrow{a_m} s_m = s$ denote the unique path from s_0 to s. For the purpose of defining the logic it is convenient to define the notion of the set of moves enabled by a strategy specification at a node s (denote $\sigma(s)$). For a strategy specification $\sigma \in \mathrm{Strat}^i(P^i)$ and a node s we define $\sigma(s)$ as follows:

- $\mathsf{null}(s) = \Sigma$.
- $[\psi \mapsto a]^i(s) = \begin{cases} \{a\} & \text{if } s \in W^i \text{ and } \rho^s_{s_0}, m \models \psi \\ \Sigma & \text{otherwise.} \end{cases}$
- $(\sigma_1 + \sigma_2)(s) = \sigma_1(s) \cup \sigma_2(s)$.
- $(\sigma_1 \cdot \sigma_2)(s) = \sigma_1(s) \cap \sigma_2(s)$.
- $(\pi \Rightarrow \sigma)(s) = \begin{cases} \sigma(s) & \text{if } \forall j : 0 \le j < m,\ a_j \in \pi(s_j) \\ \Sigma & \text{otherwise.} \end{cases}$

We say that a path $\rho^{s'}_s : s = s_1 \xRightarrow{a_1} s_2 \cdots \xRightarrow{a_{m-1}} s_m = s'$ in T $\mathsf{conforms}$ to σ if $\forall j : 1 \le j < m$, $a_j \in \sigma(s_j)$. When the path constitutes a proper play, i.e. when $s = s_0$, we say that the play conforms to σ.

The following proposition is easy to see.

Proposition 5.1. Given a strategy μ for player i along with a specification σ, $\mu \models_i \sigma$ (as defined in section 3) iff for all player i nodes $s \in \mu$ we have $\mathrm{out}(s) \in \sigma(s)$.

For a game tree T, a node s let T_s denote the tree which consists of the unique path $\rho_{s_0}^{s}$ and the subtree rooted at s. For a strategy specification $\sigma \in \mathrm{Strat}^i(P^i)$, we define $\mathrm{T}_s \upharpoonright \sigma = (S_\sigma, \Longrightarrow_\sigma, s_0)$ to be the least subtree of T_s which contains the unique path from s_0 to s and satisfies the following property.

- For every s' in S_σ such that $s \Longrightarrow_\sigma^* s'$,
  - s' is an i node: $s' \xRightarrow{a} s''$ and $a \in \sigma(s') \Leftrightarrow s' \xRightarrow{a}_\sigma s''$.
  - s' is an $\bar{\imath}$ node: $s' \xRightarrow{a} s'' \Leftrightarrow s' \xRightarrow{a}_\sigma s''$.

The truth of a formula $\alpha \in \Pi$ in a model M and position s (denoted $M, s \models \alpha$) is defined by induction on the structure of α, as usual. Let $\rho_{s_0}^{s}$ be $s_0 \xRightarrow{a_0} s_1 \cdots \xRightarrow{a_{m-1}} s_m = s$.

- $M, s \models p$ iff $p \in V(s)$.
- $M, s \models \neg\alpha$ iff $M, s \not\models \alpha$.
- $M, s \models \alpha_1 \vee \alpha_2$ iff $M, s \models \alpha_1$ or $M, s \models \alpha_2$.
- $M, s \models \langle a\rangle\alpha$ iff there exists $s' \in W$ such that $s \xrightarrow{a} s'$ and $M, s' \models \alpha$.
- $M, s \models \diamondminus\alpha$ iff there exists $j : 0 \leq j \leq m$ such that $M, s_j \models \alpha$.
- $M, s \models (\sigma)_i : c$ iff $c \in \sigma(s)$.
- $M, s \models \sigma \leadsto_i \beta$ iff for all s' such that $s \Longrightarrow_\sigma^* s'$ in $\mathrm{T}_s \upharpoonright \sigma$, we have $M, s' \models \beta \wedge (\tau_i \supset \mathrm{enabled}_\sigma)$.

where $\mathrm{enabled}_\sigma \equiv \bigvee_{a \in \Sigma}(\langle a\rangle\mathtt{True} \wedge (\sigma)_i : a)$.

Figure 1 illustrates the semantics of $\sigma \leadsto_1 \beta$. It says, for an 1 node β is ensured by playing according to σ; for a 2 node, all actions should ensure β.

The notions of satisfiability and validity can be defined in the standard way. A formula α is satisfiable iff there exists a model M such that $M, s_0 \models \alpha$. A formula α is said to be valid iff for all models M, we have $M, s_0 \models \alpha$.

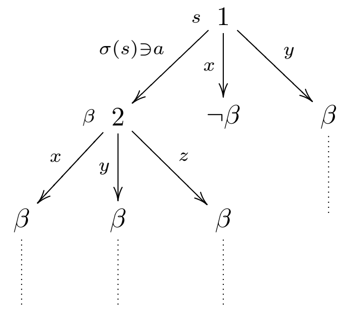


FIGURE 1.

Truth checking

The truth checking problem is given a model $M = (\mathrm{T}, V)$ and a formula α_0, determine whether $M, s_0 \models \alpha_0$. The following theorem shows the decidability of the truth checking problem.

Theorem 5.2. Given a model $M = (\mathrm{T}, V)$ and a formula α_0, we can construct a nondeterministic Büchi tree automaton $\mathcal{T}_{\alpha_0}$ such that $M, s_0 \models \alpha_0$ iff $\mathcal{T}_{\alpha_0}$ has an accepting run on M.

Proof. Let $\{\sigma_1, \dots, \sigma_m\}$ be the strategy specification formulas appearing in α_0 and $\mathcal{A}_{\sigma_1}, \dots \mathcal{A}_{\sigma_m}$ be the advice automata corresponding to the specifications. The tree automaton keeps track of the atoms (locally consistent sets of subformulas) of α_0 and the states of each of the advice automata. At any game position, it guesses a new atom which is consistent with the game position and a state for each of the advice automaton from its transition relation. For the subformula $(\sigma)_i : a$ in the atom, it only needs to check if a is the action dictated by the output function of the advice automaton for σ. However, $\neg(\sigma \leadsto_i \beta)$ is a requirement which says that there exists a game position where $\mathrm{enabled}_\sigma$ does not hold or β is false. We keep track of such formulas in a "requirement set" U. When the tree automaton branches, it guesses, for each branch, which requirements will be satisfied on that branch. The Büchi acceptance condition is simply all the states where the "requirement set" U is empty.

We shall find some abbreviations useful:

- $\mathrm{inv}_i^\sigma(a, \beta) = (\tau_i \wedge (\sigma)_i : a) \supset [a](\sigma \leadsto_i \beta)$ denotes the fact that after an "a" move by player i which conforms to σ, $\sigma \leadsto_i \beta$ continues to hold.
- $\mathrm{inv}_{\bar{\imath}}^\sigma(\beta) = \tau_{\bar{\imath}} \supset [N](\sigma \leadsto_i \beta)$ says that after any move of $\bar{\imath}$, $\sigma \leadsto_i \beta$ continues to hold.

- $\text{enabled}_\sigma = \bigvee_{a \in \Sigma}(\langle a \rangle \texttt{True} \wedge (\sigma)_i : a)$.

For a formula α, let $\text{SF}(\alpha)$ denote the subformula closure of α. In addition to the usual downward closure we also require that $\sigma \rightsquigarrow_i \beta \in \text{SF}(\alpha)$ implies $\text{enabled}_i, \text{inv}_i^\sigma(a, \beta), \text{inv}_{\bar{\imath}}^\sigma(\beta), \beta \in \text{SF}(\alpha)$. Call $C \subseteq \text{SF}(\alpha)$ an atom if it is propositionally consistent and complete, in addition we require the following to hold.

- $\sigma \rightsquigarrow_i \beta \in C \Rightarrow \text{enabled}_\sigma, \text{inv}_i^\sigma(a,\beta), \text{inv}_{\bar{\imath}}^\sigma(\beta) \in C$.
- $\neg(\sigma \rightsquigarrow_i \beta) \in C \Rightarrow (\neg\text{enabled}_\sigma \text{ or } \neg\beta) \in C$ or $(\langle X \rangle \neg(\sigma \rightsquigarrow_i \beta)) \in C$.

Let $\mathcal{AT}_\alpha$ denote the set of atoms. Let $C_0 = \{C \in \mathcal{AT}_\alpha |$ there does not exist any $\diamondminus\gamma \in C\}$. For $C, D \in \mathcal{AT}_\alpha$, define $C \xrightarrow{a} D$ iff for all $\diamondminus\gamma \in \text{SF}(\alpha)$, the following conditions hold.

- $\gamma \in C \Rightarrow \diamondminus\gamma \in D$.
- $\diamondminus\gamma \in C \Rightarrow \gamma \in C$ or $\diamondminus\gamma \in C$.
- $[a]\gamma \in C \Rightarrow \gamma \in D$.

Let $\{\sigma_1, \ldots, \sigma_m\}$ be the strategy specification formulas appearing in α_0 and let $\mathcal{A}_{\sigma_1}, \ldots \mathcal{A}_{\sigma_m}$ be the advice automata corresponding to the specifications. The tree automata $\mathcal{T} = (Q, \delta, I, F)$ where $Q \subseteq (\mathcal{AT}_{\alpha_0} \cup \textsf{reject}) \times (2^{\text{SF}(\alpha_0)})^3 \times Q_{\sigma_1} \times \ldots \times Q_{\sigma_m}$ such that $(C, U, Z, Y, q_1, \ldots, q_m) \in Q$ iff $(\sigma)_i : a, \tau_i \in C \Rightarrow o_\sigma(q_\sigma) = a$. The sets Z and Y are used to keep track of the $\langle a \rangle \alpha$ formulas and ensure that the edge relation is consistent with these formulas. The set of initial states $I = \{(C, U, Z, Y, q_1^0, \ldots, q_m^0) | C \in C_0, V(s_0) = C \cap P_{\alpha_0}, U = \varnothing, Z = \varnothing$ and $q_i^0 \in I_{\sigma_i}\}$, $Y = \{\langle a \rangle \alpha | a \in \Sigma$ and $\langle a \rangle \alpha \in C\}$.

For a node s, let $s_1, \ldots, s_k$ be its successors in $\mathcal{G}$ with $s \xrightarrow{a_j} s_j$ for $1 \leq j \leq k$. For a state $q = (C, U, Z, Y, q_1, \ldots, q_m)$ at s, the automaton guesses a partition of $U = U_1 \cup \ldots \cup U_k$ and a partition $Y = Z_1 \cup \ldots \cup Z_k$. The transition relation is then defined as:

$$\langle(((C_1, U_1', Z_1, Y_1, q_1^1, \ldots q_m^1), a_1), \ldots, ((C_k, U_k', Z_1, Y_1, q_1^k, \ldots, q_m^k), a_k)\rangle \\ \in \delta((C, U, q_1, \ldots, q_m), s)$$

iff

- $C_j = \textsf{reject}$ if there exists $\langle a \rangle \alpha \in Z_j$ such that $\alpha \notin C_j$ or $a_j \neq a$
- For $1 \leq j \leq k$, $C \xrightarrow{a_j} C_j$ and $V(s_j) = C_j \cap P_{\alpha_0}$.
- For $1 \leq j \leq k$, $1 \leq r \leq m$, $q_r^j \in \delta_r(q_r, s, a_j)$.

- $U'_j = \begin{cases} \{\sigma \rightsquigarrow_i \beta \in U_j \mid \beta, \text{enabled}_\sigma \in C_j\} & \text{if } U \neq \varnothing \\ \{\sigma \rightsquigarrow_i \beta \in C_j \mid \beta, \text{enabled}_\sigma \in C_j\} & \text{if } U = \varnothing \end{cases}$

- $Y_j = \{\langle a \rangle \alpha \mid \langle a \rangle \alpha \in C_j\}$

Once the automaton reaches the reject state then it remains in that state for all transitions. The Büchi acceptance condition is, $F = \{q = (C, U, Z, Y, q_1, \ldots, q_m) \in Q \mid U = \varnothing \text{ and } C \in \mathcal{AT}_{\alpha_0}\}$. Q.E.D.

Complexity of truth checking

For the given formula α_0, let $|\alpha_0| = n$. The states of the tree automaton are the atoms of α_0 and the states of each of the advice automaton. Since the number of strategy specifications occurring in α_0 is bounded by the size of α_0, the size of the tree automaton $|\mathcal{T}| = \mathcal{O}(n \cdot 2^n)$. Let $\mathcal{T}_\mathcal{G}$ denote the tree automaton accepting $\mathcal{G}$. We want to check for emptiness of $\mathcal{T} \cap \mathcal{T}_\mathcal{G}$. Since $\mathcal{T}$ is a Büchi tree automaton this gives us a total time complexity of $\mathcal{O}(2^n)$.

References

[1] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In W. P. de Roever, H. Langmaack, and A. Pnueli, editors, *Compositionality: The Significant Difference: International Symposium, COMPOS'97, Bad Malente, Germany, September 1997. Revised Lectures*, volume 1536 of *Lecture Notes in Computer Science*, pages 23–60. Springer, 1997.

[2] G. Bonanno. The logic of rational play in games of perfect information. *Econom. and Philos.*, 7:37–65, 1991.

[3] D. Gale. The game of Hex and the Brouwer fixed-point theorem. *Amer. Math. Monthly*, 86(10):818–827, 1979.

[4] V. Goranko. Coalition games and alternating temporal logics. *Proceedings of 8th conference on Theoretical Aspects of Rationality and Knowledge (TARK VIII)*, pages 259–272, 2001.

[5] E. Grädel, W. Thomas, and Th. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[6] P. Harrenstein, W. van der Hoek, J.-J. C. Meyer, and C. Witteveen. A modal characterization of nash equilibrium. *Fundam. Inform.*, 57(2-4):281–321, 2003.

[7] J. F. Nash, Jr. Equilibrium points in n-person games. *Proc. Nat. Acad. Sci. U. S. A.*, 36:48–49, 1950.

[8] M. J. Osborne and A. Rubinstein. *A course in game theory.* MIT Press, Cambridge, MA, 1994.

[9] R. Parikh. The logic of games and its applications. In *Topics in the theory of computation (Borgholm, 1983)*, volume 102 of *North-Holland Math. Stud.*, pages 111–139, Amsterdam, 1985. North-Holland.

[10] M. Pauly. *Logic for Social Software.* PhD thesis, University of Amsterdam, October 2001.

[11] R. Ramanujam and S. Simon. Axioms for composite strategies. *Proceedings of Logic and Foundations of Games and Decision Theory*, pages 189–198, 2006.

[12] J. van Benthem. Games in dynamic epistemic logic. *Bull. Econom. Res.*, 53(4):219–248, 2001.

[13] W. van der Hoek, W. Jamroga, and M. Wooldridge. A logic for strategic reasoning. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *AAMAS*, pages 157–164. ACM, 2005.

[14] E. Zermelo. Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels,. In *Proceedings of the Fifth Congress of Mathematicians*, volume 2, pages 501–504, Cambridge, 1913. Cambridge University Press.

Counting in trees

Helmut Seidl[1]
Thomas Schwentick[2]
Anca Muscholl[3] *

[1] Institut für Informatik, I2
Technische Universität München
Boltzmannstraße 3
85748 Garching, Germany
`seidl@in.tum.de`

[2] Lehrstuhl Informatik I
Universität Dortmund
44221 Dortmund, Germany
`Thomas.Schwentick@udo.edu`

[3] Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux 1
351, cours de la Libération
33405 Talence cedex, France
`anca@labri.fr`

Abstract

We consider automata and logics that allow to reason about numerical properties of unranked trees, expressed as Presburger constraints. We characterize non-deterministic automata by Presburger Monadic Second-Order logic, and deterministic automata by Presburger Fixpoint logic. We show how our results can be used in order to obtain efficient querying algorithms on XML trees.

1 Introduction

Tree automata and logics for finite trees have been considered since the seminal work of Thatcher and Wright [38] in the late sixties, with emphasis on *ranked* trees. More recently, research on semi-structured data and XML in particular, raised new questions about *unranked* trees, i.e., trees where the number of children of a node is not fixed *a priori*, [8, 22]. Trees in XML are unranked, labeled, and may occur in two versions, ordered or unordered, depending on whether the sequence of children of a node is ordered or not.

In XML schema description languages like DTDs and XML Schema, the possible sequences of types of children elements of a node are described

* We thank the referee for his/her careful reading and the various remarks and suggestions that helped improving the paper.

by regular expressions. Thus, most of the existing theoretical work on XML query languages has concentrated on regular tree languages. These languages can be described by tree automata on unranked ordered trees (also known as *hedge automata*) [25, 27] and a variety of other formalisms [15, 24, 26]. In these formalisms the interaction between the children of a node and the node itself are usually expressed in terms of a regular language. Other work extended these formalisms to let them formulate (at least unary) queries. The resulting query facilities usually have the expressive power of Monadic Second-Order logic (MSO).

The regular framework is sufficient in many cases. But often one is interested in expressing conditions on the frequency of occurrences of elements in the children sequence. Consider as an example a document which contains music files shared by some peer-to-peer system as Napster, Gnutella etc. as described in Figure 1.[1]

For instance, we would like to query for users who prefer jazz over pop. Such a query can be expressed by asking for nodes labeled with "music" that have more children labeled "jazz" than "pop". Querying for users who are extreme jazz fans can be expressed by requiring that the majority of the children of a node labeled by "music" is labeled by "jazz".

One way of formulating such queries, is to extend the MSO logic by formulas of Presburger arithmetics constraining the children of a node (*Presburger constraints* for short). In this new *Presburger MSO* logic (PMSO) the first query can be expressed as:

$$x \in \mathsf{Lab}_{\mathtt{music}} \wedge x/\varphi_1 ,$$

where φ_1 is the formula

$$\varphi_1 \equiv \#\mathsf{Lab}_{\mathtt{jazz}} \geq \#\mathsf{Lab}_{\mathtt{pop}} .$$

Here, $\#\mathsf{Lab}_{\mathtt{jazz}}$ and $\#\mathsf{Lab}_{\mathtt{pop}}$ denote the numbers of children labeled with `jazz` and `pop`, respectively. For the second query we replace φ_1 by φ_2, where φ_2 is the formula:

$$\varphi_2 \equiv \#\mathsf{Lab}_{\mathtt{jazz}} \geq \#\mathsf{Lab}_{\mathtt{pop}} + \#\mathsf{Lab}_{\mathtt{french}} + \#\mathsf{Lab}_{\mathtt{classic}} .$$

As an operational counterpart of the extended logic we study bottom-up tree automata that are enhanced by *Presburger constraints.* Transitions from the children of a node to the node itself may depend on the frequencies of states at the children via a Presburger arithmetic condition, i.e., a formula involving addition.

[1] It should be noted that in a realistic setting the type of music would likely be represented by an attribute and not by a separate tag for each type. But, of course, for the purpose of query processing we can interpret a tag with attribute *jazz* as a tag *jazz*.

```
<doc>
 <user>
  <name> ... </name>
   ...
  <music>
   <jazz>
    <album> Always let me go </album>
    <artist> Keith Jarrett </artist>
    <year> 2002 </year>
    <time> 3310 </time>
    <price> 42 </price>
   </jazz>
   <french>
    <tit> Aux enfants de la chance </tit>
    <artist> Serge Gainsbourg </artist>
    <album> Serge Gainsbourg, vol. 3 </album>
    <time> 247 </time>
    <price> 16 </price>
   </french>
   <classic>
    <tit> The Seven Gates of Jerusalem </tit>
    <comp> Krzystof Penderecki </comp>
    <recorded> 1999 </recorded>
   <time> 3510 </time>
    <price> 43 </price>
   </classic>
   <jazz>
    <album> Kind of Blue </album>
    <artist> Miles Davis </artist>
    <year> 1997 </year>
    <time> 3325 </time>
    <price> 29 </price>
   </jazz>

  </music>
  <video>
      ...
  </video>
  <images>
      ...
  </images>
 </user>
</doc>
```

FIGURE 1. An example document containing information about music files downloaded by users.

We start our investigation by considering automata that *only* use Presburger constraints, i.e., automata that disregard the order of the children of a node and only use cardinalities of states. Technically speaking, we study in this part automata on unordered trees. It turns out that these automata are very well-behaved. They define a class of tree languages with very regular properties like various closure properties and equivalence with PMSO logic. Further, these automata allow for effective static analysis. Emptiness and universality are decidable, and from any non-deterministic automaton an equivalent deterministic automaton can be constructed. Last but not least, they allow to define a class of (unary) queries the evaluation of which has linear time data complexity.

Next, we study automata that are allowed to combine Presburger constraints with the common regular language constraints (*Presburger tree automata, PTA*). It turns out that they have less desirable properties. Although emptiness of PTA can still be decided, universality (whether an automaton accepts *all* trees) becomes undecidable. As we show that the non-deterministic PTA can be characterized by existential PMSO logic, we can conclude that PMSO logic is undecidable. Nevertheless, the combined complexity of these automata is NP-complete, whereas the data complexity is polynomial time.

Often however, and in particular in our example, some parts of a document can be considered as textual representations of information records. This means that inside certain elements, the ordering is not significant. We therefore investigate automata on *mixed* document trees, i.e., in which element tags either identify their content as ordered or as unordered. We further assume that, as in our example, numerical constraints are only applicable to such unordered element contents. Under these assumptions, we get the same kind of nice behavior as in the totally unordered case, mentioned above.

An alternative for the querying formalism enhanced by Presburger constraints is to replace the MSO logic by fixpoint logic. This Presburger fixpoint logic turns out to be decidable (EXPTIME-complete), and its combined complexity is polynomial time. Moreover, this logic has the same expressive power as deterministic PTA.

This paper is an extended version of [35, 36].

Overview. In Section 2 we define some basic Presburger logic notions. Section 3 studies unordered Presburger tree automata and logic. Section 4 studies basic algorithmic properties of Boolean combinations of regular expressions and Presburger conditions. In Section 5, ordered Presburger tree automata and logic are considered. Section 6 takes a quick look at the case where some unordered parts of a tree allow for Presburger constraints and the others for regular expressions. Section 7 studies Presburger fixpoint logic and its relation with Presburger tree automata. Finally, Section 8 shows how our framework can be used to express unary queries.

Related work. Unordered document trees are closely related to the generalization of feature trees considered by Niehren and Podelski in [28] where they study the (classical) notion of recognizability and give a characterization of this notion by means of feature automata. No counting constraints are considered. A detailed study of automata over unranked trees has been initiated by Brüggeman-Klein, Murata and Wood [3].

Query languages for unordered trees have been proposed by Cardelli and Ghelli [5, 4, 6, 7] (and their co-workers). Their approach is based on first-order logic and fixpoint operators. An extension to numerical constraints has been proposed by Dal Zilio et al. [10]. Kupferman, Sattler and Vardi study a μ-calculus with *graded* modalities where one can express, e.g., that a node has at least n successors satisfying a given property [19]. The numbers n there, however, are hard-coded into the formula. Orderings on the successors is not considered. Klaedtke and Ruess consider automata on the unlabeled infinite binary tree, which have an accepting condition depending on a global Presburger constraint [18].

Our notion of tree automata with combined Presburger and regular constraints has been introduced independently by Dal Zilio and Lugiez in [9]. In the latter paper, the authors also propose a modal logic for XML documents, called *Sheaves logic.* This logic allows to reason about numerical properties of the contents of elements but still lacks recursion, i.e., fixpoint operators. On the automata side they obtain comparable results concerning closure properties, membership tests and decidability of emptiness. Although no precise characterization is given, the Sheaves logic is strictly less powerful than the automata model. Recently, Demri and Lugiez proposed the extended modal logic EXML, which uses regular and Presburger constraints on the sequence of children (still without recursion) [11]. The logic EXML is shown to contain the Sheaves logic and to have an EXPSPACE satisfiability problem.

2 Preliminaries on Presburger Logic

Presburger logic is first-order logic with addition and the ordering relation over $\mathbb{N}$. It can express various decision questions such as solvability of systems of linear equations, integer programming, or verification questions. The decidability of Presburger logic was established by Presburger [33] by quantifier elimination. A doubly exponential non-deterministic lower bound was shown in [13]. Later, the precise complexity was shown to be LinATIME $2^{2^{O(n)}}$, namely doubly exponential alternating time with a linear number of alternations, [1]. A long line of research was devoted to the analysis of various decision procedures for this logic, based on quantifier elimination and automata. For instance, from a formula in prenex normal form one can construct automata of triply exponential size [17].

For complexity reasons it is quite common to consider either *quantifier-free* or *existential* Presburger formulas, since their satisfiability is in NP. Both use linear terms with integer coefficients, i.e., built according to the syntax (with x a variable which is interpreted over $\mathbb{N}$):

$$t ::= 0 \mid 1 \mid \pm x \mid t_1 + t_2 \,.$$

Quantifier-free Presburger formulas are defined as the closure of atomic formulas of kind $t = 0$ and $t \equiv 0 \pmod{d}$ (with t a term and $d \in \mathbb{N}$ a constant) under the Boolean connectives. *Existential Presburger formulas* are defined as the closure of atomic formulas of kind $t = 0$ under the positive connectives $\vee, \wedge$ and existential quantification.

It is well known that each Presburger formula can be transformed into an equivalent quantifier-free formula [33]. In one more step, such a formula can be transformed into an existential formula in *normal form*, that is, into a formula of the form $\exists x_1, \ldots, x_k \bigvee_{i=1}^{m} \varphi_i$, where each disjunct φ_i is a conjunction of equations $t = 0$ with t a linear term (with integer coefficients):

Proposition 2.1. Every quantifier-free Presburger formula φ has an equivalent formula in existential normal form. This formula has at most exponentially many disjuncts, each of at most linear size (in $|\varphi|$).

Proof. Let φ be a quantifier-free Presburger formula. First we bring it into disjunctive normal form (DNF). Then we replace atomic and negated atomic formulas by equations, if necessary by introducing new existentially quantified variables. More precisely,

- $t < c$ can be replaced by $\exists y_1(t + 1 + y_1 = c)$,
- $t \neq c$ by $\exists y_2(t + y_2 + 1 = c \vee t - y_2 - 1 = c)$,
- $t \equiv c \pmod{d}$ by $\exists y_3(t - dy_3 = c \vee t + dy_3 = c)$, and
- $t \not\equiv c \pmod{d}$ by

$$\exists y_4, y_5(t - dy_4 - y_5 = 0 \vee t + dy_4 - y_5 = 0) \wedge (0 \leq y_5 < c \vee c < y_5 < d)\,.$$

The resulting formula needs not to be in DNF yet, but it is free of negations and can be easily transformed into existential normal form. Note first that the DNF is of exponential size, but each disjunct contains at most $|\varphi|$ atoms. After replacing the atoms by equations, each conjunction is transformed into DNF. The size of each resulting disjunct is still linear, and the overall number of disjuncts remains exponential. Q.E.D.

Remark 2.2. Satisfiability of existential Presburger formulas is easily seen to be NP-complete. The upper bound is obtained by assuming w.l.o.g. that such a formula is in prenex form $\exists x_1, \ldots, x_k \, \psi$, with ψ a positive Boolean combination of equations $t = 0$, with t a linear term. It suffices to guess then a disjunct of the DNF of ψ, and test in NP whether a conjunction of such equations is satisfiable.

Given a formula φ and an *assignment* σ mapping the variables of φ to numbers, we write $\sigma \models \varphi$ if φ holds for σ (in the obvious sense) and call σ a *solution* of φ. It is well known that the set of solutions of any given Presburger formula is a *semi-linear set* [14]. A semi-linear set is a finite union of *linear sets*, i.e., sets of the form $\{\bar{c} + \sum_{i=1}^{m} x_i \bar{p}_i \mid x_i \in \mathbb{N}\}$, where $\bar{c}$ and the $\bar{p}_i$ are vectors from $\mathbb{N}^k$ for a given k.

The *Parikh image* of a word $w \in \Sigma^*$ is the assignment $\sigma \in \mathbb{N}^\Sigma$ with $\sigma(a)$ being the number of occurrences of the letter a in w, for each $a \in \Sigma$. Accordingly, the Parikh image of a set $L \subseteq \Sigma^*$ is the set of Parikh images of $w \in L$.

Given the alphabet Σ, $\mathcal{T}(\Sigma)$ stands for the set of *ordered, unranked trees over* Σ. A tree $t \in \mathcal{T}(\Sigma)$ with root label a and subtrees $t_1, \ldots, t_n$ will be denoted as $t = a\langle t_1, \ldots, t_n \rangle$.

3 Unordered Presburger Tree Automata

In this section we start with tree automata and logics that are *unordered*, i.e., they consider only the vertical parent-child order, but not the order between siblings. Technically speaking, we work on unordered trees, as considered for instance in [2, 10, 4, 5].

Given a finite set Q, we shall consider a canonical set Y_Q of variables which are associated with the elements in Q. So, we define:

$$Y_Q = \{\#q \mid q \in Q\}.$$

An *unordered Presburger tree automaton* (u-PTA for short) is given by a tuple $\mathcal{A} = (Q, \Sigma, \delta, F)$ where:

- Q is a finite set of states,
- $F \subseteq Q$ is the subset of accepting states,
- Σ is the finite alphabet of tree labels, and
- δ maps pairs (q, a) of states and labels to quantifier-free Presburger formulas with variables only from the set Y_Q.

Informally, u-PTA are bottom-up tree automata, with transitions controlled by quantifier-free Presburger formulas. A formula $\varphi = \delta(q, a)$ represents the *pre-condition* on the children of a node labeled by a for the transition into state q, where the value of the variable $\#p$ represents the number of children that are in state p. Formally, we introduce a satisfaction relation $t \models_{\mathcal{A}} q$ between trees $t \in \mathcal{T}(\Sigma)$ and states q which is defined as follows. Assume that $t = a\langle t_1, \ldots, t_n\rangle$, where a is the the label of the root, and $t_1, \ldots, t_n$ are the subtrees of the root, and let $\delta(q, a) = \varphi$. Then $t \models_{\mathcal{A}} q$ if $\{1, \ldots, n\}$ can be partitioned into $|Q|$ subsets I_p of cardinalities n_p ($p \in Q$), such that:

- $t_i \models_{\mathcal{A}} p$ for all $i \in I_p$,
- $\{\#p \mapsto n_p \mid p \in Q\} \models \varphi$.

The language $\mathcal{L}(\mathcal{A})$ of trees which are accepted by $\mathcal{A}$ is

$$\mathcal{L}(\mathcal{A}) = \{t \in \mathcal{T}(\Sigma) \mid \exists\, f \in F : t \models_{\mathcal{A}} f\}\,.$$

As an example, consider the language of trees with labels in $\{a, b\}$, such that the internal nodes are all labeled by a and have at most as many subtrees with a b-leaf as ones without. A u-PTA for this language has two states, say q_0 and q_1, where state q_0 means that there is no b-leaf in the subtree, and state q_1 the converse. The transition relation is defined by $\delta(q_0, a) = (\#q_1 = 0)$, $\delta(q_0, b) = \mathit{false}$, $\delta(q_1, a) = (\#q_0 \geq \#q_1 > 0)$ and $\delta(q_1, b) = \text{leaf}$. Here, we use the Presburger constraint leaf $= (\sum_{i=0,1} \#q_i = 0)$, which is satisfied precisely at leaf nodes.

Note that u-PTA are defined as non-deterministic automata. A u-PTA $\mathcal{A} = (Q, \Sigma, \delta, F)$ is called *deterministic* if for every $a \in \Sigma$ and every tuple $(n_p)_{p \in Q} \in \mathbb{N}^Q$, there is at most one state $q \in Q$ such that

$$\{\#p \mapsto n_p \mid p \in Q\} \models \delta(q, a)\,.$$

Remark 3.1. It is not too hard to verify whether a given u-PTA is deterministic. The precise complexity is NP-complete, since it amounts to check the satisfiability of quantifier-free Presburger formulas. The lower bound can be obtained by an obvious reduction from *Integer Linear Programming (ILP).*

3.1 Closure and decidability results

The results of this section show that u-PTA enjoy several desirable properties, such as determinization and reasonable complexity.

Theorem 3.2. The non-emptiness problem for u-PTA is NP-complete.

Proof. Consider a u-PTA $\mathcal{A} = (Q, \Sigma, \delta, F)$. Let us call a state $q \in Q$ *reachable* iff there is a tree t with $t \models_{\mathcal{A}} q$. The algorithm guesses some final state $q \in F$, and checks that q is reachable. To this end, the algorithm guesses some $k \leq |Q|$ and a sequence $q_1, \ldots, q_k$ of states with $q_k = q$, and checks that, for each $1 \leq j \leq k$, the following formula is satisfiable:

$$\Big(\bigwedge_{p \in Q \setminus \{q_i \mid i < j\}} \#p = 0 \Big) \wedge \Big(\bigvee_{a \in \Sigma} \delta(q_j, a) \Big) .$$

Since each check can be done non-deterministically in polynomial time, the overall complexity is NP. Moreover, NP-hardness is again an immediate consequence of ILP, thus we conclude that non-emptiness of u-PTA is NP-complete. Q.E.D.

We show next that u-PTA are effectively closed under the Boolean operations (union, intersection and complement). In particular, we give a determinization construction for u-PTA.

Theorem 3.3. u-PTA are effectively closed under the Boolean operations and under renaming[2].

Proof. Closure under union and renaming is immediate. For intersection, assume that we are given automata $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, F_i)$, $i = 1, 2$. W.l.o.g. we assume that $Q_1 \cap Q_2 = \varnothing$. We proceed analogously to the standard construction of the product automaton for ordinary automata. Thus, we define the automaton $\mathcal{A} = (Q, \Sigma, \delta, F)$ as follows. We set $Q = Q_1 \times Q_2$, $F = F_1 \times F_2$ and define $\delta(\langle q_1, q_2 \rangle, a)$ by the formula below, where $\widetilde{\delta_1}$ and $\widetilde{\delta_2}$, resp., are obtained from δ_1, δ_2, resp., by replacing all variables $\#p$ by x_p ($p \in Q_1 \cup Q_2$):

$$\begin{aligned} &\exists_{p_1 \in Q_1} x_{p_1} . \exists_{p_2 \in Q_2} x_{p_2} . \quad \widetilde{\delta_1}(q_1, a) \wedge \widetilde{\delta_2}(q_2, a) \quad \wedge \\ &\qquad \Big(\bigwedge_{p_1 \in Q_1} \sum_{p_2 \in Q_2} \#\langle p_1, p_2 \rangle = x_{p_1} \Big) \wedge \Big(\bigwedge_{p_2 \in Q_2} \sum_{p_1 \in Q_1} \#\langle p_1, p_2 \rangle = x_{p_2} \Big) . \end{aligned}$$

In addition, we use above the notation $\exists_{i \in I} x_i$, for some index set I, to denote the existential quantification over all variables x_i ($i \in I$). This is done for convenience only, since the above formula can be rewritten directly as a quantifier-free Presburger formula. It is easy to see that $t \models_{\mathcal{A}} \langle q_1, q_2 \rangle$ iff $t \models_{\mathcal{A}_1} q_1$ and $t \models_{\mathcal{A}_2} q_2$. Thus, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$, which completes the proof.

For closure under complement it suffices to know that u-PTA can be determinized, which is shown in the proposition below. Q.E.D.

Proposition 3.4. For every u-PTA $\mathcal{A}$, a deterministic u-PTA $\mathcal{A}'$ can be constructed such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.

[2] A renaming is a letter-to-letter morphism.

Proof. The proof idea is similar to the power set construction for ordinary finite automata. Let $\mathcal{A} = (Q, \Sigma, \delta, F)$ and $\mathcal{A}' = (Q', \Sigma, \delta', F')$, where $Q' = 2^Q$ and $F' = \{B \subseteq Q \mid F \cap B \neq \varnothing\}$. For each $B \subseteq Q$, $\delta'(B, a)$ is a formula with free variables from $Y_{Q'}$. It is given by:

$$\Big(\bigwedge_{q\in B} \psi_{q,a}\Big) \wedge \Big(\bigwedge_{q\in Q\setminus B} \neg\psi_{q,a}\Big).$$

Here, the formula $\psi_{q,a}$ should be true iff q is a potential successor state. In order to specify $\psi_{q,a}$, we refer to the auxiliary variables x_p ($p \in Q$), and also to auxiliary variables $x_{\langle B,p\rangle}$ ($B \subseteq Q$, $p \in B$). The variable $x_{\langle B,p\rangle}$ is meant to count all those children resulting in the state set B in $\mathcal{A}'$, for which we choose state $p \in B$ w.r.t. the $\mathcal{A}$-run. Using these auxiliary variables, $\psi_{q,a}$ is defined below, with $\widetilde{\delta}(q, a)$ as the formula $\delta(q, a)$ where each variable $\#p$ is replaced by x_p:

$$\underset{p\in Q}{\exists}\, x_p\,.\ \underset{p\in B\subseteq Q}{\exists}\, x_{\langle B,p\rangle}\,.\ \ \widetilde{\delta}(q,a) \wedge \\ \Big(\bigwedge_{B\subseteq Q}\sum_{p\in B} x_{\langle B,p\rangle} = \#B\Big) \wedge \Big(\bigwedge_{p\in Q}\sum_{p\in B\subseteq Q} x_{\langle B,p\rangle} = x_p\Big).$$

The transitions of the subset automaton can be transformed into quantifier-free formulas by quantifier elimination. Q.E.D.

As a corollary of Proposition 3.4, we also obtain:

Corollary 3.5. The universality problem for u-PTA is decidable.

The complexity upper bound for the universality problem that follows from Proposition 3.4 is 2-NEXPTIME. Note first that the transition formulas $\delta'(B, a)$ can be written as $\exists\bar{x}\forall\bar{y}\,\psi(\bar{x}, \bar{y})$, with ψ quantifier-free. Using quantifier elimination we can first make $\forall\bar{y}\,\psi(\bar{x}, \bar{y})$ quantifier-free, then rewrite $\delta'(B, a)$ as an existential Presburger formula. The first step is exponential in the size of the universal quantifier block (see, e.g., [17]), hence we obtain an existential formula of doubly exponential size, for which satisfiability can be checked in 2-NEXPTIME.

Proposition 3.6. The combined complexity of u-PTA Model Checking is NP-complete. If the u-PTA is deterministic, the complexity is polynomial time. The data complexity is linear in the size of the input tree.

Proof. Assume we are given a u-PTA $\mathcal{A}$ and a tree t. We guess for each node of t a state of $\mathcal{A}$, and then check that the run is accepting: For each node we compute the Parikh image of the children states (note the entries have values $\leq |t|$). Then we need to check that the Presburger formulas are locally satisfied, for each node. Thus, the evaluation of all formulas at a node requires time $O(|\mathcal{A}| \cdot |t|)$ (using numbers in unary representation).

NP-hardness follows by a reduction from 0-1 ILP, where we ask whether a system S of linear equations $A\bar{x} = \bar{b}$ has a 0-1 solution, that is, one with $\bar{x} \in \{0,1\}^m$. Given A and $\bar{b}$, we define a u-PTA with state space $\{p_1, \ldots, p_m, p, f\}$, final state f and transition relation $\delta(p,a) = \delta(p_i,a) =$ leaf for all i, and $\delta(f,c) = (\varphi \wedge \#f = 0)$, where φ is the conjunction of all equations in S (with x_i replaced by $\#p_i$) and of $0 \leq \#p_i \leq 1$ for all i. Clearly, the tree t of depth 1 with root labeled c and m leaves labeled a satisfies $t \models_{\mathcal{A}} f$ iff S has a 0-1 solution.

If $\mathcal{A}$ is deterministic, then the tree can be evaluated in a bottom-up traversal in time $O(|t| \cdot |\mathcal{A}|)$. The data complexity follows immediately, using Proposition 3.4. Q.E.D.

3.2 Unordered Presburger logic

Unordered Presburger MSO logic (u-PMSO for short) is defined by extending monadic second-order logic (MSO) with Presburger predicates over the *children* of a node. As for u-PTA, the logic does not provide an ordering relation on siblings. A u-PMSO formula φ is given by the following grammar:

$$\begin{array}{lcl} \varphi & ::= & y \in \mathsf{Lab}_a \mid y \in Y \mid \mathrm{Child}(y,y') \mid y/\psi \mid \\ & & \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \exists\, y.\, \varphi \mid \exists\, Y.\, \varphi \\ \psi & ::= & t_1 = t_2 \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid \exists x.\, \psi_1 \\ t & ::= & 0 \mid 1 \mid \pm x \mid t_1 + t_2 \mid \#Y \end{array}$$

where $a \in \Sigma$, y, y' are first-order variables, Y is a second-order variable, and x is from a designated set of Presburger variables. The predicates $y \in \mathsf{Lab}_a$ and $\mathrm{Child}(y,y')$ are interpreted as usual, i.e. y is labeled by a, resp. y' is a child of y. The formula ψ in y/ψ is a quantifier-free Presburger formula. The interpretation of y/ψ is that the children of y satisfy the Presburger constraint ψ. A term $\#Y$ inside ψ is interpreted as the number of those children which are contained in Y. As usual, we write $t \models \varphi$, if t satisfies the (closed) u-PMSO formula φ.

Remark 3.7. We also allow derived predicates such as equality between variables such as $y_1 = y_2$, or $Y_1 = Y_2$, or equations $Y = \{y_1\}$. Note that the child predicate $\mathrm{Child}(y,y')$ is redundant, since it can be expressed as:

$$\exists Y.\ (Y = \{y'\} \wedge y/(\#Y = 1))\ .$$

A tree language $L \subseteq \mathcal{T}(\Sigma)$ is u-PMSO-definable if there is a closed formula φ such that $L = \{t \mid t \models \varphi\}$.

Theorem 3.8 below states that u-PMSO-definable languages are precisely characterized by unordered Presburger tree automata. The proof is analogous to the corresponding result for ordinary tree automata (over ranked trees) [38], and uses in particular Theorem 3.3.

Theorem 3.8. A set of unranked trees is accepted by some u-PTA if and only if it is definable in u-PMSO.

4 Regular expressions and Presburger formulas

The automata considered in the next section, as well as the associated logics, use pre-conditions on the children of a node in form of Boolean combinations of regular expressions and Presburger formulas. A basic question is then the satisfiability of such conditions. Since the Parikh image of a regular language (and even context-free language, [32]) is semilinear, deciding emptiness can be reduced to computing the Parikh image of the regular expressions involved.

In [36] we showed that even for an NFA, an existential Presburger formula which describes the Parikh image of the corresponding language can be computed in linear time. Later, this result was extended to context-free grammars in [39] (see also [12] for a related approach).

Theorem 4.1 (Verma-Seidl-Schwentick, [39])**.** Given a context-free grammar G, an existential Presburger formula for the Parikh image of $\mathcal{L}(G)$ can be computed in linear time.

A *Presburger regular expression* over Σ is a Boolean combination of regular expressions over Σ and quantifier-free Presburger formulas with variables only from the canonical set $Y_\Sigma = \{\#a \mid a \in \Sigma\}$.

Given a string $w \in \Sigma^*$ and a Presburger regular expression φ we define in the obvious way whether w matches φ (denoted as $w \models \varphi$). For example, if $\varphi = a(a+b)^* \wedge (\#a = \#b)$, then $w \models \varphi$ iff w contains only a's and b's, begins with an a and contains as many a's as b's. A Presburger regular expression φ is satisfiable if there exists some w with $w \models \varphi$. Before we show how to decide satisfiability for such expressions, we need the following property:

Proposition 4.2. Let $\mathcal{A}$ be a (non-deterministic) finite word automaton with n states and input alphabet of size k. Then the Parikh image of $\mathcal{L}(\mathcal{A})$ is a union of linear sets $\{\bar{c} + \sum_{i=1}^{m} x_i \cdot \bar{p}_i \mid x_i \in \mathbb{N}\}$ where each component of each vector $\bar{c}, \bar{p}_j \in \mathbb{N}^k$ is at most n.

In particular, if the size of the alphabet is k, then the number m of occurring vectors is at most $(n+1)^k$.

Proof. The proof is based on the following simple observation: any (accepting) path of $\mathcal{A}$ can be decomposed successively into loops of length at most

n, and one (accepting) path of length at most n. Thus, we define each set of vectors $\bar{c}, \bar{p}_j \in \mathbb{N}^k$ by associating $\bar{c}$ with an accepting path λ_0 of length at most n and each $\bar{p}_i$ with a loop λ_i of length at most n, in such a way that the λ_j, $0 \le j \le m$, can be combined to a (accepting) path in $\mathcal{A}$. Specifically, it suffices to fix for each j the set of states that occur in λ_j in such a way that $\cup_{j=0}^{m} \lambda_j$ is connected. Q.E.D.

Proposition 4.3. The satisfiability problem for Presburger regular expressions is PSPACE-complete.

Proof. The lower bound is immediate, since it is already PSPACE-hard to decide whether a given set of regular expressions has a non-empty intersection or whether the complement of a single regular expression is non-empty [37].

For the upper bound let φ be a Presburger regular expression of size n. First of all, we can assume w.l.o.g. that negations are used only as linear or modular inequations, or in form of negated regular expressions. The given expression φ is satisfiable iff some of the disjuncts in the DNF of φ is so. We can guess such a disjunct ψ in linear time. The formula ψ is a conjunction of regular expressions, negated regular expressions, linear (in)equations $t = 0$, $t \neq 0$ and modular (in)equations $t \equiv 0 \pmod{d}$, $t \not\equiv 0 \pmod{d}$.

We first show that ψ is satisfiable iff there exists some word w of exponential length with $w \models \psi$. Since the regular expressions in ψ all occur in φ, the sum of their sizes is at most n. The minimal automaton of each such (possibly negated) regular expression e (resp., $\neg e$) is of size $2^{|e|}$, hence the minimal automaton $\mathcal{A}_\psi$ of the intersection of all positive and negated regular expressions is of size 2^n .

By Proposition 4.2, the Parikh image of $\mathcal{L}(\mathcal{A}_\psi)$ is a union of linear sets $\{\bar{c} + \sum_{i=1}^{h} x_i \bar{p}_i \mid x_i \in \mathbb{N}\}$, where $h = O(2^{n \cdot |\Sigma|}) = O(2^{n^2})$ (as $|\Sigma| \le n$) and the entries of the vectors $\bar{c}, \bar{p}_i$ are at most $O(2^n)$.

Now, a word $w \in \Sigma^*$ satisfies ψ iff its Parikh image is in one of these linear sets and additionally fulfills the remaining (Presburger) constraints. This can be expressed by adding, for each $a \in \Sigma$, the following equation:

$$\#a = \bar{c}(a) + \sum_{i=1}^{h} x_i \cdot \bar{p}_i(a)\,.$$

Let m be the number of Presburger constraints in ψ. By Proposition 2.1, the conjunction of these constraints is equivalent to a formula in existential normal form, with disjuncts of size $O(m)$. Thus, one has to check whether the Parikh image of w satisfies a system of $M = O(m) + |\Sigma| \le O(n)$ equations with at most $N = |\Sigma| + O(m + 2^{n^2}) = O(n + 2^{n^2})$ variables, and coefficients of values bounded by $k = 2^n$. By a result of Papadimitriou [31] such a system has a solution with numbers bounded by

$$N \cdot (M \cdot k + 1)^{2M+4} = (O(n + 2^{n^2})) \cdot (O(n) \cdot 2^n + 1)^{O(n)} = 2^{O(n^2)} .$$

This shows that if some $w \models \psi$ exists, then there is some with length $2^{O(n^2)}$.

It remains to describe how to check the existence of w as above. We simply guess w symbol by symbol. For each regular expression e or $\neg e$ in ψ, we compute the set of states that can be reached in the non-deterministic automaton $\mathcal{A}_e$ for e when reading w. Further, for each $a \in \Sigma$ we count how often a occurs in w. All this can be done in polynomial space without storing w. A counter keeps track of the length of w. In the end, it can be checked whether the Parikh image of w satisfies all Presburger constraints. Q.E.D.

As Presburger regular expressions are closed under negation we immediately conclude that also universality is PSPACE-complete.

5 Presburger tree automata

In many applications, e.g., where documents are automatically generated from databases as textual representations of querying results, the element ordering on the children does not matter. In other applications, though, which are more related to classical document processing, this ordering is crucial. In this section, we extend our framework to automata and logics that take the sibling order into account. Naturally, we use then Presburger regular expressions as pre-conditions on children sequences.

We define a *Presburger tree automaton* for unranked trees (PTA for short) as a tuple $\mathcal{A} = (Q, \Sigma, \delta, F)$ where:

- Q is a finite set of states;
- $F \subseteq Q$ is the subset of accepting states;
- δ maps pairs (q, a) of states and labels from Σ to Presburger regular expressions over Q.

Accordingly, we introduce an extended satisfaction relation between trees t and states q by defining for $t = a\langle t_1 \dots t_l \rangle$ and $\delta(q, a) = \varphi$, $t \models_{\mathcal{A}} q$ iff there are states $p_1, \dots, p_l \in Q$ such that $t_j \models_{\mathcal{A}} p_j$ for all j and $p_1 \cdots p_l \models \varphi$. The language $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{T}(\Sigma)$ which is *accepted* by the automaton $\mathcal{A}$ is given by:

$$\mathcal{L}(\mathcal{A}) = \{t \in \mathcal{T}(\Sigma) \mid \exists f \in F : t \models_{\mathcal{A}} f\} .$$

A PTA $\mathcal{A}$ is called *deterministic* if for all $a \in \Sigma$ and all $w \in Q^*$, we have $w \models \delta(q, a)$ for at most one $q \in Q$.

Using Proposition 4.3, we obtain with a similar proof as for Theorem 3.2:

Theorem 5.1. The emptiness problem for PTA is PSPACE-complete.

Next we turn to the complexity of such automata.

Theorem 5.2. 1. The combined complexity of PTA Model Checking is NP-complete. If the PTA $\mathcal{A}$ is deterministic, it is $O(n \cdot |\mathcal{A}|)$, where n is the size of the input tree.

2. The data complexity of PTA is polynomial, $O(n^{k+1})$. The degree k of the polynomial is the number of states of the PTA.

Proof. Let $\mathcal{A}$ be a PTA with state set Q and t a tree. A non-deterministic algorithm guesses a run of $\mathcal{A}$ on t and checks the consistency at each node. Each consistency check amounts (1) to testing membership of a string $w \in Q^*$ of size $|t|$ for at most $|\mathcal{A}|$ many regular languages, represented by regular expressions (possibly negated), and (2) to evaluating at most $|\mathcal{A}|$ (quantifier-free) Presburger formulas on its Parikh image. All these checks can be done deterministically, in time $O(|t| \cdot |\mathcal{A}|)$. If $\mathcal{A}$ is deterministic, we perform the computation bottom-up deterministically. The NP lower bound for non-deterministic PTA follows already from the u-PTA case.

Towards (2.), suppose now that the PTA $\mathcal{A}$ is fixed and let Q be its set of states. We perform a bottom-up traversal of the input tree t, computing for each subtree t' the set of states $R = \{p \mid t' \models_{\mathcal{A}} p\} \subseteq Q$. Assume that $t' = a\langle t_1, \ldots, t_m \rangle$ and $R_i = \{p \mid t_i \models p\}$ have been already computed. Moreover, we can suppose that the Presburger regular expressions used in $\mathcal{A}$ are disjunctions of conjuncts $e_i \wedge \pi_i$ where for each e_i a deterministic automaton $\mathcal{B}_i$ is given, and each π_i is a Presburger formula. Then we may check for each $e_i \wedge \pi_i$ separately whether it is verified by $t_1, \ldots, t_m$.

To this end, let us now consider one conjunct $e \wedge \pi$, where the language of e is described by the finite automaton $\mathcal{B}$ with set of states P. Let the sets $V(i, s)$, $0 \leq i \leq m$, $s \in P$, contain all assignments $v : Y_Q \to \{0, \ldots, i\}$ verifying the following condition: there exists a sequence of states $\alpha = r_1 \cdots r_i$ with $r_j \in R_j$ for $j = 1, \ldots, i$ and with Parikh image v, such that state s can be reached from an initial state of $\mathcal{B}$ by reading α. Finally, let V be the union of all sets $V(m, f)$ where f is a final state of $\mathcal{B}$. Once V is computed, it simply remains to check whether $v \models \pi$ for some $v \in V$. Thus, assuming that the automaton $\mathcal{A}$ is of constant size, we spend $O((m+1)^{|Q|})$ time on the computation of the set of all successor states at the root of t'. Hence we can conclude that the overall runtime on a tree of size n is $O(n^{|Q|+1})$. Q.E.D.

PTA do not share the pleasant properties with u-PTA. In particular, it is a direct consequence of the following result that there is no determinization algorithm for PTA.

Theorem 5.3. The universality problem for PTA is undecidable.

Proof. The proof is a reduction from the accepting problem for 2-counter Minsky machines [23]. Given such an automaton $\mathcal{A}$ with state set Q we construct a PTA $\mathcal{B}$ such that $\mathcal{A}$ does not accept the empty input if and only if $\mathcal{B}$ accepts all trees over the alphabet $Q \cup \{\#, \$, a, b\}$. In the construction we shall concentrate on trees of depth 1 with frontier (i.e., leaf labels read from left to right) from $Qa^*b^*(\#Qa^*b^*)^*$. The root is labeled by \$. It is easy to construct a tree automaton (without Presburger constraints) which accepts all trees that are *not* of this special form. The union of this automaton with the automaton $\mathcal{B}'$ to be constructed in the remainder of the proof will be the automaton $\mathcal{B}$ we are looking for.

The PTA $\mathcal{B}'$ checks whether the frontier does *not* encode an accepting computation of the counter automaton $\mathcal{A}$. Here, a configuration of $\mathcal{A}$ with state q and counter contents n_1 and n_2, respectively, is encoded by the string $qa^{n_1}b^{n_2}$ and configurations are separated by $\#$. The automaton $\mathcal{B}'$ checks whether one of the following cases arises:

- the frontier does not start with $q_0\#$, where q_0 is the initial state of $\mathcal{A}$, or
- the frontier does not end with a string of the form $\#qa^*b^*$, where q is an accepting state of $\mathcal{A}$, or
- there are two successive configurations that are not consistent with the transition function of $\mathcal{A}$.

We only describe how the latter case can be checked, as the first two cases are straightforward. The idea is that $\mathcal{B}'$ simply marks two consecutive configurations. A regular constraint can check that two such configurations are correctly marked, whereas a Presburger constraint ensures that the two configurations are indeed inconsistent with the transition function of $\mathcal{A}$.

Formally, the state set of $\mathcal{B}'$ equals $Q \cup \{\#, a, a', b, b', ?\}$. On each leaf the automaton can enter state ?. Further, it can enter state $\#$ on all leaves labeled $\#$, and state q on all leaves with label $q \in Q$. For leaves with label a (b, resp.) it can also enter state a or a' (b or b', resp.) The automaton $\mathcal{B}'$ enters an accepting state at the root if both conditions below hold:

- the states on the frontier form the sequence $?^*\#qa^*b^*\#q'a'^*b'^*\#?^*$ with $q, q' \in Q$, and
- the numbers of occurrences of a, b, a', b' are not consistent with respect to q, q' and the transition function of $\mathcal{A}$.

The first condition above is simply a regular constraint. The second one can be expressed by a conjunction of Presburger constraints, over all possible transitions of $\mathcal{A}$ leading from state q to state q'. For instance, for a transition that increases the first counter and leaves the second one unchanged, the Presburger constraint requires that either the number of a' is not equal to the number of a plus 1, or the numbers of b and b' are not equal. Q.E.D.

5.1 Presburger MSO logic

Unordered Presburger MSO logic as defined in Section 3.2 is readily extended to take into account the sibling order, by adding the atomic predicate $\text{Next}(y, y')$, with the meaning that y' is a *right sibling* of y. We denote this logic as PMSO. We now characterize Presburger tree automata by means of existential PMSO logic.

Theorem 5.4. A set of unranked trees is accepted by a PTA if and only if it can be described by a PMSO formula of the form $\exists X_1 \ldots \exists X_k . \varphi$ where φ contains no second-order quantifier.

Proof. Let $\mathcal{A}$ be a PTA with state set Q and transition relation δ. Without loss of generality we can assume that all Presburger regular expressions used in δ are disjunctions of expressions $e_i \wedge \pi_i$, where e_i is a regular expression over Q, and π_i is a quantifier-free Presburger formula. Furthermore, let, for each i, a finite automaton $\mathcal{B}_i$ for $\mathcal{L}(e_i)$ be given. From Büchi's Theorem it follows that each automaton $\mathcal{B}_i$ is equivalent to an existential MSO formula $\psi_i = \exists Y_1 \ldots \exists Y_l . \varphi_i$. Hence, we can construct a formula $\psi = \exists X_1 \cdots \exists X_k . \varphi$ in which some of the variables X_i are used to encode the states that $\mathcal{A}$ assumes and the remaining variables are those of the formulas ψ_i. The first-order part φ of ψ describes the consistency of the states between nodes of the input tree and their children, and uses the formulas φ_i.

For the converse we show first that every PMSO formula ψ containing no second-order quantifier can be evaluated by a *deterministic* PTA. The result is then immediate as a non-deterministic automaton can guess, for each node, those sets of $X_1, \ldots, X_k$ in which the node is contained. The proof proceeds by induction on the structure of ψ. The only case which is not entirely straightforward is the case of a formula $\psi = \exists x . \varphi(x)$. Let, by induction, $\mathcal{A}$ be an automaton over the alphabet $\Sigma \cup (\Sigma \times \{x\})$ for $\varphi(x)$. I.e., $\mathcal{A}$ accepts all trees t which have exactly one node v with a symbol (a, x) from $\Sigma \times \{x\}$ such that φ holds on t, if x is bound to v and the label of v is replaced by a.

Let Q be the set of states of $\mathcal{A}$. We construct a deterministic PTA $\mathcal{A}'$ for ψ as follows. The state set of $\mathcal{A}'$ is $Q \times 2^Q$. The intuitive meaning of a state $\langle q, X\rangle$ at a node v is the following. First, if x does not occur in the subtree rooted at v, then $\mathcal{A}$ assumes state q at v. Second, X is the set of states $\mathcal{A}$ can take if for one node of the subtree at v its label

a is replaced by (a, x). We explain how the transitions of $\mathcal{A}'$ are defined. The mapping $\delta'(\langle q, X\rangle, a)$ is described by a Presburger regular expression $e_{q,a} \wedge e_{X,a}$, where $e_{q,a}$ is obtained from $\delta(q, a)$ by replacing each occurrence of a state $r \in Q$ in a regular expression by $\bigcup_{S\subseteq Q}\langle r, S\rangle$ and each occurrence of $\#r$ in a Presburger formula by $\sum_{S\subseteq Q} \#\langle r, S\rangle$. The Presburger regular expression $e_{X,a}$ is of the form $\bigwedge_{p\in X}(e^1_{p,a} \vee e^2_{p,a}) \wedge \bigwedge_{p\notin X} \neg(e^1_{p,a} \vee e^2_{p,a})$. Here, $e^1_{p,a}$ expresses that $\mathcal{A}$ takes state p at v if the label of v is (a, x). Likewise, $e^2_{p,a}$ expresses that $\mathcal{A}$ takes state p at v (labeled by a) if the label b of some node below v is replaced by (b, x). The expression $e^1_{p,a}$ is obtained from $\delta(p, (a, x))$ in an analogous fashion as $e_{q,a}$ was obtained from $\delta(q, a)$.

It remains to describe the construction of $e^2_{p,a}$. The expression $e^2_{p,a}$ is obtained as a disjunction $\bigvee_{r\in Q}\bigvee_{r'\in S\subseteq Q} \delta(p, a)_{r,r',S}$. Here, for each choice of $S \subseteq Q$, $r \in Q$ and $r' \in S$, the Presburger regular expression $\delta(p, a)_{r,r',S}$ is satisfied by a sequence $\langle q_1, S_1\rangle \cdots \langle q_m, S_m\rangle$, $q_i \in Q$, $S_i \subseteq Q$, iff there is some $i \le m$ with $q_i = r$, $S_i = S$ and $\delta(p, a)$ is satisfied by $q_1 \cdots q_{i-1}r'q_{i+1} \cdots q_m$.

The expression $\delta(p, a)_{r,r',S}$ is defined by replacing in $\delta(p, a)$ each regular expression e by $e_{r,r',S}$, and each Presburger formula π by $\pi_{r,r',S}$. We get $\pi_{r,r',S}$ as the conjunction of $\#\langle r, S\rangle > 0$ and the formula which is obtained from π by replacing $\#q$, for each $q \in Q$ with

- $\displaystyle\sum_{S'\subseteq Q} \#\langle q, S'\rangle$, if $q \notin \{r, r'\}$ or $q = r = r'$,
- $\displaystyle(\sum_{S'\subseteq Q} \#\langle q, S'\rangle) - 1$, if $q = r$ and $r \neq r'$, and
- $\displaystyle(\sum_{S'\subseteq Q} \#\langle q, S'\rangle) + 1$, if $q = r'$ and $r \neq r'$.

The language L of a regular expression $e_{r,r',S}$ is given as:

$$L = \{\langle q_1, S_1\rangle \cdots \langle q_m, S_m\rangle \mid \exists\, i\ :\ \langle q_i, S_i\rangle = \langle r, S\rangle \wedge \\ q_1 \cdots q_{i-1}r'q_{i+1} \cdots q_n \in \mathcal{L}(e)\}\,.$$

Q.E.D.

Theorem 5.4 shows that existential PMSO logic is decidable. On the other hand we immediately obtain from Theorem 5.3:

Corollary 5.5. Satisfiability of PMSO formulas is undecidable.

6 Mixed automata

In the previous section we have seen that in general we cannot expect decidability for all PMSO. Instead, we can restrict ourselves to automata and logics that work in a *mixed* mode, either pure regular or pure Presburger, depending on the tag. Formally, we work on mixed trees, where the label of a node tells whether the ordering of its children matters or not. Recall from the introduction that this restriction naturally reflects a division of documents into parts which are made up from data records whose orderings are irrelevant and formatting parts where the ordering is significant. This classification is formalized by partitioning the finite alphabet Σ into subsets $\Sigma = \Sigma_0 + \Sigma_1$ where Σ_0 and Σ_1 consist of all labels of nodes with unordered and ordered children, respectively. Mixed trees in our sense correspond to terms with one associative symbol (for accumulating the ordered contents) and one associative and commutative symbol (for accumulating multi-sets). Languages of such trees, e.g., have been studied by Lugiez [20, 21] and Ohsaki [29, 30]. Note, however, that our formalism is slightly more specific as we rule out sequences of trees where unordered sections occur dispersed between ordered ones. Instead, the significance of order is already determined by the label of the parent node.

Mixed Presburger tree automata now subsume the ability of unordered Presburger automata to check Presburger formulas, as well as the ability of hedge automata to check containment in a regular set. Formally, $\delta(q, a)$ is a quantifier-free Presburger formula if $a \in \Sigma_0$, respectively a regular expression if $a \in \Sigma_1$. We call such an automaton a *mixed* PTA. Similarly to Theorem 3.2, we obtain:

Corollary 6.1. The emptiness problem for mixed PTA is NP-complete.

It turns out that the family of languages accepted by mixed PTA enjoys the same good closure properties as u-PTA. The proof of the theorem below follows the lines of Proposition 3.4 and is omitted:

Theorem 6.2. Mixed PTA are effectively closed under the Boolean operations. In particular, for every mixed PTA an equivalent deterministic mixed PTA can be constructed.

As for unordered and general PTA, respectively, we succeed to give a logical characterization of our automata model also in the mixed case. For that we use mixed PMSO logic, in which Presburger (regular, resp.) constraints can be applied only to the children of a node labeled with some element from Σ_0 (Σ_1, resp.). We therefore speak here of *mixed* PMSO-definable languages and queries. More formally, in a mixed PMSO-formula an atom $\text{Next}(y, y')$ is allowed in a subformula φ occurring in a context $\text{Child}(x, y) \wedge y \in \mathsf{Lab}_a \wedge \varphi$, where $a \in \Sigma_1$. Likewise a formula y/ψ is allowed in a subformula φ occurring in a context $y \in \mathsf{Lab}_a \wedge \varphi$, where $a \in \Sigma_0$. Mixed PMSO-definable queries are what we have considered in the introduction, by considering, e.g., that the label `music` belongs to Σ_0. We obtain:

Theorem 6.3. A set of unranked trees is accepted by some mixed PTA iff it is mixed PMSO-definable.

We conclude that satisfiability of mixed PMSO-logic is decidable.

7 Presburger fixpoint logic

As an alternative to monadic second-order logic, we consider in this section the extension of fixpoint logics with regular and Presburger constraints on children of nodes. Our fixpoint formulas φ are thus constructed according to the following grammar:

$$\begin{array}{lcl} \varphi & ::= & \top \mid x \mid \mu\, x.\, \varphi \\ & & \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \\ & & \mid a\langle F \rangle \mid *\langle F \rangle \\ F & ::= & e \mid \pi\,. \end{array}$$

Here, "$*$" denotes an arbitrary node label, and F denotes a generic pre-condition on the children of a node. Such a pre-condition is either a regular expression e over letters φ, possibly negated, or a quantifier-free Presburger formula π with free variables $\#\varphi$, denoting the number of children satisfying φ (with φ a fixpoint formula).

In the following, we assume throughout that φ is a formula where all bound variables are distinct. Let Φ denote the set of all subformulas of φ. We consider assertions $t : \psi$, with $t \in \mathcal{T}(\Sigma), \psi \in \Phi$. We write $\vdash t : \psi$ either if $\psi \equiv \top$ (every tree satisfies $\top$) or if the assertion $t : \psi$ can be derived from valid assertions by means of the following rules:

$$\frac{t : \psi \qquad \mu\, x.\psi \in \Phi}{t : x} \qquad\qquad \frac{t : \psi \qquad \mu\, x.\psi \in \Phi}{t : \mu\, x.\psi}$$

$$\frac{t : \psi_1 \qquad t : \psi_2}{t : \psi_1 \wedge \psi_2} \qquad\qquad \frac{t : \psi_i}{t : \psi_1 \vee \psi_2}$$

$$\frac{u : F}{a\langle u \rangle : a\langle F \rangle} \qquad\qquad \frac{u : F}{a\langle u \rangle : *\langle F \rangle}$$

Thus, besides assertions $t : \psi$, $t \in \mathcal{T}(\Sigma)$, we additionally need auxiliary assertions $u : F$ where u is a sequence of trees and F is either a regular expression or a Presburger formula. A sequence $u = t_1, \ldots, t_k$ satisfies a regular pre-condition e iff there are formulas $\psi_1, \ldots, \psi_k$ such that $t_i : \psi_i$ and the sequence of formulas $\psi_1 \cdots \psi_k$ is contained in the regular language $\mathcal{L}(e)$ described by e. In case of a Presburger formula π, we first collect for every formula ψ occurring in π the number n_ψ of children t_i satisfying ψ. Then u satisfies π iff the resulting assignment $\sigma = \{\#\psi \mapsto n_\psi \mid \psi \in \Phi\}$

satisfies $\sigma \models \pi$. Thus we have the rules:

$$\frac{t_i : \psi_i \quad (i = 1, \ldots, k) \qquad \psi_1 \cdots \psi_k \in \mathcal{L}(e)}{t_1, \ldots, t_k : e}$$
$$\frac{\sigma \models \pi \quad \text{where} \quad \sigma(\#\psi) = |\{i \mid t_i : \psi\}|}{t_1, \ldots, t_k : \pi}$$

Note that according to this rule for Presburger formulas, the same tree t_i may be counted several times, once for every ψ such that $t_i : \psi$.

A *proof* of an assertion $t : \psi$ consists of all rule applications to derive this assertion. In particular this means for $t = a\langle t_1, \ldots, t_k\rangle$ and $\psi = a\langle\pi\rangle$, π a Presburger formula, that a proof of $t : \psi$ contains for every $i = 1, \ldots, k$, and every ψ' occurring in π a subproof of $\vdash t_i : \psi'$—whenever it exists. Moreover, we silently assume that a proof always has tree-like structure. Thus, we may have several copies of a subproof for distinct occurrences of the same subtree within t.

Finally, the language denoted by the formula φ is given by:

$$\mathcal{L}(\varphi) = \{t \in \mathcal{T}(\Sigma) \mid \vdash t : \varphi\}\,.$$

In particular, $\mathcal{L}(\top) = \mathcal{T}(\Sigma)$ and $\mathcal{L}(\mu\, x.\, x) = \varnothing$. Using the convenient abbreviation "$\_$" for $\top^*$, i.e., an arbitrary sequence of trees, we may write $\mu\, x.\, (a\langle\_\rangle \vee *\langle\_\ x\ \_\rangle)$ for the set of all trees with at least one inner node labeled a. Note that our fixpoint expressions do not provide an explicit notion of negation. However, we always can construct an equivalent expression with *guarded* fixpoints (see, e.g., [34]). The free variable x occurs only guarded inside the formula φ if x occurs as a free variable only within the scope of elements a or $*$. The variable x, for example, occurs only guarded inside the formula $a\langle\_\ x\ \_\rangle \vee y$ while y does not. It turns out that guarded fixpoints are *unique*. More precisely, if x occurs only guarded in φ, then $\mu x.\, \varphi$ is semantically equivalent to $\nu x.\varphi$. Once greatest fixpoints are available, complementation is easy since then we can push negations inward. For example, we have: $t : \neg(\mu x.\, \varphi(x))$ iff $t : \nu x.\, \neg\varphi(\neg x)$.

In the subsequent proofs we shall use the following notion. For a subset $B \subseteq \Phi$ of subformulas of φ, define the *closure* $\mathsf{cl}(B)$ as the least superset B' of B such that:

- $\top \in B'$;
- If $\varphi_1 \in B'$ and $\varphi_2 \in B'$ then also $\varphi_1 \wedge \varphi_2 \in B'$, whenever $\varphi_1 \wedge \varphi_2 \in \Phi$;
- If $\varphi_1 \in B'$ or $\varphi_2 \in B'$ then also $\varphi_1 \vee \varphi_2 \in B'$, whenever $\varphi_1 \vee \varphi_2 \in \Phi$;
- If $\varphi' \in B'$ then $\mu\, x.\varphi' \in B'$ and $x \in B'$, whenever $\mu\, x.\varphi' \in \Phi$.

Intuitively, the closure of a set B of subformulas contains precisely the subformulas which are implied by the formulas in B through the proof rules for fixpoint formulas. In particular, consider a given fixpoint formula, a tree t and let B be the set of all subformulas ψ of type $a\langle F\rangle$ and $*\langle F\rangle$ with $t:\psi$. Then, $\mathsf{cl}(B)$ is the set of *all* subformulas ψ with $t:\psi$.

Theorem 7.1. A set of trees is accepted by some deterministic PTA if and only if it satisfies some Presburger fixpoint formula.

Proof. Let φ be a Presburger fixpoint formula. We assume for simplicity that all regular expressions in φ are unnegated. We construct a PTA $\mathcal{A}$ as follows. Let Ψ denote the set of all subformulas of φ of the form $a\langle F\rangle$ or $*\langle F\rangle$. The set Q of states of $\mathcal{A}$ is given as the set of all subsets $B\subseteq\Psi$. The set T of accepting states consists of all subsets B such that $\varphi\in\mathsf{cl}(B)$, i.e., whose closure contains the initial formula φ.

Given a state $B\in Q$ and $a\in\Sigma$, we determine the pre-condition $\delta(B,a)$ as

$$\delta(B,a) \quad=\quad \bigwedge\nolimits_{\psi\in B}\Delta(\psi,a)\ \wedge\ \bigwedge\nolimits_{\psi\in\Psi\setminus B}\neg\Delta(\psi,a)$$

where:

$$\begin{array}{rcll}\Delta(a\langle F\rangle,a) &=& \bar{F} & \\ \Delta(*\langle F\rangle,a) &=& \bar{F} & \\ \Delta(b\langle F\rangle,a) &=& \mathsf{false} & \quad\text{if } a\neq b\end{array}$$

where $\bar{F}$ is constructed as follows. For a regular expression e, we obtain $\bar{e}$ from e by substituting $B_1+\cdots+B_m$ for every occurrence of a formula ψ if $\{B_1,\ldots,B_m\}$ is the set of all states B such that $\psi\in\mathsf{cl}(B)$. For a Presburger formula π, let $\bar{\pi}$ be obtained from π by substituting $\sum_{\psi'\in\mathsf{cl}(B)}\#B$ for every occurrence of the free variable $\#\psi'$. By construction, the resulting automaton is deterministic. We show for trees $t,t_1,\ldots,t_k$:

(1) $t\models_{\mathcal{A}} B$ iff $\mathsf{cl}(B)=\{\psi\in\Phi\mid\ \vdash t:\psi\}$;

(2) $\vdash t_1,\ldots,t_k : e$ iff $t_i\models_{\mathcal{A}} B_i$, $1\le i\le k$, such that $B_1\cdots B_k\in\mathcal{L}(\bar{e})$;

(3) $\vdash t_1,\ldots,t_k : \pi$ iff $t_i\models_{\mathcal{A}} B_i$, $1\le i\le k$, such that the Parikh image of $B_1\cdots B_k$ satisfies $\bar{\pi}$.

In particular, item (1) above implies that $\mathcal{L}(\varphi)=\mathcal{L}(\mathcal{A})$.

The three claims above are shown inductively. Items (2) and (3) above are immediate for $k=0$. For $k>0$ they follow from the definition of $\bar{e}$ and $\bar{\pi}$, together with item (1). Suppose now that $t=a\langle t_1,\ldots,t_k\rangle$, $k\ge 0$. Then $t\models_{\mathcal{A}} B$ iff $t_i\models_{\mathcal{A}} B_i$ for some B_i, $1\le i\le k$, such that:

- $B_1\cdots B_k\in\mathcal{L}(\bar{e})$ iff $a\langle e\rangle$ or $*\langle e\rangle$ is in B,

- the Parikh image of $B_1 \cdots B_k$ satisfies $\bar{\pi}$ iff $a\langle\pi\rangle$ or $*\langle\pi\rangle$ is in B.

By induction, $\mathsf{cl}(B_i) = \{\psi \mid \vdash t_i : \psi\}$ for all i. Using items (2) and (3) we infer that $a\langle F\rangle$ or $*\langle F\rangle$ is in B iff $\vdash t_1, \dots, t_k : F$, for all pre-conditions F. By the definition of $\mathsf{cl}(B)$ this is equivalent to $\mathsf{cl}(B) = \{\psi \mid \vdash t : \psi\}$.

For the converse, consider a deterministic PTA $\mathcal{A} = (Q, \Sigma, \delta, F)$. W.l.o.g. we may assume that every pre-condition is a disjunction of conjunctions of regular expressions and Presburger formulas. We introduce one variable x_q for every state $q \in Q$. For these variables, we construct an equation system $S_{\mathcal{A}}$:

$$x_q \;=\; \varphi_q \;, \quad q \in Q$$

where the right-hand sides are defined as fixpoint expressions, but without allowing the μ operator. The semantics of such equation systems is an extension of the semantics for fixpoint expressions. The only addition is the rule:

$$\frac{t : \varphi}{t : x}$$

for every equation $x = \varphi$. Thus, whenever a tree satisfies the right-hand side of an equation, then it also satisfies the variable to the left. The right-hand side φ_q for x_q in the equation system $S_{\mathcal{A}}$ is constructed from $\delta(q, a)$, $a \in \Sigma$, by:

$$\varphi_q \;=\; \bigvee_{a \in \Sigma} [\delta(q, a)]_a$$

where the transformation $[.]_a$ takes a pre-condition and returns a fixpoint expression (without fixpoints) as follows:

$$\begin{array}{lcl}
[e]_a & = & a\langle e\{q \mapsto x_q \mid q \in Q\}\rangle \,, \\
[\pi]_a & = & a\langle \pi\{\#q \mapsto \#x_q \mid q \in Q\}\rangle \,, \\
[\varphi_1 \vee \varphi_2]_a & = & [\varphi_1]_a \vee [\varphi_2]_a \,, \\
[\varphi_1 \wedge \varphi_2]_a & = & [\varphi_1]_a \wedge [\varphi_2]_a \,.
\end{array}$$

Thus, a regular expression over states q is transformed by first substituting the states by the corresponding variables and then putting a node a on top. A Presburger formula is transformed by first replacing the variables $\#q$ with $\#x_q$, and again putting a node a on top, whereas conjunctions and disjunctions are transformed recursively. By induction on the depth of terms $t, t_1, \dots, t_k$ we prove for every $q \in Q$, $a \in \Sigma$ and right-hand side φ:

(1) $t \models_{\mathcal{A}} q \quad$ iff $\quad \vdash t : x_q$;

(2) $t_i \models_{\mathcal{A}} q_i$ for $1 \leq i \leq k$, with $q_1 \cdots q_k \models \varphi \quad$ iff $\quad \vdash a\langle t_1, \dots, t_k\rangle : [\varphi]_a$.

The first claim then proves the correctness of the construction.

For the proof of the claims let us first assume that $t_i \models_{\mathcal{A}} q_i$ for all i, and $q_1 \cdots q_k \models \varphi$. We verify that for every $a \in \Sigma$, $a\langle t_1, \ldots, t_k\rangle : [\varphi]_a$ where, by inductive hypothesis, we may assume that $\vdash t_i : x_{q_i}$ for all i. If $\varphi = e$ is a regular expression, then by assumption, $q_1 \cdots q_k \in \mathcal{L}(e)$. By definition, $[\varphi]_a = a\langle e\{q \mapsto x_q \mid q \in Q\}\rangle$. Therefore, $x_{q_1} \cdots x_{q_k} \in \mathcal{L}(e\{q \mapsto x_q \mid q \in Q\})$ and hence $a\langle t_1, \ldots, t_k\rangle : [e]_a$. If $\varphi = \pi$ equals a Presburger formula, then the Parikh image of $x_{q_1} \cdots x_{q_k}$ satisfies $\pi\{\#q \mapsto \#x_q \mid q \in Q\}$. Let ρ denote the mapping defined by $\rho(\#x_q) = |\{i \mid \vdash t_i : x_q\}|$. Since the automaton $\mathcal{A}$ is deterministic, $t_i : x_q$ is provable for exactly one state q. Therefore, the number of occurrences of q in the sequence $q_1, \ldots, q_k$ precisely equals $\rho(\#x_q)$. We conclude that $t_1, \ldots, t_k : \pi\{\#q \mapsto \#x_q \mid q \in Q\}$ and therefore also $a\langle t_1, \ldots, t_k\rangle : [\pi]_a$. The cases $\varphi = \varphi_1 \wedge \varphi_2$ and $\varphi = \varphi_1 \vee \varphi_2$ are completely standard.

For the converse direction assume $a\langle t_1, \ldots, t_k\rangle : [\varphi]_a$ for some $a \in \Sigma$. By inductive hypothesis for t_i, we already know that there are (unique) states q_i such that $t_i \models_{\mathcal{A}} q_i$ and therefore also $\vdash t_i : x_{q_i}$, for all i. It remains to verify that $q_1 \cdots q_k \models \varphi$. If $\varphi = e$ is a regular expression, then $x_{q_1} \cdots x_{q_k} \in \mathcal{L}(e\{q \mapsto x_q \mid q \in Q\})$, thus $q_1 \cdots q_k \models \varphi$. If $\varphi = \pi$ equals a Presburger formula, then $[\varphi]_a = a\langle \pi\{\#q \mapsto \#x_q \mid q \in Q\}\rangle$. Since by assumption, $a\langle t_1, \ldots, t_k\rangle : [\varphi]_a$, we obtain $\rho \models \pi\{\#q \mapsto \#x_q \mid q \in Q\}$ for the assignment $\rho(\#x_q) = |\{i \mid \vdash t_i : x_q\}|$, $q \in Q$. Since $\mathcal{A}$ is deterministic, $\rho(\#x_q)$ equals the number of occurrences of q in the sequence $q_1, \ldots, q_k$. Therefore, $q_1 \cdots q_k \models \pi$. The case where $\varphi = \varphi_1 \vee \varphi_2$ or $\varphi = \varphi_1 \wedge \varphi_2$ are dealt with recursively.

To the equation system $S_{\mathcal{A}}$ we then apply *Gaussian elimination.* Thus, we take any equation $x_q = \varphi_q$ where φ_q possibly contains free occurrences of x_q, and replace it by $x_q = \mu\, x_q.\, \varphi_q$. Then we replace all free occurrences of x_q in all other right-hand sides $\varphi_{q'}, q' \neq q$, with the new fixpoint formula $\mu\, x_q.\varphi_q$. The resulting system still is equivalent to the original one but does no longer contain free occurrences of x_q in right-hand sides. We iteratively perform this step for every state q. Eventually, we arrive for each $q \in Q$ at an equation $x_q = \bar{\varphi}_q$ where $\bar{\varphi}_q$ is a closed fixpoint expression which denotes the set $\{t \in \mathcal{T}(\Sigma) \mid t \models_{\mathcal{A}} q\}$. Thus, the desired fixpoint formula $\varphi_{\mathcal{A}}$ can be chosen as:

$$\varphi_{\mathcal{A}} = \bigvee_{q \in F} \bar{\varphi}_q \,.$$

Q.E.D.

In the remainder of this section we turn to the complexity of Presburger fixpoint logic. Concerning satisfiability, Theorem 7.1 provides an EXPSPACE upper bound. The theorem below shows that this can be improved to EXPTIME, which is as good as we can hope for, since satisfiability of fixpoint formulas (without Presburger conditions) over binary trees is EXPTIME-complete (a similar result holds for model-checking μ-calculus against push-down graphs, [40]).

Theorem 7.2. The satisfiability problem for Presburger fixpoint formulas is EXPTIME-complete.

Proof. The lower bound is obtained, e.g., by encoding the accepting runs of an alternating polynomial space Turing machine through a binary tree.

It remains to prove the exponential upper bound. Let φ be a Presburger fixpoint formula. We denote by Ψ the set of its subformulas of type $a\langle F\rangle$ or $*\langle F\rangle$, and by Φ the set of *all* subformulas.

We call a subset $B \subseteq \Psi$ *obtainable* if there is a tree t such that, for each $\psi \in \Psi$, $\vdash t : \psi$ if and only if $\psi \in B$. In this case, we call t a *witness for B* and denote t by $t(B)$.

We compute in an inductive fashion the set of all obtainable sets $B \subseteq \Psi$. First, we compute the set X_0 of sets that are obtainable by some one-node tree t. Given X_i, we let X_{i+1} be the set of sets that are in X_i or are obtainable by a tree consisting of a root the subtrees of which are witnesses for the sets in X_i. As this process is monotonic it ends after at most $2^{|\Psi|}$ iterations, i.e., an exponential number of steps.

It therefore suffices to prove that each step takes no more than exponential time as well, actually we shall need here only polynomial space.

Let X denote a set of obtainable subsets of Ψ. We show that, given the fixpoint formula φ of size n and a set $B \subseteq \Psi$ it can be checked in space polynomial in n whether B is obtainable by a tree with subtrees which are witnesses for sets in X. Of course, X is not part of the input, since it might be of exponential size. We can imagine X as stored on a separate tape, and our PSPACE algorithm will access non-deterministically this tape.

A set B is only obtainable if there is some symbol a such that all formulas in B are either of the form $a\langle F\rangle$ or $*\langle F\rangle$. Accordingly, we must check whether there exists a sequence of sets $w = B_1 \dots B_h$ with $B_i \in X$ for all i, such that the tree $t = a\langle t(B_1), \cdots, t(B_h)\rangle$ makes all formulas in B true and all others false.

Consider first a formula of type $a\langle e\rangle$ ($*\langle e\rangle$, resp.), with e regular expression. By the definition of the closure of sets of formulas from Ψ, it is immediate that t satisfies $a\langle e\rangle$ ($*\langle e\rangle$, resp.) iff $w \in \mathcal{L}(\bar{e})$, where $\bar{e}$ is obtained from e by replacing every formula ψ with the disjunction of all $B' \in X$ with $\psi \in \mathsf{cl}(B')$. Likewise for $a\langle\neg e\rangle$ ($*\langle\neg e\rangle$, resp.).

For formulas $a\langle\pi\rangle, *\langle\pi\rangle$, with π Presburger formula, we first need the following definition. Let H denote the mapping which takes an assignment

$\sigma : X \to \mathbb{N}$ and computes an assignment $\tau : \Phi \to \mathbb{N}$ by

$$\tau(\psi) = \sum_{B' \in X \text{ with } \psi \in \mathsf{cl}(B')} \sigma(B') \,.$$

The tree $t = a\langle t(B_1), \cdots, t(B_h)\rangle$ (with $w = B_1 \dots B_h$) satisfies the formula $a\langle\pi\rangle$ ($*\langle\pi\rangle$, resp.) iff $H(\mathrm{Par}(w))$ satisfies π, where $\mathrm{Par}(w)$ denotes the Parikh vector of $w \in X^*$. The reader should recall here that with the fixpoint semantics a subtree can be counted several times, once for each formula it satisfies.

As in the proof of Proposition 4.3, we shall show the following:

Claim 7.3. If there exists a string which simultaneously verifies all formulas of type $a\langle F\rangle$ or $*\langle F\rangle$ in B, and falsifies all such formulas outside B, then there exists one whose length is bounded by $2^{p(n)}$ for some polynomial p.

We first show how the statement of the theorem follows from this claim. We successively guess subsets $B' \subseteq X$ (in polynomial space). For each such B', we simulate the evaluations of the non-deterministic automata corresponding to all regular expressions e occurring in $a\langle F\rangle \in \Psi$ or $*\langle F\rangle \in \Psi$. Of course, in order to do so, we need to check each time whether a subformula $\varphi' \in \Phi$ is in $\mathsf{cl}(B')$. All these simulations are done in PSPACE. During this process, we maintain an occurrence vector τ indexed by subformulas $\varphi' \in \Phi$. Whenever a set B' is processed, we increment in τ the values of all φ' contained in $\mathsf{cl}(B')$. Since each letter B' may have incremented each entry of τ at most by 1, the assignment τ can always be represented in polynomial space. Once we have guessed a sequence of length at most $2^{p(n)}$ verifying the formulas $a\langle e\rangle \in B$ and $*\langle e\rangle \in B$ and invalidating those outside B, we verify that τ satisfies the formula

$$\Big(\bigwedge_{a\langle\pi\rangle \in B \vee *\langle\pi\rangle \in B} \pi\Big) \wedge \Big(\bigwedge_{a\langle\pi\rangle \notin B \wedge *\langle\pi\rangle \notin B} \neg\pi\Big) \,.$$

The latter can be done in polynomial time (recall that each Presburger formula π is quantifier-free). This algorithm uses only space polynomial in n, therefore it can be executed in deterministic exponential time—which we wanted to prove.

It remains to show the claim above. Recall first that we defined the regular expressions $\bar{e}$ over the alphabet X by replacing each subformula φ' of φ by the disjunction of all $B' \in X$ with $\varphi' \in \mathsf{cl}(B')$. Now, we first construct an automaton $\mathcal{B}$ for the intersection of the regular expressions $\bar{e}$ (resp. $\neg\bar{e}$) occurring in formulas from B. This automaton has at most 2^n states, and its alphabet is of size 2^n. By Proposition 4.2, the Parikh image of the accepted language is a finite union $\mathrm{Par}(\mathcal{L}(\mathcal{B})) = L_1 \cup \dots \cup L_m$ of linear sets L_r of the form $\{\bar{c} + \sum_{i=1}^{h} x_i \cdot \bar{p}_i \mid x_i \geq 0\}$, where the entries of each vector $\bar{c}, \bar{p}_j$ are bounded by 2^n—whereas their number $h \leq (2^n + 1)^{2^n}$ might be doubly exponentially large. Recall however, that for the satisfiability of the Presburger formulas π occurring in formulas $a\langle\pi\rangle, *\langle\pi\rangle$ contained in B, we are not interested in the Parikh image $\mathrm{Par}(\mathcal{L}(\mathcal{B}))$ of the words accepted by $\mathcal{B}$ itself, but in the image of $\mathrm{Par}(\mathcal{L}(\mathcal{B}))$ under H. By definition, $H(\mathrm{Par}(\mathcal{L}(\mathcal{B}))) = H(L_1) \cup \cdots \cup H(L_m)$. Moreover, for every linear set of the form $L = \{\bar{c} + \sum_{i=1}^{h} x_i \cdot \bar{p}_i \mid x_i \geq 0\}$, the image $H(L)$ is given by $H(L) = \{\tau_0 + \sum_{i=1}^{h} x_i \cdot \tau_i \mid x_i \geq 0\}$ where $\tau_0 = H(\bar{c})$, $\tau_j = H(\bar{p}_j)$, $j = 1, \ldots, h$. This implies that each component in a vector τ_j is obtained by the sum of at most 2^n entries of the vectors $\bar{c}, \bar{p}_j$. Therefore, all entries of the τ_j are bounded by $2^n \cdot 2^n = 2^{2n}$. The crucial point is that the vectors τ_j now only have at most n entries (instead of 2^n for $\bar{c}, \bar{p}_j$). Accordingly, only $(2^{2n})^n = 2^{2n^2}$ of the τ_j can be distinct and therefore necessary to describe $H(L)$. Thus, now we may proceed along the same lines as in the proof of Proposition 4.3. A linear set L contained in the Parikh image $\mathrm{Par}(\mathcal{L}(\mathcal{B}))$ of $\mathcal{B}$ gives rise to a linear set $H(L)$ contained in $H(\mathrm{Par}(\mathcal{L}(\mathcal{B})))$, which in turn gives rise to at most n extra equations in 2^{2n^2} variables with coefficients bounded by 2^{2n}. These are to be added to $O(n)$ many equations obtained from the Presburger formulas from B. That is, as in Proposition 4.3 we consider a disjunct of the DNF of each formula π occurring in some Presburger formula form B (resp., with $\neg\pi$ occurring outside B), and we eliminate inequations and modulo equations using Proposition 2.1. Once again applying Papadimitriou's estimation [31], we obtain that the entries of a minimal solution $\tau \in H(\mathrm{Par}(\mathcal{L}(\mathcal{B}))) \cap S$, with

$$S = \Big(\bigwedge_{a\langle\pi\rangle \in B \vee *\langle\pi\rangle \in B} \pi \Big) \wedge \Big(\bigwedge_{a\langle\pi\rangle \notin B \wedge *\langle\pi\rangle \notin B} \neg\pi \Big)$$

are bounded by $2^{O(n^2)}$. Clearly, we have $\tau \in H(\mathrm{Par}(\mathcal{L}(\mathcal{B}))) \cap S$ iff there is some string $w \in \mathcal{L}(\mathcal{B})$ such that $H(\mathrm{Par}(w))$ satisfies S. Recall that by construction, $\top$ is contained in $\mathsf{cl}(B')$ for *every* subset $B' \subseteq \Psi$. Therefore, $H(\mathrm{Par}(w))(\top)$ precisely equals the length of w. Thus, the upper bound on the entries of τ proves the desired upper bound on the length of a shortest witness w and thus the claim. Q.E.D.

We finish this section with the following

Proposition 7.4. Given a tree t and a Presburger fixpoint formula φ, it can be checked in time $O(|t| \cdot |\varphi|^2)$ whether $t \models \varphi$.

Proof. We compute bottom-up the set of subformulas of φ that are satisfied by each subtree. For each subtree $t' = a\langle t_1, \ldots, t_k\rangle$ we simulate first the

NFA corresponding to regular expressions e ($\neg e$, resp.) occurring in preconditions $a\langle\ldots\rangle$ and $*\langle\ldots\rangle$, by keeping the set of reachable states of the NFA. Since each NFA is of size at most $|\varphi|$, each such simulation costs at most $O(k\cdot|\varphi|^2)$. For Presburger constraints $a\langle\pi\rangle, *\langle\pi\rangle$ we just need to count how many children satisfy a given subformula occurring in π, which can be done in $O(k\cdot|\varphi|)$, and to evaluate linear (in)equations and modular (in)equations. The last check is done in $O(|\varphi|^2)$. Finally, we compute $\mathsf{cl}(B)$ in $O(|\varphi|)$, with $B\subseteq\Psi$ the set of all $a\langle F\rangle$ or $*\langle F\rangle$ satisfied by $a\langle t_1,\ldots,t_k\rangle$.

Q.E.D.

8 Querying unranked trees

Presburger automata or logics can be used as a facility to express *unary queries*, i.e., to select a set of nodes in a document tree. We start this section with automata-based queries, and consider in Subsection 8.1 queries based on fixpoint logics, which exhibit a much better complexity than PTA-based queries.

With automata-based querying, a tree node is selected via an automaton $\mathcal{A}$ and a set T of states of $\mathcal{A}$. The node v is in the output, if there is an accepting computation of $\mathcal{A}$ that obtains a state from T at v. By the equivalence between Presburger automata and Presburger MSO logic (Thms. 3.8, 5.4, 7.1), this simple mechanism allows to express all (unary) queries definable in Presburger MSO logic.

Let $\bullet$ denote a fresh symbol (not in Σ). A *context* is defined as usual, as a tree $c\in\mathcal{T}(\Sigma\cup\{\bullet\})$ which contains exactly one occurrence of $\bullet$ at a leaf (the *hole*). Let $c[t']$ denote the tree which is obtained from c by substituting $\bullet$ with t' (i.e., filling the hole). Note that for a given tree t, the set $\mathcal{C}(t)$ of contexts c such that $t=c[t']$ for suitable subtrees t' is in one-to-one correspondence with the set of nodes of t. Therefore, in the following we shall no longer distinguish between contexts $c\in\mathcal{C}(t)$ and nodes of t.

A *(unary) query* is a mapping R from trees to subsets of nodes. The nodes in $R(t)$ are also called *matches*. In the following, we present a class of queries which is definable by means of (unordered, mixed) PTA. For this, we extend the definition of $\models_{\mathcal{A}}$ to contexts by defining $c,p\models_{\mathcal{A}} q$, ($p,q\in Q$) iff $c\models_{\mathcal{A}_{p,\bullet}} q$ where $\mathcal{A}_{p,\bullet}=(Q,\Sigma\cup\{\bullet\},\delta_{p,\bullet},F)$ is obtained from $\mathcal{A}$ by extending Σ with $\bullet$ and defining:

$$\delta_{p,\bullet}(q',a)=\begin{cases}\delta(q',a) & \text{if } a\in\Sigma\\ \mathsf{leaf} & \text{if } a=\bullet\wedge q'=p\\ \mathsf{false} & \text{if } a=\bullet\wedge q'\neq p\end{cases}.$$

Thus, the automaton $\mathcal{A}_{p,\bullet}$ behaves like $\mathcal{A}$ but additionally labels the hole by p. We have:

Proposition 8.1. Let $\mathcal{A} = (Q, \Sigma, \delta, F)$ be a PTA and $t = c[t']$ for a context c and $t, t' \in \mathcal{T}(\Sigma)$. Then $t \models_{\mathcal{A}} q$ iff $t' \models_{\mathcal{A}} p$ and $c, p \models_{\mathcal{A}} q$ for some $p \in Q$.

A (unary) *Presburger* pattern is a property of nodes of trees from $\mathcal{T}(\Sigma)$. We define this property by means of a pair $\langle \mathcal{A}, T\rangle$ where $\mathcal{A} = (Q, \Sigma, \delta, F)$ is a PTA (resp., a u-PTA or mixed PTA) and $T \subseteq Q$ is a set of states. Let $t \in \mathcal{T}(\Sigma)$. A context $c \in \mathcal{C}(t)$ is a *match* of the pattern $\langle \mathcal{A}, T\rangle$ in t iff $t = c[t']$ where $t' \models_{\mathcal{A}} q$ and $c, q \models_{\mathcal{A}} f$ for some $q \in T$ and $f \in F$.

We consider first mixed queries, with unordered ones as a special case. Whenever we speak about the complexity of the querying problem below, we mean the complexity of the following decision problem: given a query R, a tree t and a node v of t, is $v \in R(t)$?

Theorem 8.2. Let $\mathcal{A}$ be mixed PTA. The set of matches of a fixed Presburger pattern $\langle \mathcal{A}, T\rangle$, in a tree $t \in \mathcal{T}(\Sigma)$ of size n is computable in time $O(n)$. If the pattern is part of the input, the joint query complexity is NP-complete.

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, F)$. We proceed in two passes over the input tree t. In the first pass, we determine for every subtree t' of t the set of states:

$$B(t') = \{p \in Q \mid t' \models_{\mathcal{A}} p\} .$$

Let $\mathcal{A}'$ denote the deterministic automaton constructed from the mixed PTA $\mathcal{A}$ as in the proof of Theorem 6.2. Then we know that for every $t' \in \mathcal{T}(\Sigma)$, $t' \models_{\mathcal{A}'} B$ iff $B = \{p \in Q \mid t' \models_{\mathcal{A}} p\}$. Therefore, the sets $B(t')$ (over all subtrees t') can be determined by one bottom-up run of $\mathcal{A}'$ on t. According to Proposition 3.6, this first pass can be performed in linear time.

In the second pass, we determine for each context $c \in \mathcal{C}(t)$ with $t = c[t']$, the set of states:

$$D(c) = \{p \in B(t') \mid \exists f \in F : c, p \models_{\mathcal{A}} f\} .$$

Given the sets $D(c)$, the matches of the pattern are determined as the set of all contexts c where $T \cap D(c) \neq \varnothing$.

In order to determine the sets $D(c)$, we proceed top-down over t. For the root context c we set $D(c) = B(t) \cap F$. Assume that we are given a context c in t where $t = c[a\langle t_1, \ldots, t_k\rangle]$ for some $a \in \Sigma$ and subtrees t_i. Then we may proceed from the father node c to the son c_i which is defined as the context $c_i = c[a\langle t_1, \ldots, t_{i-1}, \bullet, \ldots, t_k\rangle]$. Remark that now $t = c_i[t_i]$. Let $B_i = B(t_i)$. Assume that we have already determined the value $D(c)$ and now want to determine the corresponding set for c_i.

Suppose first that the tag a is unordered, $a \in \Sigma_0$. For $B \subseteq Q$, let n_B denote the number of trees t_j, $1 \leq j \leq k$, such that $t_j \models_{\mathcal{A}'} B$. Let ρ denote the variable environment defined by:

$$\{x_B \mapsto n_B \mid B \subseteq Q\} .$$

We claim:
$$D(c_i) = \Big\{q' \in B(t_i) \mid \rho \models \bigvee_{q\in D(c)} \psi_{q,q'}\Big\}$$
where the formula $\psi_{q,q'}$ is given by:
$$\exists_{p\in Q} \#p\,.\ \exists_{p\in B\subseteq Q} x_{\langle B,p\rangle}\,.\ \delta(q,a)\ \wedge\ x_{\langle B_i,q'\rangle} > 0 \wedge$$
$$\Big(\bigwedge_{B\subseteq Q}\sum_{p\in B} x_{\langle B,p\rangle} = x_B\Big) \wedge \Big(\bigwedge_{p\in Q}\sum_{B,p\in B} x_{\langle B,p\rangle} = \#p\Big)\,.$$
Intuitively, formula $\psi_{q,q'}$ expresses that there is an assignment mapping the children t_j to states $q \in B(t_j)$ such that t_i receives q' and the Presburger pre-condition $\delta(q,a)$ is satisfied. Since satisfiability of Presburger formulas is decidable, we conclude that the sets $D(c_i)$ are computable.

The total complexity of our algorithm in this part consists, for each node v labeled in Σ_0, in a test of an assertion $\rho \models \varphi$. Here, the formula φ only depends on the fixed automaton $\mathcal{A}$, and the variable environment ρ is such that $\rho(x_{\langle B,p\rangle}) \leq k$ for all $x_{\langle B,p\rangle}$ in the domain of ρ, with k denoting the number of children of v. Each formula φ can be transformed into a quantifier-free formula, which is evaluated in time $O(k)$ on numbers in unary representation. Since the sum of all k is bounded by n, the total complexity is in $O(n)$.

In the case where $a \in \Sigma_1$ we have:
$$\begin{array}{rcll} D(c_i) & = & \bigcup\{D_q(i) \mid q \in D(c)\} & \text{where} \\ D_q(i) & = & \{p_i \in B_i \mid \forall j \neq i\ \exists p_j \in B_j :\ p_1 \dots p_k \in \delta(q,a)\}\,. & \end{array}$$
Given a (non-deterministic) finite automaton $\mathcal{B}$ for $\delta(q,a)$, all sets $D_q(i)$, $i = 1,\dots,k$, can be computed in time $O(k)$ as follows: by one left-to-right pass we compute at each position the set of reachable states of $\mathcal{B}$; in a second, right-to-left pass we compute at each position the set of states from which we can reach a final state of $\mathcal{B}$. With this information we compute all sets $D_q(i)$ in a final pass in $O(k)$.

Therefore, the overall complexity of the second pass is linear as well. This completes the proof in the case where the pattern is fixed.

For the joint complexity, consider first the upper bound. The first pass can be done deterministically in polynomial time, by computing bottom-up the reachable states at each node. For the top-down pass, we solve at each node an existential Presburger formula, which is done in NP. The lower bound follows from Proposition 3.6. Q.E.D.

As a special case of the querying algorithm in the proof of Theorem 8.2, we obtain a linear time querying algorithm for (fixed) queries on *classical* ordered trees (i.e., trees with $\Sigma_0 = \varnothing$).

We now consider ordered queries, i.e., queries stated as Presburger patterns $\langle \mathcal{A}, T \rangle$ where $\mathcal{A}$ is a PTA.

Theorem 8.3. The set of matches of a fixed Presburger pattern $\langle \mathcal{A}, T \rangle$, with $\mathcal{A}$ PTA, in a tree from $\mathcal{T}(\Sigma)$ is computable in polynomial time. If the pattern is part of the input, the joint query complexity is NP-complete.

Proof. Assume we have marked the root node of one subtree t' of t. Assume further that we have modified $\mathcal{A}$ in such a way that the marked node always receives a state in T. Then the modified tree is accepted iff t' is a match. Since there are only n different nodes to be marked, the theorem follows from Theorem 5.2.

For the joint query complexity we can implement easily the 2-pass approach of Theorem 8.2 in NP. The lower bound follows from the combined complexity of PTA. Q.E.D.

Let us turn to queries specified through Presburger MSO. A mixed PMSO-pattern is a mixed PMSO formula φ with at most one free variable y. A match of φ in t at a node v means that t together with the assignment of v to the free variable y satisfies φ. A query R is *mixed PMSO-definable* iff there is a mixed PMSO-pattern φ such that for every t, $R(t)$ is the set of all matches of φ in t. Replacing mixed PMSO by *existential* PMSO, we get *existential PMSO-definable* queries.

Theorem 8.4. For a query R the following statements hold:

1. R is mixed PMSO-definable iff R is definable by a Presburger pattern $\langle \mathcal{A}, T \rangle$ for some mixed PTA $\mathcal{A}$.
2. R is existential PMSO-definable iff R is definable by a Presburger pattern $\langle \mathcal{A}, T \rangle$ for some PTA $\mathcal{A}$.

In comparison with PTA-based queries, it is worth noting that the joint query complexity of mixed PMSO-definable and existential PMSO-definable queries is PSPACE-complete. Both arguments for the upper and the lower bound use that alternating polynomial time is equivalent to PSPACE.

8.1 Presburger fixpoint queries

In this section we focus on unary queries expressed in Presburger fixpoint logic. Compared to PTA, fixpoint logic allows for very efficient algorithms—linear time for fixed queries and polynomial time for the joint query complexity.

In order to get an intuition about the expressive power of Presburger fixpoint logic, consider the example document shown in Figure 2. There we might first ask for all elements (tree nodes) containing “`Bartoli`”. A second query could ask for elements containing “`Bartoli`” and having at least

```
<music> ...
   <classical> ...
      <opera>
         <title> The Salieri Album </title>
         <composer> Bartoli </composer>
         <review> ... </review>
         <review> ... </review>
         <review> ... </review>
      </opera>
      <opera>
         <title> The No. 1 Opera Album </title>
         <composer> Puccini ; Verdi </composer>
         <performer> Bartoli ; Pavarotti </name> </performer>
         <review> ... </review>
      </opera> ...
   </classical> ...
</music>
<dvd> ...
   <music dvd>
      <opera>
         <title> Rossini - La Cenerentola </title>
         <performer> Bartoli </performer>
         <review> ... </review>
         <review> ... </review>
      </opera> ...
   </music dvd>
</dvd>
```

FIGURE 2. Part of a document with music items.

three reviews. In the fixpoint Presburger logic we can express that a tree contains a node satisfying a given property, without knowing at which depth this node occurs. For instance, the formula $\varphi_1 = *\langle \_ \ \texttt{Bartoli} \ \_\rangle$ describes all nodes containing "Bartoli". Note that in order to take properties of text contents into account, it (conceptually) suffices to consider each text as a tag. We are not interested in the class of all these documents t, however, but for each such t in the subdocuments which satisfy the specific formula φ_1. Documents containing elements with the property φ_1 are described by the expression: $\mu\, x.(*\langle \_ \ x \ \_\rangle \vee \varphi_1)$. In order to indicate the subformula corresponding to the requested subdocuments, we introduce the extra marker "$\bullet$". Thus, we specify the query as $\psi_1 = \mu x.(*\langle \_ \ x \ \_\rangle \ \vee \ (\bullet \wedge \varphi_1))$. Accordingly for the second query, we describe the set of all elements containing at least three reviews by: $\varphi_2 = *\langle\#\texttt{review} \geq 3\rangle$. The query formula then can be formulated as:

$$\psi_2 = \mu x.(*\langle\_\ x\ \_\rangle \ \vee \ (\bullet \wedge \varphi_1 \wedge \varphi_2))\,.$$

In order to obtain a query language, we formally extend the language of Presburger fixpoint expressions by one extra case:

$$\varphi \quad ::= \quad \ldots \mid \bullet \mid \ldots\,.$$

Accordingly, we add new axioms $\vdash t : \bullet$ for all trees t. A *match* t' of a formula φ containing a subformula $\bullet$ is a proof for $t : \varphi$ containing the fact $t' : \bullet$. We want to construct an algorithm to determine for a fixed query formula φ, all matches inside a document tree t. We first observe that we can determine in time $O(|t|)$ for every subtree t' of t the set of all subformulas ψ of φ such that $\vdash t' : \psi$. For that, we can do as in Proposition 7.4 a bottom-up pass on t. In order to deal with the special symbol $\bullet$ occurring in φ, we extend the notion of closure of states by adding the formula $\bullet$. The rest of the construction is unchanged. Let then $S(t')$ denote the set of subformulas ψ of type $a\langle F\rangle$, $*\langle F\rangle$ such that $t' : \psi$. By construction, $\psi \in \mathsf{cl}(S(t'))$ iff $\vdash t' : \psi$, for *every* subformula ψ of φ.

It remains to determine for every subtree t' of t the subset $R(t') \subseteq \mathsf{cl}(S(t'))$ containing all those ψ which may occur in some proof of $t : \varphi$. Then t' is a match iff $\bullet \in R(t')$. The subsets $R(t')$ are determined in a second pass over the tree t, in a top-down manner. For a closed set of subformulas B, we introduce the auxiliary function core_B which takes a subformula ψ of φ and returns the set of all subformulas in B which potentially contribute to any proof of ψ (including ψ). Let $\mathsf{core'}_B(\psi) = \mathsf{core}_B(\psi) \setminus \{\psi\}$. So, $\mathsf{core'}_B(\bullet) = \mathsf{core'}_B(\top) = \varnothing$, and

$$\begin{array}{lcll}
\mathsf{core'}_B(\mu\, x.\psi) & = & \mathsf{core}_B(\psi) & \text{if } \psi \in B \\
\mathsf{core'}_B(x) & = & \mathsf{core}_B(\psi) & \text{if } \psi \in B \\
\mathsf{core'}_B(\psi_1 \wedge \psi_2) & = & \mathsf{core}_B(\psi_1) \cup \mathsf{core}_B(\psi_2) & \\
\mathsf{core'}_B(\psi_1 \vee \psi_2) & = & \left\{\begin{array}{ll} \mathsf{core}_B(\psi_i) & \text{if } \psi_{3-i} \notin B \\ \mathsf{core}_B(\psi_1) \cup \mathsf{core}(\psi_2) & \text{otherwise} \end{array}\right. & \\
\mathsf{core'}_B(a\langle F\rangle) & = & \varnothing & \\
\mathsf{core'}_B(*\langle F\rangle) & = & \varnothing\,. &
\end{array}$$

Moreover, we set: $\mathsf{core}_B(R) = \bigcup_{\psi\in R} \mathsf{core}_B(\psi)$ for every $R \subseteq B$.

The second pass over t starts at the root of t. There, we have: $R(t) = \mathsf{core}_B(\varphi)$ for $B = \mathsf{cl}(S(t))$. Now assume we have already computed the set $R(t')$ for the subtree $t' = a\langle t_1 \ldots t_k\rangle$. Let $R' = R(t') \cap S(t')$ denote the set of subformulas in $R(t')$ of the form $a\langle F\rangle$ or $*\langle F\rangle$. Then $R(t_i) = \bigcup_{\psi\in R'} R_\psi(t_i)$, where $R_\psi(t_i)$ equals the set of formulas from $\mathsf{cl}(S(t_i))$ which may have occurred in a proof of $t' : \psi$. Let $B_i = \mathsf{cl}(S(t_i))$ be the set of all subformulas that are valid at t_i. If $\psi = a\langle\pi\rangle$ or $\psi = *\langle\pi\rangle$ for a Presburger formula π, then we must compute the assignment to the variables of π. In fact, *all* subformulas from B_i contribute to this assignment. Therefore, we simply have $R_\psi(t_i) = B_i$ in this case. On the other hand, if $\psi = a\langle e\rangle$ or $\psi = *\langle e\rangle$ for a regular expression e, then $R_\psi(t_i) = \mathsf{core}_{B_i}(R_i)$ where

$$R_i = \{\psi_i \mid \exists\, \psi_1 \ldots \psi_k \in \mathcal{L}(e) : \forall\, j : \psi_j \in B_j\}\,.$$

The set R_i denotes all subformulas provable for t_i which may contribute to the validation of e. According to this definition, the sets $R_\psi(t_i)$, $i = 1, \ldots, k$ can jointly be computed by a left-to-right followed by a right-to-left pass of a finite (string) automaton for e over the children of t'. The case of negated regular expressions is treated analogously. Summarizing we conclude:

Theorem 8.5. Let φ be a fixed query in Presburger fixpoint logic. Then the set of matches of φ in an input tree t can be computed in time linear in $|t|$. If φ is part of the input, the joint query complexity is $O(|\varphi|^2 \cdot |t|)$.

9 Conclusion

We have considered extensions of logics and automata over unranked trees by arithmetical Presburger constraints. Our motivation comes from XML, where one is interested in expressing properties of such trees that go beyond regular languages, such as numerical constraints. We showed that fixpoint logic extended by Presburger constraints has particularly pleasant properties, namely good expressiveness, complexity which does not increase with the additional Presburger part, and joint querying complexity which is polynomial.

Some of our results raise open problems. The universality problem for u-PTA is one of them: we have a 2-NEXPTIME upper bound, and as lower bound only EXPTIME. Another issue is the data complexity for general PTA: can we improve the bound or is it inherently difficult (w.r.t. fixed parameter complexity, with the size of the PTA as parameter)? Finally, it would be interesting to see whether the automata and logics can be enhanced by more general arithmetical constraints, like for instance the semi-polynomial or semi-quadratic sets considered in [16].

References

[1] L. Berman. The complexity of logical theories. *Theoret. Comput. Sci.*, 11(1):71–77, 1980. With an introduction "On space, time and alternation".

[2] I. Boneva and J.-M. Talbot. Automata and logics for unranked and unordered trees. In J. Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 500–515. Springer, 2005.

[3] A. Brüggeman-Klein, M. Murata, and D. Wood. Regular tree languages over non-ranked alphabets, 1998. Unpublished manuscript.

[4] L. Cardelli and G. Ghelli. A query language based on the ambient logic. In D. Sands, editor, *ESOP*, volume 2028 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

[5] L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *POPL*, pages 365–377, 2000.

[6] G. Conforti, O. Ferrara, and G. Ghelli. TQL algebra and its implementation. In R. A. Baeza-Yates, U. Montanari, and N. Santoro, editors, *IFIP TCS*, volume 223 of *IFIP Conference Proceedings*, pages 422–434. Kluwer, 2002.

[7] G. Conforti, G. Ghelli, A. Albano, D. Colazzo, P. Manghi, and C. Sartiani. The query language TQL - demo presentation. In *SEBD*, pages 427–431, 2002.

[8] J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In M. Liskiewicz and R. Reischuk, editors, *FCT*, volume 3623 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2005.

[9] S. Dal-Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In R. Nieuwenhuis, editor, *RTA*, volume 2706 of *Lecture Notes in Computer Science*, pages 246–263. Springer, 2003.

[10] S. Dal-Zilio, D. Lugiez, and C. Meyssonnier. A logic you can count on. In N. D. Jones and X. Leroy, editors, *POPL*, pages 135–146. ACM, 2004.

[11] S. Demri and D. Lugiez. Complexity of modal logics with Presburger constraints. Technical Report LSV-06-15, LSV, ENS Cachan, 2006.

[12] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inform.*, 31(1):13–25, 1997.

[13] M. J. Fischer and M. O. Rabin. Super-exponential complexity of Presburger arithmetic. In *Complexity of computation (Proc. SIAM-AMS Sympos., New York, 1973)*, pages 27–41. SIAM–AMS Proc., Vol. VII, Providence, R.I., 1974. Amer. Math. Soc.

[14] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific J. Math.*, 16:285–296, 1966.

[15] G. Gottlob and C. Koch. Monadic Datalog and the expressive power of languages for web information extraction. In L. Popa, editor, *PODS*, pages 17–28. ACM, 2002.

[16] W. Karianto, A. Krieg, and W. Thomas. On intersection problems for polynomially generated sets. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 516–527. Springer, 2006.

[17] F. Klaedtke. On the automata size for Presburger Arithmetic. In *LICS*, pages 110–119. IEEE Computer Society, 2004.

[18] F. Klaedtke and H. Ruess. Parikh automata and monadic second-order logics with linear cardinality constraints. Technical Report 177, Institute of CS at Freiburg University, 2002.

[19] O. Kupferman, U. Sattler, and M. Y. Vardi. The complexity of the graded μ-calculus. In A. Voronkov, editor, *CADE*, volume 2392 of *Lecture Notes in Computer Science*, pages 423–437. Springer, 2002.

[20] D. Lugiez. A good class of tree automata and application to inductive theorem proving. In K. G. Larsen, S. Skyum, and G. Winskel, editors, *ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 409–420. Springer, 1998.

[21] D. Lugiez and S. Dal Zilio. Multitrees automata, Presburger's constraints and tree logics. Technical Report 08-2002, Laboratoire d'Informatique Fondamentale de Marseille, 2002.

[22] W. Martens and J. Niehren. Minimizing tree automata for unranked trees. In G. M. Bierman and C. Koch, editors, *DBPL*, volume 3774 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2005.

[23] M. L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Ann. of Math. (2)*, 74:437–455, 1961.

[24] A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In V. Arvind and R. Ramanujam, editors, *FSTTCS*, volume 1530 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 1998.

[25] F. Neven. Automata, logic, and XML. In J. C. Bradfield, editor, *CSL*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002.

[26] F. Neven and J. V. den Bussche. Expressiveness of structured document query languages based on attribute grammars. *J. ACM*, 49(1):56–100, 2002.

[27] F. Neven and T. Schwentick. Query automata over finite trees. *Theor. Comput. Sci.*, 275(1-2):633–674, 2002.

[28] J. Niehren and A. Podelski. Feature automata and recognizable sets of feature trees. In M.-C. Gaudel and J.-P. Jouannaud, editors, *TAP-SOFT*, volume 668 of *Lecture Notes in Computer Science*, pages 356–375. Springer, 1993.

[29] H. Ohsaki. Beyond regularity: Equational tree automata for associative and commutative theories. In L. Fribourg, editor, *CSL*, volume 2142 of *Lecture Notes in Computer Science*, pages 539–553. Springer, 2001.

[30] H. Ohsaki and T. Takai. Decidability and closure properties of equational tree languages. In S. Tison, editor, *RTA*, volume 2378 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2002.

[31] C. H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

[32] R. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.

[33] M. Presburger. On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. *Hist. Philos. Logic*, 12(2):225–233, 1991. Translated from the German and with commentaries by Dale Jacquette.

[34] H. Seidl and A. Neumann. On guarding nested fixpoints. In J. Flum and M. Rodríguez-Artalejo, editors, *CSL*, volume 1683 of *Lecture Notes in Computer Science*, pages 484–498. Springer, 1999.

[35] H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *PODS*, pages 155–166. ACM, 2003.

[36] H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. Counting in trees for free. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer, 2004.

[37] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9. ACM, 1973.

[38] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[39] K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational Horn clauses. In R. Nieuwenhuis, editor, *CADE*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2005.

[40] I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

Modular quantifiers*

Howard Straubing[1]
Denis Thérien[2]

[1] Computer Science Department
Boston College
Fulton Hall
Chestnut Hill MA, 02476, U.S.A.
`straubin@cs.bc.edu`

[2] School of Computer Science
McGill University
3480 University Street
Montréal QC, Canada
`denis@cs.mcgill.ca`

Abstract

In the 1980's, Wolfgang Thomas, together with the authors, introduced the study of formulas with quantifiers that are interpreted "there exist r mod q elements x such that...." and used algebraic techniques to characterize the regular languages defined with such quantifiers. The present paper surveys this work and the many other results it spawned, especially the applications to formulas with a bounded number of variables, and the rather surprising connections with circuit complexity.

In the late nineteen-eighties much of our research concerned the application of semigroup-theoretic methods to automata and regular languages, and the connection between computational complexity and this algebraic theory of automata. It was during this period that we became aware of the work of Wolfgang Thomas. Thomas had undertaken the study of concatenation hierarchies of star-free regular languages—a subject close to our hearts—by model-theoretic methods. He showed that the levels of the dot-depth hierarchy corresponded precisely to levels of the quantifier alternation hierarchy within first-order logic [26], and applied Ehrenfeucht-Fraïssé games to prove that the dot-depth hierarchy was strict [27], a result previously obtained by semigroup-theoretic means [4, 18].

Finite model theory, a subject with which we'd had little prior acquaintance, suddenly appeared as a novel way to think about problems that we had been studying for many years. We were privileged to have been introduced to this field by so distinguished a practitioner as Wolfgang Thomas,

* The second author was supported by grants from NSERC and FQRNT

and to have then had the opportunity to work together with him. The study of languages defined with modular quantifiers, the subject of the present survey, began with this collaboration.

1 Generalized first-order formulas over $<$

1.1 First-order logic and star-free sets

A little background first, about the ordinary kind of quantifier: Properties of words over a finite alphabet Σ can be expressed in predicate logic by interpreting variables as positions $\{0, 1, \ldots, n-1\}$ in a string of length n, and including for each $\sigma \in \Sigma$ a unary relation symbol Q_σ, where $Q_\sigma x$ is interpreted to mean that the letter in position x is σ. The signature typically includes other predicate symbols—"numerical predicates"—that are independent of the letters that appear in the positions concerned, but instead allow us to talk about relations between positions. It is quite interesting to see what happens when we modify this part of the signature, but for now we shall suppose that there is just one such predicate symbol: $<$, denoting the usual order on the positions.

A sentence of first-order logic thus defines a language in Σ^*, namely the set of all strings that satisfy the sentence. For example, if $\Sigma = \{\sigma, \tau\}$, then the sentence

$$\exists x(Q_\sigma x \wedge \neg\exists y(y < x))$$

says that there is a position containing the letter σ preceded by no other position, and thus defines the language $\sigma\Sigma^*$ of words whose first letter is σ.

Likewise the sentence

$$\begin{aligned} \exists x(Q_\sigma x \wedge \neg\exists y(y < x)) \quad &\wedge \quad \exists x(Q_\tau x \wedge \neg\exists y(y > x)) \\ &\wedge \quad \forall x\Big(Q_\tau x \leftrightarrow \exists y\big(Q_\sigma y \wedge (y < x) \\ &\wedge \quad \forall z(z > y \rightarrow \neg(z < x))\big)\Big) \end{aligned}$$

says that the first letter is σ, the last letter is τ, and that the positions containing τ are those immediately following positions containing σ. This defines the language $(\sigma\tau)^*$.

Now $(\sigma\tau)^*$ is a *star-free* subset of $\{\sigma, \tau\}^*$. This means that it can be defined by an extended regular expression in which arbitrary boolean operations are permitted along with concatenation, but in which the star operation is not used. This may not be obvious at first, since we have certainly used the star to write it! But in fact this language is identical to

$$\big(\tau\varnothing^c \cup \varnothing^c\sigma \cup \varnothing^c(\sigma\sigma \cup \tau\tau)\varnothing^c\big)^c,$$

where the superscript c denotes complementation in $\{\sigma,\tau\}^*$.

McNaughton and Papert [13] showed that the star-free languages are exactly those definable by first-order sentences over $<$. It is not hard to see how to express the concatenation operation in first-order logic, so let us concentrate instead on why the converse is true. Our account here is inspired by the treatment in Thomas [26]. If $w_1, w_2 \in \Sigma^*$, and $k \geq 0$, then we write $w_1 \equiv_k w_2$ to mean that w_1 and w_2 satisfy all the same sentences of quantifier depth k. We write $[w]_k$ to denote the equivalence class of the word w under this relation. One can now show that for any word v,

$$[v]_{k+1} = \bigcap_{*} [v_1]_k \sigma [v_2]_k - \bigcup_{**} [u_1]_k \sigma [u_2]_k.$$

Here, the index set $*$ in the intersection is the set of all triples (v_1,σ,v_2) such that $\sigma \in \Sigma$ and $v = v_1\sigma v_2$, and the union over the set of all triples (u_1,σ,u_2) such that v does not have a factorization $v_1\sigma v_2$ with $v_i \equiv_k u_i$ for $i = 1,2$. This can be established with a by-now routine argument using games. Since $\equiv_k$ has finite index, the intersection is in fact a finite intersection. So the above equation shows that for all k, each $\equiv_k$-class is a star-free language. Since a language defined by a sentence of depth k is a finite union of such classes, we get the desired result.

Observe that the argument outlined above makes no use of the *other* characterization of star-free languages, namely Schützenberger's Theorem that these are exactly the languages whose syntactic monoids are aperiodic (*i.e.,* contain no nontrivial groups) [16]. But algebra and semigroups are not completely absent, for the equivalences $\equiv_k$ are congruences of finite index on Σ^*, and the content of Schützenberger's Theorem shows in essence that the quotient monoids of these congruences generate all the finite aperiodic monoids.

1.2 Counting factorizations

Earlier (in our Ph.D. dissertations!) we had both studied a variant of the concatenation operation that counted factorizations modulo some period: Let $L_1, L_2 \subseteq \Sigma^*$, $\sigma \in \Sigma$, and $0 \leq r < q$. Then we define (L_1,σ,L_2,r,q) to be the set of words w for which the number of factorizations $w = w_1\sigma w_2$ if congruent to r modulo q.

In our discussions with Thomas we realized that the precise power of this operation could be captured if one introduced *modular quantifiers* into our logical languages:

$$\exists^{r \bmod q} x \varphi(x)$$

is interpreted to mean 'the number of positions x for which $\varphi(x)$ holds' is congruent to r modulo q.

As an example, consider the sentence

$$\exists^{0 \bmod 3} x(Q_\tau x \wedge \exists^{0 \bmod 2} y(Q_\sigma y \wedge y < x)).$$

This defines the set of all strings in which the number of occurrences of τ preceded by an even number of σ is divisible by 3. Observe that this particular sentence uses modular quantifers exclusively, and that it is possible to rewrite it so that it only uses modular quantifiers of modulus 6.

We were able to adapt the argument given above for star-free languages to this new quantifer: Let us fix a modulus q, and let us redefine $v_1 \equiv_k v_2$ to mean that v_1 and v_2 satisfy the same sentences of quantifier depth k, where we now allow modular quantifiers of modulus q as well as ordinary quantifiers. Let L be defined by the sentence

$$\exists^{r \bmod q} x\varphi(x),$$

where φ has depth k. We showed that L is a boolean combination of languages of the form (K, σ, K', s, q), where K and K' are $\equiv_k$-classes. The same conclusion holds if we define $\equiv_k$ in terms of modular quantifiers exclusively.

It readily follows that languages constructed using boolean operations and ordinary concatenation together with the operations (L_1, σ, L_2, r, q) are exactly those defined by sentences using both ordinary and modular quantifiers, and that languages defined using the operations (L_1, σ, L_2, r, q) alone are exactly those definable using only modular quantifiers.

Our real interest, however, stemmed from the fact that these language classes could all be characterized effectively in semigroup-theoretic terms. The example language defined above with quantifiers of modulus 2 and 3 was derived from a descripiton of the set of words in the permutations $\sigma = (1, 2)$ and $\tau = (1, 2, 3)$ that evaluate to the identity in the symmetric group S_3. This works in general for finite solvable groups, for we can derive such descriptions of word problems from the composition series for the groups. It turns out that the languages definable using only modular quantifiers are exactly the languages whose syntactic monoids are solvable groups, and those definable using both modular and ordinary quantifiers are exactly those whose syntactic monoids contain only solvable groups.

Let us denote by FO[<] the family of languages definable by first-order sentences over $<$, by $(\mathrm{FO} + \mathrm{MOD}_q)[<]$ those definable with both ordinary first-order quantifiers and modular quantifiers of modulus q, and by $\mathrm{MOD}_q[<]$ those definable using only modular quantifiers of modulus q. (We assume all of this is with respect to a fixed finite alphabet Σ.)

Theorem 1.1. (Straubing, Thérien and Thomas [22]) $\mathrm{MOD}_q[<]$ is the family of regular languages whose syntactic monoids are solvable groups of cardinality dividing a power of q. $(\mathrm{FO} + \mathrm{MOD}_q)[<]$ is the family of regular languages L such that every group in $M(L)$ is a solvable group of cardinality dividing a power of q.

This second of these facts is far deeper than the first: While a solvable group by definition decomposes into a sequence of extensions by cyclic groups, which generates the expression in terms of modular quantifiers, the existence of a comparable decomposition for monoids that contain solvable groups requires the use of the Krohn-Rhodes Theorem [12].

The result of this is that we are able to effectively decide if a regular language, given, let us say, by a regular expression or an automaton, can be defined by a sentence involving modular quantifiers, and if so actually produce the sentence. For instance, suppose L is recognized by a deterministic automaton with four states. We can explicitly write down a Krohn-Rhodes decomposition of the monoid of all transformations on a four-element set into factors that are either small aperiodic monoids or cyclic groups of order two or three. This can be used to produce a sentence for L containing ordinary quantifiers along with modular quantifiers of modulus 2 and 3. In contrast, if the minimal DFA for L has five states, and if the transition monoid contains all the even permutations of the states, then no such sentence for L is possible, irrespective of the moduli used.

1.3 Quantifiers and the block product

The two-sided decomposition theory for finite monoids developed by Rhodes and Tilson [14] permits a deep understanding of the connection between logic and algebra that underlies Theorem 1.1. Suppose that M and N are two finite monoids. We write the operation in M additively, and its identity as 0. This is not meant to imply that M is commuative, although in fact, in the critical examples we consider below, M will be commutative. We consider both a left action and a right action of N on M that are compatible in the sense that

$$(nm)n' = n(mn')$$

for all $m \in M$; $n, n' \in N$. We further suppose that these actions respect the identities in both monoids, so that

$$n0 = 0n = 0$$

for all $n \in N$, and

$$1m = m1 = m$$

for all $m \in M$. The *bilateral semidirect product* $M ** N$ with respect to these actions is the monoid whose underlying set is $M \times N$ and whose operation is given by

$$(m_1, n_1)(m_2, n_2) = (m_1 n_2 + n_1 m_2, n_1 n_2).$$

Rhodes and Tilson also define a *block product* $M\Box N$, a bilateral semidirect product of $M^{N\times N}$ and N that contains all the bilateral semidirect products $M ** N$.

The connection with quantification comes in when we consider languages recognized by bilateral semidirect products $M ** N$ (or, what is the same thing, block products $M\Box N$) in which M is either idempotent and commutative, or an abelian group. This becomes clear if we try to compute the image of a word $w = \sigma_1 \cdots \sigma_r$ under a homomorphism into the bilateral semidirect product. If we suppose that this morphism maps σ_i to (m_i, n_i), then w is mapped to:

$$\prod_{i=1}^{r}(m_i, n_i) = \Big(\sum_{i=1}^{r}(\prod_{j=1}^{i-1} n_j) m_i (\prod_{k=i+1}^{r} n_k), \prod_{i=1}^{r} n_i\Big).$$

In other words, computation in $M ** N$ keeps track in M of the factorizations $w = u\sigma v$, where the images of u and v are computed in N. It follows that if M is idempotent and commutative, then a language recognized by $M ** N$ is a boolean combination of languages of the form $L\sigma L'$, where L, L' are recognized by N; and that if M is an abelian group of exponent q, then any language recognized by $M ** N$ is a boolean combination of languages of the form (L, σ, L', r, q), where again L and L' are recognized by L. As mentioned above, these language operations can be captured by application of ordinary and modular quantifiers.

Conversely, consider a language L defined by a sentence of the form

$$\exists x \varphi,$$

or

$$\exists^{r \bmod q} x \varphi,$$

where φ is itself a formula with ordinary and modular quantifiers over the signature $\{<\}$. We can view the formula φ, which has a single free variable x, as defining a language L_φ over the extended alphabet $\Sigma \times 2^{\{x\}}$. Elements of this language are words in which one of the positions is marked, and which satisfy φ when the free variable is instantiated by the marked position. Likewise, we can view a formula with k free variables as definining a language of marked words with k distinct marks, some of which may coincide. Let $\mu_L : \Sigma^* \to M(L)$ be the syntactic morphism of L, and $\nu_L : A^* \to M(L')$.

This is where the decomposition theory of Rhodes and Tilson comes in. The relation $\nu_L \mu_L^{-1} : M(L) \to M(L')$ is a *relational morphism,* and its *kernel category* is idempotent and commutative (in the case of ordinary quantifiers) or covered by an abelian group of exponent q (in the case of modular quantifiers). This implies that $M(L)$ is recognized by a block product $K\Box M(L')$, where K is idempotent and commutative or an abelian group of exponent q, depending on the quantifiers.

Theorem 1.1 follows from these observations and the Krohn-Rhodes Theorem for block products: The solvable groups of order dividing a power of q for the smallest variety of finite monoids closed under block product and containing all the abelian groups of exponent q, and all the idempotent and commutative monoids. This is the approach taken in the journal version of our paper with Wolfgang Thomas [23], and in Straubing [19], which considers a large assortment of regular language classes defined with modular quantifiers.

2 Circuits

2.1 Constant-depth circuits and the $\mathbf{ACC}^0$ problem

Why study modular quantifiers in the first place? To be frank, when we began our work we did not have a particularly compelling answer to this question! Modular counting of factorizations was an instance of an operation that happened to be easy to describe, but not particularly easy to understand, which we were able to analyze completely with our new algebraic methods.

But, as sometimes happens when you are lucky, we subsequently found a very good reason to be interested in these matters. This came from computational complexity.

A *circuit* with n inputs is a directed acyclic graph, and in our circuits we shall require that there be a single sink node. Each source node is labeled by a variable x_i or its negation $\neg x_i$, where $1 \leq i \leq n$, and each non-source node of in-degree r by a function $f : \{0,1\}^r \to \{0,1\}$. Initially we shall just use the r-ary AND and OR functions, corresponding to standard logic gates, but later we shall play around with the gate type.

The circuit computes as follows: Given a bit string $a_1 \cdots a_n$, place a_i at each source node labeled x_i, $\neg a_i$ at each source node labeled $\neg x_i$, and recursively compute a bit value for each non-source node: If the entering edges of a node labeled f connect to nodes with bit values $b_1, \ldots, b_r$, then the node will get the value $f(b_1, \ldots, b_r)$. (In all of our examples the gate functions f are symmetric, so we needn't worry about ordering the incoming edges to a node.) The input is accepted if the bit value assigned to the sink node is 1, and rejected otherwise.

A circuit family with one circuit for each positive input length n thus recognizes a language $L \subseteq \{0,1\}^*$. If the circuits in the family contain only AND and OR gates, the depth of the circuits in the family (the length of the longest path from an input to the sink) is bounded by a constant, and the size (the number of nodes) of the nth circuit in the family is bounded by n^k for some constant k, then the language is said to belong to the class $\mathbf{AC}^0$.

$\mathbf{AC}^0$ contains the set of all strings of the form uv, where $|u| = |v|$ and the integer with binary representation u is greater than the integer with binary representation v. Indeed, we can write this circuit explicitly as

$$\bigvee_{j=1}^{n}\Big(\bigwedge_{i<j}\big((x_i \wedge x_{i+n}) \vee (\neg x_i \wedge \neg x_{i+n}) \wedge x_j \wedge \neg x_{j+n}\big)\Big).$$

If we were to allow multiple outputs, then we could use the same strategy to perform binary addition of two n-bit numbers in depth 3 and size $n^{O(1)}$. $\mathbf{AC}^0$ contains every star-free regular language in $\{0,1\}^*$, and in fact every star-free regular language over any finite alphabet Σ, provided we adopt a fixed-length encoding of letters of Σ by bit strings.

Let us contrast $\mathbf{AC}^0$ with another circuit complexity class, this one called $\mathbf{NC}^1$. $\mathbf{NC}^1$ also consists of polynomial-size families of circuits with AND and OR gates, but we allow the depth of the circuits to grow logarithmically (*i.e.,* the depth of the n^{th} circuit is $O(\log n)$) and we require every node to have in-degree 2. $\mathbf{NC}^1$ contains every regular language. If we were to allow multiple outputs, then we could multiply two n-bit numbers or add n n-bit numbers [5] and even *multiply* n n-bit numbers and perform integer division [3].

The natural question in computational complexity is whether one model is really computationally more powerful than another. It is easy to see that $\mathbf{AC}^0$ is contained in $\mathbf{NC}^1$. Can we really do more with logarithmic-depth circuits?

Furst, Saxe and Sipser [6] showed that, indeed, the PARITY language, consisting of all bit strings with an even number of 1's, requires superpolynomial-size circuit families of constant depth, and thus is not in $\mathbf{AC}^0$. The same argument shows that for any $q > 1$, the set of bit strings in which the number of 1's is divisible by q is not in $\mathbf{AC}^0$, and a reduction argument shows that we cannot do such things as multiply two integers in multiple-output $\mathbf{AC}^0$.

We can try to boost the power of the constant-depth model by adding things like PARITY as a kind of oracle gate. More formally, we let $q > 1$ and consider the functions $f_r : \{0,1\}^r \to \{0,1\}$ where $f_r(a_1, \ldots, a_r) = 1$ if and only if $a_1 + \cdots + a_r$ is divisible by q. We call such a function a MOD_q *gate.* $\mathbf{ACC}^0(q)$ is the family of languages recognized by constant-depth polynomial-size families of circuits that include AND, OR and MOD_q gates. $\mathbf{ACC}^0$ is the union of the classes $\mathbf{ACC}^0(q)$ over all $q > 0$.

The definitive result on $\mathbf{ACC}^0$ is the following theorem of Smolensky [17], which contains the result of Furst-Saxe and Sipser as a special case.

Theorem 2.1. Let p and q be distinct primes, and $k > 0$. The the set L_p of bit strings in which the number of 1's is divisible by p is not in $\mathbf{ACC}^0(q)$.

But that's not really definitive enough! It tells us that we cannot count, say, modulo 7 in $\mathbf{ACC}^0(8)$, or $\mathbf{ACC}^0(25)$, but tells us nothing about whether we can do this in $\mathbf{ACC}^0(6)$, because 6 has two distinct prime factors. We *expect* that it cannot be done in $\mathbf{ACC}^0(6)$, and more generally:

Conjecture 2.2.
(a) Let $q > 1$. If p is a prime that does not divide q, then $\mathbf{ACC}^0(q)$ does not contain L_p.
(b) $\mathbf{ACC}^0$ is properly contained in $\mathbf{NC}^1$.

We know very little about what occurs when the modulus of the modular gates is not a prime power. Not only has this problem remained unsolved for twenty years, but it stands, in a sense, at the very frontier of current knowledge about computational complexity. We do not know how to separate $\mathbf{NC}^1$ from $\mathbf{ACC}^0$, but we also do not know if there is a language in **LOGSPACE** that is not in $\mathbf{NC}^1$, nor a language in **P** that is not in **LOGSPACE**, nor, of course, a language in **NP** that is not in **P**. It is entirely consistent with the current state of our knowledge that $\mathbf{ACC}^0$ contains an **NP**-complete problem.

2.2 Circuits and predicate logic

There is a close connection between the constant-depth circuit families we described above, and formulas of first-order logic used to define languages, first observed by Gurevich and Lewis [7], and independently by Immerman [8].

We illustrate this with an example. Let us return to the language

$$L_{\text{comp}} = \{uv : |u| = |v|, (u)_2 > (v)_2\},$$

where $(w)_2$ denotes the integer whose binary representation is u. In the last section we gave a description of a circuit family recognizing a similar language.

If we are allowed to read u and v in parallel then we could consider the pair (u, v) as a string of length $n = |u| = |v|$ over the four-letter alphabet $\{0,1\}\times\{0,1\}$. With this interpretation, L_{comp} is a star-free regular language, defined by the first-order sentence

$$\exists z_1(Q_{(1,0)}z_1 \wedge \forall z_2((z_2 < z_1) \rightarrow Q_{(1,1)}z_2 \vee Q_{(0,0)}z_2)).$$

Of course, positions in this string encode pairs of positions in uv, and we can translate this into a sentence that talks directly about uv:

$$\begin{array}{rcl} \exists x_1 \exists y_1 (Q_1 x_1 & \wedge & Q_0 y_1 \wedge (y_1 = x_1 + n) \\ & \wedge & \forall x_2 \forall y_2 (x_2 < x_1 \wedge y_2 = x_2 + n \rightarrow \\ & & (Q_1 x_2 \wedge Q_1 y_2) \vee (Q_0 x_2 \wedge Q_1 y_2))) \end{array}$$

The result is a first-order sentence that defines the original language L_{comp}. Observe that we have had to introduce a new numerical predicate $y = x + n$ which says that x and y occupy corresponding positions in the two halves of uv.

Conversely, we can 'unroll' this first-order formula and obtain expressions for a circuit family recognizing L_{comp}. These will be much like the ones that we saw in the last section.

This sort of argument works in general: if we denote by $\mathcal{N}$ the family of all numerical predicates, then $\mathbf{AC}^0$ is exactly the same as the class $\mathrm{FO}[\mathcal{N}]$ of languages defined by first-order sentences with no restriction on numerical predicates. The identical argument works if we permit modular quantifiers of modulus q in our formulas and MOD_q gates in our circuits. The details are given in Barrington *et. al.* [2].

As a result, we have:

Theorem 2.3. $\mathbf{ACC}^0(q) = (\mathrm{FO} + \mathrm{MOD}_q)[\mathcal{N}]$.

2.3 The connection with regular languages

A consequence of the theorem of Furst, Saxe and Sipser cited above, noted in [2], is that the regular languages in $\mathbf{AC}^0$ are precisely those definable by first-order sentences in which, in addition to the order relation, there are predicates $\equiv_t$ for equivalence of positions modulo t, for all positive integers t. In [19], the numerical predicates that are definable by first-order formulas in $<$ and $\equiv_t$ are called *regular numerical predicates,* since this is in fact the largest class of numerical predicates that one can introduce into sentences and still guarantee that every definable language is regular.We denote by $\mathcal{R}$ this class of numerical predicates.

The languages definable in this way are not quite star-free, since they include, in particular, the languages $(\{0,1\}^t)^*$ of strings of length divisible by t. But they are almost star-free in the sense that they are the smallest class containing the star-free languages and the languages $(\{0,1\}^t)^*$ that is closed under boolean operations and concatenation. If we combine this with the logical characterization of $\mathbf{AC}^0$, we obtain:

Theorem 2.4. The family of regular languages in $\mathrm{FO}[\mathcal{N}]$ is $\mathrm{FO}[\mathcal{R}]$.

It is therefore reasonable to conjecture

Conjecture 2.5. Let $q > 0$. The family of regular languages in $(\mathrm{FO} + \mathrm{MOD}_q)[\mathcal{N}]$ is $(\mathrm{FO} + \mathrm{MOD}_q)[\mathcal{R}]$.

In fact, this is equivalent to our previous Conjecture 2.2. The principal reason for this equivalence is the fact, discovered by Barrington [1], that languages whose syntactic monoids contain a nonsolvable group are complete for $\mathbf{NC}^1$ under a particularly restrictive kind of reduction: a consequence is that as soon as $(\mathrm{FO} + \mathrm{MOD}_q)[\mathcal{N}]$ contains such a regular language, it contains all of $\mathbf{NC}^1$.

We have thus reduced our conjectured solution to one of the outstanding open problems in computational complexity to a purely model-theoretic question about the definability of regular languages in an extension of first-order logic. It makes sense to look for a model-theoretic explanation of the phenomenon. Unfortunately, the only proof we that we possess for the pure first-order case, Theorem 2.4 requires the lower bounds results from circuit complexity. And, as we have already remarked, none of the methods for proving these bounds generalizes to treat $\mathbf{ACC}^0$.

There has been some small progress on the question. Roy and Straubing [15] use model-theoretic collapse results to prove Conjecture 2.5 when the only numerical predicate allowed is the addition of positions. They also show the conjecture holds for sentences that contain only the order relation and arbitrary monadic numerical predicates. However, as they discuss, there are fundamental obstacles to generalizing these methods.

3 Sentences with a bounded number of variables

3.1 Two- and three-variable first-order sentences

An occurrence of a variable x in a sentence can lie within the scope of several different quantifiers that use this variable. It is only the innermost such quantifer that binds this occurrence of x. Thus it is possible to re-use variables within a sentence. For instance, the sentence

$$\exists x(Q_\sigma x \wedge \exists y(y < x \wedge Q_\tau y \wedge \exists x(x < y \wedge Q_\tau x \wedge \exists y(y < x \wedge Q_\sigma y))))$$

defines the set of all strings that have a subsequence $\sigma\tau\tau\sigma$.

It is known that every first-order sentence over $<$ is equivalent to one in which only three variables are used. (Kamp [10], Immerman and Kozen [9]). Thérien and Wilke showed that the languages definable by two-variable sentences could be characterized in terms of the syntactic monoid [25]: These are the languages whose syntactic monoids belong the the variety $\mathbf{DA}$. There are many equivalent definitions of this class of monoids, but here is one we shall find most useful: Two elements m and n of a monoid M are said to be $\mathcal{J}$-equivalent if $MmM = MnM$. A monoid is in $\mathbf{DA}$ if it is aperiodic, and if every element $\mathcal{J}$-equivalent to an idempotent is itself idempotent.

The language $(\sigma\tau)^*$ that we discussed earlier serves as a good example that separates two-variable definability from first-order definability. It is quite plausible that we cannot define this language without referring to one position being between two others, and that this will require three variables to do. The proof is that the words σ and $\sigma\tau\sigma$ represent the same elements of the syntactic monoid of this language, and so σ and $\sigma\tau$ are $\mathcal{J}$-equivalent, but the second of these is idempotent, while the first is not.

3.2 Modular quantifiers with a bounded number of variables

In [21] we investigated what happens when we bound the number of variables in sentences that contain modular quantifiers. If modular quantifiers are used exclusively, then every sentence is equivalent to one in which only two variables are used. When both modular and ordinary quantifiers are allowed, then three variables are again sufficient to define all the languages in $(\mathrm{FO}+\mathrm{MOD})[<]$. An interesting phenomenon occurs in the two-variable case. Consider again the language $(\sigma\tau)^*$ in the example above. It is defined by a sentence that says the length of the string is even, and that a position contains τ if and only if it is an odd-numbered position:

$$\exists^{0 \bmod 2} x(x = x) \wedge \forall x(Q_\tau x \leftrightarrow \exists^{0 \bmod 2} y(y < x)).$$

What is remarkable here is that modular quantifiers are not required at all to define this language, but allowing them leads to a more economical (in terms of the number of variables) specification. Furthermore, appearances to the contrary, the modulus used is irrelevant. It is possible to define the same language with two variables using modular quantifiers of modulus 3, a puzzle we leave for the reader.

Let us denote by $(\mathrm{FO}+\mathrm{MOD})^2[<]$ the family of languages in $(\mathrm{FO}+\mathrm{MOD})[<]$ definable by a two-variable sentence. We further denote by $\Sigma_2[\mathrm{MOD}]$ the family of languages defined by sentences over $<$ in which there is a block of existential quantifiers, followed by a block of universal quantifiers, followed by a formula in which only modular quantifiers appear. The family $\Pi_2[\mathrm{MOD}]$ is defined similarly. We showed:

Theorem 3.1. Let $L \subseteq \Sigma^*$. The following are equivalent

(a) $L \in (\mathrm{FO}+\mathrm{MOD})^2[<]$.

(b) $L \in \Sigma_2[\mathrm{MOD}] \cap \Pi_2[\mathrm{MOD}]$.

(c) The syntactic monoid $M(L)$ divides a wreath product $M \circ G$, where $M \in \mathbf{DA}$ and G is a solvable group. (That is, $M(L)$ belongs to the pseudovariety $\mathbf{DA} * \mathbf{G}_{\mathrm{sol}}$.)

Interestingly, we do not know how to determine effectively if a given finite monoid belongs to $\mathbf{DA} * \mathbf{G}_{\mathrm{sol}}$. The problem is equivalent to determining whether a set of partial one-to-one functions on a finite set X can be extended to a solvable permutation group on a larger set Y. We refer the reader to [21] for a discussion of this problem, as well as an apparent connection to computational complexity; and also to [20], where we give a different proof of the equivalence of (a) and (c) above, based on the block product.

On the other hand, we do possess an effective test for whether a given finite monoid M divides a wreath product of a monoid in **DA** and a finite group (which may not be solvable): If e and f are $\mathcal{J}$-equivalent idempotents of M, and ef lies in the same $\mathcal{J}$-class, then ef is itself idempotent. To see how this criterion works in an example, consider the language $L = (\sigma+\tau)^*\sigma\sigma(\sigma+\tau)^*$ of all strings over $\{\sigma, \tau\}$ in which there are two consecutive occurrences of σ. Since τ and $\tau\sigma\tau$ are equivalent in $M(L)$, as are σ and $\sigma\tau\sigma$, we conclude that σ, τ, $\sigma\tau$ and $\tau\sigma$ are in the same $\mathcal{J}$-class. Of these, all but σ are idempotent. The condition is then violated by choosing $e = \sigma\tau$ and $f = \tau\sigma$, since the product ef is equal to the non-idempotent σ. We conclude that this language requires three variables to define, even if modular quantifiers are permitted. Observe how this purely model-theoretic conclusion, which might be difficult to obtain otherwise, follows from a relatively simple calculation in the minimal automaton of L.

3.3 The placement of the modular quantifiers, and more circuit complexity

An important element in the proof of Theorem 3.1 above is a kind of normal form for two-variable sentences over $<$ containing modular quantifiers: Every sentence of $(\mathrm{FO} + \mathrm{MOD})^2[<]$ is equivalent to one in which an ordinary quantifier never appears within the scope of a modular quantifier.

We therefore should expect the expressive power of two-variable logic to decrease if we require instead that modular quantifiers not appear inside the scope of other quantifiers. Tesson and Thérien [24] showed that in this case, the syntactic monoids of the languages defined are in the pseudovariety **DO** of monoids in which every regular $\mathcal{J}$-class (*i.e.,* every $\mathcal{J}$-class that contains an idempotent) is an *orthodox semigroup*–that is, a semigroup in which the product of two idempotents is idempotent. More precisely, they show:

Theorem 3.2. A language L is definable by a two-variable sentence over $<$ in which no modular quantifier appears within the scope of an ordinary quantifier if and only if $M(L) \in$ **DO** and every group in $M(L)$ is solvable.

Furthermore, L is definable by such a sentence in which no modular quantifier appears within the scope of another quantifier if and only if $M(L) \in$ **DO** and every group in $M(L)$ is abelian.

Let us illustrate this theorem with two examples. As already noted, our canonical example $(\sigma\tau)^*$ has a syntactic monoid in which the $\mathcal{J}$-class containing the idempotents $\sigma\tau$ and $\tau\sigma$ is not a subsemigroup. Thus this language cannot be defined by a two-variable sentence in which the modular quantifiers appear outside the ordinary quantifiers.

Second, consider the language consisting of words over $\{\sigma, \tau\}$, of the form $w\tau\sigma^k$, where $k \geq 0$, and w contains an even number of occurrences of σ. This is defined by the sentence

$$\exists^{0 \bmod 2} x(Q_\sigma x \wedge \exists y(x < y \wedge Q_\tau y),$$

in which the modular quantifier appears outside the ordinary quantifier. The underlying set of the syntactic monoid M is

$$\mathbf{Z}_2 \cup (\mathbf{Z}_2 \times \mathbf{Z}_2).$$

Words of the form σ^i are map to the element $i \bmod 2$. Words of the form $w\tau\sigma^k$, where w contains j occurrences of σ are mapped to $(j \bmod 2, k \bmod 2)$. The multiplication in M is given by

$$\begin{aligned} i \cdot j &= (i + j) \bmod 2, \\ i \cdot (j, k) &= ((i + j) \bmod 2, k), \\ (j, k) \cdot i &= (j, (k + i) \bmod 2), \\ (j, k)(j', k') &= ((j + k + j') \bmod 2, k'). \end{aligned}$$

The two $\mathcal{J}$-classes have underlying sets $\mathbf{Z}_2$ itself, and $(\mathbf{Z}_2 \times \mathbf{Z}_2)$ and the idempotents are 0,(0,0) and (1,1). Observe that this monoid is itself an orthodox semigroup.

Once again, there is a connection to computational complexity: Koucky *et.al.* [11] show that the languages whose syntactic monoids are in **DO** and contain only abelian groups are precisely the regular languages recognized by $\mathbf{ACC}^0$ circuits with only a linear number of wires.

4 Conclusion

Problems about the expressive power of modular quantifiers with unrestricted numerical predicates lie at the very edge of current knowledge about computational complexity. In all likeliehood, we are a long way from solving them. We have, however, been able to apply algebraic methods to obtain a thorough understanding of what happens when we use regular numerical predicates. This has led to large array of results that are deep and interesting in their own right, and provides valuable intuition about what is probably going on in the elusive general case.

References

[1] D. A. M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[2] D. A. M. Barrington, K. J. Compton, H. Straubing, and D. Thérien. Regular languages in NC^1. *J. Comput. Syst. Sci.*, 44(3):478–499, 1992.

[3] P. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM J. Comput.*, 15(4):994–1003, 1986.

[4] J. A. Brzozowski and R. Knast. The dot-depth hierarchy of star-free languages is infinite. *J. Comput. Syst. Sci.*, 16(1):37–55, 1978.

[5] A. K. Chandra, L. J. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM J. Comput.*, 13(2):423–439, 1984.

[6] M. L. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[7] Y. Gurevich and H. R. Lewis. A logic for constant-depth circuits. *Information and Control*, 61(1):65–74, 1984.

[8] N. Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.

[9] N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Inf. Comput.*, 83(2):121–139, 1989.

[10] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Berkeley, 1968.

[11] M. Koucký, P. Pudlák, and D. Thérien. Bounded-depth circuits: separating wires from gates. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 257–265. ACM, 2005.

[12] K. Krohn and J. Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Trans. Amer. Math. Soc.*, 116:450–464, 1965.

[13] R. McNaughton and S. Papert. *Counter-free automata.* The M.I.T. Press, Cambridge, Mass.-London, 1971.

[14] J. Rhodes and B. Tilson. The kernel of monoid morphisms. *J. Pure Appl. Algebra*, 62(3):227–268, 1989.

[15] A. Roy and H. Straubing. Definability of languages by generalized first-order formulas over $(\mathbb{N},+)$. *SIAM J. Comput.*, 37(2):502–521, 2007.

[16] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[17] R. Smolensky. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *STOC*, pages 77–82. ACM, 1987.

[18] H. Straubing. A generalization of the Schützenberger product of finite monoids. *Theor. Comput. Sci.*, 13:137–150, 1981.

[19] H. Straubing. *Finite automata, formal logic, and circuit complexity.* Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, MA, 1994.

[20] H. Straubing and D. Thérien. Weakly iterated block products of finite monoids. In S. Rajsbaum, editor, *LATIN*, volume 2286 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2002.

[21] H. Straubing and D. Thérien. Regular languages defined by generalized first-order formulas with a bounded number of bound variables. *Theory Comput. Syst.*, 36(1):29–69, 2003.

[22] H. Straubing, D. Thérien, and W. Thomas. regular languages defined with generalized quantifiers. In T. Lepistö and A. Salomaa, editors, *ICALP*, volume 317 of *Lecture Notes in Computer Science*, pages 561–575. Springer, 1988.

[23] H. Straubing, D. Thérien, and W. Thomas. Regular languages defined with generalized quanifiers. *Inf. Comput.*, 118(2):289–301, 1995.

[24] P. Tesson and D. Thérien. Restricted two-variable sentences, circuits and communication complexity. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 526–538. Springer, 2005.

[25] D. Thérien and Th. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC*, pages 234–240, 1998.

[26] W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25(3):360–376, 1982.

[27] W. Thomas. An application of the Ehrenfeucht-Fraïssé game in formal language theory. *Mém. Soc. Math. France (N.S.)*, 16:11–21, 1984. Logic (Paris, 1983).

Automata: from logics to algorithms*

Moshe Y. Vardi[1]
Thomas Wilke[2]

[1] Department of Computer Science
Rice University
6199 S. Main Street
Houston, TX 77005-1892, U.S.A.
`vardi@cs.rice.edu`

[2] Institut für Informatik
Christian-Albrechts-Universität zu Kiel
Christian-Albrechts-Platz 4
24118 Kiel, Germany
`wilke@ti.informatik.uni-kiel.de`

Abstract

We review, in a unified framework, translations from five different logics—monadic second-order logic of one and two successors (S1S and S2S), linear-time temporal logic (LTL), computation tree logic (CTL), and modal μ-calculus (MC)—into appropriate models of finite-state automata on infinite words or infinite trees. Together with emptiness-testing algorithms for these models of automata, this yields decision procedures for these logics. The translations are presented in a modular fashion and in a way such that optimal complexity bounds for satisfiability, conformance (model checking), and realizability are obtained for all logics.

1 Introduction

In his seminal 1962 paper [17], Büchi states: "Our results [...] may therefore be viewed as an application of the theory of finite automata to logic." He was referring to the fact that he had proved the decidability of the monadic-second order theory of the natural numbers with successor function by translating formulas into finite automata, following earlier work by himself [16], Elgot [35], and Trakthenbrot [122]. Ever since, the approach these pioneers were following has been applied successfully in many different contexts and emerged as a major paradigm. It has not only brought about a number of decision procedures for mathematical theories, for instance, for the monadic second-order theory of the full binary tree [100],

* We are grateful to Detlef Kähler, Christof Löding, Oliver Matz, Damian Niwiński, and Igor Walukiewicz for comments on drafts of this paper.

but also efficient algorithms for problems in verification, such as a highly useful algorithm for LTL model checking [124].

The "automata-theoretic paradigm" has been extended and refined in various aspects over a period of more than 40 years. On the one hand, the paradigm has led to a wide spectrum of different models of automata, specifically tailored to match the distinctive features of the logics in question, on the other hand, it has become apparent that there are certain automata-theoretic constructions and notions, such as determinization of automata on infinite words [85], alternation [90], and games of infinite duration [12, 54], which form the core of the paradigm.

The automata-theoretic paradigm is a common thread that goes through many of Wolfgang Thomas's scientific works. In particular, he has written two influential survey papers on this topic [118, 120].

In this paper, we review translations from five fundamental logics, monadic second-order logic of one successor function (S1S), monadic second-order logic of two successor functions (S2S), linear-time temporal logic (LTL), computation tree logic (CTL), and the modal μ-calculus (MC) into appropriate models of automata. At the same time, we use these translations to present some of the core constructions and notions in a unified framework. While adhering, more or less, to the chronological order as far as the logics are concerned, we provide modern translations from the logics into appropriate automata. We attach importance to present the translations in a modular fashion, making the individual steps as simple as possible. We also show how the classical results on S1S and S2S can be used to derive first decidability results for the three other logics, LTL, CTL, and MC, but the focus is on how more refined techniques can be used to obtain good complexity results.

While this paper focuses on the translations from logics into automata, we refer the reader to the excellent surveys [118, 120] and the books [52, 96] for the larger picture of automata and logics on infinite objects and the connection with games of infinite duration.

Basic notation and terminology

Numbers. In this paper, the set of natural numbers is denoted ω, and each natural number stands for the set of its predecessors, that is, $n = \{0, \dots, n-1\}$.

Words. An alphabet is a nonempty finite set, a word over an alphabet A is a function $n \to A$ where $n \in \omega$ for a finite word and $n = \omega$ for an infinite word. When $u \colon n \to A$ is a word, then n is called its length and denoted $|u|$, and, for every $i < n$, the value $u(i)$ is the letter of u in position i. The set of all finite words over a given alphabet A is denoted A^*, the set of all infinite words over A is denoted A^ω, the empty word is denoted ε, and A^+

stands for $A^* \setminus \{\varepsilon\}$.

When u is a word of length n and $i, j \in \omega$ are such that $0 \leq i, j < n$, then $u[i,j] = u(i) \dots u(j)$, more precisely, $u[i,j]$ is the word u' of length $\max\{j - i + 1, 0\}$ defined by $u'(k) = u(i + k)$ for all $k < |u'|$. In the same fashion, we use the notation $u[i,j)$. When u denotes a finite, nonempty word, then we write $u(*)$ for the last letter of u, that is, when $|u| = n$, then $u(*) = u(n-1)$. Similarly, when u is finite or infinite and $i < |u|$, then $u[i,*)$ denotes the suffix of u starting at position i.

Trees. In this paper, we deal with trees in various contexts, and depending on these contexts we use different types of trees and model them in one way or another. All trees we use are directed trees, but we distinguish between trees with unordered successors and n-ary trees with named successors.

A tree with unordered siblings is, as usual, a tuple $\mathscr{T} = (V, E)$ where V is a nonempty set of vertices and $E \subseteq V \times V$ is the set of edges satisfying the usual properties. The root is denoted $\mathrm{root}(\mathscr{T})$, the set of successors of a vertex v is denoted $\mathrm{sucs}^{\mathscr{T}}(v)$, and the set of leaves is denoted $\mathrm{lvs}(\mathscr{T})$.

Let n be a positive natural number. An n-ary tree is a tuple $\mathscr{T} = (V, \mathrm{suc}_0, \dots, \mathrm{suc}_{n-1})$ where V is the set of vertices and, for every $i < n$, suc_i is the ith successor relation satisfying the condition that for every vertex there is at most one ith successor (and the other obvious conditions). Every n-ary tree is isomorphic to a tree where V is a prefix-closed nonempty subset of n^* and $\mathrm{suc}_i(v, v')$ holds for $v, v' \in V$ iff $v' = vi$. When a tree is given in this way, simply by its set of vertices, we say that the tree is given in implicit form. The full binary tree, denoted $\mathscr{T}_{\mathrm{bin}}$, is 2^* and the full ω-tree is ω^*. In some cases, we replace n in the above by an arbitrary set and speak of D-branching trees. Again, D-branching trees can be in implicit form, which means they are simply a prefix-closed subset of D^*.

A branch of a tree is a maximal path, that is, a path which starts at the root and ends in a leaf or is infinite. If an n-ary tree is given in implicit form, a branch is often denoted by its last vertex if it is finite or by the corresponding infinite word over n if it is infinite.

Given a tree $\mathscr{T}$ and a vertex v of it, the subtree rooted at v is denoted $\mathscr{T}{\downarrow}v$.

In our context, trees often have vertex labels and in some rare cases edge labels too. When L is a set of labels, then an L-labeled tree is a tree with a function l added which assigns to each vertex its label. More precisely, for trees with unordered successors, an L-labeled tree is of the form (V, E, l) where $l\colon V \to E$; an L-labeled n-ary tree is a tuple $(V, \mathrm{suc}_0, \dots, \mathrm{suc}_{n-1}, l)$ where $l\colon V \to E$; an L-labeled n-ary tree in implicit form is a function $t\colon V \to L$ where $V \subseteq n^*$ is the set of vertices of the tree; an L-labeled D-branching tree in implicit form is a function $t\colon V \to L$ where $V \subseteq D^*$ is the set of vertices of the tree. Occasionally, we also have more than one vertex labeling or edge labelings, which are added as other components to the tuple.

When $\mathscr{T}$ is an L-labeled tree and u is a path or branch of $\mathscr{T}$, then the labeling of u in $\mathscr{T}$, denoted $l^{\mathscr{T}}(u)$, is the word w over L of the length of u and defined by $w(i) = l(u(i))$ for all $i < |u|$.

Tuple Notation. Trees, graphs, automata and the like are typically described as tuples and denoted by calligraphic letters such as $\mathscr{T}$, $\mathscr{G}$, and so on, possibly furnished with indices or primed. The individual components are referred to by $V^{\mathscr{T}}$, $E^{\mathscr{G}}$, $E^{\mathscr{T}'}$, ... The ith component of a tuple $t = (c_0, \dots, c_{r-1})$ is denoted $\mathrm{pr}_i(t)$.

2 Monadic second-order logic of one successor

Early results on the close connection between logic and automata, such as the Büchi–Elgot–Trakhtenbrot Theorem [16, 35, 122] and Büchi's Theorem [17], center around monadic second-order logic with one successor relation (S1S) and its weak variant (WS1S). The formulas of these logics are built from atomic formulas of the form $\mathrm{suc}(x, y)$ for first-order variables x and y and $x \in X$ for a first-order variable x and a set variable (monadic second-order variable) X using boolean connectives, first-order quantification ($\exists x$), and second-order quantification for sets ($\exists X$). The two logics differ in the semantics of the set quantifiers: In WS1S quantifiers only range over finite sets rather than arbitrary sets.

S1S and WS1S can be used in different ways. First, one can think of them as logics to specify properties of the natural numbers. The formulas are interpreted in the structure with the natural numbers as universe and where suc is interpreted as the natural successor relation. The most important question raised in this context is:

Validity. Is the (weak) monadic second-order theory of the natural numbers with successor relation decidable? (Is a given sentence valid in the natural numbers with successor relation?)

A slightly more general question is:

Satisfiability. Is it decidable whether a given (W)S1S formula is satisfiable in the natural numbers?

This is more general in the sense that a positive answer for closed formulas only already implies a positive answer to the first question. Therefore, we only consider satisfiability in the following.

Second, one can think of S1S and WS1S as logics to specify the behavior of devices which get, at any moment in time, a fixed number of bits as input and produce a fixed number of bits as output (such as sequential circuits), see Figure 1. Then the formulas are interpreted in the same structure as above, but for every input bit and for every output bit there will be exactly

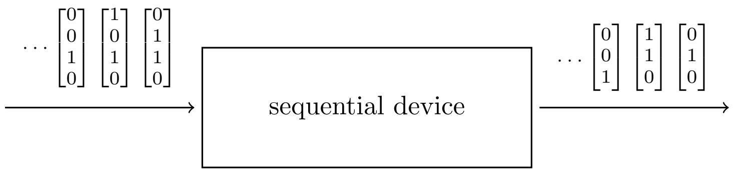


FIGURE 1. Sequential device

one free set variable representing the moments in time where the respective bit is true. (The domain of time is assumed discrete; it is identified with the natural numbers.) A formula will then be true for certain input-output pairs—coded as variable assignments—and false for the others.

For instance, when we want to specify that for a given device with one input bit, represented by the set variable X, and one output bit, represented by Y, it is the case that for every other moment in time where the input bit is true the output bit is true in the subsequent moment in time, we can use the following formula:

$$\exists Z(\text{“}Z \text{ contains every other position where } X \text{ is true”} \wedge \\ \forall x(x \in Z \rightarrow \text{“the successor of } x \text{ belongs to } Y\text{”})).$$

That the successor of x belongs to Y is expressed by $\forall y(\mathrm{suc}(x, y) \rightarrow y \in Y)$. That Z contains every other position where X is true is expressed by the conjunction of the three following conditions, where we assume, for the moment, that the “less than” relation on the natural numbers is available:

- Z is a subset of X, which can be stated as $\forall x(x \in Z \rightarrow x \in X)$,
- If X is nonempty, then the smallest element of X does not belong to Z, which can be stated as $\forall x(x \in X \wedge \forall y(y < x \rightarrow \neg y \in X) \rightarrow \neg x \in Z)$.
- For all $x, y \in X$ such that $x < y$ and such that there is no element of X in between, either x or y belongs to Z, which can be stated as

$$\forall x \forall y(x \in X \wedge y \in X \wedge x < y \wedge \\ \forall z(x < z \wedge z < y \rightarrow \neg z \in X) \rightarrow (x \in Z \leftrightarrow \neg y \in Z)).$$

To conclude the example, we need a formula that specifies that x is less than y. To this end, we express that y belongs to a set which does not contain x but with each element its successor:

$$\exists X(\neg x \in X \wedge \forall z \forall z'(z \in X \wedge \mathrm{suc}(z, z') \rightarrow z' \in X) \wedge y \in X).$$

The most important questions that are raised with regard to this usage of (W)S1S are:

Conformance. Is it decidable whether the input-output relation of a given device satisfies a given formula?

Realizability. Is it decidable whether for a given formula there exists a device with an input-output relation satisfying the formula (and if so, can a description of such a device be produced effectively)?

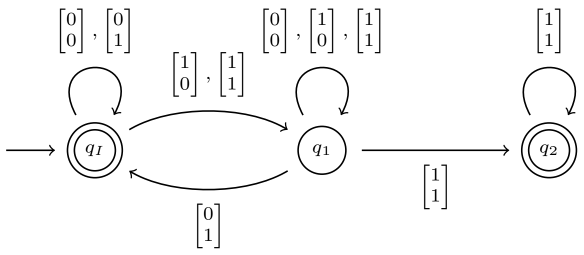
Obviously, it is important what is understood by "device". For instance, Church, when he defined realizability in 1957 [23], was interested in boolean circuits. We interpret device as "finite-state device", which, on a certain level of abstraction, is the same as a boolean circuit.

In this section, we first describe Büchi's Theorem (Section 2.1), from which we can conclude that the first two questions, satisfiability and conformance, have a positive answer. The proof of Büchi's Theorem is not very difficult except for a result about complementing a certain type of automaton model for infinite words, which we then establish (Section 2.2). After that we prove a result about determinization of the same type of automaton model (Section 2.3), which serves as the basis for showing that realizability is decidable, too. The other ingredient of this proof, certain games of infinite duration, are then presented, and finally the proof itself is given (Section 2.4).

2.1 Büchi's Theorem

The connection of S1S and WS1S to automata theory, more precisely, to the theory of formal languages, is established via a simple observation. Assume that φ is a formula such that all free variables are set variables among $V_0, \dots, V_{m-1}$, which we henceforth denote by $\varphi = \varphi(V_0, \dots, V_{m-1})$. Then the infinite words over $[2]_m$, the set of all column vectors of height m with entries from $\{0,1\}$, correspond in a one-to-one fashion to the variable assignments $\alpha\colon \{V_0, \dots, V_{m-1}\} \to 2^\omega$, where 2^M stands for the power set of any set M. More precisely, for every infinite word $u \in [2]_m^\omega$ let α_u be the variable assignment defined by $\alpha_u(V_j) = \{i < \omega\colon u(i)_{[j]} = 1\}$, where, for every $a \in [2]_m$, the expression $a_{[j]}$ denotes entry j of a. Then α_u ranges over all variable assignments as u ranges over all words in $[2]_m^\omega$. As a consequence, we use $u \models \varphi$, or, when "weak quantification" (only finite sets are considered) is used, $u \models^{\mathrm{w}} \varphi$ rather than traditional notation such as $\mathfrak{N}, \alpha \models \varphi$ (where $\mathfrak{N}$ stands for the structure of the natural numbers). Further, when φ is a formula as above, we define two formal languages of infinite words depending on the type of quantification used:

$$\mathscr{L}(\varphi) = \{u \in [2]_m^\omega\colon u \models \varphi\}, \qquad \mathscr{L}^{\mathrm{w}}(\varphi) = \{u \in [2]_m^\omega\colon u \models^{\mathrm{w}} \varphi\}.$$


q_I
q_1
q_2
$\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$

The initial state has an incoming edge without origin; final states are shown as double circles.

FIGURE 2. Example for a Büchi automaton

We say that φ defines the language $\mathscr{L}(\varphi)$ and weakly defines the language $\mathscr{L}^{\mathrm{w}}(\varphi)$. Note that, for simplicity, the parameter m is not referred to in our notation.

Büchi's Theorem states that the above languages can be recognized by an appropriate generalization of finite-state automata to infinite words, which we introduce next. A Büchi automaton is a tuple

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

where A is an alphabet, Q is a finite set of states, $Q_I \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times A \times Q$ is a set of transitions of $\mathscr{A}$, also called its transition relation, and $F \subseteq Q$ is a set of final states of $\mathscr{A}$. An infinite word $u \in A^\omega$ is accepted by $\mathscr{A}$ if there exists an infinite word $r \in Q^\omega$ such that $r(0) \in Q_I$, $(r(i), u(i), r(i+1)) \in \Delta$ for every i, and $r(i) \in F$ for infinitely many i. Such a word r is called an accepting run of $\mathscr{A}$ on u. The language recognized by $\mathscr{A}$, denoted $\mathscr{L}(\mathscr{A})$, is the set of all words accepted by $\mathscr{A}$.

For instance, the automaton in Figure 2 recognizes the language corresponding to the formula

$$\forall x(x \in V_0 \to \exists y(x < y \wedge y \in V_1)),$$

which says that every element from V_0 is eventually followed by an element from V_1. In Figure 2, q_I is the state where the automaton is not waiting for anything; q_1 is the state where the automaton is waiting for an element from V_1 to show up; q_2 is used when from some point onwards all positions belong to V_0 and V_1. Nondeterminism is used to guess that this is the case.

Büchi's Theorem can formally be stated as follows.

Theorem 2.1 (Büchi, [17]).

1. There exists an effective procedure that given a formula $\varphi = \varphi(V_0, \ldots, V_{m-1})$ outputs a Büchi automaton $\mathscr{A}$ such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\varphi)$
2. There exists an effective procedure that given a Büchi automaton $\mathscr{A}$ over an alphabet $[2]_m$ outputs a formula $\varphi = \varphi(V_0, \ldots, V_{m-1})$ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathscr{A})$.

The proof of part 2 is straightforward. The formula which needs to be constructed simply states that there exists an accepting run of $\mathscr{A}$ on the word determined by the assignment to the variables V_i. One way to construct φ is to write it as $\exists X_0 \ldots \exists X_{n-1} \psi$ where each set variable X_i corresponds exactly to one state of $\mathscr{A}$ and where ψ is a first-order formula (using $<$ in addition to suc) which states that the X_i's encode an accepting run of the automaton (the X_i's must form a partition of ω and the above requirements for an accepting run must be satisfied): 0 must belong to one of the sets X_i representing the initial states; there must be infinitely many positions belonging to sets representing final states; the states assumed at adjacent positions must be consistent with the transition relation.

The proof of part 1 is more involved, although the proof strategy is simple. The desired automaton $\mathscr{A}$ is constructed inductively, following the structure of the given formula. First-order variables, which need to be dealt with in between, are viewed as singletons. The induction base is straightforward and two of the three cases to distinguish in the inductive step are so, too: disjunction on the formula side corresponds to union on the automaton side and existential quantification corresponds to projection. For negation, however, one needs to show that the class of languages recognized by Büchi automata is closed under complementation. This is not as simple as with finite state automata, especially since deterministic Büchi automata are strictly weaker than nondeterministic ones, which means complementation cannot be done along the lines known from finite words.

In the next subsection, we describe a concrete complementation construction.

Büchi's Theorem has several implications, which all draw on the following almost obvious fact. Emptiness for Büchi automata is decidable. This is easy to see because a Büchi automaton accepts a word if and only if in its transition graph there is a path from an initial state to a strongly connected component which contains a final state. (This shows that emptiness can even be checked in linear time and in nondeterministic logarithmic space.)

Given that emptiness is decidable for Büchi automata, we can state that the first question has a positive answer:

Corollary 2.2 (Büchi, [17])**.** Satisfiability is decidable for S1S.

Proof. To check whether a given S1S formula $\varphi = \varphi(V_0, \dots, V_{m-1})$ is satisfiable one simply constructs the Büchi automaton which is guaranteed to exist by Büchi's Theorem and checks this automaton for non-emptiness. Q.E.D.

Observe that in the above corollary we use the term "satisfiability" to denote the decision problem (Given a formula, is it satisfiable?) rather than the question from the beginning of this section (Is it decidable whether ...). For convenience, we do so in the future too: When we use one of the terms satisfiability, conformance, or realizability, we refer to the corresponding decision problem.

For conformance, we first need to specify formally what is meant by a finite-state device, or, how we want to specify the input-output relation of a finite-state device. Remember that we think of a device as getting inputs from $[2]_m$ and producing outputs from $[2]_n$ for given natural numbers m and n. So it is possible to view an input-output relation as a set of infinite words over $[2]_{m+n}$. To describe an entire input-output relation of a finite-state device we simply use a nondeterministic finite-state automaton. Such an automaton is a tuple

$$\mathscr{D} = (A, S, S_I, \Delta)$$

where A is an alphabet, S is a finite set of states, $S_I \subseteq S$ is a set of initial states, and $\Delta \subseteq S \times A \times S$ is a transition relation, just as with Büchi automata. A word $u \in A^\omega$ is accepted by $\mathscr{D}$ if there exists $r \in S^\omega$ with $r(0) \in S_I$ and $(r(i), u(i), r(i+1)) \in \Delta$ for every $i < \omega$. The set of words accepted by $\mathscr{D}$, denoted $\mathscr{L}(\mathscr{D})$, is the language recognized by $\mathscr{D}$. Observe that $\mathscr{L}(\mathscr{D})$ is exactly the same as the language recognized by the Büchi automaton which is obtained from $\mathscr{D}$ by adding the set S as the set of final states.

Conformance can now be defined as follows: Given an S1S formula $\varphi = \varphi(X_0, \dots, X_{m-1}, Y_0, \dots, Y_{n-1})$ and a finite-state automaton $\mathscr{D}$ with alphabet $[2]_{m+n}$, determine whether $u \models \varphi$ for all $u \in \mathscr{L}(\mathscr{D})$.

There is a simple approach to decide conformance. We construct a Büchi automaton that accepts all words $u \in \mathscr{L}(\mathscr{D})$ which do not satisfy the given specification φ, which means we construct a Büchi automaton which recognizes $\mathscr{L}(\mathscr{D}) \cap \mathscr{L}(\neg\varphi)$, and check this automaton for emptiness. Since Büchi's Theorem tells us how to construct an automaton $\mathscr{A}$ that recognizes $\mathscr{L}(\neg\varphi)$, we only need a construction which, given a finite-state automaton $\mathscr{D}$ and a Büchi automaton $\mathscr{A}$, recognizes $\mathscr{L}(\mathscr{A}) \cap \mathscr{L}(\mathscr{D})$. The construction

The product of a Büchi automaton $\mathscr{A}$ and a finite-state automaton $\mathscr{D}$, both over the same alphabet A, is the Büchi automaton denoted $\mathscr{A} \times \mathscr{D}$ and defined by

$$\mathscr{A} \times \mathscr{D} = (A, Q \times S, Q_I \times S_I, \Delta, F \times S)$$

where

$$\Delta = \{((q,s), a, (q',s')) \colon (q,a,q') \in \Delta^{\mathscr{A}} \text{ and } (s,a,s') \in \Delta^{\mathscr{D}}\}.$$

FIGURE 3. Product of a Büchi automaton with a finite-state automaton

depicted in Figure 3, which achieves this, is a simple automata-theoretic product. Its correctness can be seen easily.

Since we already know that emptiness is decidable for Büchi automata, we obtain:

Corollary 2.3 (Büchi, [17])**.** Conformance is decidable for S1S.

From results by Stockmeyer and Meyer [112, 111], it follows that the complexity of the two problems from Corollaries 2.2 and 2.3 is nonelementary, see also [102].

Another immediate consequence of Büchi's Theorem and the proof of part 2 as sketched above is a normal form theorem for S1S formulas. Given an arbitrary S1S formula, one uses part 1 of Büchi's Theorem to turn it into an equivalent Büchi automaton and then part 2 to reconvert it to a formula. The proof of part 2 of Büchi's Theorem is designed in such a way that a formula will emerge which is of the form $\exists V_0 \ldots \exists V_{n-1} \psi$ where ψ is without second-order quantification but uses $<$. Such formulas are called existential S1S formulas.

Corollary 2.4 (Büchi-Thomas, [17, 117])**.** Every S1S formula is equivalent to an existential S1S formula, moreover, one existential set quantifier is sufficient.

To conclude this subsection we note that using the theory of finite automata on finite words only, one can prove a result weaker than Büchi's Theorem. In the statement of this theorem, automata on finite words are used instead of Büchi automata and the weak logic is used instead of the full logic. Moreover, one considers only variable assignments for the free set variables that assign finite sets only. The latter is necessary to be able to describe satisfying assignments by finite words. Such a result was obtained independently by Büchi [16], Elgot [35], and Trakhtenbrot [122], preceding Büchi's work on S1S.

2.2 Complementation of Büchi automata

Büchi's original complementation construction, more precisely, his proof of the fact that the complement of a language recognized by a Büchi automaton can also be recognized by a Büchi automaton, as given in [17], follows an algebraic approach. Given a Büchi automaton $\mathscr{A}$, he defines an equivalence relation on finite words which has

1. only a finite number of equivalence classes and
2. the crucial property that $UV^\omega \subseteq \mathscr{L}(\mathscr{A})$ or $UV^\omega \cap \mathscr{L}(\mathscr{A}) = \varnothing$ for all its equivalence classes U and V.

Here, UV^ω stands for the set of all infinite words which can be written as $uv_0v_1v_2\dots$ where $u \in U$ and $v_i \in V$ for every $i < \omega$. To complete his proof Büchi only needs to show that

(a) each set UV^ω is recognized by a Büchi automaton,

(b) every infinite word over the given alphabet belongs to such a set, and

(c) the class of languages recognized by Büchi automata is closed under union.

To prove (b), Büchi uses a weak variant of Ramsey's Theorem; (a) and (c) are easy to see. The equivalence relation Büchi defines is similar to Nerode's congruence relation. For a given word u, he considers

(i) all pairs (q, q') of states for which there exists a path from q to q' labeled u and

(ii) all pairs (q, q') where, in addition, such a path visits a final state,

and he defines two nonempty finite words to be equivalent if they agree on these pairs. If one turns Büchi's "complementation lemma" into an actual complementation construction, one arrives at a Büchi automaton of size $2^{\theta(n^2)}$ where n denotes the number of states of the given Büchi automaton.

Klarlund [65] and Kupferman and Vardi [74] describe complementation constructions along the following lines. Given a Büchi automaton $\mathscr{A}$ and a word u over the same alphabet, they consider the run DAG of $\mathscr{A}$ on u, which is a narrow DAG which contains exactly the runs of $\mathscr{A}$ on u. Vertices in this run DAG are of the form (q, i) with $q \in Q$ and $i \in \omega$ and all runs where the ith state is q visit this vertex. They show that u is not accepted by $\mathscr{A}$ if and only if the run DAG can be split into at most $2n$ alternating layers of two types where within the layers of the first type every vertex has proper descendants which are labeled with nonfinal states and where within the layers of the second type every vertex has only a finite number of descendants (which may be final or nonfinal). This can easily be used to construct a Büchi automaton for the complement: It produces the run DAG step by step, guesses for each vertex to which layer it belongs, and checks that its guesses are correct. To check the requirement for the layers of the second type, it uses the Büchi acceptance condition. The size of the resulting automaton is $2^{\theta(n \log n)}$. Optimizations lead to a construction with $(0.97n)^n$ states [46], while the best known lower bound is $(0.76n)^n$, established by Yan [131]. For practical implementations of the construction by Kupferman and Vardi, see [55].

In Section 2.2.2, we describe a complementation construction which is a byproduct of the determinization construction we explain in Section 2.3. Both constructions are based on the notion of reduced acceptance tree, introduced by Muller and Schupp [91] and described in what follows.

2.2.1 Reduced acceptance trees

Recall the notation and terminology with regard to binary trees introduced in Section 1.

Let $\mathscr{A}$ be a Büchi automaton as above, u an infinite word over the alphabet A. We consider a binary tree, denoted $\mathscr{T}_u$, which arranges all runs of $\mathscr{A}$ on u in a clever fashion, essentially carrying out a subset construction that distinguishes between final and nonfinal states, see Figure 4 for a graphical illustration.

The tree $\mathscr{T}_u$ is given as $l_u \colon V_u \to 2^Q$ in implicit form and defined inductively as follows.

(i) $\varepsilon \in V_u$ and $l_u(\varepsilon) = Q_I$.

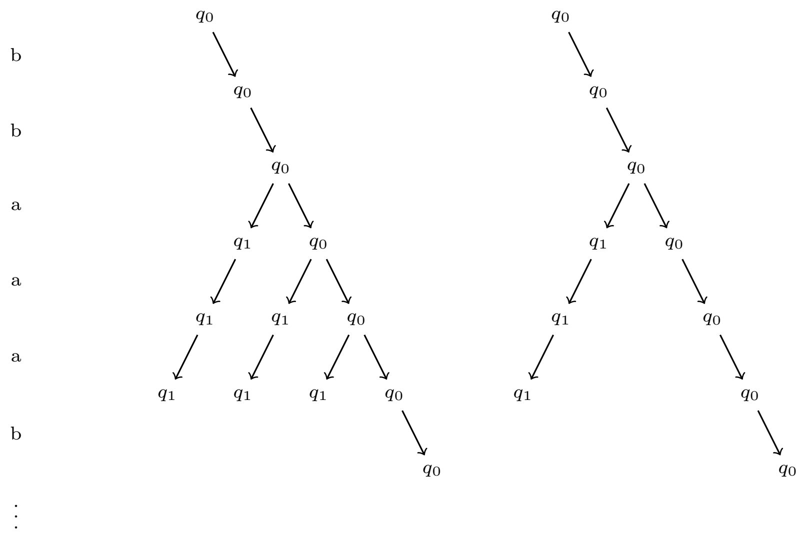
(ii) Let $v \in V_u$, $Q' = l_u(v)$, $a = u(|v|)$, and $Q'' = \bigcup\{\Delta(q,a) \colon q \in Q'\}$. Here and later, we use $\Delta(q,a)$ to denote $\{q' \in Q \colon (q,a,q') \in \Delta\}$.

- If $Q'' \cap F \neq \varnothing$, then $v0 \in V_u$ and $l_u(v0) = Q'' \cap F$.
- If $Q'' \setminus F \neq \varnothing$, then $v1 \in V_u$ and $l_u(v1) = Q'' \setminus F$.

The resulting tree is called the run tree of u with respect to $\mathscr{A}$.

A partial run of $\mathscr{A}$ on u is a word $r \in Q^+ \cup Q^\omega$ satisfying $r(0) \in Q_I$ and $(r(i), u(i), r(i+1)) \in \Delta$ for all i such that $i+1 < |r|$. A run is an infinite partial run.

Every partial run r of $\mathscr{A}$ on u determines a path b in the run tree: The length of b is $|r| - 1$ and $b(i) = 0$ if $r(i+1) \in F$ and $b(i) = 1$ otherwise, for $i < |r| - 1$. We write $r{\downarrow\downarrow}$ for this path and call it the 2-projection of r.



Depicted are the run tree and the reduced run tree of the automaton to the right for the given word. Note that in the trees the labels of the vertices should be sets of states, but for notational convenience we only list their elements.

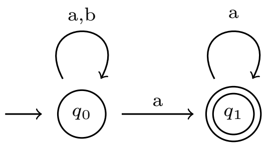


FIGURE 4. Run tree and reduced run tree

Clearly, if r is an accepting run of $\mathscr{A}$ on u, then $r{\downarrow\downarrow}$ has infinitely many left turns, where a left turn is a vertex which is a left successor. Conversely, if b is an infinite branch of $\mathscr{T}_u$, then there exists a run r of $\mathscr{A}$ on u such that $r{\downarrow\downarrow} = b$, and if b has infinitely many left turns, then r is accepting. This follows from Kőnig's lemma.

From this, we can conclude:

Remark 2.5. An infinite word u is accepted by a Büchi automaton $\mathscr{A}$ if and only if its run tree has a branch with an infinite number of left turns. We call such a branch an acceptance witness.

The tree $\mathscr{T}_u$ has two other interesting properties, which we discuss next. The first one is that $\mathscr{T}_u$ has a "left-most" acceptance witness, provided there is one at all. This acceptance witness, denoted b_u, can be constructed as follows. Inductively, assume $b_u(i)$ has already been defined for all $i < n$ in a way such that there is an acceptance witness with prefix $b' = b_u(0) \dots b_u(n-$

1). If there is an acceptance witness with prefix $b'0$, we set $b_u(n) = 0$. Otherwise, there must be an acceptance witness with prefix $b'1$, and we set $b_u(n) = 1$. Clearly, this construction results in an acceptance witness. One can easily prove that b_u is the left-most acceptance witness in the sense that it is minimal among all acceptance witnesses with respect to the lexicographical ordering (but we do not need this here).

The second interesting property says something about the states occurring to the left of b_u. We say a state q is persistent in a vertex v of a branch b of $\mathscr{T}_u$ if there is a run r of $\mathscr{A}$ on u such that $r{\downarrow\downarrow} = b$ and $q \in r(|v|)$, in other words, q is part of a run whose 2-projection contains v. A word $v \in \{0,1\}^*$ is said to be left of a word $w \in \{0,1\}^*$, denoted $v <_{\text{lft}} w$, if $|v| = |w|$ and $v <_{\text{lex}} w$, where $<_{\text{lex}}$ denotes the lexicographical ordering. The crucial property of b_u is:

Lemma 2.6. Let u be an infinite word accepted by a Büchi automaton $\mathscr{A}$, w a vertex on the left-most acceptance witness b_u, and q a state which is persistent in w on b_u. Then $q \notin l_u(v)$ for every $v \in V_u$ such that $v <_{\text{lft}} w$.

Proof. Assume that w is a vertex on b_u and that $v \in V_u$ is left of w, let $n = |v|$ $(= |w|)$. For contradiction, assume q is persistent in w on b_u and $q \in l_u(v) \cap l_u(w)$. Since $q \in l_u(v)$, we know there is a partial run r of $\mathscr{A}$ on u with $r{\downarrow\downarrow} = v$ and $r(n) = q$.

Since q is persistent in w on b_u there exists a run r' of $\mathscr{A}$ on u such that $r'{\downarrow\downarrow} = b_u$ and $r'(n) = q$. Then $r'[n,\infty)$ is an uninitialized run of $\mathscr{A}$ on $u[n,\infty)$ starting with q, where an uninitialized run is one where it is not required that the first state is the initial state. This implies that $r'' = rr'(n,\infty)$ is a run of $\mathscr{A}$ on u. Moreover, $r(i) = r''(i)$ for all $i \geq n$, which implies $r''{\downarrow\downarrow}$ is an acceptance witness, too. Let c be the longest common prefix of $r''{\downarrow\downarrow}$ and b_u. We know that $c0 \leq_{\text{prf}} r''{\downarrow\downarrow}$ and $c1 \leq_{\text{prf}} b_u$, which is a contradiction to the definition of b_u—recall that $r''{\downarrow\downarrow}$ is an acceptance witness. Q.E.D.

The above fact can be used to prune $\mathscr{T}_u$ in such a way that it has finite width, but still contains an acceptance witness if and only if u is accepted by $\mathscr{A}$. We denote the pruned tree by $\mathscr{T}'_u$, write it as $l'_u\colon V'_u \to 2^Q$), and call it the reduced acceptance tree. Informally, $\mathscr{T}'_u$ is obtained from $\mathscr{T}_u$ by keeping on each level only the first occurrence of a state, reading the level from left to right, see Figure 4. Formally, the reduced acceptance tree is inductively defined as follows.

(i) $\varepsilon \in V'_u$ and $l'_u(\varepsilon) = Q_I$.

(ii) Let $v \in V'_u$, $Q' = l'_u(v)$, $a = u(|v|)$, and $Q'' = \bigcup\{\Delta(q,a)\colon q \in Q'\}$, just as above. Assume $l'_u(w)$ has already been defined for $w <_{\text{lft}} v0$ and let $\bar{Q} = \bigcup\{l'_u(w)\colon w \in V'_u \text{ and } w <_{\text{lft}} v0\}$.

- If $Q'' \cap F \setminus \bar{Q} \neq \varnothing$, then $v0 \in V'_u$ and $l'_u(v0) = Q'' \cap F \setminus \bar{Q}$.
- If $Q'' \setminus (F \cup \bar{Q}) \neq \varnothing$, then $v1 \in V'_u$ and $l'_u(v1) = Q'' \setminus (F \cup \bar{Q})$.

As a consequence of Lemma 2.6, we have:

Corollary 2.7. Let $\mathscr{A}$ be a Büchi automaton and u an infinite word over the same alphabet. Then $u \in \mathscr{L}(\mathscr{A})$ iff $\mathscr{T}'_u$ contains an acceptance witness.

Since $\mathscr{T}'_u$ is a tree of width at most $|Q|$, it has at most $|Q|$ infinite branches. So u is not accepted by $\mathscr{A}$ if and only if there is some number n such that $b(i)$ is not a left turn for all infinite branches b of $\mathscr{T}'_u$. This fact can be used to construct a Büchi automaton for the complement language, as is shown in what follows.

2.2.2 The complementation construction

Let n be an arbitrary natural number and $v_0 <_{\text{lft}} v_1 <_{\text{lft}} \dots <_{\text{lft}} v_{r-1}$ be such that $\{v_0, \dots, v_{r-1}\} = \{v \in V'_u \colon |v| = n\}$, that is, $v_0, \dots, v_{r-1}$ is the sequence of all vertices on level n of $\mathscr{T}'_u$, from left to right. We say that $l'_u(v_0) \dots l'_u(v_{r-1})$, which is a word over the alphabet 2^Q, is slice n of $\mathscr{T}'_u$.

It is straightforward to construct slice $n+1$ from slice n, simply by applying the transition relation to each element of slice n and removing multiple occurrences of states just as with the construction of $\mathscr{T}'_u$. Suppose $Q_0 \dots Q_{r-1}$ is slice n and $a = u(n)$. Let $Q'_0, \dots, Q'_{2r-1}$ be defined by

$$Q'_{2i} = \Delta(Q_i, a) \cap F \setminus \bar{Q}_i, \qquad Q'_{2i+1} = \Delta(Q_i, a) \setminus (F \cup \bar{Q}_i),$$

where $\bar{Q}_i = \bigcup_{j<2i} Q'_j$. Further, let $j_0 < j_1 < \dots < j_{s-1}$ be such that $\{j_0, \dots, j_{s-1}\} = \{j < 2r \colon Q'_j \neq \varnothing\}$. Then $Q'_{j_0} \dots Q'_{j_{s-1}}$ is slice $n+1$ of $\mathscr{T}'_u$. This is easily seen from the definition of the reduced run tree.

We say that a tuple $U = Q_0 \dots Q_{r-1}$ is a slice over Q if $\varnothing \neq Q_i \subseteq Q$ holds for $i < r$ and if $Q_i \cap Q_j = \varnothing$ for all $i, j < r$ with $i \neq j$. The sequence $Q'_{j_0} \dots Q'_{j_{s-1}}$ from above is said to be the successor slice for U and a and is denoted by $\delta_{\text{slc}}(Q_0 \dots Q_{r-1}, a)$.

The automaton for the complement of $\mathscr{L}(\mathscr{A})$, denoted $\mathscr{A}^{\mathsf{C}}$, works as follows. First, it constructs slice after slice as it reads the given input word. We call this the initial phase. At some point, it guesses

(i) that it has reached slice n or some later slice, with n as described right after Corollary 2.7, and

(ii) which components of the slice belong to infinite branches.

The rest of its computation is called the repetition phase. During this phase it carries out the following process, called verification process, over and over again. It continues to construct slice after slice, checking that (i) the components corresponding to vertices on infinite branches all continue to the right (no left turn anymore) and (ii) the components corresponding to the other branches die out (do not continue forever). The newly emerging components corresponding to branches which branch off to the left from the vertices on the infinite branches are marked. As soon as all branches supposed to die out have died out, the process starts all over again, now with the marked components as the ones that are supposed to die out.

To be able to distinguish between components corresponding to infinite branches, branches that are supposed to die out, and newly emerging branches, the components of the slice tuples are decorated by *inf*, *die*, or *new*. Formally, a decorated slice is of the form $(Q_0 \dots Q_{r-1}, f_0 \dots f_{r-1})$ where $Q_0 \dots Q_{r-1}$ is a slice and $f_i \in \{\mathit{inf}, \mathit{die}, \mathit{new}\}$ for $i < r$. A decorated slice where $f_i \neq \mathit{die}$ for all $i < r$ is called final.

The definition of the successor of a decorated slice is slightly more involved than for ordinary slices, and such a successor may not even exist. Assume a decorated slice as above is given, let V stand for the entire slice and U for its first component (which is an ordinary slice). Let the Q'_j's and j_i's be defined as above. The successor slice of V with respect to a, denoted $\delta_{\mathrm{d}}(V, a)$, does not exist if there is some $i < r$ such that $Q'_{2i+1} = \varnothing$ and $f_i = \mathit{inf}$, because this means that a branch guessed to be infinite and without left turn dies out. In all other cases, $\delta_{\mathrm{d}}(V, a) = (\delta_{\mathrm{slc}}(U, a), f'_{j_0} \dots f'_{j_{s-1}})$ where the f'_j's are defined as follows, depending on whether the automaton is within the verification process (V is not final) or at its end (V is final):

Slice V is not final. Then $f'_{2i} = f'_{2i+1} = f_i$ for every $i < r$, except when $f_i = \mathit{inf}$. In this case, $f'_{2i} = \mathit{new}$ and $f_{2i+1} = f_i$.

Slice V is final. Then $f'_{2i} = f'_{2i+1} = \mathit{die}$ for every $i < r$, except when $f_i = \mathit{inf}$. In this case, $f'_{2i+1} = \mathit{inf}$ and $f'_{2i} = \mathit{die}$.

These choices reflect the behavior of the automaton as described above.

To describe the transition from the first to the second phase formally, assume U is a slice and $a \in A$. Let $\Delta_s(U, a)$ contain all decorated slices $(\delta_{\mathrm{slc}}(U, a), f_0 \dots f_{s-1})$ where $f_i \in \{\mathit{inf}, \mathit{die}\}$ for $i < s$. This reflects that the automaton guesses that certain branches are infinite and that the others are supposed to die out. The full construction of $\mathscr{A}^{\mathsf{C}}$ as outlined in this section is described in Figure 5. A simple upper bound on its number of states is $(3n)^n$.

Using L^{C} to denote the complement of a language, we can finally state:

Theorem 2.8. Let $\mathscr{A}$ be a Büchi automaton with n states. Then $\mathscr{A}^{\mathsf{C}}$ is a Büchi automaton with $(3n)^n$ states such that $\mathscr{L}(\mathscr{A}^{\mathsf{C}}) = \mathscr{L}(\mathscr{A})^{\mathsf{C}}$.

Let $\mathscr{A}$ be a Büchi automaton. The Büchi automaton $\mathscr{A}^{\mathsf{C}}$ is defined by

$$\mathscr{A}^{\mathsf{C}} = (A, Q^{\mathrm{s}} \cup Q^{\mathrm{d}}, Q_I, \Delta', F')$$

where the individual components are defined as follows:

$$\begin{aligned} Q^{\mathrm{s}} &= \text{set of slices over } Q, \\ Q^{\mathrm{d}} &= \text{set of decorated slices over } Q, \\ F' &= \text{set of final decorated slices over } Q, \end{aligned}$$

and where for a given $a \in A$ the following transitions belong to Δ':

- $(U, a, \delta_{\mathrm{slc}}(U, a))$ for every $U \in Q^{\mathrm{s}}$,
- (U, a, V) for every $U \in Q^{\mathrm{s}}$ and $V \in \Delta_{\mathrm{s}}(U, a)$,
- $(V, a, \delta_{\mathrm{d}}(V, a))$ for every $V \in Q^{\mathrm{d}}$, provided $\delta_{\mathrm{d}}(V, a)$ is defined.

FIGURE 5. Complementing a Büchi automaton

2.3 Determinization of Büchi automata

As noted above, determinstic Büchi automata are strictly weaker than nondeterministic ones in the sense that there are ω-languages that can be recognized by a nondeterministic Büchi automaton but by no deterministic Büchi automaton. (Following classical terminology, a Büchi automaton is called deterministic if $|Q_I| = 1$ and there is a function $\delta\colon Q \times A \to Q$ such that $\Delta = \{(q, a, \delta(q, a))\colon a \in A \wedge q \in Q\}$.) It turns out that this is due to the weakness of the Büchi acceptance condition. When a stronger acceptance condition—such as the parity condition—is used, every nondeterministic automaton can be converted into an equivalent deterministic automaton.

The determinization of Büchi automata has a long history. After a flawed construction had been published in 1963 [89], McNaughton, in 1966 [85], was the first to prove that every Büchi automaton is equivalent to a deterministic Muller automaton, a model of automata on infinite words with an acceptance condition introduced in Muller's work. In [43, 42], Emerson and Sistla described a determinization construction that worked only for a subclass of all Büchi automata. Safra [105] was the first to describe a construction which turns nondeterministic Büchi automata into equivalent deterministic Rabin automata—a model of automata on infinite words with yet another acceptance condition—which has optimal complexity in the sense that the size of the resulting automaton is $2^{\theta(n \log n)}$ and one can prove that this is also a lower bound [86]. In 1995, Muller and Schupp [91] presented a proof of Rabin's Theorem via an automata-theoretic construction which has an alternative determinization construction with a similar complexity built-in; Kähler [76] was the first to isolate this construction, see also [1]. Kähler [76] also showed that based on Emerson and Sistla's construction one can design another determinization construction for all Büchi automata which yields automata of size $2^{\theta(n \log n)}$, too. In 2006, Piterman [97] showed how Safra's construction can be adapted so as to produce a parity automaton of the same complexity.

The determinization construction described below is obtained by applying Piterman's improvement of Safra's construction to Muller and Schupp's determinization construction. We first introduce parity automata, then continue our study of the reduced acceptance tree, and finally describe the determinization construction.

2.3.1 Parity automata

A parity automaton is very similar to a Büchi automaton. The only difference is that a parity automaton has a more complex acceptance condition, where every state is assigned a natural number, called priority, and a run is accepting if the minimum priority occurring infinitely often (the limes inferior) is even. States are not just accepting or rejecting; there is a whole spectrum. For instance, when the smallest priority is even, then all states with this priority are very similar to accepting states in Büchi automata: If a run goes through these states infinitely often, then it is accepting. When, on the other hand, the smallest priority is odd, then states with this priority should be viewed as being the opposite of an accepting state in a Büchi automaton: If a run goes through these states infinitely often, the run is not accepting. So parity automata allow for a finer classification of runs with regard to acceptance and rejection.

Formally, a parity automaton is a tuple

$$\mathscr{A} = (A, Q, Q_I, \Delta, \pi)$$

where A, Q, Q_I, and Δ are as with Büchi automata, but π is a function $Q \to \omega$, which assigns to each state its priority. Given an infinite sequence r of states of this automaton, we write $\text{val}_\pi(r)$ for the limes inferior of the sequence $\pi(r(0)), \pi(r(1)), \dots$ and call it the value of the run with respect to π. Since Q is finite, the value of each run is a natural number. A run

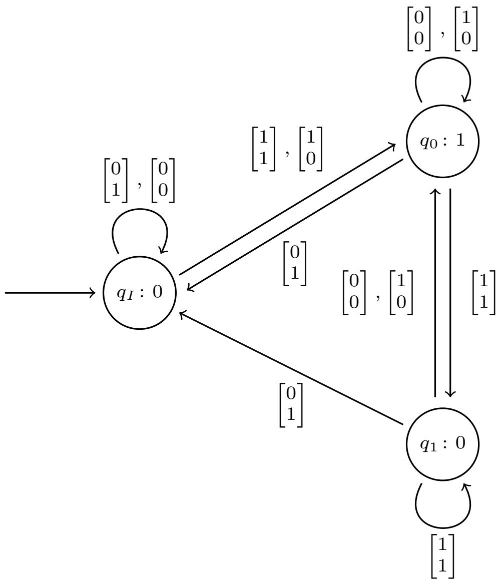

$q_I : 0$
$q_0 : 1$
$q_1 : 0$
$\begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$

FIGURE 6. Deterministic parity automaton. The values in the circles next to the names of the states are the priorities.

r of $\mathscr{A}$ is accepting if its value is even. In other words, a run r of $\mathscr{A}$ is accepting if there exists an even number v and a number k such that

(i) $\pi(r(j)) \geq v$ for all $j \geq k$ and

(ii) $\pi(r(j)) = v$ for infinitely many $j \geq k$.

Consider, for example, the parity automaton depicted in Figure 6. It recognizes the same language as the Büchi automaton in Figure 2.

As far as nondeterministic automata are concerned, Büchi automata and parity automata recognize the same languages. On the one hand, every Büchi automaton can be viewed as a parity automaton where priority 1 is assigned to every non-final state and priority 0 is assigned to every final state. (That is, the parity automaton in Figure 6 can be regarded as a deterministic Büchi automaton.) On the other hand, it is also easy to see that every language recognized by a parity automaton is recognized by some Büchi automaton: The Büchi automaton guesses a run of the parity automaton and an even value for this run and checks that it is indeed the value of the run. To this end, the Büchi automaton runs in two phases. In the first phase, it simply simulates the parity automaton. At some point, it

Let $\mathscr{A}$ be a parity automaton. The Büchi automaton A^{par} is defined by

$$A^{\text{par}} = (A, Q \cup Q \times E, Q_I, \Delta \cup \Delta', \{(q,k) \colon \pi(q) = k\})$$

where $E = \{\pi(q) \colon q \in Q \wedge \pi(q) \bmod 2 = 0\}$ and Δ' contains

- $(q, a, (q', k))$ for every $(q, a, q') \in \Delta$, provided $k \in E$, and
- $((q,k), a, (q', k))$ for every $(q, a, q') \in \Delta$, provided $\pi(q') \geq k$ and $k \in E$.

FIGURE 7. From parity to Büchi automata

concludes the first phase, guesses an even value, and enters the second phase during which it continues to simulate the parity automaton but also verifies (i) and (ii) from above. To check (i), the transition relation is restricted appropriately. To check (ii), the Büchi acceptance condition is used. This leads to the construction displayed in Figure 7. The state space has two different types of states: the states from the given Büchi automaton for the first phase and states of the form (q, k) where $q \in Q$ and k is a priority for the second phase. The priority in the second component never changes; it is the even value that the automaton guesses.

Remark 2.9. Let $\mathscr{A}$ be a parity automaton with n states and k different even priorities. Then the automaton A^{par} is an equivalent Büchi automaton with $(k+1)n$ states.

2.3.2 Approximating reduced run trees

Let $\mathscr{A}$ be a Büchi automaton as above and $u \in A^\omega$ an infinite word. The main idea of Muller and Schupp's determinization construction is that the reduced acceptance tree, $\mathscr{T}'_u$, introduced in Section 2.2.1, can be approximated by a sequence of trees which can be computed by a deterministic finite-state automaton. When these approximations are adorned with additional information, then from the sequence of the adorned approximations one can read off whether there is an acceptance witness in the reduced acceptance tree, which, by Remark 2.5, is enough to decide whether u is accepted.

For a given number n, the nth approximation of $\mathscr{T}'_u$, denoted $\mathscr{T}^n_u$, is the subgraph of $\mathscr{T}'_u$ which consists of all vertices of distance at most n from the root and which are on a branch of length at least n. Only these vertices can potentially be on an infinite branch of $\mathscr{T}'_u$. Formally, $\mathscr{T}^n_u$ is the subtree of $\mathscr{T}'_u$ consisting of all vertices $v \in V'_u$ such that there exists $w \in V'_u$ satisfying $v \leq_{\text{prf}} w$ and $|w| = n$, where $\leq_{\text{prf}}$ denotes the prefix order on words.

Note that from Lemma 2.6 we can conclude:

Remark 2.10. When u is accepted by $\mathscr{A}$, then for every n the prefix of length n of b_u is a branch of $\mathscr{T}^n_u$.

The deterministic automaton to be constructed will observe how approximations evolve over time. There is, however, the problem that, in general, approximations grow as n grows. But since every approximation has at most $|Q|$ leaves, it has at most $|Q| - 1$ internal vertices with two successors—all other internal vertices have a single successor. This means that their structure can be described by small trees of bounded size, and only their structure is important, except for some additional information of bounded size. This motivates the following definitions.

A segment of a finite tree is a maximal path where every vertex except for the last one has exactly one successor, that is, it is a sequence $v_0 \dots v_r$ such that

(i) the predecessor of v_0 has two successors or v_0 is the root,

(ii) v_i has exactly one successor for $i < r$, and

(iii) v_r has exactly two successors or is a leaf.

Then every vertex of a given finite tree belongs to exactly one segment.

A contraction of a tree is obtained by merging all vertices of a segment into one vertex. Formally, a contraction of a finite tree $\mathscr{T}$ in implicit form is a tree $\mathscr{C}$ together with a function $c\colon V^{\mathscr{T}} \to V^{\mathscr{C}}$, the contraction map, such that the following two conditions are satisfied:

(i) For all $v, w \in V^{\mathscr{T}}$, $c(v) = c(w)$ iff v and w belong to the same segment.

When p is a segment of $\mathscr{T}$ and v one of its vertices, we write $c(p)$ for $c(v)$ and we say that $c(v)$ represents p.

(ii) For all $v \in V^{\mathscr{T}}$ and $i < 2$, if $vi \in V^{\mathscr{T}}$ and $c(v) \neq c(vi)$, then $\text{suc}^{\mathscr{C}}_1(c(v), c(vi))$.

Note that this definition can easily be adapted to the case where the given tree is not in implicit form.

We want to study how approximations evolve over time. Clearly, from the nth to the $(n+1)$st approximation of $\mathscr{T}'_u$ segments can disappear, several segments can be merged into one, new segments of length one can emerge, and segments can be extended by one vertex. We reflect this in the corresponding contractions by imposing requirements on the domains of consecutive contractions.

A sequence $\mathscr{C}_0, \mathscr{C}_1, \dots$ of contractions with contraction maps $c_0, c_1, \dots$ is a contraction sequence for u if the following holds for every n:

(i) $\mathscr{C}_n$ is a contraction of the nth approximation of $\mathscr{T}'_u$.

(ii) Let p and p' be segments of $\mathscr{T}_u^n$ and $\mathscr{T}_u^{n+1}$, respectively. If p is a prefix of p' (including $p = p'$), then $c_{n+1}(p') = c_n(p)$ and p' is called an extension of p in $n+1$.

(iii) If p' is a segment of $\mathscr{T}_u^{n+1}$ which consists of vertices not belonging to $\mathscr{T}$, then $c_{n+1}(p') \notin V^{\mathscr{C}_n}$, where $V^{\mathscr{C}_n}$ denotes the set of vertices of $\mathscr{C}_n$.

Since we are interested in left turns, we introduce one further notion. Assume that p and p' are segments of $\mathscr{T}_u^n$ and $\mathscr{T}_u^{n+1}$, respectively, and p is a prefix of p', just as in (ii) above. Let p'' be such that $p' = pp''$. We say that $c_{n+1}(p')$ (which is equal to $c_n(p)$) is left extending in $n+1$ if there is a left turn in p''.

For a graphical illustration, see Figure 8.

We can now give a characterization of acceptance in terms of contraction sequences.

Lemma 2.11. Let $\mathscr{C}_0, \mathscr{C}_1, \dots$ be a contraction sequence for an infinite word u with respect to a Büchi automaton $\mathscr{A}$. Then the following are equivalent:

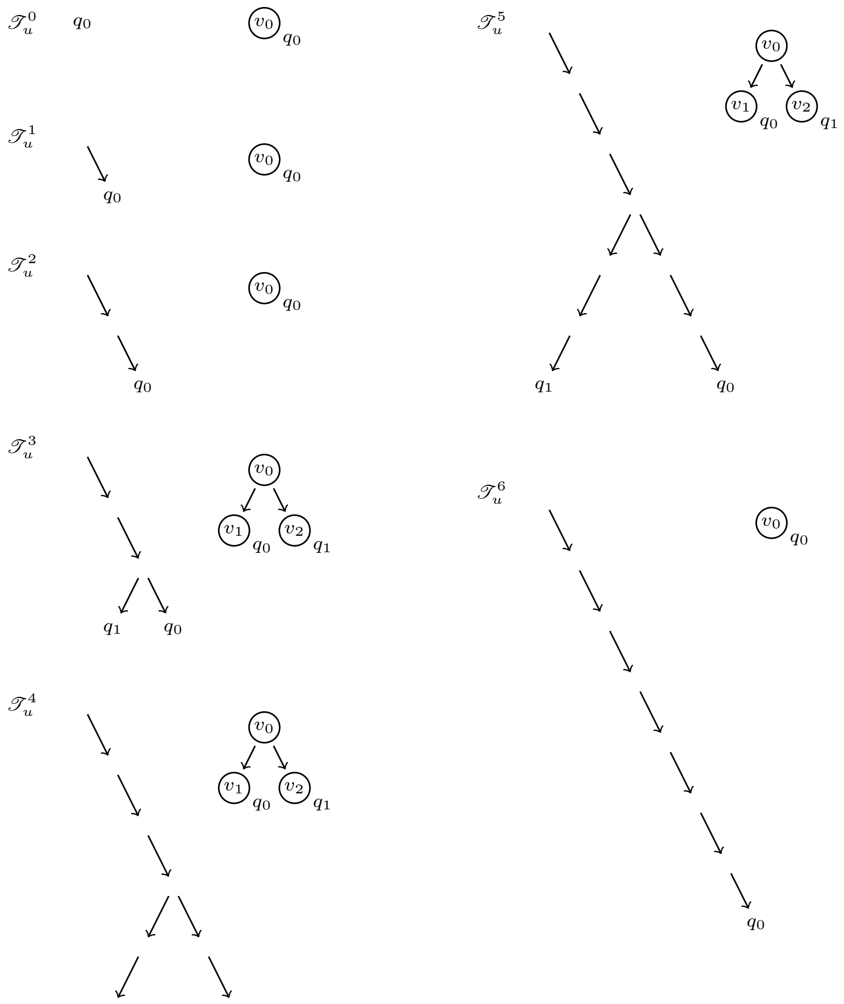
(A) $\mathscr{A}$ accepts u.

(B) There is a vertex v such that

 (a) $v \in V^{\mathscr{C}_n}$ for almost all n and

 (b) v is left extending in infinitely many n.

Proof. For the implication from (A) to (B), we start with a definition. We say that a segment p of the nth approximation is part of b_u, the left-most acceptance witness, if there are paths p_0 and p_1 such that $b_u = p_0 p p_1$. We say a vertex v represents a part of b_u if there exists i such that for all $j \geq i$ the vertex v belongs to $V^{\mathscr{C}_j}$ and the segment represented by v is part of b_u. Observe that from Remark 2.10 we can conclude that the root of $\mathscr{C}_0$ is such a vertex (where we can choose $i = 0$). Let V be the set of all vertices that represent a part of b_u and assume i is chosen such that $v \in V^{\mathscr{C}_j}$ for all $j \geq i$ and all $v \in V$. Then all elements from V form the same path in every $\mathscr{C}_j$ for $j \geq i$, say $v_0 \dots v_r$ is this path.



Depicted is the beginning of the contraction sequence for $u = bbaaab\dots$ with respect to the automaton from Figure 4. Note that, just as in Figure 4, we simply write q_i for $\{q_i\}$.

FIGURE 8. Contraction sequence

If the segment representing v_r is infinitely often extended, it will also be extended by a left turn infinitely often (because b_u is an acceptance witness), so v_r will be left extending in infinitely many i.

So assume that v_r is not extended infinitely often and let $i' \geq i$ be such that the segment represented by v_r is not extended any more for $j \geq i'$. Consider $\mathscr{C}_{i'+1}$. Let v' be the successor of v_r such that the segment represented by v' is part of b_u, which must exist because of Remark 2.10. Clearly, for the same reason, v' will be part of $V^{\mathscr{C}_j}$ for $j \geq i'+1$, hence $v' \in V$—a contradiction.

For the implication from (B) to (A), let v be a vertex as described in (B), in particular, let i be such that $v \in V^{\mathscr{C}_j}$ for all $j \geq i$. For every $j \geq i$, let p^j be the segment represented by v in $\mathscr{C}_j$. Since $p^i \leq_{\text{prf}} p^{i+1} \leq_{\text{prf}} p^{i+2} \leq_{\text{prf}} \dots$ we know there is a vertex w such that every p^j, for $j \geq i$, starts with w. Since the number of left turns on the p^j's is growing we know there is an infinite path d starting with w such that $p^j \leq_{\text{prf}} d$ for every $j \geq i$ and such that d is a path in $\mathscr{T}'_u$ with infinitely many left turns. The desired acceptance witness is then given by the concatenation of the path from the root to w, the vertex w itself excluded, and d. Q.E.D.

2.3.3 Muller–Schupp trees

The only thing which is left to do is to show that a deterministic finite-state automaton can construct a contraction sequence for a given word u and that a parity condition is strong enough to express (2.11) from Lemma 2.11. It turns out that when contractions are augmented with additional information, they can actually be used as the states of such a deterministic automaton. This leads us to the definition of Muller–Schupp trees.

Before we get to the definition of these trees, we observe that every contraction has at most $|Q|$ leaves, which means it has at most $2|Q| - 1$ vertices. From one contraction to the next in a sequence of contractions, at most $|Q|$ new leaves—and thus at most $|Q|$ new vertices—can be introduced. In other words:

Remark 2.12. For every infinite word u, there is a contraction sequence $\mathscr{C}_0, \mathscr{C}_1, \dots$ such that $V^{\mathscr{C}_i} \subseteq V$ for every i for the same set V with $3|Q|$ vertices, in particular, $V = \{0, \dots, 3|Q| - 1\}$ works.

A Muller-Schupp tree for $\mathscr{A}$ is a tuple

$$\mathscr{M} = (\mathscr{C}, l_q, l_l, R, h)$$

where

- $\mathscr{C}$ is a contraction with $V^{\mathscr{C}} \subseteq \{0, \dots, 3\,|Q| - 1\}$,
- $l_q \colon \text{lvs}(\mathscr{C}) \to 2^Q$ is a leaf labeling,

- $l_l\colon V^{\mathscr{C}} \to \{0,1,2\}$ is a left labeling,
- $R \in \{0,\dots,3|Q|-1\}^*$ is a latest appearance record, a word without multiple occurrences of letters, and
- $h \leq |R|$ is the hit number.

To understand the individual components, assume $\mathscr{C}_0, \mathscr{C}_1, \dots$ is a contraction sequence for u with $V^{\mathscr{C}_n} \subseteq \{0,\dots,3|Q|-1\}$ for every n. (Recall that Remark 2.12 guarantees that such a sequence exists.) The run of the deterministic automaton on u to be constructed will be a sequence $\mathscr{M}_0, \mathscr{M}_1, \dots$ of Muller-Schupp trees $\mathscr{M}_n = (\mathscr{C}_n, l_q^n, l_l^n, R^n, h^n)$, such that the following conditions are satisfied, where c_n denotes the contraction map for $\mathscr{C}_n$:

Leaf labeling. For every n and every leaf $v \in \operatorname{lvs}(\mathscr{T}_u^n)$, the labeling of v will be the same as the labeling of the vertex of the segment representing the segment of this leaf, that is, $l_q^n(c_n(v)) = l'_u(v)$.

Left labeling. For every n and every $v \in V^{\mathscr{C}_n}$:

(i) if v represents a segment without left turn, then $l_n(v) = 0$,

(ii) if v is left extending in n, then $l_l^n(v) = 2$, and

(iii) $l_l^n(v) = 1$ otherwise.

Clearly, this will help us to verify (b) from Lemma 2.11(2.11).

Latest appearance record. The latest appearance record R^n gives us the order in which the vertices of $\mathscr{C}_n$ have been introduced. To make this more precise, for every n and $v \in V^{\mathscr{C}_n}$, let

$$d_n(v) = \min\{i\colon v \in V^{\mathscr{C}_j} \text{ for all } j \text{ such that } i \leq j \leq n\}$$

be the date of introduction of v. Then R^n is the unique word $v_0 \dots v_{r-1}$ over $V^{\mathscr{C}_n}$ without multiple occurrences such that

- $\{v_0, \dots, v_{r-1}\} = V^{\mathscr{C}_n}$,
- either $d_n(v_j) = d_n(v_k)$ and $v_j < v_k$ or $d_n(v_j) < d_n(v_k)$, for all j and k such that $j < k < r$.

We say that $v \in V^{\mathscr{C}_n}$ has index j if $R^n(j) = v$.

Hit number. The hit number h^n gives us the number of vertices whose index has not changed. Let $R^n = v_0 \dots v_{r-1}$ as above. The value h^n is the maximum number $\leq r$ such that $d_n(v_j) < n$ for $j < h$. In other words, the hit number gives us the length of the longest prefix of R^n which is a prefix of R^{n-1}.

We need one more definition before we can state the crucial property of Muller–Schupp trees. Let $\mathscr{M}$ be any Muller–Schupp tree as above and m the minimum index of a vertex with left labeling 2 (it is left extending). If such a vertex does not exist, then, by convention, $m = n$. We define $\pi(\mathscr{M})$, the priority of $\mathscr{M}$, as follows. If $m < h$, then $\pi(\mathscr{M}) = 2m$, and else $\pi(\mathscr{M}) = 2h + 1$.

Lemma 2.13. Let $\mathscr{A}$ be a Büchi automaton, u a word over the same alphabet, and $\mathscr{M}_0, \mathscr{M}_1, \dots$ a sequence of Muller–Schupp trees satisfying the above requirements (leaf labeling, left labeling, latest appearance record, hit number). Let $p^\infty = \mathrm{val}_\pi(\mathscr{M}_0\mathscr{M}_1\dots)$, that is, the smallest value occurring infinitely often in $\pi(\mathscr{M}_0)\pi(\mathscr{M}_1)\dots$. Then the following are equivalent:

(A) $\mathscr{A}$ accepts u.

(B) p^∞ is even.

Proof. For the implication from (A) to (B), let v be a vertex as guaranteed by (B) in Lemma 2.11. There must be some n and some number i such that $v = R^n(i) = R^{n+1}(i) = \dots$ and $R^n[0,i] = R^{n+1}[0,i] = \dots$. This implies $h^j > i$ for all $j \geq n$, which means that if p^j is odd for some $j \geq n$, then $p^j > 2i$. In addition, since v is left extending for infinitely many j, we have $p^j \leq 2i$ and even for infinitely many j. Thus, p^∞ is an even value (less than or equal to $2i$).

For the implication from (B) to (A), assume that p^∞ is even and n is such that $p^j \geq p^\infty$ for all $j \geq n$. Let $n' \geq n$ be such that $p^{n'} = p^\infty$ and let v be the vertex of $\mathscr{C}_{n'}$ which gives rise to $p^{n'}$ (left extending with minimum index). Then $v \in V^{\mathscr{C}_j}$ for all $j \geq n'$ and v has the same index in all these $\mathscr{C}_j$. That is, whenever $p^j = p^\infty$ for $j \geq n'$, then v is left extending. So (B) from Lemma 2.11 is satisfied and we can conclude that u is accepted by $\mathscr{A}$. Q.E.D.

2.3.4 The determinization construction

In order to arrive at a parity automaton, we only need to convince ourselves that a deterministic automaton can produce a sequence $\mathscr{M}_0, \mathscr{M}_1, \dots$ as above. We simply describe an appropriate transition function, that is, we assume a Muller–Schupp tree $\mathscr{M}$ and a letter a are given, and we describe how $\mathscr{M}'$ is obtained from $\mathscr{M}$ such that if $\mathscr{M} = \mathscr{M}_n$ and $a = u(n)$, then $\mathscr{M}' = \mathscr{M}_{n+1}$. This is, in principle, straightforward, but it is somewhat technical. One of the issues is that during the construction of $\mathscr{M}'$ we have trees with more than $3\,|Q|$ vertices. This is why we assume that we are also given a set W of $2\,|Q|$ vertices disjoint from $\{0, \dots, 3\,|Q| - 1\}$.

A Muller-Schupp tree $\mathscr{M}'$ is called an a-successor of $\mathscr{M}$ if it is obtained from $\mathscr{M}$ by applying the following procedure.

(i) Let $V_{\text{new}} = \{0, \dots, 3\,|Q| - 1\} \setminus V^{\mathscr{C}}$.

(ii) To each leaf v, add a left and right successor from W.

Let $w_0, \dots, w_{2r-1}$ be the sequence of these successors in the order from left to right.

(iii) For $i = 0$ to $r - 1$, do:

   (a) Let v be the predecessor of w_{2i} and $Q' = l(w_0) \cup \dots \cup l(w_{2i-1})$.

   (b) Set $l_q(w_{2i}) = \Delta(l_q(v), a) \cap F \setminus Q'$ and $l_q(w_{2i+1}) = \Delta(l_q(v), a) \setminus (F \cup Q')$.

   (c) Set $l_q(w_{2i}) = 2$ and $l_q(w_{2i+1}) = 0$.

(iv) Remove the leaf labels from the old leaves, that is, make l_q undefined for the predecessors of the new leaves. Mark every leaf which has label $\varnothing$. Recursively mark every vertex whose two successors are marked. Remove all marked vertices.

(v) Replace every nontrivial segment by its first vertex, and set its left labeling to

   (a) 2 if one of the other vertices of the segment is labeled 1 or 2,

   (b) 0 if each vertex of the segment is labeled 0, and

   (c) 1 otherwise.

(vi) Replace the vertices from W by vertices from V_{new}.

(vii) Let R_0 be obtained from R by removing all vertices from $V^{\mathscr{C}} \setminus V^{\mathscr{C}'}$ from R and let R_1 be the sequence of all elements from $V^{\mathscr{C}'} \setminus V^{\mathscr{C}}$ according to the order $<$ on V. Then $R' = R_0 R_1$.

(viii) Let $h' \leq |R|$ be the maximal number such that $R(i) = R'(i)$ for all $i < h'$.

The full determinization construction is given in Figure 9. Summing up, we can state:

Theorem 2.14. (McNaughton-Safra-Piterman, [17, 105, 97]) Let $\mathscr{A}$ be a Büchi automaton with n states. Then $\mathscr{A}^{\text{det}}$ is an equivalent deterministic parity automaton with $2^{\theta(n \log n)}$ states and $2n + 1$ different priorities.

Proof. The proof of the correctness of the construction described in Figure 9 is obvious from the previous analysis. The claim about the size of the resulting automaton can be established by simple counting arguments. Q.E.D.

Let $\mathscr{A}$ be a Büchi automaton. The deterministic parity automaton $\mathscr{A}^{\text{det}}$ is defined by

$$\mathscr{A}^{\text{det}} = (A, M, \mathscr{M}_I, \delta, \pi)$$

where

- M is the set of all Muller–Schupp trees over Q,
- $\mathscr{M}_I$ is the Muller–Schupp tree with just one vertex and leaf label Q_I,
- δ is such that $\delta(\mathscr{M}, a)$ is an a-successor of $\mathscr{M}$ (as defined above), and
- π is the priority function as defined for Muller–Schupp trees.

FIGURE 9. Determinization of a Büchi automaton

The previous theorem enables us to determine the expressive power of WS1S:

Corollary 2.15. There exists an effective procedure that given an S1S formula $\varphi = \varphi(V_0, \ldots, V_{m-1})$ produces a formula ψ such that $\mathscr{L}(\varphi) = \mathscr{L}^{\text{w}}(\psi)$. In other words, every S1S formula is equivalent to a WS1S formula.

Sketch of proof. Given such a formula φ, one first uses Büchi's Theorem to construct a Büchi automaton $\mathscr{A}$ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathscr{A})$. In a second step, one converts $\mathscr{A}$ into an equivalent deterministic parity automaton $\mathscr{B}$, using the McNaughton–Safra–Piterman Theorem. The subsequent step is the crucial one. Assume $Q' = \{q_0, \ldots, q_{n-1}\}$ and, for every $u \in [2]^\omega_m$, let r_u be the (unique!) run of $\mathscr{B}$ on u. For every $i < n$, one constructs a formula $\psi_i = \psi_i(x)$ such that $u, j \models \psi_i(x)$ if and only if $r_u(j) = q_i$ for $u \in [2]^\omega_m$ and $j \in \omega$. These formulas can be built as in the proof of part 2 of Büchi's Theorem, except that one can restrict the sets X_i to elements $\leq j$, so weak quantification is enough. Finally, the formulas $\psi_i(x)$ are used to express acceptance. Q.E.D.

2.4 The Büchi–Landweber Theorem

The last problem remaining from the problems listed at the beginning of this section is realizability, also known as Church's problem [23, 24]. In our context, it can be formalized more precisely as follows.

For letters $a \in [2]_m$ and $b \in [2]_n$, we define $a^\frown b \in [2]_{m+n}$ by $(a^\frown b)_{[i]} = a_{[i]}$ for $i < m$ and $a^\frown b_{[i]} = b_{[i-m]}$ for i with $m \leq i < m+n$. Similarly, when u and v are words of the same length over $[2]_n^\infty$ and $[2]_n^\infty$, respectively, then $u^\frown v$ is the word over $[2]_{m+n}$ with the same length defined by $(u^\frown v)(i) = u(i)^\frown v(i)$ for all $i < |u|$. Realizability can now be defined as follows: Given a formula

$$\varphi = \varphi(X_0, \dots, X_{m-1}, Y_0, \dots, Y_{n-1}),$$

determine whether there is a function $f\colon [2]_m^+ \to [2]_n$ such that $u^\frown v \models \varphi$ holds for every $u \in [2]_m^\omega$ and $v \in [2]_n^\omega$ defined by $v(i) = f(u[0,i])$.

Using the traditional terminology for decision problems, we say that φ is an instance of the realizability problem, f is a solution if it has the desired property, and φ is a positive instance if it has a solution.

Observe that the function f represents the device that produces the output in Figure 1: After the device has read the sequence $a_0 \dots a_r$ of bit vectors (with m entries each), it outputs the bit vector $f(a_0 \dots a_r)$ (with n entries).

In the above definition of realizability, we do not impose any bound on the complexity of f. In principle, we allow f to be a function which is not computable. From a practical point of view, this is not very satisfying. A more realistic question is to ask for a function f which can be realized by a finite-state machine, which is a tuple

$$\mathscr{M} = (A, B, S, s_I, \delta, \lambda)$$

where A is an input alphabet, B is an output alphabet, S is a finite set of states, $s_I \in S$ the initial state, $\delta\colon S \times A \to S$ the transition function, and $\lambda\colon S \to B$ the output function. To describe the function realized by $\mathscr{M}$ we first define $\delta^*\colon A^* \to S$ by setting $\delta(\varepsilon) = s_I$ and $\delta^*(ua) = \delta(\delta^*(u), a)$ for all $u \in A^*$ and $a \in A$. The function realized by $\mathscr{M}$, denoted $f_{\mathscr{M}}$, is defined by $f_{\mathscr{M}}(u) = \lambda(\delta^*(s_I, u))$ for every $u \in A^+$.

A solution f of an instance of the realizability problem is called a finite-state solution if it is realized by a finite-state machine.

Finite-state realizability is the variant of realizability where one is interested in determining whether a finite-state solution exists. We later see that there is no difference between realizability and finite-state realizability.

Several approaches have been developed to solving realizability; we follow a game-based approach. It consists of the following steps: We first show that realizability can be viewed as a game and that solving realizability means deciding who wins this game. We then show how the games associated with instances of the realizability problem can be reduced to finite games with a standard winning objective, namely the parity winning condition. Finally, we use known results on finite games with parity winning conditions to prove the desired result.

2.4.1 Game-theoretic formulation

There is a natural way to view the realizability problem as a round-based game between two players, the environment and the (device) builder. In each round, the environment first provides the builder with an input, a vector $a \in [2]_m$, and then the builder replies with a vector $b \in [2]_n$, resulting in a combined vector $a^\frown b$. In this way, an infinite sequence of vectors is constructed, and the builder wins the play if this sequence satisfies the given S1S formula. Now, the builder has a winning strategy in this game if and only if the instance of the realizability problem we are interested in is solvable.

We make this more formal in what follows. A game is a tuple

$$\mathscr{G} = (P_0, P_1, p_I, M, \Omega)$$

where P_0 is the set of positions owned by Player 0, P_1 is the set of positions owned by Player 1 (and disjoint from P_0), $p_I \in P_0 \cup P_1$ is the initial position, $M \subseteq (P_0 \cup P_1) \times (P_0 \cup P_1)$ is the set of moves, and $\Omega \subseteq (P_0 \cup P_1)^\omega$ is the winning objective for Player 0. The union of P_0 and P_1 is the set of positions of the game and is denoted by P.

A play is simply a maximal sequence of positions which can be obtained by carrying out moves starting from the initial position, that is, it is a word $u \in P^+ \cup P^\omega$ such that $u(0) = p_I$, $(u(i), u(i+1)) \in M$ for every $i < |u|$, and if $|u| < \omega$, then there is no p such that $(u(*), p) \in M$. This can also be thought of as follows. Consider the directed graph (P, M), which is called the game graph. A play is simply a maximal path through the game graph (P, M) starting in p_I.

A play u is a win for Player 0 if $u \in \Omega \cup P^* P_1$, else it is a win for Player 1. In other words, if a player cannot move he or she loses early.

A strategy for Player α are instructions for Player α how to move in every possible situation. Formally, a strategy for Player α is a partial function $\sigma \colon P^* P_\alpha \to P$ which

(i) satisfies $(u(*), \sigma(u)) \in M$ for all $u \in \operatorname{dom}(\sigma)$ and

(ii) is defined for every $u \in P^* P_\alpha \cap p_I P^*$ satisfying $u(i+1) = \sigma(u[0,i])$ for all $i < |u| - 1$ where $u(i) \in P_\alpha$.

Observe that these conditions make sure that a strategy is defined when Player α moves according to it. A play u is consistent with a strategy σ if $u(i+1) = \sigma(u[0,i])$ for all i such that $u(i) \in P_\alpha$. A strategy σ is called a winning strategy for Player α if every play consistent with σ is a win for Player α. We then say that Player α wins the game.

Let $\varphi = \varphi(X_0, \dots, X_{m-1}, Y_0, \dots, Y_{n-1})$ be an S1S formula. The game $\mathscr{G}[\varphi]$ is defined by

$$\mathscr{G}[\varphi] = ([2]_m, \{p_I\} \cup [2]_n, p_I, M, \Omega)$$

where p_I is some initial position not contained in $[2]_m \cup [2]_n$ and

$$M = ([2]_m \times [2]_n) \cup ((\{p_I\} \cup [2]_n) \times [2]_m),$$
$$\Omega = \{p_I u_0 v_0 \dots : (u_0{}^\frown v_0)(u_1{}^\frown v_1) \dots \models \varphi\}.$$

FIGURE 10. Game for a realizability instance

The analogue of a finite-state solution is defined as follows. A strategy σ for Player α is finite-memory if there exists a finite set C, called memory, an element $m_I \in C$, the initial memory content, a function $\mu\colon C \times P \to C$, called update function, and a function $\xi\colon C \times P_\alpha \to P$ such that $\sigma(u) = \xi(\mu^*(u), u(*))$ for every $u \in \mathrm{dom}(\sigma)$, where μ^* is defined as δ^* above. That is, the moves of Player α depend on the current memory contents and the current position.

An even stronger condition than being finite-state is being memoryless. A strategy σ is memoryless if it is finite-state for a memory C which is a singleton set. As a consequence, if σ is memoryless, then $\sigma(up) = \sigma(u'p)$ for all $u, u' \in P^*$ with $up, u'p \in \mathrm{dom}(\sigma)$. So in this case, we can view a strategy as a partial function $P_\alpha \to P$. In fact, we use such functions to describe memoryless strategies.

We can now give the game-theoretic statement of the realizability problem. For an instance φ, consider the game $\mathscr{G}[\varphi]$ described in Figure 10.

Lemma 2.16. Let $\varphi = \varphi(X_0, \dots, X_{m-1}, Y_0, \dots, Y_{n-1})$ be an S1S formula. Then the following are equivalent:

(A) The instance φ of the realizability problem is solvable.

(B) Player 0 wins the game $\mathscr{G}[\varphi]$.

Moreover, φ is a positive instance of finite-state realizability if and only if Player 0 has a finite-memory winning strategy in $\mathscr{G}[\varphi]$.

Proof. For the implication from (A) to (B), let $f\colon [2]_m^+ \to [2]_n$ be the solution of an instance φ of the realizability problem. We define a partial function $\sigma\colon p_I([2]_m[2]_n)^*[2]_m \to [2]_n$ by setting $\sigma(p_I a_0 b_1 \dots b_{r-1} a_r) = f(a_0 \dots a_r)$ where $a_i \in [2]_m$ for $i \leq r$ and $b_j \in [2]_n$ for $j < r$. It is easy to see that σ is a winning strategy for Player 0 in $\mathscr{G}[\varphi]$. Conversely, a winning strategy σ for Player 0 can easily be transformed into a solution of the instance φ of the realizability problem.

To prove the additional claim, one simply needs to observe that the transformations used in the first part of the proof convert a finite-state solution into a finite-memory strategy, and vice versa. The state set of the finite-state machine used to show that a solution to the realizability problem is finite-state can be used as memory in a proof that the winning strategy constructed above is finite-memory, and vice versa. Q.E.D.

In our definition of game, there is no restriction on the winning objective Ω, but since we are interested in winning objectives specified in S1S, we focus on parity winning conditions—remember that every S1S formula can be turned into a deterministic parity automaton. It will turn out that parity conditions are particularly apt to an algorithmic treatment while being reasonably powerful.

2.4.2 Reduction to finite parity games

A winning objective Ω of a game $\mathscr{G}$ is a parity condition if there is a natural number n and a function $\pi\colon P \to n$ such that $u \in \Omega$ iff $\mathrm{val}_\pi(u) \bmod 2 = 0$ for all $u \in P^\omega$. If this is the case, we replace Ω by π and speak of a parity game.

We next show that if Ω is a winning objective and $\mathscr{A}$ a deterministic parity automaton such that $\mathscr{L}(\mathscr{A}) = \Omega$, then we can "expand" a game $\mathscr{G}$ with winning objective Ω into a parity game, simply by running $\mathscr{A}$ in parallel with the moves of the players. The respective product construction is given in Figure 11.

Lemma 2.17. Let $\mathscr{G}$ be a finite game and $\mathscr{A}$ a deterministic parity automaton such that $\mathscr{L}(\mathscr{A}) = \Omega$. Then the following are equivalent:

(A) Player 0 wins $\mathscr{G}$.

(B) Player 0 wins $\mathscr{G} \times \mathscr{A}$.

Moreover, there exists a finite-memory winning strategy for Player 0 in $\mathscr{G}$ iff there exists such a strategy in $\mathscr{G} \times \mathscr{A}$.

Proof. The proof is straightforward. We transform a winning strategy for Player 0 in $\mathscr{G}$ into a winning strategy for Player 0 in $\mathscr{G} \times \mathscr{A}$ and vice versa.

First, we define u^δ for every $u \in P^*$ to be a word of the same length where the letters are determined by $u^\delta(i) = (u(i), \delta^*(q_I, u[0,i]))$ for every $i < |u|$.

Let $\mathscr{G}$ be a game and $\mathscr{A}$ a deterministic parity automaton with alphabet P such that $\mathscr{L}(\mathscr{A}) = \Omega$. The expansion of $\mathscr{G}$ by $\mathscr{A}$ is the game

$$\mathscr{G} \times \mathscr{A} = (P_0 \times Q, P_1 \times Q, (p_I, q_I), M', \pi')$$

where

$$M' = \{((p,q),(p',\delta(q,p'))) \colon q \in Q \wedge (p,p') \in \Delta\}$$

and $\pi'((p,q)) = \pi(q)$ for all $p \in P$ and $q \in Q$.

FIGURE 11. Product of a game with a deterministic parity automaton

Let $\sigma \colon P^* P_0 \to P$ be a winning strategy for Player 0 in $\mathscr{G}$. We transform this into $\sigma' \colon (P \times Q)^*(P_0 \times Q) \to P \times Q$ by letting $\sigma'(u^\delta) = \sigma(u)$ for every $u \in \mathrm{dom}(\sigma)$. It is easy to check that this defines a strategy and that this strategy is indeed winning.

Given a winning strategy $\sigma' \colon (P \times Q)^*(P_0 \times Q) \to P \times Q$, we define a winning strategy $\sigma \colon P^* P_0 \to P$ for Player 0 simply by forgetting the second component of the positions. That is, for every u such that $u^\delta \in \mathrm{dom}(\sigma')$ we set $\sigma(u) = \sigma'(u^\delta)$. Observe that this does not lead to any ambiguities, that is, σ is well-defined, because $\mathscr{A}$ is a deterministic automaton. It is easy to check that this defines a strategy and that this strategy is indeed winning.

If we have a finite-memory strategy σ for $\mathscr{G}$, say with memory C, we can use the same memory C and a modified update function to show that σ' as defined above is finite-state. Conversely, if we have a finite-memory strategy σ', say with memory C, we can use memory $Q \times C$ to show that σ as constructed above is finite-memory, too. Q.E.D.

Corollary 2.18. Let $\varphi = \varphi(X_0, \ldots, X_{m-1}, Y_0, \ldots, Y_{n-1})$ be an instance of the realizability problem for S1S and $\mathscr{A}$ a deterministic parity automaton recognizing $\mathscr{L}(\varphi)$. Then the following are equivalent:

(A) The instance φ of the realizability problem is solvable.

(B) Player 0 wins the game $\mathscr{G} \times \mathscr{A}$.

Moreover, if Player 0 has a finite-memory winning strategy in $\mathscr{G} \times \mathscr{A}$, then φ has a finite-state solution.

Using the fact that it can be determined effectively whether Player 0 wins a finite parity game (see Theorem 2.20 below), we obtain:

Theorem 2.19 (Büchi-Landweber, [19])**.** The realizability problem is decidable for S1S.

2.4.3 Background on games

In this section, we provide background on games, which we already used to solve Church's problem and which we need in various places.

Since plays of games may be infinite, it is not at all clear whether in a given game one of the two players has a winning strategy, that is, whether the game has a winner. When this is the case one says that the game is determined. It is said to be memoryless determined if there exists a memoryless winning strategy.

Theorem 2.20 (Emerson-Jutla-Mostowski, [40, 88])**.** Every parity game is memoryless determined.

That every parity game is determined follows immediately from a result by Martin [82].

For S1S realizability it is enough to know that the winner in a parity game can be effectively determined. In a later section, we need to know more about the computational complexity of this problem, in particular, we need to know how it depends on the number of priorities occurring in a game:

Theorem 2.21 (Jurdziński, [62])**.** Every parity game with n positions, m edges, and at most d different priorities in every strongly connected component of its game graph can be decided in time $O(n + mn^{\lfloor d/2 \rfloor})$ and an appropriate memoryless winning strategy can be computed within the same time bound.

2.5 Notes

Büchi's Theorem has been the blueprint for many theorems characterizing monadic second-order logic by automata. The most important theorem to mention is Rabin's Theorem [100], which extends Büchi's Theorem to the monadic theory of two successor functions and is the subject of the next section. Other early results, besides the Büchi–Elgot–Trakthenbrot theorem and Büchi's Theorem, are a result by Büchi [18] on ordinals and a result by Doner [31] (see also Thatcher and Wright [115]), which characterizes monadic second-order logic over finite trees in terms of automata and allows to prove that the weak monadic theory of two successor relations is decidable. Later results deal, for instance, with finite and infinite traces (certain partial orders) [119, 33], see also [30], pictures (matrices with letters as entries) [51], see also [50, 83], and weighted automata [32]. In some of these cases, the proofs are much harder than for S1S and Büchi automata.

When establishing a characterization of automata in terms of monadic second-order logic, proving part 2—the description of the behavior of an automaton by a monadic second-order formula—is straightforward very often and leads to existential monadic second-order formulas, just as for S1S. The other direction—from full monadic second-order logic to automata—fails, however, for various automaton models because closure under complementation (negation) cannot be shown. In such cases, a partial result can sometimes nevertheless be obtained by showing that every existential monadic second-order formula can be translated into an automaton. This is, for instance, the case for pictures [51], see also [83].

Büchi's Theorem characterizes monadic second-order logic in terms of finite-state automata on infinite words. It is only natural to ask whether there are fragments of monadic second-order logics or other logics similar in expressive power to monadic second-order logic that can be characterized in a comparable fashion. We have already seen that the existential fragment of S1S has the same expressive power as S1S, but one can prove that first-order logic with ordering (and successor) or with successor only is strictly less expressive than S1S. The first of the two logics can be characterized just as in the case of finite words as defining exactly

(i) the star-free languages of infinite words,

(ii) the languages expressible in linear-time temporal logic, and

(iii) the languages of infinite words which are recognized by counter-free automata

(see [64, 116, 95, 132]), the second can be characterized as the weak version of locally threshold testability [117].

Ever since Büchi's seminal work automata on infinite words and formal languages of infinite words have been a major topic in research, motivated both from a mathematical and a computer science perspective. There have been many (successful) attempts to adapt the facts known from classical automata theory and the classical theory of formal languages to the setting with infinite words, for instance, regular expressions were extended to ω-regular expressions and the algebraic theory of regular languages was extended to an algebraic theory of ω-regular languages. But there are also new issues that arise for infinite words, which are essentially irrelevant for finite words. For example, the set of infinite words over a given alphabet can easily be turned into a topological space and it is interesting to study how complex languages are that can be recognized by finite-state automata.

One particularly interesting issue are the different types of acceptance conditions that are available for automata on infinite words. In our exposition, we work with Büchi and parity acceptance, but there are many more acceptance conditions which are suggested and widely used throughout the literature. The most prominent are: Streett [113], Rabin [100], and

Muller conditions [89]. An important question regarding all these different acceptance conditions is which expressive power they have, depending on whether they are used with deterministic or nondeterministic automata. It turns out that when used with nondeterministic automata all the aforementioned conditions are not more powerful than nondeterministic Büchi automata and when used with deterministic automata they are all as powerful as deterministic parity automata. In other words, each of the three conditions is just as good as the parity condition. Given McNaughton's Theorem, this is not very difficult to show. In almost all cases, asymptotically optimal conversions between the various conditions are known [105]. Recent improvements are due to Yan [131].

It is not only the type of acceptance condition that can be varied, but also the type of "mode". In this section, we have dealt with deterministic and nondeterministic automata. One can either look for

(i) modes in between or

(ii) modes beyond nondeterminism.

As examples for (i) we mention unambiguous automata [4], which are Büchi automata which admit at most one accepting run for each word, and prophetic automata [21], which are Büchi automata with the property that there is exactly one run on each word (besides partial runs that cannot be continued), be it accepting or not.

Examples for (ii) are alternating automata on infinite words, which are explained in detail in Section 4. Since they are, in principle, stronger than nondeterministic automata, they often allow for more succinct representations, which is why they have been studied extensively from a practical and complexity-theoretic point of view. Moreover, they can often be used to make automata-theoretic constructions more modular and transparent and help to classify classes of languages. For instance, the Kupferman–Vardi complementation construction for Büchi automata uses what are called weak alternating automata as an intermediate model of automaton, see also [121].

As can be seen from the Büchi–Landweber theorem, games of infinite duration are intimately connected with the theory of automata on infinite words. This becomes even more obvious as soon as alternating automata come into the picture, because they can be viewed as defining families of games in a uniform fashion. These games play a similar role in the theory of automata on infinite trees, as will be explained in the next section. Regardless of this, these games are interesting in their own right and there is an extensive literature on them. One of the major open problems is the computational complexity of finite parity games. The best upper bounds are that the problem is in UP $\cap$ co-UP, which is a result by Jurdziński [61], that it can be solved by subexponential algorithms, see, for instance, [63], and polynomial time algorithms when the underlying game graphs belong to certain restricted classes of graphs, see, for instance, [8].

3 Monadic-second order logic of two successors

Büchi's Theorem is a blueprint for Rabin's result on monadic second-order logic of two successors (S2S). The formulas of that logic are built just as S1S formulas are built, except that there are two successor relations and not only one. More precisely, while in S1S the atomic formulas are of the form $x \in X$ and $\text{suc}(x, y)$ only, in S2S the atomic formulas are of the form $x \in X$, $\text{suc}_0(x, y)$, and $\text{suc}_1(x, y)$, where $\text{suc}_0(x, y)$ and $\text{suc}_1(x, y)$ are read as "y is the left successor of x" and "y is the right successor of x", respectively. S2S formulas are interpreted in the full binary tree $\mathscr{T}_{\text{bin}}$.

As a first simple example, we design a formula with one free set variable X which holds true if and only if the set assigned to X is finite. This can be expressed by saying that on every branch there is a vertex such that the subtree rooted at this vertex does not contain any element from X. This leads to:

$$\forall Y(\text{"}Y \text{ is a branch of the binary tree"} \rightarrow \\ \exists y(y \in Y \wedge \forall z(y \leq z \rightarrow \neg z \in X))),$$

where $\leq$ is meant to denote the prefix order on the vertices of $\mathscr{T}_{\text{bin}}$. That Y is a branch of $\mathscr{T}_{\text{bin}}$ can easily be expressed as a conjunction of several simple conditions:

- Y is not empty, which can be stated as $\exists x(x \in Y)$,
- with each element of Y its predecessor (provided it exists) belongs to Y, which can be stated as $\forall x \forall y(y \in Y \wedge (\text{suc}_0(x, y) \vee \text{suc}_1(x, y)) \rightarrow x \in Y)$, and
- each element of Y has exactly one successor in Y, which can be stated as $\forall x \forall y \forall z(x \in Y \wedge \text{suc}_0(x, y) \wedge \text{suc}_1(x, z) \rightarrow (y \in Y \leftrightarrow z \notin Y))$.

To conclude the example, we define $x \leq y$ by stating that every successor-closed set containing x contains y as well:

$$\forall X(x \in X \wedge \forall z \forall z'(z \in X \wedge (\text{suc}_0(z, z') \vee \\ \text{suc}_1(z, z')) \rightarrow z' \in X) \rightarrow y \in X).$$

Observe that we have a universally quantified set variable in this formula, whereas in Section 2 we use an existentially quantified set variable to define ordering for the natural numbers. In both situations, one can use either type of quantifier.

As a second example, we consider the property that on every branch there are only finitely many elements from X. This can be specified by:

$$\forall Y(\text{“}Y \text{ is a branch of the binary tree”} \to \\ \exists y(y \in Y \wedge \forall z(x \leq z \wedge z \in Y \to \neg z \in X))),$$

using the same auxiliary formulas from above.

The most important question about S2S is whether satisfiability is decidable. A positive answer to this question implies decidability of the monadic second-order theory of the binary tree and a number of related theories as Rabin showed in his 1969 paper [100].

That satisfiability of an S2S formula is decidable can, in principle, be shown in the same way as the analogous statement for S1S: One first proves that every S2S formula can be translated into an equivalent automaton—this time a tree automaton—and then shows that emptiness for the automata involved is decidable. This is the approach that Rabin took in [100], and which we follow here, too.

3.1 Rabin's Theorem

In his original paper [100] Rabin used what we nowadays call Rabin tree automata to characterize S2S. We use the same model of tree automaton but with a simpler acceptance condition, the parity acceptance condition, which we also use in the context of S1S.

It is not clear right away how a tree automaton model should look like, but it turns out that it is reasonable to envision a tree automaton as follows. Starting in an initial state at the root of the tree the automaton splits up into two copies, one which proceeds at the left successor of the root and one which proceeds at the right successor of the root. The states which are assumed at these vertices are determined by the initial state and the label of the root. Then, following the same rules, the copy of the automaton residing in the left successor of the root splits up into two copies which proceed at the left successor of the left successor of the root and the right successor of the left successor of the root, and so on. In this way, every vertex of the tree gets assigned a state, and a tree is accepted if the state labeling of each branch satisfies the acceptance condition.

Formally, a parity tree automaton is a tuple

$$\mathscr{A} = (A, Q, q_I, \Delta, \pi),$$

where A, Q, and π are as with parity (word) automata (see Section 2.3.1), q_I is an initial state instead of a set of initial states, and Δ is a transition relation satisfying $\Delta \subseteq Q \times A \times Q \times Q$. Such an automaton runs on full A-labeled binary trees which are given implicitly. A run of $\mathscr{A}$ on a binary tree

$t\colon 2^* \to A$ is a binary tree $r\colon 2^* \to Q$ such that $(r(u), t(u), r(u0), r(u1)) \in \Delta$ for all $u \in 2^*$. It is accepting if for every infinite branch $u \in 2^\omega$ its labeling satisfies the parity condition, that is, if $\mathrm{val}_\pi(r(u(0))r(u(1))\dots) \bmod 2 = 0$.

As an example, consider the set L of all binary trees over $\{0,1\}$ with only finitely many vertices labeled 1 on each branch, which is very similar to the second property discussed above. It is straightforward to construct a parity tree automaton that recognizes L. The main idea is to use two states, q_0 and q_1, to indicate which label has just been read and to use the parity condition to check that on every path there are only finitely many vertices labeled q_1. In other words, we have $A = \{0,1\}$, $Q = \{q_I, q_0, q_1\}$, $\Delta = \{(q, a, q_a, q_a) \colon a \in A, q \in Q\}$, $\pi(q_I) = 0$, and $\pi(q_a) = a + 1$ for $a \in A$.

Rabin, in [100], proved a complete analogue of Büchi's theorem. We state Rabin's Theorem using the same notation as in the statement of Büchi's Theorem, which means, for instance, that we write $\mathscr{L}(\mathscr{A})$ for the set of all trees accepted by a parity tree automaton $\mathscr{A}$.

Theorem 3.1 (Rabin, [100])**.**

(i) There exists an effective procedure that given an S2S formula $\varphi = \varphi(V_0, \dots, V_{m-1})$ outputs a parity tree automaton $\mathscr{A}$ such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\varphi)$.

(ii) There exists an effective procedure that given a parity tree automaton $\mathscr{A}$ over an alphabet $[2]_m$ outputs a formula $\varphi = \varphi(V_0, \dots, V_{m-1})$ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathscr{A})$.

To prove part 2 one can follow the same strategy as with S1S: One simply constructs a formula that describes an accepting run of a given parity tree automaton. Proofs of part 2 of Theorem 3.1 can also be carried out as with S1S: One uses a simple induction on the structure of the formula. The induction base and all but one case to be considered in the inductive step are almost straightforward. The difficult step is—just as with Büchi automata—negation. One has to show that the complement of a tree language recognized by a parity tree automaton can be recognized by a parity tree automaton. This result, also known as Rabin's complementation lemma, can be proved in different ways. We present a proof which, in spirit, is very similar to what can be found in Büchi's [12] and Gurevich and Harrington's [54] work. At its heart, there is a game-theoretic description of acceptance (Section 3.2). The complementation construction itself has the determinization from Theorem 2.14 built in (Section 3.3).

3.2 The automaton-pathfinder game

Let $\mathscr{A}$ be a parity tree automaton as above and $t\colon 2^* \to A$ a binary tree. We consider a parity game where one can think of Player 0 as proving to

Let $\mathscr{A}$ be a parity tree automaton and $t\colon 2^* \to A$ a tree over the same alphabet. The automaton-pathfinder game for $\mathscr{A}$ and t is the parity game $\mathscr{G}[\mathscr{A}, t]$ defined by

$$\mathscr{G}[\mathscr{A}, t] = (2^* \times Q, 2^* \times \Delta, (\varepsilon, q_I), M_0 \cup M_1, \pi')$$

where

- for every word $u \in 2^*$, state $q \in Q$, and $(q, t(u), q_0, q_1) \in \Delta$, the move $((u, q), (u, (q, t(u), q_0, q_1)))$ belongs to M_0,
- for every word $u \in 2^*$, transition $(q, t(u), q_0, q_1) \in \Delta$, and $i < 2$, the move $((u, (q, t(u), q_0, q_1)), (ui, q_i))$ belongs to M_1, and
- $\pi'((u, q)) = \pi(q)$ for all $u \in 2^*$ and $q \in Q$.

FIGURE 12. Automaton-pathfinder game

Player 1 that t is accepted by $\mathscr{A}$, as follows. The game starts at the root of the tree and Player 0 suggests a transition which works at the root of the tree, which means it must start with the initial state and it must show the symbol the root is labeled with. Then Player 1 chooses the left or right successor of the root, say she chooses the left successor. Now it's Player 0's turn again. He must choose a transition which works for the left successor, which means it must start with the state chosen for the left successor in the transition chosen in the previous round and it must show the symbol the left successor is labeled with. Then Player 1 chooses one of the two successors, and so on. As the play proceeds, a sequence of transitions is constructed. Player 0 wins this play when the respective sequence of the source states of the transitions satisfies the parity condition.

The precise definition of the parity game is given in Figure 12. Observe that for convenience the priority function is only partially defined. This does not cause any problems since there is an infinite number of vertices with priorities assigned to them on every infinite path through the game graph.

Lemma 3.2 (Gurevich-Harrington, [54])**.** Let $\mathscr{A}$ be a parity tree automaton and $t\colon 2^* \to A$ a tree over the same alphabet. Then the following are equivalent:

(A) $\mathscr{A}$ accepts t.

(B) Player 0 wins $\mathscr{G}[\mathscr{A}, t]$.

Proof. For the implication from (A) to (B), we show how to convert an accepting run $r\colon 2^* \to Q$ of $\mathscr{A}$ on t into a winning strategy for Player 0 in $\mathscr{G}[\mathscr{A}, t]$. A strategy σ for Player 0 is defined on words of the form

$$u = (\varepsilon, q_0)(\varepsilon, \tau_0)(a_0, q_1)(a_0, \tau_1)(a_0 a_1, q_2) \dots (a_0 \dots a_{n-1}, q_n)$$

with $q_0 = q_I$, $q_i \in Q$ for $i \leq n$, $\tau_i \in \Delta$, and $a_i \in \{0, 1\}$ for $i < n$. For such a word u, we set $v_u = a_0 \dots a_{n-1}$. After the explanations given above on how one should think of the game, it should be clear that we set $\sigma(u) = (u, (q_n, a, q^0, q^1))$ with $q^i = r(v_u i)$ for $i < 2$. It is easy to check that this defines a winning strategy, because every play conform with σ corresponds to a branch of the run r.

Conversely, assume σ is a winning strategy for Player 0 in the above game. Then an accepting run r can be defined as follows. For every partial play u as above which is conform with σ, we set $r(v_u) = q_n$. It is straightforward to check that this defines an accepting run, because every path in r corresponds to a play of $\mathscr{G}[\mathscr{A}, t]$ conform with σ. Q.E.D.

There is a similar parity game—the emptiness game—which describes whether a given parity tree automaton accepts some tree. In this game, when Player 0 chooses a transition, he does not need to take into account any labeling; he simply needs to make sure that the transition is consistent with the previously chosen transition. The full game is described in Figure 13.

With a proof similar to the one of Lemma 3.2, one can show:

Lemma 3.3. Let $\mathscr{A}$ be a parity tree automaton. Then $\mathscr{L}(\mathscr{A}) \neq \varnothing$ if and only if Player 0 wins $\mathscr{G}_\varnothing[\mathscr{A}]$.

Taking Theorem 2.21 into account, we obtain:

Corollary 3.4 (Rabin, [100]). The emptiness problem for parity tree automata is decidable.

Rabin proved, in some sense, a stronger result, because he used tree automata with Rabin acceptance condition. As a further consequence of Lemma 3.3, taking Rabin's Theorem into account, we note:

Corollary 3.5 (Rabin, [100]). Satisfiability is decidable for S2S.

Let $\mathscr{A}$ be a parity tree automaton. The emptiness game $\mathscr{G}_\varnothing[\mathscr{A}]$ is defined by

$$\mathscr{G}_\varnothing[\mathscr{A}] = (Q, \Delta, q_I, M_0 \cup M_1, \pi)$$

where

- for $q \in Q$ and $(q, a, q_0, q_1) \in \Delta$, the move $(q, (q, a, q_0, q_1))$ belongs to M_0,
- for every $(q, a, q_0, q_1) \in \Delta$ and $i < 2$, the move $((q, a, q_0, q_1), q_i)$ belongs to M_1.

FIGURE 13. Emptiness game for a parity tree automaton

3.3 Complementation of parity tree automata

We can finally turn to the question of how to arrive at a parity tree automaton for the complement of a set of trees accepted by a given parity tree automaton. We are given a parity tree automaton $\mathscr{A}$ and we want to construct a parity tree automaton which recognizes $\mathscr{L}(\mathscr{A})^C$, where for each tree language L over some alphabet A we write L^C for the set of all trees over A which do not belong to L.

We describe the entire construction as a composition of several simpler constructions. More precisely, we first show that for every tree in the complement there exists a tree over an enhanced alphabet which witnesses its membership to the complement. The second step is to prove that the set of these witnesses can be recognized by a universal parity tree automaton. The third step consists in showing that universal parity tree automaton can be converted into (ordinary nondeterministic) parity tree automata, and the final step shows how to reduce the enhanced alphabet to the real one.

The first key step is to combine the automaton-pathfinder game with memoryless determinacy. To this end, we encode memoryless (winning) strategies for the pathfinder in trees. Observe that a memoryless strategy for the pathfinder in $\mathscr{G}[\mathscr{A}, t]$ for some automaton $\mathscr{A}$ and some tree t is simply a (partial) function $\sigma\colon 2^* \times \Delta \to 2^* \times Q$. Since, by construction of $\mathscr{G}[\mathscr{A}, t]$, we always have $\sigma(u, (q, a, q_0, q_1)) = (ui, q_i)$ for some $i < 2$, we can view such a function as a function $2^* \times \Delta \to 2$, which, in turn, can be viewed as a function $2^* \to 2^\Delta$. The latter is simply a 2^Δ-labeled tree. When we further encode the given tree t in that tree, we arrive at the following notion of complement witness.

Let $\mathscr{A}$ be a parity tree automaton and $t' \colon 2^* \to A \times 2^\Delta$ a tree. For simplicity, we write $t'(u)$ as (a_u, f_u) for every $u \in 2^*$. The tree t' is a complement witness if for every branch $u \in 2^\omega$ the following holds. If $\tau_0\tau_1 \cdots \in \Delta^\omega$ with $\tau_i = (q_i, a_{u[0,i)}, q_i^0, q_i^1)$ is such that $q_0 = q_I$ and $q_{i+1} = q_i^b$ where $b = f_{u[0,i)}(\tau_i)$ for every i, then $\mathrm{val}_\pi(q_0q_1\dots) \bmod 2 = 1$, that is, $q_0q_1\dots$ is not accepting with respect to π.

After the explanation given above, Theorem 2.20 now yields the lemma below, where we use the following notation. Given a tree $t' \colon 2^* \to A \times B$ for alphabet A and B, we write $\mathrm{pr}_0(t')$ for the tree defined by $\mathrm{pr}_0(t')(u) = \mathrm{pr}_0(t'(u))$ for every $u \in 2^*$, that is, we simply forget the second component of every label.

Lemma 3.6. Let $\mathscr{A}$ be a parity tree automaton and $t \colon 2^* \to A$ a tree over the same alphabet. Then the following are equivalent:

(A) $t \in \mathscr{L}(\mathscr{A})^C$.

(B) There is a complement witness t' for $\mathscr{A}$ such that $\mathrm{pr}_0(t') = t$. Q.E.D.

Using more notation, we can state the above lemma very concisely. First, we extend projection to tree languages, that is, given a tree language L over some alphabet $A \times B$, we write $\mathrm{pr}_0(L)$ for $\{\mathrm{pr}_0(t) \colon t \in L\}$. Second, given a parity tree automaton $\mathscr{A}$, we write $\mathscr{C}(\mathscr{A})$ for the set of all complement witnesses for $\mathscr{A}$. Then Lemma 3.6 simply states:

Remark 3.7. For every parity tree automaton $\mathscr{A}$,

$$\mathscr{L}(\mathscr{A})^{\mathsf{C}} = \mathrm{pr}_0(\mathscr{C}(\mathscr{A})).$$

So, clearly, once we have a parity tree automaton for $\mathscr{C}(\mathscr{A})$, we also have a parity tree automaton for $\mathscr{L}(\mathscr{A})^{\mathsf{C}}$, because we only need to omit the second component from the letters in the transition function to obtain the desired automaton.

It is not straightforward to find a parity tree automaton that recognizes $\mathscr{C}(\mathscr{A})$; it is much easier to show that $\mathscr{C}(\mathscr{A})$ is recognized by a universal parity tree automaton. Such an automaton is a tuple

$$\mathscr{A} = (A, Q, q_I, \Delta, \pi)$$

where A, Q, q_I, and π are as with parity tree automata and $\Delta \subseteq Q \times A \times 2 \times Q$. Let $t \colon 2^* \to A$ be a tree over A. A word $r \in Q^\omega$ is said to be a run for branch $u \in 2^\omega$ if $(r(i), t(u[0,i)), u(i), r(i+1)) \in \Delta$ for every i and $r(0) = q_I$. A tree is accepted if every $r \in Q^\omega$ which is a run for some branch satisfies the parity acceptance condition.

Let $\mathscr{A}$ be a parity tree automaton. The universal parity tree automaton $\mathscr{A}^{\mathrm{cw}}$ is defined by

$$\mathscr{A}^{\mathrm{cw}} = (A \times 2^{\Delta}, Q, q_I, \Delta', \pi + 1)$$

where $(q, (a, f), d, q') \in \Delta'$ if there exists $\tau = (q, a, q_0, q_1) \in \Delta$ such that $f(\tau) = d$ and $q_d = q'$, and where $\pi + 1$ stands for the priority function π' defined by $\pi'(q) = \pi(q) + 1$.

FIGURE 14. Universal parity tree automaton for complement witnesses

We can now rephrase Lemma 3.6 in terms of the new automaton model. We can express the complement of a tree language recognized by a parity tree automaton as the projection of a tree language recognized by a universal parity tree automaton. The latter is defined in Figure 14. Observe that the runs for the branches in this automaton correspond to the words $\tau_0\tau_1\ldots$ in the definition of complement witness.
We immediately obtain:

Remark 3.8. For every parity tree automaton $\mathscr{A}$,

$$\mathscr{C}(\mathscr{A}) = \mathscr{L}(\mathscr{A}^{\mathrm{cw}}).$$

To complete the description of the complementation procedure, we need to explain how a universal parity tree automaton can be converted into a parity tree automaton. One option for such a construction is to use McNaughton's Theorem, namely that every nondeterministic Büchi automaton can be turned into a deterministic parity automaton. The idea is that the tree automaton follows all runs of a given branch at the same time by running a deterministic word automaton in parallel.

Let Q be a finite set of states and $\pi \colon Q \to \omega$ a priority function. Let $\mathscr{Q}$ be the alphabet consisting of all binary relations over Q. Then every word $u \in \mathscr{Q}^{\omega}$ generates a set of infinite words $v \in Q^{\omega}$, denoted $\langle u \rangle$, defined by

$$\langle u \rangle = \{v \in Q^{\omega} \colon \forall i((v(i), v(i+1)) \in u(i))\},$$

and called the set of paths through u, because one can think of $\langle u \rangle$ as the set of all infinite paths through the graph which is obtained by "collating" $u(0), u(1), \ldots$. We are interested in a deterministic parity automaton $\mathscr{A}[Q, \pi]$ which checks that all paths through a given u satisfy the given parity condition, that is, which has the following property. For every $u \in \mathscr{Q}^{\omega}$,

$$u \in \mathscr{L}(\mathscr{A}[Q, \pi]) \qquad \text{iff} \qquad \forall v(v \in \langle u \rangle \to \mathrm{val}_{\pi}(v) \bmod 2 = 0). \tag{1.1}$$

Let Q be a finite set of states and $\pi\colon Q \to \omega$ a priority function. Consider the parity word automaton

$$\mathscr{B} = (\mathscr{Q}, Q, Q_I, \Delta, \pi + 1)$$

where $\Delta = \{(q, R, q')\colon (q, q') \in R\}$. Let $\mathscr{C}$ be an equivalent Büchi automaton (Figure 7) and $\mathscr{D}$ a deterministic parity automaton equivalent to $\mathscr{C}$ (Figure 9). The automaton $\mathscr{A}[Q, \pi]$ is defined by

$$\mathscr{A}[Q, \pi] = (\mathscr{Q}, Q^{\mathscr{D}}, q_I^{\mathscr{D}}, \delta^{\mathscr{D}}, \pi + 1).$$

FIGURE 15. Generic automaton for state set and priority function

Using Theorem 2.14, such an automaton, which we call a generic automaton for Q and π, can easily be constructed, as can be seen from Figure 15. Observe that, by construction,

$$u \in \mathscr{L}(\mathscr{C}) \qquad \text{iff} \qquad \exists v(v \in \langle u \rangle \wedge \mathrm{val}_\pi(v) \bmod 2 = 1),$$

for every $u \in \mathscr{Q}^\omega$. We conclude:

Remark 3.9. Let Q be a finite state set and $\pi\colon Q \to \omega$ a priority function. Then 1.1 holds for every $u \in \mathscr{Q}^\omega$.

Given the generic automaton, it is now easy to convert universal tree automata into nondeterministic ones: One only needs to run the generic automaton on all paths. This is explained in detail in Figure 16.

Lemma 3.10. Let $\mathscr{A}$ be a universal parity tree automaton. Then $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}^{\mathrm{nd}})$.

Proof. For convenience, we write $\mathscr{B}$ for $\mathscr{A}[Q^{\mathscr{A}}, \pi^{\mathscr{A}}]$.

First observe that for every $t\colon 2^* \to A$ there is exactly one run of $\mathscr{A}^{\mathrm{nd}}$ on t. This is because Δ is such that for every $s \in S$ and $a \in A$, there is exactly one transition in Δ of the form (s, a, s_0, s_1). For a given t, let r_t denote this run. So in order to determine whether a tree is accepted by $\mathscr{A}^{\mathrm{nd}}$, we only need to determine whether r_t is accepting. To this end, we consider a branch $w \in 2^\omega$ of this tree.

By construction of $\mathscr{A}^{\mathrm{nd}}$, the labeling of w in r_t is the run of $\mathscr{B}$ on $u = R_0^u R_1^u \dots$ where $R_i^u = R_{t(w[0,i)),w(i)}$. So $\langle u \rangle$ is the set of runs of $\mathscr{A}$

Let $\mathscr{A}$ be a universal parity tree automaton and assume that the generic automaton for $Q^{\mathscr{A}}$ and $\pi^{\mathscr{A}}$ is given as $\mathscr{A}[Q^{\mathscr{A}}, \pi^{\mathscr{A}}] = (\mathscr{Q}^{\mathscr{A}}, S, s_I, \delta, \pi)$. The parity tree automaton $\mathscr{A}^{\mathrm{nd}}$ is defined by

$$\mathscr{A}^{\mathrm{nd}} = (A, S, s_I, \Delta, \pi)$$

where for every $a \in A$ and $s \in S$,

$$\tau_{s,a} = (s, a, \delta(s, R_{a,0}), \delta(s, R_{a,1}))$$

with $R_{a,d} = \{(q, q') \colon (q, a, d, q') \in \Delta^{\mathscr{A}}\}$ for $d < 2$ and

$$\Delta = \{\tau_{s,a} \colon a \in A \wedge s \in S\}.$$

FIGURE 16. From universal to nondeterministic parity tree automata

on branch w. In view of Remark 3.9, this implies that w is accepting as a branch of r_t if and only if all runs of $\mathscr{A}$ on w are accepting. From this, the claim of the lemma follows immediately. Q.E.D.

This was also the last missing piece in the construction from a given parity tree automaton to a parity tree automaton for its complement:

Lemma 3.11 (Rabin, [100])**.** There is an effective procedure that turns a given parity tree automaton $\mathscr{A}$ into a parity tree automaton $\mathscr{A}^C$ that recognizes the complement of the language recognized by $\mathscr{A}$. Q.E.D.

3.4 Notes

Rabin's Theorem is important from a mathematical (logical) point of view because it is a very strong decidability result and can as such be used to show the decidability of many theories, see, for instance, Rabin's original paper [100] and the book [15]. A very specific question to ask is how one can prove that the monadic second-order (or first-order) theory of a certain structure is decidable using the fact that it is decidable for the binary tree. There is a wide spectrum of techniques that have been developed to this end and are explained in detail in [9], see also [22].

It may seem that the results proved for S1S and automata on infinite words extend to S2S and automata on infinite trees in a straightforward fashion. This is true in many respects, but there are important differences.

Most importantly, it is neither true that every tree language recognized by a parity tree automaton can be recognized by a Büchi tree automaton nor is it true that WS2S and S2S are equally expressive. There is, however, an interesting connection between Büchi tree automata and WS2S: a set of trees is definable in WS2S if and only if it is recognized by a Büchi tree automaton and its complement is so, too, which was proved by Rabin [101]. Moreover, being definable in WS2S is equivalent to being recognized by a weak alternating tree automaton [73]. It is true though that every S2S formula is equivalent to an existential S2S formula. Also note that the second formula given as example at the beginning of this section is one which cannot be recognized by a Büchi tree automaton, let alone specified in WS2S. Another noticeable difference between automata on infinite words and automata on infinite trees is that unambiguous tree automata are weaker than nondeterministic ones, which is a result due to Niwiński and Walukiewicz [94]. Its proof was recently simplified considerably by Carayol and Löding [20].

The most complicated automata-theoretic building block of our proof of Rabin's theorem is McNaughton's Theorem, the determinization of word automata. It is not clear to which extent McNaughton's Theorem is necessary for the proof of Rabin's Theorem. The proof presented here is based on a translation in the sense that for every S2S formula φ we construct an automaton $\mathscr{A}$ such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\varphi)$ and it makes full use of a determinization construction. There are other proofs, such as the one by Kupferman and Vardi [75], which do not rely on the entire construction but only on the fact that there are determinization constructions with a certain bound on the number of states. These constructions, however, yield a slightly weaker result in the sense that they only reduce S2S satisfiability to tree automaton emptiness. In the proof presented here, determinization is used to turn a universal automaton into a nondeterministic one, which could be called a de-universalization construction. It would be interesting to see if one can also go in the reverse direction, that is, whether there is a determinization construction which can be built on a de-universalization construction.

At the end of the previous section, we mentioned that topological questions are interesting in the context of infinite words and automata on infinite words. This is even more true for infinite trees, see [5].

4 Linear-time temporal logic

Although originally introduced in this context, S1S and WS1S have only very rarely been used to specify properties of (finite-state) devices (see [56] for a noticeable exception). For S2S, this is even more true; it has almost always been used to obtain decidability for logical theories as pointed out in Section 3. But the ever-increasing number of real computational devices and large-scale production lines of such devices has called for appropriate specification logics. In this section, we consider a logic that was introduced in this regard and show how it can be dealt with using automata theory, in particular, we show how specifically tailored automata can be used to obtain optimal upper bounds for problems such as satisfiability, conformance—in this context called model checking—, and realizability.

4.1 LTL and S1S

Linear-time temporal logic (LTL) is a modal logic designed to specify temporal relations between events occurring over time, designed by Kamp [64] to formally describe temporal relationships expressible in natural language and introduced into computer science by Pnueli [98] (see also the work by Burstall [13] and Kröger [69]) as an appropriate specification language for systems with nonterminating computations. Nowadays, LTL is widely spread and used in practice.

From a syntactic point of view LTL is propositional logic augmented by temporal operators. LTL formulas are built from tt and propositional variables using negation ($\neg$), disjunction ($\vee$), and the binary temporal operator XU called "strict until" and used in infix notation. For instance, when p is a propositional variable, then $\neg p \wedge \mathsf{tt}\,\mathsf{XU}\,p$ is an LTL formula. When P is a finite set of propositional variables and φ an LTL formula with propositional variables from P, then φ is called a formula over P.

LTL formulas are typically interpreted in infinite words, more precisely, given a finite set P of propositional variables, an LTL formula φ over P, a word $u \in (2^P)^\omega$, and $i \geq 0$, it is defined what it means that φ holds in u at position i, denoted $u, i \models \varphi$:

- $u, i \models \mathsf{tt}$,
- $u, i \models p$ if $p \in u(i)$, for every $p \in P$,
- $u, i \models \neg\varphi$ if $u, i \not\models \varphi$, for every LTL formula φ over P,
- $u, i \models \varphi \vee \psi$ if $u, i \models \varphi$ or $u, i \models \psi$, for LTL formulas φ and ψ over P, and
- $u, i \models \varphi\,\mathsf{XU}\,\psi$ if there exists $j > i$ such that $u, j \models \psi$ and $u, i' \models \varphi$ for all i' such that $i < i' < j$.

So $\varphi\,\mathsf{XU}\,\psi$ means that the formula φ holds true in the future until a point is reached where ψ holds true. For a word u as above and an LTL formula φ we say that φ holds in u, denoted $u \models \varphi$, if $u, 0 \models \varphi$. The language defined by φ is $\mathscr{L}(\varphi) = \{u \in (2^P)^\omega : u \models \varphi\}$, where, for convenience, we do not refer to P in the notation.

Clearly, there are many more basic temporal relations than just "until". So, often, other temporal operators are used:

- "next" is denoted X and defined by $\mathsf{X}\varphi = \neg\mathsf{tt}\,\mathsf{XU}\,\varphi$,
- "sometime in the future" is denoted XF and defined by $\mathsf{XF}\varphi = \mathsf{tt}\mathsf{XU}\,\varphi$, and
- "always in the future" is denoted XG and defined by $\mathsf{XG}\varphi = \neg\mathsf{XF}\neg\varphi$.

In many situations, it is convenient to include the current point in time, which leads to defining F by $\mathsf{F}\varphi = \varphi \vee \mathsf{XF}\varphi$ and, similarly, G by $\mathsf{G}\varphi = \neg\mathsf{F}\neg\varphi$ as well as U by $\varphi\,\mathsf{U}\,\psi = \psi \vee (\varphi \wedge \varphi\,\mathsf{XU}\,\psi)$.

It is remarkable that Kamp in his 1968 thesis [64] proved that every temporal relation expressible in natural (English) language can be expressed in linear-time temporal logic as defined above. As a yardstick for what is expressible in natural language he used first-order logic, considering formula with one free variable. To be precise, to obtain his result Kamp also had to add a past version of until, called since. That until by itself is enough to express everything expressible in first-order logic when only sentences are considered was proved by Gabbay, Pnueli, Shelah, and Stavi [47].

A typical LTL formula is

$$\mathsf{G}(p_r \to \mathsf{XF}p_a)$$

which expresses that for every occurrence of p_r there is a later occurrence of p_a, or, simply, every "request" is followed by an "acknowledge".

Another, more complicated, example is a formula expressing that competing requests are served in order. We assume that r_0 and r_1 are propositional variables indicating the occurrence of requests and a_0 and a_1 are matching propositional variables indicating the occurrence of acknowledgments. We want to specify that whenever an r_0 request occurs while no r_1 request is pending, then a_1 does not occur before the next occurrence of a_0.

We first specify that starting from an r_0 request there is an a_1 acknowledgment before an a_0 acknowledgment:

$$\alpha = r_0 \wedge (\neg a_0\,\mathsf{XU}\,(a_1 \wedge \neg a_0)).$$

Next, we observe that there are two different types of situations where an r_0 request can occur while no r_1 request is pending. The first type of situation is when there has been no r_1 request before the r_0 request in question. The second type is when a_1 occurred before the r_0 request in question and in between there has been no r_1 request. For each type of situation, we have a separate disjunct in our formula:

$$\neg(\neg r_1\,\mathsf{U}\,(\neg r_1 \wedge \alpha)) \vee \neg\mathsf{F}(a_1 \wedge \neg r_1\,\mathsf{U}\,(\neg r_1 \wedge \alpha)).$$

Clearly, in the context of LTL all the algorithmic problems discussed for S1S—satisfiability, conformance (model checking), and realizability—can be discussed. For instance, we can ask whether a given formula φ over P is satisfiable in the sense that there exists a word $u \in (2^P)^\omega$ such that $u \models \varphi$ or, given φ and a finite-state automaton $\mathscr{D}$ over 2^P, we can ask whether $u \models \varphi$ for all $u \in \mathscr{L}(\mathscr{D})$.

We can show in just one step that all these problems are decidable, namely by showing that every LTL formula is equivalent to an S1S formula; the results from Section 2 then apply. Unfortunately, the decision procedures that one obtains in this way have a nonelementary complexity. We can do better by using specifically tailored automata-theoretic constructions. We first present, however, the translation into S1S and then only turn to better decision procedures.

We start by defining the notion of equivalence we use to express the correctness of our translation. Let $P = \{p_0, \ldots, p_{r-1}\}$ be a set of propositional variables. Rather than interpreting LTL formulas over P in words over 2^P, we interpret them in words over $[2]_r$, where we think of every letter $a \in 2^P$ as the letter $b \in [2]_r$ with $b_{[j]} = 1$ iff $p_j \in a$ for every $j < r$. We say that an S1S formula $\psi = \psi(V_0, \ldots, V_{r-1})$ is equivalent to an LTL formula φ over P if for every $u \in [2]_r^\omega$ the following holds: $u \models \varphi$ iff $u \models \psi$.

In the proposition below, we make a stronger statement, and this involves the notion of global equivalence, which is explained next. Given a word $u \in [2]_r^\omega$, a position i, and an S1S formula $\psi = \psi(V_0, \ldots, V_{r-1}, x)$ where x is a first-order variable, we write $u, i \models \psi$ if ψ holds true when the set variables are assigned values according to u and x is assigned i. We say that ψ is globally equivalent to an LTL formula φ over P if the following holds: $u, i \models \varphi$ iff $u, i \models \psi$ for every $u \in [2]_r^\omega$ and every i.

Proposition 4.1. Let $P = \{p_0, \ldots, p_{r-1}\}$ be a finite set of propositional variables and x a first-order variable. For every LTL formula φ over P a globally equivalent S1S formula $\tilde{\varphi} = \varphi(V_0, \ldots, V_{r-1}, x)$ can be constructed.

Observe that $\exists x(\forall y \neg \text{suc}(y, x) \wedge \tilde{\varphi})$ is equivalent to φ.

Proof. A proof can be carried out by a straightforward induction on the structure of φ. When $\varphi = \mathsf{tt}$, we choose $\tilde{\varphi} = (x = x)$, and when $\varphi = p_j$, we take $\tilde{\varphi} = x \in V_j$.

In the inductive step, we distinguish various cases. When $\varphi = \neg\psi$, we can choose $\tilde{\varphi} = \neg\tilde{\psi}$. Similarly, when $\varphi = \psi \vee \chi$, we can choose $\tilde{\varphi} = \tilde{\psi} \vee \tilde{\chi}$. Finally, assume $\varphi = \psi \,\mathsf{XU}\, \chi$. Then we choose

$$\tilde{\varphi} = \exists z(x < z \wedge \tilde{\chi}(V_0, \ldots, V_{r-1}, z) \wedge \forall y(x < y < z \rightarrow \tilde{\psi}(V_0, \ldots, V_{r-1}, y))),$$

which simply reflects the semantics of XU. Q.E.D.

Observe that the above proof even shows that every formula is equivalent to a first-order formula (without set quantification but with ordering), and a slightly more careful proof would show that three first-order variables are sufficient [58]. Kamp's seminal result [64] is the converse of the above proposition when first-order logic with ordering is considered instead of S1S.

As a consequence of Proposition 4.1, we can state:

Corollary 4.2. LTL satisfiability, model-checking, and realizability are decidable.

This result is not very satisfying, because in view of [112, 111] the decision procedures obtained in this way have nonelementary complexity. As it turns out, it is much better to translate LTL directly into Büchi automata and carry out the same constructions we have seen for S1S all over again. The key is a good translation from LTL into Büchi automata.

4.2 From LTL to Büchi automata

Vardi and Wolper [124, 126] were the first to describe and advocate a separate translation from LTL into Büchi automata, resulting in essentially optimal bounds for the problems dealt with in Section 2. These bounds were originally achieved by Sistla and Clarke [108, 109], for satisfiability and model checking, and by Pnueli and Rosner [99], for realizability.

There are several ways of translating LTL into Büchi automata. We present two translations, a classical and a modern translation: the first one goes from an LTL formula via a generalized Büchi automaton to an ordinary Büchi automaton, while the second one goes via very weak alternating automata.

Both of the constructions we are going to present are based on formulas in positive normal form, which we define next. The operator "release", denoted XR, is defined by $\varphi \mathsf{XR} \psi = \neg(\neg\varphi \mathsf{XU} \neg\psi)$. In a certain sense, $\varphi \mathsf{XR} \psi$ expresses that the requirement of ψ to hold is released by the occurrence of φ. LTL formulas in positive normal form are built starting from tt, ff, p, and $\neg p$ using $\vee$, $\wedge$, XU, and XR, that is, negations are only allowed to occur right in front of propositional variables.

The following identities show that every LTL formula can be transformed into an equivalent LTL formula in positive normal form which is not longer than the given one (not counting negation symbols).

Lemma 4.3. For LTL formulas φ and ψ over a finite set P of propositional variables, $u \in (2^P)^\omega$, and $i \geq 0$, the following holds:

$$\begin{array}{lll} u,i \models \neg\mathsf{tt} & \text{iff} & u,i \models \mathsf{ff}, \\ u,i \models \neg\neg\varphi & \text{iff} & u,i \models \varphi, \\ u,i \models \neg(\varphi \vee \psi) & \text{iff} & u,i \models \neg\varphi \wedge \neg\psi, \\ u,i \models \neg(\varphi \,\mathsf{XU}\, \psi) & \text{iff} & u,i \models \neg\varphi \,\mathsf{XR}\, \neg\psi. \end{array}$$

Proof. A proof can be carried out in a straightforward fashion, using the definition of the semantics of LTL. Q.E.D.

As mentioned above, the other ingredient for our translation are generalized Büchi automata, introduced in [49]. Such an automaton is a tuple

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

where the first four components are as with ordinary Büchi automata, the only difference is in the last component: $\mathscr{F}$ is a set of subsets of Q, each called an acceptance set of $\mathscr{A}$. A run r is accepting if for every acceptance set $F \in \mathscr{F}$ there exist infinitely many i such that $r(i) \in F$. So generalized Büchi automata can express conjunctions of acceptance conditions in a simple way.

The essential idea for constructing a generalized Büchi automaton equivalent to a given LTL formula is as follows. As the automaton reads a given word it guesses which subformulas are true. At the same time it verifies its guesses. This is straightforward for almost all types of subformulas, for instance, when the automaton guesses that $\neg p$ is true, it simply needs to check that $p \notin a$ if a is the current symbol read. The only subformulas that are difficult to handle are XU-subformulas, that is, subformulas of the form $\psi \,\mathsf{XU}\, \chi$. Checking that such a subformula is true cannot be done directly or in the next position in general because the "satisfaction point" for an XU-formula—the position where χ becomes true—can be in the far future. Of course, by keeping $\psi \,\mathsf{XU}\, \chi$ in the state the automaton can remember the obligation to eventually reach a satisfaction point, but the acceptance condition is the only feature of the automaton which can be used to really check that reaching the satisfaction point is not deferred forever.

The complete construction is described in Figure 17; it uses $\mathrm{sub}(\varphi)$ to denote the set of all subformulas of a formula φ including φ itself. Note that for every XU-subformula $\psi \,\mathsf{XU}\, \chi$ there is a separate acceptance set, which contains all states which do not have an obligation for eventually satisfying this subformula or satisfy it in the sense that χ is an obligation too.

Theorem 4.4 (Gerth-Peled-Vardi-Wolper, [49]). Let P be a finite set of propositional variables and φ an LTL formula over P with n subformulas and k XU-subformulas. Then $\mathscr{A}[\varphi]$ is a generalized Büchi automaton with 2^n states and k acceptance sets such that $\mathscr{L}(\mathscr{A}[\varphi]) = \mathscr{L}(\varphi)$.

Let P be a finite set of propositional variables and φ an LTL formula over P in positive normal form. The generalized Büchi automaton for φ with respect to P, denoted $\mathscr{A}[\varphi]$, is defined by

$$\mathscr{A}[\varphi] = (2^P, 2^{\mathrm{sub}(\varphi)}, Q_I, \Delta, \mathscr{F})$$

where a triple (Ψ, a, Ψ') with $\Psi, \Psi' \subseteq \mathrm{sub}(\varphi)$ and $a \in 2^P$ belongs to Δ if the following conditions are satisfied:

(i) $\mathsf{ff} \notin \Psi$,

(ii) $p \in \Psi$ iff $p \in a$, for every $p \in P$,

(iii) if $\psi \vee \chi \in \Psi$, then $\psi \in \Psi$ or $\chi \in \Psi$,

(iv) if $\psi \wedge \chi \in \Psi$, then $\psi \in \Psi$ and $\chi \in \Psi$,

(v) if $\psi \,\mathsf{XU}\, \chi \in \Psi$, then $\chi \in \Psi'$ or $\{\psi, \psi \,\mathsf{XU}\, \chi\} \subseteq \Psi'$,

(vi) if $\psi \,\mathsf{XR}\, \chi$, then $\{\psi, \chi\} \subseteq \Psi'$ or $\{\chi, \psi \,\mathsf{XR}\, \chi\} \subseteq \Psi'$,

and where

$$Q_I = \{\Psi \subseteq \mathrm{sub}(\varphi) \colon \varphi \in \Psi\},$$
$$\mathscr{F} = \{F_{\psi \mathsf{XU} \chi} \colon \psi \,\mathsf{XU}\, \chi \in \mathrm{sub}(\varphi)\},$$

with $F_{\psi \mathsf{XU} \chi}$ defined by

$$F_{\psi \mathsf{XU} \chi} = \{\Psi \subseteq \mathrm{sub}(\varphi) \colon \chi \in \mathrm{sub}(\varphi) \text{ or } \psi \,\mathsf{XU}\, \chi \notin \Psi\}.$$

FIGURE 17. From LTL to generalized Büchi automata

Proof. We first show $\mathscr{L}(\mathscr{A}[\varphi]) \subseteq \mathscr{L}(\varphi)$. Let $u \in \mathscr{L}(\mathscr{A}[\varphi])$ and let r be an accepting run of $\mathscr{A}[\varphi]$ on u. We claim that for every i, if $\psi \in r(i)$, then $u, i \models \psi$. The proof is by induction on the structure of ψ. If $\psi = \mathsf{tt}$, $\psi = \mathsf{ff}$, $\psi = p$, or $\psi = \neg p$, then this follows directly from (i) or (ii). If $\psi = \chi \vee \zeta$, the claim follows from the induction hypothesis and (iii). Similarly, the claim holds for a conjunction.

Assume $\psi = \chi \,\mathsf{XR}\, \zeta$. Then (vi) tells us that

(a) $\chi \,\mathsf{XR}\, \zeta i \subseteq r(j)$ for every $j > i$ or

(b) there exists $j \geq i$ such that $\chi \, \mathsf{XR} \, \zeta \subseteq r(i')$ for i' with $i < i' < j$ and $\chi, \zeta \in r(j)$.

From the induction hypothesis and (a), we can conclude that we have $u, i' \models \zeta$ for all $i' > i$, which means $u, i \models \psi$. Similarly, from the induction hypothesis and (b), we can conclude that we have $u, i' \models \zeta$ for all i' such that $i < i' \leq j$ and $u, j \models \chi$, which implies $u, i \models \psi$, too.

Finally, assume $\psi = \chi \, \mathsf{XU} \, \zeta$. From (v), we obtain that

(a) $\chi \, \mathsf{XU} \, \zeta \in r(j)$ for all $j > i$ or

(b) there exists j such that $\chi \, \mathsf{XU} \, \zeta \in r(i')$ for all i' with $i < i' < j$ and $\zeta \in r(j)$.

Just as with XR, we obtain $u, i \models \psi$ from the induction hypothesis and (b). So we only need to show that if (a) occurs, we also have (b). Since r is accepting, there is some $\Psi \in F_{\chi \mathsf{XU} \zeta}$ such that $r(j) = \Psi$ for infinitely many j. Assuming (a), we can can conclude $\zeta \in \Psi$, which, by induction hypothesis, means we also have (b).

For the other inclusion, $\mathscr{L}(\varphi) \subseteq \mathscr{L}(\mathscr{A}[\varphi])$, we simply show that for a given u such that $u \models \varphi$ the word r defined by $r(i) = \{\psi \in \mathrm{sub}(\varphi) \colon u, i \models \psi\}$ is an accepting run of $\mathscr{A}[\varphi]$ on u. To this end, we need to show that

(a) r starts with an initial state,

(b) $(r(i), u(i), r(i+1)) \in \Delta$ for all i, and

(c) $r(i) \in F_{\psi \mathsf{XU} \chi}$ for infinitely many i, for every formula $\psi \, \mathsf{XU} \, \chi \in \mathrm{sub}(\varphi)$.

That (a) is true follows from the assumption $u \models \varphi$. Condition (b) is true simply because of the semantics of LTL. To see that (c) is true, let $\psi \, \mathsf{XU} \, \chi \in \mathrm{sub}(\varphi)$. We distinguish two cases. First, assume there exists i such that $u, j \not\models \chi$ for all $j > i$. Then $u, j \not\models \psi \, \mathsf{XU} \, \chi$ for all $j \geq i$, hence $r(j) \in F_{\psi \mathsf{XU} \chi}$ for all $j \geq i$, which is enough. Second, assume there are infinitely many i such that $u, i \models \chi$. Then $\chi \in r(i)$ for the same values of i, which is enough, too. Q.E.D.

Generalized Büchi automata can be converted into equivalent Büchi automata in a straightforward fashion. The idea is to check that every acceptance set is visited infinitely often by visiting these sets one after the other, in a fixed order, and repeating this process over and over again. In Figure 18, a respective construction is described. The second component of the state space is a counter which is used to keep track of the acceptance set to be visited next. When this counter reaches its maximum, every acceptance set has been visited once, and it can be reset.

Let $\mathscr{A}$ be a generalized Büchi automaton with $\mathscr{F} = \{F_0, \dots, F_{k-1}\}$. The Büchi automaton $\mathscr{A}^{\mathrm{BA}}$ is defined by

$$\mathscr{A}^{\mathrm{BA}} = (A, Q \times \{0, \dots, k\}, Q_I, \Delta', Q \times \{k\})$$

where Δ' contains for every $(q, a, q') \in \Delta$ the following transitions:

- $((q, k), a, (q', 0))$,
- $((q, i), a, (q', i))$ for every $i < k$,
- $((q, i), a, (q', i + 1))$ for every $i < k$ such that $q' \in F_i$.

FIGURE 18. From generalized Büchi to ordinary Büchi automata

Remark 4.5. Let $\mathscr{A}$ be a generalized Büchi automaton with n states and k acceptance sets. Then $\mathscr{A}^{\mathrm{BA}}$ is an equivalent Büchi automaton with at most $(k + 1)n$ states.

Corollary 4.6 (Vardi-Wolper, [124, 126])**.** There exists an effective procedure that given an LTL formula φ with n states and k XU-subformulas outputs a Büchi automaton $\mathscr{A}$ with at most $(k + 1)2^n$ states such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\varphi)$.

4.3 From LTL to alternating automata

The above translation from LTL into Büchi automata serves our purposes perfectly. We can use it to derive all the desired results about the complexity of the problems we are interested in, satisfiability, model checking, and realizability, as will be shown in the next subsection. There is, however, a translation using alternating automata, which is interesting in its own right. The motivation behind considering such a translation is to pass from the logical framework to the automata-theoretic framework in an as simple as possible fashion (to be able to apply powerful automata-theoretic tools as early as possible).

Alternating automata are provided with a feature to spawn several copies of themselves while running over a word. Formally, an alternating Büchi automaton is a tuple

$$\mathscr{A} = (P, Q, q_I, \delta, F)$$

where P, Q, and q_I are as usual, F is a Büchi acceptance condition, and δ is a function which assigns to each state q a transition condition, where every transition condition $\delta(q)$ is a positive boolean combination of formulas of the form p and $\neg p$, for $p \in P$, and $\bigcirc q$, for $q \in Q$. More precisely, the set of transition conditions over P and Q, denoted $\mathrm{TC}(P,Q)$, is the smallest set such that

(i) $\mathsf{tt}, \mathsf{ff} \in \mathrm{TC}(P,Q)$,

(ii) $p, \neg p \in \mathrm{TC}(P,Q)$ for every $p \in P$,

(iii) $\bigcirc q \in \mathrm{TC}(P,Q)$ for every $q \in Q$,

(iv) $\gamma \wedge \gamma', \gamma \vee \gamma' \in \mathrm{TC}(P,Q)$ for $\gamma, \gamma' \in \mathrm{TC}(P,Q)$.

A run of such an automaton on a word $u \in (2^P)^\omega$ is a tree $\mathscr{R}$ labeled with elements from $(Q \cup \mathrm{TC}(P,Q)) \times \omega$ such that $l^{\mathscr{R}}(\mathrm{root}(\mathscr{R})) = (q_I, 0)$ and the following conditions are satisfied for every $v \in V^{\mathscr{R}}$, assuming $l^{\mathscr{R}}(v) = (\gamma, i)$:

(i) $\gamma \neq \mathsf{ff}$,

(ii) if $\gamma = p$ for some $p \in P$, then $p \in u(i)$,

(iii) if $\gamma = \neg p$ for some $p \in P$, then $p \notin u(i)$.

(iv) if $\gamma = q$, then v has a successor v' such that $l^{\mathscr{R}}(v') = (\delta(q), i)$,

(v) if $\gamma = \bigcirc q'$, then v has a successor v' such that $l^{\mathscr{R}}(v') = (q', i+1)$,

(vi) if $\gamma = \gamma_0 \wedge \gamma_1$, then v has successors v_0 and v_1 such that $l^{\mathscr{R}}(v_j) = (\gamma_j, i)$ for $j < 2$,

(vii) if $\gamma = \gamma_0 \vee \gamma_1$, then there exists $j < 2$ such that v has a successor v' with $l^{\mathscr{R}}(v') = (\gamma_j, i)$.

An infinite branch b of $\mathscr{R}$ is accepting if there are infinitely many i such that $l^{\mathscr{R}}(b(i)) \in F \times \omega$, in other words, there are infinitely many vertices with a final state in the first component of their labeling. The run is accepting if every infinite branch of it is accepting.

As a simple example, consider the language L_{10} over 2^P where $P = \{p\}$ which contains all words u satisfying the following condition: There exists some number i such that $p \in u(j+10)$ for all $j \geq i$ with $p \in u(j)$. If we wanted to construct a nondeterministic automaton for this language, we could not do with less than 1000 states, but there is a small alternating automaton that recognizes this language. It simply guesses the right position i and for each position j it spawns off a copy of itself checking that after 10 further steps p holds true again. The details are given in Figure 19.

The automaton has states $q_I, q_0, q_1, \dots, q_{10}$ where q_0 is the only final state and the transition function δ is defined by

- $\delta(q_I) = \bigcirc q_I \vee \bigcirc q_0$,
- $\delta(q_0) = \bigcirc q_0 \wedge ((p \wedge \bigcirc q_1) \vee \neg p)$,
- $\delta(q_i) = \bigcirc q_{i+1}$ for all i such that $0 < i < 10$,
- $\delta(q_{10}) = p$.

FIGURE 19. Example for an alternating automaton

It is (vi) from above which forces the tree to become a real tree, that is, it requires that a vertex has two successors (unless $\gamma_0 = \gamma_1$). So this is the condition that makes the automaton alternating: For a run to be accepting, both alternatives have to be pursued.

The translation from LTL to alternating Büchi automata, given in Figure 20, is straightforward as it simply models the semantics of LTL. It exploits the fact that $\psi \mathsf{XU} \chi$ and $\psi \mathsf{XR} \chi$ are equivalent to $\mathsf{X}\chi \vee (\mathsf{X}\psi \wedge \mathsf{X}(\psi \mathsf{XU} \chi))$ and $\mathsf{X}\chi \wedge (\mathsf{X}\psi \vee \mathsf{X}(\psi\, \mathsf{XR}\, \chi))$, respectively. Note that we use the notation $[\psi]$ to distinguish subformulas of φ from transition conditions ($p_0 \wedge p_1$ is different from $[p_0 \wedge p_1]$).

The transition function of $\mathscr{A}^{\text{alt}}[\varphi]$ has an interesting property, which we want to discuss in detail. Let $\leq$ be any linear ordering which extends the partial order on Q defined by $[\psi] \leq [\chi]$ if $\psi \in \text{sub}(\chi)$. For every $\psi \in \text{sub}(\varphi)$ and every $[\chi]$ occurring in $\delta([\psi])$, we have $[\chi] \leq [\psi]$. Following Gastin and Oddoux [48], we call an automaton satisfying this property a very weak alternating automaton.

The transition function of $\mathscr{A}^{\text{alt}}[\varphi]$ has an even stronger structural property, which we explain next. For a given symbol $a \in 2^P$, a transition condition γ, a state $q \in Q$, and a set $Q' \subseteq Q$, we define what it means that Q' is an a-successor of q with respect to γ, denoted $q \rightarrow^{a,\gamma} Q'$. This is defined inductively:

- $q \rightarrow^{a,\mathsf{tt}} \varnothing$,
- $q \rightarrow^{a,p} \varnothing$ if $p \in a$, and, similarly, $q \rightarrow^{a,\neg p} \varnothing$ if $p \notin a$,
- $q \rightarrow^{a,\bigcirc q'} \{q'\}$,
- $q \rightarrow^{a,\gamma_0 \vee \gamma_1} Q'$ if $q \rightarrow^{a,\gamma_0} Q'$ or $q \rightarrow^{a,\gamma_1} Q'$,

Let φ be an LTL formula in positive normal form over P and Q the set which contains for each $\psi \in \mathrm{sub}(\varphi)$ an element denoted $[\psi]$. The automaton $\mathscr{A}^{\mathrm{alt}}[A]$ is defined by

$$\mathscr{A}^{\mathrm{alt}}[\varphi] = (P, Q, [\varphi], \delta, F)$$

where

$$\begin{aligned} \delta([\mathsf{tt}]) &= \mathsf{tt}, & \delta([\mathsf{ff}]) &= \mathsf{ff}, \\ \delta([p]) &= p, & \delta([\neg p]) &= \neg p, \\ \delta([\psi \vee \chi]) &= \delta([\varphi]) \vee \delta([\psi]), & \delta([\psi \wedge \chi]) &= \delta([\varphi]) \wedge \delta([\psi]), \end{aligned}$$

$$\begin{aligned} \delta([\psi \,\mathsf{XU}\, \chi]) &= \bigcirc[\chi] \vee (\bigcirc[\psi] \wedge \bigcirc[\psi \,\mathsf{XU}\, \chi]), \\ \delta([\psi \,\mathsf{XR}\, \chi]) &= \bigcirc[\psi] \wedge (\bigcirc[\psi] \vee \bigcirc[\psi \,\mathsf{XR}\, \chi]), \end{aligned}$$

and F contains all the elements $[\psi] \in Q$ where ψ is not a XU-formula.

FIGURE 20. From an LTL formula to an alternating automaton

- $q \to^{a,\gamma_0 \wedge \gamma_1} Q'$ if there exists $Q_0, Q_1 \subseteq Q$ such that $Q' = Q_0 \cup Q_1$, $q \to^{a,\gamma_0} Q_0$, and $q \to^{a,\gamma_1} Q_1$.

Note that $q \to^{a,\gamma} Q'$ has a natural interpretation in terms of runs. If a vertex v of a run is labeled (q, i) and Q' is the set of all states q' such that $(q', i+1)$ is a label of a descendant of v, then $q \to^{a,\gamma} Q'$, provided, of course, that the run is minimal, which we can and will henceforth assume without loss of generality.

We use $q \to^{a} Q'$ as an abbreviation for $q \to^{a,\delta(q)} Q'$. We say a state q is persistent if there exists Q' such that $q \in Q'$ and $q \to^{a} Q'$ for some letter a.

Using the new notation, we can give an equivalent definition of being a very weak alternating automaton. It simply means that there exists a linear ordering $\leq$ on the states of the automaton such that if $q \to^{a} Q'$, then $q' \leq q$ for all $q' \in Q'$.

The automaton $\mathscr{A}^{\mathrm{alt}}[\varphi]$ has the following property. For every persistent state $q \in F$ there exists a state q' such that

(i) $q \to^{a} \{q'\}$ for every letter a and

(ii) whenever $q \to^{a} Q'$, then either $q \in Q'$ or $Q' = \{q'\}$.

(Every $q \notin F$ is of the form $[\psi \,\mathsf{XU}\, \chi]$, which means that we can choose $q' = [\chi]$.) We call very weak alternating automata that have this property ultra weak alternating automata and a state as q' above a discharging state for q and denote it by q^{d}.

Lemma 4.7. Let φ be an LTL formula with n subformulas. Then $\mathscr{A}^{\mathrm{alt}}[\varphi]$ is an ultra weak alternating automaton with n states such that $\mathscr{L}(\mathscr{A}^{\mathrm{alt}}[\varphi]) = \mathscr{L}(\varphi)$.

Proof. We only need to prove its correctness, which we do by an induction on the structure of φ. We start with a simple observation. Let $\mathscr{R}$ be an accepting run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u and $v \in V^{\mathscr{R}}$ labeled $([\psi], i)$ for some $\psi \in \mathrm{sub}(\varphi)$. Then $\mathscr{R}{\downarrow}v$ can be turned into an accepting run of $\mathscr{A}^{\mathrm{alt}}[\psi]$ on $u[i,*)$ by changing each second component j of a vertex label to $j - i$. Clearly, for this to be true $\mathscr{R}$ needs to be minimal (see above).

For the induction base, first assume $\varphi = \mathsf{tt}$ or $\varphi = \mathsf{ff}$. There is nothing to show. Second, assume $\varphi = p$ and suppose $u \models \varphi$. Then $p \in u(0)$, that is, the two-vertex tree where the root is labeled $([p], 0)$ and its only successor is labeled $(p, 0)$ is an accepting run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u. Conversely, if $\mathscr{R}$ is a (minimal) run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u, then $\mathscr{R}$ has two vertices labeled $([p], 0)$ and $(p, 0)$, respectively. This implies $p \in u(0)$, which, in turn, implies $u \models \varphi$. An analogous argument applies to $\neg p$.

In the inductive step, first assume $\varphi = \psi_0 \wedge \psi_1$. If there exists an accepting run $\mathscr{R}$ of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u, then, because of $\delta([\varphi]) = \delta([\psi_0]) \wedge \delta([\psi_1])$, the root has successors v_0 and v_1 such that $l^{\mathscr{R}}(v_i) = (\delta([\psi_i]), 0)$. For every i, we can turn $\mathscr{R}{\downarrow}v_i$ into an accepting run $\mathscr{R}_i$ of $\mathscr{A}^{\mathrm{alt}}[\psi_i]$ on u by adding a new root labeled $([\psi_i], 0)$. By induction hypothesis, we obtain $u \models \psi_i$ for every i, hence $u \models \varphi$. Conversely, assume $u \models \varphi$. Then $u \models \psi_i$ for $i < 2$, and, by induction hypothesis, there exist accepting runs $\mathscr{R}_i$ of $\mathscr{A}^{\mathrm{alt}}[\psi_i]$ on u for $i < 2$. These runs can be turned into an accepting run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u by simply making their vertex sets disjoint, removing their roots, and adding a new common root labeled $([\varphi], 0)$.

A similar argument applies to formulas of the form $\psi_0 \vee \psi_1$.

Next, assume $\varphi = \psi \mathsf{XU} \chi$. Suppose $\mathscr{R}$ is an accepting run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u and let v_0 be the root of this run. Also, let $u_i = u[i,*)$ for every i. Then, by definition of accepting run, $l^{\mathscr{R}}(v_0) = ([\psi \mathsf{XU} \chi], 0)$. From the definition of the transition function we can conclude that v_0 has a successor, say v_1, which is labeled by $(\bigcirc[\chi] \vee (\bigcirc[\psi] \wedge \bigcirc[\psi \,\mathsf{XU}\, \chi]), 0)$, which, in turn, has a successor, say v_2, which is labeled by either $(\bigcirc[\chi], 0)$ or $(\bigcirc[\psi] \wedge \bigcirc[\psi \,\mathsf{XU}\, \chi], 0)$. In the first case, there is a further successor labeled $([\chi], 1)$ and we obtain $u_1 \models \chi$ from the induction hypothesis, hence, $u \models \varphi$. In the second case, we know there exist successors v_3 and v_3' of v_2 labeled $(\bigcirc[\psi \,\mathsf{XU}\, \chi], 0)$ and $(\bigcirc[\psi], 0)$, respectively, which themselves have successors v_4 and v_4' labeled

$([\psi \mathsf{XU}\, \chi], 1)$ and $([\psi], 1)$, respectively. By induction hypothesis, we obtain $u_1 \models \psi$. Applying the same arguments as before, we find that either there is a vertex labeled $([\chi], 2)$ or there are vertices v_8 and v_8' labeled $[(\psi \mathsf{XU}\, \chi, 2)]$ and $([\psi], 2)$, respectively. In the first case, we get $u \models \varphi$ because we also know $u_1 \models \psi$, whereas in the second case we can again apply the same arguments as before. Continuing in this fashion, we find that the only case which remains is the one where we have an infinite sequence of vertices $v_4, v_8, v_{12}, \dots$ on the same branch and every vertex with label in $Q \times \omega$ is labeled $([\varphi], i)$, which means that this branch is not accepting—a contradiction.

For the other direction, assume $u \models \varphi$ and use the same notation as before. Then there is some $j > 0$ such that $u_j \models \chi$ and $u_i \models \psi$ for all i with $0 < i < j$. By induction hypothesis, there are accepting runs $\mathscr{R}_i$ for i with $0 < i < j$ of $\mathscr{A}^{\text{alt}}[\psi]$ on u_i and an accepting run $\mathscr{R}_j$ of $\mathscr{A}^{\text{alt}}[\chi]$ on u_j. Assume that $v_1, \dots, v_j$ are the roots of these trees and assume that their sets of vertices are pairwise disjoint. Then we can construct an accepting run $\mathscr{R}$ for $\mathscr{A}^{\text{alt}}[\varphi]$ on u as follows. The vertices of $\mathscr{R}$ are the vertices of the $\mathscr{R}_k$'s and, in addition, the vertices $w_0, w_0', w_0'', w_0''', \hat{w}_0, w_1, \dots, w_{j-1}, w_{j-1}', w_{j-1}''$. The labeling is as follows:

- w_i is labeled $([\varphi], i)$ for $i < j$,
- w_i' is labeled $(\bigcirc[\chi] \vee (\bigcirc[\psi] \wedge \bigcirc[\varphi]), i)$ for $i < j$,
- w_i'' is labeled $(\bigcirc[\psi] \wedge \bigcirc[\varphi], i)$ for $i < j - 1$,
- w_i''' is labeled $(\bigcirc[\varphi], i)$ for $i < j - 1$,
- $\hat{w}_i$ is labeled $(\bigcirc[\psi], i)$ for $i < j - 1$, and
- w_j'' is labeled $(\bigcirc[\chi], j - 1)$.

The tree $\mathscr{R}$ has all edges from the $\mathscr{R}_k$'s and, in addition,

- edges such that $w_0 w_0' w_0'' w_0''' \dots w_{j-1} w_{j-1}' w_{j-1}'' v_j$ is a path and
- edges $(w_i', \hat{w}_i)$ and $(\hat{w}_i, v_i)$ for every $i < j$.

This yields an accepting run of $\mathscr{A}^{\text{alt}}[\varphi]$ on u.

Finally, XR can be dealt with in a similar fashion. Q.E.D.

It is not very difficult to translate alternating Büchi automata into nondeterministic Büchi automata, as was shown by Miyano and Hayashi [87], but it yields a worse upper bound compared to a translation from ultra weak alternating automata to Büchi automata. This is why we present the

Let $\mathscr{A}$ be an ultra weak alternating automaton over a finite set P of propositional variables. The generalized Büchi automaton for $\mathscr{A}$, denoted $\mathscr{A}^{\mathrm{gBA}}$, is defined by

$$\mathscr{A}^{\mathrm{gBA}} = (2^P, 2^Q, \{q_I\}, \Delta, \mathscr{F})$$

where

- the transition relation Δ contains a transition (Q', a, Q'') if for every $q \in Q'$ there exists a set Q_q such that $q \to^{a,\delta(q)} Q_q$ and $\bigcup_{q \in Q'} Q_q \subseteq Q''$ and
- the set $\mathscr{F}$ of acceptance sets contains for every $q \notin F$ the set F_q defined by $\{Q' \subseteq Q \colon q^{\mathrm{d}} \in Q' \text{ or } q \notin Q'\}$.

FIGURE 21. From ultra weak to generalized Büchi automata

latter. Another advantage of this translation is that it can be simplified by going through alternating generalized Büchi automata.

The main idea of the translation from ultra weak alternating automata to (generalized) Büchi automata is to use a powerset construction to keep track of the individual branches of an accepting run of the alternating automaton. There are two technical problems that we face in the translation. First, we need to take care of the vertices in the runs which are not labeled with a state (but with a transition condition), and, second, we need to take care of the acceptance condition. The first problem is similar to removing ε-transitions and the second problem can be solved by using the fact that the automata are ultra weak. The entire construction is described in Figure 21.

Lemma 4.8. Let $\mathscr{A}$ be an ultra weak alternating automaton with n states and k final states. Then $\mathscr{A}^{\mathrm{gBA}}$ is an equivalent generalized Büchi automaton with 2^n states and k acceptance sets.

Proof. The claim about the number of states and the number of acceptance sets is obvious. We only need to show that the translation is correct.

First, assume $u \in \mathscr{L}(\mathscr{A})$. Then there is an accepting run $\mathscr{R}$ of $\mathscr{A}$ on u (which we assume to be minimal again). We say a vertex $v \in V^{\mathscr{R}}$ is a state vertex if the first component of its label is a state. Let $\mathscr{R}'$ be the tree which is obtained from $\mathscr{R}$ by "removing" the non-state vertices while keeping their edges. Formally, $\mathscr{R}'$ is constructed inductively as follows. We start with the root of $\mathscr{R}$, which is a state vertex by definition. Then, once we have a vertex v of $\mathscr{R}'$, we add all state vertices v' of $\mathscr{R}$ as successors of v to $\mathscr{R}'$ which can be reached from v in $\mathscr{R}$ via a path without state vertices (not counting the first and last vertex).

The tree $\mathscr{R}'$ has the following property. When v is a vertex labeled (q, i) and $\{v_0, \dots, v_{m-1}\}$ is the set of its successors where v_j is labeled (q_j, i_j), then $q \rightarrow^{u(i)} \{q_0, \dots, q_{m-1}\}$ and $i_j = i + 1$ for every $j < m$. This is because the definition of $\rightarrow^{a,\gamma}$ simply models the requirements of a run.

Using the above property of $\mathscr{R}'$ we can easily construct a run r of $\mathscr{A}^{\mathrm{gBA}}$ on u as follows. We simply let $r(i)$ be the set of all q such that there exists a vertex v in $\mathscr{R}'$ labeled (q, i). By definition of $\mathscr{A}^{\mathrm{gBA}}$, this is a run. What remains to be shown is that r is an accepting run.

Assume $q \notin F$ and i is an arbitrary number. We have to show that there exists $j \geq i$ such that $r(j) \in F_q$. If there is some $j \geq i$ such that $q \notin r(j)$, this is true. So assume that $q \in r(j)$ for all $j \geq i$. By construction of $\mathscr{R}'$ there exists a vertex v_0 in $\mathscr{R}'$ which is labeled (q, i). If one of the successors of v_0 is labeled q^{d} in the first component, then $r(i + 1) \in F_q$, which is enough. If, on the other hand, all successors are labeled distinct from q^{d} in their first component, then, since $\mathscr{A}$ is assumed to be ultra weak, one of the successors, say v_1, is labeled q in the first component. We can apply the same argument as before to v_1 now. We find that $r(i + 2) \in F_q$ or we find a successor v_2 of v_1 with q in the first component of its label, too. If we continue like this and we do not find $r(j)$ such that $r(j) \in F_q$, we obtain an infinite path $v_0v_1\dots$ in $\mathscr{R}'$ where every v_i is labeled q in the first component. This path can be prefixed such that it becomes a branch of $\mathscr{R}$, and this branch is not accepting—a contradiction to the assumption that $\mathscr{R}$ is accepting.

For the other direction, assume $u \in (2^P)^\omega$ is accepted by $\mathscr{A}^{\mathrm{gBA}}$ and let r be an accepting run of $\mathscr{A}^{\mathrm{gBA}}$ on u. For every i and every $q \in r(i)$, let Q_q^i be a set such that $q \rightarrow^{u(i),\delta(q)} Q_q^i$ for all $q \in r(i)$ and $\bigcup\{Q_q^i \colon q \in r(i)\} \subseteq r(i)$. By definition of $\mathscr{A}^{\mathrm{gBA}}$, such sets exist. For some combinations of q and i there might be several choices for Q_q^i. By convention, if $q^{\mathrm{d}} \in r(i + 1)$, we let $Q_q^i = \{q^{\mathrm{d}}\}$, which is a possible choice since $\mathscr{A}$ is assumed to be ultra weak. Using these sets, we construct a tree $\mathscr{R}'$ from r inductively as follows. We start with the root and label it $(q_I, 0)$. If we have a vertex v labeled (q, i), we add a successor to v for every $q' \in Q_q^i$ and label it $(q', i + 1)$. By expanding $\mathscr{R}'$ according to the semantics of the transition conditions, we obtain a tree $\mathscr{R}$ which is a run of $\mathscr{A}$ on u. It remains to be shown that this run is accepting. Assume this is not the case. Then, because $\mathscr{A}$ is ultra weak, there is a non-final state q, a branch $v_0v_1\dots$ of $\mathscr{R}'$, and a number i such that the label of v_j is (q, j) for all $j \geq i$. This implies $q \in Q_q^i$ for all $j \geq i$. Since r is accepting, we know that there exists $j > i$ such that $q \notin r(j)$ or $q^{\mathrm{d}} \in r(j)$. The first condition is an immediate contradiction. So assume $q^{\mathrm{d}} \in r(j)$ for some $j > i$. Since we have $q \in r(j-1)$, we also have $Q_q^j = \{q^{\mathrm{d}}\}$ by construction—a contradiction. Q.E.D.

Combining the previous lemma and Remark 4.5 yields an alternative proof of Corollary 4.6. Very weak alternating automata are interesting for another reason, too:

Theorem 4.9 (Rohde, [103]). For every very weak alternating automaton $\mathscr{A}$ there exists an LTL formula φ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathscr{A})$.

This was also proved by Löding and Thomas [81] and a proof of it can be found in [30].

4.4 LTL satisfiability, model checking, and realizability

We can now return to the problems we are interested in, satisfiability, validity, model checking, and realizability.

Theorem 4.10 (Clarke-Emerson-Sistla, [27]). LTL satisfiability is PSPACE-complete.

Proof. Given an LTL formula φ over a set P of propositional variables, we construct a Büchi automaton equivalent to φ and check this automaton for nonemptiness. Clearly, this procedure is correct. To determine its complexity, we use the following simple fact from complexity theory.

(†) Let $f\colon A^* \to B^*$ be a function computable in PSPACE and $L \subseteq B^*$ a problem solvable in nondeterministic logarithmic space. Then $f^{-1}(P) \in$ PSPACE.

When we apply (†) to the situation where f computes the above Büchi automaton equivalent to φ and L is the problem whether a Büchi automaton accepts some word, then we obtain that our problem is in PSPACE.

For the lower bound, we refer the reader to [27] or [106]. Q.E.D.

For model checking, the situation is essentially the same as with S1S. When we are given a finite-state automaton $\mathscr{D}$ over the alphabet 2^P for some finite set P of propositional variables and φ is an LTL formula over P, we write $\mathscr{D} \models \varphi$ if $u \models \varphi$ for all $u \in \mathscr{L}(\mathscr{D})$. LTL model checking is the problem, given $\mathscr{D}$ and φ, to determine whether $\mathscr{D} \models \varphi$, that is, whether $\mathscr{L}(\mathscr{D}) \subseteq \mathscr{L}(\varphi)$.

Theorem 4.11. (Sistla-Clarke-Lichtenstein-Pnueli, [109, 80])

(1) LTL model checking is PSPACE-complete.

(2) Given a formula φ with n subformulas and a finite-state automaton $\mathscr{D}$ of size m, whether $\mathscr{D} \models \varphi$ holds can be checked in time $2^{O(n)}m$.

Proof. The same approach as in Section 2.1 yields the desired upper bounds. Given a finite set of propositional variables P, a finite-state automaton $\mathscr{D}$ over 2^P, and an LTL formula over P, we first construct the product $\mathscr{A} \times \mathscr{D}$ where $\mathscr{A}$ is a Büchi automaton equivalent to $\neg\varphi$. We have $\mathscr{L}(\mathscr{A} \times \mathscr{D}) = \varnothing$ if and only if $\mathscr{D} \models \varphi$. So, to conclude, we apply an emptiness test.

The number of states of the product is at most $(k+1)2^n \cdot m$ where n is the size of φ, the number k is the number of XU-formulas in φ (after transformation to positive normal form), and m is the number of states of $\mathscr{D}$. Using the same complexity-theoretic argument as in the proof of Theorem 4.10, we obtain part 1.

Part 2 follows from the fact that an emptiness test for a Büchi automaton can be carried out in time linear in the size of the automaton.

For the lower bound, we refer the reader to [27]. Q.E.D.

Finally, we turn to realizability, which is defined as with S1S (see Section 2.4). An LTL realizability instance is an LTL formula over a set $P = \{p_0, \ldots, p_{m-1}, q_0, \ldots, q_{n-1}\}$ of propositional variables. Just as earlier in this section, we interpret such formulas in words over $[2]_{m+n}$, which means that a solution of such an instance is a function $f \colon [2]_m^+ \to [2]_n$ satisfying the requirement known from the S1S setting, that is, $u^\frown v \models \varphi$ holds for every $u \in [2]_m^\omega$ and $v \in [2]_n^\omega$ defined by $v(i) = f(u[0,i])$ for every i.

We can use the same technique as in Section 3 to obtain the following result:

Theorem 4.12 (Pnueli-Rosner, [99])**.** LTL realizability is complete for doubly exponential time. Moreover, for every positive instance a finite-state machine realizing a finite-state solution can be computed within the same time bound.

Proof. Consider the following algorithm for solving a given instance φ over $\{p_0, \ldots, p_{m-1}, q_0, \ldots, q_{n-1}\}$. First, consider the game $\mathscr{G}[\varphi]$ which is obtained using the construction from Figure 10 with the S1S formula replaced by the LTL formula. Second, compute a Büchi automaton $\mathscr{A}$ equivalent to φ according to Corollary 4.6. Third, turn $\mathscr{A}$ into a deterministic parity automaton $\mathscr{B}$ according to 2.14. Fourth, let $\mathscr{G} = \mathscr{G}[\varphi] \times \mathscr{B}$ be the game obtained from expanding $\mathscr{G}[\varphi]$ by $\mathscr{B}$. Fifth, solve the game $\mathscr{G}$ using Theorem 2.21. Player 0 wins $\mathscr{G}$ if and only if φ is a positive instance of realizability.

To prove the desired complexity bound let n be the number of subformulas of φ and observe the following. The size of $\mathscr{A}$ is at most $(n+1)2^n$. Therefore, the worst-case size of $\mathscr{B}$ is $2^{O(2^n n \log n)}$ and $\mathscr{B}$ has at most $3(n+1)2^n$ priorities. Theorem 2.21 now gives the desired upper bound.

The additional claim about the finite-state solution follows from Lemmas 2.16 and 2.17. For the lower bound, see [104]. Q.E.D.

In the remainder of this section, we present an alternative approach to solving the realizability problem, which is interesting in its own right.

Let φ be an instance of the realizability problem as above. Formally, a solution of φ is a function $f\colon [2]_m^+ \to [2]_n$. Such a function is the same as a $[2]_m$-branching $[2]_n$-labeled tree (where the root label is ignored). In other words, the set of all solutions of a given instance of the realizability problem is a tree language. This observation transforms the realizability problem into the framework of tree languages and tree automata, and we can apply tree-automata techniques to solve it.

Let $t\colon [2]_m^* \to [2]_n$ be any $[2]_m$-branching $[2]_n$-labeled tree. The tree can be turned into a potential solution to the instance φ if the label of the root is forgotten. The resulting function is denoted by $t_{-\varepsilon}$. We set $\mathscr{L}_{\text{sol}}(\varphi) = \{t\colon [2]_m^* \to [2]_n : t_{-\varepsilon} \text{ solves } \varphi\}$.

We next show that $\mathscr{L}_{\text{sol}}(\varphi)$ is a tree language which can be recognized by a universal tree automaton. We need, however, a more general notion of universal tree automaton as in Section 3.3. Also, we need to massage the formula φ a little to arrive at a simple automata-theoretic construction.

A universal co-Büchi tree automaton with set of directions D is a tuple

$$(A, D, Q, q_I, \Delta, F)$$

where A, Q, q_I, and F are as usual, and where D is a finite set of directions and $\Delta \subseteq Q \times A \times D \times Q$ is a transition relation. Following the definition from Section 3.3, a word $r \in Q^\omega$ is said to be a run for branch $u \in D^\omega$ if $(r(i), t(u[0,i)), u(i), r(i+1)) \in \Delta$ for every i and $r(0) = q_I$. A tree is accepted if every $r \in Q^\omega$ which is a run for some branch satisfies the co-Büchi acceptance condition. The latter means that $r(i) \in F$ only for finitely many i.

The technical problem one faces when constructing an automaton for $\mathscr{L}_{\text{sol}}(\varphi)$ is that a tree automaton has transitions of the form (q, a, d, q'), so, when applied to the above setting, in one transition the automaton consumes an output of the device we are looking for and the next input. For our construction it would be much better to have automata that in one transition consume an input and a corresponding output. Rather than modifying our standard automaton model, we resolve the issue on the logical side. For a given formula $\varphi = \varphi(p_0, \ldots, p_{m-1}, q_0, \ldots, q_{n-1})$ we consider the formula φ^{X} defined by

$$\varphi^{\mathsf{X}} = \varphi(p_0, \ldots, p_{m-1}, \mathsf{X}q_0, \ldots, \mathsf{X}q_{n-1}).$$

(Recall that X stands for the temporal operator "next".) This formula moves the output one position to the right, more precisely,

$$\mathscr{L}(\varphi) = \{d^0{}^\frown a^1 d^1{}^\frown a^2 \ldots : d^0 \frown a^0 d^1{}^\frown a^1 \cdots \in \mathscr{L}(\varphi^{\mathsf{X}})\}. \qquad (1.2)$$

Let $\varphi = \varphi(p_0, \dots, p_{m-1}, q_0, \dots, q_{n-1})$ be an instance of the LTL realizability problem and $\mathscr{A}$ a Büchi automaton such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\neg\varphi^{\mathsf{X}})$. The universal co-Büchi tree automaton for φ, denoted $\mathscr{A}^{\text{real}}[\varphi]$, is defined by

$$\mathscr{A}^{\text{real}}[\varphi] = ([2]_n, [2]_m, Q, q_I, \Delta', F)$$

where

$$\Delta' = \{(q, a, d, q') \colon (q, d^\frown a, q') \in \Delta\}.$$

FIGURE 22. From an LTL realizability instance to a universal tree automaton

A universal co-Büchi tree automaton for a given LTL formula φ as above is now easily constructed, as can be seen in Figure 22.

Lemma 4.13. Let $\varphi = \varphi(p_0, \dots, p_{m-1}, q_0, \dots, q_{n-1})$ be an instance of the LTL realizability problem. Then $\mathscr{L}(\mathscr{A}^{\text{real}}[\varphi]) = \mathscr{L}_{\text{sol}}(\varphi)$. Q.E.D.

Universal co-Büchi tree automata for D-branching trees as defined above are a special case of universal parity tree automata for D-branching trees, which can be turned into nondeterministic parity tree automata for D-branching trees in the same fashion as this was explained for automata on binary trees in Figure 16. The same is true for the emptiness test for parity tree automata on D-branching trees, which can be solved by constructing a parity game along the lines of the construction depicted in Figure 13 and solving this game.

4.5 Notes

The automata-theoretic decision procedure for LTL model checking described in this section has had a great practical impact, because it has been implemented in an industrial setting, see, for instance, [57], and used to verify real-world computing systems (mostly hardware). Much research has gone into improving the algorithm in several respects, but also into extending its applicability, for instance, more expressive logics and larger classes of devices have been looked at, see, for instance, [14, 25, 44, 70]. It is also noteworthy that LTL is the basis for industrial specification languages such as ForSpec [2] and PSL [34] and that the automata-theoretic approach underlies industrial implementations of specification languages [3].

An important aspect of this section is the use of alternating automata, which were introduced into the theory of automata on infinite objects by Muller and Schupp [90]. The only gain from this presented in the current section is Theorem 4.9, but this is probably the least important aspect in this context. What is more important is that weak alternating automata are as powerful as nondeterministic Büchi automata, which was proved by Kupferman and Vardi [72, 73]. This result motivated new research, which, for instance, brought about new complementation constructions [72, 73, 121]. As we see in the remaining two sections, alternation is even more important in the context of tree languages.

We refer to [123] for a collection of open algorithmic issues with regard to automata-theoretic LTL model checking.

5 Computation tree logic

Certain temporal properties of a system cannot be specified when runs of the system are considered separately, as we do this with LTL. For instance, when one wants to specify that no matter which state a system is in there is some way to get back to a default state, then this cannot be stated in LTL. The reason is that the property says something about how a run can evolve into different runs.

This observation motivates the introduction of specification logics that compensate for the lack of expressive power in this regard. The first logic of this type, called UB, was introduced by Ben-Ari, Manna, and Pnueli [7] in 1981. Another logic of this type is computation tree logic (CTL), designed by Clarke and Emerson [36], which is interpreted in the "computation tree" of a given transition system. This is the logic we study in this section, in particular, we study satisfiability and model checking for this logic.

Many of the proofs in this section are very similar to proofs in the previous section. In these cases, we only give sketches, but describe the differences in detail.

5.1 CTL and monadic second-order logic

CTL mixes path quantifiers and temporal operators in a way such that a logic arises for which model checking can be carried out in polynomial time. The syntax of CTL is as follows:

- tt and ff are CTL formulas,
- every propositional variable is a CTL formula,
- if φ is a CTL formula, then so is $\neg\varphi$,
- if φ and ψ are formulas, then so are $\varphi \vee \psi$, $\mathsf{E}(\varphi \,\mathsf{XU}\, \psi)$, and $\mathsf{A}(\varphi \,\mathsf{XU}\, \psi)$.

CTL formulas are interpreted in transition systems, which we introduce next. Such a system is a simple, state-based abstraction of a computing device. Formally, it is a tuple

$$\mathscr{S} = (P, S, \rightarrow, l)$$

where P is a finite set of propositional variables, S is a set of states, $\rightarrow \subseteq S \times S$ is a transition relation in infix notation, and $l\colon S \rightarrow 2^P$ is a labeling function assigning to each state which propositional variables are true in it. A computation of such a transition system starting in a state s is a word $u \in S^+ \cup S^\infty$ such that

(i) $u(0) = s$,

(ii) $u(i) \rightarrow u(i+1)$ for all i with $i + 1 < |u|$, and

(iii) u is maximal in the sense that if u is finite, then $u(*)$ must not have any successor.

Given a CTL formula φ, a transition system $\mathscr{S}$ over the same set P of propositional variables, and a state s of $\mathscr{S}$, it is defined whether φ holds true in $\mathscr{S}$ at s, denoted $\mathscr{S}, s \models \varphi$:

- $\mathscr{S}, s \models \mathsf{tt}$ and $\mathscr{S}, s \not\models \mathsf{ff}$,
- $\mathscr{S}, s \models p$ if $p \in l(s)$,
- $\mathscr{S}, s \models \neg\varphi$ if $\mathscr{S}, s \not\models \varphi$,
- $\mathscr{S}, s \models \psi \vee \chi$ if $\mathscr{S}, s \models \psi$ or $\mathscr{S}, s \models \chi$, for ψ and χ CTL formulas,
- $\mathscr{S}, s \models \mathsf{E}(\psi \,\mathsf{XU}\, \chi)$ if there exists a computation u of $\mathscr{S}$ starting at s and $j > 0$ such that $\mathscr{S}, u(j) \models \chi$ and $\mathscr{S}, u(i) \models \psi$ for all i with $0 < i < j$.
- $\mathscr{S}, s \models \mathsf{A}(\psi \,\mathsf{XU}\, \chi)$ if for all computations u of $\mathscr{S}$ starting at s there exists $j > 0$ such that $\mathscr{S}, u(j) \models \chi$ and $\mathscr{S}, u(i) \models \psi$ for all i with $0 < i < j$.

Just as with LTL, other operators can be defined:

- "in all computations always" is defined by $\mathsf{AG}\varphi = \varphi \wedge \neg\mathsf{E}(\mathsf{tt}\,\mathsf{XU}\,\neg\varphi)$,
- "in some computation eventually" is defined by $\mathsf{EF}\varphi = \varphi \vee \mathsf{E}(\mathsf{tt}\,\mathsf{XU}\,\varphi)$.

An interesting property one can express in CTL is the one discussed above, namely that from every state reachable from a given state a distinguished state, indicated by the propositional variable p_d, can be reached:

$$\mathsf{AG}\,\mathsf{EF}p_d. \tag{1.3}$$

Another property that can be expressed is that every request, indicated by the propositional variable p_r, is eventually acknowledged, indicated by the propositional variable p_a:

$$\mathsf{AG}(p_r \to \mathsf{AX}\,\mathsf{AF}p_a). \tag{1.4}$$

It is interesting to compare the expressive power of CTL with that of LTL. To this end, it is reasonable to restrict the considerations to infinite computations only and to say that a CTL formula φ and an LTL formula ψ are equivalent if for every transition system $\mathscr{S}$ and every state $s \in S$ the following holds: $\mathscr{S}, s \models \varphi$ iff $l(u(0))l(u(1))\ldots \models \psi$ for all infinite computations u of $\mathscr{S}$ starting in s.

The second property from above can be expressed easily in LTL, namely by the formula $\mathsf{G}(p_r \to \mathsf{XF}p_a)$, that is, this formula and (1.4) are equivalent. Clarke and Draghicescu showed that a CTL property is equivalent to some LTL formula if and only if it is equivalent to the LTL formula obtained by removing the path quantifiers [26]. But it is not true that every LTL formula which can be expressed in CTL is expressible by a CTL formula which uses universal path quantifiers only. This was shown by Bojanczyk [10].

An LTL formula which is not expressible in CTL is

$$\mathsf{GF}p, \tag{1.5}$$

which was already pointed out by Lamport [77].

In order to be able to recast satisfiability and model checking in a (tree) automata setting, it is crucial to observe that CTL formulas cannot distinguish between a transition system and the transition system obtained by "unraveling" it. Formally, the unraveling of the transition system $\mathscr{S}$ at state $s \in S$, denoted $\mathscr{T}_s(\mathscr{S})$, is the tree inductively defined by:

- s is the root of $\mathscr{T}_s(\mathscr{S})$,
- if $v \in S^+$ is an element of $V^{\mathscr{T}_s(\mathscr{S})}$ and $v(*) \to s'$, then $vs' \in V^{\mathscr{T}_s(\mathscr{S})}$ and $(v, vs') \in E^{\mathscr{T}_s(\mathscr{S})}$,
- $l^{\mathscr{T}_s(\mathscr{S})}(v) = l^{\mathscr{S}}(v(*))$ for every $v \in V^{\mathscr{T}_s(\mathscr{S})}$.

Henceforth, a tree with labels from 2^P, such as the unraveling of a transition system, is viewed as a transition system in the canonical way. When we interpret a CTL formula in a tree and do not indicate a vertex, then the formula is interpreted at the root of the tree.

The formal statement of the above observation can now be phrased as follows.

Lemma 5.1. For every CTL formula φ, transition system $\mathscr{S}$, and state $s \in S$,

$$\mathscr{S}, s \models \varphi \qquad \text{iff} \qquad \mathscr{T}_s(\mathscr{S}) \models \varphi.$$

Proof. This can be proved by a straightforward induction on the structure of φ, using a slightly more general claim:

$$\mathscr{S}, s' \models \varphi \qquad \text{iff} \qquad \mathscr{T}_s(\mathscr{S}), v \models \varphi$$

for every state $s' \in S$ and every vertex v of $\mathscr{T}_s(\mathscr{S})$ where $v(*) = s'$. Q.E.D.

The previous lemma says that we can restrict attention to trees, in particular, a CTL formula is satisfiable if and only if there is a tree which is a model of it. So when we translate CTL formulas into logics on trees which satisfiability is decidable for, then we also know that CTL satisfiability is decidable.

We present a simple translation of CTL into monadic second-order logic. There is, however, an issue to be dealt with: S2S formulas specify properties of binary trees, but CTL is interpreted in transition systems where each state can have more than just two successors. A simple solution is to use a variant of S2S which allows any number of successors but has only a single successor predicate, suc. Let us denote the resulting logic by SUS. As with LTL, we identify the elements of 2^P for $P = \{p_0, \dots, p_{n-1}\}$ with the elements of $[2]_n$.

Proposition 5.2. Let $P = \{p_0, \dots, p_{n-1}\}$ be an arbitrary finite set of propositional variables. For every CTL formula φ over P an SUS formula $\tilde{\varphi} = \tilde{\varphi}(X_0, \dots, X_{n-1})$ can be constructed such that $\mathscr{T} \models \varphi$ if and only if $\mathscr{T} \models \tilde{\varphi}$ for all trees $\mathscr{T}$ over 2^P (or $[2]_n$).

Proof. What we actually prove is somewhat stronger, analogous to the proof for LTL. We construct a formula $\hat{\varphi} = \hat{\varphi}(X_0, \dots, X_{n-1}, x)$ such that $\mathscr{T}, v \models \varphi$ if and only if $\mathscr{T}, v \models \hat{\varphi}$ for all trees $\mathscr{T}$ and $v \in V^{\mathscr{T}}$. We can then set $\tilde{\varphi} = \exists x(\varphi_{root}(x) \wedge \hat{\varphi})$ where $\varphi_{root}(x) = \forall y(\neg \mathrm{suc}(y, x))$ specifies that x is the root.

For the induction base, assume $\varphi = p_i$. We can set $\hat{\varphi}$ to $x \in X_i$. Similarly, for $\varphi = \neg p_i$ we can set $\hat{\varphi}$ to $\neg x \in X_i$.

In the inductive step, we consider only one of the interesting cases, namely where $\varphi = \mathsf{A}(\psi \,\mathsf{XU}\, \chi)$. We start with a formula $\varphi_{closed} = \varphi_{closed}(X)$

which is true if every element of X has a successor in X provided it has a successor at all:

$$\varphi_{closed} = \forall x(x \in X \wedge \exists y(\mathrm{suc}(x,y)) \to \exists y(\mathrm{suc}(x,y) \wedge y \in X)).$$

We next write a formula $\varphi_{path}(x,X)$ which is true if X is a maximum path starting in x:

$$\begin{aligned}\varphi_{path} = x \in X \wedge \varphi_{closed}(X) \wedge& \\ &\forall Y(x \in Y \wedge \varphi_{closed}(Y) \wedge Y \subseteq X \to X = Y).\end{aligned}$$

We can then set

$$\begin{aligned}\hat{\varphi} = \forall X(\varphi_{path}(x,X) \to& \\ &\exists z(z \in X \wedge \neg z = x \wedge \hat{\chi}(z) \wedge \forall y(x < y < z \to \hat{\psi}(y))).\end{aligned}$$

The other CTL operators can be dealt with in a similar fashion. Q.E.D.

The desired decidability result now follows from the following result on SUS.

Theorem 5.3 (Walukiewicz, [128])**.** SUS satisfiability is decidable.

This result can be proved just as we proved the decidability of satisfiability for S2S, that is, using an analogue of Rabin's Theorem. This analogue will use a different kind of tree automaton model which takes into account that the branching degree of the trees considered is unbounded and that there is one predicate for all successors. More precisely, a transition in such an automaton is of the form $(q, a, Q^{\mathsf{E}}, Q^{\mathsf{A}})$ where $Q^{\mathsf{E}}, Q^{\mathsf{A}} \subseteq Q$. Such a transition is to be read as follows: If the automaton is in state q at a vertex labeled a, then for every $q' \in Q^{\mathsf{E}}$ there exists exactly one successor that gets assigned q' and all the successors that do not get assigned any state in this fashion get assigned exactly one state from Q^{A}. In particular, if $Q^{\mathsf{E}} = Q^{\mathsf{A}} = \varnothing$, then the vertex must not have a successor. In [128], Walukiewicz actually presents a theorem like Büchi's and Rabin's: He shows that there is a translation in both directions, from SUS formulas to such automata and back.

Corollary 5.4. CTL satisfiability and model checking are decidable.

That model checking is decidable follows from the simple observation that in SUS one can define the unraveling of every finite transition system.

We conclude this introduction to CTL with further remarks on SUS and its relationship to CTL. There is a logic related to SUS which was already studied by Rabin and which he denoted SωS. This is the logic interpreted in the countably branching tree ω^* where, for each i, there is a separate successor relation $\mathrm{suc}_i(\cdot,\cdot)$. Observe that—as noted in [60]—in this logic one cannot even express that all successors of the root belong to a certain set, which can easily be expressed in CTL and SUS.

Observe, too, that in SUS one can express that every vertex of a tree has at least two successors, namely by

$$\forall x(\exists y_0 \exists y_1(\mathrm{suc}(x,y_0) \wedge \mathrm{suc}(x,y_1) \wedge \neg y_0 = y_1).$$

This is, however, impossible in CTL. More precisely, CTL cannot distinguish between bisimilar transition systems whereas SUS can do this easily.

5.2 From CTL to nondeterministic tree automata

We next show how to arrive at good complexity bounds for satisfiability and model checking by following a refined automata-theoretic approach. For satisfiability, we can use nondeterministic automata and vary the approach we used for handling LTL in Section 4, while for model checking, we have to use alternating tree automata.

As pointed out above, the nondeterministic tree automaton model we defined in Section 3 was suited for binary trees only, which is not enough in the context of CTL. Here, we need an automaton model that can handle trees with arbitrary branching degree. We could use the tree automaton model explained in Section 5.1, but there is another model which is more appropriate. Following Janin and Walukiewicz [59], we use a tree automaton model which takes into account that properties like the one mentioned at the end of Section 5.1 cannot be expressed in CTL.

A generalized Büchi tree automaton in this context is a tuple

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

where A, Q, Q_I, and $\mathscr{F}$ are as with generalized Büchi (word) automata and $\Delta \subseteq Q \times A \times 2^Q \times 2^Q$ is a transition relation.

A transition of the form $(q, a, Q^{\mathsf{E}}, Q^{\mathsf{A}})$ is to be read as follows: If the automaton is in state q at vertex v and reads the label a, then it sends each state from Q^{E} to at least one of the successors of v and every successor of v is at least sent one of the states from $Q^{\mathsf{E}} \cup Q^{\mathsf{A}}$; the same successor can get sent several states.

Formally, a run of $\mathscr{A}$ on a tree $\mathscr{T}$ is a $(Q \times V^{\mathscr{T}})$-labeled tree $\mathscr{R}$ satisfying the following conditions.

(i) The root of $\mathscr{R}$ is labeled $(q, \mathrm{root}(\mathscr{T}))$ for some $q \in Q_I$.

(ii) For every vertex $w \in V^{\mathscr{R}}$, if (q, v) is the label of w, then there exists a transition $(q, l^{\mathscr{R}}(v), Q^{\mathsf{E}}, Q^{\mathsf{A}}) \in \Delta$ such that:

(a) For every $v' \in \mathrm{sucs}^{\mathscr{T}}(v)$ there exists $w' \in \mathrm{sucs}^{\mathscr{R}}(w)$ labeled (q', v') for some $q' \in Q^{\mathsf{E}} \cup Q^{\mathsf{A}}$, that is, every successor of v occurs in a label of a successor of w.

(b) For every $q' \in Q^{\mathsf{E}}$ there exist $v' \in \mathrm{sucs}^{\mathscr{T}}(v)$ and $w' \in \mathrm{sucs}^{\mathscr{R}}(w)$ such that w' is labeled (q', v). That is, every state from Q^{E} occurs at least once among all successors of w.

Such a run is accepting if every branch is accepting with respect to the given generalized Büchi condition just as this was defined for generalized Büchi word automata.

Observe that in this model the unlabeled tree underlying a run may not be the same as the unlabeled tree underlying a given input tree. Copies of subtrees may occur repeatedly.

As an example, let $P = \{p\}$ and $A = 2^P$ and consider the tree language L which contains all trees over A that satisfy the property that every branch is either finite or labeled $\{p\}$ infinitely often. An appropriate Büchi automaton has two states, $q_\varnothing$ and $q_{\{p\}}$, where $q_\varnothing$ is initial and $q_{\{p\}}$ is final, and the transitions are $(q, a, \{q_a\})$ and $(q, a, \varnothing, \varnothing)$ for any state q and letter a.

The idea for translating a given CTL formula into a nondeterministic tree automaton follows the translation of LTL into nondeterministic word automata: In each vertex, the automaton guesses which subformulas of the given formula are true and verifies this. The only difference is that the path quantifiers E and A are taken into account, which is technically somewhat involved. The details are given in Figure 23, where the following notation and terminology is used. Given a set Ψ of CTL formulas over a finite set P of propositional variables and a letter $a \in 2^P$ we say that Ψ is consistent with a if

- $\mathsf{ff} \notin \Psi$,
- $p \in \Psi$ iff $p \in a$, for all $p \in P$, and
- for $\psi \in \Psi$, if $\psi = \psi_0 \vee \psi_1$, then $\psi_i \in \Psi$ for some $i < 2$, and if $\psi = \psi_0 \wedge \psi_1$, then $\{\psi_0, \psi_1\} \subseteq \Psi$.

Further, a set Ψ' is a witness for $\mathsf{E}(\psi \mathsf{XU} \chi)$ if $\chi \in \Psi'$ or $\{\psi, \mathsf{E}(\psi \mathsf{XU} \chi)\} \subseteq \Psi'$. Similarly, Ψ' is a witness for $\mathsf{E}(\psi \mathsf{XR} \chi)$ if $\{\psi, \chi\} \subseteq \Psi'$ or $\{\chi, \mathsf{E}(\psi \mathsf{XR} \chi)\} \subseteq \Psi'$. The analogue terminology is used for A-formulas. When Ψ is a set of CTL formulas, then Ψ_{E} denotes the formulas of the form $\mathsf{E}(\psi \mathsf{XU} \chi)$ and $\mathsf{E}(\psi \mathsf{XR} \chi)$, that is, the set of all E-formulas in Ψ, and, similarly, Ψ_{A} denotes the set of all A-formulas in Ψ.

The only interesting aspect of the construction is (iv) of the definition of a transition. It would be more natural to omit (iv), and, indeed, the construction would then also be correct, but the resulting automaton would

Let P be a finite set of propositional variables and φ a CTL formula over P in positive normal form. The generalized Büchi tree automaton for φ with respect to P, denoted $\mathscr{A}[\varphi]$, is defined by

$$\mathscr{A}[\varphi] = (2^P, 2^{\mathrm{sub}(\varphi)}, Q_I, \Delta, \mathscr{F})$$

where $Q_I = \{\Psi \subseteq 2^{\mathrm{sub}(\varphi)} : \varphi \in \Psi\}$ and

$$\mathscr{F} = \{F_{\mathsf{Q}[\psi \mathsf{XU} \chi]} : \mathsf{Q}[\psi \,\mathsf{XU}\, \chi] \in \mathrm{sub}(\varphi) \text{ and } \mathsf{Q} \in \{\mathsf{E}, \mathsf{A}\}\}$$

with

$$F_{\mathsf{Q}[\psi \mathsf{XU} \chi]} = \{\Psi \subseteq \mathrm{sub}(\varphi) : \chi \in \Psi \text{ or } \mathsf{Q}[\psi \,\mathsf{XU}\, \chi] \notin \Psi\},$$

and where Δ contains a transition $(\Psi, a, Q^{\mathsf{E}}, Q^{\mathsf{A}})$ if the following conditions are satisfied:

(i) Ψ is consistent with a,

(ii) for every $\psi \in \Psi_{\mathsf{E}}$ there exists $\Psi' \in Q^{\mathsf{E}}$ which witnesses it and Q^{A} contains all $\Psi \subseteq \mathrm{sub}(\varphi)$ that contain a witness for every $\psi \in \Psi_{\mathsf{A}}$,

(iii) every $\Psi' \in Q^{\mathsf{E}}$ witnesses every $\psi \in \Psi_{\mathsf{A}}$,

(iv) $|Q^{\mathsf{E}}| \leq |\mathrm{sub}(\varphi)_{\mathsf{E}}|$.

FIGURE 23. From CTL to generalized Büchi tree automata

be too large. On the other hand, (iv) is not a real restriction, because the semantics of CTL requires only one "witness" for every existential path formula.

Before formally stating the correctness of the construction, we introduce a notion referring to the number of different states which can be assigned in a transition. We say that a nondeterministic tree automaton $\mathscr{A}$ is m-bounded if $|Q^{\mathsf{E}}| \leq m$ holds for every $(q, a, Q^{\mathsf{E}}, Q^{\mathsf{A}}) \in \Delta$.

Lemma 5.5. Let φ be an arbitrary CTL formula with n subformulas, m E-subformulas, and k U-subformulas. Then $\mathscr{A}[\varphi]$ is an $(m+1)$-bounded generalized Büchi tree automaton with 2^n states, k acceptance sets, and

Let $\mathscr{A}$ be a nondeterministic Büchi tree automaton. The emptiness game for $\mathscr{A}$, denoted $\mathscr{G}_\varnothing[\mathscr{A}]$, is defined by

$$\mathscr{G}_\varnothing[\mathscr{A}] = (Q, \Delta, q_I, M_0 \cup M_1, F)$$

where

$$M_0 = \{(q, Q^{\mathsf{E}}, Q^{\mathsf{A}}) \colon \exists a \exists Q^{\mathsf{A}}((q, a, Q^{\mathsf{E}}, Q^{\mathsf{A}}) \in \Delta)\}, \text{ and}$$
$$M_1 = \{(Q', q) \colon q \in Q'\}.$$

FIGURE 24. Emptiness game for nondeterministic Büchi tree automaton

such that $\mathscr{L}(\mathscr{A}[\varphi]) = \mathscr{L}(\varphi)$.

Proof sketch. The claim about the size of the automaton is trivial. The proof of its correctness can be carried out similar to the proof of Theorem 4.4, that is, one proves $\mathscr{L}(\mathscr{A}[\varphi]) \subseteq \mathscr{L}(\varphi)$ by induction on the structure of φ and $\mathscr{L}(\varphi) \subseteq \mathscr{L}(\mathscr{A}[\varphi])$ by constructing an accepting run directly. Q.E.D.

It is very easy to see that the construction from Figure 18 can also be used in this context to convert a generalized Büchi tree automaton into a Büchi automaton. To be more precise, an m-bounded generalized Büchi tree automaton with n states and k acceptance sets can be converted into an equivalent m-bounded Büchi tree automaton with $(k+1)n$ states.

So in order to solve the satisfiability problem for CTL we only need to solve the emptiness problem for Büchi tree automata in this context. There is a simple way to perform an emptiness test for nondeterministic tree automata, namely by using the same approach as for nondeterministic tree automata working on binary trees: The nonemptiness problem is phrased as a game. Given a nondeterministic Büchi tree automaton $\mathscr{A}$, we define a game which Player 0 wins if and only if some tree is accepted by $\mathscr{A}$. To this end, Player 0 tries to suggest suitable transitions while Player 1 tries to argue that Player 0's choices are not correct. The details of the construction are given in Figure 24.

Lemma 5.6. Let $\mathscr{A}$ be a nondeterministic Büchi tree automaton. Then the following are equivalent:

(A) $\mathscr{L}(\mathscr{A}) \neq \varnothing$.

(B) Player 0 wins $\mathscr{G}_\varnothing[\mathscr{A}]$.

Proof. The proof of the lemma can be carried out along the lines of the proof of Lemma 3.3. The only difference is due to the arbitrary branching degree, which can easily be taken care of. One only needs to observe that if there exists a tree which is accepted by $\mathscr{A}$, then there is a tree with branching degree at most $|Q|$ which is accepted. Q.E.D.

We have the following theorem:

Theorem 5.7 (Emerson-Halpern-Fischer-Ladner, [37, 45])**.** CTL satisfiability is complete for deterministic exponential time.

Proof. The decision procedure is as follows. A given CTL formula φ is first converted into an equivalent generalized Büchi tree automaton $\mathscr{A}$ using the construction from Figure 23. Then $\mathscr{A}$ is converted into an equivalent Büchi tree automaton $\mathscr{B}$ using the natural adaptation of the construction presented in Figure 18 to trees. In the third step, $\mathscr{B}$ is converted into the Büchi game $\mathscr{G}_\varnothing[\mathscr{B}]$, and, finally, the winner of this game is determined. (Recall that a Büchi condition is a parity condition with two different priorities.)

From Theorem 2.21 on the complexity of parity games it follows immediately that Büchi games (parity games with two different priorities) can be solved in polynomial time, which means we only need to show that the size of $\mathscr{G}_\varnothing[\mathscr{B}]$ is exponential in the size of the given formula φ and can be constructed in exponential time. The latter essentially amounts to showing that $\mathscr{B}$ is of exponential size.

Let n be the number of subformulas of φ. Then $\mathscr{A}$ is n-bounded with 2^n states and at most n acceptance sets. This means that the number of sets Q' occurring in the transitions of $\mathscr{A}$ is at most 2^{n^2}, so there are at most 2^{n^2+2n} transitions (recall that there are at most 2^n letters in the alphabet). Similarly, $\mathscr{B}$ is n-bounded, has at most $(n+1)2^n$ states, and $2^{O(n^2)}$ transitions.

The lower bound is given in [45]. Q.E.D.

5.3 From CTL to alternating tree automata

One of the crucial results of Emerson and Clarke on CTL is that model checking of CTL can be carried out in polynomial time. The decision procedure they suggested in [28] is a simple labeling algorithms. For every subformula ψ of a given formula φ they determine in which states of a given transition system $\mathscr{S}$ the formula ψ holds and in which it does not hold. This is trivial for atomic formulas. It is straightforward for conjunction and disjunction, provided it is known which of the conjuncts and disjuncts, respectively, hold. For XR- and XU-formulas, it amounts to simple graph searches.

Emerson and Clarke's procedure cannot easily be seen as a technique which could also be derived following an automata-theoretic approach. Consider the nondeterministic tree automaton we constructed in Figure 23. Its size is exponential in the size of the given formula (and this cannot be avoided), so it is unclear how using this automaton one can arrive at a polynomial-time procedure.

The key for developing an automata-theoretic approach, which is due to Kupferman, Vardi, and Wolper [71], is to use alternating tree automata similar to how we used alternating automata for LTL in Section 4 and to carefully analyze their structure.

An alternating Büchi tree automaton is a tuple

$$\mathscr{A} = (P, Q, q_I, \delta, F)$$

where P, Q, q_I, and F are as usual and δ is the transition function which assigns to each state a transition condition. The set of transition conditions over P and Q, denoted $\mathrm{TC}(P, Q)$, is the smallest set such that

(i) $\mathsf{tt}, \mathsf{ff} \in \mathrm{TC}(P, Q)$,

(ii) $p, \neg p \in \mathrm{TC}(P, Q)$ for every $p \in P$,

(iii) every positive boolean combination of states is in $\mathrm{TC}(P, Q)$,

(iv) $\Diamond\gamma, \Box\gamma \in \mathrm{TC}(P, Q)$ where γ is a positive boolean combination of states.

This definition is very similar to the definition for alternating automata on words. The main difference reflects that in a tree a "position" can have several successors: $\Diamond$ expresses that a copy of the automaton should be sent to one successor, while $\Box$ expresses that a copy of the automaton should be sent to all successors. So $\Diamond$ and $\Box$ are the two variants of $\bigcirc$.

There is another, minor difference: For tree automata, we allow positive boolean combinations of states in the scope of $\Diamond$ and $\Box$. We could have allowed this for word automata, too, but it would not have helped us. Here, it makes our constructions simpler, but the proofs will be slightly more involved.

Let $\mathscr{T}$ be a 2^P-labeled tree. A tree $\mathscr{R}$ with labels from $\mathrm{TC}(P, Q) \times V^{\mathscr{T}}$ is a run of $\mathscr{A}$ on $\mathscr{T}$ if $l^{\mathscr{R}}(\mathrm{root}(\mathscr{R})) = (q_I, \mathrm{root}(\mathscr{T}))$ and the following conditions are satisfied for every vertex $w \in V^{\mathscr{R}}$ with label (γ, v):

- $\gamma \neq \mathsf{ff}$,
- if $\gamma = p$, then $p \in l^{\mathscr{T}}(w)$, and if $\gamma = \neg p$, then $p \notin l^{\mathscr{T}}(w)$,

- if $\gamma = \Diamond\gamma'$, then there exists $v' \in \text{sucs}^{\mathscr{T}}(v)$ and $w' \in \text{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = (\gamma', v')$,
- if $\gamma = \Box\gamma'$, then for every $v' \in \text{sucs}^{\mathscr{T}}(v)$ there exists $w' \in \text{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = (\gamma', v')$,
- if $\gamma = \gamma_0 \vee \gamma_1$, then there exists $i < 2$ and $w' \in \text{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = (\gamma_i, v)$,
- if $\gamma = \gamma_0 \wedge \gamma_1$, then for every $i < 2$ there exists $w' \in \text{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = (\gamma_i, v)$.

Such a run is accepting if on every infinite branch there exist infinitely many vertices w labeled with an element of F in the first component.

The example language from above can be recognized by an alternating Büchi automaton which is slightly more complicated than the nondeterministic automaton, because of the restrictive syntax for transition conditions. We use the same states as above and four further states, q, $q'_{\{p\}}$, $q_\perp$, and $q'_\perp$. The transition function is determined by

$$\begin{aligned} \delta(q_I) &= q_\perp \vee q, \\ \delta(q_{\{p\}}) &= q'_{\{p\}} \wedge (q_\perp \vee q), & \delta(q'_{\{p\}}) &= p, \\ \delta(q_\perp) &= \Box q'_\perp, & \delta(q'_\perp) &= \mathsf{ff}, \\ \delta(q) &= \Box(q_I \vee q_{\{p\}}). \end{aligned}$$

The state $q_\perp$ is used to check that the automaton is at a vertex without successor.

In analogy to the construction for LTL, we can now construct an alternating tree automaton for a given CTL formula. This construction is depicted in Figure 25.

Compared to the construction for LTL, there are the following minor differences. First, the definition of the transition function is no longer inductive, because we allow positive boolean combinations in the transition function. Second, we have positive boolean combinations of states in the scope of $\Diamond$ and $\Box$. This was not necessary with LTL, but it is necessary here. For instance, if we instead had $\delta([\mathsf{E}(\psi \mathsf{XU} \chi)]) = \Diamond[\chi] \vee (\Diamond[\psi] \wedge \Diamond[\mathsf{E}(\psi \mathsf{XU} \chi)])$, then this would clearly result in a false automaton because of the second disjunct.

We can make a similar observation as with the alternating automata that we constructed for LTL formulas. The automata are very weak in the sense that when we turn the subformula ordering into a linear ordering $\leq$ on the states, then for each state q, the transition conditions $\delta(q)$ contains only states q' such that $q \geq q'$.

Let φ be a CTL formula in positive normal form over P and Q the set which contains for each $\psi \in \text{sub}(\varphi)$ an element denoted $[\psi]$. The automaton $\mathscr{A}^{\text{alt}}[\varphi]$ is defined by

$$\mathscr{A}^{\text{alt}}[\varphi] = (P, Q, [\varphi], \delta, F)$$

where

$$\begin{aligned} \delta([\mathsf{tt}]) &= \mathsf{tt}, & \delta([\mathsf{ff}]) &= \mathsf{ff}, \\ \delta([p]) &= p, & \delta([\neg p]) &= \neg p, \\ \delta([\psi \vee \chi]) &= [\varphi] \vee [\psi], & \delta([\psi \wedge \chi]) &= [\varphi] \wedge [\psi], \end{aligned}$$

$$\begin{aligned} \delta([\mathsf{E}(\psi \,\mathsf{XU}\, \chi)]) &= \Diamond([\chi] \vee ([\psi] \wedge [\mathsf{E}(\psi \,\mathsf{XU}\, \chi)])), \\ \delta([\mathsf{E}(\psi \,\mathsf{XR}\, \chi)]) &= \Diamond([\chi] \wedge ([\chi] \vee [\mathsf{E}(\psi \,\mathsf{XR}\, \chi)])), \\ \delta([\mathsf{A}(\psi \,\mathsf{XU}\, \chi)]) &= \Box([\chi] \vee ([\psi] \wedge [\mathsf{A}(\psi \,\mathsf{XU}\, \chi)])), \\ \delta([\mathsf{A}(\psi \,\mathsf{XR}\, \chi)]) &= \Box([\chi] \wedge ([\chi] \vee [\mathsf{A}(\psi \,\mathsf{XR}\, \chi)])), \end{aligned}$$

and F contains all the elements $[\psi]$ where ψ is not an XU-formula.

FIGURE 25. From CTL to alternating tree automata

Lemma 5.8 (Kupferman-Vardi-Wolper, [71])**.** Let φ be a CTL formula with n subformulas. The automaton $\mathscr{A}^{\text{alt}}[\varphi]$ is a very weak alternating tree automaton with n states and such that $\mathscr{L}(\mathscr{A}^{\text{alt}}[A]) = \mathscr{L}(\varphi)$.

Proof. The proof can follow the lines of the proof of Lemma 4.7. Since the automaton is very weak, a simple induction on the structure of the formula can be carried out, just as in the proof of Lemma 4.7. Branching makes the proof only technically more involved, no new ideas are necessary to carry it out. Q.E.D.

As pointed out above, it is not our goal to turn $\mathscr{A}^{\text{alt}}[\varphi]$ into a nondeterministic automaton (although this is possible), because such a translation cannot be useful for solving the model checking problem. What we rather do is to define a product of an alternating automaton with a transition system, resulting in a game, in such a way that the winner of the product of $\mathscr{A}^{\text{alt}}[\varphi]$ with some transition system $\mathscr{S}$ reflects whether φ holds true in a certain state s_I of $\mathscr{S}$.

The idea is that a position in this game is of the form (γ, s) where γ is a transition condition and s is a state of the transition system. The goal is to design the game in such a way that Player 0 wins the game starting from (q_I, s_I) if and only if there exists an accepting run of the automaton on the unraveling of the transition system starting at s_I. This means, for instance, that if γ is a disjunction, then we make the position (γ, s) a position for Player 0, because by moving to one of the two successor positions he should show which of the disjuncts holds. If, on the other hand, $\gamma = \Box\gamma'$, then we make the position a position for Player 1, because she should be able to challenge Player 0 with any successor of s. The details are spelled out in Figure 26, where the following notation and terminology is used. Given an alternating automaton $\mathscr{A}$, we write $\mathrm{sub}(\mathscr{A})$ for the set of subformulas of the values of the transition function of $\mathscr{A}$. In addition, we write $\mathrm{sub}^+(\mathscr{A})$ for the set of all $\gamma \in \mathrm{sub}(\mathscr{A})$ where the maximum state occurring belongs to the set of final states.

Assume $\mathscr{A}$ is a very weak alternating Büchi automaton. Then $\mathscr{A} \times_{s_I} \mathscr{S}$ is not very weak in general in the sense that the game graph can be extended to a linear ordering. Observe, however, that the following is true for every position (q, s): All states in the strongly connected component of (q, s) are of the form (γ, s') where q is the largest state occurring in γ. So, by definition of $\mathscr{A} \times_{s_I} \mathscr{S}$, all positions in a strongly connected component of $\mathscr{A} \times_{s_I} \mathscr{S}$ are either final or nonfinal. We turn this into a definition. We say that a Büchi game is weak if for every strongly connected component of the game graph it is true that either all its positions are final or none of them is.

Lemma 5.9. Let $\mathscr{A}$ be an alternating Büchi tree automaton, $\mathscr{S}$ a transition system over the same finite set of propositional variables, and $s_I \in S$. Then $\mathscr{T}_{s_I}(\mathscr{S}) \in \mathscr{L}(\mathscr{A})$ iff Player 0 wins $\mathscr{A} \times_{s_I} \mathscr{S}$. Moreover, if $\mathscr{A}$ is a very weak alternating automaton, then $\mathscr{A} \times_{s_I} \mathscr{S}$ is a weak game.

Proof. The additional claim is obvious. For the other claim, first assume $\mathscr{R}$ is an accepting run of $\mathscr{A}$ on $\mathscr{T}_{s_I}(\mathscr{S})$. We convert $\mathscr{R}$ into a winning strategy σ for Player 0 in $\mathscr{A} \times \mathscr{S}$. To this end, let w be a vertex of $\mathscr{R}$ with label (γ, v) such that (γ, v) is a position for Player 0. Since $\mathscr{R}$ is an accepting run, w has a successor, say w'. Assume $l^{\mathscr{R}}(w') = (\gamma', v')$. We set $\sigma(u) = (\gamma, v'(*))$ where u is defined as follows. First, let $n = |v|$. Assume $l^{\mathscr{R}}(u(i)) = (\gamma_i, v_i)$ for every $i < n$. We set $u = (\gamma_0, v_0(*))(\gamma_1, v_1(*)) \dots (\gamma_{n-1}, v_{n-1}(*))$. It can be shown that this defines a strategy. Moreover, since $\mathscr{R}$ is accepting, σ is winning.

For the other direction, a winning strategy is turned into an accepting run in a similar manner. Q.E.D.

The proof shows that essentially there is no difference between a run and a strategy—one can think of a run as a strategy. From this point of

Let $\mathscr{A}$ be an alternating Büchi tree automaton, $\mathscr{S}$ a transition system over the same set of propositional variables, and $s_I \in S$. The product of $\mathscr{A}$ and $\mathscr{S}$ at s_I, denoted $\mathscr{A} \times_{s_I} \mathscr{S}$, is the Büchi game defined by

$$\mathscr{A} \times_{s_I} \mathscr{S} = (P_0, P_1, (q_I, s_I), M, \mathrm{sub}^+(\mathscr{A}) \times S)$$

where

- P_0 is the set of pairs $(\gamma, s) \in \mathrm{sub}(\mathscr{A}) \times S$ where γ is
  - (i) a disjunction,
  - (ii) a $\Diamond$-formula,
  - (iii) p for $p \notin l(s)$,
  - (iv) $\neg p$ for $p \in l(s)$, or
  - (v) ff,

  and
- P_1 is the set of pairs $(\gamma, s) \in \mathrm{sub}(\mathscr{A}) \times S$ where γ is
  - (i) a conjunction,
  - (ii) a $\Box$-formula,
  - (iii) p for some $p \in l(s)$,
  - (iv) $\neg p$ for some $p \notin l(s)$, or
  - (v) tt.

Further, M contains for every $\gamma \in \mathrm{sub}(\mathscr{A})$ and every $s \in S$ moves according to the following rules:

- if $\gamma = q$ for some state q, then $((\gamma, s), (\delta(q), s)) \in M$,
- if $\gamma = \gamma_0 \vee \gamma_1$ or $\gamma = \gamma_0 \wedge \gamma_1$, then $((\gamma, s), (\gamma_i, s)) \in M$ for $i < 2$,
- if $\gamma = \Diamond\gamma'$ or $\gamma = \Box\gamma'$, then $((\gamma, s), (\gamma', s')) \in M$ for all $s' \in \mathrm{sucs}^{\mathscr{S}}(s)$.

Figure 26. Product of a transition system and an alternating automaton

view, an alternating automaton defines a family of games, for each tree a separate game, and the tree language recognized by the tree automaton is the set of all trees which Player 0 wins the game for.

The additional claim in the above lemma allows us to prove the desired complexity bound for the CTL model checking problem:

Theorem 5.10 (Clarke-Emerson-Sistla, [28])**.** The CTL model checking problem can be solved in time $O(mn)$ where m is the size of the transition system and n the number of subformulas of the CTL formula.

Proof. Consider the following algorithm, given a CTL formula φ, a transition system $\mathscr{S}$, and a state $s_I \in S$. First, construct the very weak alternating Büchi automaton $\mathscr{A}^{\mathrm{alt}}[\varphi]$. Second, build the product $\mathscr{A}^{\mathrm{alt}}[\varphi] \times_{s_I} \mathscr{S}$. Third, solve $\mathscr{A}^{\mathrm{alt}}[\varphi] \times_{s_I} \mathscr{S}$. Then Player 0 is the winner if and only if $\mathscr{S}, s_I \models \varphi$.

The claim about the complexity follows from the fact that the size of $\mathscr{A}^{\mathrm{alt}}[\varphi] \times_{s_I} \mathscr{S}$ is mn and from Theorem 2.21. Note that weak games are parity games with one priority in each strongly connected component. Q.E.D.

Obviously, given a CTL formula φ, a transition system $\mathscr{S}$, and a state s_I one can directly construct a game that reflects whether $\mathscr{S}, s_I \models \varphi$. This game would be called the model checking game for $\mathscr{S}$, s_I, and φ. The construction via the alternating automaton has the advantage that starting from this automaton one can solve both, model checking and satisfiability, the latter by using a translation from alternating Büchi tree automata into nondeterministic tree automata. We present such a translation in Section 6.

The translation from CTL into very weak alternating automata has another interesting feature. Just as the translation from LTL to weak alternating automata, it has a converse. More precisely, following the lines of the proof of Theorem 4.9, one can prove:

Theorem 5.11. Every very weak alternating tree automaton is equivalent to a CTL formula. Q.E.D.

5.4 Notes

The two specification logics that we have dealt with, LTL and CTL, can easily be combined into a single specification logic. This led Emerson and Halpern to introduce CTL* in 1986 [38].

An automata-theoretic proof of Corollary 5.7 was given first by Vardi and Wolper in 1986 [125]. Kupferman, Vardi, and Wolper, when proposing an automata-theoretic approach to CTL model checking in [71], also showed how other model checking problems can be solved following the automata-theoretic paradigm. One of their results is that CTL model checking can be solved in space polylogarithmic in the size of the transition system.

6 Modal μ-calculus

The logics that have been discussed thus far—S1S, S2S, LTL, and CTL—could be termed declarative in the sense that they are used to *describe* properties of sequences, trees, or transition systems rather than to specify how it can be determined whether such properties hold. This is different for the logic we discuss in this section, the modal μ-calculus (MC), introduced by Kozen in 1983 [66]. This calculus has a rich and deep mathematical and algorithmic theory, which has been developed over more than 20 years. Fundamental work on it has been carried out by Emerson, Streett, and Jutla [114, 40], Walukiewicz [129], Bradfield and Lenzi [79, 11], and others, and it has been treated extensively in books, for instance, by Arnold and Niwiński [6] and Stirling [110]. In this section, we study satisfiability (and model checking) for MC from an automata-theoretic perspective. Given that MC is much more complex than LTL or CTL, our exposition is less detailed, but gives a good impression of how the automata-theoretic paradigm works for MC.

6.1 MC and monadic second-order logic

MC is a formal language consisting of expressions which are evaluated in transition systems; every closed expression (without free variables) is evaluated to a set of states. The operations available for composing sets of states are boolean operations, local operations, and fixed point operations.

Formally, the set of MC expressions is the smallest set containing

- p and $\neg p$ for any propositional variable p,
- any fixed-point variable X,
- $\varphi \wedge \psi$ and $\varphi \vee \psi$ if φ and ψ are MC expressions,
- $\langle\,\rangle\varphi$ and $[\,]\varphi$ if φ is an MC expression, and
- $\mu X \varphi$ and $\nu X \varphi$ if X is a fixed-point variable and φ an MC expression.

The operators μ and ν are viewed as quantifiers in the sense that one says they bind the following variable. As usual, an expression without free occurrences of variables is called closed. The set of all variables occurring free in an MC expression φ is denoted by free(φ). An expression is called a fixed-point expression if it starts with μ or ν.

To define the semantics of MC expressions, let φ be an MC expression over some finite set P of propositional variables, $\mathscr{S}$ a transition system, and α a variable assignment which assigns to every fixed-point variable a set of states of $\mathscr{S}$. The value of φ with respect to $\mathscr{S}$ and α, denoted $||\varphi||^{\alpha}_{\mathscr{S}}$, is defined as follows. The fixed-point variables and the propositional variables are interpreted according to the variable assignment and the transition system:

$$||p||^\alpha_{\mathscr{S}} = \{s \in S^{\mathscr{S}} : p \in l^{\mathscr{S}}(s)\}, \qquad ||\neg p||^\alpha_{\mathscr{S}} = \{s \in S^{\mathscr{S}} : p \notin l^{\mathscr{S}}(s)\},$$

and

$$||X||^\alpha_{\mathscr{S}} = \alpha(X).$$

Conjunction and disjunction are translated into union and intersection:

$$||\varphi \wedge \psi||^\alpha_{\mathscr{S}} = ||\varphi||^\alpha_{\mathscr{S}} \cap ||\psi||^\alpha_{\mathscr{S}}, \qquad ||\varphi \vee \psi||^\alpha_{\mathscr{S}} = ||\varphi||^\alpha_{\mathscr{S}} \cup ||\psi||^\alpha_{\mathscr{S}}.$$

The two local operators, $\langle\rangle$ and $[]$, are translated into graph-theoretic operations:

$$\begin{aligned} ||\langle\rangle\varphi||^\alpha_{\mathscr{S}} &= \{s \in S \colon \mathrm{sucs}^{\mathscr{S}}(s) \cap ||\varphi||^\alpha_{\mathscr{S}} \neq \varnothing\}, \\ ||[]\varphi||^\alpha_{\mathscr{S}} &= \{s \in S \colon \mathrm{sucs}^{\mathscr{S}}(s) \subseteq ||\varphi||^\alpha_{\mathscr{S}}\}. \end{aligned}$$

The semantics of the fixed-point operators is based on the observation that for every expression φ, the function $S' \mapsto ||\varphi||^{\alpha[X\mapsto S']}_{\mathscr{S}}$ is a monotone function on 2^S with set inclusion as ordering, where $\alpha[X \mapsto S']$ denotes the variable assignment which coincides with α, except for the value of the variable X, which is S'. The Knaster–Tarski Theorem then guarantees that this function has a least and a greatest fixed point:

$$\begin{aligned} ||\mu X\varphi||^\alpha_{\mathscr{S}} &= \bigcap \left\{ S' \subseteq S \colon ||\varphi||^{\alpha[X\mapsto S']}_{\mathscr{S}} = S' \right\}, \\ ||\nu X\varphi||^\alpha_{\mathscr{S}} &= \bigcup \left\{ S' \subseteq S \colon ||\varphi||^{\alpha[X\mapsto S']}_{\mathscr{S}} = S' \right\}. \end{aligned}$$

In the first equation the last equality sign can be replaced by $\subseteq$, while in the second equation it can be replaced by $\supseteq$. The above equations are—contrary to what was said at the beginning of this section—declarative rather than operational, but this can easily be changed because of the Knaster–Tarski Theorem. For a given system $\mathscr{S}$, a variable assignment α, an MC expression φ, and a fixed-point variable X, consider the ordinal sequence $(S_\lambda)_\lambda$, called approximation sequence for $||\mu X\varphi||^\alpha_{\mathscr{S}}$, defined by

$$S_0 = \varnothing, \qquad S_{\lambda+1} = ||\varphi||^{\alpha[X\mapsto S_\lambda]}_{\mathscr{S}}, \qquad S_{\lambda'} = \bigcup_{\lambda<\lambda'} S_\lambda,$$

where λ' stands for a limit ordinal. Because of monotonicity, we have $S_0 \subseteq S_1 \subseteq \dots$. The definition of the sequence implies that if $S_\lambda = S_{\lambda+1}$ for any λ, then $S_{\lambda'} = S_\lambda = ||\mu X \varphi||^\alpha_{\mathscr{S}}$ for all $\lambda' \geq \lambda$. Clearly, we have $\lambda \leq \mathrm{card}(S)$ for the smallest such λ, which, for finite transition systems, means there is a simple (recursive) way to evaluate $\mu X \varphi$. The same holds true for $\nu X \varphi$, where the approximation is from above, that is, $S_0 = S$ and the inclusion order is reversed.

For notational convenience, we also use $\mathscr{S}, \alpha, s \models \varphi$ to denote $s \in ||\varphi||^\alpha_{\mathscr{S}}$ for any state $s \in S$. When φ is a closed MC expression, then the variable assignment α is irrelevant for its interpretation, so we omit it and simply write $||\varphi||_{\mathscr{S}}$ or $\mathscr{S}, s \models \varphi$.

For examples of useful expressions, recall the CTL formula (1.3) from Section 5.1. We can express its subformula $\mathsf{EF}p_d$ by

$$\varphi_{\text{inner}} = \mu X(p_d \vee \langle\,\rangle X),$$

so that the full formula can be written as

$$\nu Y(\varphi_{\text{inner}} \wedge [\,]Y).$$

In a similar fashion, (1.4) can be expressed:

$$\nu Y((\neg p_r \vee \mu X(p_a \vee [\,]X)) \wedge [\,]Y).$$

It is more complicated to express the LTL formula (1.5); it needs a nested fixed-point expression with mutually dependent fixed-point variables. We first build an expression which denotes all states from which on all paths a state is reachable where p is true and which belongs to a set Y:

$$\varphi'_{\text{inner}} = \mu X((p \wedge Y) \vee [\,]X).$$

Observe that Y occurs free in φ'_{inner}. The desired expression can then be phrased as a greatest fixed point:

$$\nu Y \varphi'_{\text{inner}}.$$

It is no coincidence that we are able to express the two CTL formulas in MC:

Proposition 6.1. For every CTL formula φ there exists a closed MC expression $\tilde{\varphi}$ such that for every transition system $\mathscr{S}$ and $s \in S$,

$$\mathscr{S}, s \models \varphi \qquad \text{iff} \qquad \mathscr{S}, s \models \tilde{\varphi}.$$

Proof. The proof is a straightforward induction. We describe one case of the inductive step. Assume ψ and χ are CTL formulas and $\tilde{\psi}$ and $\tilde{\chi}$ are MC expressions such that the claim holds. We consider $\varphi = \mathsf{E}(\psi \,\mathsf{XU}\, \chi)$ and want to construct $\tilde{\varphi}$ as desired. We simply express the semantics of φ by a fixed-point computation:

$$\tilde{\varphi} = \langle\,\rangle \mu X(\tilde{\chi} \vee (\tilde{\psi} \wedge \langle\,\rangle \tilde{X})).$$

The other cases can be dealt with in the same fashion. Q.E.D.

The next observation is that as far as satisfiability is concerned, we can restrict our considerations to trees, just as with CTL (recall Lemma 5.1).

Lemma 6.2. For every MC expression φ, transition system $\mathscr{S}$, variable assignment α, and state $s \in S$,

$$\mathscr{S}, \alpha, s \models \varphi \qquad \text{iff} \qquad \mathscr{T}_s(\mathscr{S}), \alpha \models \varphi.$$

(Recall that when we view a tree as a transition system, then we interpret formulas in the root of the tree unless stated otherwise.)

Proof. This can be proved by a straightforward induction on the structure of φ, using the following inductive claim:

$$\{v \in V^{\mathscr{T}_s(\mathscr{S})} : \mathscr{S}, \alpha, v(*) \models \varphi\} = ||\varphi||^{\alpha}_{\mathscr{T}_s(\mathscr{S})}.$$

This simply says that with regard to MC, there is no difference between a state s' in a given transition system $\mathscr{S}$ and every vertex v with $v(*) = s'$ in the unraveling of $\mathscr{S}$. Q.E.D.

Just as with CTL, the lemma allows us to work henceforth in the tree framework. For a closed MC expression φ with propositional variables from a set $P = \{p_0, \dots, p_{n-1}\}$, the tree language defined by φ, denoted $\mathscr{L}(\varphi)$, is the set of all trees $\mathscr{T}$ over 2^P such that $\mathscr{T} \models \varphi$.

The next observation is that every MC expression can be translated into a monadic second-order formula, similar to Proposition 5.2. Before we can state the result, we define an appropriate equivalence relation between SUS formulas and MC expressions. Recall that an SUS formula is true or not for a given tree, while an MC expression evaluates to a set of vertices.

Let $P = \{p_0, \dots, p_{n-1}\}$ be a set of propositional variables and φ an MC expression over P with free fixed-point variables among $X_0, \dots, X_{m-1}$. We view the variables $X_0, \dots, X_{m-1}$ as further propositional variables and identify each X_i with a set variable V_i and each p_j with a set variable V_{m+j}. So we can interpret φ and every SUS formula $\psi = \psi(V_0, \dots, V_{m+n-1})$ in trees over $[2]_{m+n}$. We say φ is equivalent to such a formula ψ if $\mathscr{L}(\varphi) = \mathscr{L}(\psi)$.

Proposition 6.3. For every MC expression φ, an equivalent SUS formula $\tilde{\varphi}$ can be constructed.

Proof. This can be proved by induction on the structure of φ, using a more general claim. For every MC expression φ as above, we construct an SUS formula $\hat{\varphi} = \hat{\varphi}(V_0, \dots, V_{m+n-1}, z)$ such that for every tree $\mathscr{T}$ over $[2]_{m+n}$ and $v \in V^{\mathscr{T}}$, we have:

$$\mathscr{T}{\downarrow}v \models \varphi \qquad \text{iff} \qquad \mathscr{T}, v \models \hat{\varphi}(V_0, \dots, V_{m+n-1}, z),$$

where $\mathscr{T}, v \models \hat{\varphi}(V_0, \dots, V_{m+n-1}, z)$ is defined in the obvious way, see Section 4.1 for a similar definition in the context of LTL. We can then set $\tilde{\varphi} = \exists x(\forall y(\neg \mathrm{suc}(y, x) \wedge \hat{\varphi}(V_0, \dots, V_{m+n-1}, x)))$.

The interesting cases in the inductive step are the fixed-point operators. So let $\varphi = \mu X_i \psi$ and assume $\hat{\psi}$ is already given. The formula $\hat{\varphi}$ simply says that z belongs to a fixed point and that every other fixed point is a superset of it:

$$\begin{aligned}\hat{\varphi} = \exists Z(z \in Z \wedge \forall z'(\hat{\psi}(\dots, V_{i-1}, Z, V_{i+1}, \dots, z') \leftrightarrow z' \in Z) \wedge \\ \forall Z'(\forall z'(\psi(\dots, V_{i-1}, Z', V_{i+1}, \dots, z') \leftrightarrow z' \in Z') \rightarrow Z \subseteq Z')).\end{aligned}$$

For the greatest fixed-point operator, the construction is analogous. Q.E.D.

As a consequence, we can state:

Corollary 6.4 (Kozen-Parikh, [67]). MC satisfiability is decidable.

But, just as with LTL and CTL, by a translation into monadic second-order logic we get only a nonelementary upper bound for the complexity.

6.2 From MC to alternating tree automata

Our overall objective is to derive a good upper bound for the complexity of MC satisfiability. The key is a translation of MC expressions into nondeterministic tree automata via alternating parity tree automata. We start with the translation of MC expressions into alternating parity tree automata.

Alternating parity tree automata are defined exactly as nondeterministic Büchi tree automata are defined in Section 5.3 except that the Büchi acceptance condition is replaced by a parity condition π.

Just as with LTL and CTL, the translation into alternating automata reflects the semantics of the expressions in a direct fashion. The fixed-point operators lead to loops, which means that the resulting tree automata will no longer be very weak (not even weak). For least fixed points these loops may not be traversed infinitely often, while this is necessary for greatest fixed points. To control this, priorities are used: Even priorities are used for greatest fixed-points, odd priorities for least fixed points. Different priorities are used to take into account the nesting of fixed points, the general rule being that outer fixed points have smaller priorities, because they are more important.

For model checking, it will be important to make sure as few different priorities as possible are used. That is why a careful definition of alternation depth is needed. In the approach by Emerson and Lei [41], one counts the number of alternations of least and greatest fixed points on the paths of the parse tree of a given expression. Niwiński's approach [92] yields a coarser hierarchy, which gives better upper bounds for model checking. It requires that relevant nested subexpressions are "mutually recursive".

Let $\leq$ denote the relation "is subexpression of", that is, $\psi \leq \varphi$ if $\psi \in \mathrm{sub}(\varphi)$. Let φ be an MC expression. An *alternating μ-chain* in φ of length l is a sequence

$$\varphi \geq \mu X_0 \psi_0 > \nu X_1 \psi_1 > \mu X_2 \psi_2 > \cdots > \mu/\nu X_{l-1} \psi_{l-1} \tag{1.6}$$

where, for every $i < l-1$, the variable X_i occurs free in every formula ψ with $\psi_i \geq \psi \geq \psi_{i+1}$. The maximum length of an alternating μ-chain in φ is denoted by $m^\mu(\varphi)$. Symmetrically, ν-chains and $m^\nu(\varphi)$ are defined. The *alternation depth* of a μ-calculus expression φ is the maximum of $m^\mu(\varphi)$ and $m^\nu(\varphi)$ and is denoted by $\mathrm{d}(\varphi)$.

We say an MC expression is in normal form if for every fixed-point variable X occurring the following holds:

- every occurrence of X in φ is free or
- all occurrences of X in φ are bound in the same subexpression $\mu X \psi$ or $\nu X \psi$, which is then denoted by φ_X.

Clearly, every MC expression is equivalent to an MC expression in normal form.

The full translation from MC into alternating parity tree automata can be found in Figure 27, where the following notation is used. When φ is an MC expression and $\mu X \psi \in \mathrm{sub}(\varphi)$, then

$$\mathrm{d}_\varphi(\mu X \psi) = \begin{cases} \mathrm{d}(\varphi) + 1 - 2\lceil \mathrm{d}(\mu X \psi)/2 \rceil, & \text{if } \mathrm{d}(\varphi) \bmod 2 = 0, \\ \mathrm{d}(\varphi) - 2\lfloor \mathrm{d}(\mu X \psi)/2 \rfloor, & \text{otherwise.} \end{cases}$$

Similarly, when $\nu X \psi \in \mathrm{sub}(\varphi)$, then

$$\mathrm{d}_\varphi(\nu X \psi) = \begin{cases} \mathrm{d}(\varphi) - 2\lfloor \mathrm{d}(\nu X \psi)/2 \rfloor, & \text{if } \mathrm{d}(\varphi) \bmod 2 = 0, \\ \mathrm{d}(\varphi) + 1 - 2\lceil \mathrm{d}(\nu X \psi)/2 \rceil, & \text{otherwise.} \end{cases}$$

This definition reverses alternation depth so it can be used for defining the priorities in the alternating parity automaton for an MC expression. Recall that we want to assign priorities such that the higher the alternation depth the lower the priority and, at the same time, even priorities go to ν-formulas

Let φ be a closed MC expression in normal form and Q a set which contains for every $\psi \in \text{sub}(\varphi)$ a state $[\psi]$. The alternating parity tree automaton for φ, denoted $\mathscr{A}[\varphi]$, is defined by

$$\mathscr{A}[\varphi] = (P, Q, \varphi, \delta, \pi)$$

where the transition function is given by

$$\begin{aligned}
\delta([p]) &= p, & \delta([\neg p]) &= \neg p,\\
\delta([\psi \vee \chi]) &= [\psi] \vee [\chi], & \delta([\psi \wedge \chi]) &= [\psi] \wedge [\chi],\\
\delta([\langle\,\rangle\psi]) &= \Diamond[\psi], & \delta([[\,]\psi]) &= \Box[\psi],\\
\delta([\mu X \psi]) &= [\psi], & \delta([\nu X \psi]) &= [\psi],\\
\delta([X]) &= [\varphi_X],
\end{aligned}$$

and where

$$\pi([\psi]) = \mathrm{d}_\varphi(\psi)$$

for every fixed-point expression $\psi \in \text{sub}(\varphi)$.

FIGURE 27. From μ-calculus to alternating tree automata

and odd priorities to μ-formulas. This is exactly what the above definition achieves.

It is obvious that $\mathscr{A}[\varphi]$ will have $\mathrm{d}(\varphi) + 1$ different priorities in general, but from a complexity point of view, these cases are not harmful. To explain this, we introduce the notion of index of an alternating tree automaton. The transition graph of an alternating tree automaton $\mathscr{A}$ is the graph with vertex set Q and where (q, q') is an edge if q' occurs in $\delta(q)$. The index of $\mathscr{A}$ is the maximum number of different priorities in the strongly connected components of the transition graph of $\mathscr{A}$. Clearly, $\mathscr{A}[\varphi]$ has index $\mathrm{d}(\varphi)$.

Theorem 6.5 (Emerson-Jutla, [40]). Let φ be an MC expression in normal form with n subformulas. Then $\mathscr{A}[\varphi]$ is an alternating parity tree automaton with n states and index $\mathrm{d}(\varphi)$ such that $\mathscr{L}(\mathscr{A}[\varphi]) = \mathscr{L}(\varphi)$.

To be more precise, $\mathscr{A}[\varphi]$ may have $\mathrm{d}(\varphi) + 1$ different priorities, but in every strongly connected component of the transition graph of $\mathscr{A}[\varphi]$ there are at most $\mathrm{d}(\varphi)$ different priorities, see also Theorem 2.21.

Proof. The claims about the number of states and the index are obviously true. The proof of correctness is more involved than the corresponding proofs for LTL and CTL, because the automata which result from the translation are, in general, not weak.

The proof of the claim is by induction on the structure of φ. The base case is trivial and so are the cases in the inductive step except for the cases where fixed-point operators are involved. We consider the case where $\varphi = \mu X \psi$.

So assume $\varphi = \mu X\psi$ and $\mathscr{T} \models \varphi$. Let $f \colon 2^V \to 2^V$ be defined by $f(V') = ||\psi||_{\mathscr{T}}^{X \mapsto V'}$. Let $(V_\lambda)_\lambda$ be the sequence defined by $V_0 = \varnothing$, $V_{\lambda+1} = f(V_\lambda)$, and $V_{\lambda'} = \bigcup_{\lambda<\lambda'} V_\lambda$ for limit ordinals λ'. We know that f has a least fixed point, which is the value of φ in $\mathscr{T}$, and that there exists κ such that V_κ is the least fixed-point of f. We show by induction on λ that there exists an accepting run of $\mathscr{A}[\varphi]$ on $\mathscr{T}{\downarrow}v$ for every $v \in V_\lambda$. This is trivial when $\lambda = 0$ or when λ is a limit ordinal. When λ is a successor ordinal, say $\lambda = \lambda_0 + 1$, then $V_\lambda = f(V_{\lambda_0})$. Consider the automaton $\mathscr{A}[\psi]$ where X is viewed as a propositional variable. By the outer induction hypothesis, there exists an accepting run $\mathscr{R}$ of $\mathscr{A}[\psi]$ on $\mathscr{T}[X \mapsto V_{\lambda_0}]{\downarrow}v$, where $\mathscr{T}[X \mapsto V_{\lambda_0}]$ is the obvious tree over $2^{P\cup\{X\}}$. We can turn $\mathscr{R}$ into a prefix $\mathscr{R}'$ of a run of $\mathscr{A}[\varphi]$ on $\mathscr{T}{\downarrow}v$ by adding a new root labeled $([\varphi], v)$ to it. Observe that some of the leaves w of $\mathscr{R}'$ may be labeled (X, v') with $v' \in V_{\lambda_0}$. For each such v' there exists, by the inner induction hypothesis, an accepting run $\mathscr{R}_{v'}$ of $\mathscr{A}[\varphi]$ on $\mathscr{T}{\downarrow}v$. Replacing w by $\mathscr{R}_{v'}$ for every such leaf w yields a run $\hat{\mathscr{R}}$ of $\mathscr{A}[\varphi]$ on $\mathscr{T}{\downarrow}v$. We claim this run is accepting. To see this, observe that each infinite branch of $\hat{\mathscr{R}}$ is an infinite branch of $\mathscr{R}'$ or has an infinite path of $\mathscr{R}_{v'}$ for some v' as a suffix. In the latter case, the branch is accepting for a trivial reason, in the former case, the branch is accepting because the priorities in $\mathscr{A}[\psi]$ differ from the priorities in $\mathscr{A}[\varphi]$ by a fixed even number. This completes the inductive proof. Since, by assumption, the root of $\mathscr{T}$ belongs to V_κ, we obtain the desired result.

For the other direction, assume $\mathscr{T}$ is accepted by $\mathscr{A}[\varphi]$, say by a run $\mathscr{R}$. Let W be the set of all $w \in V^{\mathscr{R}}$ such that φ is the first component of $l^{\mathscr{R}}(w)$. Observe that because of the definition of the priority function π there can only be a finite number of elements from W on each branch of $\mathscr{R}$. This is because the priority function π is defined in a way such that if $\psi \in \mathrm{sub}(\varphi)$ is a fixed-point formula with $[\psi]$ in the strongly connected component of $[\varphi]$ in the transition graph of $\mathscr{A}[\varphi]$, then $\pi([\varphi]) \leq \pi([\psi])$.

Consider the sequence $(V_\lambda)_\lambda$ of subsets of $V^{\mathscr{R}}$ defined as follows:

- $V_0 = \varnothing$,
- $w \in V_{\lambda+1}$ if all proper descendants of w in $\mathscr{R}$ belong to $V_\lambda \cup V^{\mathscr{R}} \setminus W$, and

- $V_{\lambda'} = \bigcup_{\lambda<\lambda'} V_\lambda$ for every limit ordinal λ'.

Using the induction hypothesis, one can prove by induction on λ that for every $w \in V_\lambda$ the second component of its label belongs to $||\varphi||_{\mathscr{T}}$.

Since there are only a finite number of elements from W on each branch of V_λ, one can also show that $\text{root}(\mathscr{R}) \in W$, which proves the claim. Q.E.D.

Before we turn to the conversion of alternating into nondeterministic parity tree automata, we discuss model checking MC expressions briefly. Model checking an MC expression, that is, evaluating it in a finite transition system is "trivial" in the sense that one can simply evaluate the expression according to its semantics, using approximation for evaluating fixed-point operators as explained in Section 6.1. Using the fact that fixed-points of the same type can be evaluated in parallel one arrives at an algorithm which is linear in the product of the size of the expression and the size of the system, but exponential in the depth of the alternation between least and greatest fixed points.

An alternative approach to model checking MC expressions is to proceed as with CTL. Given a finite transition system $\mathscr{S}$, an initial state $s_I \in S$, and an expression φ, one first constructs the alternating automaton $\mathscr{A}[\varphi]$, then the product game $\mathscr{A}[\varphi] \times_{s_I} \mathscr{S}$ (with a parity condition rather than a Büchi condition), and finally solves this game. (Of course, on can also directly construct the game.) As a consequence of the previous theorem and Theorem 2.21, one obtains:

Theorem 6.6 (Seidl-Jurdziński, [107, 62])**.** An MC expression of size l and alternation depth d can be evaluated in a finite transition system with m states and n transitions in time $O((lm + ln(lm)^{\lfloor d/2 \rfloor}))$. Q.E.D.

In fact, there is a close connection between MC model checking and solving parity games: The two problems are interreducible, which means all the remarks on the complexity of solving parity games at the end of Section 2.5 are equally valid for MC model checking.

The above theorem tells us something about AMC, the set of all MC expressions with alternation depth ≤ 1. These expressions can be evaluated in time linear in the product of the size of the transition system and the length of the formula, which was first proved by Cleaveland, Klein, and Steffen [29] in general and by Kupferman, Vardi, and Wolper using automata-theoretic techniques [71]. This yields a different proof of Theorem 5.10: The straightforward translation from CTL into the μ-calculus, see Proposition 6.1, yields alternation-free expressions of linear size. From a practical point, it is interesting to note that model checking tools indeed use the translation of CTL into AMC, see [84].

6.3 From alternating to nondeterministic tree automata

In view of Theorem 6.5, what we need to solve MC satisfiability is a translation of alternating tree automata into nondeterministic tree automata, because we already know how to decide emptiness for these automata. To be precise, we proved this only for Büchi acceptance conditions, see Figure 24, but this extends to parity tree automata in a straightforward manner.

One way of achieving a translation from alternating into nondeterministic automata is to proceed in two steps, where the intermediate result is an alternating automaton with very restrictive transition conditions. We say a transition condition is in normal form if it is a disjunction of transition conditions of the form

$$\bigwedge_{q \in Q^{\mathsf{A}}} \Box q \wedge \bigwedge_{q \in Q^{\mathsf{E}}} \Diamond q.$$

The conversion of an ordinary alternating tree automaton into an alternating tree automaton with transition conditions in normal form is similar to removing ε-transitions. We describe it here for the case where the transition conditions are simpler as in the general case, namely where each subformula $\Box\gamma$ or $\Diamond\gamma$ is such that γ is a state. Observe that all the transition conditions in the construction described in Figure 27 are of this form. At the same time, we change the format of the transition function slightly. We say an alternating automaton is in normal form if its transition function δ is of the form $\delta\colon Q \times 2^P \to \mathrm{TC}(P, Q)$ where $\delta(q, a)$ is a transition condition in normal form for $q \in Q$ and $a \in 2^P$. The notion of a run of an alternating automaton is adapted appropriately.

To convert alternating automata into normal form, we start with a crucial definition. Let $\mathscr{A}$ be an alternating parity tree automaton, $a \in 2^P$, and $q \in Q$. We say a tree $\mathscr{R}$ labeled with transition conditions is a transition tree for q and a if its root is labeled q and every vertex w with label γ satisfies the following conditions:

- if $\gamma = p$, then $p \in a$, and if $\gamma = \neg p$, then $p \notin a$,
- if $\gamma = q'$, then there exists $w' \in \mathrm{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = \delta(q')$,
- if $\gamma = \Diamond q'$ or $\gamma = \Box q'$, then w has no successor,
- if $\gamma = \gamma_0 \vee \gamma_1$, then there exists $i < 2$ and $w' \in \mathrm{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = \gamma_i$,
- if $\gamma = \gamma_0 \wedge \gamma_1$, then for every $i < 2$ there exists $w' \in \mathrm{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = \gamma_i$.

Let $\mathscr{A}$ be an alternating parity tree automaton. The normalization of $\mathscr{A}$ is the alternating parity tree automaton $\mathscr{A}^{\text{norm}}$ defined by

$$\mathscr{A}^{\text{norm}} = (P, Q \times \pi(Q), (q_I, j), \delta', \pi')$$

where

- j is any element of $\pi(Q)$,
- $\pi'((q,i)) = i$ for all $q \in Q$ and $i \in \pi(Q)$, and
- $\delta'((q,i),a) = \bigvee \gamma_{\mathscr{R}}$ for $q \in Q$, $i \in \pi(Q)$, and $a \in 2^P$, with $\mathscr{R}$ ranging over all transition trees for q and a.

FIGURE 28. Normalizing transition conditions of alternating tree automata

Further, every infinite branch of $\mathscr{R}$ is accepting with respect to π.

A transition tree as above can easily be turned into a transition condition in normal form over an extended set of states, namely $\bar{Q} = Q \times \pi(Q)$. The second component is used to remember the minimum priority seen on a path of a transition tree, as explained below. Let Q^{A} be the set of pairs (q', i) such that $\Box q'$ is a label of a leaf of $\mathscr{R}$, say w, and i is the minimum priority on the path from the root of $\mathscr{R}$ to w. Similarly, let Q^{E} be the set of pairs (q', i) such that $\Diamond q'$ is a label of a leaf of $\mathscr{R}$, say w, and i is the minimum priority on the path from the root of $\mathscr{R}$ to w. The transition condition for the transition tree $\mathscr{R}$, denoted $\gamma_{\mathscr{R}}$, is defined by

$$\gamma_{\mathscr{R}} = \bigwedge_{(q',i) \in Q^{\mathsf{A}}} \Box(q', i) \wedge \bigwedge_{(q',i) \in Q^{\mathsf{E}}} \Diamond(q', i).$$

The entire normalization construction is depicted in Figure 28.

Lemma 6.7. Let $\mathscr{A}$ be an alternating parity tree automaton with n states and k different priorities. Then $\mathscr{A}^{\text{norm}}$ is an alternating parity tree automaton in normal form with kn states and k different priorities. Q.E.D.

The second step in our construction is a conversion of an alternating automaton in normal form into a nondeterministic tree automaton, similar to the conversion of universal parity tree automata into nondeterministic tree automata explained in Section 3.3. Again, we heavily draw on the generic automaton introduced in that section. Recall that given a finite state set Q and a priority function π, the generic automaton is a deterministic automaton over $\mathscr{Q}$, the alphabet consisting of all binary relations over Q, which accepts a word $u \in \mathscr{Q}^\omega$ if all $v \in \langle u \rangle$ satisfy the parity condition π.

Given an alternating automaton in normal form, a set $Q' \subseteq Q$, and a letter $a \in 2^P$, a pair $(\mathscr{Q}', R)$ with $\mathscr{Q}' \subseteq \mathscr{Q}$ and $R \in \mathscr{Q}$ is a choice for Q' and a if for every $q \in Q'$ there exists a disjunct in $\delta(q)$ of the form

$$\bigwedge_{q' \in Q_q^{\mathsf{A}}} \Box q' \wedge \bigwedge_{q' \in Q_q^{\mathsf{E}}} \Diamond q'$$

such that the following conditions are satisfied:

(i) $R = \{(q, q') \colon q \in Q' \wedge q' \in Q_q^{\mathsf{A}}\}$,

(ii) $R \subseteq R'$ for every $R' \in \mathscr{Q}'$,

(iii) for every $q \in Q'$ and every $q' \in Q_q^{\mathsf{E}}$ there exists $R' \in \mathscr{Q}'$ such that $(q, q') \in R'$, and

(iv) $|\mathscr{Q}'| \leq |Q| \times |Q| + 1$.

For a set $Q' \subseteq Q$ and a relation $R \subseteq Q \times Q$, we write $Q'R$ for the set $\{q' \in Q \colon \exists q (q \in Q' \wedge (q, q') \in R\}$.

The details of the conversion from alternating parity tree automata in normal form into nondeterministic tree automata can be found in Figure 29. It is analogous to the construction depicted in Figure 16, which describes how a universal parity tree automaton over binary trees can be turned into a nondeterministic parity tree automaton. The situation for alternating automata is different in the sense that the transition conditions of the form $\Diamond q'$ have to be taken care of, too, but this is captured by (iii) in the above definition.

Lemma 6.8. Let $\mathscr{A}$ be an alternating parity automaton in normal form with n states and k different priorities. Then A^{nd} is an equivalent nondeterministic automaton with a number of states exponential in n and a number of priorities polynomial in n.

Proof. The claims about the number of states and number of priorities are obvious. The correctness proof can be carried out almost in the same fashion as the proof of Lemma 3.10, except for one issue. In order to see that it is admissible to merge all branches of a run on a certain branch of a given tree into one element of $\mathscr{Q}^\omega$, one has to use Theorem 2.20, the memoryless determinacy of parity games. Q.E.D.

Let $\mathscr{A}$ be an alternating parity tree automaton in normal form and $\mathscr{B} = \mathscr{A}[Q^{\mathscr{A}}, \pi^{\mathscr{A}}]$ the generic automaton for $Q^{\mathscr{A}}$ and $\pi^{\mathscr{A}}$.

The nondeterministic automaton $\mathscr{A}^{\mathrm{nd}}$ is defined by

$$\mathscr{A}^{\mathrm{nd}} = (2^P, 2^{Q^{\mathscr{A}}} \times Q^{\mathscr{B}}, (\{q_I^{\mathscr{A}}\}, q_I^{\mathscr{B}}), \Delta, \pi)$$

where $\pi((Q', q)) = \pi^{\mathscr{B}}(q)$ and $((Q', q), q, \bar{\mathscr{Q}}, \{(Q'R', \delta^{\mathscr{B}}(q, R'))\}) \in \Delta$ if there exists a choice $(\mathscr{Q}', R)$ for Q' and a such that

$$\bar{\mathscr{Q}} = \{(Q'R, \delta^{\mathscr{B}}(q, R)) \colon R \in \mathscr{Q}'\}.$$

FIGURE 29. From alternating to nondeterministic tree automata

As a consequence of Theorem 6.5 and Lemmas 6.7 and 6.8, we obtain:

Corollary 6.9. (Emerson-Streett-Jutla, [40]) Every MC expression can be translated into an equivalent nondeterministic parity tree automaton with an exponential number of states and a polynomial number of different priorities.

In view of Lemma 5.6 and Theorem 2.21, we can also conclude:

Corollary 6.10 (Emerson-Jutla, [39])**.** MC satisfiability is complete for exponential time.

For the lower bound, we refer to [39]. We finally note that a converse of Corollary 6.9 also holds:

Theorem 6.11 (Niwiński-Emerson-Jutla-Janin-Walukiewicz, [93, 40, 59])**.** Let P be a finite set of propositional variables. For every alternating parity tree automaton and every nondeterministic tree automaton over 2^P, there exists an equivalent closed MC expression.

6.4 Notes

Satisfiability for MC is not only complexity-wise simpler than satisfiability for S2S. The proofs for showing decidability of satisfiability for S2S all make use of a determinization construction for automata on infinite words. The "safraless decision procedures" advocated by Kupferman and Vardi

[75] avoid this, but they still use the fact that equivalent deterministic word automata of a bounded size exist.

The nondeterministic tree automaton models for SUS and MC are not only similar on the surface: A fundamental result by Janin and Walukiewicz [60] states that the bisimulation-invariant tree languages definable in SUS are exactly the tree languages definable in MC, where the notion of bisimulation exactly captures the phenomenon that MC expressions (just as CTL formulas) are resistant against duplicating subtrees.

MC has been extended in various ways with many different objectives. With regard to adding to its expressive power while retaining decidability, one of the most interesting results is by Grädel and Walukiewicz [53], which says that satisfiability is decidable for guarded fixed-point logic. This logic can be seen as an extension of the modal μ-calculus insofar as guarded logic is considered a natural extension of modal logic, and guarded fixed-point logic is an extension of guarded logic just as modal μ-calculus is an extension of model logic by fixed-point operators. For further extensions, see [78, 127] and [68]. Other important work with regard to algorithmic handling of MC was carried out by Walukiewicz in [130], where he studies the evaluation of MC expressions on pushdown graphs.

References

[1] C. S. Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of büchi automata. *Theor. Comput. Sci.*, 363(2):224–233, 2006.

[2] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In J.-P. Katoen and P. Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*, volume 2280 of *Lecture Notes in Computer Science*, pages 296–211. Springer, 2002.

[3] R. Armoni, D. Korchemny, A. Tiemeyer, and M. Y. V. Y. Zbar. Deterministic dynamic monitors for linear-time assertions. In *Proc. Workshop on Formal Approaches to Testing and Runtime Verification*, volume 4262 of *Lecture Notes in Computer Science*. Springer, 2006.

[4] A. Arnold. Rational omega-languages are non-ambiguous. *Theor. Comput. Sci.*, 26:221–223, 1983.

[5] A. Arnold, J. Duparc, D. Niwiński, and F. Murlak. On the topological complexity of tree languages. This volume.

[6] A. Arnold and D. Niwiński. *Rudiments of μ-Calculus.* Elsevier, Amsterdam, The Netherlands, 2001.

[7] M. Ben-Ari, Z. Manna, and A. Pnueli. The logic of nexttime. In *Proc. 8th ACM Symp. on Principles of Programming Languages (POPL)*, pages 164–176, 1981.

[8] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. Dag-width and parity games. In B. Durand and W. Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23–25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 524–536, 2006.

[9] A. Blumensath, T. Colcombet, and C. Löding. Logical theories and compatible operations. This volume.

[10] M. Bojanczyk. The common fragment of ctl and ltl needs existential modalities. Available as `http://www.mimuw.edu.pl/~bojan/papers/paradox.pdf`, 2007.

[11] J. C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theor. Comput. Sci.*, 195(2):133–153, 1998.

[12] J. R. Büchi. Using determinancy of games to eliminate quantifiers. In *FCT*, pages 367–378, 1977.

[13] R. M. Burstall. Program proving as hand simulation with a little induction. In *Information Processing 74*, pages 308–312, Stockholm, Sweden, Aug. 1974. International Federation for Information Processing, North-Holland Pub. Co.

[14] D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M. Y. Vardi. Regular vacuity. In *Proc. 13th Conf. on Correct Hardware Design and Verification Methods*, volume 3725 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 2005.

[15] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problems.* Springer, 1997.

[16] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[17] J. R. Büchi. On a decision method in restricted second-order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Logic, Methodology, and Philosophy of Science: Proc. of the 1960 International Congress*, pages 1–11, Stanford, Calif., 1962. Stanford University Press.

[18] J. R. Büchi. Decision methods in the theory of ordinals. *Bull. Am. Math. Soc*, 71:767–770, 1965.

[19] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[20] A. Carayol and C. Löding. MSO on the infinite binary tree: Choice and order. In *CSL'07*, volume 4646 of *LNCS*, pages 161–176. Springer, 2007.

[21] O. Carton, D. Perrin, and J.-É. Pin. Automata and semigroups recognizing infinite words. This volume.

[22] D. Caucal. Deterministic graph grammars. This volume.

[23] A. Church. Application of recursive arithmetics to the problem of circuit analysis. In I. f. D. A. Communications Research Division, editor, *Summaries of Talks Presented at the Summer Institute for Symbolic Logic, Ithaca, Cornell University, July 1957*, pages 3–50, 1960.

[24] A. Church. Logic, arithmetics, and automata. In I. Mittag-Leffler, editor, *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35, 1963.

[25] A. Cimatti, M. Roveri, S. Semprini, and S. Tonetta. From psl to nba: A modular symbolic encoding. In *Proc. 6th Int'l Conf. on Formal Methods in Computer-Aided design*, 2006.

[26] E. M. Clarke and I. A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30–June 3, 1988, Proceedings*, volume 354 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 1988.

[27] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications: a practical approach. In ACM, editor, *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages*, pages 117–126, Austin, Texas, 1983. ACM Press.

[28] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, Apr. 1986.

[29] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal μ-calculus. In G. von Bochmann and D. K. Probst, editors, *Computer Aided Verification, Fourth International Workshop, CAV '92, Montreal, Canada, June 29 – July 1, 1992, Proceedings*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422. Springer, 1992.

[30] V. Diekert and P. Gastin. First-order definable languages. This volume.

[31] J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, Oct. 1970.

[32] M. Droste and P. Gastin. Weighted automata and weighted logics. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 2005.

[33] W. Ebinger and A. Muscholl. Logical definability on infinite traces. In A. Lingas, R. Karlsson, and S. Carlsson, editors, *Automata, Languages and Programming: 20th International Colloquium*, volume 700 of *Lecture Notes in Computer Science*, pages 335–346, Lund, Sweden, 1993. EATCS, Springer.

[34] C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Springer, 2006.

[35] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, Jan. 1961.

[36] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.

[37] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. System Sci.*, 30:1–24, 1985.

[38] E. A. Emerson and J. Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *J. Assoc. Comput. Mach.*, 33(1):151–178, 1986.

[39] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science*, pages 328–337, San Juan, Puerto Rico, 1991. IEEE.

[40] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, October 1991. IEEE.

[41] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *1st IEEE Symposium on Symposium on Logic in Computer Science*, pages 267–278, Cambridge, Massachusetts, 16–18 June 1986. IEEE Computer Society.

[42] E. A. Emerson and A. P. Sistla. Deciding branching time logic. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, 1984, Washington, D.C., USA*, pages 14–24. ACM, 1984.

[43] E. A. Emerson and A. P. Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, 1984.

[44] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown automata. In F. Moller, editor, *Proc. 2nd Int. Workshop on Verification of Infinite States Systems*, 1997.

[45] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *J. Comput. System Sci.*, 18:194–211, 1979.

[46] E. Friedgut, O. Kupferman, and M. Y. Vardi. Büchi complementation made tighter. *Int. J. Found. Comput. Sci.*, 17(4):851–868, 2006.

[47] D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Conference Record of the 12th ACM Symposium on Principles of Programming Languages*, pages 163–173, Las Vegas, Nev., 1980.

[48] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

[49] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In P. Dembiski and M. Sredniawa, editors, *Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, 1995.

[50] D. Giammarresi and A. Restivo. Matrix based complexity functions and recognizable picture languages. This volume.

[51] D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Information and Computation*, 125(1):32–45, Feb. 1996.

[52] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[53] E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *LICS*, pages 45–54, 1999.

[54] Y. Gurevich and L. Harrington. Trees, automata, and games. In *14th ACM Symposium on the Theory of Computing*, pages 60–65, San Francisco, 1982. ACM Press.

[55] S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Vardi. On complementing nondeterministic Büchi automata. In *Proc. 12th Conf. on Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer, 2003.

[56] J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS: Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19–20, 1995, Proceedings*, volume 1019 of *Lecture Notes in Computer Science*, pages 89–110, 1995.

[57] G. J. Holzmann. The model checker SPIN. *IEEE Trans. Software Engrg.*, 23(5):279–295, 1997.

[58] N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Information and Computation*, 83(2):121–139, Nov. 1989.

[59] D. Janin and I. Walukiewicz. Automata for the modal mu-calculus and related results. In J. Wiedermann and P. Hájek, editors, *Mathematical Foundations of Computer Science 1995, 20th International Symposium, MFCS'95, Prague, Czech Republic, August 28 – September 1, 1995, Proceedings (MFCS)*, pages 552–562, 1995.

[60] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In U. Montanari and V. Sassone, editors, *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277, 1996.

[61] M. Jurdziński. Deciding the winner in parity games is in UP $\cap$ co-UP. *Information Processing Letters*, 68(3):119–124, November 1998.

[62] M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000.

[63] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 117–123. ACM/SIAM, 2006.

[64] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Los Angeles, Calif., 1968.

[65] N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *32nd Annual Symposium on Foundations of Computer Science, 1–4 October 1991, San Juan, Puerto Rico*, pages 358–367, 1991.

[66] D. Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[67] D. Kozen and R. Parikh. A decision procedure for the propositional μ-calculus. In *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 313–325. Springer, 1984.

[68] S. Kreutzer and M. Lange. Non-regular fixed-point logics and games. This volume.

[69] F. Kröger. LAR: A logic of algorithmic reasoning. *Acta Informatica*, 8(3), August 1977.

[70] O. Kupferman, N. Piterman, and M. Y. Vardi. Model checking linear properties of prefix-recognizable systems. In *Proc 14th Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002.

[71] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. Assoc. Comput. Mach.*, 47(2):312–360, 2000.

[72] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. In *ISTCS*, pages 147–158, 1997.

[73] O. Kupferman and M. Y. Vardi. The weakness of self-complementation. In *STACS*, pages 455–466, 1999.

[74] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Logic*, 2(3):408–429, 2001.

[75] O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings (FOCS)*, pages 531–542. IEEE Computer Society, 2005.

[76] D. Kähler. Determinisierung von É-Automaten. Diploma thesis, Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 2001.

[77] L. Lamport. "Sometimes" is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th ACM Symp. on Principles of Programming Languages (POPL)*, pages 174–185, 1980.

[78] M. Lange and C. Stirling. Model checking fixed point logic with chop. In M. Nielsen and U. Engberg, editors, *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 250–263. Springer, 2002.

[79] G. Lenzi. A hierarchy theorem for the μ-calculus. In F. M. auf der Heide and B. Monien, editors, *Automata, Languages and Programming, 23rd International Colloquium, ICALP96, Paderborn, Germany, 8-12 July 1996, Proceedings*, volume 1099 of *Lecture Notes in Computer Science*, pages 87–97. Springer, 1996.

[80] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages"*, pages 97–107, 1985.

[81] C. Löding and W. Thomas. Alternating automata and logics over infinite words. In *Proceedings of the IFIP International Conference*

on Theoretical Computer Science, IFIP TCS2000, volume 1872 of *Lecture Notes in Computer Science*, pages 521–535. Springer, 2000.

[82] D. A. Martin. Borel determinacy. *Ann. Math.*, 102:363–371, 1975.

[83] O. Matz and N. Schweikardt. Expressive power of monadic logics on words, trees, pictures, and graphs. This volume.

[84] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[85] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[86] M. Michel. Complementation is more difficult with automata on infinite words, 1988.

[87] S. Miyano and T. Hayashi. Alternating finite automata on omega-words. In *CAAP*, pages 195–210, 1984.

[88] A. W. Mostowski. Games with forbidden positions. Preprint 78, Uniwersytet Gdańsk, Instytyt Matematyki, 1991.

[89] D. E. Muller. Infinite sequences and finite machines. In *Proceedings of the 4th Annual IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 3–16, 1963.

[90] D. E. Muller and P. E. Schupp. Alternating automata on infinite objects, determinacy and rabin's theorem. In M. Nivat and D. Perrin, editors, *Automata on Infinite Words, Ecole de Printemps d'Informatique Théorique, Le MontDore, May 14–18, 1984*, volume 192 of *Lecture Notes in Computer Science*, pages 100–107. Springer, 1985.

[91] D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of rabin, mcnaughton and safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.

[92] D. Niwiński. On fixed point clones. In L. Kott, editor, *Automata, Languages and Programming: 13th International Colloquium*, volume 226 of *Lecture Notes in Computer Science*, pages 464–473, Rennes, France, 1986. Springer-Verlag, Berlin.

[93] D. Niwinski. Fixed points vs. infinite generation. In *Proceedings, Third Annual Symposium on Logic in Computer Science, 5–8 July 1988, Edinburgh, Scotland, UK (LICS)*, pages 402–409. IEEE Computer Society, 1988.

[94] D. Niwiński and I. Walukiewicz. Ambiguity problem for automata on infinite trees. Unpublished note.

[95] D. Perrin. Recent results on automata on infinite words. In L. Kott, editor, *13th Intern. Coll. on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 134–148, Rennes, France, 1986. Springer-Verlag, Berlin.

[96] D. Perrin and J.-É. Pin. *Infinite Words: Automata, Semigroups, Logic and Games.* Pure and Applied Mathematics. Elsevier, 2003.

[97] N. Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 255–264. IEEE Computer Society, 2006.

[98] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Rhode Island, Providence, 1977. IEEE.

[99] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

[100] M. O. Rabin. Decidability of second-order theories and finite automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[101] M. O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Mathematical Logic and Foundation of Set Theory*, pages 1–23. North Holland, 1970.

[102] K. Reinhardt. The complexity of translating logic to finite automata. In *Automata, Logics, and Infinite Games*, pages 231–238, 2001.

[103] G. S. Rohde. *Alternating automata and the temporal logic of ordinals.* PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[104] R. Rosner. *Modular Synthesis of Reactive Systems.* PhD thesis, Weizmann Institute of Science, 1992.

[105] S. Safra. On the complexity of ω-automata. In *29th Annual Symposium on Foundations of Computer Science*, pages 319–327, White Plains, New York, 1988. IEEE.

[106] P. Schnoebelen. The complexity of temporal logic model checking. In *Advances in Modal Logic*, pages 393–436. King's College Publications, 2002.

[107] H. Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, 1996.

[108] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, 5–7 May 1982, San Francisco, California, USA (STOC)*, pages 159–168. ACM, 1982.

[109] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. Assoc. Comput. Mach.*, 32(3):733–749, 1985.

[110] C. Stirling. *Modal and temporal properties of processes.* Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[111] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic.* PhD thesis, Dept. of Electrical Engineering, MIT, Boston, Mass., 1974.

[112] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Fifth Annual ACM Symposium on Theory of Computation*, pages 1–9, Austin, Texas, 1973. ACM Press.

[113] R. S. Streett. Propositional dynamic logic of looping and converse. *Inform. Contr.*, 54:121–141, 1982.

[114] R. S. Streett and E. A. Emerson. The propositional mu-calculus is elementary. In J. Paredaens, editor, *Automata, Languages and Programming, 11th Colloquium, Antwerp, Belgium, July 16-20, 1984, Proceedings (ICALP)*, volume 172 of *Lecture Notes in Computer Science*, pages 465–472. Springer, 1984.

[115] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order arithmetic. *Mathematical System Theory*, 2(1):57–81, 1968.

[116] W. Thomas. Star-free regular sets of ω-sequences. *Inform. and Control*, 42:148–156, 1979.

[117] W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25:360–376, 1982.

[118] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, pages 134–191. Elsevier, Amsterdam, 1990.

[119] W. Thomas. On logical definability of regular trace languages. In V. Diekert, editor, *Proceedings of a Workshop of the ESPRIT Basic Research Action No. 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS)*, pages 172–182, Kochel am See, BRD, 1990. Bericht TUM-I901902, Technische Universität München.

[120] W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 389–455. Springer, Berlin, 1997.

[121] W. Thomas. Complementation of büchi automata revised. In J. Karhumäki, H. A. Maurer, G. Paun, and G. Rozenberg, editors, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–120. Springer, 1999.

[122] B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Siberian Math. J.*, 3:103–131, 1962. (English translation in: AMS Transl. 59 (1966) 23–55.).

[123] M. Y. Vardi. Automata-theoretic model checking revisited. In *Proc. 7th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 137–150. Springer, 2007.

[124] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In D. Kozen, editor, *First Annual IEEE Symposium on Logic in Computer Science*, pages 322–331, Cambridge, Mass., 16–18 June 1986. IEEE.

[125] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32(2):182–221, 1986.

[126] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 15 Nov. 1994.

[127] M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In P. Gardner and N. Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 512–528. Springer, 2004.

[128] I. Walukiewicz. Monadic second order logic on tree-like structures. In C. Puech and R. Reischuk, editors, *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22–24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pages 401–413, 1996.

[129] I. Walukiewicz. Completeness of kozen's axiomatisation of the propositional μ-calculus. *Inf. Comput.*, 157(1-2):142–182, 2000.

[130] I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

[131] Q. Yan. Lower bounds for complementation of *mega*-automata via the full automata technique. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10–14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 589–600, 2006.

[132] L. D. Zuck. *Past Temporal Logic.* PhD thesis, The Weizmann Institute of Science, Rehovot, Israel, Aug. 1986.