Foreword

Since 2002, FoLLI, the Association for Logic, Language, and Information (www.folli.org), has awarded an annual prize for an outstanding dissertation in the fields of logic, language, and information. The prize is named after the well-known Dutch logician Evert Willem Beth, whose interdisciplinary interests are in many ways exemplary for the aims of FoLLI. It is sponsored by the E.W. Beth Foundation. Dissertations submitted for the prize are judged on technical depth and strength, originality, and impact made in at least two of the three fields of logic, language, and computation. Every year the competition is strong and the interdisciplinary character of the award stimulates lively discussions and debates.

Recipients of the award are given the opportunity to prepare a book version of their thesis for publication in the FoLLI Publications on Logic, Language and Information.

This volume is based on the PhD thesis of Gabriele Puppis, who was the winner of the E.W. Beth dissertation award for 2007. Puppis's thesis focuses on logic and computation and, more specifically, on automata-based decidability techniques for time granularity and on a new method for deciding monadic second-order theories of trees. In the first part of the thesis Puppis defines and systematically exploits various classes of sequential automata in order to solve a number of relevant problems in time granularity (e.g., equivalence, conversion, minimization, optimization, etc.). For each application, he investigates expressiveness and complexity and provides algorithms working on automata-based representations of time granularity. The core of the remaining part of Puppis's thesis is a generalization of the Elgot–Rabin automata-based decision method. He defines a powerful reduction over colored trees and obtains the generalization by overcoming a number of technical difficulties, and thus not only solves the original decision problems, but also gives a precise and inspiring comparison of the newly introduced technique with more classical ones, such as, for example, Shelah's composition method. In both parts of the thesis Dr. Puppis shows mastering and deep understanding of the topic, an elegant and concise presentation of the results, and an insightful overall view on the subject.

The results presented by Puppis represent a significant step towards a better understanding of the changes in granularity levels that humans make so easily in cognition of time, space, and other phenomena, whereas their logical and computational structure poses difficult conceptual and computational challenges.

Alberto Policriti (Chairman of the Beth Prize)
Michael Moortgat (President of the Association for Logic, Language, and Information)

Preface

The aim of the thesis is to exploit different classes of (sequential and tree) automata for modeling and reasoning on infinite complex systems.

The leitmotif underlying the results provided herein is that, given any infinite complex system (e.g., a computer program) to be verified against a finite set of properties, there often exists a simpler system that satisfies the same properties and, in addition, presents strong regularities (e.g., periodicity) in its structure. Those regularities can then be exploited to decide, in an effective way, which property is satisfied by the system and which is not.

Perhaps the most natural and effective way to deal with inherent regularities of infinite systems is through the notion of finite-state automaton. Intuitively, a finite-state automaton is an abstract machine with only a bounded amount of memory at its disposal, which processes an input (e.g., a sequence of symbols) and eventually outputs true or false, depending on the way the machine was designed and on the input itself. The present book focuses precisely on automaton-based approaches that ease the representation of and the reasoning on properties of infinite complex systems.

The most simple notion of finite-state automaton is that of single-string automaton. Such a device outputs true on a single (finite or infinite) sequence of symbols and false on any other sequence. We show how single-string automata processing infinite sequences of symbols can be successfully applied in various frameworks for temporal representation and reasoning. In particular, we use them to model single ultimately periodic time granularities, namely, temporal structures that are left-bounded and that, ultimately, periodically group instants of the underlying temporal domain (a simple example of such a structure is given by the partitioning of the temporal domain of days into weeks). The notion of single-string automaton can be further refined by introducing counters in order to compactly represent repeated occurrences of the same subsequence in the given input. By introducing restricted policies of counter update and by exploiting suitable abstractions of the configuration space for the resulting class of automata, we devise efficient algorithms for reasoning on quasi-periodic time granularities (e.g., the partitioning of the temporal domain of days into years).

Similar abstractions can be used when reasoning on infinite branching (temporal) structures. In such a case, one has to consider a generalized notion of automaton, which is able to process labeled branching structures (hereafter called trees), rather than linear sequences of symbols. We show that sets of trees featuring the same properties can be identified with the equivalence classes induced by a suitable automaton. More precisely, given a property to be verified, one can first define a corresponding automaton that accepts all and only the trees satisfying that property, then introduce a suitable equivalence relation that refines the standard language equivalence and groups all trees being indistinguishable by the automaton, and, finally, exploit such an equivalence to reduce several instances of the verification problem to equivalent simpler instances, which can be eventually decided.

The level of mathematics involved in this book is rigorous, but not inherently difficult. Even though the intended audience is the researcher or the advanced student with background knowledge on algorithm complexity, automata theory, and logics, a motivated non-specialist will be able to follow the presentation, as well as to understand the key ideas behind the various proofs.

For background material on algorithm complexity, automata theory, and logics we refer to [86, 50, 107]

Acknowledgements

I owe many thanks to the many people who assisted me in the process of writing this book. I am particularly indebted to my supervisor Prof. Angelo Montanari, for guidance, suggestions, and feedback he gave me during and after my doctoral studies. I thank the two reviewers of my PhD thesis, Prof. Wolfgang Thomas and Prof. Stphane Demri, as well as two anonymous reviewers, whose comments have greatly helped to improve this manuscript. I thank all my friends, including my colleagues at the University of Udine, for the stimulating discussions and for the pleasant company. Last but not the least, I thank my parents, who gave me the best education and unconditional support over the past years, and my girlfriend, Francesca, for her love, encouragement, and unstinted patience.

Contents

1 Introduction

The present book aims at studying (different classes of) automata to model and reason on infinite complex systems. It focuses on two different notions of automata: word automata and tree automata.

The first class of automata reveals itself as a helpful tool for representing temporal information and for dealing with periodic phenomena. These tasks are widely recognized as relevant ones in a variety of application areas ranging from temporal database design and inter-operability to data conversion and data mining, to the specification and verification of reactive systems, to the synthesis, execution, and monitoring of timed workflow systems, to temporal constraint representation and reasoning, and to temporal abstraction [3]. One of the most effective attempts at dealing with these problems takes advantage of the notion of time granularity.

Different time granularities can be used to specify the occurrence times of different classes of events. For instance, temporal characterizations of a flight departure, a business appointment, and a birthdate are usually given in terms of minutes, hours, and days, respectively. Moreover, when a computation involves pieces of information expressed at different time granularities, the system needs the ability of properly relating time granularities (this is usually the case, for instance, with query processing in federated database systems). Such an integration presupposes the formalization of the notion of granularity and the analysis of the relationships between different time granularities. According to a commonly accepted perspective, any time granularity can be viewed as the partitioning of a temporal domain in groups of elements, where each group is perceived as an indivisible unit (a granule). Most granularities of practical interest are modeled as infinite sequences of time granules, which present a repeating pattern and, possibly, temporal gaps within and between granules. A representation formalism can then use these granules to provide facts, actions or events with a temporal qualification, at the appropriate abstraction level. Even though conceptually clean, this point of view does not address the problem of providing infinite granularities with a finite (and compact) representation to make it possible to deal with them in an effective (and efficient) way. To be computationally acceptable, any formal system for time granularity should indeed satisfy the following requirements:

- **Suitable to algorithmic manipulation.** The formalism must provide infinite granularities with a finite representation. Furthermore, data structures, which are used to actually store information, should ease access to and manipulation of time granularities.
- **Powerful.** The set of all possible time granularities is not countable. Consequently, every representation formalism is bound to be incomplete. The class of granularities captured by the formalism should be expressive enough to be of practical interest.
- **Compact.** The formalism should exploit regularities exhibited by the considered time granularities to make their representation as compact as possible.

In Chapter 2, we exploit the notion of word automaton, possibly equipped with counters, to devise suitable formalisms for the representation and the management of time granularities. The proposed automaton-based approach to time granularity turns out to be at least as expressive as the formalism of Calendar Algebra and Wijsen's string-based models. Moreover, in developing automaton-based formalisms, we focus on some crucial problems, like, for instance, granule conversion (that is, the problem of relating granules of a given granularity to those of another one), equivalence (that is, the problem of deciding whether two given representations define the same time granularity), optimization (that is, the problem of computing compact and/or tractable representations of time granularities), and granularity comparison (that is, the problem of deciding whether, for two given sets $\mathcal{G}$ and $\mathcal{H}$ of time granularities, represented by means of suitable automata, there exist granularities $\mathsf{G} \in \mathcal{G}$ and $\mathsf{H} \in \mathcal{H}$ that satisfy a prescribed relation). Finally, we provide some real-world applications of automaton-based representations of time-granularities.

The notion of tree automaton comes into play in the automatic verification of properties of infinite transition systems. A natural approach to this problem is to model a system as a directed graph, whose vertices (resp., edges) represent system configurations (resp., transitions). An expected property of the system is then expressed by a logical formula, which can be satisfied or not by the corresponding graph, thought of as a relational structure. In this perspective, the verification problem reduces to the model checking problem, namely, the problem of deciding the truth of a given formula interpreted over a fixed relational structure. Monadic second-order logic has been extensively used as a specification language, because it is powerful enough to express relevant properties of graph structures such as reachability, planarity, vertex k-colorability (for any fixed k), and confluency properties [22] and it subsumes, besides first-order logic, many propositional (temporal) modal logics, in particular the propositional modal μ-calculus [51]. Unfortunately, the model checking problem for monadic second-order logic turns out to be highly undecidable for many structures. However, if one restricts monadic second-order formulas to be evaluated over tree structures, then the model checking problem can be reduced to the acceptance problem for Rabin tree automata [44], namely, the problem of deciding whether a given tree automaton accepts a fixed relational structure, viewed as a deterministic vertex-colored tree. Such a problem is easily proved to be decidable in the case of regular trees.

In Chapter 3, we develop a general method, called contraction method, for establishing the decidability of the model checking problem for monadic second-order logic interpreted over (possibly non-regular) deterministic vertex-colored trees. More precisely, we exploit a suitable notion of tree equivalence in order to reduce a number of instances of the acceptance problem for tree automata to simpler (i.e., decidable) instances. The resulting framework allows one to transfer decidability results from simple (e.g., regular) trees to more complex (e.g., non-regular) structures. We show that such a method works effectively for a large class of trees, called reducible trees, which naturally extends the class of regular trees and includes all deterministic trees in the Caucal hierarchy as well as interesting relational structures outside it. Finally, we consider the model checking problem for the chain fragment of monadic second-order logic interpreted over the so-called layered temporal structures, which are tree-shaped structures well-suited for modeling and reasoning on temporal relationships at different 'grain levels'.

The results of Section 2.3, dealing with the class of nested counter single-string automata, have been originally presented in [26] and later refined in [27, 28]. The results of Section 2.4, dealing with the class of ultimately periodic automata, have been published in [5]. A preliminary version of the results presented in Chapter 3 has been published in [72] and later refined in [74]. Finally, the results of Section 3.5.4, dealing with the monadic-second order theories of layered temporal structures, appeared in [73, 70].

2

Word Automata and Time Granularities

The relevance of the problem of managing periodic phenomena is widely recognized in different areas of computer science. One of the most effective attempts to manage periodic phenomena takes advantage of the notion of time granularity, which can be thought of as the partitioning of a subset of a temporal domain into groups of elements, where each group is perceived as an indivisible unit (a granule). This chapter provides a detailed account of various automaton-based formalisms that cope with the notion of periodicity and that of time granularity.

We first review the notion of single-string automaton [25], which is a very restricted form of Büchi automaton accepting a single infinite word. Single-string automata provide the first basic formalism for representing and reasoning on ultimately periodic time granularities, that is, temporal structures that, starting from a given point, periodically group instants of the underlying temporal domain.

We then consider the possibility of equipping automata with counters in order to succinctly encode the inherent redundancies (e.g., repeated occurrences of patterns) of time granularities. Such an investigation leads to the notion of counter single-string automaton, which is a single-string automaton extended with counters and multiple transitions, whose activation rules take into account both the state and the values of the counters. On the one hand, counter single-string automata yield compact representations of time granularities of practical interest (e.g., the granularity of years of the Gregorian Calendar). On the other hand, counter single-string automata are not suited for algorithmic manipulations. As an example, the equivalence problem for such a class of automata, which consists of deciding whether two given automata specify the same time granularity, turns out to be PSPACE-complete.

A trade-off between the handiness of plain single-string automata and the compactness of counter single-string automata can be obtained by introducing suitable restrictions on the structure of the automata and by adopting a uniform policy of counter update. We exploit such an idea in the definition of nested counter single-string automaton. By taking advantage of the restrictions enforced on the structure of these automata, we are able to devise improved algorithms for a number of crucial problems related to the notion of time granularity. As an example, we show that, in many relevant cases (i.e., those in which there are no gaps within and between granules), the granule conversion problem, that is the problem of relating the granules of a given granularity to those of another one, can be solved in polynomial time with respect to the size of the involved automaton-based representations.

Subsequently, we extend the automaton-based approach to make it possible to deal with (possibly infinite) sets of ultimately periodic time granularities, rather than single time granularities. To this end, we introduce a suitable class of automata, called ultimately periodic automata, that captures rational languages consisting of ultimately periodic words only. Ultimately periodic automata can be used to represent single time granularities, (possibly infinite) sets of time granularities that feature the same repeating pattern, but different prefixes, sets of time granularities characterized by a finite set of non-equivalent repeating patterns (a formal notion of equivalence for repeating patterns will be given in the sequel), as well as any possible combination of them. Moreover, ultimately periodic automata form a proper subclass of Büchi automata and they inherit interesting properties from non-deterministic finite-state automata. Taking advantage of the similarities among ultimately periodic automata, Büchi automata, and non-deterministic finite-state automara, we devise efficient solutions to a number of basic problems involving sets of time granularities. In particular, we provide a solution to the crucial problem of granularity comparison, that is, the problem of deciding, given two sets of granularities $\mathcal{G}$ and $\mathcal{H}$, whether there exist a granularity $\mathsf{G} \in \mathcal{G}$ and a granularity $\mathsf{H} \in \mathcal{H}$ such that $\mathsf{G} \sim \mathsf{H}$, where $\sim$ is one of the commonly-used relations between granularities, e.g, partition, group, refinement, aligned refinement.

The chapter is organized as follows. In Section 2.1, we briefly recall some basic definitions and results about words, languages, periodicity, automata, and time granularities. In Section 2.2, we introduce the string-based and automaton-based approaches for the management of time granularity. In Section 2.3, we fully exploit the notion of single-string automaton extended with counters to compactly represent and efficiently reason on ultimately periodic time granularities. In particular, we provide efficient solutions to the granule conversion problem and the equivalence problem and we describe suitable optimization procedures for automaton-based representations of time granularities. In Section 2.4, we focus on the problem of representing and reasoning on (possibly infinite) sets of ultimately periodic time granularities, rather than single ones. We give a precise characterization of the class of rational languages of ultimately periodic words and we introduce the class of ultimately periodic automata, which recognize precisely the rational languages of ultimately periodic words. Finally, we exploit well-known results from classical automata theory to provide simple, but efficient, solutions to the emptiness, acceptance, equivalence, inclusion, and state optimization problems for ultimately periodic automata. Section 2.5 provides an assessment of the achieved results and outlines future research directions, with a special emphasis on possible improvements of the proposed algorithms.

2.1 Background Knowledge

In this section, we recall some basic definitions and results about words, languages, periodicity, automata, and time granularities.

2.1.1 Words and Languages

Hereafter, we call *alphabet* any finite set A of symbols. A *finite word* over an alphabet A is a finite sequence of symbols from A, that is, a mapping of the form $w : \{1, ..., n\} \to A$, where $\{1, ..., n\}$ is an initial segment of the set $\mathbb{N}_{>0}$ of all positive natural numbers. Given a finite word $w : \{1, ..., n\} \to A$, we denote by $|w|$ its *length* n. We further denote by ε the special empty word, which has length 0. An *infinite* (ω-) *word* over A is an infinite sequence of symbols from A, that is, a mapping of the form $w : \mathbb{N}_{>0} \to A$. For convenience, we assume that the length of an infinite word w is the ordinal ω. Given a finite (resp., infinite) word w and an index $1 \leqslant i \leqslant |w|$ (resp., $i \geqslant 1$), we denote by $w(i)$ the i-th symbol of w

Concatenations of (finite or infinite) words are defined as follows. Given a finite word u and a finite (resp., infinite) word v, we denote by $u\,v$ the finite (resp., infinite) word w of length $|w| = |u| + |v|$ (resp., ω) such that, for every $1 \leqslant i \leqslant |w|$ (resp., $i \geqslant 1$),

$$w(i) = \begin{cases} u(i) & \text{if } i \leqslant |u|, \\ v(i - |u|) & \text{if } i > |u|. \end{cases}$$

Note that the concatenation operation is associative, that is, $(u\,v)\,z = u\,(v\,z)$ holds for every pair of finite words u, v and for every (finite or infinite) word z. This allows us denote by $u_1\,u_2 \ldots u_n$ any finite sequence of concatenations of words u_1, u_2, ..., u_n.

We generalize the notion of concatenation to infinite sequences as follows. Given a (possibly empty) finite word u_i, for every $i \geqslant 1$, the infinite concatenation $u_1\,u_2\,u_3 \ldots$ is defined as the (finite or infinite) word w of length $\sum_{i \geqslant 1} |u_i|$ such that, for every $1 \leqslant i \leqslant |w|$ ($i \geqslant 1$ if $|w| = \omega$), $w(i)$ is the j-th symbol of the k-th word u_k, where k is the (unique) index in $\mathbb{N}_{>0}$ such that $\sum_{h<k} |u_h| < i \leqslant \sum_{h \leqslant k} |u_h|$ and j is the position $i - \sum_{h<k} |u_h|$.

Given a finite word u and a natural number n, we shortly denote by u^n the finite word that results from the finite concatenation $\underbrace{u\,u \ldots u}_{n \text{ times}}$ (for convenience, we assume that $u^n = \varepsilon$ if $n = 0$). Similarly, we denote by u^ω word that results from the infinite concatenation $u\,u\,u \ldots$ (note that u^ω is empty whenever u is empty and u^ω is infinite whenever u is non-empty). As an example, we have $(ab)^\omega = ababab...$.

Given a finite (resp., infinite) word w and two indices i, j such that $i \geqslant 1$ and $j \leqslant |w|$ (resp., $i \geqslant 1$, $j < |w|$), we denote by $w[i, j]$ the *substring* $w(i)w(i+1)...w(j)$ of w (if $i > j$, then $w[i, j]$ is assumed to be the empty word ε). A *prefix* of a finite (resp., infinite) word w is either the empty word ε, the word w itself, or a non-empty finite substring of the form $w[1, j]$, with $1 \leqslant j < |w|$ (resp., $j \geqslant 1$). A *suffix* of a finite word w is either the word ε, the word w itself, or a non-empty substring of the form $w[i, |w|]$, with $1 \leqslant i \leqslant |w|$; similarly, a suffix of an infinite word w is either the word w itself or an infinite substring of the form $w[i, \omega[= w(i)w(i+1)w(i+2)...)$, with $i \geqslant 1$.

Finally, we recall basic definitions related to languages of words. Given an alphabet A, we call *language* (resp., ω-language) over A any set of finite (resp., infinite) words over A. A language L is *prefix-closed* (resp., *prefix-free*) if, for every word $u \in L$ and every proper prefix v of u, $v \in L$ (resp., $v \notin L$).

Set-theoretic operations over (ω-)languages are defined in the standard way (e.g., $U \cup V$ denotes the union of two languages U and V). The operation of concatenation of words is extended to (ω-)languages as follows. Given two languages U and V (resp., a language U and an ω-language V), we denote by UV the set of all finite (resp., infinite) words of the form $w = uv$, with $u \in U$ and $v \in V$. Given a language U, we denote by U^* (resp., U^+) the set of all finite words of the form $w = u_1 u_2 ... u_n$, with $n \in \mathbb{N}$ (resp., $n \in \mathbb{N}_{>0}$) and $u_i \in U$ for all $1 \leqslant i \leqslant n$. Similarly, we denote by U^ω the set of all infinite words of the form $w = u_1 u_2 u_3 ...$, with $u_i \in U$ for all $i \geqslant 1$. Note that, by definition, $U^* = U^+ \cup \{\varepsilon\}$ and U^ω contains no finite word (even if $\varepsilon \in U$). This implies that, for any alphabet A, the sets A^*, A^+, and A^ω contains, respectively, all possibly empty finite words over A, all non-empty finite words over A, and all infinite words over A.

2.1.2 Periodicity of Words

Let u be a non-empty finite word and let w be a non-empty finite (resp., infinite) word. We say that u is a *repeating pattern* of w and $|u|$ is a *period* of w if we have $w = u^k$ for some $k \geqslant 1$ (resp., if we have $w = u^\omega$). In addition, we say that u is the *primitive* repeating pattern of w and $|u|$ is the *minimum* period of w, if u is the *shortest* repeating pattern of w.

An infinite word w is said to be *ultimately periodic* if it can be written as uv^ω, where u is a finite word and v is a finite non-empty word. By a slight abuse of terminology, we say that u and v are respectively an *initial pattern* and a *repeating pattern* of w.

There also exist generalized notions of repeating pattern and period that characterize partial repetitions of substrings. Precisely, given two non-empty finite words u and w, we say that u is a *partial repeating pattern* of w and $|u|$ is a *partial period* of w if w is a prefix of u^ω. Clearly, the periods of w are exactly those partial periods of w that divide $|w|$. In analogy with previous

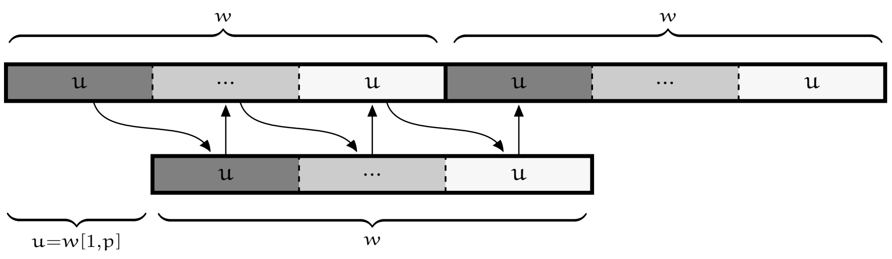


Fig. 2.1. The minimal period of a word

definitions, we say that u is the *primitive* partial repeating pattern of w if it is the shortest partial repeating pattern of w. As an example, the word $u = abc$ is the primitive partial repeating pattern of $w = abcabcab$.

Note that, given a word w (e.g., $w = abababab$), there may exist many different repeating patterns of w (e.g., $u_1 = ab$, $u_2 = abab$, $u_3 = abababab$) and many different partial repeating patterns of w (e.g., $u'_1 = ab$, $u'_2 = abab$, $u'_3 = ababab$, $u'_4 = abababab$). However, the primitive repeating pattern of w and the primitive partial repeating pattern of w are always unique.

The following lemma, which is a straightforward generalization of Fine-Wilf's periodicity lemma [41], shows that the minimum period (resp., the minimum partial period) of a word w is the greatest common divisor of all periods (resp., partial periods) of w.

Lemma 1. *Given a non-empty finite word* w*, if* p *and* q *are partial periods of* w *and* $p + q \leqslant |w|$*, then* $\gcd(p, q)$ *is a partial period of* w *as well.*

Proof. We prove the claim by induction on $p + q$. Assume that $p < q$ and denote by r the value $q - p$. Since p and q are both partial periods of w, for every $1 \leqslant i \leqslant |w| - q$, we have $w(i) = w(i + q) = w(i + q - p) = w(i + r)$. Similarly, for every $|w|-q+1 \leqslant i \leqslant |w|-r$, $w(i) = w(i-p)$ (since $|w| \geqslant p+q$) and $w(i-p) = w(i+q-p) = w(i+r)$ hold. This shows that r is partial period of w. Moreover, since $p+r < p+q \leqslant |w|$, we can apply the inductive hypothesis to the partial periods p, r and hence conclude that $\gcd(p, r)$ $(= \gcd(p, q))$ is a partial period of w. □

Below, we review some classical algorithms that allow one to compute the primitive (partial) repeating pattern and the minimum (partial) period of a given word w. Let us fix a non-empty finite word w.

As a first remark, note that p is the minimum period of w, and hence $u = w[1, p]$ is the primitive repeating pattern of w, if and only if p is the offset of the *first non-trivial occurrence* of w as a substring of ww, namely, p is the least positive natural number such that $w = w[1 + p, |w|]\, w[1, p]$. An intuitive explanation of such a property is given in Figure 2.1.

Given the above property, one can exploit Knuth-Morris-Pratt string matching algorithm [56] to compute, in linear time $\mathcal{O}(|w|)$, the primitive repeating pattern and the minimum period of the word w. In fact, the same

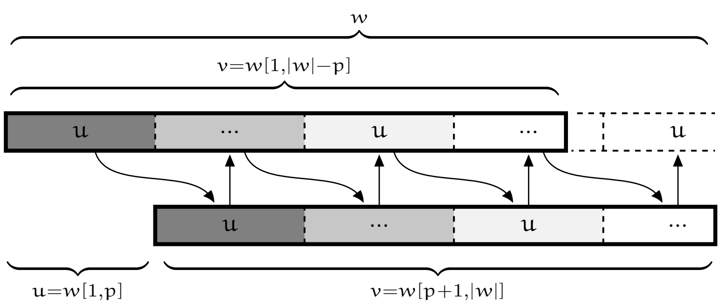


Fig. 2.2. Relationship between partial periods and borders

algorithm can be used to compute, in linear time $\mathcal{O}(|w|)$, the primitive repeating pattern and the minimum period of *every prefix* of w.

The primitive partial repeating pattern and the minimum partial period of (every prefix of) w can be computed using similar ideas. Let us call *border* of a non-empty finite word w any (possibly empty) finite word v ($\neq w$) which is both a proper prefix of w and a proper suffix of w. For convenience, we assume that the unique border of the empty word is ε. As an example, the words $v_1 = \varepsilon$, $v_2 = ab$, and $v_3 = abcab$ are borders of $w = abcabcab$.

Given a finite word w, it is easy to see that p is a partial period (resp., the minimum partial period) of w if and only if $w[1, |w| - p]$ is a border (resp., the longest border) of w. Figure 2.2 gives an intuitive account of such a property.

The following proposition provides a characterization of the borders of w in terms of the borders of the proper prefixes of w.

Proposition 1. *Let* $u_0, ..., u_n$ *be a sequence of finite words such that* $u_0 = \varepsilon$ *and, for every* $0 \leqslant i < n$, u_i *is the longest border of* u_{i+1}. *Then, for every symbol* a *and every finite word* v, *we have that* v *is the longest border of* $w = u_n\, a$ *if and only if there exists an index* $0 \leqslant i \leqslant n$ *such that (i)* $v = u_i\, a$, *(ii)* $w(|u_i| + 1) = a$, *and (iii)* $w(|u_j| + 1) \neq a$ *for every* $j > i$.

Proof. Suppose that v is the longest border of $w = u_n a$. Since v is prefix of w, we know that $v[1, |v| - 1]$ is a prefix of u_n. Similarly, since v is a suffix of w, we know that $v[1, |v| - 1]$ is a suffix of u_n. This shows that $v[1, |v| - 1]$ is a border of u_n and hence $v[1, |v] - 1| = u_i$ for some index $0 \leqslant i \leqslant n$. Moreover, since v is a suffix of $u_n a$, we have $v(|v|) = a$. This proves the first property $v = u_i a$. Similarly, since v is a prefix of w, we have $w(|v|) = w(|u_i| + 1) = a$, which proves the second property. As for the last property, suppose, by way of contradiction, that $w(|u_j| + 1) = a$ for some index $j > i$. It follows that $v' = u_j a$ is both a prefix of w and a suffix of w, which contradicts the hypothesis that v is the *longest* border of w.

As for the converse implication, let i be the greatest index such that $w(|u_i|+1) = a$ and let $v = u_i a$. Since u_i is a prefix of u_n and $w(|u_i|+1) = a$, we know that v is a prefix of w. Similarly, since u_i is a suffix of u_n, v is a suffix of w.

This proves that v is a border of w. We now prove that v is the *longest* border of w. Suppose, by way of contradiction, that there exists another border v' of w such that $|v'| > |v|$. Since v' is a prefix of w, we know that $v'[1, |v'| - 1]$ is a prefix of u_n. Moreover, since v' is a suffix of w, we know that $v'[1, |v'| - 1]$ is suffix of u_n and $v'(|v'|) = w(|w|) = a$. This shows that $v'[1, |v'| - 1]$ is a border of u_n and hence $v' = u_j a$ for some index $j > i$. It thus follows that $w(|u_j| + 1) = v'(|v'|) = a$, which contradicts the hypothesis that i is the *greatest* index such that $w(|u_i| + 1) = a$. □

We now describe a simple procedure, based on dynamic programming, that computes the longest border of a given finite word w. For the sake of brevity, for every $0 \leqslant n \leqslant |w|$ and every $i > 1$, we denote by $r(n)$ the minimum partial period of the prefix $w[1, n]$ of w and by $r^i(n)$ the i-fold iteration of $r(n)$, which is defined by $r^1(n) = r(n)$ and $r^{i+1}(n) = r\big(r^i(n)\big)$.

Proposition 1 implies the following recursive equation for the function r:

$$r(n) = \begin{cases} 0 & \text{if } n = 0, \\ \max\big\{0, r^i(n-1) + 1 \,:\, i > 0,\ w\big(r^i(n-1) + 1\big) = w(n)\big\} & \text{if } n > 0. \end{cases}$$

It is worth pointing out that the above-defined function r coincides with the so-called 'prefix function' (or 'failure function') of w, which is used in the algorithm of Knuth, Morris, and Pratt to compute the partial matchings of w in a given text. In [56] an linear time procedure that computes the function r is provided. For the sake of clearness, we briefly describe such a procedure (see Algorithm 2.1) and we recall its complexity analysis.

Algorithm 2.1. ComputePrefixFunction(w)

```
$r(0) \leftarrow 0$
for $n \leftarrow 1$ to $|w|$
    do  $r' \leftarrow r(n-1)$
        while $r' > 0$ and $w(r'+1) \neq w(n)$        (2.1a)
          do   $r' \leftarrow r(r')$
        if $w(r'+1) = w(n)$
          then $r(n) \leftarrow r' + 1$
          else $r(n) \leftarrow 0$
```

We use amortized analysis (see, for instance, [19]) to show that Algorithm 2.1 runs in linear time $\mathcal{O}(|w|)$. First of all, note that the variable r' decreases at least by 1 at each iteration of the inner loop 2.1a. Moreover, before entering such a loop, r' is assigned value $r(n-1)$ and, after exiting the loop, r' is assigned value $r(n) + 1$. This shows that the number of iterations of the loop 2.1a, for a fixed n, is at most $r(n-1) - r(n) + 1$. Therefore, the total time required to execute the above algorithm is $\mathcal{O}\big(\sum_{1 \leqslant n \leqslant |w|}(r(n-1) - r(n) + 1)\big)$, which is linear in $|w|$.

Finally, we remark that Algorithm 2.1 can be used to compute, in linear time $\mathcal{O}(|w|)$, the minimal partial periods of all prefixes of a given finite word w. The algorithms described in this section will be extensively used in the sequel.

2.1.3 Word Automata

A word automaton is an abstract finite-state machine that receives a (finite or infinite) word as input, processes it symbol by symbol, from left to right, and eventually accepts or rejects it depending on the existence of a successful 'computation' (hereafter called run). We first introduce automata processing finite words.

Definition 1. *A* non-deterministic finite-state automaton *(NFA for short), is a tuple $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$, where*

- A *is a finite alphabet,*
- S *is a finite set of states,*
- $\Delta \subseteq S \times A \times S$ *is a transition relation,*
- $\mathcal{I} \subseteq S$ *is a set of initial states,*
- $\mathcal{F} \subseteq S$ *is a set of final states.*

A *run* of a non-deterministic finite-state automaton $\mathcal{A}$ on a finite word $w \in A^*$ is a finite sequence ρ of states such that (i) $|\rho| = |w| + 1$ and (ii) $\big(\rho(i),\ w(i),\ \rho(i+1)\big) \in \Delta$ for every $1 \leqslant i \leqslant |w|$. A run ρ is said to be *successful* if we have $\rho(1) \in \mathcal{I}$ and $\rho(|\rho|) \in \mathcal{F}$, namely, if the first and the last state of ρ belong, respectively, to the set of initial states and to the set of final states of $\mathcal{A}$. Note that there may exist different successful runs of $\mathcal{A}$ on the same word w.

We say that $\mathcal{A}$ *accepts* a finite word $w \in A^*$ if there is a successful run ρ of $\mathcal{A}$ on w; otherwise, we say that $\mathcal{A}$ *rejects* w. The language *recognized* by $\mathcal{A}$ is defined as the set $\mathscr{L}(\mathcal{A})$ of all finite words $w \in A^*$ that are accepted by $\mathcal{A}$.

Given a language $L \subseteq A^*$, we say that L is *rational*[1] if it is recognized by an NFA, namely, if there is a non-deterministic finite-state automaton $\mathcal{A}$ such that $\mathscr{L}(\mathcal{A}) = L$. It is worth pointing out that DFA are equally expressive as NFA, namely, for every NFA $\mathcal{A}$, there is an equivalent DFA $\mathcal{A}'$ such that $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\mathcal{A}')$ (a very simple proof of such a result, which exploits the so-called 'subset construction' of NFA, can be found in [50]).

[1] In the literature, the term 'regular' is often used as a synonym for 'rational'. However, to avoid any confusion, we shall use the term 'rational' to refer to languages recognized by automata and the term 'regular' to refer to special forms of trees, which will be introduced in Section 3.1.1.

A *deterministic finite-state automaton* (DFA for short) is an NFA $\mathcal{A} = (\mathsf{A}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F})$, where the component Δ is a function from $\mathsf{S} \times \mathsf{A}$ to S and the component $\mathcal{I}$ is a singleton (therefore, differently form NFA, a DFA admits at most one successful run on each input word w). It is easy to see that for every NFA $\mathcal{A}$, there is an equivalent DFA $\mathcal{A}'$ that recognizes the same language [50].

Automata processing infinite words are defined in a similar way, with the only exception of the acceptance condition, which now envisages the states that occur *infinitely often* along a run. In particular, we adopt Büchi acceptance condition, keeping in mind that alternative acceptance conditions exist, e.g., parity acceptance condition, Muller acceptance condition, etc. (we refer the reader to [77] for the details).

Definition 2. *A* Büchi automaton*, is a tuple* $\mathcal{A} = (\mathsf{A}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F})$*, where*

- A *is a finite alphabet,*
- S *is a finite set of states,*
- $\Delta \subseteq \mathsf{S} \times \mathsf{A} \times \mathsf{S}$ *is a transition relation,*
- $\mathcal{I} \subseteq \mathsf{S}$ *is a set of initial states,*
- $\mathcal{F} \subseteq \mathsf{S}$ *is a set of final states.*

A *run* of a Büchi automaton $\mathcal{A}$ on an infinite word $w \in \mathsf{A}^\omega$ is an infinite sequence ρ of states such that, for every $i \geqslant 1$, $\big(\rho(i),\, w(i),\, \rho(i+1)\big) \in \Delta$. A run ρ is said to be *successful* if we have $\rho(1) \in \mathcal{I}$ and $\rho(i) \in \mathcal{F}$ for infinitely many indices $i \geqslant 1$.

In analogy with the previous definitions, we say that $\mathcal{A}$ *accepts* an infinite word $w \in \mathsf{A}^\omega$ if there is a successful run ρ of $\mathcal{A}$ on w; otherwise, we say that $\mathcal{A}$ *rejects* w. The ω-language *recognized* by $\mathcal{A}$ is defined as the set $\mathscr{L}^\omega(\mathcal{A})$ of all infinite words $w \in \mathsf{A}^\omega$ that are accepted by $\mathcal{A}$.

Given an ω-language $\mathsf{L} \subseteq \mathsf{A}^\omega$, we say that L is *rational* if it is recognized by a Büchi automaton. It is well-known that deterministic Büchi automata are less expressive than non-deterministic ones. As an example, the rational ω-language $\{a, b\}^*\{a\}^\omega$, which consists of all infinite words over the alphabet $\{a, b\}$ that contain only finitely many occurrences of b, cannot be recognized by any deterministic Büchi automaton.

From now on, we denote by $|\mathcal{A}|$ the *size* of a word automaton $\mathcal{A}$, which is usually defined as the number of states and transitions of $\mathcal{A}$ (if the automaton $\mathcal{A}$ is extended with counters, as it happens, for instance, with the automata described in Section 2.2.3 and in Section 2.3, then the size $|\mathcal{A}|$ comprises also the size of the constants used to update and compare the values of the counters).

We refer the reader to [50, 107] for additional material on word automata.

2.1.4 Time Granularities

According to a commonly accepted perspective [15], any time granularity can be viewed as the partitioning of a subset of a temporal domain into groups of

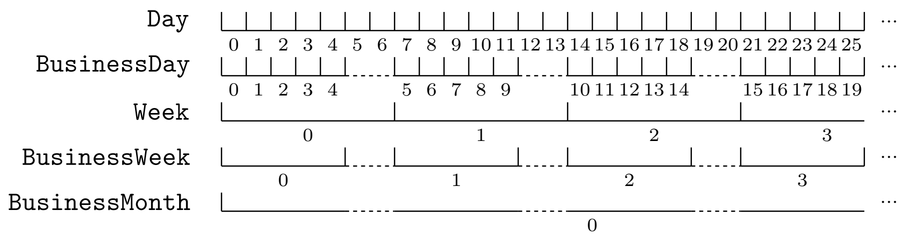


Fig. 2.3. Some examples of time granularities

elements, where each group is perceived as an indivisible unit (a granule). Such a notion comes into play in a variety of computer science scenarios, including time representation and management in database applications, specification and verification of temporal properties of reactive systems, temporal representation and reasoning in artificial intelligence. In particular, time granularity is a key concept in any representation formalism that provides facts, actions, or events with a temporal qualification. To give a few examples, the occurrence time of a flight departure, a business appointment, and a birthday are often specified in terms of the granularity minutes, hours, and days, respectively.

As it happens in most application domains, we assume the underlying temporal domain to be (isomorphic to) the linear order $(\mathbb{N}, <)$ of the natural numbers.

Definition 3. *A* time granularity *is a collection* $\mathsf{G} \subseteq \mathscr{P}(\mathbb{N})$ *of subsets of the temporal domain such that distinct sets in* G *(henceforth called* granules*) do not overlap, namely, for every pair of distinct granules* $\mathsf{g}, \mathsf{g}' \in \mathsf{G}$*, we have either* $\mathsf{t} < \mathsf{t}'$ *for all* $\mathsf{t} \in \mathsf{g}$ *and* $\mathsf{t}' \in \mathsf{g}'$ *or* $\mathsf{t}' < \mathsf{t}$ *for all* $\mathsf{t} \in \mathsf{g}$ *and* $\mathsf{t}' \in \mathsf{g}'$*.*

The above definition captures both time granularities that cover the whole temporal domain, such as `Day`, `Week`, and `Month`, and time granularities with gaps within (e.g., `BusinessMonth`) and between granules (e.g., `BusinessDay` and `BusinessWeek`). Figure 2.3 depicts some of these granularities.

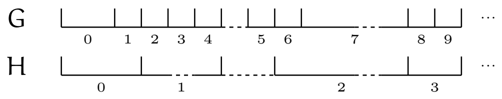
Note that the order on the elements of the temporal domain $\mathbb{N}$ induces a similar order on the granules of a granularity G. Thus, given two granules $\mathsf{g}, \mathsf{g}' \in \mathsf{G}$, we can write $\mathsf{g} < \mathsf{g}'$ whenever $\mathsf{t} < \mathsf{t}'$ holds for every $\mathsf{t} \in \mathsf{g}$ and $\mathsf{t}' \in \mathsf{g}'$. Such an order naturally yields a labeling of the granules of G: we say that $x \in \mathbb{N}$ is the *index* of a granule $\mathsf{g} \in \mathsf{G}$, and we write $\mathsf{G}(x) = \mathsf{g}$, if g is the $x + 1$-th element of G according to the induced order on the granules.

It is worth pointing out that, in the literature, one can find slightly different notions of time granularity. As an example, in [3, 81], a time granularity is defined as a mapping from $\mathbb{Z}$ to subsets of an arbitrary linearly ordered temporal domain (e.g., $(\mathbb{Q}, <)$). Compared to Definition 3, such a notion is more general in that (i) it allows both right-unbounded granularities and left-unbounded ones and (ii) it envisages the existence of sets of arbitrarily fine granularities. However, the usual restrictions that one enforces to the calendric systems in order to ease algorithmic manipulation make the two definitions actually equivalent.

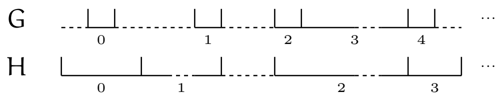
Relationships between Time Granularities

The ability to properly relate the information expressed at different time granularities (this is usually the case, for instance, for query processing in federated database systems), presupposes the analysis of various relationships between time granularities. In the following, we summarize the most relevant ones (a comprehensive survey can be found in [3]).

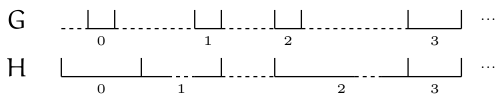
- **Grouping.** A granularity G *groups into* a granularity H (equivalently, H *groups* G) if every granule h of H is a union of granules of G. The following figure represents two granularities G and H such that G groups into H. Note that the granule $\mathsf{G}(3)$ and the granule $\mathsf{G}(5)$ are not included in any granule of H.



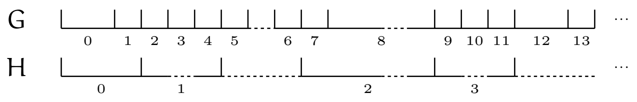
- **Refinement.** A granularity G *refines* a granularity H if every granule of G is contained in some granule of H. The following figure represents two granularities G and H such that G refines H.



- **Partition.** A granularity G *partitions* a granularity H if G both groups into H and refines H.
- **Aligned refinement.** A granularity G is an *aligned refinement* of H if for every label x, the $x+1$-th granule of G is included in the $x+1$-th granule of H. The following figure represents two granularities G and H such that G is an aligned refinement of H.



- **Periodic grouping.** A granularity G *groups periodically into* H (equivalently, H *periodically groups* G) if G groups into H and there are two positive natural numbers p and q such that, for every label x and for every sequence of labels $y_1, y_2, ..., y_n$, $\mathsf{H}(x) = \bigcup_{1 \leqslant i \leqslant n} \mathsf{G}(y_i)$ iff $\mathsf{H}(x+q) = \bigcup_{1 \leqslant i \leqslant n} \mathsf{G}(y_i + p)$. Roughly speaking, G groups periodically into H if the granules of G are grouped into granules of H with the same 'repeating pattern'. The following figure represents two granularities G and H such that G groups periodically into H (with parameters $p = 7$ and $q = 2$).



A Summary of Various Formalisms

It is immediate to realize that the set of all structures that satisfy Definition 3 becomes uncountable as soon as the underlying temporal domain is infinite. As a consequence, it is not possible to deal with all these structures by means of a finitary formalism. However, the problem of mastering time granularities can be tackled in an effective way by restricting to those structures that group periodically, with finitely many exceptions, a fixed bottom granularity. As a matter of fact, such a restriction is at the basis of almost any formalism for representing time granularities. Below, we survey the most important formalisms proposed in the literature.

Among algebraic approaches, the most widely known are perhaps the formalism of collection expressions [60], that of slice expressions [80], and the Calendar Algebra [81]. All of them capture large sets of time granularities, including ultimately periodic ones, by means of algebraic expressions (e.g., $\texttt{Group}_7(\texttt{Day})$). The different sets of algebraic operators provided by the three formal systems and their expressiveness are investigated in [3], where it is proved that Calendar Algebra actually subsumes the other two formalisms. However, a common limitation of these formalisms is that they focus on expressiveness issues and almost completely ignore some basic problems, like that of establishing the equivalence of two given representations, of obvious theoretical and practical importance [42, 40].

A logical account of Calendar Algebra has been provided by Combi et al. in [17]. The idea is to identify time granularities with models of suitable formulas in propositional Linear Time Logic (LTL for short) [36], where the propositional variables are used to mark the starting and ending points of granules. The expressiveness of LTL makes it possible to represent, beside single time granularities, sets of time granularities that start at arbitrary time points (these are called *unanchored* time granularities in [46]). Furthermore, problems like checking the consistency of a granularity specification and testing the equivalence of two granularity expressions can be solved in a uniform way by reducing them to the satisfiability problem for LTL, which is known to be decidable in polynomial space [98, 103].

A somehow similar, but less expressive, framework has been developed in [83, 82], where a propositional logic extended with interval-based modalities (e.g., $[2000, \mathit{year}]$, $\langle 2000, \mathit{year}\rangle$) is used to specify temporal relationships between events. The resulting logic, called Calendar Logic, can be translated into propositional logic, thus showing that the problem of checking consistency of granularity specifications is decidable. In addition, since the translation from Calendar Logic to propositional logic is exponential, an alternative tableau-based decision procedure, which has more possibilities for guiding and optimizing the proof search, has been also developed.

Another logical framework for dealing with integer periodicity constraints, and, in particular, with time granularities has been described in [33]. Such a framework is based on a fragment of Presburger Linear Temporal Logic, denoted $\mathrm{PLTL}^{\mathrm{mod}}$, which is obtained from LTL by replacing propositional variables with formulas in a suitable first-order constraint language (a strict fragment of Presburger Arithmetic). Such a combination of logical languages makes it possible to express both qualitative and quantitative temporal aspects and to compactly represent periodicities of time granularities. Moreover, like plain LTL (but differently from full Presburger LTL, which is known to be highly undecidable), $\mathrm{PLTL}^{\mathrm{mod}}$ enjoys a PSPACE-complete satisfiability problem. As a consequence of such a result, several automaton-theoretic problems (among all the equivalence problem for single-string automata extended with counters [25]) are proved to be in PSPACE.

An alternative approach to the representation and manipulation of time granularities has been proposed in [113]. Such an approach models infinite left-bounded time granularities as sequences of symbols taken from a fixed finite alphabet. More precisely, it shows that any ultimately periodic granularity can be identified with an ultimately periodic word $w = u v^{\omega}$, which can then be represented by a pair, called granspec, consisting of the finite initial pattern u and the finite repeating pattern v.

Finally, an automaton-based approach to time granularity has been originally proposed in [25] and later revisited in [26, 27, 28]. Such an approach identifies time granularities with ultimately periodic words generated by a specific class of automata, called single-string automata, thus making it possible to (re)use well-known results from automata theory.

In Section 2.2 we give a short but comprehensive account of the string-based and automaton-based approaches.

2.2 The String-Based and Automaton-Based Approaches

In this section, we highlight the key features of the string-based and automaton-based formalisms for representing and reasoning on time granularities. In particular, we show that, on the one hand, granspecs are as expressive as single-string automata, on the other hand, single-string automata can be easily extended with counters to obtain succinct representations of time granularities.

2.2.1 The Granspec Formalism

In [113], Wijsen shows that time granularities over the temporal domain $(\mathbb{N}, <)$ can be naturally expressed in terms of infinite words over a fixed alphabet consisting of three symbols, namely, $\blacksquare$ (filler), $\square$ (gap), and $\wr$ (separator), which are respectively used to denote time points covered by some granule, to denote time points not covered by any granule, and to delimit granules. For instance, the granularity `BusinessWeek` can be encoded by the infinite word $\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square\wr\ \blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square\wr\ldots$ or, equivalently, by the infinite word $\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\wr\square\square\ \blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\wr\square\square\ldots$

In order to enforce a one-to-one correspondence between infinite words over $\{\blacksquare, \square, \wr\}$ and time granularities, Wijsen introduces suitable *aligned forms*, in which separators are forced to occur immediately after an occurrence of $\blacksquare$. Moreover, as observed in [25], one can encode each occurrence of the substring $\blacksquare\wr$ by a single symbol $\blacktriangleleft$. In the following, we shall adopt this simplified setting to represent time granularities.

Definition 4. *Given an infinite word* w *over the alphabet* $\{\blacksquare, \square, \blacktriangleleft\}$, *we say that* w represents *a time granularity* G *if, for every pair of natural numbers* t, x, *we have* $t \in \mathsf{G}(x)$ *iff* $w(t+1) \in \{\blacksquare, \blacktriangleleft\}$ *and the substring* $w[1,t]$ *contains exactly* x *occurrences of the symbol* $\blacktriangleleft$.

According to the above definition, one can identify time granularities with infinite words over $\{\blacksquare, \square, \blacktriangleleft\}$. Moreover, as long as one confines oneself to ultimately periodic time granularities, finite representations based on pairs consisting of an initial pattern and a repeating pattern can be adopted.

Definition 5. *A* granspec *is a pair of the form* (u, v), *where* u *is a (possibly empty) finite word over the alphabet* $\{\blacksquare, \square, \blacktriangleleft\}$ *and* v *is a non-empty finite word over* $\{\blacksquare, \square, \blacktriangleleft\}$.

We say that the granspec (u, v) represents time granularity G if the ultimately periodic word $w = uv^\omega$ represents G.

Although it is common practice to let the repeating pattern of a granspec coincide with granule boundaries, Wijsen's formalism allows repeating patterns starting or ending in the middle of a granule. For instance, the time granularity `BusinessWeek` can be equivalently represented by the granspecs $(\varepsilon, \blacksquare\blacksquare\blacksquare\blacksquare\blacktriangleleft\square\square)$ and $(\blacksquare\blacksquare, \blacksquare\blacksquare\blacktriangleleft\square\square\blacksquare\blacksquare)$.

A simple solution to the granspec equivalence problem takes advantage of the notion of canonical form. Formally, a granspec (u, v) is said to be in *canonical form* if it has minimum size $|u| + |v|$ among all equivalent granspecs. In [113], it is proved that, for every ultimately periodic granularity G, there exists exactly one canonical granspec (u, v) that represents G. This allows one to reduce the granspec equivalence problem to the problem of computing canonical forms of granspecs. We conclude the section by showing how the latter problem can be solved in linear time.

First of all, it is easy to see that a granspec (u, v) is in canonical form if and only if it satisfies the following two conditions:

i) u and v do not end with the same symbol,

ii) v is primitive, namely, it is not a repetition of a smaller substring.

Now, given any granspec (u, v), one can compute, in linear time, the longest (possibly empty) word z such that $u = x\,z$ and $v = y\,z$, for some $x, y \in \{\blacksquare, \square, \blacktriangleleft\}^*$. Note that $u\,v^\omega = (x\,z)\,(y\,z)^\omega = x\,(z\,y)^\omega$ and hence the pair $(x,\ z\,y)$ is a granspec equivalent to (u, v) which satisfies the first condition. Moreover, using the algorithm described in Section 2.1.2, one can compute the primitive repeating pattern v' of $z\,y$. Finally, one observes that (x, v') is a granspec in canonical form equivalent to (u, v).

2.2.2 From Granspecs to Single-String Automata

The idea of viewing time granularities as ultimately periodic words naturally connects the notion of time granularity to the fields of formal languages and automata. The basic idea underlying the automaton-based approach to time granularity is the following one: we take a Büchi automaton $\mathcal{A}$ recognizing a *single* infinite word w over the alphabet $\{\blacksquare, \square, \blacktriangleleft\}$ (hence the name single-string automaton) and we say that $\mathcal{A}$ represents the granularity G whenever w represents G. Such an idea has been systematically explored in [25, 26, 27, 28].

Definition 6. *A* single-string automaton *(SSA for short) is a quadruple* $\mathcal{A} = (A, S, \delta, s_0)$*, where*

- A *is a finite alphabet (usually the set* $\{\blacksquare, \square, \blacktriangleleft\}$*),*
- S *is a finite set of states,*
- $\delta : S \to A \times S$ *is a transition function,*
- $s_0 \in S$ *is an initial state.*

Below, given a pair $t = (s, a)$, with $s \in S$ and $a \in A$, we shortly denote by $\downarrow_1 t$ (resp., $\downarrow_2 t$) the first element s (resp., the second element a) of t. The *run* of $\mathcal{A}$ is defined as the (unique) infinite sequence ρ of pairs from $S \times A$ such that

- $\downarrow_1 \rho(1) = s_0$,
- $\delta\big(\downarrow_1 \rho(i)\big) = \big(\downarrow_2 \rho(i),\ \downarrow_1 \rho(i+1)\big)$ for every $i \geqslant 1$.

The (unique) infinite word *recognized* by $\mathcal{A}$ is given by the sequence $\downarrow_2 \rho$ $(= \downarrow_2 \rho(1)\ \downarrow_2 \rho(2)\ \downarrow_2 \rho(3) \ldots)$. As an example, Figure 2.4 depicts an SSA representing the granularity `BusinessWeek` over the temporal domain of days.

It is immediate to see that SSA over the alphabet $\{\blacksquare, \square, \blacktriangleleft\}$ are as expressive as granspecs, namely, they capture all and only the (encodings of) ultimately periodic time granularities. Moreover, the equivalence problem for SSA-based representations of time granularities can be easily reduced to the equivalence problem for granspecs and thus it can be solved in linear time. Indeed, given

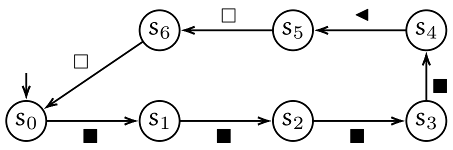


Fig. 2.4. An SSA representing `BusinessWeek`

two SSA $\mathcal{A}_1$ and $\mathcal{A}_2$ recognizing the ultimately periodic words $w_1 = u_1 v_1^\omega$ and $w_2 = u_2 v_2^\omega$, respectively, we have $w_1 = w_2$ if and only if the two granspects (u_1, v_1) and (u_2, v_2) are equivalent.

2.2.3 Counters and Multiple Transitions

A major limitation of the automaton-based approach, as well as of the string-based one, is that, whenever the granularity to be represented has a long initial pattern and/or a long repeating pattern, it produces lengthy representations. As an example, recall that leap years recur with exactly the same pattern every 400 years; then, it is easy to see that the size of any granspec/SSA representing years (or months) of the Gregorian Calendar in terms of days must have size greater than 10^5. In addition, computations involving these lengthy representations of time granularities may become rather expensive.

In the following, we extend and refine the automaton-based approach by introducing counters and multiple transitions in order to compactly encode redundancies of temporal structures. Precisely, we distinguish between two kinds of transition, respectively called primary and secondary transitions, and we exploit the possibility of activating different transitions from the same (control) state. However, at any point of the computation, at most one (primary or secondary) transition is taken according to an appropriate activation rule that envisages the values of counters. Such an activation rule is, by convention, as follows: secondary transitions are taken whenever they are enabled, otherwise, primary transitions are taken.

Definition 7. *A* counter single-string automaton *(CSSA for short) is a tuple* $\mathcal{A} = (A, I, S, \delta, \gamma, s_0, c_0)$*, where*

- A *is a finite alphabet,*
- I *is a finite set of counters, usually denoted by* $i, j, k, \ldots$*, whose* valuations *belong to the set* $\mathcal{C}_I = \mathbb{N}^I$ *of functions from* I *to* $\mathbb{N}$*,*
- S *is a finite set of (control) states,*
- $\delta : S \to A \times S \times \mathcal{C}_I^{\mathcal{C}_I}$ *is a* primary transition *function, which maps every state* $s \in S$ *to a triple of the form* (a, r, f)*, where* $a \in A$*,* $r \in S$*, and* $f : \mathcal{C}_I \to \mathcal{C}_I$ *is an* update operator*,*
- $\gamma : S \to \mathscr{P}(\mathcal{C}_I) \times A \times S \times \mathcal{C}_I^{\mathcal{C}_I}$ *is a* secondary transition *function, which maps every state* $s \in S$ *to a quadruple of the form* (C, a, r, f)*, where* $C \subseteq \mathcal{C}_I$ *is a* guard*,* $a \in A$*,* $r \in S$*, and* $f : \mathcal{C}_I \to \mathcal{C}_I$ *is an update operator,*

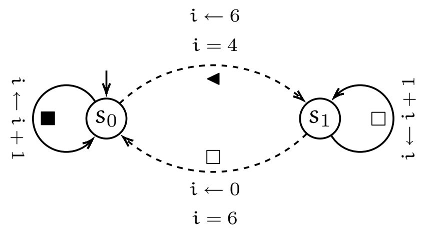


Fig. 2.5. An CSSA representing `BusinessWeek`

- $s_0 \in S$ *is an initial state,*
- $c_0 \in \mathcal{C}_I$ *is an initial valuation.*

As in the case of single-string automata, we assume that any CSSA has a unique run. In order to formally define it, we need to introduce the notion of configuration. A *configuration* for $\mathcal{A}$ is a pair state-valuation (s, c), where $s \in S$ and $c \in \mathcal{C}_I$. The transitions of $\mathcal{A}$ are taken according to a global transition function $[\delta, \gamma] : S \times \mathcal{C}_I \to A \times S \times \mathcal{C}_I$ such that, for every configuration $(s, c) \in S \times \mathcal{C}_I$,

$$[\delta,\gamma](s,c) = \begin{cases} \big(a, r, f(c)\big) & \text{if } \delta(s) = (a, r, f) \text{ and } \gamma(s) \text{ is not defined,} \\ \big(a, r, f(c)\big) & \text{if } \delta(s) = (a, r, f),\ \gamma(s) = (C, a', r', f'), \text{ and } c \notin C, \\ \big(a, r, f(c)\big) & \text{if } \gamma(s) = (C, a, r, f) \text{ and } c \in C. \end{cases}$$

Intuitively, a secondary transition is activated whenever its guard is satisfied by the current valuation; otherwise, a primary transition is activated.

The (unique) *run* of $\mathcal{A}$ is then defined as the infinite sequence ρ of triples from $S \times \mathcal{C}_I \times A$ such that

- $\downarrow_1 \rho(1) = s_0$,
- $\downarrow_2 \rho(1) = c_0$,
- $[\delta,\gamma]\big(\downarrow_1 \rho(i),\ \downarrow_2 \rho(i)\big) = \big(\downarrow_3 \rho(i),\ \downarrow_1 \rho(i+1),\ \downarrow_2 \rho(i+1)\big)$ for every $i \geqslant 1$.

The (unique) infinite word *recognized* by $\mathcal{A}$ is given by the sequence $\downarrow_3 \rho = \downarrow_3 \rho(1)\ \downarrow_3 \rho(2)\ \downarrow_3 \rho(3) \ldots$.

As an example, Figure 2.5 depicts a CSSA representing the time granularity `BusinessWeek`. Such an automaton uses one counter i, which is initialized to 0. Control states are represented by circles, while primary and secondary transitions are represented by solid and dashed arrows, respectively, which are annotated with recognized symbols (e.g., ■) and update operators (e.g., $i \leftarrow i + 1$); the guards of the secondary transitions are specified as additional annotations, having the form of constraints like $i = 4$, of the corresponding arrows.

A more interesting example is given in Figure 2.6, which depicts a CSSA representing the time granularity `Month` over the temporal domain of days.

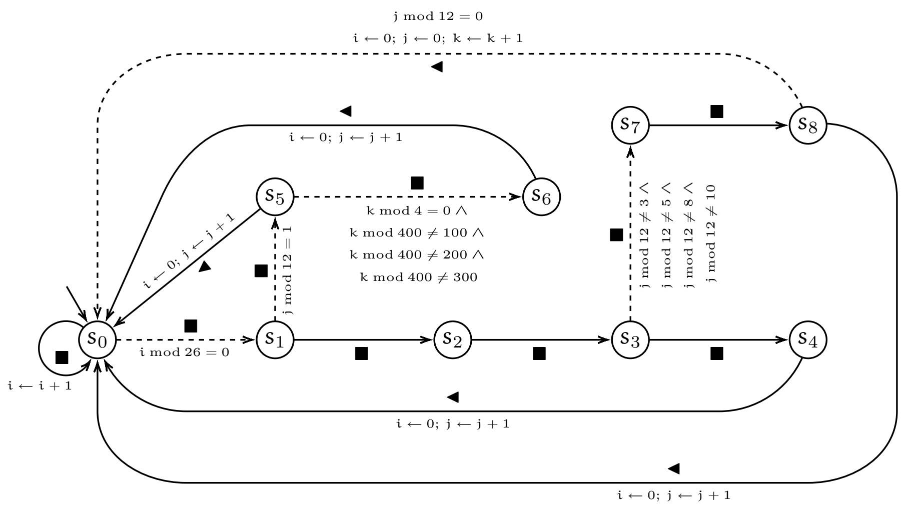


Fig. 2.6. An CSSA representing `Month`

The automaton uses three counters, i, j, and k, to store the index of the current day, current month, and current year, respectively. For the sake of simplicity, secondary transitions with empty guards are not depicted.

The above examples make it clear how counters can be exploited to compactly encode redundancies of time granularities. However, the notion of CSSA is too general to be of practical interest: if we do not restrict the form of admissible guardsa and update operators for primary and secondary transitions, several basic problems on CSSA turn out to be undecidable. As an example, if we allow guards of the form $i = 0$ and update operators of the form $i \leftarrow i - 1$ and $i \leftarrow i + 1$, then CSSA can be viewed as special forms of Minsky machines [64] and thus their halting problem turns out to be undecidable. In [25], the decidability of basic problems on CSSA has been recovered by

i) restricting to guards that result from conjunctions of basic constraints of the form $t_1 = t_2$ or $t_1 \neq t_2$, where t_1 and t_2 are integer constants or terms like $i \bmod d$, with $i \in I$ and $d \in \mathbb{N}_{>0}$;
ii) restricting to update operators that result from functional compositions of basic operators of the form $i \leftarrow 0$ or $i \leftarrow i+1$, where i is a counter and $i \leftarrow 0$ (resp., $i \leftarrow i + 1$) denotes the function that maps any valuation c to the valuation $c[0/i]$ (resp., $c[c(i) + 1/i]$), with $c[x/i]$ denoting the valuation defined by $c[x/i](i) = x$ and $c[x/i](j) = c(j)$ for all $j \neq i$.

The automata resulting from the above restrictions are called *Reducible Counter Single-String Automata* (RCSSA for short). It turns out that many granularities of practical interest are represented by RCSSA with a few states and counters. As an example, the automaton depicted in Figure 2.6 can be viewed as a RCSSA.

The class of RCSSA is well behaved with respect to the decidability of basic problems, like, for instance, the equivalence problem. This follows from the fact that RCSSA enjoy finite bisimilarity quotients and hence they belong to the first class of symbolic transition systems according to the classification introduced by Henzinger and Majumdar [49]. As a matter of fact, this also proves that any RCSSA can be effectively translated into an equivalent SSA. For the sake of completeness, below we explicitly describe the translation from RCSSA to SSA. From such a result and from the fact that the equivalence problem for SSA is solvable in linear time, we have that the equivalence problem for RCSSA is in EXPTIME.

Proposition 2. *Any RCSSA $\mathcal{A}$ can be effectively translated into an equivalent SSA $\mathcal{A}'$ having size at most exponential in the size of $\mathcal{A}$ (note that the size of $\mathcal{A}$ comprises the size of the binary expansion of every constant that appears inside a guard of a secondary transition of $\mathcal{A}$).*

Proof. Let $\mathcal{A} = (A, I, S, \delta, \gamma, s_0, c_0)$ be a RCSSA. For each counter $i \in I$, we denote by d_i the *least common multiple* of all constants d that appear inside the guards of the secondary transition function γ. We define a binary relation $\approx$ over the configuration space $S \times \mathcal{C}_I$ of $\mathcal{A}$ such that, for every pair of configurations (s, c) and (s', c'), $(s, c) \approx (s', c')$ holds if and only if $s = s'$ and $c(i) \bmod d_i = c'(i) \bmod d_i$ for all $i \in I$. It is easy to see that $\approx$ is an equivalence relation of finite index and, in particular, there exist at most $|S| \prod_{i \in I} d_i$ different $\approx$-equivalence classes. Moreover, $\approx$ is a congruence with respect to the global transition function $[\delta, \gamma]$ of $\mathcal{A}$, namely, for every pair of configurations (s, c) and (s', c'), $(s, c) \approx (s', c')$ implies $[\delta, \gamma](s, c) \approx [\delta, \gamma](s', c')$. This shows that the SSA $\mathcal{A}' = (A, S', \delta', s'_0)$, where

- S' is the set of all $\approx$-equivalence classes of the form $[s, c]_\approx = \big\{(s', c') : (s, c) \approx (s', c')\big\}$, with $(s, c) \in S \times \mathcal{C}_I$,
- $\delta'\big([s, c]_\approx\big) = [s', c']_\approx$ whenever $[\delta, \gamma](s, c) = (s', c')$,
- s'_0 is the $\approx$-equivalence class $[s_0, c_0]_\approx$ of the initial configuration (s_0, c_0),

is equivalent to the RCSSA $\mathcal{A}$. Finally, note that the number of states of $\mathcal{A}'$ (and hence the size of $\mathcal{A}'$) is at most exponential in the size of $\mathcal{A}$. □

2.2.4 The Logical Counterpart of RCSSA

In [33] Demri describes a logical framework that allows one to express, in a concise way, integer periodicity constraints over a discrete linear temporal domain. The formalism is based on a fragment of Presburger Linear Temporal Logic, denoted PLTL$^{\text{mod}}$, which is obtained by combining a suitable first-order constraint language IPC^{++} with the standard linear temporal logic with past-time operators.

The formulas of the first-order constraint language IPC^{++} are built up, using standard boolean connectives and existential quantifications, from basic formulas of the form $x = d$, $x < d$, $x > d$, $x = y$, $x \equiv_k d$, and $x \equiv_k y + [d_1, d_2]$, where $x, y, ...$ are variables interpreted over $\mathbb{Z}$ and $d, k, d_1, d_2, ...$ are integer constants. Given a valuation $c : \{x, y, ...\} \rightarrow \mathbb{Z}$ for the variables $x, y, ...$, the semantics of a basic formula is defined in the following natural way:

- $c \vDash (x = d)$ iff $c(x) = d$,
- $c \vDash (x < d)$ iff $c(x) < d$,
- $c \vDash (x > d)$ iff $c(x) > d$,
- $c \vDash (x = y)$ iff $c(x) = c(y)$,
- $c \vDash (x \equiv_k d)$ iff $c(x) \bmod k = d$,
- $c \vDash (x \equiv_k y + [d_1, d_2])$ iff $c(x) \bmod k = \big(c(y) + d\big) \bmod k$ for some $d_1 \leqslant d \leqslant d_2$ (as a matter of fact, the formula $x \equiv_k y + [d_1, d_2]$ can be viewed as a shorthand for the disjunction $\bigvee_{d_1 \leqslant d \leqslant d} x \equiv_k y + d$).

Given the above definition, it is clear that IPC^{++} turns out to be a strict fragment of Presburger Arithmetic [45].

The Linear Temporal Logic (LTL) with past-time operators is the logic that comprises, among standard propositional variables and boolean connectives, the 'next' modal operator $\mathtt{X}$, the 'always in the future' modal operator $\mathtt{G}$, the 'always in the past' modal operator $\mathtt{H}$, the 'until' modal operator $\mathtt{U}$, and the 'since' modal operator $\mathtt{S}$. We refer the reader to [36] for further details about the semantics of LTL formulas and to [99, 110] for the decidability of the satisfiability and model checking problems for LTL.

We now define the logical language PLTL$^{\text{mod}}$ as the fragment of Presburger LTL which is obtained from the temporalization of IPC^{++} via LTL with past-time operators. Formally, let $\varphi(y_1, ..., y_k)$ be a formula of IPC^{++}, let $\mathtt{X}^i\, x_j$ be the value of a variable x_j at the i-th successor of the current time point, and let $\varphi\big[\mathtt{X}^{i_1}\, x_{j_1}, ..., \mathtt{X}^{i_k}\, x_{j_k}\big]$ be the formula obtained from $\varphi(y_1, ..., y_k)$ by replacing every free occurrence of y_l by $\mathtt{X}^{i_l}\, x_{j_l}$, for all $1 \leqslant l \leqslant k$. Formulas of PLTL$^{\text{mod}}$ are obtained from LTL formulas by replacing propositional variables by formulas of the form $\varphi\big[\mathtt{X}^{i_1}\, x_{j_1}, ..., \mathtt{X}^{i_k}\, x_{j_k}\big]$. Given an infinite sequence $\bar{c} : \mathbb{N} \times \{x, y, ...\} \rightarrow \mathbb{Z}$ of valuations for the variables $\{x, y, ...\}$, the semantics of a formula of PLTL$^{\text{mod}}$ is the obvious one.

Example 1. As an example, we show the encoding of some granularities of the Gregorian Calendar taken from [33]. These granularities are modeled as infinite sequences of valuations for the corresponding integer variables as follows:

- $\mathit{sec} = 0 \;\wedge\; \mathtt{G}(0 \leqslant \mathit{sec} < 60) \;\wedge\; \mathtt{G}\big((\mathtt{X}\,\mathit{sec}) \equiv_{60} (\mathit{sec} + 1)\big)$;
- $\mathit{min} = 0 \;\wedge\; \mathtt{G}(0 \leqslant \mathit{min} < 60) \;\wedge\; \mathtt{G}\big(\mathit{sec} = 59 \;\rightarrow\; (\mathtt{X}\,\mathit{min}) \equiv_{60} (\mathit{min} + 1)\big) \;\wedge\; \mathtt{G}\big(\mathit{sec} < 59 \;\rightarrow\; (\mathtt{X}\,\mathit{min}) = \mathit{min}\big)$;
- $\mathit{hour} = 0 \;\wedge\; \mathtt{G}(0 \leqslant \mathit{hour} < 24) \;\wedge\; \mathtt{G}\big(\mathit{min} = 59 \;\wedge\; \mathit{sec} = 59 \;\rightarrow\; (\mathtt{X}\,\mathit{hour}) \equiv_{24} (\mathit{hour} + 1)\big) \;\wedge\; \mathtt{G}\big(\mathit{min} < 59 \;\vee\; \mathit{sec} < 59 \;\rightarrow\; (\mathtt{X}\,\mathit{hour}) = \mathit{hour}\big)$;

- $\mathit{weekday} = 0 \ \wedge\ \mathtt{G}(0 \leqslant \mathit{weekday} < 7) \ \wedge\ \mathtt{G}\big(\mathit{hour} = 23 \ \wedge\ \mathit{min} = 59 \ \wedge\ \mathit{sec} = 59 \ \rightarrow\ (\mathtt{X}\,\mathit{weekday}) \equiv_7 (\mathit{weekday} + 1)\big) \ \wedge\ \mathtt{G}\big(\mathit{hour} < 23 \vee \mathit{min} < 59 \vee \mathit{sec} < 59 \rightarrow (\mathtt{X}\,\mathit{weekday}) = \mathit{weekday}\big)$;
- as for the granularities of months and years, one can easily encode them by fixing some end dates far ahead in the time line (such an assumption is necessary since we cannot use constraints like $(\mathtt{X}\,\mathit{year}) = (\mathit{year} + 1)$ without incurring in undecidability [18, 33]).

Note that the logic PLTL$^{\text{mod}}$ does not envisage the use of propositional variables. However, it can easily encode a propositional variable p by an integer variable x_p, whose value is constrained to range over the set $\{0, 1\}$ by means of a suitable IPC^{++} formula. As a consequence, PLTL$^{\text{mod}}$ turns out to be a well suited logical language for expressing both qualitative and quantitative temporal constraints.

Moreover, like plain LTL, but unlike full Presburger LTL, PLTL$^{\text{mod}}$ enjoys a PSPACE-complete satisfiability problem [33]. Such a result is achieved by first defining suitable automaton-based representations for (abstracted) models of PLTL$^{\text{mod}}$ formulas and then reducing the satisfiability problem to the emptiness problem for the resulting class of automata.

In [33] an interesting connection between RCSSA and LTL with integer periodicity constraints is established. To this end, the guards associated with RCSSA secondary transitions are rewritten as boolean combinations of formulas like $x \equiv_k d$ and $\exists\, z.\ (x \equiv_k z \ \wedge\ y \equiv_h z)$ and thus they are shown to belong to a (strict) fragment of IPC^{++}, denoted IPC*. Then, the equivalence problem for RCSSA is reduced to the satisfiability problem for the Presburger LTL fragment PLTL*, which is defined as the temporalization of IPC*. Finally, the latter problem is solved by reducing it to the emptiness problem for a suitable class of Büchi automata, where the input symbols are atomic IPC* formulas.

Since the above reductions can be computed in polynomial space, the equivalence problem for RCSSA is shown to be in PSPACE. Such a result improved the previously known EXPTIME upper bound [25]. Moreover, in [33] a reduction from the satisfiability problem for quantified boolean formulas to the equivalence problem for RCSSA has been given, thus proving that the equivalence problem for RCSSA is actually PSPACE-complete.

2.3 Compact and Tractable Representations

In this section, we introduce a new class of automata, called nested counter single-string automata, which is an attempt to find a suitable trade-off between the handiness of SSA and the compactness of (reducible) CSSA. Restricted labeled single-string automata are similar to CSSA, since they exploit counters to compactly encode redundancies of time granularities. However, the distinctive feature of this class of automata lies in the structure of the transitions, which is now more restricted (as an example, we adopt a uniform

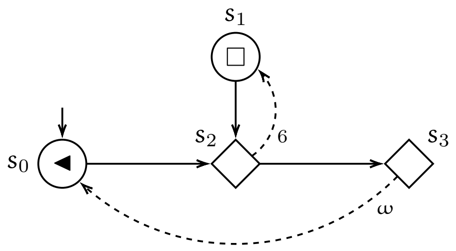


Fig. 2.7. An NCSSA representing `Monday`

policy of counter update). By exploiting these restrictions, we are able to devise improved algorithms for several problems on time granularities, including granule conversion, equivalence, and optimization.

We first give an intuitive description of the structure and behavior of nested counter single-string automata. First of all, to simplify the notation and the formalization of properties, labels are moved from transitions to states. The set of control states is then partitioned into two groups, respectively denoted by S_A and S_ε. S_A is the set of states where the labeling function is defined, while S_ε is the set of states where it is not defined. Exactly one counter is associated with each unlabeled state. Moreover, like in the case of CSSA, we distinguish between primary and secondary transitions. Primary transitions can depart from any state, while secondary transitions depart from unlabeled states only. A primary transition is activated in an unlabeled state s only once the secondary transition associated with s has been consecutively taken $c_0(s)$ times, where $c_0(s)$ denotes the initial valuation for the counter associated with s.

Figure 2.7 depicts an NCSSA recognizing the word $(◀\square^6)^\omega$, which represents the granularity `Monday` over the temporal domain of days. The labeled states (i.e., s_0 and s_1) are represented by circles labeled with symbols from $A = \{■, \square, ◀\}$; the unlabeled states (i.e., s_2 and s_3) are represented by diamonds. Primary and secondary transitions are represented by continuous and dashed arrows, respectively. The (initial values of) the counters are associated with the corresponding states in S_ε (for the sake of readability, we annotate them along the dashed arrows exiting from unlabeled states).

Definition 8. *A* Nested Counter Single-String Automaton *(NCSSA for short) is a tuple* $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$, *where*

- S_A *and* S_ε *are disjoint finite sets of (control) states (we shortly denote by* S *the set* $S_A \cup S_\varepsilon$*),*
- A *is a finite alphabet,*
- $\Omega : S_A \to A$ *is a* labeling function,
- $\delta : S \rightharpoonup S$ *is a partial function, called* primary transition *function,*
- $\gamma : S_\varepsilon \to S$ *is a total function, called* secondary transition *function,*

- $s_0 \in S$ *is an initial state,*
- $c_0 : S_\varepsilon \rightarrow \mathbb{N} \cup \{\omega\}$ *is an initial valuation.*

Hereafter, we assume that the transition functions δ and γ of any NCSSA $\mathcal{A}$ satisfy the following two conditions:

i) $(s, s) \notin \delta^+$ for all $s \in S$, namely, the transitive closure δ^+ of δ is irreflexive,
ii) $(\gamma(s), s) \in \delta^+$ for all $s \in S_\varepsilon$, namely, the source state of any secondary transition is reachable from the target state through a path that consists of primary transitions only.

Note that the above conditions enforce a nesting of the structure of primary and secondary transition functions. In the sequel, we will see how one can take advantage of such a property to devise efficient algorithms that solve basic problems involving NCSSA.

Counters of NCSSA range over the set $\mathbb{N}$ extended with a special value ω. During a computation, they can be either set to their initial value or decremented (we tacitly assume that $n < \omega$ for all $n \in \mathbb{N}$ and $\omega - 1 = \omega$). Let $\mathcal{C}_S$ be the set of all valuations of the form $c : S_\varepsilon \rightarrow \mathbb{N} \cup \{\omega\}$. A *configuration* of an NCSSA $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$ is a pair of the form (s, c), with $s \in S$ and $c \in \mathcal{C}_S$. The transitions of $\mathcal{A}$ are taken according to the partial *global transition* function $[\delta, \gamma] : S \times \mathcal{C}_S \rightharpoonup S \times \mathcal{C}_S$, which is defined as follows:

$$[\delta,\gamma](s,c) = \begin{cases} \big(\delta(s), c\big) & \text{if } s \in S_A \text{ and } \delta(s) \text{ is defined,} \\ \big(\delta(s), c[c_0(s)/s]\big) & \text{if } s \in S_\varepsilon,\ c(s) = 0, \text{ and } \delta(s) \text{ is defined,} \\ \big(\gamma(s), c[c(s) - 1/s]\big) & \text{if } s \in S_\varepsilon \text{ and } c(s) > 0, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Intuitively, the above definition of global transition function implies that

1. if the automaton lies in a labeled state and a primary transition exits from there, then the automaton takes the primary transition and does not change the valuation;
2. if the automaton lies in an unlabeled state whose counter has value 0, then the automaton takes the primary transition exiting from that state (if such a transition is defined) and it re-initializes the counter;
3. if the automaton lies in an unlabeled state whose counter has a positive value, then the automaton takes the secondary transition and it decrements the counter by 1;
4. if none of the above conditions holds, then the automaton halts.

The run of an NCSSA $\mathcal{A}$ is defined on the basis of its global transition function $[\delta, \gamma]$ (note that, since $[\delta, \gamma]$ may be not defined on some configurations, the run of $\mathcal{A}$ may be finite). For the sake of brevity, given a finite set X, we denote by $X^{*+\omega}$ the set $X^* \cup X^\omega$, which consists of all (finite or infinite) words over X. The (unique) *run* of $\mathcal{A}$ is defined as the *maximal* (possibly infinite) sequence $\rho \in (S \cup \mathcal{C}_S)^{*+\omega}$ such that

- $\downarrow_1 \rho(1) = s_0$,
- $\downarrow_2 \rho(1) = c_0$,
- $[\delta, \gamma]\big(\downarrow_1 \rho(i),\ \downarrow_2 \rho(i)\big) = \big(\downarrow_1 \rho(i+1),\ \downarrow_2 \rho(i+1)\big)$ for every $1 \leqslant i < |\rho|$.

Given the run ρ of $\mathcal{A}$, we can extract the maximal (finite or infinite) subsequence ρ_A of ρ that contains configurations of the form (s, c), with $s \in S_A$ and $c \in \mathcal{C}_S$. The sequence ρ_A is called *labeled run* of $\mathcal{A}$. Finally, the unique (finite or infinite) word *recognized* by $\mathcal{A}$ is defined as the sequence $\Omega\big(\downarrow_1 \rho_A\big) = \Omega\big(\downarrow_1 \rho_A(1)\big)\ \Omega\big(\downarrow_1 \rho_A(2)\big)\ \Omega\big(\downarrow_1 \rho_A(3)\big) \ldots$.

Note that the values of the counters in the run of an NCSSA $\mathcal{A}$ range over finite domains, namely, they belong either to the singleton $\{\omega\}$ or to an initial segment $\{0, ..., c_0(s)\}$ of the natural numbers. Thus, it is immediate to see that NCSSA recognize either finite words or ultimately periodic words.

In the following, we show that the nested structure of the transition functions of an NCSSA is closely related to the nested repetitions featured by the recognized word. Successively, we shall take advantage of such a property to address the fundamental problems of granule conversion, equivalence, and optimization for NCSSA-based representations of time granularities.

2.3.1 Nested Repetitions of Words

As already pointed out, the distinctive feature of NCSSA is the way they encode nested repetitions of words. In order to disclose the relationships between these repetitions and the structure of the transition functions, we need to introduce some preliminary definitions.

First of all, note that Definition 8 allows situations where states and transitions of an NCSSA form an unconnected directed graph. Moreover, the initial valuation of a counter may be 0, which implies that the associated secondary transition is never activated during the computation. We can overcome these clumsy situations by discarding useless states and transitions and by assuming that the initial valuation c_0 is positive on every unlabeled state.

We now define the δ-*degree* and the γ-*degree* of the states of an NCSSA. Let $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$ be an NCSSA. For each state $s \in S$, the δ-degree of s is the (unique) natural number n such that $\delta^n(s)$ is defined, but $\delta^{n+1}(s)$ is not (note that, by convention, $\delta^0(s) = s$ and, since the transitive closure δ^+ of δ is irreflexive, there always exists a natural number n such that $\delta^{n+1}(s)$ is not defined). For each *unlabeled* state $s \in S_\varepsilon$, the γ-*degree* of s is the least natural number n such that $\big(\gamma(s), s\big) \in \delta^n$.

The above definitions can be used to represent the nested structure of the transition functions of $\mathcal{A}$. Precisely, we define the binary relation $\Gamma_\mathcal{A} \subseteq S \times S$, called *nesting relation* of $\mathcal{A}$, such that, for every pair of states r, s of $\mathcal{A}$,

$$(r, s) \in \Gamma_\mathcal{A} \qquad \text{iff} \qquad \begin{cases} s \in S_\varepsilon \\ r = \delta^i\big(\gamma(r)\big) \quad \text{for some } i \text{ less than the } \gamma\text{-degree of } s. \end{cases}$$

Note that the transitive closure $\Gamma_{\mathcal{A}}^+$ of the nesting relation of $\mathcal{A}$ is antisymmetric, namely, it never happens that both $(r,s) \in \Gamma_{\mathcal{A}}^+$ and $(s,r) \in \Gamma_{\mathcal{A}}^+$ hold. This shows that $\Gamma_{\mathcal{A}}^+$ can be given the status of a well-founded partial order over the set of states of $\mathcal{A}$. Such a partial order immediately suggests an induction principle, called *γ-induction*, which can be used for both definitions and proofs.

As an example, let us consider the NCSSA $\mathcal{A}$ of Figure 2.7. We have that the δ-degree of s_0 (resp., s_1, s_2, s_3) is 2 (resp., 2, 1, 0), the γ-degree of s_2 (resp., s_3) is 1 (resp., 2), the nesting relation $\Gamma_{\mathcal{A}}$ is the set $\{(s_1,s_2),(s_0,s_3),(s_2,s_3)\}$, and its transitive closure $\Gamma_{\mathcal{A}}^+$ consists of all pairs in $\Gamma_{\mathcal{A}}$ plus the pair (s_1,s_3).

Finally, by exploiting γ-induction, we define, for every state $s \in S$ (resp., for every unlabeled state $s \in S_\varepsilon$), the word $u_s^{\mathcal{A}}$ (resp., the word $v_s^{\mathcal{A}}$):

$$u_s^{\mathcal{A}} = \begin{cases} \Omega(s) & \text{if } s \in S_A, \\ \left(v_s^{\mathcal{A}}\right)^{c_0(s)} & \text{if } s \in S_\varepsilon, \end{cases} \tag{2.1a}$$

$$v_s^{\mathcal{A}} = u_{\gamma(s)}^{\mathcal{A}}\ u_{\delta(\gamma(s))}^{\mathcal{A}}\ u_{\delta^{m-1}(\gamma(s))}^{\mathcal{A}} \qquad \text{where } m \text{ is the } \gamma\text{-degree of } s \tag{2.1b}$$

(here, for the sake of simplicity, we assume that $w_1\,w_2 = w_1$ and $w_1^\omega = w_1$ whenever w_1 is an infinite word).

The following proposition shows that the words recognized by NCSSA can be written as expressions that feature nested repetitions (e.g., $(■^4◀□^2)^\omega$ and $■^6((■^2□)^2□^2)^\omega$).

Proposition 3. *The word recognized by an NCSSA $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$ is of the form $u_{s_0}^{\mathcal{A}}\ u_{\delta(s_0)}^{\mathcal{A}} \dots u_{\delta^n(s_0)}^{\mathcal{A}}$ where n is the δ-degree of the initial state s_0.*

As an example, consider the NCSSA depicted in Figure 2.7. According to Proposition 3, such an automaton recognizes the word

$$\begin{aligned} w &= (\ u_{s_0}^{\mathcal{A}}\ )\ \ (\ u_{s_1}^{\mathcal{A}}\ )\ \ (\ \ u_{s_2}^{\mathcal{A}}\ \ ) \\ &= (\ ◀\ )\ \ \left(v_{s_1}^{\mathcal{A}}\right)^6\ \ (\ v_{s_2}^{\mathcal{A}}\ )^\omega \\ &= (\ ◀\ )\ \ \left(u_{s_2}^{\mathcal{A}}\right)^6\ \ (\ u_{s_0}^{\mathcal{A}}\,u_{s_1}^{\mathcal{A}}\ )^\omega \\ &= (\ ◀\ )\ \ (\ □\ )^6\ \ (\ ◀□^6\ )^\omega \end{aligned}$$

In order to prove Proposition 3, we introduce a technical lemma. Below, for the sake of brevity, we write $(s,c) \xrightarrow{w} (s',c')$ to denote the existence of a partial run of an NCSSA $\mathcal{A}$ that starts from the configuration (s,c), reaches the configuration (s',c'), and recognizes the finite word w. Similarly, we write $(s,c) \xrightarrow{w}$ whenever $\mathcal{A}$ recognizes the infinite word w starting from the configuration (s,c).

Lemma 2. *Let $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$ be an NCSSA, s a unlabeled state, and $c : S_\varepsilon \to \mathbb{N} \cup \{\omega\}$ a valuation such that $c(s) > 0$ and $c(r) = c_0(r)$ for every unlabeled state r such that $(r,s) \in \Gamma_{\mathcal{A}}^+$. We have that*

$$\big(s, c\big) \xrightarrow{v_s^{\mathcal{A}}} \big(s, c[c(s) - 1/s]\big).$$

Proof. We prove the lemma by exploiting γ-induction on the state s. For the sake of brevity, we denote by m the γ-degree of the state s, by r_i the state $\delta^i\big(\gamma(s)\big)$, by c' the valuation $c[c(s) - 1/s]$, and by $d_{i,j}$ the valuation $c'[j/r_i]$, for every $0 \leqslant i < m$ and every $0 \leqslant j \leqslant c(r_i)$. Now, let us consider a word of the form $u_{r_i}^{\mathcal{A}}$, with $0 \leqslant i < m$. If r_i is a labeled state, then $u_i^{\mathcal{A}} = \Omega(r_i)$ and hence

$$\big(r_i, c'\big) \xrightarrow{u_{r_i}^{\mathcal{A}}} \big(\delta(r_i), c'\big).$$

Otherwise, if r_i is a unlabeled state, then $(r_i, s) \in \Gamma_{\mathcal{A}}^+$. Moreover, $d_{i,j}(r_i) > 0$ and $d_{i,j}(q) = c'(q) = c(q) = c_0(q)$ for every $1 \leqslant j \leqslant c(r_i)$. Thus, by exploiting the inductive hypothesis on the state r_i, we obtain

$$\left.\begin{array}{c}
\big(r_i, d_{i,c(r_i)}\big) \xrightarrow{v_{r_i}^{\mathcal{A}}} \big(r_i, d_{i,c(r_i)-1}\big) \\
\big(r_i, d_{i,c(r_i)-1}\big) \xrightarrow{v_{r_i}^{\mathcal{A}}} \big(r_i, d_{i,c(r_i)-2}\big) \\
\dots \quad \dots \quad \dots \\
\big(r_i, d_{i,1}\big) \xrightarrow{v_{r_i}^{\mathcal{A}}} \big(r_i, d_{i,0}\big)
\end{array}\right\} c(r_i) \text{ times.}$$

In particular, since $u_{r_i}^{\mathcal{A}} = \big(v_{r_i}^{\mathcal{A}}\big)^{c_0(r_i)}$, $c_0(r_i) = c(r_i)$, and $c' = d_{i,c(r_i)}$, we have

$$\big(r_i, c'\big) \xrightarrow{u_{r_i}^{\mathcal{A}}} \big(\delta(r_i), c'\big).$$

We just proved that $\big(r_i, c'\big) \xrightarrow{u_{r_i}^{\mathcal{A}}} \big(\delta(r_i), c'\big)$ for every $0 \leqslant i < m$. To conclude the proof, it is sufficient to observe that $v_s^{\mathcal{A}} = u_{r_0}^{\mathcal{A}}\, u_{r_1}^{\mathcal{A}} \dots u_{r_{m-1}}^{\mathcal{A}}$ and, by hypothesis, $c'(q) = c(q)$ for every unlabeled state q such that $(q, s) \in \Gamma_{\mathcal{A}}^+$. Therefore, it follows that $\big(s, c\big) \xrightarrow{v_s^{\mathcal{A}}} \big(s, c'\big)$. □

Proof (Proof of Proposition 3). Let us denote by n the δ-degree of the initial state s_0 and let us consider a state s_i of the form $\delta^i(s_0)$, with $0 \leqslant i \leqslant n$. If s_i is a labeled state, then, by definition, $u_{s_i}^{\mathcal{A}} = \Omega(s_i)$. Otherwise, if s_i is an unlabeled state, then, by Lemma 2, we know that

$$\underbrace{\big(s_i, c_0\big) \xrightarrow{v_{s_i}^{\mathcal{A}}} \big(s_i, c_0[c_0(s_i) - 1/s_i]\big) \xrightarrow{v_{s_i}^{\mathcal{A}}} \dots \xrightarrow{v_{s_i}^{\mathcal{A}}} \big(s_i, c_0[0/s_i]\big).}_{c_0(s_i) \text{ times}}$$

Moreover, if $i < n$, then $\big(s_i, c_0[0/s_i]\big) \xrightarrow{\varepsilon} \big(s_{i+1}, c_0\big)$.

It is now clear that $\mathcal{A}$ recognizes the word $u_{s_0}^{\mathcal{A}}\, u_{\delta(s_0)}^{\mathcal{A}} \dots u_{\delta^n(s_0)}^{\mathcal{A}}$. □

2.3.2 Algorithms on NCSSA

In this section, we exploit the nested structure of NCSSA transition functions (precisely, Proposition 3), to devise efficient algorithms that operate on NCCSA. In particular, we will show that, under suitable conditions, granule conversion problems can be solved in polynomial time with respect to the size of NCSSA-based representations of time granularities. We will also provide a non-deterministic polynomial-time solution to the (non-)equivalence problem for NCSSA. Later, in Section 2.3.3, we shall describe in detail some optimization techniques of NCSSA-based representations of time granularities that improve the efficiency of the proposed algorithms.

Searching for Symbol Occurrences

We first address a basic problem, which arises very often when dealing with time granularities as well as with words in general, namely, the problem of finding the n-th occurrence of a given symbol in a (finite or infinite) word. Such a problem can be easily solved in linear time *with respect to the number of transitions* needed to reach the n-th occurrence of the symbol: it suffices to follow the transitions of the automaton until the n-th occurrence of the symbol is recognized. Fortunately, we can improve this straightforward solution by taking advantage of the structure of NCSSA. In the following, we first give an intuitive account of the improved algorithm and then describe it in detail. For the sake of brevity, hereafter we denote by $|w|_a$ the number of occurrences of the symbol a in the word w (as usual, we assume that $|w|_a = \omega$ if w contains infinitely many occurrences of a).

Let $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$ be an NCSSA that recognizes the word w and suppose that we have to find the position p of first occurrence of the symbol a in w. If the initial state s_0 belongs to S_A, then either $\Omega(s_0) = a$, thus implying $p = 1$, or $\Omega(s_0) \neq a$, thus implying $p > 1$. Otherwise, if the initial state s_0 belongs to S_ε, then we denote by m its γ-degree and we define $r_i = \delta^i(\gamma(s_0))$ for every $0 \leqslant i < m$. We then distinguish between the following two cases: either $|v^{\mathcal{A}}_{r_i}|_a > 0$ for some $0 \leqslant i < m$, or $|v^{\mathcal{A}}_{r_i}|_a = 0$ for every $0 \leqslant i < m$. In the former case, we know that the position p of the first occurrence of a in w is given by the expression $p_k + \sum_{0 \leqslant i < k} c_0(r_i)\,|v^{\mathcal{A}}_{r_i}|$, where k is the least index such that $|v^{\mathcal{A}}_{r_k}|_a > 0$ and p_k is the position of the first occurrence of a in $v^{\mathcal{A}}_{r_k}$. In the latter case, we know that $p > c_0(s_0)\big(\sum_{0 \leqslant i < m} c_0(r_i)\,|v_{s_i}|^{\mathcal{A}}\big)$.

By generalizing the above idea and by exploiting the principle of γ-induction, one can devise an efficient procedure (see Algorithm 2.2) that, given an NCSSA $\mathcal{A}$, a configuration (s, c), a set B of symbols, and a positive natural number n, returns the configuration of $\mathcal{A}$ that is reached after reading n occurrences of symbols in B, starting from the configuration (s, c). In addition, the described procedure can store, in a given array *counter*, the number of processed occurrences of each symbol of the alphabet (note that the position of the n-occurrence of symbols in B can be retrieved by examining the content of the array *counter*).

Algorithm 2.2. SEEKATOCCURRENCE($\mathcal{A}, s, c, B,$ *counter*)

```
let $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$
for each $a \in A$
  do  $counter(a) \leftarrow 0$
$i \leftarrow 0$
while $i < n$
  do
    if $s = \perp$
      then fail
    if $s \in S_A$
      then
        if $\Omega(s) \in B$
          then $i \leftarrow i + 1$
        $counter(\Omega(s)) \leftarrow counter(\Omega(s)) + 1$
        $s \leftarrow \delta(s)$
        comment: $s$ is set to $\perp$ if $\delta(s)$ is undefined
      else
        $j \leftarrow \sum_{a \in B} |v_s^{\mathcal{A}}|_a$
        $l \leftarrow |v_s^{\mathcal{A}}|$
        if $i + c(s)\, j < n$
          then
            if $c(s) = \omega$ or $|v_s^{\mathcal{A}}| = \omega$
              then fail
              else
                $q \leftarrow c(s)$
                $c(s) \leftarrow c_0(s)$
                $s \leftarrow \delta(s)$
          else
            $q \leftarrow \left\lfloor \frac{n-i-1}{j} \right\rfloor$
            $c(s) \leftarrow c(s) - q$
            $s \leftarrow \gamma(s)$
        $i \leftarrow i + q\, j$
        for each $a \in A$
          do  $counter(a) \leftarrow counter(a) + q\, l$
return $(s, C)$
```

In spite of the simplicity of the idea, the analysis of the complexity of the above algorithm is rather involved. To make it precise, we introduce a complexity measure $\|\mathcal{A}\|$, which takes into account the nested structure of the transition functions of an NCSSA $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$. Let s be a state of $\mathcal{A}$. We first define the value $\|\mathcal{A}\|_s$ by exploiting γ-induction on s:

$$\|\mathcal{A}\|_s = \begin{cases} 1 & \text{if } s \in S_A, \\ 1 + \max\limits_{0 \leqslant i < m} \Bigl(i + \|\mathcal{A}\|_{\delta^i(\gamma(s))}\Bigr) & \text{if } s \in S_\varepsilon \text{ and } m \text{ is the } \gamma\text{-degree of } s. \end{cases} \tag{2.2}$$

The *complexity* $\|\mathcal{A}\|$ of $\mathcal{A}$ is then defined as

$$\|\mathcal{A}\| = \max_{0 \leqslant i \leqslant n} \Bigl(i + \|\mathcal{A}\|_{\delta^i(s_0)}\Bigr) \tag{2.3}$$

where n is the δ-degree of the initial state s_0.

As an example, let us consider again the NCSSA $\mathcal{A}$ of Figure 2.7. We have that $\|\mathcal{A}\|_{s_0} = 1$, $\|\mathcal{A}\|_{s_1} = 1$, $\|\mathcal{A}\|_{s_2} = 1 + \|\mathcal{A}\|_{s_1} = 2$, $\|\mathcal{A}\|_{s_3} = 1 + \max(\|\mathcal{A}\|_{s_0}$, $1 + \|\mathcal{A}\|_{s_2}) = 4$, and, finally, $\|\mathcal{A}\| = \max(\|\mathcal{A}\|_{s_0}$, $1 + \|\mathcal{A}\|_{s_2}$, $2 + \|\mathcal{A}\|_{s_3}) = 6$.

A more interesting example is the following one. By exploiting the optimization algorithms which are described later in Section 2.3.3, one can obtain an NCSSA representing the granularity `Month` in terms of days having 87 states and complexity 14. Both these values are significantly less than the size of any equivalent granspec/SSA (see Section 2.2.3 for an explanation). The resulting NCSSA is succinctly described by the following expression:

$$\Biggl(\Biggl(\blacksquare^2(\blacksquare^{28}\blacktriangleleft)^2\Bigl(\blacksquare^{30}\bigl((\blacktriangleleft\blacksquare^{29})^2\blacksquare\bigr)^2\blacktriangleleft\Bigr)^2\Bigl(\blacksquare^3(\blacksquare^{27}\blacktriangleleft)^2\Bigl(\blacksquare^{30}\bigl((\blacktriangleleft\blacksquare^{29})^2\blacksquare\bigr)^2\blacktriangleleft\Bigr)^2\Bigr)^3\Biggr)^{25}$$

$$\Biggl(\Biggl(\blacksquare^3(\blacksquare^{27}\blacktriangleleft)^2\Bigl(\blacksquare^{30}\bigl((\blacktriangleleft\blacksquare^{29})^2\blacksquare\bigr)^2\blacktriangleleft\Bigr)^2\Biggr)^4$$

$$\Bigl(\blacksquare^2(\blacksquare^{28}\blacktriangleleft)^2\Bigl(\blacksquare^{30}\bigl((\blacktriangleleft\blacksquare^{29})^2\blacksquare\bigr)^2\blacktriangleleft\Bigr)^2\Bigl(\blacksquare^3(\blacksquare^{27}\blacktriangleleft)^2\Bigl(\blacksquare^{30}\bigl((\blacktriangleleft\blacksquare^{29})^2\blacksquare\bigr)^2\blacktriangleleft\Bigr)^2\Bigr)^3\Bigr)^{24}\Biggr)^3\Biggr)^{\omega}$$

Let $\mathcal{A}$ be an NCSSA and s a state of it. By exploiting Equation 2.2 and the principle of γ-induction, one can prove that the value $\|\mathcal{A}\|_s$ is less than or equal to the number of pairs $(q, r) \in \Gamma^+_{\mathcal{A}}$ such that either $r = s$ or $(r, s) \in \Gamma^+_{\mathcal{A}}$. It is thus clear that the complexity $\|\mathcal{A}\|$ of $\mathcal{A}$ is at most quadratic in the number of states of $\mathcal{A}$ and hence we can write $\|\mathcal{A}\| = \mathcal{O}(|\mathcal{A}|^2)$.

Moreover, under the assumption that the values $|\nu^{\mathcal{A}}_s|_a$, for every unlabeled state s and every symbol a, are pre-computed and stored into appropriate data structures for $\mathcal{A}$, it is possible to show that the worst-case time complexity of Algorithm 2.2 is $\Theta(\|\mathcal{A}\|)$ (and hence it is at most quadratic in the size $|\mathcal{A}|$).

It turns out that the running time of several algorithms operating on NCSSA run in linear time with respect to the complexities of the involved automata. This is the case, for instance, with simple procedures based on Algorithm 2.2 that search for occurrences of distinguished symbols in the word recognized by a given NCSSA $\mathcal{A}$. As an example, the following procedure, receives as input an NCSSA $\mathcal{A}$, a set B of symbols, a natural number n, and two positions $i, j \in \mathbb{N} \cup \{\omega\}$ and, after $\mathcal{O}(\|\mathcal{A}\|)$ time, it returns as output the position of the n-th occurrence of a symbol in B in the substring $w[i, j[$, where w is the word recognized by $\mathcal{A}$ (if $n = 0$, then the procedure returns the position i).

Algorithm 2.3. FindOccurrence($\mathcal{A}, B, n, i, j$)

```
let $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$
$(s, c) \leftarrow (s_0, c_0)$
SeekAtOccurrence($\mathcal{A}, s, c, A, i,$ counter)
SeekAtOccurrence($\mathcal{A}, s, c, B, n,$ counter)
if failed
  then return $j$
$k \leftarrow i + \sum_{a \in A}$ counter$(a)$
return $\min(j, k)$
```

Other procedures like GetSymbol ($\mathcal{A}, i$) and CountOccurrences ($\mathcal{A}, B, i, j$) allow one to compute, in time $\mathcal{O}(\|\mathcal{A}\|)$, the symbol $w(i)$ and the number of occurrences of symbols in B in the substring $w[i, j[$, where w is the word recognized by a given NCSSA $\mathcal{A}$. The structure of these procedures is quite similar to that of Algorithm 2.3 and thus omitted. Finally, it is worth pointing out that, in general, the complexity of these algorithms is *sub-linear* with respect to the number of transitions needed to reach the addressed symbol occurrence. This shows that algorithms working on NCSSA outperform those running on equivalent granspecs/SSA.

Granule Conversion Problems

Here, we provide efficient solutions to the granule conversion problem. The importance of such a problem has been highlighted by several authors (see, for instance, [3, 34, 102]). Nevertheless, in many approaches it has been only partially worked out in a rather intricate and non-uniform way.

In its most common formulation, the *granule conversion* problem is viewed as the problem of determining the set of granules of a granularity H which are in some specific relation with a given set of granules of a coarser/finer granularity G. According to such a definition, the granule conversion problem is actually a family of problems, whose different concrete instances are obtained by specifying the relation that must hold between the granules of the source granularity G and the destination granularity H.

In the following, we consider granule conversion problems for the relations 'intersect', 'cover', and 'covered-by' (other standard relations can be dealt with in a similar way [3]). Let G and H be two time granularities and let G' (resp., H') be a set of granules of G (resp., H).

- **Intersect relation.** The pair $(\mathsf{G}', \mathsf{H}')$ is an instance of the *intersect* relation if
$$\mathsf{H}' = \big\{\mathsf{h} \in \mathsf{H} : \exists\, \mathsf{g} \in \mathsf{G}'.\ \mathsf{g} \cap \mathsf{h} \neq \emptyset\big\}$$
namely, H' contains all and only the granules of H that intersect at least one granule in G'. Note that the intersect relation can be viewed as a *total* function, which maps sets of granules of G to sets of granules of H.
- **Cover relation.** The pair $(\mathsf{G}', \mathsf{H}')$ is an instance of the *cover* relation if H' is the *smallest* set such that
$$\bigcup_{\mathsf{h} \in \mathsf{H}'} \mathsf{h} \supseteq \bigcup_{\mathsf{g} \in \mathsf{G}'} \mathsf{g}$$
namely, H' is the smallest set of granules of H whose union contains the union of all granules in G'. Note that the cover relation can be viewed as *partial* function, which maps sets of granules of G to sets of granules of H. Moreover, every instance of the cover relation is also an instance of the intersect relation.
- **Covered-by relation.** The pair $(\mathsf{G}', \mathsf{H}')$ is an instance of the *covered-by* relation if H' is the *largest* set such that
$$\bigcup_{\mathsf{h} \in \mathsf{H}'} \mathsf{h} \subseteq \bigcup_{\mathsf{g} \in \mathsf{G}'} \mathsf{g}$$
namely, H' is the largest set of granules of H whose union is contained in the union of all granules in G'. The covered-by relation can be viewed as a *total* function, which maps sets of granules of G to (possibly empty) sets of granules of H.

For any possible instance of the granule conversion problem, we distinguish two variants of increasing complexity. In the simplest case (case 1), the time granularities involved have no gaps within or between granules; in the second case (case 2), gaps may occur both within and between the granules of the time granularities involved. Efficient automaton-based solutions for most common relations can be obtained in the first case, provided that we work with intervals. Indeed, if we restrict to case 1, the set of granules of the destination time granularity H that correspond to a given interval of granules of G can always be represented by an interval and hence it can be dealt with as a whole. On the contrary, if case 2 holds, we cannot guarantee that the resulting set of granules is an interval and thus we must consider one granule at a time.

The solutions to the granule conversion problems take advantage of some auxiliary functions, called downward/upward conversion functions, which are quite similar to the conversion operators introduced by Snodgrass et al. in

[34, 102]. The *downward conversion* function maps a time granularity G and a set (an interval, if we restrict to case 1) X of indices of granules of G to the set (respectively, the interval) $\mathsf{T} = \bigcup_{x \in \mathsf{X}} \mathsf{G}(x)$ of time points. Upward conversion is the dual operation and it comes in three different variants:

- the *upward intersect conversion* function maps a time granularity G and a non-empty set (or interval) T of time points to the possibly empty set (or interval) X of all indices x of granules of G such that $\mathsf{G}(x) \cap \mathsf{T} \neq \emptyset$;
- the *upward cover conversion* function maps a time granularity G and a non-empty set (or interval) T of time points to the smallest set (or interval) X of indices of granules of G such that $\bigcup_{x \in \mathsf{X}} \mathsf{G}(x) \supseteq \mathsf{T}$ (if the granularity G does not cover every time point in T, then the upward cover conversion function is undefined on G and T);
- the *upward covered-by conversion* function maps a time granularity G and a non-empty set (or interval) T of time points to the largest set (or interval) X of indices of granules of G such that $\bigcup_{x \in \mathsf{X}} \mathsf{G}(x) \subseteq \mathsf{T}$.

We shall implement downward and upward conversion functions separately for case 1 (no gaps allowed) and case 2 (gaps allowed). In case 1, since the set X is assumed to be an interval of granules, we use $\min(\mathsf{X})$ and $\max(\mathsf{X})$ to denote the least and the greatest element of X (we assume that $\max(\mathsf{X}) = \omega$ if X is not bounded). As previously mentioned, since intervals can be dealt with as a whole, the procedures that computer downward and upward conversion functions in case 1 use only a finite number of calls to Algorithm 2.2 and hence they require time $\mathcal{O}(\|\mathcal{A}\|)$, where $\mathcal{A}$ is the NCSSA representing the involved timed granularity. On the contrary, the procedures that compute downward and upward conversion functions in case 2 are necessarily less efficient.

Here are the algorithms for downward conversion in case 1 and case 2; those for upward conversions will be given later.

Algorithm 2.4. DOWNWARDCONVERSION-CASE1($\mathcal{A}$, X)

```
$t_1 \leftarrow$ FINDOCCURRENCE($\mathcal{A}, \{◀\}, \min(X), 1, \omega$)
$t_2 \leftarrow$ FINDOCCURRENCE($\mathcal{A}, \{◀\}, \max(X)+1, 1, \omega$)
return $[t_1, t_2[$
```

Algorithm 2.5. DOWNWARDCONVERSION-CASE2($\mathcal{A}$, X)

```
$[t_1, t_2[ \leftarrow$ DOWNWARDCONVERSION-CASE1($\mathcal{A}$, X)          (2.5a)
$T \leftarrow \emptyset$
for each $t \in [t_1, t_2[$                                          (2.5b)
    do { if GETSYMBOL($\mathcal{A}, t+1$) $\in \{■, ◀\}$
       {   then $T \leftarrow T \cup \{t\}$
return T
```

It is worth noticing that termination of Algorithm 2.5 is not guaranteed if X represents an infinite set of indices or if the last granule $\mathsf{G}(\max(\mathsf{X}))$ is infinite. Indeed, in both cases, the instruction 2.5a assigns the value ω to the variable t_2, thus entering an infinite loop in 2.5b. In order to guarantee termination, one can exploit the fact that $\mathcal{A}$ recognizes an ultimately periodic word w. More precisely, by reasoning on the initial and repeating patterns of w, one can detect whether a non-terminating loop has been reached and, accordingly, return the (possibly infinite) set T of converted time points as a linear progression of the form $\mathsf{X} \cup \{m + n\,q : m \in \mathsf{Y}, n \in \mathbb{N}\}$, where X, Y are finite disjoint sets of natural numbers and q is a positive natural number. A similar idea can be applied if X is an infinite set of granules represented by a linear progression. For the sake of simplicity, from now on, we shall not consider cases where infinite sets of granules or infinite sets of time points are involved (the reader should keep in mind that it is always possible to deal with such cases in an effective way).

Below, we implement upward conversion functions in case 1 (note that, in case 1, the upward *intersect* conversion function and the upward *cover* conversion function coincide).

Algorithm 2.6. UpwardIntersectConversion-Case1($\mathcal{A}$, T)
UpwardCoverConversion-Case1($\mathcal{A}$, T)

```
$x_1 \leftarrow$ CountOccurrences($\mathcal{A}, \{◀\}, 1, \min(\mathsf{T})$)
$x_2 \leftarrow$ CountOccurrences($\mathcal{A}, \{◀\}, 1, \max(\mathsf{T})$)
return $[x_1, x_2]$
```

Algorithm 2.7. UpwardCoveredByConversion-Case1($\mathcal{A}$, T)

```
if $\min(\mathsf{T}) > 0$
  then $x_1 \leftarrow$ CountOccurrences($\mathcal{A}, \{◀\}, 1, \min(\mathsf{T}) - 1$) $+ 1$
  else $x_1 \leftarrow 0$
if $\max(\mathsf{T}) < \omega$
  then $x_2 \leftarrow$ CountOccurrences($\mathcal{A}, \{◀\}, 1, \max(\mathsf{T}) + 1$) $- 1$
  else $x_2 \leftarrow \omega$
return $[x_1, x_2]$
```

The algorithms for case 2 are clearly more general, but less efficient, since they need to process one element of the input set T at a time. Note that the upward covered-by conversion function is computed by first collecting all indices of granules of G that intersect the time points in T (instruction 2.10a) and then discarding those granules which are not entirely covered by T (loop 2.10b).

Algorithm 2.8. UpwardIntersectConversion-Case2($\mathcal{A}, \mathsf{T}$)

```
$X \leftarrow \emptyset$
for each $t \in T$
    do { if GetSymbol($\mathcal{A}$, $t+1$) $\in \{\blacksquare, \blacktriangleleft\}$
           then { $x \leftarrow$ CountOccurrences($\mathcal{A}$, $\{\blacktriangleleft\}$, $1$, $t$)
                  $X \leftarrow X \cup \{x\}$
return $X$
```

Algorithm 2.9. UpwardCoverConversion-Case2($\mathcal{A}, \mathsf{T}$)

```
$X \leftarrow \emptyset$
for each $t \in T$
    do { if GetSymbol($\mathcal{A}$, $t+1$) $\in \{\blacksquare, \blacktriangleleft\}$
           then { $x \leftarrow$ CountOccurrences($\mathcal{A}$, $\{\blacktriangleleft\}$, $1$, $t$)
                  $X \leftarrow X \cup \{x\}$
           else fail
return $X$
```

Algorithm 2.10. UpwardCoveredByConversion-Case2($\mathcal{A}, \mathsf{T}$)

```
$X \leftarrow$ UpwardIntersectionConversion-Case2($\mathcal{A}$, $T$)   (2.10a)
for each $x \in X$                                                   (2.10b)
    do { $T_x \leftarrow$ DownwardConversion-Case2($\mathcal{A}$, $\{x\}$)
         if $T_x \not\subseteq T$
           then $X \leftarrow X \setminus \{x\}$
return $X$
```

The downward and upward conversion functions are strictly related to the intersect/cover/covered-by relations described at the beginning of this section. Precisely, given two time granularities G and H and two sets of granules $\mathsf{G}' \subseteq \mathsf{G}$ and $\mathsf{H}' \subseteq \mathsf{H}$, we have that $(\mathsf{G}', \mathsf{H}')$ is an instance of the intersect (resp., cover, covered-by) relation if and only if the indices of the granules of H' can be obtained by first applying a downward conversion of the indices of the granules of G', which results in a set T of time points, and then applying an upward intersect (resp., cover, covered-by) conversion of T.

The above arguments show that granule conversions between time granularities can be performed by exploiting basic algorithms on NCSSA. Moreover, if the involved granularities contain no gaps within or between granules, these conversions can be efficiently computed in linear time (resp., in polynomial time) with respect to the complexity (resp., the size) of the input NCSSA.

The Equivalence Problem

We now focus our attention on the equivalence problem for NCSSA, namely, the problem of deciding whether two given NCSSA recognize the same (finite or infinite) word. As a matter of fact, solving such a problem makes it possible to check whether two given NCCSA represent the same time granularity and hence to choose the most compact, or the most suitable, representation.

As a preliminary remark, recall that, in Section 2.2.4, we proved that the equivalence problem for the class of Reducible Counter Single-string Automata (RCSSA) is PSPACE-complete. Here, we show that the (non-)equivalence problem for the class of NCSSA can be solved in *non-deterministic polynomial time*. Intuitively, such a result follows from the fact that, given two NCSSA $\mathcal{A}_1$ and $\mathcal{A}_2$ recognizing the words w_1 and w_2, respectively, it is possible to guess a (sufficiently small) position i and then check in polynomial time whether $w_1(i) \neq w_2(i)$. Such a procedure suggests that the equivalence problem for NCSSA is in co-NP. In the following, we give a more formal argument for such a result, but we do not provide any proof for the co-NP-hardness. In fact, we conjecture that the equivalence problem for NCSSA may be solved by a deterministic algorithm which takes time polynomial in the size of the input NCSSA (unfortunately, at the moment, we are not able to provide such an algorithm).

Hereafter, we restrict our attention to the subclass of NCSSA that recognize infinite (ultimately periodic) words (the results provided below can be easily generalized to the whole class of NCSSA). We start with the following basic lemma.

Lemma 3. *Given two ultimately periodic words* $w_1 = u_1 v_1^{\omega}$ *and* $w_2 = u_2 v_2^{\omega}$*, with* $u_1, u_2 \in A^*$ *and* $v_1, v_2 \in A^+$*, we have that*

$$w_1 = w_2 \quad \textit{iff} \quad \forall\, 1 \leqslant i \leqslant \max\big(|u_1|, |u_2|\big) + \mathrm{lcm}\big(|v_1|, |v_2|\big).\ \ w_1(i) = w_2(i).$$

Proof. The left to right implication is trivial. As for the converse implication, let $p = \max\big(|u_1|, |u_2|\big)$ and $q = \mathrm{lcm}\big(|v_1|, |v_2|\big)$ and suppose that $w_1(i) = w_2(i)$ holds for every $1 \leqslant i \leqslant p + q$. For every $i > p + q$, we shortly denote by $[i]_{p,q}$ the value $\big((i - p - 1) \bmod q\big) + p + 1$. Since $[i]_{p,q} \leqslant p + q$ holds for every $i > p + q$ and since the ultimately periodic words w_1 and w_2 have initial patterns of length p and repeating patterns of length q, we have that, for every $i > p + q$,

$$w_1(i) \ = \ w_1\big([i]_{p,q}\big) \ = \ w_2\big([i]_{p,q}\big) \ = \ w_2(i).$$

This proves that $w_1 = w_2$. □

Now, recall that any NCSSA $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$ that recognizes an infinite word w can be viewed as a special form of RCSSA. Using an argument similar to that of Proposition 2, one can prove that the infinite word w recognized by $\mathcal{A}$ is ultimately periodic and it features an initial pattern u and a repeating pattern v of length less than or equal to $N_{\mathcal{A}} = |S| \prod_{s \in S_\varepsilon} n_s$,

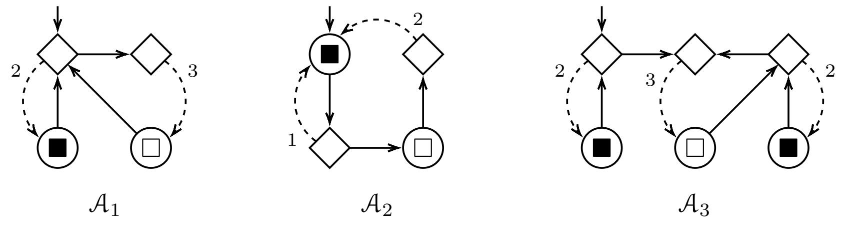


Fig. 2.8. State-optimal and complexity optimal NCSSA

where, for every $s \in S_\varepsilon$, n_s is either $c_0(s)$ or 1, depending on whether $c_0(s) < \omega$ or $c_0(s) = \omega$. Moreover, note that the value $N_\mathcal{A}$ is at most exponential in the size of $\mathcal{A}$. Therefore, by Lemma 3, one can decide whether two given NCCSA $\mathcal{A}_1$ and $\mathcal{A}_2$ recognize different infinite (ultimately periodic) words w_1 and w_2 by first guessing a position $1 \leqslant i \leqslant N_{\mathcal{A}_1} N_{\mathcal{A}_2}$, with $N_{\mathcal{A}_1} = \mathcal{O}\big(|\mathcal{A}_1|\, 2^{|\mathcal{A}_1|}\big)$ and $N_{\mathcal{A}_2} = \mathcal{O}\big(|\mathcal{A}_2|\, 2^{|\mathcal{A}_2|}\big)$, and then checking whether $w_1(i) \neq w_2(i)$ holds. Note that guessing the position i can be done in non-deterministic polynomial time $\mathcal{O}\big(|\mathcal{A}_1| \log |\mathcal{A}_1| + |\mathcal{A}_2| \log |\mathcal{A}_2|\big)$, while checking whether $w_1(i) \neq w_2(i)$ can be can be done in deterministic polynomial time $\mathcal{O}\big(|\mathcal{A}_1|^2 + |\mathcal{A}_2|^2\big)$ (e.g., by using the procedure GetSymbol described at the beginning of this section).

The above arguments prove that the equivalence problem for NCSSA is in co-NP. It is an open problem to establish whether there exists a deterministic polynomial-time algorithm that solves the equivalence problem for NCSSA.

2.3.3 Optimizing Representations

In Section 2.3.2 we outlined some basic algorithms that operate on NCSSA and that run in time linear in their complexity. In view of this operational flavor of NCSSA, the problem of reducing as much as possible the complexity $\|\mathcal{A}\|$ of a given NCSSA $\mathcal{A}$ becomes crucial. Furthermore, there is a widespread recognition of the fact that state minimization is an important problem both in classical automata theory [50] and in software and hardware design applications [101]. Therefore, another goal of practical interest is the minimization of the number of states of a given NCSSA $\mathcal{A}$.

The former problem is called *complexity optimization problem*, while the latter is called *state optimization problem*. Even though the complexity of an NCSSA and the number of its states are clearly related one to the other (e.g., the former is at most quadratic in the latter), complexity optimal and state optimal automata may look quite different.

As an example, the three NCSSA $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ of Figure 2.8 recognize the same finite word ■■□■■□■■□. The NCSSA $\mathcal{A}_1$ and $\mathcal{A}_2$ are state optimal ($n(\mathcal{A}_1) = n(\mathcal{A}_2) = 4$, while $n(\mathcal{A}_3) = 6$), and the NCSSA $\mathcal{A}_1$ and $\mathcal{A}_3$ are complexity optimal ($\|\mathcal{A}_1\| = \|\mathcal{A}_3\| = 5$, while $\|\mathcal{A}_2\| = 7$).

Moreover, the state optimization problem seems to be harder than the complexity optimization problem and only a partial solution to it will be given here. It is also worth remarking that optimal NCSSA are not guaranteed to be unique (up to isomorphism) among all equivalent NCSSA, as it happens, for instance, with the case of deterministic finite automata. Hereafter, we denote by $n(\mathcal{A})$ the number of states of an NCCSA $\mathcal{A}$.

Automata optimization problems can be solved in many different ways, e.g., by partitioning the state space or by exploiting characterizations of recognizable words in terms of suitable expressions. In the following, we tackle both the complexity optimization problem and the state optimization problem by using dynamic programming, namely, by computing optimal NCSSA starting from smaller (optimal) ones in a bottom-up fashion. The key ingredient of such an approach is the proof that the considered optimization problem exhibits an *optimal-substructure* property. In the following, we describe three basic operations on NCSSA and we prove closure properties for them. We then compare the complexity and the number of states of compound automata with that of their components. Finally, we take advantage of these compositional properties to prove optimal-substructure properties for the two optimization problems.

Compositional Properties

Hereafter, for the sake of simplicity, we denote by $\mathcal{A}_\varepsilon$ the *empty* NCSSA, which recognizes the empty word ε and has complexity 0. Similarly, for each symbol $a \in a$, we denote by $\mathcal{A}_a$ the *singleton* NCSSA, which consists of a single a-labeled state and has complexity 1.

The class of all NCSSA is easily proved to be closed under the operations of concatenation and repetition. Precisely, given a symbol a and an NCSSA $\mathcal{A}$ that recognizes a (finite or infinite) word w, we denote by $\textsc{AppendChar}(a, \mathcal{A})$ the concatenation of the singleton NCSSA $\mathcal{A}_a$ to $\mathcal{A}$, which results in an NCSSA that recognizes the word $a\,w$. Such an automaton can be obtained from $\mathcal{A}$ by (i) adding a new a-labeled state s_0, (ii) linking it to the initial state of $\mathcal{A}$, and (iii) marking s_0 as the new initial state (see Figure 2.9). If the argument $\mathcal{A}$ of $\textsc{AppendChar}$ is the empty automaton $\mathcal{A}_\varepsilon$, then $\textsc{AppendChar}(a, \mathcal{A}_\varepsilon)$ coincides with the singleton NCSSA $\mathcal{A}_a$.

In a similar way, we define concatenations involving repetitions of NCSSA. Precisely, given two NCSSA $\mathcal{A}$ and $\mathcal{B}$ that recognize, respectively, a finite word u and a (finite or infinite) word v and given $k \in \mathbb{N} \cup \{\omega\}$, we denote by $\textsc{AppendRepetition}(\mathcal{A}, k, \mathcal{B})$ the NCSSA that recognizes the word $u^k\,v$ and that is obtained from $\mathcal{A}$ and $\mathcal{B}$ by introducing (i) a new unlabeled state s_{loop}, which is represented by the diamond in Figure 2.10, (ii) a secondary transition from s_{loop} to the initial state of $\mathcal{A}$, (iii) a primary transition from the final state of $\mathcal{A}$ (i.e. the state that appears in the last position of the unique run of $\mathcal{A}$) to s_{loop}, (iv) a primary transition from s_{loop} to the initial state of $\mathcal{B}$ and by finally marking s_{loop} as the new initial state (see Figure 2.10). If the argument $\mathcal{B}$ of $\textsc{AppendRepetition}$ is the empty automaton $\mathcal{A}_\varepsilon$, then the resulting NCSSA $\textsc{AppendRepetition}(\mathcal{A}, k, \mathcal{A}_\varepsilon)$ recognizes the word u^k.

We can actually give $\textsc{AppendChar}$ and $\textsc{AppendRepetition}$ the status of linear-time algorithms that operate on NCSSA. Moreover, the complexity (resp., the number of states) of the resulting automata can be specified in

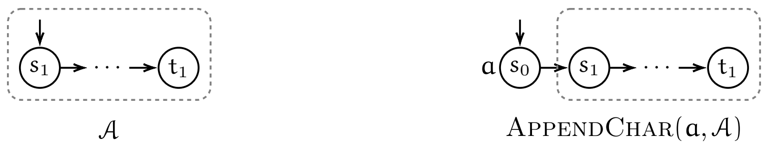


Fig. 2.9. The concatenation of a to $\mathcal{A}$

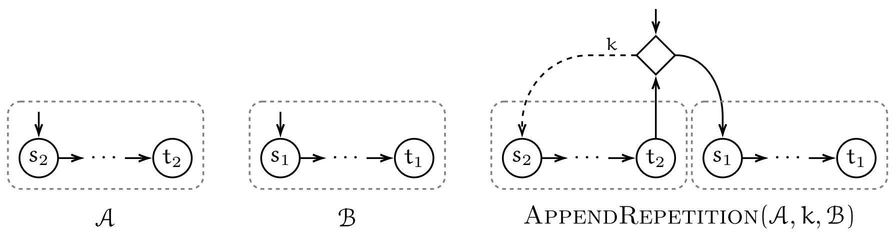


Fig. 2.10. The concatenation of a k-repetition of $\mathcal{A}$ to $\mathcal{B}$

terms of the complexity (resp., the number of states) of the component automata as follows:

- $\textsc{AppendChar}(a, \mathcal{A})$ has complexity $1 + \|\mathcal{A}\|$ and $1 + n(\mathcal{A})$ states;
- $\textsc{AppendRepetition}(\mathcal{A}, k, \mathcal{B})$ has complexity $1 + \max\big(\|\mathcal{A}\|, \|\mathcal{B}\|\big)$ and $1 + n(\mathcal{A}) + n(\mathcal{B})$ states.

We say that an NCSSA $\mathcal{A}$ is *decomposable* if it can be obtained from the empty NCSSA $\mathcal{A}_\varepsilon$ by applying a suitable sequence of operations AppendChar and AppendRepetition. Clearly, there exist NCSSA, including complexity optimal and state optimal ones (e.g., the automaton $\mathcal{A}_1$ of Figure 2.8), which are not decomposable. Below, we prove that for every NCSSA $\mathcal{A}$, there exists a decomposable NCSSA $\mathcal{A}'$ equivalent to $\mathcal{A}$ and having the same *complexity*.

In virtue of such a result, one can compute a complexity optimal NCSSA that recognizes a given (finite or ultimately periodic) word w by combining smaller (complexity optimal) NCSSA. Unfortunately, a similar property does not hold for state optimal NCSSA.

Lemma 4. *For every NCSSA $\mathcal{A}$ and every unlabeled state s of $\mathcal{A}$, there is a decomposable NCSSA $\mathcal{B}_s$ that recognizes the word $v_s^{\mathcal{A}}$ and such that $\|\mathcal{B}_s\| \leqslant \|\mathcal{A}\|_s - 1$.*

Proof. Let $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$ and $s \in S_\varepsilon$. We prove the claim by exploiting γ-induction. If s is a minimal element with respect to the partial order induced by $\Gamma_{\mathcal{A}}^+$, then we have $\gamma(s) = s$. In such a case, we define $\mathcal{B}_s$ as the empty NCSSA. Let us now consider the case of s being not a minimal element. Let m be the γ-degree of s and let $r_i = \delta^i(\gamma(s))$ for all $0 \leqslant i < m$. By inductive hypothesis, we know that, for every $0 \leqslant i < m$, if r_i is an unlabeled state, then there exists a decomposable (possibly empty) NCSSA $\mathcal{A}_{r_i}$ that recognizes the word $v_{r_i}^{\mathcal{A}}$ and such that $\|\mathcal{A}_{r_i}\| \leqslant \|\mathcal{A}\|_{r_i} - 1$. Now, we define

$\mathcal{B}_{s,0}$ as the empty NCSSA and, for every $1 \leqslant i \leqslant m$, we recursively define the NCSSA $\mathcal{B}_{s,i}$ as follows:

$$\mathcal{B}_{s,i} = \begin{cases} \textsc{AppendChar}\big(\Omega(r_{m-i}), \mathcal{B}_{s,i-1}\big) & \text{if } r_{m-i} \in S_A, \\ \textsc{AppendRepetition}\big(\mathcal{A}_{r_{m-i}}, c_0(r_{m-i}), \mathcal{B}_{s,i-1}\big) & \text{if } r_{m-i} \in S_\varepsilon. \end{cases}$$

It is easy to see that, for every $1 \leqslant i \leqslant m$, the NCSSA $\mathcal{B}_{s,i}$ recognizes the word $u^{\mathcal{A}}_{m-i}\, u^{\mathcal{A}}_{m-i+1} \dots u^{\mathcal{A}}_{m-1}$ and, furthermore, we have

$$\|\mathcal{B}_{s,i}\| \leqslant \max_{0 \leqslant j < i} \Big(j + \|\mathcal{A}\|_{r_j} \Big).$$

This proves that the (decomposable) NCSSA $\mathcal{B}_s = \mathcal{B}_{s,m}$ recognizes the word $v^{\mathcal{A}}_s$ and it has complexity $\|\mathcal{B}_s\| = \|\mathcal{B}_{s,m}\| \leqslant \|\mathcal{A}\|_s - 1$. □

Proposition 4. *For every NCSSA $\mathcal{A}$, there is an equivalent decomposable NCSSA $\mathcal{B}$ such that $\|\mathcal{B}\| \leqslant \|\mathcal{A}\|$.*

Proof. Let $\mathcal{A} = (A, S_A, S_\varepsilon, \Omega, \delta, \gamma, s_0, c_0)$, let n be the δ-degree of the initial state s_0, and let $s_i = \delta^i(s_0)$ for every $1 \leqslant i \leqslant n$. By Lemma 4, for every $0 \leqslant i \leqslant n$, if s_i is an unlabeled state, then there exists a decomposable (possibly empty) NCSSA $\mathcal{A}_{s_i}$ that recognizes the word $v^{\mathcal{A}}_{s_i}$ and such that $\|\mathcal{A}_{s_i}\| \leqslant \|\mathcal{A}\|_{s_i} - 1$. We now denote by $\mathcal{B}_0$ the empty NCSSA and, for every $1 \leqslant i \leqslant n$, we recursively define the NCSSA $\mathcal{B}_i$ as follows:

$$\mathcal{B}_i = \begin{cases} \textsc{AppendChar}\big(\Omega(s_{n-i}), \mathcal{B}_{i-1}\big) & \text{if } s_{n-i} \in S_A, \\ \textsc{AppendRepetition}\big(\mathcal{A}_{s_{n-i}}, c_0(s_{n-i}), \mathcal{B}_{s,i-1}\big) & \text{if } s_{n-i} \in S_\varepsilon. \end{cases}$$

It is easy to see that, for every $1 \leqslant i \leqslant n$, the NCSSA $\mathcal{B}_i$ recognizes the word $u^{\mathcal{A}}_{n-i}\, u^{\mathcal{A}}_{n-i+1} \dots u^{\mathcal{A}}_{n-1}$ and, furthermore, we have

$$\|\mathcal{B}_i\| \leqslant \max_{0 \leqslant j < i} \Big(j + \|\mathcal{A}\|_{s_j} \Big).$$

This proves that the (decomposable) NCSSA $\mathcal{B} = \mathcal{B}_n$ is equivalent to $\mathcal{A}$ and it has complexity $\|\mathcal{B}\| = \|\mathcal{B}_n\| \leqslant \|\mathcal{A}\|$. □

As an example of application of Proposition 4, consider the NCSSA $\mathcal{A}_1$ and $\mathcal{A}_3$ of Figure 2.8. They have the same complexity ($\|\mathcal{A}_1\| = \|\mathcal{A}_2\| = 5$), but $\mathcal{A}_3$ is decomposable, while $\mathcal{A}_1$ is not. It is easy to see that $\mathcal{A}_3$ can be obtained from $\mathcal{A}_1$ by applying the construction described in the proof of Proposition 4.

For what concerns the complexity of algorithms running on NCSSA (see, for instance, the granule conversion procedures described in Section 2.3.2), Proposition 4 basically states that there is no disadvantage in restricting to the subclass of decomposable NCSSA.

Computing Complexity Optimal NCSSA

Here we exploit compositional properties of NCSSA to devise a polynomial-time solution for the complexity optimization problem. In virtue of Proposition 4, we have that for any (finite or ultimately periodic) word $w \in A^{*+\omega}$, there exists a decomposable complexity optimal NCSSA $\mathcal{A}$ that recognizes w. In fact, we shall prove that, for any word w, there exists one such $\mathcal{A}$ that is decomposable into *complexity optimal* NCSSA. As a preliminary result, we establish the following technical lemma.

Lemma 5. *Given an NCSSA $\mathcal{A}$ recognizing a finite (resp., an infinite) word w and given a natural number $0 \leqslant n \leqslant |w|$ (resp., $n \geqslant 0$), there is a decomposable NCSSA* $\textsc{Prefix}(\mathcal{A}, n)$ *that recognizes the prefix* $w[1, n]$ *of* w *and such that* $\|\textsc{Prefix}(\mathcal{A}, n)\| \leqslant \|\mathcal{A}\|$.

Proof. By Proposition 4, we know that there is a decomposable NCSSA $\mathcal{A}'$ which is equivalent to $\mathcal{A}$ and satisfies $\|\mathcal{A}'\| \leqslant \|\mathcal{A}\|$. We define the NCSSA $\textsc{Prefix}(\mathcal{A}, n)$ by exploiting induction on the decomposition structure of $\mathcal{A}'$. We only consider the non-trivial cases, namely, the cases of non-empty NCSSA.

1. Suppose that $\mathcal{A}' = \textsc{AppendChar}(a, \mathcal{B})$. We further distinguish between the following sub-cases:
   a. if $n = 0$, then we define $\textsc{Prefix}(\mathcal{A}, n)$ as the empty NCSSA $\mathcal{A}_\varepsilon$;
   b. if $n > 0$, then we define $\textsc{Prefix}(\mathcal{A}, n) = \textsc{AppendChar}(a, \mathcal{B}')$, where $\mathcal{B}' = \textsc{Prefix}(\mathcal{B}, n - 1)$ (note that this is well-defined by inductive hypothesis).
2. Suppose that $\mathcal{A}' = \textsc{AppendRepetition}(\mathcal{B}, k, \mathcal{C})$, with $k \in \mathbb{N}$. Let u and v be the words recognized by $\mathcal{B}$ and $\mathcal{C}$, respectively. We distinguish between the following sub-cases:
   a. if $n \leqslant |u|$, then we define $\textsc{Prefix}(\mathcal{A}, n) = \textsc{Prefix}(\mathcal{B}, n)$ (note that this is well-defined by inductive hypothesis);
   b. if $|u| < n \leqslant k\,|u|$, then we define $p = n \bmod |u|$, $q = \left\lceil \frac{n}{|u|} \right\rceil$, and $\textsc{Prefix}(\mathcal{A}, n) = \textsc{AppendRepetition}(\mathcal{B}, q, \mathcal{B}')$, where $\mathcal{B}' = \textsc{Prefix}$ $(\mathcal{B}, p)$ (note that this is well-defined by inductive hypothesis);
   c. if $n > k\,|u|$, then we define $p = n - k\,|u|$ and $\textsc{Prefix}(\mathcal{A}, n) = \textsc{AppendRepetition}(\mathcal{B}, k, \mathcal{C}')$, where $\mathcal{C}' = \textsc{Prefix}(\mathcal{C}, p)$ (note that this is well-defined by inductive hypothesis).

It is routine to verify that $\textsc{Prefix}(\mathcal{A}, n)$ is a decomposable NCSSA that recognizes the prefix $w[1, n]$ of w and satisfies $\|\textsc{Prefix}(\mathcal{A}, n)\| \leqslant \|\mathcal{A}\|$. □

The following proposition is the basic ingredient of the solution to the complexity optimization problem. It proves an optimal-substructure property for (decomposable) complexity optimal NCSSA recognizing finite words.

Proposition 5. *For every non-empty finite word w, at least one of the following conditions holds:*

1. *for every complexity optimal NCSSA $\mathcal{B}$ that recognizes the suffix $w[2, |w|]$ of w,* $\textsc{AppendChar}(w(1), \mathcal{B})$ *is a complexity optimal NCSSA that recognizes w;*

2. *there exists a position $1 \leqslant \mathtt{r} \leqslant |w|$ such that, for every complexity optimal NCSSA $\mathcal{B}$ that recognizes the prefix $w[1, \mathtt{q}]$ of w, where $\mathtt{q}$ is the minimum period of $w[1, \mathtt{r}]$, and for every complexity optimal NCSSA $\mathcal{C}$ that recognizes the suffix $w[\mathtt{r}+1, |w|]$ of w,* $\textsc{AppendRepetition}(\mathcal{B}, \mathtt{r}/\mathtt{q}, \mathcal{C})$ *is a complexity optimal NCSSA that recognizes w.*

Proof. By Proposition 4, we know that there is a decomposable complexity optimal NCSSA $\mathcal{A}'$ that recognizes w. We prove the claim by exploiting induction on the decomposition structure of $\mathcal{A}'$. We only consider the non-trivial cases.

1. Suppose that $\mathcal{A}' = \textsc{AppendChar}(w(1), \mathcal{B}')$. Let $\mathcal{B}$ be a complexity optimal NCSSA that recognizes the suffix $w[2, |w|]$ of w. Since $\mathcal{B}$ is a complexity optimal NCSSA equivalent to $\mathcal{B}'$, we have
$$\begin{aligned}\big\|\textsc{AppendChar}(w(1), \mathcal{B})\big\| &= 1 + \|\mathcal{B}\| \\ &\leqslant 1 + \|\mathcal{B}'\| \\ &= \big\|\textsc{AppendChar}(w(1), \mathcal{B}')\big\| \\ &= \big\|\mathcal{A}'\big\|.\end{aligned}$$
This shows that $\textsc{AppendChar}(w(1), \mathcal{B})$ is a complexity optimal NCSSA that recognizes w.
2. Suppose $\mathcal{A}' = \textsc{AppendRepetition}(\mathcal{B}', \mathtt{k}, \mathcal{C}')$. Let $\mathtt{x}$ be the finite word recognized by $\mathcal{B}'$, $\mathtt{q}$ the minimum period of $\mathtt{x}$, and $\mathtt{r} = \mathtt{k}\,|\mathtt{x}|$. Furthermore, let $\mathcal{B}$ be a complexity optimal NCSSA that recognizes the prefix $w[1, \mathtt{q}]$ of w and let $\mathcal{C}$ be a complexity optimal NCSSA that recognizes the suffix $w[\mathtt{r}+1, |w|]$ of w. Clearly, we have $\mathtt{q} \leqslant |\mathtt{x}|$. Since $w[1, \mathtt{q}]$ is a prefix of $\mathtt{x}$ and $\mathcal{B}$ is a complexity optimal NCSSA that recognizes $w[1, \mathtt{q}]$, by Lemma 5, we obtain $\|\mathcal{B}\| \leqslant \|\mathcal{B}'\|$. Moreover, since $\mathcal{C}$ is a complexity optimal NCSSA equivalent to $\mathcal{C}'$, we obtain $\|\mathcal{C}\| \leqslant \|\mathcal{C}'\|$. Therefore, we have
$$\begin{aligned}\big\|\textsc{AppendRepetition}(\mathcal{B}, \mathtt{r}/\mathtt{q}, \mathcal{C})\big\| &= 1 + \max\big(\|\mathcal{B}\|, \|\mathcal{C}\|\big) \\ &\leqslant 1 + \max\big(\|\mathcal{B}'\|, \|\mathcal{C}'\|\big) \\ &= \big\|\textsc{AppendRepetition}(\mathcal{B}', \mathtt{k}, \mathcal{C}')\big\| \\ &= \big\|\mathcal{A}'\big\|.\end{aligned}$$
This shows that $\textsc{AppendRepetition}(\mathcal{B}, \mathtt{r}/\mathtt{q}, \mathcal{C})$ is a complexity optimal NCSSA that recognizes w. □

Proposition 5 above implies that, for every finite word w, there only exist finitely many ways of combining complexity optimal NCSSA and obtaining a complexity optimal NCSSA recognizing w. This yields a polynomial-time procedure that, given a finite word w as input, computes a complexity optimal NCSSA that recognizes w. Algorithm 2.11 describes such a procedure by using the following data structures and auxiliary procedures:

- a matrix P, where each entry $\mathsf{P}(\mathtt{i}, \mathtt{j})$, with $1 \leqslant \mathtt{i} \leqslant \mathtt{j} \leqslant |w|$, stores the minimum period of the substring $w[\mathtt{i}, \mathtt{j}]$ of w (these values can be computed in time $\Theta(|w|^2)$ by exploiting the algorithms presented in Section 2.1.2);

- a matrix $\mathcal{A}$, where each entry $\mathcal{A}(i, j)$, with $1 \leqslant i \leqslant j \leqslant |w|$, stores the generated complexity optimal NCSSA recognizing the substring $w[i, j]$ of w; (for convenience, for every $1 \leqslant i \leqslant |w|$, the entry $\mathcal{A}(i+1, i)$ stores the empty NCSSA $\mathcal{A}_\varepsilon$);
- a procedure BestComplexity, which receives a tuple of NCSSA as input and returns the one having the minimum complexity as output.

Note that a straightforward implementation of the inner loop 2.11a requires time $\mathcal{O}(|\mathcal{B}_r|)$ to generate each intermediate NCSSA $\mathcal{B}_r$. Pairing such a complexity with the threefold nesting of the loops yields $\mathcal{O}(|w|^4)$ as an upper bound to the time required to execute the algorithm. As a matter of fact, it is possible to compare the complexities of the intermediate NCSSA without really building them. By using appropriate data structures and by computing in constant time only the complexities of the intermediate NCSSA, one obtains a more efficient implementation that runs in time $\Theta(|w|^3)$.

Algorithm 2.11. ComplexityOptimal-FiniteCase(w)

```
for each $1 \leqslant i \leqslant j \leqslant |w|$
  let  $P(i, j)$ = minimum period of $w[i, j]$
for each $1 \leqslant i \leqslant |w|$
  do   $\mathcal{A}(i+1, i) \leftarrow \mathcal{A}_\varepsilon$
for $n \leftarrow 1$ to $|w|$   and   for each $1 \leqslant i \leqslant |w| - n + 1$
  do   $j \leftarrow i + n - 1$
       $\mathcal{B}_0 \leftarrow$ AppendChar($w(i)$, $\mathcal{A}(i+1, j)$)
       for each $1 \leqslant r \leqslant n$                           (2.11a)
         do   $q \leftarrow P(i, i+r-1)$
              $\mathcal{B}_r \leftarrow$ AppendRepetition($\mathcal{A}(i, i+q-1)$, $r/q$, $\mathcal{A}(i+r, j)$)
       $\mathcal{A}(i, j) \leftarrow$ BestComplexity($\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_n$)
return $\mathcal{A}(1, n)$
```

Proving an optimal substructure property for NCCSA recognizing infinite (ultimately periodic) words is more problematic, because it may happen that a complexity optimal NCSSA encodes a non-minimal initial pattern. Consider, for instance, the ultimately periodic word $w = (abc)^2\ ab\ (ce)^\omega$. The length of the shortest initial pattern of w is 8 and the length of the shortest repeating pattern is 2; however, any complexity optimal NCSSA recognizing w encodes the initial pattern $(abc)^3$ (of length 9) and the repeating pattern ec.

Proposition 6. *For every ultimately periodic word w with shortest initial pattern u and shortest (primitive) repeating pattern v, at least one of the following conditions holds:*

1. *for every complexity optimal NCSSA $\mathcal{B}$ that recognizes the suffix $w[2, \omega[$ of w,* $\textsc{AppendChar}(w(1), \mathcal{B})$ *is a complexity optimal NCSSA that recognizes w;*
2. *u is the empty word and, for every complexity optimal NCSSA $\mathcal{B}$ recognizing the repeating pattern v,* $\textsc{AppendRepetition}(\mathcal{B}, \omega)$ *is a complexity optimal NCSSA that recognizes w;*
3. *there exists a position $1 \leqslant r < 2|u| + 2|v|$ such that, for every complexity optimal NCSSA $\mathcal{B}$ that recognizes the prefix $w[1, q]$ of w, where q is the minimum period of $w[1, r]$, and for every complexity optimal NCSSA $\mathcal{C}$ that recognizes the suffix $w[r+1, \omega[$ of w,* $\textsc{AppendRepetition}(\mathcal{B}, r/q, \mathcal{C})$ *is a complexity optimal NCSSA that recognizes w.*

Proof. By Proposition 4, we know that there is a decomposable complexity optimal NCSSA $\mathcal{A}'$ that recognizes w. We prove the claim by exploiting induction on the decomposition structure of $\mathcal{A}'$. We only consider the most complex case, that is, the case in which $\mathcal{A}' = \textsc{AppendRepetition}(\mathcal{B}', k, \mathcal{C}')$, with k being a positive natural number. Below, we distinguish between two sub-cases: $k = 1$ and $k > 1$.

1. Suppose that $k = 1$. Let x be the finite word recognized by $\mathcal{B}'$ and let r be either $|x|$ or $((|x| - |u|) \bmod |v|) + |u|$, depending on whether $|x| < |u|$ or $|x| \geqslant |u|$. Note that, by definition, $r \leqslant |x|$ and $r < |u| + |v|$. Now, let $\mathcal{B}$ be a complexity optimal NCSSA that recognizes the prefix $w[1, r]$ of w and let $\mathcal{C}$ be a complexity optimal NCSSA that recognizes the suffix $w[r + 1, \omega[$ of w. Since $w[1, r]$ is a prefix of x and $\mathcal{B}$ is a complexity optimal NCSSA recognizing $w[1, r]$, by Lemma 5, we obtain $\|\mathcal{B}\| \leqslant \|\mathcal{B}'\|$. Moreover, since $w[r + 1, \omega[= w[|x| + 1, \omega[$ and $\mathcal{C}$ is a complexity optimal NCSSA recognizing $w[r+1, \omega]$, we obtain $\|\mathcal{C}\| \leqslant \|\mathcal{C}'\|$. It thus follows that
$$\begin{aligned}\big\|\textsc{AppendRepetition}(\mathcal{B}, 1, \mathcal{C})\big\| &= 1 + \max\big(\|\mathcal{B}\|, \|\mathcal{C}\|\big)\\ &\leqslant 1 + \max\big(\|\mathcal{B}'\|, \|\mathcal{C}'\|\big)\\ &= \big\|\textsc{AppendRepetition}(\mathcal{B}', 1, \mathcal{C}')\big\|\\ &= \big\|\mathcal{A}'\big\|.\end{aligned}$$
This shows that the NCSSA $\textsc{AppendRepetition}(\mathcal{B}, 1, \mathcal{C})$, which recognizes the ultimately periodic word w, is complexity optimal.
2. Suppose that $k > 1$. We proceed in the usual way by letting x be the finite word recognized by $\mathcal{B}'$, q the minimum period of x, and $r = k\,|x|$. Then, given two complexity optimal NCSSA $\mathcal{B}$ and $\mathcal{C}$ recognizing, respectively, the prefix $w[1, q]$ and the suffix $w[r + 1, \omega[$ of w, we verify that
$$\begin{aligned}\big\|\textsc{AppendRepetition}(\mathcal{B}, r/q, \mathcal{C})\big\| &= 1 + \max\big(\|\mathcal{B}\|, \|\mathcal{C}\|\big)\\ &\leqslant 1 + \max\big(\|\mathcal{B}'\|, \|\mathcal{C}'\|\big)\\ &= \big\|\textsc{AppendRepetition}(\mathcal{B}', k, \mathcal{C}')\big\|\\ &= \big\|\mathcal{A}'\big\|.\end{aligned}$$
This shows that the NCSSA $\textsc{AppendRepetition}(\mathcal{B}, r/q, \mathcal{C})$, which recognizes the ultimately periodic word w, is complexity optimal.

It remains to show that $r < 2|u|+2|v|$. Suppose, by way of contradiction, that $r \geqslant 2|u| + 2|v|$. Let us consider the substring $y = w[|u| + 1, r]$ of w. Since y is a suffix of the word $\big(w[1, q]\big)^{r/q}$, we know that y has *partial period* q (see Section 2.1.2 for a definition of partial period). Similarly, since y is a prefix of v^{ω}, where v is the primitive repeating pattern of w, we know that y has partial period $|v|$. Moreover, since $k \geqslant 2$, we have $r \geqslant 2|x|$ and, by assumption, $r \geqslant 2|u| + 2|v|$. This implies $r \geqslant |u| + |v| + |x|$, whence $|x| \geqslant |v| + |x|$. Therefore, we can apply Lemma 1 and obtain that $p = \gcd\big(q, |v|\big)$ is a partial period of y as well. Let us consider now the substring $z = w[2|u| + 1,\ 2|u| + 2|v|]$ of y. It has length equal to $|v|$ and partial period p. By construction, p divides $|v|$ and hence p is an exact period of z. However, from the fact that v is *primitive*, we know that $p = |v|$. Similarly, p divides the minimum period q of x and hence $q = p$ $(= |v|)$. We thus conclude that x has minimum period $|v|$. Finally, since u is the shortest initial pattern of w and x is a prefix of w with period $|v|$, we obtain $|u| = 0$.

We just proved that $r \geqslant 2|u| + 2|v|$ implies that $w[r + 1, r + |v|]$ is a repeating pattern of x, from which we obtain

$$\begin{aligned} w &= w[1, r]\ w[r + 1, \omega[\\ &= x^k\ w[r + 1, r + |v|]^{\omega} \\ &= w[r + 1, r + |v|]^{\omega} \\ &= w[r + 1, \omega[. \end{aligned}$$

The above equality contradicts the hypothesis that $\mathcal{A}'$ is a complexity optimal NCSSA (the automaton $\mathcal{C}$ has smaller complexity and it recognizes $w[r + 1, \omega] = w$). Therefore, we must conclude that $r < 2|u| + 2|v|$. □

Proposition 6 extends the optimal sub-structure property to the case of infinite words. However, in order to devise an effective procedure that computes a complexity optimal NCSSA recognizing a given ultimately periodic word w, we must provide an upper bound to the number of possible applications of case 3 of Proposition 6.

Given a natural number n, we denote by $\mathscr{F}(n)$ the set of all decomposable NCSSA that recognize finite words of length exactly n. Moreover, for every natural number n and every pair of positive natural numbers r, q, we recursively define the set $\mathscr{U}(n, r, q)$ as follows:

$$\mathscr{U}(0, r, q) = \big\{\textsc{AppendRepetition}(\mathcal{A}, \omega, \mathcal{A}_{\varepsilon}) \,:\, \mathcal{A} \in \mathscr{F}(q)\big\}$$

$$\begin{aligned} \mathscr{U}(n+1, r, q) = \mathscr{U}(n, r, q) & \\ &\cup \big\{\textsc{AppendChar}(a, \mathcal{B}) \,:\, a \in A,\ \mathcal{B} \in \mathscr{U}(n, r, q)\big\} \\ &\cup \left\{ \textsc{AppendRepetition}(\mathcal{A}, k, \mathcal{B}) \,: \begin{array}{l} k, p \in \mathbb{N},\ kp \leqslant r, \\ \mathcal{A} \in \mathscr{F}(p), \\ \mathcal{B} \in \mathscr{U}(n, r, q) \end{array} \right\}. \end{aligned}$$

Note that $\mathscr{U}(0, r, q)$ contains all and only the decomposable NCSSA that recognize ultimately periodic words of the form v^ω, with $|v| = q$. For every $n > 0$, $\mathscr{U}(n, r, q)$ contains only, but not all, decomposable NCSSA that recognize ultimately periodic words of the form $u v^\omega$, with $|u| \leqslant n\,r$ and $|v| = q$.

Proposition 7. *For every ultimately periodic word w with shortest initial pattern u and shortest repeating pattern v, there is a complexity optimal NCSSA that recognizes w and belongs to the set $\mathscr{U}\big(|u|+|v|-1,\ 2|u|+2|v|-1,\ |v|\big)$.*

Proof. Let w be an ultimately periodic word and let u and v be, respectively, the shortest initial pattern and the shortest repeating pattern of w. By recursively applying Proposition 6, we end up with a finite sequence of symbols $a_1, ..., a_n$, two finite sequences of natural numbers $k_1, ..., k_n$ and $p_0, p_1, ..., p_n$, and two finite sequences of complexity optimal NCSSA $\mathcal{A}_0, \mathcal{A}_1, ..., \mathcal{A}_n$ and $\mathcal{B}_0$, $\mathcal{B}_1, ..., \mathcal{B}_n$ such that

- $p_0 = |v|$,
- for every $1 \leqslant i \leqslant n$, $k_i\, p_i < 2|u| + 2|v|$,
- for every $0 \leqslant i \leqslant n$, $\mathcal{A}_i \in \mathscr{F}(p_i)$,
- $\mathcal{B}_0 = \textsc{AppendRepetition}(\mathcal{A}_0, \omega, \mathcal{A}_\varepsilon)$,
- for every $1 \leqslant i \leqslant n$, $\mathcal{B}_i$ is either the NCSSA $\textsc{AppendChar}(a_i, \mathcal{B}_{i-1})$ or the NCSSA $\textsc{AppendRepetition}(\mathcal{A}_i, k_i, \mathcal{B}_{i-1})$,
- $\mathcal{B}_n$ recognizes the ultimately periodic word w.

It is easy to see that $\|\mathcal{B}_i\| > i+1$ for all $0 \leqslant i \leqslant n$. Thus, we have $n < |u|+|v|$ (otherwise, $\mathcal{B}_n$ would not be complexity optimal). This proves that $\mathcal{B}_n$ belongs to the set $\mathscr{U}\big(|u| + |v| - 1,\ 2|u| + 2|v| - 1,\ |v|\big)$. □

On the basis of the above results, we can devise a polynomial-time procedure (see Algorithm 2.12) that solves the complexity optimization problem for ultimately periodic words. Such a procedure receives as input an ultimately periodic word w, which is represented by a pair (u, v), with u and v being, respectively, the shortest initial pattern and the shortest repeating pattern of w, and it returns as output a decomposable complexity optimal NCSSA that recognizes w. Similarly to the case of finite words, Algorithm 2.12 runs in time $\Theta\big((|u| + |v|)^3\big)$ and uses the following data structures and auxiliary procedures:

- a matrix P, where each entry $P(i, j)$, with $1 \leqslant i \leqslant j \leqslant 3|u|+3|v|-2$, stores the minimum period of the substring $w[i, j]$ of w;
- a matrix $\mathcal{A}$, where each entry $\mathcal{A}(i, j)$, with $1 \leqslant i \leqslant j \leqslant 3|u| + 3|v| - 2$, stores the generated complexity optimal NCSSA recognizing the substring $w[i, j]$ of w; (these automata can computed in time $\Theta\big((|u|+|v|)^3\big)$ by using Algorithm 2.11);
- an array $\mathcal{B}$, where each entry $\mathcal{B}(i)$, with $1 \leqslant i \leqslant |u| + |v|$, stores the generated complexity optimal NCSSA recognizing the suffix $w[i, \omega[$ of w (for convenience, each entry $\mathcal{B}(i)$, with $1 \leqslant i \leqslant |u|$, is initialized with a dummy NCSSA $\mathcal{A}_\infty$ of sufficiently high complexity);

- a procedure Normalize, which receives as input the initial pattern u, the repeating pattern v, and a position i of the ultimately periodic word $w = uv^\omega$ and it returns as output either the value i or the value $\big((i - |u| - 1) \bmod |v|\big) + |u| + 1$, depending on whether $i \leqslant |u|$ or $i > |u|$ (note that in both cases we have $w(i) = w(\textsc{Normalize}(u, v, i))$);
- a procedure BestComplexity, which receives a tuple of NCSSA as input and returns the one having the minimum complexity as output.

Moreover, some heuristics can be introduced in order to significantly improve the performance of the optimization algorithm. As an example, one can exit the outer loop 2.12a on n if the array $\mathcal{B}$ has not changed within an iteration.

Algorithm 2.12. ComplexityOptimal-InfiniteCase(u, v)

```
$w \leftarrow (uv^\omega)\,[1, 3|u| + 3|v| - 2]$
for each $1 \leqslant i \leqslant j \leqslant 3|u| + 3|v| - 2$
  let  $P(i,j) =$ minimum period of $w[i,j]$
       $\mathcal{A}(i,j) =$ complexity optimal NCSSA recognizing $w[i,j]$
for each $1 \leqslant i \leqslant |u|$
  do  $\mathcal{B}(i) \leftarrow \mathcal{A}_\infty$
for each $|u| + 1 \leqslant i \leqslant |u| + |v|$
  do  $\mathcal{B}(i) \leftarrow \textsc{AppendRepetition}(\mathcal{A}(i, i + |v| - 1), \omega, \mathcal{A}_\varepsilon)$
for each $1 \leqslant n \leqslant |u| + |v| - 1$  and  $1 \leqslant i \leqslant |u| + |v|$   (2.12a)
  do  $i' \leftarrow \textsc{Normalize}(u, v, i + 1)$
      $\mathcal{C}_0 \leftarrow \textsc{AppendChar}(w(i), \mathcal{B}(i'))$
      for each $1 \leqslant r \leqslant 2|u| + 2|v| - 1$
        do  $q \leftarrow P(i, i + r - 1)$
            $i' \leftarrow \textsc{Normalize}(u, v, i + r)$
            $\mathcal{C}_r \leftarrow \textsc{AppendRepetition}\begin{pmatrix}\mathcal{A}(i, i + q - 1),\\ r/q,\\ \mathcal{B}(i')\end{pmatrix}$
      $\mathcal{B}(i) \leftarrow \textsc{BestComplexity}(\mathcal{B}(i), \mathcal{C}_0, \mathcal{C}_1, ..., \mathcal{C}_{2|u|+2|v|-1})$
return $\mathcal{B}(1)$
```

Computing State Optimal NCSSA

We now adapt the results we obtained for complexity optimal NCSSA to state optimal ones. Unfortunately, we do not have an analogous of Proposition 4 for state optimal NCSSA. This basically means that, given a (finite or ultimately periodic) word w, we are able to compute an NCSSA which is state optimal among all equivalent decomposable NCSSA, but it may not be state optimal among all equivalent (non-decomposable) NCSSA. From now on, we tacitly assume that state optimality is evaluated over the restricted subclass of decomposable NCSSA.

In the sequel, we proceed by analogy with the case of complexity optimal NCSSA (some complications arise since we must take into account also non-minimal periods of substrings).

Proposition 8. *For every non-empty finite word w, at least one of the following conditions holds:*

1. *for every state optimal NCSSA $\mathcal{B}$ that recognizes the suffix $w[2, |w|]$ of w,* $\textsc{AppendChar}(w(1), \mathcal{B})$ *is a state optimal NCSSA that recognizes w;*
2. *there exists a position $1 \leqslant r \leqslant |w|$ such that, for every state optimal NCSSA $\mathcal{B}$ that recognizes the prefix $w[1, q]$ of w, where q is a period of $w[1, r]$ (not necessarily the minimum one), and for every state optimal NCSSA $\mathcal{C}$ that recognizes the suffix $w[r+1, |w|]$ of w,* $\textsc{AppendRepetition}(\mathcal{B}, r/q, \mathcal{C})$ *is a state optimal NCSSA that recognizes w.*

Proof. Let $\mathcal{A}'$ be a (decomposable) state optimal NCSSA recognizing w. We prove the claim by exploiting induction on the decomposition structure of $\mathcal{A}'$. We only consider the non-trivial cases.

1. Suppose that $\mathcal{A}' = \textsc{AppendChar}(w(1), \mathcal{B}')$. Let $\mathcal{B}$ be a state optimal NCSSA that recognizes the suffix $w[2, |w|]$ of w. Since $\mathcal{B}$ is a state optimal NCSSA equivalent to $\mathcal{B}'$, we have

$$\begin{aligned} n\big(\textsc{AppendChar}(w(1), \mathcal{B})\big) &= 1 + n\big(\mathcal{B}\big) \\ &\leqslant 1 + n\big(\mathcal{B}'\big) \\ &= n\big(\textsc{AppendChar}(w(1), \mathcal{B}')\big) \\ &= n\big(\mathcal{A}'\big). \end{aligned}$$

   This shows that $\textsc{AppendChar}(w(1), \mathcal{B})$ is a state optimal NCSSA that recognizes w.

2. Suppose $\mathcal{A}' = \textsc{AppendRepetition}(\mathcal{B}', k, \mathcal{C}')$. Let $\mathcal{B}$ be a state optimal NCSSA equivalent to $\mathcal{B}'$ and let $\mathcal{C}$ be a state optimal NCSSA equivalent to $\mathcal{C}'$. Clearly, we have $n(\mathcal{B}) \leqslant n(\mathcal{B}')$ and $n(\mathcal{C}) \leqslant n(\mathcal{C}')$ and hence

$$\begin{aligned} n\big(\textsc{AppendRepetition}(\mathcal{B}, r/q, \mathcal{C})\big) &= 1 + n\big(\mathcal{B}\big) + n\big(\mathcal{C}\big) \\ &\leqslant 1 + n\big(\mathcal{B}'\big) + n\big(\mathcal{C}'\big) \\ &= n\big(\textsc{AppendRepetition}(\mathcal{B}', k, \mathcal{C}')\big) \\ &= n\big(\mathcal{A}'\big). \end{aligned}$$

   This shows that $\textsc{AppendRepetition}(\mathcal{B}, r/q, \mathcal{C})$ is a state optimal NCSSA that recognizes w. □

Algorithm 2.13 below describes a polynomial-time procedure that solves the state optimization problem for (decomposable) NCSSA recognizing finite

Algorithm 2.13. StateOptimal-FiniteCase(w)

```
procedure BestPeriod($P, \mathcal{A}, i, j$)
    $p \leftarrow P(i,j)$
    $k \leftarrow 1$
    for each $2 \leqslant h \leqslant \frac{j-i+1}{p}$
        do  if $\mathcal{A}(i, i+hp-1)$ has fewer states than $\mathcal{A}(i, i+kp-1$
              then $k \leftarrow h$
    return $kp$

for each $1 \leqslant i \leqslant j \leqslant |w|$
  let   $P(i,j) =$ minimum period of $w[i,j]$
for each $1 \leqslant i \leqslant |w|$
  do   $\mathcal{A}(i+1, i) \leftarrow \mathcal{A}_\varepsilon$
for $n \leftarrow 1$ to $|w|$   and   for each $1 \leqslant i \leqslant |w| - n + 1$
  do    $j \leftarrow i + n - 1$
        $Q(i,j) \leftarrow$ BestPeriod($P, \mathcal{A}, i, j$)
        $\mathcal{B}_0 \leftarrow$ AppendChar($w(i)$, $\mathcal{A}(i+1, j)$)
        for each $1 \leqslant r \leqslant n$
            do   $q \leftarrow Q(i, i+r-1)$
                 $\mathcal{B}_r \leftarrow$ AppendRepetition($\mathcal{A}(i, i+q-1)$, $r/q$, $\mathcal{A}(i+r, j)$)
        $\mathcal{A}(i,j) \leftarrow$ BestStatesNum($\mathcal{B}_0, \mathcal{B}_1, ..., \mathcal{B}_n$)
return $\mathcal{A}(1, n)$
```

words. Such a procedure receives as input a finite word w and it returns as output a state optimal NCSSA (among all equivalent decomposable NCSSA) that recognizes w. It uses the following data structures and auxiliary procedures:

- a matrix P, where each entry $P(i,j)$, with $1 \leqslant i \leqslant j \leqslant |w|$, stores the minimum period of the substring $w[i,j]$ of w;
- a matrix $\mathcal{A}$, where each entry $\mathcal{A}(i,j)$, with $1 \leqslant i \leqslant j \leqslant |w|$, stores the generated state optimal NCSSA recognizing the substring $w[i,j]$ of w; (for convenience, for every $1 \leqslant i \leqslant |w|$, the entry $\mathcal{A}(i+1,i)$ stores the empty NCSSA $\mathcal{A}_\varepsilon$);
- a matrix Q, where each entry $Q(i,j)$, with $1 \leqslant i \leqslant j \leqslant |w|$, stores a (possibly non-minimal) period q of the substring $w[i,j]$ such that any state optimal NCSSA recognizing $w[i, i+q-1]$ turns out to have the minimum number of states among all NCSSA recognizing repeating patterns of $w[i,j]$ (each entry $Q(i,j)$ is computed by an auxiliary procedure

BestPeriod which only accesses the minimum period $P(i,j)$ and some entries of $\mathcal{A}$ of the form $\mathcal{A}(i,j')$, with $i \leqslant j' < j$);

- a procedure BestStatesNum, which receives a tuple of NCSSA as input and returns the one having the minimum number of states as output.

It is easy to see that the described optimization algorithm can be executed in time $\Theta(|w|^3)$.

Proposition 9. *For every ultimately periodic word w with shortest initial pattern u and shortest (primitive) repeating pattern v, at least one of the following conditions holds:*

1. *for every state optimal NCSSA $\mathcal{B}$ that recognizes the suffix $w[2,\omega[$ of w,* AppendChar$(w(1),\mathcal{B})$ *is a state optimal NCSSA that recognizes w;*
2. *u is the empty word and there is a positive natural number k such that for every state optimal NCSSA $\mathcal{B}$ that recognizes the repeating pattern v^k,* AppendRepetition$(\mathcal{B},\omega)$ *is a state optimal NCSSA that recognizes w;*
3. *there exists a position $1 \leqslant r < 2|u| + 2|v|$ such that, for every state optimal NCSSA $\mathcal{B}$ that recognizes the prefix $w[1,q]$ of w, where q is a period of $w[1,r]$, and for every state optimal NCSSA $\mathcal{C}$ that recognizes the suffix $w[r+1,\omega[$ of w,* AppendRepetition$(\mathcal{B}, r/q, \mathcal{C})$ *is a state optimal NCSSA that recognizes w.*

Proof. Let $\mathcal{A}'$ be a (decomposable) state optimal NCSSA recognizing w. We prove the claim by exploiting induction on the decomposition structure of $\mathcal{A}'$. We only consider the most difficult case, namely, $\mathcal{A}'$=AppendRepetition$(\mathcal{B}', k, \mathcal{C}')$, with $k > 1$ (it is immediate to see that k cannot be equal to 1 in a state optimal NCSSA). First of all, we can replace $\mathcal{B}'$ and $\mathcal{C}'$ by equivalent state optimal NCSSA $\mathcal{B}$ and $\mathcal{C}$, respectively. Let z be the finite word recognized by $\mathcal{B}$ and let $r = k\,|u|$. We have to show that $r < 2|u| + 2|v|$. This can be proved by contradiction assuming that $r \geqslant 2|u| + 2|v|$ and considering the substring $x = w[|u|+1, r]$ of w, which has partial periods $|v|$ and $|z|$. The rest of the proof goes on as the proof of Proposition 6 and thus is omitted. □

Now, given a natural number n, we denote by $\mathscr{F}(n)$ the set of all decomposable NCSSA that recognize finite words of length exactly n. Moreover, for every natural number m and every positive natural number t, we recursively define the set $\mathscr{F}'(m,t)$ as follows:

$$\mathscr{F}'(0,t) = \{\mathcal{A}_\varepsilon\}$$

$$\begin{aligned}\mathscr{F}'(m+1,t) = {} & \mathscr{F}'(n,r) \\ & \cup \left\{\textsc{AppendChar}(a,\mathcal{B}) \,:\, a \in A,\ \mathcal{B} \in \mathscr{F}(m,t)\right\} \\ & \cup \left\{\textsc{AppendRepetition}(\mathcal{A},k,\mathcal{B}) \,:\, \begin{array}{l} k,p \in \mathbb{N},\ kp \leqslant t, \\ \mathcal{A} \in \mathscr{F}(p), \\ \mathcal{B} \in \mathscr{F}'(m,t) \end{array}\right\}.\end{aligned}$$

Finally, for every pair of natural numbers n, m and every pair of positive natural numbers r, t, we recursively define the set $\mathscr{U}'(n, r, m, t)$ as follows:

$$\mathscr{U}'(0, r, m, t) = \big\{\textsc{AppendRepetition}(\mathcal{B}, \omega, \mathcal{A}_\varepsilon) \,:\, \mathcal{B} \in \mathscr{F}'(m, t)\big\}$$

$$\begin{aligned}\mathscr{U}'(n+1, r, m, t) = {} & \mathscr{U}'(n, r, m, t) \\ & \cup \big\{\textsc{AppendChar}(a, \mathcal{C}) \,:\, a \in A,\ \mathcal{C} \in \mathscr{U}'(n, r, m, t)\big\} \\ & \cup \left\{ \textsc{AppendRepetition}(\mathcal{A}, k, \mathcal{C}) \,:\! \begin{array}{r} k, p \in \mathbb{N},\ k\,p \leqslant r, \\ \mathcal{A} \in \mathscr{F}(p), \hfill \\ \mathcal{C} \in \mathscr{U}'(n, r, m, t) \end{array} \right\}.\end{aligned}$$

As proved by the following proposition, we can assume the parameters m, t, n, r to be bounded by suitable functions linear in the lengths of the initial and repeating patterns of the given ultimately periodic word.

Proposition 10. *For every ultimately periodic word w with shortest initial pattern u and shortest repeating pattern v, there is a complexity optimal NC-SSA that recognizes w and belongs to the set $\mathscr{U}\big(|u| + |v| - 1,\ 2|u| + 2|v| - 1,\ |v|,\ 2|v| - 1\big)$.*

Proof. Let w be an ultimately periodic word and let u and v be, respectively, the shortest initial pattern and the shortest repeating pattern of w. We first prove the upper bounds for the first two parameters, that is, n and r. By recursively applying Proposition 9, we end up with a finite sequence of symbols $a_1, ..., a_n$, some finite sequences of natural numbers $m, t, k_1, ..., k_n$, and $p_1, ..., p_n$, and some finite sequences of state optimal NCSSA $\mathcal{B}, \mathcal{A}_1, ..., \mathcal{A}_n$,and $\mathcal{C}_0, \mathcal{C}_1, ..., \mathcal{C}_n$ such that

- $\mathcal{B} \in \mathscr{F}'(m, t)$,
- for every $1 \leqslant i \leqslant n$, $k_i\,p_i < 2|u| + 2|v|$,
- for every $1 \leqslant i \leqslant n$, $\mathcal{A}_i \in \mathscr{F}(p_i)$,
- $\mathcal{C}_0 = \textsc{AppendRepetition}(\mathcal{B}, \omega, \mathcal{A}_\varepsilon)$,
- for every $1 \leqslant i \leqslant n$, $\mathcal{C}_i$ is either the NCSSA $\textsc{AppendChar}(a_i, \mathcal{C}_{i-1})$ or the NCSSA $\textsc{AppendRepetition}(\mathcal{A}_i, k_i, \mathcal{C}_{i-1})$,
- $\mathcal{C}_n$ recognizes the ultimately periodic word w.

It is easy to see that, for every $0 \leqslant i \leqslant n$, the NCSSA $\mathcal{C}_i$ contains more than $i + 1$ states. Thus, we have $n < |u| + |v|$ (otherwise, $\mathcal{C}_n$ would not be state optimal). This proves that $\mathcal{C}_n$ belongs to the set $\mathscr{U}'\big(|u| + |v| - 1,\ 2|u| + 2|v| - 1,\ m,\ t\big)$.

To prove the upper bound for the parameter m, we consider the NC-SSA $\mathcal{B}$, which belongs to the set $\mathscr{F}'(m, t)$ and has the minimum number of states among all decomposable NCSSA that recognize repeating patterns of w the form v^k, with $k > 0$. By Proposition 8, there exist a finite sequence of symbols $a'_1, ..., a'_m$, two finite sequences of natural numbers $k'_1, ..., k'_m$ and $p'_1, ..., p'_m$, and two finite sequences of state optimal NCSSA $\mathcal{A}'_1, ..., \mathcal{A}'_m$ and $\mathcal{B}'_0, \mathcal{B}'_1, ..., \mathcal{B}'_m$ such that

- for every $1 \leqslant i \leqslant n$, $\mathcal{A}'_i \in \mathscr{F}(p'_i)$,
- $\mathcal{B}'_0$ is the empty NCSSA $\mathcal{A}_\varepsilon$,
- for every $1 \leqslant i \leqslant n$, $\mathcal{B}'_i$ is either the NCSSA $\textsc{AppendChar}(a'_i, \mathcal{B}'_{i-1})$ or the NCSSA $\textsc{AppendRepetition}(\mathcal{A}'_i, k'_i, \mathcal{B}'_{i-1})$,
- $\mathcal{B}'_m$ is the NCSSA $\mathcal{B}$.

Using an argument similar to the one we used to establish the bound $n < |u| + |v|$, one can prove that $m \leqslant |v|$.

It remains to prove that $t < 2|v|$, namely, that for every index $1 \leqslant i \leqslant n$, $\mathcal{B}'_i = \textsc{AppendRepetition}(\mathcal{A}'_i, k'_i, \mathcal{B}'_{i-1})$ implies $k'_i p'_i < 2|v|$. Suppose, by way of contradiction, that there is an index $1 \leqslant i \leqslant n$ such that $\mathcal{B}'_i = \textsc{AppendRepetition}(\mathcal{A}'_i, k'_i, \mathcal{B}'_{i-1})$ and $k'_i p'_i \geqslant 2|v|$. Furthermore, let x_i the word recognized by $\mathcal{A}'_i$, let $y_i = x_i^{k'_i}$, and let z_{i-1} be the word recognized by $\mathcal{B}'_{i-1}$. First of all, one observes that $k'_i \geqslant 2$ (otherwise, $y_i\, z_{i-1} = x_i\, z_{i-1}$ would follow and $\mathcal{B}'_i$ would not be a size-optimal NCSSA). We thus have

$$|y_i| \;=\; k'_i\,|x_i| \;\geqslant\; \max\bigl(2\,|x_i|,\, 2\,|v|\bigr) \;\geqslant\; |x_i| + |v|.$$

Moreover, both p'_i and $|v|$ are partial periods of y_i. We can thus apply Lemma 1 and obtain that $q_i = \gcd\bigl(p'_i, |v|\bigr)$ is a partial period of y_i as well. Let us now consider the prefix $z_i = y_i[1, |v|]$ of y_i. It has length equal to $|v|$ and partial period q_i. By construction, q_i divides $|v|$ and hence q_i is an exact period of z_i. However, from the fact that v is *primitive*, we know that $q_i = |v|$. Similarly, q_i divides p'_i and hence both words x_i and y_i $(= x_i^{k'_i})$ are repetitions of v. This basically shows that the NCSSA $\mathcal{B}$ $(= \mathcal{B}'_m)$, which recognizes the repeating pattern v^k of w, can be replaced by a decomposable NCSSA $\mathcal{B}'$, which has fewer states and recognizes the repeating pattern $v^{k-k'_i+1}$ of w. This contradicts the hypothesis that $\mathcal{C}_n$ is a state optimal NCSSA recognizing w. We thus conclude that $\mathcal{B}_m$ belongs to the set $\mathscr{F}'(|v|, 2|v| - 1)$ and hence $\mathcal{C}_n$ belongs to the set $\mathscr{U}'(|u| + |v| - 1,\ 2|u| + 2|v| - 1,\ |v|,\ 2|v| - 1)$. □

Putting all the above results together, we can devise a polynomial-time procedure (see Algorithm 2.14) that solves the state optimization problem. Such a procedure receives as input an ultimately periodic word w, which is represented by a pair (u, v), with u and v being, respectively, the shortest initial pattern and the shortest repeating pattern of w, and it returns as output a decomposable NCSSA which recognizes w and which is state optimal (among all equivalent decomposable NCSSA). It uses the same data structures and auxiliary procedures of Algorithm 2.13, plus the following ones:

- a matrix $\mathcal{B}$, where each entry $\mathcal{B}(i, j)$, with $|u| + 1 \leqslant i, j \leqslant |u| + |v|$, stores an NCSSA which recognizes a non-empty substring of the form $w[i, j + h\,|v|]$, with $h \geqslant 0$, and having the minimum number of states among all decomposable NCSSA recognizing non-empty substrings of the same form namely, $w[i, j + h'\,|v|]$, with $h' \geqslant 0$;
- an array $\mathcal{C}$, where each entry $\mathcal{C}(i)$, with $1 \leqslant i \leqslant |u| + |v|$, stores the generated complexity optimal NCSSA recognizing the suffix $w[i, \omega[$ of w; (for convenience, each entry $\mathcal{C}(i)$, with $1 \leqslant i \leqslant |u|$, is initialized with a dummy NCSSA $\mathcal{A}_\infty$ having a sufficiently large number of states);
- a procedure Normalize, which receives as input the initial pattern u, the repeating pattern v, and a position i of the ultimately periodic word $w = uv^\omega$ and it returns as output either the value i or the value $\big((i - |u| - 1) \bmod |v|\big) + |u| + 1$, depending on whether $i \leqslant |u|$ or $i > |u|$.

It is easy to see that the described optimization algorithm can be executed in time $\Theta\big((|u| + |v|)^3\big)$.

Algorithm 2.14. StateOptimal-InfiniteCase(w)

```
$w \leftarrow (uv^\omega)\,[1, 3|u| + 3|v| - 2]$
for each $1 \leqslant i \leqslant j \leqslant 3|u| + 3|v| - 2$
  let  $P(i, j) = $ minimum period of $w[i, j]$
       $\mathcal{A}(i, j) = $ complexity optimal NCSSA recognizing $w[i, j]$
       $Q(i, j) = $ best period of $w[i, j]$
for each $|u| + 1 \leqslant i \leqslant j \leqslant |u| + |v|$
  do   $\mathcal{B}(i, j) \leftarrow \mathcal{A}(i, j)$
for each $|u| + 1 \leqslant j < i \leqslant |u| + |v|$
  do   $\mathcal{B}(i, j) \leftarrow \mathcal{A}(i, |v| + j)$
for each $1 \leqslant n|v| - 1$   and   $|u| + 1 \leqslant i, j \leqslant |u| + |v|$
  do   $i' \leftarrow$ Normalize$(u, v, i + 1)$
       $\mathcal{D}_0 \leftarrow$ AppendChar$(w(i),\ \mathcal{B}(i', j))$
       for each $1 \leqslant r \leqslant 2|v| - 1$
         do   $q \leftarrow Q(i, i + r - 1)$
              $i' \leftarrow$ Normalize$(u, v, i + r)$
              $\mathcal{D}_r \leftarrow$ AppendRepetition$\begin{pmatrix} \mathcal{A}(i, i + q - 1), \\ r/q, \\ \mathcal{B}(i', j) \end{pmatrix}$
       $\mathcal{B}(i, j) \leftarrow$ BestStatesNum$(\mathcal{B}(i, j), \mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{2|v|-1})$
```

(continues on next page)

Algorithm 2.14 (continued)

```
for each $1 \leqslant i \leqslant |u|$
  do  $\mathcal{C}(i) \leftarrow \mathcal{A}_\infty$
for each $|u| + 1 \leqslant i \leqslant |u| + |v|$
  do  $j \leftarrow$ Normalize$(u, v, i + |v| - 1)$
      $\mathcal{C}(i) \leftarrow$ AppendRepetition$(\mathcal{B}(i, j), \omega, \mathcal{A}_\varepsilon)$
for each $1 \leqslant n \leqslant |u| + |v| - 1$  and  $1 \leqslant i \leqslant |u| + |v|$
  do  $i' \leftarrow$ Normalize$(u, v, i + 1)$
      $\mathcal{D}_0 \leftarrow$ AppendChar$(w(i), \mathcal{C}(i'))$
      for each $1 \leqslant r \leqslant 2|u| + 2|v| - 1$
        do  $q \leftarrow Q(i, i + r - 1)$
            $i' \leftarrow$ Normalize$(u, v, i + r)$
            $\mathcal{D}_r \leftarrow$ AppendRepetition$(\mathcal{A}(i, i + q - 1), r/q, \mathcal{C}(i'))$
      $\mathcal{C}(i) \leftarrow$ BestStatesNum$(\mathcal{C}(i), \mathcal{D}_0, \mathcal{D}_1, ..., \mathcal{D}_{2|u|+2|v|-1})$
return $\mathcal{C}(1)$
```

2.4 Reasoning on Sets of Granularities

In the previous sections, we showed how to exploit various classes of single-string automata, that is, SSA, RCSSA, and NCSSA, to provide compact and tractable representations of single time granularities. Things become more involved when we consider sets of time granularities instead of single time granularities. In this section, we extend the automaton-based approach to make it possible to deal with (possibly infinite) sets of time granularities. To this end, we identify a proper subclass of Büchi automata, called Ultimately Periodic Automata, which captures rational ω-languages consisting of ultimately periodic words only. Ultimately Periodic Automata allow one to encode single time granularities, (possibly infinite) sets of time granularities that feature the same repeating pattern, but different prefixes, and sets of time granularities characterized by a finite set of non-equivalent repeating patterns (a formal notion of equivalence for repeating patterns will be given in the sequel), as well as any possible combination of them.

2.4.1 Languages of Ultimately Periodic Words

In this section, we study some fundamental properties of rational ω-languages, that is, languages of infinite words recognized by Büchi automata, consisting of ultimately periodic words only. We first recall some fundamental results about rational ω-languages (for basic definitions and notations about rational ω-languages, we refer the reader to Section 2.1.3).

Proposition 11 (Büchi [2]). *The class of rational ω-languages is effectively closed under union, intersection, and complementation, namely, given two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, one can compute a Büchi automaton $\mathcal{A} \cup \mathcal{B}$ (resp., $\mathcal{A} \cap \mathcal{B}$, $\bar{\mathcal{A}}$) that recognizes the ω-language $\mathscr{L}^\omega(\mathcal{A}) \cup \mathscr{L}^\omega(\mathcal{B})$ (resp., $\mathscr{L}^\omega(\mathcal{A}) \cap \mathscr{L}^\omega(\mathcal{B})$, $A^\omega \setminus \mathscr{L}^\omega(\mathcal{A})$).*

Proposition 12 (Büchi [2]). *An ω-language L is rational iff it is a finite union of sets of the form $U\,V^\omega$, where U and V are rational languages of finite words.*

Hereafter, we denote by U_A the universal ω-language that consists of all and only the ultimately periodic words over A. Moreover, given an ω-language $L \subseteq A^\omega$, we denote by $\mathcal{UP}(L)$ the ω-language $L \cap U_A$, which consists of all and only the ultimately periodic words that belong to L. Clearly, an ω-language L consists only of ultimately periodic words if and only if $L = \mathcal{UP}(L)$.

Proposition 13 (Büchi [2], Calbrix et al. [8]). *Every non-empty rational ω-language contains at least one ultimately periodic word. Moreover, if L_1 and L_2 are two rational ω-languages, then $L_1 = L_2$ iff $\mathcal{UP}(L_1) = \mathcal{UP}(L_2)$.*

Proof. As for the first claim, by Proposition 12, any rational ω-language L can be written as $\bigcup_{1 \leqslant i \leqslant n} U_i\,V_i^\omega$, with $\varepsilon \notin V_i$ for every $1 \leqslant i \leqslant n$. Since L is not empty, there exists an index $1 \leqslant i \leqslant n$ such that both U_i and V_i are not empty. Therefore, L must contain an ultimately periodic word of the form $w = u v^\omega$, with $u \in U_i$ and $v \in V_i$.

As for the second claim, let L_1 and L_2 be two rational ω-languages containing the same ultimately periodic words. The left-to-right implication is trivial. For the converse implication, we know, from closure properties of rational ω-languages (see Proposition 11), that $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is a rational ω-language, which contains no ultimately periodic words. Thus $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is empty and $L_1 = L_2$ follows. □

In the following, we provide a characterization of rational ω-languages of ultimately periodic words only, in analogy with that of Proposition 12.

To start with, we point out that there exist non-rational ω-languages consisting of ultimately periodic words only: for instance, since A^ω is a rational ω-language, $\mathcal{UP}(U_A) = \mathcal{UP}(A^\omega)$, and $U_A \neq A^\omega$, then, by Proposition 13, U_A cannot be a rational ω-language.

Proposition 14. *The following closure properties hold:*

i) if v is a non-empty finite word, $\{v\}^\omega$ is a rational ω-language consisting of a single ultimately periodic word;

ii) if U is a rational language and V is a rational ω-language of ultimately periodic words, then $U\,V$ is a rational ω-language of ultimately periodic words;

iii) if L_1 and L_2 are rational ω-languages of ultimately periodic words, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are rational ω-languages of ultimately periodic words.

Proof. The claim immediately follows from closure properties of rational ω-languages, since the above operations do not introduce words which are not ultimately periodic. $\square$

As for the complementation of an ω-language of ultimately periodic words, it must be obviously defined with respect to the universe U_A, that is, the complement of $L \in U_A$ is $\bar{L} = U_A \setminus L$. Notice that there is no guarantee that $\bar{L}$ is rational whenever L is rational.

Proposition 15. *Rational ω-languages of ultimately periodic words are not closed under complementation.*

Proof. A counterexample is given by the empty set: it trivially is a rational ω-language of ultimately periodic words, but its complement is the universal ω-language U_A, which is not rational. In fact, the complement of *any* rational ω-language L of ultimately periodic words is not rational, since $U_A = L \cup \bar{L}$. $\square$

Given an ultimately periodic word $w = u\,v^{\omega}$, the set of its repeating patterns is clearly infinite and it contains, among others, the finite words v, v^2, v^3, ... To group together the different repeating patterns of an ultimately periodic word, we define a suitable equivalence relation. Such an equivalence will play an essential role in the characterization of rational ω-languages of ultimately periodic words we are going to provide.

Definition 9. *Let $\approx \subseteq A^* \times A^*$ be an equivalence relation such that $u \approx v$ iff the two infinite periodic words u^{ω} and v^{ω} share a common suffix, namely, there exist $x, y \in A^*$ and $z \in A^{\omega}$ such that $u^{\omega} = x\,z$ and $v^{\omega} = y\,z$.*

Notice that in Definition 9 one can always assume either x or y to be ε.

It can be easily checked that all repeating patterns of a given ultimately periodic word w are equivalent. Moreover, they can be obtained by choosing different repetitions and/or different rotations[2] of the *primitive* repeating pattern of w, namely, the shortest substring $w[i,j]$ of w such that (i) $w = w[1,i-1]\,(w[i,j])^{\omega}$ and (ii) either $i = 1$ or $w(j) \neq w(i-1)$. Conversely, if v is a repeating pattern of an ultimately periodic word w and v' is equivalent to v, then v' is also a repeating pattern of w.

Given an ω-language L and a finite word v, we say that L *features* v as a repeating pattern if L contains an ultimately periodic word w having v as a repeating pattern; moreover, if v belongs to a language of the form V^+, with $V \subseteq A^*$, then we say that v is a V-*aligned* repeating pattern.

Below, we prove some fundamental properties of ω-languages of the form V^{ω}, where $V \subseteq A^*$, with respect to the repeating patterns they feature.

Lemma 6. *Given a language V, for every repeating pattern v featured by V^{ω}, there exists an equivalent V-aligned repeating pattern z featured by V^{ω}.*

[2] A word v' is said to be a rotation of a word v if there exist two possibly empty words x, y such that $v = x\,y$ and $v' = y\,x$.

Proof. Let ν be a repeating pattern featured by V^ω. By definition, V^ω contains an infinite word $w = u_1 u_2 u_3 \dots$, with $u_i \in V$ for all $i \geqslant 0$, which is ultimately periodic with ν as a repeating pattern. Thus, w can be written as $u \nu^\omega$, where u is a suitable finite word. Let i_0 be a sufficiently large index such that u turns out to be a prefix of $u_1 u_2 \dots u_{i_0}$ (or, equivalently, $u_{i_0+1} u_{i_0+2} \dots$ turns out to be a suffix of ν^ω). Moreover, let f be the function that maps any natural number $i \geqslant i_0$ to the value

$$f(i) = \big(|u_1 u_2 \dots u_i| - |u|\big) \bmod |\nu|.$$

Since the image of f is finite, by the Pigeonhole Principle there exist two indices i, i', with $i_0 \leqslant i < i'$, such that $f(i) = f(i')$. By definition of f, we have that the length of the substring $z = u_{i+1} \dots u_{i'}$ of w is a multiple of $|\nu|$. Since $i \geqslant i_0$, z is also a substring of ν^ω, which implies that (i) z is a repeating pattern equivalent to ν and (ii) $z \in V^+$. □

Proposition 16. *Given a language* V*, if* V^ω *is non-empty and it features only equivalent repeating patterns, then* $V^\omega = \{\nu\}^\omega$ *for a suitable non-empty finite word* ν*.*

Proof. Suppose that V^ω is a non-empty ω-language featuring only equivalent repeating patterns. Let $\nu_1, \nu_2, \nu_3, \dots$ be all and only the V-aligned repeating patterns featured by V^ω.

We first prove that, for every pair of indices $i, j > 0$, $\nu_i^\omega = \nu_j^\omega$. Let $i, j > 0$ be two generic indices and let q_i and q_j be two positive natural numbers such that $q_i|\nu_i| = q_j|\nu_j|$. We define $\nu'_i = \nu_i^{q_i}$ and $\nu'_j = \nu_j^{q_j}$. By hypothesis, we have $\nu_i \approx \nu_j$, from which $\nu'_i \approx \nu'_j$ follows. Below, we prove that ν'_i and ν'_j coincide. Since $\nu'_i \approx \nu'_j$ and $|\nu'_i| = |\nu'_j|$, ν'_i must be a rotation of ν'_j, namely, there exist two finite words x and y such that $\nu'_i = x\,y$ and $\nu'_j = y\,x$. Since both ν'_i and ν'_j belong to V^+, we have that $\nu'_{i,j} = \nu'_i \nu'_j$ belongs to V^+ and thus it is a repeating pattern for V^ω. Hence, by hypothesis, $\nu'_{i,j} \approx \nu'_i$. This implies that $\nu'_{i,j}$ ($= x\,y\,y\,x$) is a rotation of $\nu'_i\nu'_i$ ($= x\,y\,x\,y$) and hence there is a suitable (possibly empty) word z such that $(z)\,(x\,y\,y\,x)$ is a prefix of $(x\,y)\,(x\,y)\,(x\,y)$. Now, let us denote by u the primitive repeating pattern of $x\,y$. Since $(z)\,(x\,y)$ is a prefix of $(x\,y)\,(x\,y)\,(x\,y)$, we have that either $z = \varepsilon$ or $z^\omega = (x\,y)^\omega$. From the minimality of u, it follows that $z = u^p$ for some $p \geqslant 0$. Therefore, since $(z\,x\,y)\,(y\,x)$ is also a prefix of $(x\,y)\,(x\,y)\,(x\,y)$, $|z\,x\,y|$ is a multiple of $|u|$, and $|y\,x| = |x\,y|$, we have that $y\,x = u^q$ ($= x\,y$) for some $q > 0$. This allows us to conclude that, for every pair of indices $i, j > 0$, ν'_i ($= x\,y$) $= \nu'_j$ ($= y\,x$) and hence $\nu_i^\omega = \nu_j^\omega$.

Now, let ν be the shortest repeating pattern of the infinite periodic word ν_i^ω, where $i > 0$ is an arbitrary index (ν does not depend on i). We have that $V^\omega = \{\nu\}^\omega$. □

Proposition 17. *Given a language* V*, if* V^ω *features at least two repeating patterns which are not equivalent, then* V^ω *contains an infinite word which is not ultimately periodic.*

Proof. Let V^ω be an ω-language featuring two non-equivalent repeating patterns u and v. By Lemma 6, there exist two V-aligned repeating patterns $u', v' \in V^+$ such that $u' \approx u$ and $v'v$. For every $i > 0$, we denote by z_i the finite word $(u')^i\,(v')^i$ and we define the infinite word $w = z_1\,z_2 \ldots$. Clearly, $z_i \in V^+$ holds for all $i > 0$ and hence w belongs to V^ω. It remains to show that w is not an ultimately periodic word. Suppose, by way of contradiction, that w is an ultimately periodic word having z' as repeating pattern. By construction, there exists an index i such that $(u')^{|z'|}$ is a substring of z_i and thus of w. Since z' is a repeating pattern of w and $|(u')^{|z'|}|$ is a multiple of $|z'|$, we have that $(u')^{|z'|}$ is a repetition of some rotation of z' and hence $u' \approx z'$. In a similar way, we can show that $v' \approx z'$. By transitivity, we have $u' \approx v'$ and thus $u \approx v$, which is against the hypothesis of u and v being two non-equivalent repeating patterns. □

Proposition 18. *Given a language* V*, exactly one of the following conditions holds:*

1. V^ω *features only equivalent repeating patterns;*
2. V^ω *features infinitely many non-equivalent repeating patterns.*

Proof. Let us assume that V^ω features at least two non-equivalent repeating patterns u and v. By Lemma 6, V^ω features two V-aligned repeating patterns u' and v', with $u' \approx u$ and $v' \approx v$. Moreover, since $\approx$ is an equivalence relation and $u \not\approx v$, we have $u' \not\approx v'$. Now, let $n = |u'| + |v'|$ and, for every $i > 0$, $p_i = n^{i-1}$ and $z_i = (u')^{p_i}\,(v')^{p_i}$. Every word z_i is clearly a V-aligned repeating pattern featured by V^ω. We prove that the words z_i are pairwise non-equivalent, that is, $z_i \not\approx z_j$ for every pair of distinct indices $i, j > 0$.

Suppose, by way of contradiction, that there exist two indices $0 < i < j$ such that $z_i \approx z_j$. By definition of $\approx$, there exists an infinite periodic word w that features both z_i and z_j $(= (u')^{p_j}\,(v')^{p_j})$ as repeating patterns. Moreover, since $i < j$, we have that p_j $(= n^{j-1})$ is a multiple of $|z_i|$ $(= n^i)$, which implies that $(u')^{p_j}$ is a rotation of some repetition of z_i. This shows that $u' \approx z_i$. A similar argument shows that $v' \approx z_i$. Thus, by transitivity, we obtain $u' \approx v'$, which contradicts the hypothesis of u' and v' being non-equivalent repeating patterns. □

The following theorem shows how the above characterization results can be easily generalized to the whole class of rational ω-languages consisting of ultimately periodic words only.

Theorem 1. *Given a rational* ω*-language* L*, the following conditions are equivalent:*

i) L *consists of ultimately periodic words only;*

ii) L *features only finitely many non-equivalent repeating patterns;*

iii) L *is a finite union of* ω*-languages of the form* $U\{v\}^\omega$*, where* U *is a rational language and* v *is a non-empty finite word.*

Proof. We first prove the implication from i) to ii) by contraposition. Let L be a rational ω-language. We can write L as a finite union of the form $\bigcup_{1 \leqslant i \leqslant n} U_i\, V_i^{\omega}$. If L features infinitely many non-equivalent repeating patterns, then there exists an index $1 \leqslant i \leqslant n$ such that $U_i\, V_i^{\omega}$ (and hence V_i^{ω}) features at least two non-equivalent repeating patterns. Thus, by Proposition 17, it would follow that V_i^{ω} (and hence L) contains an infinite word which is not ultimately periodic.

As for the implication from ii) to iii), let L be a rational ω-language featuring only finitely many non-equivalent repeating patterns. By Proposition 12, we can write L as a finite union of the form $\bigcup_{1 \leqslant i \leqslant n} U_i\, V_i^{\omega}$. Moreover, from Proposition 18, it follows that each ω-language V_i^{ω} features only equivalent repeating patterns (otherwise, L would feature infinitely many non-equivalent repeating patterns). Then, by exploiting Proposition 16, we can write each ω-language V_i^{ω} as $\{v_i\}^{\omega}$, where v_i is a suitable non-empty finite word. As a consequence, L can be written as a finite union of the form $\bigcup_{1 \leqslant i \leqslant n} U_i\, \{v_i\}^{\omega}$.

The last implication from iii) to i) is trivial. □

2.4.2 Ultimately Periodic Automata

In this section, we provide an automata counterpart to rational ω-languages of ultimately periodic words. Theorem 1 basically states that these languages model sets of ultimately periodic words with possibly infinitely many initial patterns, but only a finite number of non-equivalent repeating patterns. Moreover, it yields a straightforward definition of a restricted class of Büchi automata that captures exactly the rational ω-languages of ultimately periodic words. As a matter of fact, an alternative view of such a class of automata is also possible: they can be seen as a natural extension of non-deterministic finite-state automata (NFA for short), where final states actually recognize infinite words of the form v^{ω}. This alternative view will clearly show up in the definition of prefix automaton given in Section 2.4.2.

As a preliminary step, we introduce the notion of strongly connected component of a state of an automaton $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$. Let us view $\mathcal{A}$ as a finite labeled graph. The *strongly connected component* of a state $s \in S$ is the subgraph of $\mathcal{A}$ induced by the maximal set of states M_s that are reachable from s and from which s is reachable, that is, M_s consists of all and only the states s' in S such that both (s, s') and (s', s) belong to Δ^*. A state $s \in S$ is called *transient* if $(s, s) \notin \Delta^+$ (notice that it immediately follows that a transient state does not belong to any loop of $\mathcal{A}$). Let us consider the following subclass of Büchi automata (for the sake of simplicity, we assume every state of the automaton to be reachable from any initial state).

Definition 10. *An* ultimately periodic automaton *(UPA for short) is a Büchi automaton* $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ *such that, for every final state* $s \in \mathcal{F}$,

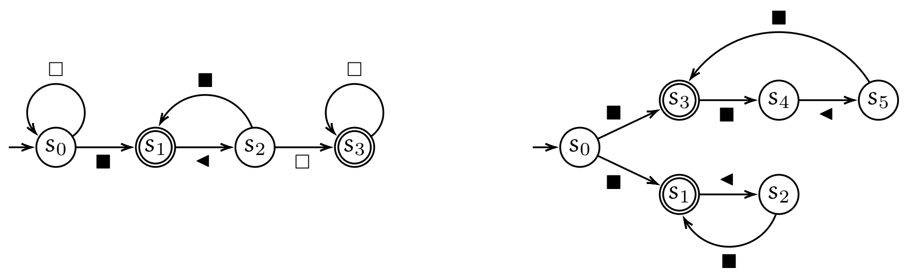


Fig. 2.11. Two examples of UPA

either s *is a transient state or the strongly connected component of* s *is a simple loop*[3].

Notice that Definition 10 does not prevent non-transient final states from having in-degree or out-degree greater than 1 in (the finite labeled graph corresponding to) $\mathcal{A}$.

Examples of UPA are given in Figure 2.11, with reference to the alphabet for time granularity introduced in Section 2.2.1 which consists of the three symbols $\blacksquare$, $\square$, and $\blacktriangleleft$. The UPA to the left recognizes the ω-language $\{\square\}^*\{\blacksquare\blacktriangleleft\}^\omega \cup \{\square\}^*\{\blacksquare\blacktriangleleft\}^*\{\square\}^\omega$ while that to the right recognizes the ω-language $\{\blacksquare\blacksquare\blacktriangleleft\}^\omega \cup \{\blacksquare\blacktriangleleft\}^\omega$. The former represents the (unanchored finite or infinite) granularities that group days two by two, while the latter represents the set consisting of two infinite granularities that group days respectively two by two and three by three.

By exploiting standard construction methods for Büchi automata, one can easily show that UPA-recognizable languages are effectively closed under unions, intersections with rational ω-languages, left-concatenations with rational languages, generalized products, and homomorphisms (i.e., substitutions of non-empty strings for symbols):

i) if L_1 and L_2 are two UPA-recognizable ω-languages, then $L_1 \cup L_2$ is an UPA-recognizable ω-language as well;
ii) if L_1 is a rational ω-language and L_2 is an UPA-recognizable ω-language, then $L_1 \cap L_2$ is an UPA-recognizable ω-language as well;
iii) if L_1 and L_2 are two UPA-recognizable ω-languages, then the ω-language $L_1 \times L_2 = \big\{w \,:\, \exists\, u \in L_1.\ \exists\, v \in L_2.\ \forall\, i > 0.\ w(i) = \big(u(i), v(i)\big)\big\}$ is UPA-recognizable as well;
iv) if L_1 is a rational language and L_2 is an UPA-recognizable ω-language, then L_1L_2 is an UPA-recognizable ω-language as well;
v) if L is an UPA-recognizable ω-language and τ is a function from A to A^+, then the ω-language $\tau(L) = \big\{\tau(a_1)\,\tau(a_2)\dots \ :\ a_1\,a_2\dots \in L\big\}$ is an UPA-recognizable ω-language as well.

[3] A strongly connected component is said to be a simple loop if and only if all its vertices have both in-degree and out-degree equal to 1.

In addition, it is easy to see that UPA satisfy a weak form of closure under ω-exponentiation, namely, for every non-empty finite word v, there exists an UPA recognizing the singleton ω-language $\{v\}^{\omega}$. On the other hand, UPA-recognizable languages are not closed under complementation: this is an immediate consequence of Proposition 15 and Theorem 2 below, which characterizes UPA-recognizable languages. Moreover, the *deterministic* versions of UPA are strictly less expressive than the non-deterministic ones: as it is well-known, the UPA-recognizable ω-language $\{a, b\}^{*}\{b\}^{\omega}$ is not recognizable by any deterministic Büchi automaton (and thus by any deterministic UPA).

We refer the reader to Section 2.5 for further details about UPA-recognizable languages and their complements, as well as for a short survey of related classes of word automata.

Theorem 2. *UPA recognize exactly the rational* ω*-languages of ultimately periodic words.*

Proof. Let $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ be an UPA, w be an infinite word accepted by $\mathcal{A}$, and ρ be a successful run of $\mathcal{A}$ on w. We denote by s a final state of $\mathcal{A}$ that occurs infinitely often in ρ. Clearly, s is not a transient state and hence, by definition of UPA, its strongly connected component is a simple loop. Thus, there is a *unique* infinite run ρ' of $\mathcal{A}$ that starts from s and visits s infinitely often. Such a run is a suffix of ρ of the form $\rho' = \big(\rho(i)\,\rho(i+1)\ \ldots\rho(j-1)\big)^{\omega}$, where i and j are the positions of two consecutive occurrences of s in ρ. This proves that ρ, and hence w, are ultimately periodic sequences.

As for the converse implication, we have to show that, given a rational ω-language L of ultimately periodic words, there is an UPA recognizing L. By exploiting Theorem 1, we know that $L = \bigcup_{1 \leqslant i \leqslant n} U_i \{v_i\}^{\omega}$ for a suitable n, suitable rational languages $U_1, \ldots, U_n$, and suitable non-empty finite words $v_1, \ldots, v_n$. Such a characterization implicitly defines the three basic operations on UPA: the ω-exponentiation of a non-empty finite word, the concatenation with a rational language, and the finite union. Thus, from closure properties of UPA, it follows that there exists an UPA recognizing L. □

In the following subsections, we will introduce three normalized forms for UPA, which we will respectively call normal form, prefix-friendly form, and canonical form. We will prove that these normalized forms satisfy several desirable properties (e.g., the canonical form is proved to be unique, up to isomorphisms, among all equivalent UPA) and ease algorithmic manipulation. We will also prove that normal and prefix-friendly forms can be computed at low cost (precisely, the former can be computed in linear time and the latter can be computed in quadratic time with respect to the input UPA).

A Normal Form for UPA

Given a loop C of an UPA $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$, we say that C is a *final* loop if it contains at least one final state. Moreover, we say that a final loop C

encodes the repeating pattern $\nu \neq \varepsilon$ if and only if there is a run ρ of $\mathcal{A}$ on ν^ω that starts with a state $s \in C$. It is easy to see that a final loop C encodes only equivalent repeating patterns and, conversely, if ν and ν' are equivalent repeating patterns, then C encodes ν iff C encodes ν'. Thus, given two final loops C_1 and C_2, either C_1 and C_2 encode the same repeating patterns, or C_1 and C_2 encode repeating patterns which are pairwise non-equivalent.

Due to the peculiar structure of UPA, every successful run of an UPA consists of a finite prefix followed by an infinite repetition of a final loop. In particular, given a final loop C, the number and the positions of the final states of C are irrelevant (C encodes the same set of repeating patterns, independently from which states of C are chosen to be final). Similarly, marking a transient state as final state has no effect, since in any run of the automaton it occurs at most once. Finally, we can assume that no transitions exit from final loops (if this were the case, we could simply duplicate the final loop and let one copy of it to be final with no exiting transition and the other copy to be non-final with some exiting transitions). Putting together the above observations, we can obtain a normal form for UPA, which forces final states to coincide with the states of the final loops and forbids transitions exiting from final loops.

Definition 11. *An UPA $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ is said to be in* normal form *if the following conditions hold:*

- *every final state is reachable from an initial state,*
- *every final state belongs to a (final) loop,*
- *every state in a final loop is final,*
- *there are no transitions exiting from final loops, namely, for every triple (r, a, s) in Δ, $r \in \mathcal{F}$ implies $s \in \mathcal{F}$.*

By restricting to UPA in normal form, one can easily distinguish between components recognizing initial patterns and components recognizing repeating patterns of ultimately periodic words (note that the former components behave like NFA, while the latter ones behave like single-string automata [25]). The following proposition proves that there is no loss of expressiveness if we restrict ourselves to UPA in normal form.

Proposition 19. *Given an UPA $\mathcal{A}$, one can compute in time $\mathcal{O}(|\mathcal{A}|)$ an equivalent UPA $\mathcal{B}$ in normal form. Moreover, $|\mathcal{B}|$ is linear in $|\mathcal{A}|$.*

Proof. Let $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ and $C_1, ..., C_k$ be all and only the final loops of $\mathcal{A}$. By definition of UPA, $C_1, ..., C_k$ are disjoint subgraphs of S. For every $1 \leqslant i \leqslant k$, we introduce a copy $\widetilde{C}_i$ of each final loop C_i and, for every state s of C_i, we denote by $\widetilde{s}$ the corresponding state of $\widetilde{C}_i$. We then define $\mathcal{B} = (A, S', \Delta', \mathcal{I}', \mathcal{F}')$ as follows:

- $S' = S \cup \bigcup_{1 \leqslant i \leqslant k} \{\widetilde{s} : s \in C_i\}$;
- Δ' contains all triples of the form

1. (r, a, s), with $(r, a, s) \in \Delta$,
2. $(r, a, \widetilde{s})$, with $(r, a, s) \in \Delta$, $r \notin C_i$, $s \in C_i$, and $1 \leqslant i \leqslant k$,
3. $(\widetilde{r}, a, \widetilde{s})$, with $(r, a, s) \in \Delta$, $r, s \in C_i$, and $1 \leqslant i \leqslant k$;

- $\mathcal{I}' = \mathcal{I} \cup \bigcup_{1 \leqslant i \leqslant k} \{\widetilde{s} : s \in \mathcal{I}, s \in C_i\}$;
- $\mathcal{F}' = \bigcup_{1 \leqslant i \leqslant k} \{\widetilde{s} : s \in C_i\}$.

It can be easily checked that $\mathcal{B}$ is an UPA in normal form equivalent to $\mathcal{A}$. □

On the grounds of Proposition 19, one can devise a simple linear-time algorithm that receives a generic UPA as input and returns an equivalent UPA in normal form as its output.

A Prefix-Friendly Form for UPA

In the following, we introduce an alternative way of representing rational ω-languages of ultimately periodic words as NFA over an extended alphabet. By definition, any UPA $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ in normal form contains only finitely many (pairwise disjoint) final loops, say $C_1, ..., C_k$. Hereafter, we denote by $A_{\mathcal{A}}$ an extended alphabet, which consists of symbols from A plus symbols of the form (s, C_i), with $1 \leqslant i \leqslant k$ and s being a state of C_i. Intuitively, an NFA representing $\mathcal{A}$ can be obtained by (i) adding a new global final state f, (ii) removing every transition departing from a final state, and (iii) adding a (s, C_i)-labeled transition from s to f for each state s belonging to the final loop C_i.

Definition 12. *Given an UPA $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ in normal form, we define the* prefix automaton *of $\mathcal{A}$ as the NFA $\mathcal{A}_{pre} = (A_{\mathcal{A}}, S', \Delta', \mathcal{I}', \mathcal{F}')$, where*

- *$S' = S \cup \{f\}$, with f being a fresh state not belonging to S;*
- *Δ' contains all triples of the form*
  1. *(q, a, s), with $(q, a, s) \in \Delta$, $q \in S \setminus \mathcal{F}$, and $s \in S$,*
  2. *(s, b, f), with $b = (s, C_i)$, $s \in C_i$, and $1 \leqslant i \leqslant k$;*
- *$\mathcal{I}' = \mathcal{I}$;*
- *$\mathcal{F}' = \{f\}$.*

As an example, Figure 2.12 depicts an UPA $\mathcal{A}$ in normal form, which recognizes the language $\{\square(\blacktriangleleft\blacktriangleleft)^n\blacktriangleleft\square\square^\omega : n \in \mathbb{N}\} \cup \{\square\blacktriangleleft^\omega\}$, together with its prefix automaton $\mathcal{A}_{pre}$, which recognizes the language $\{\square(\blacktriangleleft\blacktriangleleft)^n\square\, b_3 : n \in \mathbb{N}\} \cup \{\square\, b_4\}$, where $b_3 = (s_3, C_3)$, $b_4 = (s_4, C_4)$, C_3 is the final loop of s_3, and C_4 is the final loop of s_4.

Notice that the prefix automaton $\mathcal{A}_{pre}$ uniquely identifies the UPA $\mathcal{A}$, that is, one can obtain $\mathcal{A}$ from $\mathcal{A}_{pre}$ by (i) marking as final all states in $C_1, ..., C_k$, (ii) adding the transitions of $C_1, ..., C_k$, which can be recovered from the symbols belonging to extended alphabet $A_{\mathcal{A}}$, and (ii) removing the global final

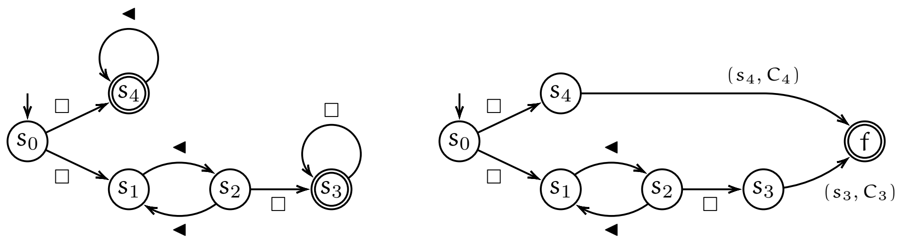


Fig. 2.12. An UPA in normal form and its prefix automaton

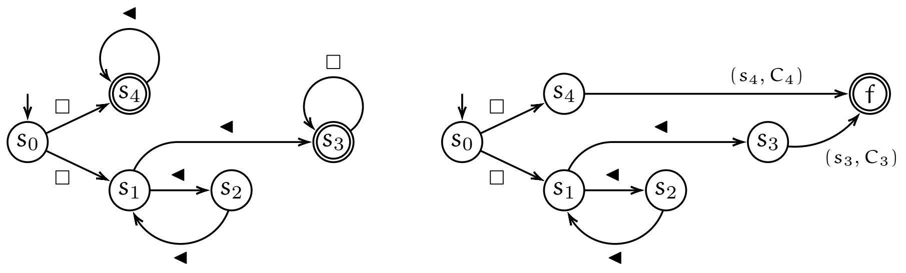


Fig. 2.13. Equivalence of UPA does not transfer to their prefix automata

state f together with its entering transitions. This basically means that the NFA $\mathcal{A}_{pre}$ is nothing but an alternative representation of $\mathcal{A}$.

For the sake of brevity, given two states r, s of $\mathcal{A}$ and a finite word u, we write $r \xrightarrow{u} s$ whenever there exists a run of $\mathcal{A}$ on u that starts with r and ends with s. Similarly, we write $r \overset{u}{-\!\circledcirc\!\!\rightarrow} s$ (resp., $r \overset{u}{-\!\otimes\!\!\rightarrow} s$) whenever there exists a run of $\mathcal{A}$ on u that starts with r, ends with s, and traverses at least one final state of $\mathcal{A}$ (resp., no final states of $\mathcal{A}$). It is easy to see that the prefix automaton $\mathcal{A}_{pre}$ recognizes the language

$$\big\{ub \,:\, \exists\, 1 \leqslant i \leqslant k.\ \exists\, s_0 \in \mathcal{I}.\ \exists\, s \in C_i.\ b = (s, C_i),\ s_0 \overset{u}{-\!\otimes\!\!\rightarrow} s\big\},$$

which is called the *prefix language* of $\mathcal{A}$.

The correspondence between UPA (in normal form) and prefix automata does not lift directly to the language level: the prefix languages of two UPA $\mathcal{A}$ and $\mathcal{A}'$ may be different even in the case in which $\mathscr{L}^{\omega}(\mathcal{A}) = \mathscr{L}^{\omega}(\mathcal{A}')$. As an example, Figure 2.13 depicts an UPA $\mathcal{A}'$, which is equivalent to the UPA of Figure 2.12, and its prefix automaton $\mathcal{A}'_{pre}$, which is not equivalent to the prefix automaton $\mathcal{A}_{pre}$ of Figure 2.12. To get rid of such an asymmetry, that is, to guarantee that $\mathscr{L}(\mathcal{A}_{pre}) = \mathscr{L}(\mathcal{A}'_{pre})$ if and only if $\mathscr{L}^{\omega}(\mathcal{A}) = \mathscr{L}^{\omega}(\mathcal{A}')$, we must impose suitable conditions on the structure of the transition relations of $\mathcal{A}$ and $\mathcal{A}'$.

Definition 13. *An UPA $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ is said to be* prefix-friendly *if it satisfies the following conditions:*

(C1) $\mathcal{A}$ is in normal form,

(C2) every final loop C of $\mathcal{A}$ has the minimum number of states (among all loops that encode the same set of repeating patterns),

(C3) $\mathcal{A}$ has the minimum number of final loops (among all equivalent UPA),

(C4) there are no pairs of transitions of the form (q, a, s) and (r, a, s), with $q \in S \setminus \mathcal{F}$ and $r, s \in \mathcal{F}$.

Figure 2.12 and Figure 2.13 respectively show an *prefix-friendly* UPA and a *prefix-friendly* (equivalent) one.

Lemma 7. *Final loops of a prefix-friendly UPA are pairwise non-isomorphic.*

Proof. This trivially follows from the minimality of the number of final loops. □

Lemma 8. *If $\mathcal{A}$ and $\mathcal{A}'$ are equivalent prefix-friendly UPA, then $\mathcal{A}$ and $\mathcal{A}'$ have isomorphic final loops.*

Proof. Let C be a final loop of $\mathcal{A}$. Since all states of $\mathcal{A}$ are reachable from initial states and since $\mathcal{A}$ has the minimum number of final loops, there exists a word $w \in \mathscr{L}^{\omega}(\mathcal{A})$ that features all and only the repeating patterns encoded by C. Moreover, since $\mathcal{A}'$ is equivalent to $\mathcal{A}$, we have $w \in \mathscr{L}^{\omega}(\mathcal{A}')$ and hence there is a final loop C' in $\mathcal{A}'$ that encodes all and only the repeating patterns featured by w. From this, it easily follows that C and C' are bisimilar loops. Finally, since both C and C' have the minimum number of states, it immediately follows that C and C' are isomorphic. □

In virtue of Lemma 7 and Lemma 8, given two equivalent prefix-friendly UPA $\mathcal{A}$ and $\mathcal{A}'$, we can identify the symbols of the alphabet $A_{\mathcal{A}}$ and those of the alphabet $A_{\mathcal{A}'}$. Formally, we say that two symbols $b \in A_{\mathcal{A}}$ and $b' \in A_{\mathcal{A}'}$ *coincide* iff

1. either they both belong to A and $b = b'$,
2. or $b = (s, C)$ and $b' = (s', C')$, where C and C' are isomorphic final loops of $\mathcal{A}$ and $\mathcal{A}'$, respectively, and s is the state of C that corresponds to s' in C' under the unique[4] isomorphism between C and C'.

The above correspondence can be naturally lifted to languages over $A_{\mathcal{A}}$ and $A_{\mathcal{A}'}$: we say that the two prefix automata $\mathcal{A}_{pre}$ and $\mathcal{A}'_{pre}$ are equivalent iff $\mathscr{L}(\mathcal{A}_{pre}) = \mathscr{L}(\mathcal{A}'_{pre})$, that is, for every finite word $u \in \mathscr{L}(\mathcal{A}_{pre})$ (resp., $u' \in \mathscr{L}(\mathcal{A}'_{pre})$), there is a finite word $u' \in \mathscr{L}(\mathcal{A}'_{pre})$ (resp., $u \in \mathscr{L}(\mathcal{A}_{pre})$) such that $|u| = |u'|$ and, for all $1 \leqslant i \leqslant |u|$, the symbols $u(i)$ and $u'(i)$ coincide.

[4] Given two loops C and C' with the minimum number of states, there exists at most one isomorphism between C and C', since otherwise, by transitivity, there would exist a non-trivial endomorphism in C, thus contradicting the minimality of C.

Lemma 9. *Given a prefix-friendly UPA $\mathcal{A}$ over the alphabet A, its prefix automaton $\mathcal{A}_{pre}$ recognizes all and only the finite words of the form ub, with $b \in A_{\mathcal{A}} \setminus A$ and u being the shortest initial pattern of some word $w \in \mathscr{L}^{\omega}(\mathcal{A})$.*

Proof. Let $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$. Suppose that the prefix automaton $\mathcal{A}_{pre}$ accepts a finite word of the form ub, with $u \in A^*$ and $b = (s, C)$. From the definition of prefix automaton, we know that there is a run ρ of $\mathcal{A}$ on u that starts from an initial state, traverses only non-final states, and finally enters the final loop C at state s. Let v be a repeating pattern encoded by C starting from s. Note that $\mathcal{A}$ accepts the ultimately periodic word $w = uv^{\omega}$. We have to show that u is the shortest initial pattern of w, namely, that the last symbol $u(|u|)$ of u differs from the last symbol $v(|v|)$ of v. Let q be the (non-final) state that immediately precedes s inside the run ρ and let r be the (final) state that immediately precedes s inside the loop C. Clearly, we have that $\big(q, u(|u|), s\big) \in \Delta$. Moreover, since v is encoded by C starting from s, we have that t $s \overset{v}{-\!\circledcirc\!\rightarrow} s$. This shows that $\big(r, v(|v|), s\big) \in \Delta$. Finally, since $\mathcal{A}$ satisfies Condition C4 of Definition 13, we obtain $u(|u|) \neq v(|v|)$.

As for the converse direction, let w be an ultimately periodic word accepted by $\mathcal{A}$ and let ρ be a successful run of $\mathcal{A}$ on w. We denote by i the position of the first final state s that occurs in ρ and by j the position of the second occurrence of s in ρ. We then denote by u and v, respectively, the substrings $w[1, i-1]$ and $w[i, j-1]$ of w. By definition of prefix automaton, the sequence $\rho(1)...\rho(i)f$, where f is the global final state of $\mathcal{A}_{pre}$, is a successful run of $\mathcal{A}_{pre}$ on the finite word ub, where $b = (s, C)$ and C is the (unique) final loop of $\mathcal{A}$ that contains s. This shows that ub is accepted by $\mathcal{A}_{pre}$. It remains to prove that u is the shortest initial pattern of w. Let $q = \rho(i-1)$ and $r = \rho(j-1)$. Since, $\mathcal{A}$ satisfies Condition C4 of Definition 13 and, by construction, $q \in S \setminus \mathcal{F}$, $r, s \in \mathcal{F}$, and $(q, u(|u|), s)$, $(r, v(|v|), s) \in \Delta$, we have that $u(|u|) \neq v(|v|)$. This shows that u is the shortest initial pattern of w. □

The following theorem shows that equivalent prefix-friendly UPA have equivalent corresponding prefix automata.

Theorem 3. *Let $\mathcal{A}$ and $\mathcal{A}'$ be two prefix-friendly UPA. We have that*

$$\mathscr{L}^{\omega}(\mathcal{A}) = \mathscr{L}^{\omega}(\mathcal{A}') \quad \textit{iff} \quad \mathscr{L}(\mathcal{A}_{pre}) = \mathscr{L}(\mathcal{A}'_{pre}).$$

Proof. Every ultimately periodic word has a unique shortest initial pattern. Therefore, by Lemma 9, the ω-language recognized by $\mathcal{A}$ (resp., $\mathcal{A}'$) is uniquely determined by the language recognized by $\mathcal{A}_{pre}$ (resp., $\mathcal{A}'_{pre}$) and, vice versa, the language recognized by $\mathcal{A}_{pre}$ (resp., $\mathcal{A}'_{pre}$) is uniquely determined by the ω-language recognized by $\mathcal{A}$ (resp., $\mathcal{A}'$). □

Given an UPA $\mathcal{A}$ in normal form, one can efficiently build an equivalent prefix-friendly UPA $\mathcal{B}$ by applying the following sequence of normalization steps:

i) **Minimize the size of each final loop.** Such an operation collapses all equivalent states in each final loop, thus producing an UPA that satisfies Conditions C1, C2 of Definition 13.

ii) **Minimize the number of final loops.** Such an operation collapses all isomorphic (minimal) final loops, thus producing an UPA that satisfies Conditions C1–C3 of Definition 13.

iii) **Add shortcuts towards final loops.** Such an operation produces an UPA that satisfies all conditions of Definition 13.

The above normalizations steps can be implemented as follows. As a preliminary remark, we note that the minimization procedures for the size and the number of final loops can be viewed as particular cases of the solution to the *single function coarsest partition problem*. We thus refer the reader to [85] for further details and proofs.

Let $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ be an UPA in normal form and let us consider a final loop C of $\mathcal{A}$. Two states s, s' of C are said to be *equivalent* if we have $s \xrightarrow{v} s$ and $s' \xrightarrow{v} s'$, for some finite word v. Minimizing the size of the final loop C amounts to collapse all equivalent states of C. Thus, suppose that C is a final loop of the form

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} s_n \xrightarrow{a_n} s_1$$

and let v be the repeating pattern $a_1a_2...a_n$. One can easily verify that two states s_i and s_j, with $1 \leqslant i \leqslant j \leqslant n$, are equivalent iff the value $j - i \bmod n$ is the offset of an occurrence of v as a substring of vv. Thus, the equivalence class $[s_i]$ of a state s_i of C is given by the set $\{s_j : 1 \leqslant j \leqslant n,\ j \equiv i \pmod{p}\}$, where p denotes the offset of the *first non-trivial* occurrence of v as a substring of vv (note that the value p can be efficiently computed in linear time using Knuth-Morris-Pratt string matching algorithm [56]).

Finally, the operation of collapsing equivalence classes of C into single states can be implemented by first replacing the final loop C with a new final loop $[C]$ of the form

$$[s_1] \xrightarrow{a_1} [s_2] \xrightarrow{a_2} \dots \xrightarrow{a_{p-1}} [s_p] \xrightarrow{a_p} [s_1]$$

and then replacing every transition of $\mathcal{A}$ of the form (q, a, s_i), where $q \notin \mathcal{F}$, by the triple $\big(q, a, [s_i]\big)$. On the ground of the above arguments, it is easy to devise a linear time procedure that minimizes the size of each final loop of a given UPA $\mathcal{A}$ in normal form.

As for the minimization of the number of final loops, this amounts to collapse all isomorphic final loops of $\mathcal{A}$, under the assumption that $\mathcal{A}$ is an UPA that satisfies Conditions C1, C2 of Definition 13. Indeed, two final final loops C and C' of an UPA encode the same set of repeating patters iff they are bisimilar. Moreover, if C and C' have the minimum number of states, then they are bisimilar iff they are isomorphic.

Isomorphic final loops of $\mathcal{A}$ can be efficiently found by (i) defining a total ordering on the alphabet A, (ii) representing each final loop C of $\mathcal{A}$ with the *lexicographically least primitive repeating pattern* ν_{C} encoded by C, and (iii) sorting the loops according to the lexicographic order of their representatives (in this way, isomorphic final loops have the same representatives and hence they appear contiguous in the ordered list).

We recall that, given a final loop C, the lexicographically least primitive repeating pattern ν_{C} of C can be computed in linear time by using the algorithms described in [4, 96]. Moreover, sorting the loops according to the lexicographic order of their representatives can be done in linear time using the well-known radix-sort algorithm. This shows that the minimization of the number of final loops of a given UPA can be done in linear time.

The last normalization step consists of the removal of redundant transitions of $\mathcal{A}$ and in the addition of shortcuts towards their target states. Let $\mathcal{A} = (\mathsf{A}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F})$ be an UPA that satisfies Conditions C1–C3 of Definition 13. We say that a transition $(\mathsf{q}, \mathsf{a}, \mathsf{s})$ of $\mathcal{A}$ is *redundant* with respect to another transition $(\mathsf{r}, \mathsf{a}', \mathsf{s}')$, if $\mathsf{q} \in \mathsf{S} \setminus \mathcal{F}$, $\mathsf{r} \in \mathcal{F}$, $\mathsf{s} = \mathsf{s}' \in \mathcal{F}$, and $\mathsf{a} = \mathsf{a}'$ (notice that the UPA that satisfy Condition C4 of Definition 13 are exactly those UPA which contain no redundant transitions). The addition of shortcuts and the removal of redundant transitions are implemented in two phases as follows.

The first phase iteratively performs the following steps: (i) select two transition of $\mathcal{A}$ of the form $(\mathsf{q}, \mathsf{a}, \mathsf{s})$ and $(\mathsf{r}, \mathsf{a}, \mathsf{s})$, with $(\mathsf{q}, \mathsf{a}, \mathsf{s})$ being redundant with respect to $(\mathsf{r}, \mathsf{a}, \mathsf{s})$, (ii) mark the transition $(\mathsf{q}, \mathsf{a}, \mathsf{s})$ so that it cannot be selected again, (iii) add a new transition $(\mathsf{t}, \mathsf{b}, \mathsf{r})$ (if it does not exist already) for each existing transition $(\mathsf{t}, \mathsf{b}, \mathsf{q})$, with $\mathsf{t} \in \mathsf{S} \setminus \mathcal{F}$, and (iv) mark r as a new initial state whenever q is an initial state.

The second phase starts when there are no more redundant (unmarked) transitions and it consists of the removal of previously-marked redundant transitions. Notice that it is necessary to postpone the removal of the redundant transitions to this second phase, since, otherwise, the algorithm may enter an infinite loop where a single transition is alternatively added to and removed from the automaton.

One can easily verify that the resulting automaton is a prefix-friendly UPA equivalent to the input automaton $\mathcal{A}$. In particular, the following invariant holds during the first phase: for every successful run on an infinite word w, there is a successful run on w that uses only unmarked transitions. Moreover, the whole process can be implemented by a procedure which takes quadratic time in the size of the input automaton $\mathcal{A}$.

Proposition 20. *Given an UPA $\mathcal{A}$ in normal form, one can compute in time $\mathcal{O}(|\mathcal{A}|^2)$ an equivalent prefix-friendly UPA $\mathcal{B}$. Moreover, $|\mathcal{B}|$ is at most quadratic in $|\mathcal{A}|$ and the number of states of $\mathcal{B}$ is less than or equal to the number of states of $\mathcal{A}$.*

Proof. By applying the proposed sequence of normalization steps to a given UPA $\mathcal{A}$ in normal form, one obtains an equivalent prefix-friendly UPA $\mathcal{B}$. All normalization steps, but the last one, can be computed by suitable linear-time algorithms and the intermediate results are UPA with size (resp., number of states) less than or equal to the size (resp., number of states) of the original UPA $\mathcal{A}$. As for the last normalization step, the resulting prefix-friendly UPA $\mathcal{B}$ is obtained from an intermediate UPA $\mathcal{A}'$ by simply adding new transitions and removing redundant ones. This shows that the size of $\mathcal{B}$ is at most quadratic in the size of the original UPA $\mathcal{A}$ and, similarly, that the number of states of $\mathcal{B}$ is less than or equal to the number of states of $\mathcal{A}$. □

The Canonical Form for UPA

We conclude the section by introducing a canonical form for UPA, , that is, a representation of rational ω-language of ultimately periodic words which turns out to be unique up to isomorphisms.

Definition 14. *An UPA $\mathcal{A} = (\mathbb{A}, \mathbb{S}, \Delta, \mathcal{I}, \mathcal{F})$ is said to be in* canonical form *if $\mathcal{A}$ is prefix-friendly and, in addition, the prefix automaton $\mathcal{A}_{pre}$ of $\mathcal{A}$ is a deterministic finite-state automaton (DFA for short) having the minimum number of states (among all equivalent DFA).*

As a matter of fact, an UPA in canonical form may be exponentially larger than an equivalent UPA in prefix-friendly form (this basically follows from the fact that DFA may be exponentially larger than equivalent NFA [50]).

The following theorem shows that the canonical form of an UPA is unique, up to isomorphisms, among all equivalent UPA.

Theorem 4. *Let $\mathcal{A}$ and $\mathcal{A}'$ be two UPA in canonical form. We have that $\mathcal{A}$ and $\mathcal{A}'$ are equivalent iff they are isomorphic.*

Proof. By Theorem 3, the prefix-friendly UPA $\mathcal{A}$ and $\mathcal{A}'$ are equivalent iff the corresponding prefix automata $\mathcal{A}_{pre}$ and $\mathcal{A}'_{pre}$ are equivalent. Moroever, since $\mathcal{A}$ and $\mathcal{A}'$ are in canonical form, $\mathcal{A}_{pre}$ and $\mathcal{A}'_{pre}$ are DFA having the minimum number of states. Hence they are equivalent iff they are isomorphic. Finally, since the UPA $\mathcal{A}$ (resp., $\mathcal{A}'$) is uniquely determined by its prefix automaton $\mathcal{A}_{pre}$ (resp., $\mathcal{A}'_{pre}$), we can conclude that $\mathcal{A}$ and $\mathcal{A}'$ are equivalent iff they are isomorphic. □

Let us show now how to compute the canonical form of a given prefix-friendly UPA $\mathcal{A}$. As a preliminary remark, observe that any transformation of an UPA $\mathcal{A}$ that preserves the corresponding prefix language $\mathscr{L}(\mathcal{A}_{pre})$ results in an UPA $\mathcal{A}'$ which has the same structure of $\mathcal{A}$ with respect to the transitions towards final states (intuitively, such a property follows from the fact that the final loops of $\mathcal{A}$ are "hidden" inside the alphabet of the prefix automaton). In particular, it follows that $\mathcal{A}'$ is prefix-friendly whenever $\mathcal{A}$ is prefix-friendly. Given such a property, it becomes clear that the canonical form of a prefix-friendly UPA $\mathcal{A}$ can be obtained by applying the following sequence of transformations:

i) **Compute the prefix automaton.** Such an operation produces an NFA that implicitly represents the original prefix-friendly UPA.
ii) **Compute the minimal equivalent DFA.** Such an operation produces a DFA that has the minimum number of states and that recognizes the prefix language of the original UPA.
iii) **Convert the minimal DFA back to an UPA.** Such an operation eventually produces an UPA in canonical form.

Note that all transformations, but the second one, can be computed at low cost. On the other hand, the second transformation is the most demanding one from the computational point of view, since, in the general case, deterministic prefix automata may be exponentially larger than equivalent nondeterministic prefix ones. From experimental comparisons [9], it turns out that Brzozowski's algorithm [6] is the most efficient solution to the problem of determinizing and minimizing prefix automata.

For the sake of completeness, we provide a short description of Brzozowski's algorithm. Given a generic NFA $\mathcal{A}'$, the algorithm first reverses the transitions of $\mathcal{A}'$ (we denote such an operation by *Rev*), then it performs a subset construction to build a DFA equivalent to the reversed copy of $\mathcal{A}'$ (we denote such an operation by *Det*), and finally it iterates such two operations once more. It can be proved that $Det(Rev(Det(Rev(\mathcal{A}'))))$ is a DFA equivalent to the NFA $\mathcal{A}'$ having the minimum number of states among all equivalent DFA. Moreover, such a construction requires, in the worst case, $\mathcal{O}(2^n)$ time and space, where n is the number of states of the input automaton $\mathcal{A}'$.

Proposition 21. *Given an UPA $\mathcal{A}$ in normal form, one can compute in time $\mathcal{O}(2^{|\mathcal{A}|})$ an equivalent UPA $\mathcal{B}$ in canonical form. Moreover, the size of $\mathcal{B}$ is, in the worst case, a simple exponential in the size of $\mathcal{A}$.*

Proof. Let $\mathcal{A}$ be an UPA in normal form. By exploiting Proposition 20, one can compute an equivalent prefix-friendly UPA $\mathcal{A}'$ whose number of states is less than or equal to the number of states of $\mathcal{A}$. Then, by applying the above described sequence of normalization steps, one can transform $\mathcal{A}'$ into an equivalent UPA $\mathcal{B}$ in canonical form. As for the complexity of the whole procedure, recall that $\mathcal{A}'$ can be computed from $\mathcal{A}$ in quadratic time. Moreover, computing the prefix automaton of $\mathcal{A}'$ (and, vice-versa, computing the UPA which corresponds to any given prefix automaton), can be done in linear time. Finally, since Brzozowski's algorithm takes simple exponential time with respect to the number of states of the input automaton, computing the minimal DFA equivalent to the prefix automaton of $\mathcal{A}'$ requires time $\mathcal{O}(2^{|\mathcal{A}'|}) = \mathcal{O}(2^{|\mathcal{A}|})$. $\square$

2.4.3 Algorithms on UPA

UPA can be successfully exploited to solve a number of classical problems about sets of ultimately periodic words. We will focus our attention on the following basic problems:

- **Emptiness problem.** It consists of deciding whether a given UPA $\mathcal{A}$ recognizes the empty language.
- **Acceptance problem.** It consists of deciding whether a given UPA $\mathcal{A}$ recognizes a given ultimately periodic word w, represented as a pair (u, v) consisting of a finite prefix and a finite non-empty repeating pattern.
- **Equivalence problem.** Given two UPA $\mathcal{A}$ and $\mathcal{B}$, it consists of deciding whether $\mathscr{L}^{\omega}(\mathcal{A}) = \mathscr{L}^{\omega}(\mathcal{B})$.
- **Inclusion problem.** Given two UPA $\mathcal{A}$ and $\mathcal{B}$, it consists of deciding whether $\mathscr{L}^{\omega}(\mathcal{A}) \subseteq \mathscr{L}^{\omega}(\mathcal{B})$.
- **State optimization problem.** Given an UPA $\mathcal{A}$ in a specific normalized form, it consists of building an equivalent UPA $\mathcal{B}$ having the smallest number of states among all equivalent UPA in that normalized form.

As UPA can be viewed both as a restricted class of Büchi automata and as an extension of NFA, we will compare the structure and the complexity of the proposed algorithms with those of both of them.

The Emptiness and Acceptance Problems

In the case of a Büchi automaton, the emptiness problem is solved by (i) searching for a path departing from an initial state and reaching a final state and (ii) searching for a loop that contains such a final state. Since every final state of an UPA in normal form belongs to a final loop, the emptiness problem for UPA in normal form reduces to the problem of searching for a path from an initial state to a final state, as it happens with NFA. Thus, the emptiness problem can be solved in linear time $\mathcal{O}(|\mathcal{A}|)$.

As for the acceptance problem, there is a straightforward algorithm, which exploits basic closure properties, that decides whether a given UPA $\mathcal{A}$ accepts a given ultimately periodic word $w = uv^{\omega}$ in time $\mathcal{O}(|\mathcal{A}|(|u| + |v|))$. The problem of checking whether $w = uv^{\omega}$ belongs to the ω-language recognized by a given UPA $\mathcal{A}$ is indeed equivalent to the problem of testing the (non-)emptiness for the ω-language $\mathscr{L}^{\omega}(\mathcal{A}) \cap \mathscr{L}^{\omega}(\mathcal{B})$, where $\mathcal{B}$ is an UPA that recognizes the singleton $\{uv^{\omega}\}$. A (slightly) more efficient solution, which takes time $\mathcal{O}(|\mathcal{A}||u| + |v|)$, takes advantage of prefix automata. Given an UPA $\mathcal{A}$ and an ultimately periodic word $w = uv^{\omega}$, one can decide whether $w \in \mathscr{L}^{\omega}(\mathcal{A})$ by performing the following steps:

1. compute the prefix automaton $\mathcal{A}_{pre}$ of $\mathcal{A}$;
2. replace in $\mathcal{A}_{pre}$ every transition of the form (s, b, f), with $b = (s, C) \in A_{\mathcal{A}} \setminus A$, by a transition of the form (s, x, f), where x is either the symbol $\top$ or the symbol $\bot$, depending on whether C encodes the repeating pattern v starting from s or not;
3. decide whether the resulting NFA accepts the word $u\top$.

Note that the above steps, but the last one, can be performed in linear time with respect to the size of $\mathcal{A}$, u, and v, while the last step can be performed in time linear in $|\mathcal{A}||u|$.

The Equivalence and Inclusion Problems

It is well known that the equivalence and inclusion problems, for any class of automata which is closed under intersection, are inter-reducible. As an example, given two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, we have $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{B})$ iff $\mathscr{L}^\omega(\mathcal{A}) \subseteq \mathscr{L}^\omega(\mathcal{B}) \ \wedge \ \mathscr{L}^\omega(\mathcal{B}) \subseteq \mathscr{L}^\omega(\mathcal{A})$ and, similarly, $\mathscr{L}^\omega(\mathcal{A}) \subseteq \mathscr{L}^\omega(\mathcal{B})$ iff $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{A}) \cap \mathscr{L}^\omega(\mathcal{B})$. Moreover, if the class of automata is also closed under complementation, then both problems can be reduced to the emptiness problem. As an example, given two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, we have $\mathscr{L}^\omega(\mathcal{A}) \subseteq \mathscr{L}^\omega(\mathcal{B})$ iff $\mathscr{L}^\omega(\mathcal{A}) \cap \mathscr{L}^\omega(\mathcal{C}) = \emptyset$, where $\mathcal{C}$ is the Büchi automaton recognizing the complement language of $\mathscr{L}^\omega(\mathcal{B})$.

In [100] an *implicit* construction of a complement Büchi automaton has been given, which allows one to solve the equivalence problem for Büchi automata in polynomial space. Such a construction is based on the ability to encode each state and checking each transition of the complement automaton by using only a polynomial amount of space. Since, in the worst case, the number of states of the complement automaton is $\Omega(2^{n \log n})$, where n is the number of states of the input automaton (see [58, 63] for lower-bound results and [78, 92, 93] for constructions that match these bounds), it turns out that that any deterministic or non-deterministic algorithm based on an explicit or implicit construction of a complement automaton must use $\Omega(n \log n)$ space.

As for NFA, both the equivalence and inclusion problems are proved to be PSPACE-complete [50]. Standard algorithms solving these problems are based on either explicit or implicit constructions of equivalent deterministic finite-state automata and they use either simple exponential time $\Theta(2^n)$ or linear space $\Theta(n)$, where n is the number of states of the input NFA. As an example, the inclusion problem for two NFA $\mathcal{A}$ and $\mathcal{B}$ can be solved by guessing a finite word u which is a witness for the non-inclusion of $\mathscr{L}(A)$ in $\mathscr{L}(\mathcal{B})$, namely, such that $u \in \mathscr{L}(\mathcal{A})$ and $u \in \mathscr{L}(\mathcal{C})$, where $\mathcal{C}$ is the DFA recognizing the complement of $\mathscr{L}(\mathcal{B})$. Verifying that $u \in \mathscr{L}(\mathcal{C})$ can be done directly on the NFA $\mathcal{B}$ by first computing the set of states of $\mathcal{B}$ that are reachable from an initial state by reading u and then verifying that such a set does not contain any final state. The described algorithm thus requires linear space with respect to the size of the input NFA $\mathcal{A}$ and $\mathcal{B}$.

Since UPA-recognizable languages have equivalent representations in terms of prefix automata, by exploiting existing algorithms for NFA we can devise suitable algorithms for the equivalence and inclusion problems of UPA of the same complexity. As a preliminary result, we provide a polynomial lower bound to the space complexity of the equivalence and inclusion problems for UPA.

Proposition 22. *The equivalence problem and inclusion problem for UPA are PSPACE-hard.*

Proof. We provide a reduction from the equivalence problem for NFA, which is known to be PSPACE-hard [50], to the equivalence problem for UPA. Let $\mathcal{A}$ and $\mathcal{B}$ be two NFA recognizing the languages $\mathscr{L}(\mathcal{A})$ and $\mathscr{L}(\mathcal{B})$, respectively. We extend the input alphabet with a new symbol # and we transform the NFA $\mathcal{A}$ (resp., $\mathcal{B}$) into an UPA $\mathcal{A}'$ (resp., $\mathcal{B}'$) that recognizes the ω-language $\mathscr{L}(\mathcal{A})\{\#\}^\omega$ (resp., $\mathscr{L}(\mathcal{A})\{\#\}^\omega$) as follows:

i) we add a new state f that becomes the unique final state of $\mathcal{A}'$ (resp., $\mathcal{B}'$),
ii) we add a new transition of the form $(\mathsf{f}, \#, \mathsf{f})$,
iii) for each final state s in $\mathcal{A}$ (resp., $\mathcal{B}$), we add a new transition $(\mathsf{s}, \#, \mathsf{f})$.

Clearly, we have $\mathscr{L}^\omega(\mathcal{A}') = \mathscr{L}^\omega(\mathcal{B}')$ iff $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\mathcal{B})$. This allows us to conclude that the equivalence problem and the inclusion problem for UPA are PSPACE-hard. □

Let us now provide optimal algorithms that solve the equivalence and inclusion problems for UPA.

The first solution to the equivalence and inclusion problems for UPA stems from the fact that the canonical form of an UPA is unique, up to isomorphisms, among all equivalent UPA. Thus, the problem of deciding whether two given UPA $\mathcal{A}$ and $\mathcal{B}$ recognize the same ω-language can be reduced to the problem of testing whether the canonical forms $\mathcal{A}'$ and $\mathcal{B}'$ of $\mathcal{A}$ and $\mathcal{B}$, respectively, are isomorphic. By Theorem 4, we know that the canonical form of an UPA is computable in exponential time. Moreover, since canonical forms of UPA are, basically, deterministic labeled graphs (with the only exception of the transitions entering the final loops), one can easily decide by a linear time procedure whether the canonical UPA $\mathcal{A}'$ and $\mathcal{B}'$ are isomorphic. This allows us to conclude that the equivalence problem for two generic UPA $\mathcal{A}$ and $\mathcal{B}$ can be decided by a deterministic procedure that requires *exponential time* in the size of the input UPA $\mathcal{A}$ and $\mathcal{B}$.

As for the inclusion problem, one can exploit the fact that, given two UPA $\mathcal{A}'$ and $\mathcal{B}'$ in canonical form, $\mathcal{O}(|\mathcal{A}'| + |\mathcal{B}'|)$ time suffices to compute an UPA $\mathcal{C}'$ in canonical form that recognizes the intersection language $\mathscr{L}^\omega(\mathcal{A}') \cap \mathscr{L}^\omega(\mathcal{B}')$. This yields a straightforward procedure that, given two UPA $\mathcal{A}$ and $\mathcal{B}$, decides in exponential time whether $\mathscr{L}^\omega(\mathcal{A}) \subseteq \mathscr{L}^\omega(\mathcal{B})$: such a procedure first computes the canonical forms $\mathcal{A}'$ and $\mathcal{B}'$ of $\mathcal{A}$ and $\mathcal{B}$, respectively, then it computes the UPA $\mathcal{C}'$ in canonical form recognizing $\mathscr{L}^\omega(\mathcal{A}') \cap \mathscr{L}^\omega(\mathcal{B}')$, and finally it decides whether $\mathscr{L}^\omega(\mathcal{A}') = \mathscr{L}^\omega(\mathcal{C}')$.

It is worth pointing out that the proposed deterministic solutions to the equivalence and inclusion problems for UPA outperform classical solutions for Büchi automata, which are based on the construction of complement languages, and their time complexity is comparable to the time complexity of analogous algorithms working on NFA.

We now describe alternative non-deterministic algorithms that solve the equivalence problem and the inclusion problem for UPA using at most a linear amount of space. These are modified versions of standard algorithms working on NFA, which, basically, exploit an implicit subset construction to decide the (non-)inclusion problem for two given rational languages. Similar constructions have been also proposed in [65, 59].

Let $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ and $\mathcal{B} = (A, S', \Delta', \mathcal{I}', \mathcal{F}')$ be two generic UPA. By Theorem 3, one can apply the standard non-inclusion algorithm for NFA directly to the prefix automata corresponding to prefix-friendly UPA $\mathcal{A}'$ and $\mathcal{B}'$ obtained from $\mathcal{A}$ and $\mathcal{B}$, respectively. However, in such a case, the worst-case space complexity of the resulting algorithm is quadratic with respect to the size of the input UPA $\mathcal{A}$ and $\mathcal{B}$ (recall that prefix-friendly UPA $\mathcal{A}'$ and $\mathcal{B}'$ may have quadratic size with respect to the original UPA $\mathcal{A}$ and $\mathcal{B}$). Here, we describe a modified version of the non-inclusion algorithm that directly works on UPA and that requires linear space with respect to their size.

Without loss of generality, we can assume that the two input UPA $\mathcal{A}$ and $\mathcal{B}$ are in normal form (by Proposition 19, this does not imply any blowup of the size). The proposed algorithm exploits non-determinism to guess an ultimately periodic word w that belongs to $\mathscr{L}^{\omega}(\mathcal{A})$ and then verifies that w does not belong to $\mathscr{L}^{\omega}(\mathcal{B})$, thus certifying that $\mathscr{L}^{\omega}(\mathcal{A}) \not\subseteq \mathscr{L}^{\omega}(\mathcal{B})$. Since any run of $\mathcal{A}$ that accepts an ultimately periodic word w is eventually trapped inside a simple final loop, we can distinguish two phases of the algorithm, the first one dealing with a prefix of the run of $\mathcal{A}$ that reaches the final loop, the second one dealing with the (unique) suffix encoded by the final loop[5].

During the first phase, a finite prefix $s_0 s_1 ... s_n$ of a run of $\mathcal{A}$ and a word $a_1 ... a_n$ recognized by it are guessed. At the same time, the sets $S'_0, S'_1, ..., S'_n$ of states of $\mathcal{B}$ which are visited while reading the word $a_1 ... a_n$ are computed. Configurations are thus described by pairs of the form (s_i, S'_i), with $s_i \in S$ and $S'_i \subseteq S'$. The algorithm starts with an initial configuration of the form (s_0, S'_0), where s_0 is an initial state of $\mathcal{A}$ and S'_0 is the set of initial states of $\mathcal{B}$. At each step, the algorithm chooses a transition of $\mathcal{A}$ of the form (s_i, a_{i+1}, s_{i+1}), with a_{i+1} being a symbol from A and s_{i+1} being a state of $\mathcal{A}$, and it then computes the next configuration (s_{i+1}, S'_{i+1}), where S'_{i+1} is defined as the set of all states r' of $\mathcal{B}$ such that there exists $s' \in S'_i$, with (s', a_{i+1}, r') being a valid transition of $\mathcal{B}$. If, at some step n, s_n turns out to be a final state, then we know that $\mathcal{A}$ recognizes an ultimately periodic word with prefix $a_1 ... a_n$. At this point, the algorithm switches to the second phase. Note that, even though the first phase can be carried on for arbitrarily many steps (this is the case, for instance, when a non-final loop of $\mathcal{A}$ is reached), we can limit the number of iterations during this phase to $|S| 2^{|S'|}$: indeed, if $n \geqslant |S| 2^{|S'|}$ steps were performed during the first phase, then, by the Pigeonhole Principle, there would exist two indices n', n'', with $n' < n'' \leqslant |S| 2^{|S'|}$, such that $(s_{n'}, S'_{n'}) = (s_{n''}, S'_{n''})$ and hence

[5] It is worth remarking that such a technique cannot be extended to generic Büchi automata, since their runs may visit distinct final loops infinitely often.

the word $a_1 ... a_{n'} a_{n''+1} ... a_n$ would be the prefix of an alternative witness for the non-inclusion.

During the second phase, the computation proceeds in a deterministic way on the basis of the (unique) infinite periodic word $b_1 b_2 ...$ which is recognized by $\mathcal{A}$ starting from the last visited (final) state s_n. Configurations are again described by pairs of the form (q_i, Q'_i), with $q_i \in S$ and $Q'_i \subseteq S'$, and they obey to the following constraints:

- (q_0, Q'_0) coincides with the last configuration (s_n, S'_n) computed during the first phase;
- (b_{i+1}, q_{i+1}) is the unique pair such that (q_i, b_{i+1}, q_{i+1}) is a valid transition of $\mathcal{A}$;
- Q'_{i+1} is the set of all states r' of $\mathcal{B}$ for which there exists $q' \in Q'_i$ such that (q', b_{i+1}, r') is a valid transition of $\mathcal{B}$.

By a simple application of the Pigeonhole Principle, we have that there exist two indices m, m', with $m < m' \leqslant |S| 2^{|S'|}$, such that $(q_m, Q'_m) = (q_{m'}, Q'_{m'})$. Hence, if Q'_m contains no final state, then we know that the ultimately periodic word

$$(a_1 ... a_n)(b_1 ... b_m)(b_{m+1} ... b_{m'})^\omega$$

is recognized by $\mathcal{A}$, but not by $\mathcal{B}$, thus certifying that $\mathscr{L}^\omega(\mathcal{A}) \nsubseteq \mathscr{L}^\omega(\mathcal{B})$. Otherwise, if Q'_m contains at least one final state, then the computation is discarded. It is worth pointing out that we do not need to exhibit the candidate ultimately periodic word $(a_1 ... a_n)(b_1 ... b_m)(b_{m+1} ... b_{m'})^\omega$ in $\mathscr{L}^\omega(\mathcal{A}) \setminus \mathscr{L}^\omega(\mathcal{B})$, and thus we do not need to keep track of its symbols during the computation.

The described non-deterministic algorithm solves the non-inclusion (and hence the non-equivalence) problem by using *linear space* in the size of the input UPA (precisely, it requires $\mathcal{O}(\log |S| + |S'|)$ space to store a configuration of the form (s_i, S'_i), with $s_i \in S$ and $S'_i \subseteq S'$, and the value of a counter i ranging over $\{0, ..., |S| 2^{|S'|}\}$). The space complexity of the proposed algorithm is comparable with that of classical non-deterministic algorithms working on NFA and it is strictly lower than that of classical non-deterministic algorithms working on Büchi automata.

Finally, we recall that, by exploiting Savitch's Theorem [86], the above non-deterministic procedure can be turned into a deterministic one that solves the inclusion (and hence the equivalence) problem for UPA by using at most quadratic space.

The State Optimization Problem

Let us consider now the minimization problem for UPA. We first prove that, similarly to the case of NFA (see [52, 53, 62]), the state optimization problem for UPA is PSPACE-complete and it may yield different (non-isomorphic) solutions. Then, we show that standard minimization algorithms for NFA can be *directly* exploited to minimize the number of states of any UPA in normal form. These algorithms can be applied to generic UPA as well; however, in such a case the resulting UPA are, in general, approximations of state optimal ones.

To start with, we provide a sufficient condition for UPA to be state optimal among all equivalent UPA in normal form.

Proposition 23. *Let $\mathcal{A}$ be an UPA in prefix-friendly form and let $\mathcal{A}_{pre}$ be its prefix automaton. If $\mathcal{A}_{pre}$ has the minimum number of states among all equivalent NFA, then $\mathcal{A}$ has the minimum number of states among all equivalent UPA in normal form.*

Proof. For any given automaton (UPA or NFA) $\mathcal{A}$, let us denote by $n(\mathcal{A})$ the number of its states. Assume $\mathcal{A}$ to be a prefix-friendly UPA, whose prefix automaton $\mathcal{A}_{pre}$ has the minimum number of states among all equivalent NFA. By definition of prefix automaton, we have that $n(\mathcal{A}_{pre}) = n(\mathcal{A}) + 1$. Let us now consider a generic UPA $\mathcal{B}$ in normal form equivalent to $\mathcal{A}$. By Proposition 20, there is a prefix-friendly UPA $\mathcal{B}'$ equivalent to $\mathcal{B}$ such that $n(\mathcal{B}') \leqslant n(\mathcal{B})$. Moreover, since $\mathcal{B}'$ is prefix-friendly, by Theorem 3 we have that the prefix automaton $\mathcal{B}'_{pre}$ is equivalent to $\mathcal{A}_{pre}$. From the minimality of $\mathcal{A}_{pre}$, we obtain $n(\mathcal{A}_{pre}) \leqslant n(\mathcal{B}'_{pre})$. Moreover, by construction, $n(\mathcal{B}'_{pre}) = n(\mathcal{B}')+1$. Summing up, we have

$$n(\mathcal{A}) = n(\mathcal{A}_{pre}) - 1 \leqslant n(\mathcal{B}'_{pre}) - 1 = n(\mathcal{B}') \leqslant n(\mathcal{B})$$

which proves that $\mathcal{A}$ has the minimum number of states among all equivalent UPA in normal form. □

By exploiting Proposition 23, we can devise a PSPACE solution to the state optimization problem for UPA in normal form. Such a solution exploits an auxiliary procedure that minimizes the number of states of prefix automata (see [53, 62] for implementation details). Since the state optimization problem for NFA is known to be PSPACE-complete [52, 53, 62] and since (by an argument similar to that of Proposition 22) it is inter-reducible with the state optimization problem for UPA in normal form, we can conclude that the latter problem is PSPACE-complete as well.

If we do not restrict to the class of UPA in normal form, then the number of states of UPA can be further reduced. Below, we show that some final loops of UPA may be safely removed, under the proviso that another 'subsuming' non-final loop takes their role.

Definition 15. *Let $\mathcal{A} = (A, S, \Delta, \mathcal{I}, \mathcal{F})$ be an UPA, C a final loop, and C' a simple non-final loop. We say that C is* subsumed *by C' if C has no exiting transitions and there is a surjective function $f : C' \rightarrow C$ that satisfies the following two conditions:*

- *for every pair of states r, s in C', $(r, a, s) \in \Delta$ iff $(f(r), a, f(s)) \in \Delta$ (intuitively, C and C' are bisimilar),*

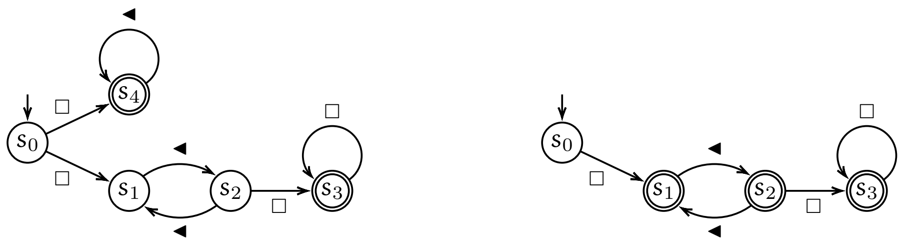


Fig. 2.14. An example of subsumed final loop

- *for every state* r *neither in* C *nor in* C' *and every state* s *in* C*,* $(\mathsf{r}, \mathsf{a}, \mathsf{s}) \in \Delta$ *iff there is a state* s' *in* C' *such that* $\mathsf{s} = \mathsf{f}(\mathsf{s}')$ *and* $(\mathsf{r}, \mathsf{a}, \mathsf{s}') \in \Delta$ *(intuitively, the loop* C' *augmented with its entering transitions simulates the loop* C *augmented with its entering transitions).*

If C is a final loop of $\mathcal{A}$ which is subsumed by a simple non-final loop C', then we can obtain an equivalent UPA $\mathcal{B}$ with fewer states by (i) removing the loop C and (ii) letting C' be a final loop in $\mathcal{B}$. As an example, the right-hand side UPA in Figure 2.14 is obtained from the left-hand side one by removing the final loop on state s_4 and by letting the subsuming loop on states s_1 and s_2 be final.

Proposition 24. *Given an UPA* $\mathcal{A} = (\mathsf{A}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F})$*, a final loop* C*, and a simple non-final loop* C' *that subsumes* C*, the automaton* $\mathcal{B} = (\mathsf{A}, \mathsf{S} \setminus \mathsf{C}, \Delta, \mathcal{I}, (\mathcal{F} \cup \mathsf{C}') \setminus \mathsf{C})$ *is an UPA equivalent to* $\mathcal{A}$*.*

Proof. Since every final state s of $\mathcal{B}$ is contained either in C' (and thus the strongly connected component of s in $\mathcal{B}$ is a simple loop) or in $\mathcal{F} \setminus \mathsf{C}$ (and thus the strongly connected component of s in $\mathcal{B}$ is a single transient state or a simple final loop), it immediately follows that $\mathcal{B}$ is an UPA. To complete the proof, it suffices to show that $\mathcal{B}$ is equivalent to $\mathcal{A}$. Let f be the surjective function from C' to C of Definition 15. We first prove that $\mathscr{L}^{\omega}(\mathcal{A}) \subseteq \mathscr{L}^{\omega}(\mathcal{B})$. Let w an ultimately periodic word in $\mathscr{L}^{\omega}(\mathcal{A})$ and let ρ be a successful run of $\mathcal{A}$ on w. By Definition 15 the subsumed loop C has no exiting transition and thus either ρ does not contain any state in C or, all, but finitely many, states in ρ belong to C. In the former case, it immediately follows that ρ is also a successful run of $\mathscr{L}^{\omega}(\mathcal{B})$ on w. In the latter case, we can obtain a successful run ρ' of $\mathcal{B}$ on w by simply replacing each state s of C with a suitable state s' such that $\mathsf{f}(\mathsf{s}') = \mathsf{s}$ (given the properties of the function f, ρ' respects the transitions of $\mathcal{B}$ on w). This shows that w is accepted by $\mathcal{B}$. As for the converse inclusion, let w be an ultimately periodic word accepted by $\mathcal{B}$ and let ρ' be a successful run of $\mathcal{B}$ on w. By definition of UPA, the set of all states that occur infinitely often in ρ' is either disjoint from C' or it coincides with the set of states of C'. In the former case, ρ' is a successful run of $\mathcal{A}$ on w as well and thus w is accepted by $\mathcal{A}$; in the latter case, we can obtain a successful

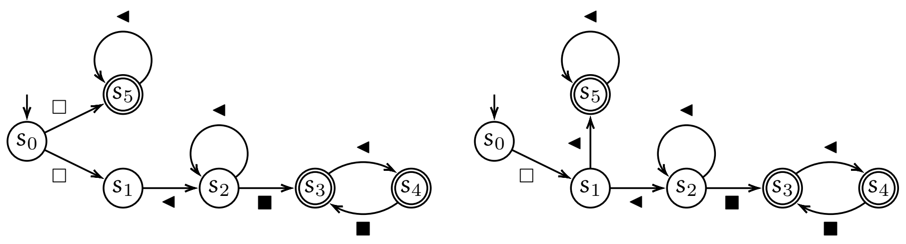


Fig. 2.15. Two equivalent UPA in normal form

run ρ of $\mathcal{A}$ on w by simply replacing each state s that occur infinitely often in ρ' with the state $f(s)$. $\square$

On the basis of Proposition 24, we can devise an algorithm that uses polynomial space to remove all subsumed final loops from a given UPA $\mathcal{A}$ in normal form. Clearly, the removal of all subsumed final loops of $\mathcal{A}$ (if any) results in an equivalent UPA $\mathcal{B}$ with fewer states. Moreover, since all final loops of $\mathcal{A}$ are disjoint, the number of states of the resulting UPA $\mathcal{B}$ does not depend on the removal order of the subsumed final loops.

It is worth pointing out that, in general, the UPA $\mathcal{B}$ resulting from the removal of all subsumed final loops of $\mathcal{A}$ is not guaranteed to be state optimal among all equivalent UPA, even in the case in which $\mathcal{A}$ is a state optimal UPA in normal form. As an example, consider the two equivalent UPA in normal form depicted in Figure 2.15. The left-hand side automaton has no subsumed final loops, while the right-hand side automaton has a final loop (the one on state s_5) which is subsumed by a non-final loop (the one on state s_2). The optimization algorithm has no effect on the left-hand side UPA, while it reduces the number of states of the equivalent right-end side UPA. In general, to compute an UPA with the minimum number of states among all equivalent UPA one must resort to costly algorithms based on trace equivalence. These algorithms can be obtained as generalizations of standard minimization algorithms for NFA [53, 62].

2.4.4 Applications to Time Granularity

The algorithms for solving the emptiness, acceptance, equivalence, inclusion, and state optimization problems, which have been described in the previous section, can be directly exploited to reason on possibly infinite sets of time granularities, e.g., the algorithm that solves the equivalence problem for UPA can be used to check whether or not two given automaton-based representations define the same set of granularities. In the following, we show that UPA turn out to be useful also in solving another basic problem for time granularity, namely, the comparison of pairs of sets of time granularities with respect to various standard relations, e.g., partition, group, sub-granularity, aligned refinement.

In its most common formulation, the granularity comparison problem is viewed as the problem of deciding whether a designated relation $\sim$ holds between two granularities $G \in \mathcal{G}$ and $H \in \mathcal{H}$, where $\mathcal{G}$ and $\mathcal{H}$ are two given sets of granularities. According to such a definition, the granularity comparison problem is actually a family of problems, whose different concrete instances are obtained by specifying the relation that must hold between the pairs of granularities that belong to the two sets $\mathcal{G}$ and $\mathcal{H}$.

Within the framework of automaton-based representations, a granularity comparison problem is reducible to the problem of deciding, given two UPA $\mathcal{A}$ and $\mathcal{B}$ representing two sets of ultimately periodic granularities, whether there exist some words $w_G \in \mathscr{L}^\omega(\mathcal{A})$ and $w_H \in \mathscr{L}^\omega(\mathcal{B})$ that satisfy a certain relation $\sim$, which basically capture the designated relation between time granularities.

To start with, we provide string-based characterizations of standard relations between time granularities, that is, group into, refine, partition, and aligned refinement (note that, under the restriction to ultimately periodic granularities, the grouping relation and the periodic grouping relation actually coincides). We recall that, given an infinite word w over the alphabet $A = \{\blacksquare, \square, \blacktriangleleft\}$ and given a symbol $a \in A$, $|w|_a$ denotes the number of occurrences (possibly ω) of a in w. According to Definition 4, given an infinite word w that represents a time granularity G and given two natural numbers $t, x \in \mathbb{N}$, x is the index of the (unique) granule of G that contains t iff $w[t+1] \in \{\blacksquare, \blacktriangleleft\}$ and $|w[1,t]|_{\blacktriangleleft} = x$.

Proposition 25. *Let* u *and* v *be two infinite words over the alphabet* $\{\blacksquare, \square, \blacktriangleleft\}$ *that represent, respectively, two granularities* G *and* H. *We have that*

- G *groups into* H *iff for every* $t \in \mathbb{N}$,
  i) $v(t+1) = \blacksquare$ *implies* $u(t+1) \in \{\blacksquare, \blacktriangleleft\}$,
  ii) $v(t+1) = \blacktriangleleft$ *implies* $u(t+1) = \blacktriangleleft$,
  iii) $v(t+1) = \square$ *implies either* $u(t+1) = \square$ *or* $v(t'+1) = \square$ *for all* $t' \in \mathbb{N}$ *such that* $|u[1,t]|_{\blacktriangleleft} = |u[1,t']|_{\blacktriangleleft}$;
- G *refines* H *iff for every* $t \in \mathbb{N}$,
  i) $u(t+1) = \blacksquare$ *implies* $v(t+1) = \blacksquare$,
  ii) $u(t+1) = \blacktriangleleft$ *implies* $v(t+1) \in \{\blacksquare, \blacktriangleleft\}$;
- G *partitions* H *iff for every* $t \in \mathbb{N}$,
  i) $u(t+1) = \square$ *if and only if* $v(t+1) = \square$,
  ii) $u(t+1) = \blacksquare$ *implies* $v(t+1) = \blacksquare$;
- G *is an aligned refinement of* H *iff for every* $t \in \mathbb{N}$,
  i) $u(t+1) = \blacksquare$ *implies* $v(t+1) = \blacksquare$,
  ii) $u(t+1) = \blacktriangleleft$ *implies* $v(t+1) \in \{\blacksquare, \blacktriangleleft\}$;
  iii) $u(t+1) = \blacktriangleleft$ *implies* $|u[1,t]|_{\blacktriangleleft} = |v[1,t]|_{\blacktriangleleft}$.

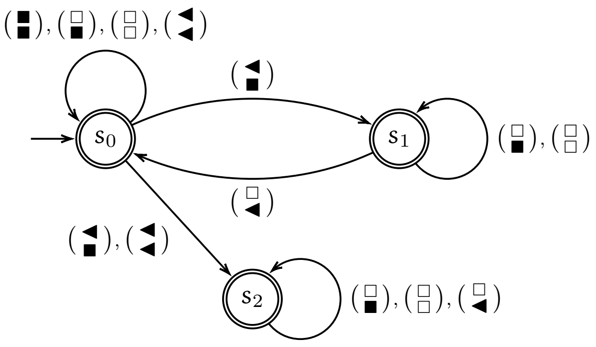


Fig. 2.16. The automaton for the aligned refinement relation

Proof. We only prove the characterization for the alignment refinement relation. Suppose that G is an aligned refinement of H. By definition, every time point $t \in \mathbb{N}$ which is covered by G is covered by H as well. Thus, for every $t \in \mathbb{N}$, $u(t+1) \in \{\blacksquare, \blacktriangleleft\}$ implies $v(t+1) \in \{\blacksquare, \blacktriangleleft\}$. Moreover, it cannot happen that $u(t+1) = \blacksquare$ and $v(t+1) = \blacktriangleleft$ because, otherwise, the granule of G that contains the time point t would not be included in a granule of H. Finally, for every $t \in \mathbb{N}$, if $u(t+1) = \blacktriangleleft$, then t is the last time point of the granule $\mathsf{G}(x)$, where $x = |u[1,t]|_{\blacktriangleleft}$. Since $\mathsf{G}(x) \subseteq \mathsf{H}(x)$, we have that $|v[1,t]|_{\blacktriangleleft} = x$. The converse implication follows by a symmetric argument. □

Let $\sim$ be one of the standard relations between time granularities. By exploiting Proposition 25, it is immediate to see that the ω-language $\mathsf{L}_\sim$ of all pairs of infinite words (including non-periodic ones) that represent pairs of time granularities G, H such that $\mathsf{G} \sim \mathsf{H}$ is Büchi-recognizable. In particular, one can build a Büchi automaton $\mathcal{C}_\sim$ over the alphabet $\{\blacksquare, \square, \blacktriangleleft\} \times \{\blacksquare, \square, \blacktriangleleft\}$ that recognizes the language $\mathsf{L}_\sim$. As an example, Figure 2.16 depicts the Büchi automaton $\mathcal{C}_\sim$, where $\sim$ is the aligned refinement relation.

Moreover, given two UPA $\mathcal{A}$ and $\mathcal{B}$ representing some sets $\mathcal{G}$ and $\mathcal{H}$ of time granularities, one can build a suitable product automaton (an UPA) that recognizes the ω-language $\mathsf{L} = \mathscr{L}^\omega(\mathcal{A}) \times \mathscr{L}^\omega(\mathcal{B})$, which basically represents all pairs (G, H) of time granularities, with $\mathsf{G} \in \mathcal{G}$ and $\mathsf{H} \in \mathcal{H}$. Thus, one can solve the comparison problem for the designated relation $\sim$ by simply testing the emptiness of the intersection language $\mathsf{L} \cap \mathsf{L}_\sim$, which is known to be UPA-recognizable. Note that, since the Büchi automaton $\mathcal{C}_\sim$ is fixed with the designated relation $\sim$, the complexity of the resulting algorithm is $\mathcal{O}(|\mathcal{A}||\mathcal{B}|)$.

We conclude this section by showing how a concrete problem, taken from the medical domain of heart transplant patients, can be addressed and efficiently solved by exploiting the notion of time granularity and the properties of UPA-recognizable ω-languages.

Table 2.1. A hypothetical schedule for therapies/check-ups

`patientId`	`date` (MM/DD/YYYY)	`treatment`
1001	02/10/2003	transplant
1001	04/26/2003	GFR
1002	06/07/2003	GFR
1001	06/08/2003	biopsy
1001	02/10/2004	GFR
1001	01/11/2005	GFR
1001	01/29/2006	GFR

Posttransplantation guidelines require outpatients to take drugs and to submit to periodical visits for life. These requirements are usually collected in formal protocols with schedules specifying the therapies and the frequency of the check-ups. We report an excerpt of the guidelines for a heart transplant patient in [61]. Depending on the physical conditions of the patient, the guidelines can require, together with other treatments, an estimation of the glomerular filtration rate (GFR) with one of the following schedules:

- 3 months and 12 months posttransplantation and every year thereafter;
- 3 months and 12 months posttransplantation and every 2 years thereafter.

These protocols involve the so-called unanchored granularities, to manage the various admissible starting points for the scheduled therapies (and/or check-ups), as well as sets of granularities with different repeating patterns, to capture the set of distinct periodicities of the scheduled therapies. In particular, since different protocols can be specified for the same class of patients by different people/institutions, it is a crucial problem to decide whether two protocols define the same set of therapies/granularities (equivalence problem). Solving this problem makes it possible to choose the most compact, or the most suitable, representation for a given protocol.

Another meaningful reasoning task is the so-called consistency-checking problem, namely, the problem of checking whether a given therapy assigned to a patient satisfies the prescribed protocol. As an example, consider the (sub)set of therapies/check-ups of the above protocol for heart transplant patients and the instance of the temporal relation `Visits`(`patientId`, `date`, `treatment`), represented in Table 2.1. Given a representation of the single granularity G for the specific therapy (up to a certain date) of a certain patient and given a representation of the set of (periodic) time granularities for the prescribed therapies/check-ups, the consistency-checking problem can be decided by testing whether granularity G properly relates to some granularity in $\mathcal{H}$. The consistency-checking problem can thus be viewed as a particular case of granularity comparison problem. Below, we show how such a problem can be effectively solved by means of UPA.

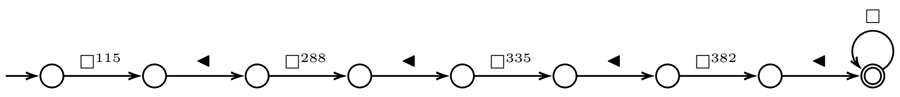


Fig. 2.17. An UPA representing GFR measurements for a patient

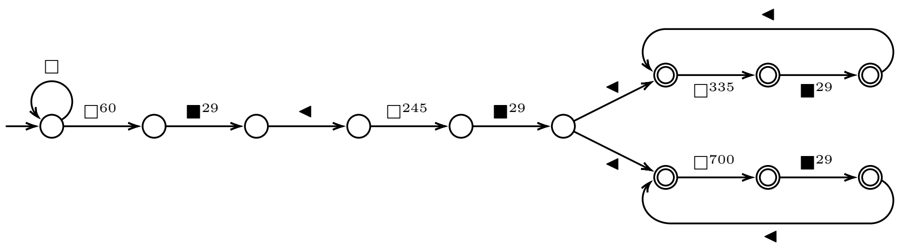


Fig. 2.18. An UPA-based specification of the protocol

For the sake of simplicity, we consider months of 30 days and years of 365 days (relaxing such a simplification is tedious, but trivial). By properly selecting records in Table 2.1, we can build the granularity G of GFR measurements for the patient identified by `patientId` 1001. We represent such a granularity as a single ultimately periodic word w (starting from 01/01/2003), in which the occurrences of $\blacktriangleleft$ denote the days of the visits. The UPA $\mathcal{A}$ that accepts w is depicted in Figure 2.17, where we use the shorthand $\circ\xrightarrow{a^n}\circ$ to denote a sequence of $n+1$ states and n a-labeled transitions.

The set $\mathcal{H}$ of periodic time granularities that encode the set of valid therapies of the protocol is represented by the UPA $\mathcal{B}$ depicted in Figure 2.18.

Checking the consistency of GFR measurements for patient 1001 with respect to the prescribed protocol amounts to check whether granularity G is an aligned refinement of some granularity in $\mathcal{H}$. We can solve the latter problem by first building the UPA $\mathcal{D} = (\mathcal{A} \times \mathcal{B}) \cap \mathcal{C}_{\sim}$, where $\mathcal{C}_{\sim}$ is the Büchi automaton depicted in Figure 2.16, and then checking whether the recognized ω-language $\mathscr{L}^{\omega}(\mathcal{D})$ is non-empty. In our case, we easily see that $\mathcal{D}$ recognizes the (singleton) ω-language

$$\Big\{ \tbinom{\square}{\square}^{100}\tbinom{\square}{\blacksquare}^{15}\tbinom{\blacktriangleleft}{\blacksquare}\tbinom{\square}{\blacksquare}^{13}\tbinom{\square}{\blacktriangleleft}\tbinom{\square}{\square}^{245}\tbinom{\square}{\blacksquare}^{29}\tbinom{\blacktriangleleft}{\blacktriangleleft}\tbinom{\square}{\square}^{335}\tbinom{\blacktriangleleft}{\blacksquare}\tbinom{\square}{\blacksquare}^{28}\tbinom{\square}{\blacktriangleleft}\cdot$$
$$\cdot\tbinom{\square}{\square}^{335}\tbinom{\square}{\blacksquare}^{18}\tbinom{\blacktriangleleft}{\blacksquare}\tbinom{\square}{\blacksquare}^{10}\tbinom{\square}{\blacktriangleleft}\Big(\tbinom{\square}{\square}^{335}\tbinom{\square}{\blacksquare}^{29}\tbinom{\square}{\blacktriangleleft}\Big)^{\omega} \Big\}.$$

Since the resulting language is non-empty, we can conclude that G is an aligned refinement of some granularity in $\mathcal{H}$ and thus the considered therapy satisfies the prescribed protocol.

2.5 Discussion

In this chapter, we considered the problem of representing and reasoning on time granularities by means of automaton-based formalisms.

In Section 2.2, we introduced the string-based and the automaton-based approaches to time granularity. These two approaches are closely related one to each other and subsume previous formalisms proposed in the literature (e.g., collection expressions, slice expressions, Calendar Algebra). Moreover, both approaches ease access to and manipulation of data associated with different time granularities, allowing one to solve some basic problems, such as the equivalence and the granule conversion ones, which have been neglected by many existing formalisms.

The algorithmic nature of string-based and automaton-based representations of time granularities suggests an alternative point of view on their role: strings and automata can be used not only as a formalism for the direct specification of time granularities, but also (and mainly) as a low-level formalism into which high-level time granularity specifications (e.g., Calendar Algebra expressions) can be mapped. However, algorithms may potentially process every element (symbol) of string-based and automaton-based representations, independently from their redundancy, thus requiring a large amount of computational time. In the automaton-based framework, this efficiency problem can be dealt with by equipping automata with counters. Such an extension makes the structure of the automata more compact, and it allows one to efficiently deal with those granularities which have a quasi-periodic structure.

In Section 2.3, we explored in full detail the possibility of improving automaton-based representations of time granularities by means of counters. We showed that regularities of common temporal structures can be succinctly modeled within the class of Nested Counter Single-String Automata (NCSSA). Besides making the structure of the automata more compact, the use of counters, paired with suitable restrictions on the activation rules for the transition functions, yields efficient algorithms for manipulating time granularity specifications. As an example, we described some polynomial-time procedures that compute granule conversions between different time granularities and we showed that the equivalence problem for NCSSA is in co-NP (it is currently unknown whether such a problem can be solved by a deterministic polynomial-time algorithm).

We also dealt with optimization problems for NCSSA-based representations of time granularities in a systematic way. These problems, which are extremely relevant from a computational point of view, have often been overlooked in the literature. We started by showing that the algorithmic complexity of crucial algorithms working on NCSSA is closely related to a suitable measure of automaton complexity, which takes into account the nesting structure of the transition functions. Then, we focused on the problem of minimizing NCSSA with respect to either such a complexity measure or the traditional measure that only considers the number of states of the automaton. By exploiting dynamic programming, we gave polynomial-time algorithms that respectively compute complexity optimal and state optimal NCSSA from a given string-based specification of a time granularity. While the former computes automata which are complexity optimal with respect to the whole class of NCSSA, the latter confines itself to the restricted class of decomposable NCSSA. We believe it possible to further improve these algorithms by exploiting subtle relationships between repeating patterns of strings and secondary transition functions of optimal NCSSA. As a matter of fact, we conjecture that the loops determined by the transition functions of a complexity optimal NCSSA can be related to the *maximal repetitions* in the recognized word (a maximal repetition of w is a periodic substring $w[i,j]$ whose minimum period increases as soon as $w[i,j]$ is prolonged to the right, e.g., $w[i,j+1]$, or to the left, e.g., $w[i-1,j]$). Moreover, we are exploring the possibility of generalizing the state optimization algorithm in order to produce optimal automata with respect to the whole class of NCSSA. We believe that such a task could be accomplished by introducing a suitable operator, in addition to AppendChar and AppendRepeat, which collapses *indistinguishable* states of NCSSA.

In Section 2.4, we developed a theory of rational ω-languages that consist of ultimately periodic words only and we provided it with an automaton-based characterization. Furthermore, we showed how well-known results coming from automata theory can be exploited to solve a number of basic problems about rational ω-languages of ultimately periodic words. In particular, we provided effective solutions to the problems of emptiness, membership, equivalence, inclusion, and size-optimization. We also showed that the proposed framework allows one to represent and reason about sets of time granularities that feature a possibly infinite number of different initial patterns, but only a finite number of non-equivalent repeating patterns.

As it has been shown in Section 2.4.2, UPA-recognizable languages, which are a proper subclass of Büchi-recognizable ones, are not closed under complementation. In fact, for every UPA-recognizable language L, the complement $\bar{L}$ of L is not UPA-recognizable. The relationships among UPA-recognizable languages, their complements, and Büchi-recognizable languages can be summarized as follows. Let us call *co-UPA-recognizable* language the complement $\bar{L}$, with respect to the set U_Σ of all ultimately periodic words, of any UPA-recognizable language L. Any co-UPA-recognizable language features all, but finitely many, non-equivalent repeating patterns and, as pointed out in the proof of Proposition 15, it is not Büchi-recognizable. Moreover, for every pair of UPA-recognizable languages L_1 and L_2, the language $L_1 \cap \bar{L}_2$ $(= L_1 \cap (U_\Sigma \setminus L_2) = L_1 \cap (\Sigma^\omega \setminus L_2))$ is UPA-recognizable as well. Analogously, the union of an UPA-recognizable language with a co-UPA-recognizable language is a co-UPA-recognizable language. This basically means that the class of languages that contains all UPA-recognizable languages and all their complements is the smallest class that includes all regular ω-languages of ultimately periodic words and is closed under finite unions, finite intersections, and complementation with respect to U_Σ. Moreover, it is straightforward to generalize the solutions to the emptiness, membership, equivalence, and inclusion problems for UPA to solutions of analogous problems for co-UPA.

It is worth pointing out that there exist interesting connections between UPA-recognizable languages and other subclasses of regular ω-languages. In particular, UPA can be viewed as a special forms of non-deterministic co-Büchi (or safety) automata. Moreover, non-deterministic co-Büchi automata are known to be equivalent to their deterministic versions [65]. It thus follows that the languages recognized by UPA can be also recognized by a proper subclass of deterministic co-Büchi automata.

A natural development of the above results is the definition of a high-level logical language, e.g., a variant of propositional Linear Temporal Logic [36], which allows one to represent all (co-)UPA-recognizable languages by means of suitable formulas. Besides its theoretical relevance, pairing such a logical framework with the proposed automaton-based one would allow one to use the former as a high-level interface for the specification of granularities and the latter as an internal formalism for efficiently reasoning about them.

As a concluding remark, we would like to emphasize that most constructs for time granularity are typically one-dimensional. A possible extension may concern moving from linear structures to branching ones, which are more suited to deal with (infinite) hierarchies of differently grained time granularities. As an example, the problem of representing and reasoning on such a kind of layered temporal structures is systematically dealt with in [66, 67, 71] by means of the notions of n-layered structure, downward-unbounded layered structure, and upward-unbounded layered structure. At the end of Chapter 3 (Section 3.5.4), we shall come back to this subject by briefly reviewing some definitions, expressiveness results, and decidability results for layered temporal structures. Then, we shall introduce a more general notion of layered temporal structure, which subsumes the previous definitions, and we shall apply the decidability results outlined in previous sections to this kind of temporal structures.

3

Tree Automata and Logics

The nature of this chapter is more theoretical in that we deal with decision procedures for a very expressive logic, called monadic second-order logic. More precisely, we focus our attention on the model checking problem for monadic second-order formulas interpreted over deterministic vertex-colored trees. Basically, such a problem consists of establishing, by an effective procedure, whether a fixed tree-shaped structure (e.g., the infinite complete binary tree) satisfies a given property, expressed via suitable a monadic second-order formula.

In the area of verification of infinite-state systems, monadic second-order logic has been extensively used as a specification language, because it is powerful enough to express relevant properties of graph structures such as reachability, planarity, vertex k-colorability (for any fixed k), and confluency properties [22] and it subsumes, besides first-order logic, many propositional (temporal) modal logics, in particular the propositional modal μ-calculus [51]. Unfortunately, its model checking problem turns out to be highly undecidable for many relational structures. In this respect, two important exceptions are the theorems of Büchi [2] and Rabin [88], which prove the decidability of the model checking problem for monadic second-order formulas interpreted over the linear order $(\mathbb{N}, <)$ and over the infinite complete binary tree $(\{0, 1\}^*, succ_0, succ_1)$, respectively. Both decidability results have been obtained by reducing the model checking problem to the problem of testing the emptiness of the language recognized by a suitable automaton, precisely, a Büchi (word) automaton in the case of Büchi's theorem and a Rabin tree automaton in the case of Rabin's theorem [107].

Büchi's and Rabin's results have been also exploited to decide the model checking problem for monadic second-order formulas interpreted over linear and branching structures expanded with unary predicates. As an example, the problem of establishing whether a given monadic second-order formula φ holds over an expanded linear structure of the form $(\mathbb{N}, <, P)$, with $P \subseteq \mathbb{N}$, is reduced to the acceptance problem of $(\mathbb{N}, <, P)$, namely, the problem of establishing whether a Büchi automaton $\mathcal{A}_\varphi$ accepts (the infinite word that characterizes)

$(\mathbb{N}, <, \mathsf{P})$. Elgot and Rabin [35] gave a positive answer to the decidability of the acceptance problem for various meaningful predicates $\mathsf{P} \subseteq \mathbb{N}$. Intuitively, their technique consists of defining a transformation, called *contraction*, of a given infinite word w into another infinite word $\vec{w}$ and a transformation of a Büchi automaton $\mathcal{A}$ into another Büchi automaton $\vec{\mathcal{A}}$ in such a way that $\mathcal{A}$ accepts w iff $\vec{\mathcal{A}}$ accepts $\vec{w}$. If $\vec{w}$ happens to be ultimately periodic, then the latter condition can be easily checked, thus providing an effective solution to the original problem. In [97, 12] Elgot-Rabin contraction method has been generalized in several respects to cover, for instance, all morphic predicates and the so-called profinitely ultimately periodic words. As a matter of fact, in [90], the class of profinitely ultimately periodic words has been proved to capture exactly all the expanded linear structres that enjoy a decidabile the model checking problem.

In this chapter, we show that the contraction method can be generalized to deal with expanded tree structures. Here the role of ultimately periodic words is taken by regular colored trees and the notion of factorization of infinite words is accordingly generalized to branching structures.

We first show that the model checking problem for monadic second-order formulas interpreted over deterministic vertex-colored tree structures can be reduced to the acceptance problem for (alternating Muller) tree automata, that is, to the problem of establishing whether a given tree automaton accepts a designated relational structure, viewed as a deterministic vertex-colored tree. Such a problem is trivially decidable in the case of regular trees. Then, by exploiting a suitable notion of tree equivalence with respect to alternating Muller tree automata, we reduce a number of instances of the acceptance problem for deterministic vertex-colored trees to the easy case of regular trees.

We show that the proposed technique works effectively for a large class of trees (most of them non-regular), which we call reducible trees. Moreover, we prove that the class of reducible trees is closed with respect to several natural tree transformations (e.g., transductions with rational lookahead and unfoldings with backward edges and loops) and it contains all deterministic trees in the Caucal hierarchy as well as meaningful relational structures outside it (a short review of Caucal hierarchy and its properties is given in Section 3.1.4). These results, besides showing the robustness of the notion of reducible tree, provide a natural framework to reason on tree transformations and to easily transfer decidability results.

It is also worth mentioning that the proposed contraction method for tree automata presents some similarities with Shelah's composition method [95], which directly exploits a notion of indistiguishability of relational structures with respect to monadic second-order formulas. However, unlike the contraction method for tree automata, Shelah's composition method finds it difficult to manage the different possible valuations of a given variable over distinct copies of the same factor (a formal notion of factor will be given in Section 3.2.3). This is a problem whenever one needs to reduce a branching structure to a linear one (some examples of this situation are given in Section 3.2.4).

In the last part of this chapter, we consider the model checking problem for monadic second-order logic (and its chain fragment) over the so-called layered temporal structures, which are tree-shaped structures well-suited for modeling and reasoning on temporal relationships at different 'grain levels'. We introduce a new notion of layered temporal structure, which subsumes previous definitions found in the literature, and we apply our technique to solve the model checking problem over such a kind of structure.

The chapter is organized as follows. In Section 3.1, we introduce basic notation and terminology about graphs, trees, alternating Muller tree automata, monadic second-order logic, and its model checking problem. In Section 3.2 we present the contraction method for tree automata in full detail. In Section 3.3, we introduce a set of natural tree transformations and we systematically explore their relationships. In Section 3.4, we show that the contraction method works effectively for a large class of complex tree structures, which we call reducible trees. Furthermore, we prove that the class of reducible trees is closed under several tree transformations. In Section 3.5, we discuss meaningful applications of the contraction method. In particular, we show that the class of reducible trees contains all deterministic trees in the Caucal hierarchy, we provide a characterization of the languages recognized by the so-called two-way alternating Muller tree automata, we prove the decidability of the acceptance problem for morphic trees, and, finally, we study the decidability of the model checking problem for monadic second-order logic (and its chain fragment) interpreted over layered temporal structures. Section 3.6 provides an assessment of the achieved results. Some lengthy proofs have been included for the readers convenience in the appendix, but they can be safely skipped.

3.1 Background Knowledge

In this section, we recall some basic definitions and results about graphs, trees, alternating Muller tree automata, monadic second-order logic, and its model checking problem.

3.1.1 Graphs and Trees

We use the term *label* (resp., *color*) to identify a symbol associated with an edge (resp., a vertex) of a graph. The set of labels (resp., colors) is usually denoted by A (resp., C). An A-*labeled* (directed simple) graph is a tuple $G = (V, (E_a)_{a \in A})$, where V (also denoted $\mathcal{D}om(G)$) is a countable set of vertices and $(E_a)_{a \in A}$ are binary relations defining graph edges and their labels. An *expanded graph* is a graph equipped with some unary predicates, namely, a structure $(G, \bar{P})$, where G is a graph and $\bar{P} = (P_1, ..., P_m)$, with $P_i \subseteq \mathcal{D}om(G)$ for all $1 \leqslant i \leqslant m$. Any expanded graph $(G, \bar{P})$ is canonically represented by a

C-*colored graph* $\mathsf{G}_{\bar{\mathsf{p}}} = (\mathsf{G}, \Omega)$, where $\mathsf{C} = \mathscr{P}(\{1, ..., m\})$ and $\Omega : \mathcal{D}om(\mathsf{G}) \to \mathsf{C}$ is a coloring function mapping a vertex $v \in \mathcal{D}om(\mathsf{G})$ to the set of all indices $1 \leqslant i \leqslant m$ such that $v \in \mathsf{P}_i$ (such a set is called the color of v).

We focus our attention on (unranked) rooted trees, namely, graphs such that for every vertex v, there exists a unique path, called *access path*, from a designated source vertex, called *root*, to v. In order to simplify the description of some compositional properties of trees, we shall distinguish between the set C of colors for the internal vertices of a tree and the set D of colors for its leaves. Precisely, we say that T is a C-*colored* D-*augmented tree* if the colors of internal vertices range over C and the colors of leaves range over D (D may be different, but not necessarily disjoint, from C).

We identify each vertex of a tree (resp., a deterministic tree) with its access path (resp., with the sequence of labels in its access path). In particular, we view a deterministic A-labeled C-colored D-augmented tree T as a partial function from a *prefix-closed* language L over the alphabet A to the set $\mathsf{C} \cup \mathsf{D}$. Accordingly, we denote by va the a-successor of v in T and by $\mathsf{T}(v)$ the color of the vertex v of T. Sometimes, if there is a well-understood ordering on the set A of edge labels, we can use (unranked) terms to denote trees; for instance, if $\mathsf{A} = \{a_1 < a_2 < a_3\}$, then $\emptyset$ denotes the empty tree and $c\langle d, d, \emptyset\rangle$ denotes the ternary tree consisting of a c-colored root and two d-colored leaves, which are target nodes of two edges labeled respectively by a_1 and a_2.

A tree T is said to be *regular* if it contains only finitely many non-isomorphic subtrees. It is easy to see that any regular (colored) tree is bisimilar to a finite (colored) graph (this basically implies that any regular tree can be viewed as the 'unraveling' of a finite graph from a designated source vertex). A deterministic tree T is said to be *full* if all its vertices have 0 or k children, with $k = |\mathsf{A}|$, namely, if for every $v \in \mathcal{D}om(\mathsf{T})$ and every $a, a' \in \mathsf{A}$, we have that $va \in \mathcal{D}om(\mathsf{T})$ if and only if $va' \in \mathcal{D}om(\mathsf{T})$. A deterministic A-labeled (infinite) tree T is said to be *complete* if (it has ω levels and) each level n contains exactly $|\mathsf{A}|^n$ nodes.

Hereafter, we fix a set A of labels, a set C of colors for the internal vertices, and a set D of colors for the leaves. Unless otherwise stated, from now on, we assume that trees are deterministic, A-labeled, C-colored, and D-augmented.

3.1.2 Tree Automata

The proposed contraction method works for various classes of automata. The easiest way to describe it is by means of alternating tree automata [78]. Such a class of automata generalizes the class of non-deterministic tree automata by allowing the association of more than one state with each successor of a node of the input tree. As an effect, the input tree and the computation tree may look quite different from each other.

Given a finite set A of labels and a finite set S of states, we denote by $\mathcal{B}^+(\mathsf{A} \times \mathsf{S})$ the set of *positive boolean formulas* over the set of propositional variables $\mathsf{A} \times \mathsf{S}$. For instance, both the formula `false` and the formula

$(\langle a_1, s_1\rangle \wedge \langle a_2, s_2\rangle) \vee (\langle a_1, s_2\rangle \wedge \langle a_2, s_2\rangle \wedge \langle a_2, s_3\rangle)$ belong to $\mathcal{B}^+(\{a_1, a_2\}\times \{s_1, s_2, s_3\})$. Below, we define alternating Muller tree automata, which run on *infinite complete* trees.

Definition 16. *An* alternating Muller tree automaton *is a tuple* $\mathcal{A} = (A, C, S, \Delta, \mathcal{I}, \mathcal{F})$*, where*

- S *is a finite set of states,*
- $\Delta : S \times C \rightarrow \mathcal{B}^+(A \times S)$ *is a transition function,*
- $\mathcal{I} \subseteq S$ *is a set of initial states,*
- $\mathcal{F} \subseteq \mathscr{P}(S)$ *is a family of accepting sets.*

Given an A-labeled C-colored *infinite complete* tree T, a *run* of $\mathcal{A}$ on T is an unlabeled $(\mathcal{D}om(T) \times S)$-colored[1] tree R such that

a) the root of R is colored with a pair (ε, s), where ε is the root of T and s is a state from S;

b) for every (v, s)-colored vertex u of R with k $(\geqslant 0)$ successors $u_1, ..., u_k$, there exists a set $M = \{(a_1, s_1), ..., (a_k, s_k)\}$, with $R(u_i) = (v\, a_i,\, s_i)$ for all $1 \leqslant i \leqslant k$, that satisfies the formula $\Delta(s, T(v))$.

According to Muller acceptance condition, a run R is *successful* if it satisfies the following two conditions:

i) the state associated with the root of R is an initial state from $\mathcal{I}$,

ii) for every infinite path π in R, the set of states that occur infinitely often along π, denoted $\mathcal{I}nf(R|\pi)$, is an accepting set from $\mathcal{F}$.

We say that $\mathcal{A}$ accepts T if and only if there exists a successful run of $\mathcal{A}$ on T. The language *recognized* by $\mathcal{A}$ is defined as the set $\mathscr{L}(\mathcal{A})$ of all trees T that are accepted by $\mathcal{A}$. It is worth remarking that alternating (Muller) tree automata are as expressive as non-deterministic (Muller) tree automata. Given an alternating tree automaton $\mathcal{A}$, one can indeed compute an equivalent non-deterministic tree automaton $\mathcal{A}'$ that recognizes the same language [78].

Without loss of generality, we impose some restrictions on the runs of alternating tree automata that allow us to simplify definitions and proofs. First of all, we forbid the use of the tautology `true` in the transition function of an alternating tree automaton $\mathcal{A}$. As a consequence, any run R of $\mathcal{A}$ on an infinite complete tree T contains no leaves. In addition, notice that if $M \subseteq A \times S$ is a model of a positive boolean formula $\varphi \in \mathcal{B}^+(A \times S)$, then every set $M' \supseteq M$ is a model of φ as well. This basically means that a run R of an alternating tree automaton $\mathcal{A}$ can be extended by adding arbitrary successors to any vertex

[1] The colors of the vertices of a run are pairs of the form (v, s), meaning that the automaton $\mathcal{A}$ reads the vertex v of T while being in the control state s. In a run of an alternating tree automaton, any vertex v of T may be associated with different states, while in a run of a non-deterministic tree automaton it is associated with a unique state.

of it, possibly generating a new run R' which includes some redundant paths. We get rid of the redundant paths of a run (if any) by restricting to *minimal* models of positive boolean formulas. Formally, given a run R of an alternating tree automaton $\mathcal{A}$, we rewrite condition b) as follows:

b') for every (v, s)-colored vertex u of R with k (> 0) successors $u_1, ..., u_k$, there exists a minimal set $M = \{(a_1, s_1), ..., (a_k, s_k)\}$, where $R(u_i) = (v\, a_i,\ s_i)$ for all $1 \leqslant i \leqslant k$, that satisfies the formula $\Delta(s, T(v))$.

As already pointed out, such a restriction is safe with respect to the existence of successful runs.

Below, we generalize the notion of alternating Muller tree automaton to allow computations over *incomplete trees*.

Definition 17. *A* D-augmented alternating Muller tree automaton *is a tuple* $\mathcal{A} = (A, C, D, S, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$*, where*

- $A, C, S, \Delta, \mathcal{I}, \mathcal{F}$ *are defined as in Definition 16,*
- D *is the set of leaf colors,*
- $\mathcal{G} \subseteq D \times S$ *specifies the acceptance condition for the leaves of the input tree.*

Given an A-labeled C-colored D-augmented *non-empty full* tree T, a run of $\mathcal{A}$ on T is defined in the standard way, except for the fact that the transitions are now restricted to occur at the internal vertices of T only[2]. More precisely, a run of $\mathcal{A}$ on T is an unlabeled $(\mathcal{D}om(T) \times S)$-colored tree R such that

a) the root of R is colored with a pair (ε, s), where ε is the root of T and s is a state from S;

b") for every internal (v, s)-colored vertex u of R with k (> 0) successors $u_1, ..., u_k$, there exists a *minimal* set $M = \{(a_1, s_1), ..., (a_k, s_k)\}$, with $R(u_i) = (v\, a_i,\ s_i)$ and $v\, a_i \in \mathcal{D}om(T)$ for all $1 \leqslant i \leqslant k$, that satisfies the formula $\Delta(s, T(v))$.

The run R is *successful* if it satisfies the following conditions:

i) the state associated with the root of R is an initial state from $\mathcal{I}$;

ii) for every infinite path π in R, $\mathcal{I}nf(R|\pi) \in \mathcal{F}$;

iii) for every leaf v of R, $\big(T(\downarrow_1 R(v)),\ \downarrow_2 R(v)\big) \in \mathcal{G}$, where $\downarrow_1 R(v)$ and $\downarrow_2 R(v)$ denote, respectively, the first and the second component of the pair $R(v)$.

Notice that every leaf v of R has a unique corresponding leaf u $(= \downarrow_1 R(v))$ in T, while a leaf u of T may have many corresponding leaves in R.

[2] In finite tree automata, transitions are also defined at the leaves of the tree and the acceptance condition refers to the states of the outer frontier. Here, we exclude transitions exiting from leaves to maintain a correspondence between the vertices of the input tree and the vertices of the run of the tree automaton.

3.1.3 Monadic Second-Order Logic

Monadic Second-Order (MSO) logic is the extension of first-order logic with set variables, namely, variables that are going to be interpreted by unary predicates. Let us fix a *signature* (Σ, k), namely, a finite set Σ of relational symbols equipped with a ranking function $k : \Sigma \to \mathbb{N}_{>0}$, where $\mathbb{N}_{>0}$ denotes the set of positive natural numbers.

MSO formulas over the signature (Σ, k) are build up starting from atoms of the form $x = y$, $x \in Y$, $X \subseteq Y$, and $\mathtt{r}(x_1, ..., x_{k(\mathtt{r})})$, where $\mathtt{r}$ is a relational symbol in Σ. Atomic formulas can be combined by means of the boolean connectives $\vee$ and $\neg$ and the existential quantifiers $\exists\, x.$ and $\exists\, X.$ over individual variables (denoted by lowercase letters) and set variables (denoted by uppercase letters). We say that a variable occurring in a formula φ is *free* if it is not bounded by any existential quantifier. We then write $\varphi(x_1, ..., x_n, X_1, ..., X_m)$ to mean that the free variables of φ are only $x_1, ..., x_n, X_1, ..., X_m$. An *MSO sentence* is simply an MSO formula without free variables.

In order to define the semantics of an MSO formula, we briefly review the notion of *relational structure* over the signature (Σ, k): this is any tuple $S = \big(V, (E_\mathtt{r})_{\mathtt{r}\in\Sigma}\big)$, where V is a (possibly infinite) set of elements and, for every relational symbol $\mathtt{r} \in \Sigma$, $E_\mathtt{r}$ is a $k(\mathtt{r})$-ary relation over V. Note that any expanded graph (and hence any colored tree) can be viewed as a relational structure over a signature whose symbols have arity at most 2. As a matter of fact, we shall evaluate MSO formulas mainly over expanded graphs and colored trees (in such a case, the signature is uniquely determined by the set A of edge labels and the set C of vertex colors).

An *assignment* for a tuple $(x_1, ..., x_n, X_1, ..., X_m)$ of free variables is a tuple $(v_1, ..., v_n, P_1, ..., P_m)$, where $v_i \in V$ for all $1 \leqslant i \leqslant n$ and $P_j \subseteq V$ for all $1 \leqslant j \leqslant m$. We say that an MSO formula $\varphi(x_1, ..., x_n, X_1, ..., X_m)$ *holds* in the relational structure $S = \big(V, (E_\mathtt{r})_{\mathtt{r}\in\Sigma}\big)$ under the assignment $(v_1, ..., v_n, P_1, ..., P_m)$, and we shortly write $S \vDash \varphi[v_1, ..., v_n, P_1, ..., P_m]$, if and only if one of the following conditions holds:

- φ is of the form $x_i = x_j$ and $v_i = v_j$;
- φ is of the form $x_i \in X_j$ and $v_i \in P_j$;
- φ is of the form $X_i \subseteq X_j$ and $P_i \subseteq P_j$;
- φ is of the form $\mathtt{r}(x_{i_1}, ..., x_{i_{k(\mathtt{r})}})$ and $(v_{i_1}, ..., v_{i_{k(\mathtt{r})}}) \in E_\mathtt{r}$;
- φ is of the form $\varphi_1(x_1, ..., x_n, X_1, ..., X_m) \vee \varphi_2(x_1, ..., x_n, X_1, ..., X_m)$ and $S \vDash \varphi_1[v_1, ..., v_n, P_1, ..., P_m]$ or $S \vDash \varphi_2[v_1, ..., v_n, P_1, ..., P_m]$;
- φ is of the form $\neg\varphi'(x_1, ..., x_n, X_1, ..., X_m)$ and $S \nvDash \varphi'[v_1, ..., v_n, P_1, ..., P_m]$;
- φ is of the form $\exists\, x_{n+1}.\ \varphi'(x_1, ..., x_{n+1}, X_1, ..., X_m)$ and there exists $v_{n+1} \in V$ such that $S \vDash \varphi'[v_1, ..., v_{n+1}, P_1, ..., P_m]$
- φ is of the form $\exists\, X_{m+1}.\ \ \varphi'(x_1, ..., x_n, X_1, ..., X_{m+1})$ and there exists $P_{m+1} \subseteq V$ such that $S \vDash \varphi'[v_1, ..., v_n, P_1, ..., P_{m+1}]$.

The *MSO theory* of a relational structure $S = (V, (E_r)_{r \in \Sigma})$ is defined as the set of all MSO sentences φ that hold in S.

When writing MSO formulas, we shall often use natural shorthands like $X = Y$ for $X \subseteq Y \wedge Y \subseteq X$, $X = \emptyset$ for $\forall\, Y.\ X \subseteq Y$, etc. Moreover, by a slight abuse of notation, we shall not distinguish anymore between the relational symbols of a signature (Σ, k) and the corresponding relations of a structure S over (Σ, k).

It is worth mentioning some fragments of MSO logic that have been intensively studied in the literature. For instance, when considering expanded graphs structures, it is possible to use the *path fragment* of MSO logic, which is obtained from standard MSO logic by allowing quantifications over paths only. Another interesting fragment of MSO logic is the *chain fragment*, which is obtained by restricting to quantifications over chains (i.e., subsets of paths) only.

Finally, in several definitions and proofs, it is common practice to restrict to an equivalent fragment of MSO logic, which only uses set variables. According to such a simplified fragment, each individual variable x is replaced by a set variable X, which is then restricted, by means of a suitable formula (e.g. $\varphi_{sing}(X) = \forall\, Y.\ (Y = \emptyset \vee X = Y \vee X \subseteq Y)$), to be instantiated by singletons only.

Below, we recall the notion of MSO-definability. Given a relational structure $S = \big(V, (E_r)_{r \in \Sigma}\big)$ and an n-ary relation E over V, we say that E is *MSO-definable* in S if there is an MSO formula $\varphi(x_1, ..., x_n)$ such that, for every $v_1, ..., v_n \in V$,

$$(v_1, ..., v_n) \in E \qquad \text{iff} \qquad S \vDash \varphi[v_1, ..., v_n].$$

As an example, given a relational structure S containing a binary relation E, the reflexive and transitive closure of E, denoted E^*, is MSO-definable in S, precisely by the formula

$$\varphi^*_E(x, y) = \forall\, Z.\ \big(Z(x) \wedge \forall\, z_1, z_2.\ (Z(z_1) \wedge E(z_1, z_2)) \rightarrow Z(z_2)\big) \rightarrow Z(y).$$

Note that, in such a case, the addition of a new symbol E^* to the signature (Σ, k) and the expansion of S with the corresponding relation E^* does not increase the expressive power of the logic (this is because every occurrence of the atomic formula $E^*(x, y)$ can be replaced by the equivalent formula $\varphi_{E^*}(x, y)$).

In the following section, we show how one can exploit the notion of MSO-definability to build new relational structures starting from simple ones.

3.1.4 The Model Checking Problem

The *model checking problem* for a relational structure S is the problem of deciding whether a given MSO sentence φ holds in S. We thus say that the MSO theory of a structure S is recursive (or decidable) if the model checking problem for $\mathcal{S}$ is decidable. Below, we give an overview of some powerful techniques that were proposed in the literature to deal with the model checking problem in an effective way.

The celebrated Rabin's theorem [88] states that the MSO theory of the infinite complete binary tree is decidable. Such a result has been obtained by reducing the model checking problem for MSO formulas interpreted over expanded tree structures to the acceptance problem for a suitable class of tree automata (originally, Rabin tree automata).

The correspondence between MSO formulas and Rabin tree automata (or, equivalently, alternating Muller tree automata) can be formalized as follows: given an MSO formula $\varphi(X_1, ..., X_m)$, one can compute a tree automaton $\mathcal{A}$ (and, conversely, given a tree automaton $\mathcal{A}$, one can compute an MSO formula $\varphi(X_1, ..., X_m)$) such that, for every infinite complete binary tree T and every tuple of predicates $P_1, ..., P_m \subseteq \mathcal{D}om(T)$,

$$T \vDash \varphi[P_1, ..., P_m] \qquad \text{iff} \qquad T_{P_1,...,P_m} \in \mathscr{L}(\mathcal{A})$$

where $T_{P_1,...,P_m}$ denotes the $\{1, ..., m\}$-colored tree obtained from T by associating with each vertex $v \in \mathcal{D}om(T)$ the (unique) color $i \in \{1, ..., m\}$ such that $v \in P_i$.

In virtue of the above correspondence, the model checking problem of an expanded tree structure $(T, P_1, ..., P_m)$ is decidable if and only if the acceptance problem for the corresponding colored tree $T_{P_1,...,P_m}$ is decidable. Moreover, if $m = 0$, namely, if the formula φ has no free variables, then the problem of establishing whether or not φ holds in the infinite complete binary tree is reducible to the emptiness problem for tree automata, which is known to be decidable [107]. It is also worth pointing out that Rabin's theorem can be easily generalized to incomplete non-empty full trees (for instance, by introducing a fresh symbol $\perp$ and assuming that the tree automata read $\perp$ on the missing vertices).

In [76], Muller and Schupp brought the interest to logical theories of graphs by identifying a large class of relational structures that enjoy decidable MSO theories. These structures can be viewed as the configuration graphs of pushdown automata and hence they are called context-free (or end-regular) graphs. A simplified proof of the decidability of the MSO theories of context-free graphs can be found in [108]. Basically, it stems from the fact that every context-free graph can be obtained from the infinite binary complete tree by applying an *MSO-compatible* transformation, namely, a transformation that preserves the decidability of the MSO theories (for a more formal definition of MSO-compatible transformation, we refer the reader to [21]).

In the following, we describe two notable examples of MSO-compatible transformations, namely, MSO-definable interpretations and unfoldings. Subsequently, we introduce a well-known hierarchy of graphs and trees, which is obtained by repeatedly applying MSO-definable interpretations and unfoldings, starting from finite graphs.

MSO-Definable Interpretations and Unfoldings

Let us fix two signatures (Σ, k) and (Σ', k'). An *MSO-definable interpretation* from (Σ, k) to (Σ', k') is a tuple of the form $\mathcal{I} = \big(\psi(x), (\varphi_r(x_1, ..., x_{k'(r)})_{r\in\Sigma'}\big)$, where $\psi(x)$ and $\varphi_r(x_1, ..., x_{k'(r)}$, for every $r \in \Sigma'$ are MSO formulas. Such a tuple $\mathcal{I}$ describes a transformation of any relational structure $S = \big(V, (E_r)_{r\in\Sigma}\big)$, over the signature (Σ, k), into a new relational structure $\mathcal{I}(S) = \big(V', (E'_r)_{r\in\Sigma'}\big)$, over the signature (Σ', k'), where

- the domain V' consists of all and only the elements $v \in V$ such that $S \vDash \psi[v]$;
- for every symbol $r \in \Sigma'$, the relation E_r consists of all and only the $k'(r)$-tuples of elements $v_1, ..., v_{k'(r)} \in \mathcal{D}om(V')$ such that $S \vDash \varphi[v_1, ..., v_{k'(r)}]$.

Given the above definition, it becomes clear that if the relational structure S has a decidable MSO theory, then the relational structure $\mathcal{I}(S)$ has a decidable MSO theory as well (to see this, notice that any formula φ interpreted over $\mathcal{I}(S)$ can be translated into an equi-satisfiable formula φ' over S).

As an example, we show how the context-free graph of Figure 3.1 can be obtained from the infinite complete binary tree via an MSO-definable interpretation. Let (Σ, k) be the signature consisting of two binary relational symbols E_1 and E_2 and let (Σ', k') be the signature consisting of three binary relational symbols E'_a, E'_b, E'_c. Furthermore, let us define the following MSO formulas over the signature (Σ, k):

$$\begin{aligned}
\varphi_{E'_a}(x, y) &= E_1(x, y) \\
\varphi_{E'_b}(x, y) &= E_2(x, y) \\
\varphi_{E'_c}(x, y) &= \exists\, z_1, z_2.\ E_1(z_1, z_2)\ \wedge\ E_2(z_1, y)\ \wedge\ E_2(z_2, x) \\
\psi(x) &= \varphi^*_{E'_a}(\mathit{root}, x)\ \vee\ \exists\, y.\ \big(\varphi^*_{E'_a}(\mathit{root}, y)\ \wedge\ \varphi_{E'_b}(y, x)\big).
\end{aligned}$$

Intuitively, the formulas φ_{E_a}, φ_{E_b}, and φ_{E_c} specify the edge relations of an $\{a, b, c\}$-labeled graph embedded into the infinite complete binary tree, while the formula ψ specifies the domain of such a graph (note that $\varphi^*_{E'_a}$ defines the reflexive and transitive closure of the relation E'_a). It follows that the MSO-definable interpretation $\mathcal{I} = \big(\psi(x), \varphi_a(x, y), \varphi_b(x, y), \varphi_c(x, y)\big)$ maps the infinite complete binary tree to the context-free graph of Figure 3.1.

The operation of unfolding of a graph is defined as follows. Given a (colored) graph G and a designated source vertex v_0 in it, the *unfolding* of G from v_0 is the (colored) tree $\mathcal{U}nf(G, v_0)$ whose domain consists of all and only the finite paths from v_0 to a vertex v in G and where the a-labeled edges are all and only the pairs of paths (π, π') such that π' extends π through an a-labeled edge of G (the color associated with each vertex π of $\mathcal{U}nf(G, v_0)$ is that of the target vertex of π).

It is easy to see that the unfolding of a graph G from a source vertex v_0 is a tree bisimilar to G, provided that every vertex of G is reachable from v_0. Moreover, it can be proved the operation of unfolding is MSO-compatible,

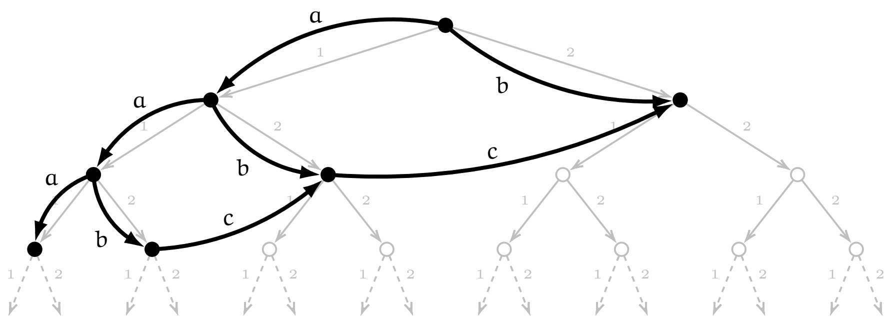


Fig. 3.1. A context-free graph embedded into the infinite binary tree

meaning that the model checking problem for the unfolding $\mathcal{U}nf(\mathsf{G}, \mathsf{v}_0)$ of a graph G is reducible to the model checking problem for the graph G itself. Such a property has been proved through the following series of results. First, in [21] the operation of unfolding restricted to *deterministic* graphs is shown to be MSO-compatible. Subsequently, in [94, 112] MSO-compatibility is proved for a more powerful transformation, called tree iteration. Finally, in [24], the unfolding of a graph G is shown to be MSO-definable in the tree iteration of G, thus proving the MSO-compatibility of the unrestricted unfolding operation.

The Caucal Hierarchy

By alternating MSO-definable interpretations and unfoldings, starting from some basic relational structures, it is possible to generate a number of interesting structures that enjoy decidable MSO-theories. For instance, one can start with finite graphs, which obviously enjoy decidable MSO-theories, and then apply unfoldings to them, thus obtaining (possibly non-deterministic) regular trees of finite out-degree. In succession, one can apply MSO-definable interpretations, thus obtaining the so-called prefix-recognizable graphs [14]. By repeatedly applying the above operations one obtains an infinite hierarchy, known as *Caucal hierarchy* (or pushdown hierarchy) [13]. Since the operations of MSO-definable interpretation and unfolding are MSO-compatible, it is easy to see that every graph/tree in the Caucal hierarchy has a decidable MSO theory. Below, we give a formal definition of the Caucal hierarchy in accordance with [108].

Definition 18. *The* level 0 graphs *of the Caucal hierarchy are all and only the finite rooted graphs. For every* $\mathsf{n} > 0$, *the* level n trees *(resp., the* level n graphs*) of the Caucal hierarchy are obtained by unfolding level* $\mathsf{n} - 1$ *graphs (resp., by applying MSO-definable interpretations to level* $\mathsf{n} - 1$ *trees).*

Equivalent characterizations of the Caucal hierarchy can be found in the literature. Such a hierarchy has been originally introduced in [13] by repeatedly applying inverse rational substitutions, rational restrictions, and unfoldings starting from finite graphs (the former two operations can be viewed as special forms of MSO-definable interpretations and are described in detail in Section 3.3.4. In [11] the same hierarchy is obtained by means of MSO-definable transductions (i.e., generalized versions of MSO-definable interpretations) and treegraph operations (i.e., a variant of the tree iteration). Moreover, always in [11], it is proved that, for every $n > 0$, the level n Caucal graphs coincide with the (ε-closures of) transition graphs of level n higher-order pushdown automata, which can be viewed as generalizations of pushdown automata that use level n stacks (a level 1, or simple, stack is a finite word, a level $n + 1$ stack is a simple stack containing level n stacks) (we refer the reader to [31, 32] for further details about higher-order pushdown automata). Given the above characterizations and the fact that the hierarchy of the languages recognized by higher-order pushdown automata is strictly increasing [30, 39], it follows that the Caucal hierarchy is strictly increasing as well.

Finally, in [13] the family of all deterministic colored (ranked) trees that belong to the Caucal hierarchy is studied. Such a class of tree, called *term hierarchy*, is obtained by tailoring the notions of unfolding, inverse rational substitution, and rational restriction to the case of deterministic colored trees. Equivalent characterizations of the term hierarchy has been also given in terms of safe higher-order recursive schemes [31, 79, 54] and evaluations of first-order substitutions over regular trees [23].

3.2 The Contraction Method for Tree Automata

In this section, we develop a powerful method to decide the acceptance problem for alternating tree automata equipped with Muller acceptance condition (hereafter, shortly called tree automata) running over (possibly non-regular) deterministic colored trees. The proposed technique can be viewed as a generalization of the contraction method for infinite words [35]. Intuitively, the contraction method exploits a suitable 'indistinguishability' relation for a given class of automata in order to reduce instances of the acceptance problem to trivial instances involving basic 'regular' structures (e.g., ultimately periodic words or deterministic regular trees). Given the very close connection between automata and MSO logic [2, 88], such a method allows one to decide the model checking problem for MSO logic interpreted over a large class of relational structures.

Given a *non-empty full* tree T, we shall denote by Acc_{T} the *acceptance problem for* T, namely, the problem of deciding whether any given tree automaton $\mathcal{A}$ accepts T (if T is a D-augmented tree, then D-augmented tree automata are used in place of standard ones).

Such a definition of acceptance problem can be extended to any (possibly empty or non-full) tree T, by simply appending to every missing successor of an internal vertex of T an infinite complete $\perp$-colored tree. Such an operation is called *completion* of T. More precisely, if T is the empty tree, then the completion of T, denoted $\mathsf{T}_\perp$, is the infinite complete $\perp$-colored tree. If T is a non-empty (D-augmented) tree, then the completion $\mathsf{T}_\perp$ of T is defined as follows:

$$\mathcal{D}om(\mathsf{T}_\perp) = \mathcal{D}om(\mathsf{T}) \cup \bigcup_{a \in A} \big(V_a \{a\} A^*\big)$$

$$\mathsf{T}_\perp(v) = \begin{cases} \mathsf{T}(v) & \text{for every } v \in \mathcal{D}om(\mathsf{T}), \\ \perp & \text{for every } v \in \mathcal{D}om(\mathsf{T}_\perp) \setminus \mathcal{D}om(\mathsf{T}), \end{cases}$$

where V_a is the set of all internal vertices of T that have no an a-successor.

Notice that $\mathsf{T}_\perp$ is a *non-empty full* tree and it is D-augmented if and only if T is D-augmented. This makes it possible to apply the notions of run to any arbitrary tree, under the proviso that the input alphabet of the automaton contains the dummy symbol $\perp$.

As a preliminary step, we show that the acceptance problem is decidable for every regular tree T, provided that a suitable representation of T, e.g., a rooted finite graph, is available. Such a result is based on a simple reduction of the acceptance problem to the emptiness problem, that is, to the problem of establishing whether any given tree automaton recognizes a non-empty language, which is known to be decidable.

Proposition 26. *Given (a representation of) a regular tree* T *and an augmented tree automaton* $\mathcal{A}$*, one can compute an input-free tree automaton* $\mathcal{A}'$ *such that* $\mathscr{L}(\mathcal{A}') \neq \emptyset$ *iff* $\mathsf{T} \in \mathscr{L}(\mathcal{A})$*.*

Proof. Without loss of generality, we can assume that T is infinite and complete (if this is not the case, simply add $\perp$-colored vertices to T and properly extend the transition function of $\mathcal{A}$). The automaton $\mathcal{A}'$ is obtained from the synchronized product of $\mathcal{A}$ and the finite graph representing T as follows. Let (G, Ω), with $G = (V, (E_a)_{a \in A})$, be a finite graph and let v_0 be one of its vertices such that the unfolding of G from v_0 is isomorphic to T and let $\mathcal{A} = (A, C, S, \Delta, \mathcal{I}, \mathcal{F})$. We define $\mathcal{A}' = (A, C', S', \Delta', \mathcal{I}', \mathcal{F}')$, where

- $C' = \{\#\}$;
- $S' = S \times V$;
- for every state $s' = (s, v) \in S'$, $\Delta'(s', \#)$ is the formula obtained from $\Delta(s, \Omega(v))$ by replacing every atom of the form $\langle a, r \rangle$ with $\langle a, (r, v') \rangle$, where v' is the a-successor of v;
- $\mathcal{I}' = \mathcal{I} \times \{v_0\}$;
- $\mathcal{F}' = \big\{F' \subseteq S \times V : \downarrow_1 F' \in \mathcal{F}\big\}$.

It is easy to verify that every (successful) run of $\mathcal{A}$ on T can be obtained from a (successful) run R' of $\mathcal{A}'$ on the infinite complete $\#$-colored tree by erasing the second component of each state appearing in R'. Symmetrically, for every (successful) run R' of $\mathcal{A}'$, the tree obtained by erasing the second component of each state appearing in R' is a (successful) run R of $\mathcal{A}$ on T. $\square$

3.2.1 Features and Types

In the following, we introduce the basic ingredients of the contraction method for tree automata[3]. The relevant information about a run of a given automaton on a D-augmented tree can be collected in a suitable finite data structure, called *feature*.

Definition 19. *Let* T *be a non-empty full* D*-augmented tree and let* R *be a run of a* D*-augmented tree automaton* $\mathcal{A}$ *on* T*. We define the* feature $[\mathsf{T}, \mathsf{R}]$ *as the triple*

$$\begin{pmatrix} \downarrow_2 \mathsf{R}(\varepsilon) \\ \big\{ \mathcal{I}nf(\mathsf{R}|\pi) \;:\; \pi \in \mathcal{B}ch(\mathsf{R}) \big\} \\ \big\{ \big(\mathsf{T}(\downarrow_1 \mathsf{R}(\nu)),\ \downarrow_2 \mathsf{R}(\nu),\ \mathcal{I}mg(\mathsf{R}|\pi_\nu)\big) \;:\; \nu \in \mathcal{F}r(\mathsf{R}) \big\} \end{pmatrix}$$

where ε *denotes the root of* R*,* $\downarrow_1 \mathsf{R}(\nu)$ *denotes the vertex of* T *that corresponds to* ν*,* $\downarrow_2 \mathsf{R}(\nu)$ *denotes the state that appears at the vertex* ν*,* $\mathcal{B}ch(\mathsf{R})$ *denotes the set of all infinite paths in* R*,* $\mathcal{I}mg(\mathsf{R}|\pi_\nu)$ *denotes the set of all states that occur at least once along the access path* π_ν *of* ν*, and* $\mathcal{F}r(\mathsf{R})$ *denotes the set of all leaves of* R*.*

The above definition accounts for occurrences of states along maximal (finite or infinite) paths in R. In particular, the first component of the feature $[\mathsf{T}, \mathsf{R}]$ identifies the state appearing at the root of the run, the second component identifies, for every infinite path π in R, the set of states that occur infinitely often, and the third component identifies, for every leaf ν of R, the color of the vertex of T that corresponds to ν, the state that appears at ν, and the set of states that occur at least once along the access path π_ν of ν.

Given a non-empty full D-augmented tree T and a D-augmented tree automaton $\mathcal{A}$, in order to decide whether $\mathsf{T} \in \mathscr{L}(\mathcal{A})$, we introduce the notion of $\mathcal{A}$-type of T, which is a collection of features of the form $[\mathsf{T}, \mathsf{R}]$, where R ranges over a suitable set $\mathcal{R}$ of runs of $\mathcal{A}$ on T (different choices for $\mathcal{R}$ may result into different $\mathcal{A}$-types of T). We allow $\mathcal{R}$ to be a *proper subset* of the set of all runs of $\mathcal{A}$ on T, because there can exist runs which are subsumed by other ones and thus can be 'forgotten'. The notion of subsumed run is defined as follows. Given two runs R, R' of $\mathcal{A}$ on T, we say that R' is *subsumed* by R, and we write $\mathsf{R} \preceq \mathsf{R}'$, if and only if the following conditions hold:

i) the state that appears at the root of R coincides with the state that appears at the root of R';
ii) for every infinite path π in R, there is an infinite path π' in R' such that $\mathcal{I}nf(\mathsf{R}|\pi) = \mathcal{I}nf(\mathsf{R}'|\pi')$;

[3] As a matter of fact, the proposed definitions can be easily adapted to different classes of automata, such as, for instance, non-deterministic/alternating parity/Rabin tree automata.

iii) for every leaf ν of R, there is a leaf ν' of R' such that $\mathsf{T}(\downarrow_1 \mathsf{R}(\nu)) = \mathsf{T}(\downarrow_1 \mathsf{R}'(\nu'))$, $\downarrow_2 \mathsf{R}(\nu) = \downarrow_2 \mathsf{R}'(\nu')$, and $\mathcal{I}mg(\mathsf{R}|\pi_\nu) = \mathcal{I}mg(\mathsf{R}'|\pi_{\nu'})$.

Example 2. Let $\mathsf{A} = \{a_1, a_2\}$ be a set of labels, $\mathsf{C} = \{c\}$ be a set of colors, and $\mathsf{D} = \{d_1, d_2\}$ be a set of markers for the leaves of the trees. We consider the tree $\mathsf{T} = c\langle d_1, d_2\rangle$ and a D-augmented tree automaton $\mathcal{A} = (\mathsf{A}, \mathsf{C}, \mathsf{D}, \{s_1, s_2\}, \Delta, \mathcal{I}, \mathcal{F})$ such that

$$\begin{aligned}\Delta(s_1, c) &= \big(\langle a_1, s_1\rangle \wedge \langle a_2, s_2\rangle\big) \vee \big(\langle a_1, s_1\rangle \wedge \langle a_2, s_1\rangle \wedge \langle a_2, s_2\rangle\big)\\ \Delta(s_2, c) &= \langle a_1, s_2\rangle \vee \big(\langle a_2, s_1\rangle \wedge \langle a_2, s_2\rangle\big)\end{aligned}$$

The automaton $\mathcal{A}$ admits the following possible runs on T:

$$\begin{tikzcd}
\boxed{(\varepsilon, s_1)} \arrow[d] \arrow[dr] & & & \boxed{(\varepsilon, s_1)} \arrow[dl] \arrow[d] \arrow[dr] & & \boxed{(\varepsilon, s_2)} \arrow[d] & \boxed{(\varepsilon, s_2)} \arrow[d] \arrow[dr] & \\
\boxed{(a_1, s_1)} & \boxed{(a_2, s_2)} & \boxed{(a_1, s_1)} & \boxed{(a_2, s_1)} & \boxed{(a_2, s_2)} & \boxed{(a_1, s_2)} & \boxed{(a_2, s_1)} & \boxed{(a_2, s_2)}
\end{tikzcd}$$

It is easy to see that the first run subsumes the second one and that this is the only relationship that holds between the above runs.

The relation $\preceq$ on the runs is a *preorder*, i.e., a reflexive and transitive relation, and it induces a *partial order* $\leqslant$ on the set of features of $\mathcal{A}$ on T. Precisely, for every pair of features t, t', we have $t \leqslant t'$ if and only if (i) $\downarrow_1 t = \downarrow_1 t'$, (ii) $\downarrow_2 t \subseteq \downarrow_2 t'$, and (iii) $\downarrow_3 t \subseteq \downarrow_3 t'$. It is easy to see that if R and R' are two runs of $\mathcal{A}$ on T such that $\mathsf{R} \preceq \mathsf{R}'$, then R is successful whenever R' is successful. Given a set $\mathcal{R}$ of runs of $\mathcal{A}$ on T, we say that $\mathcal{R}$ is *complete* if for every run R' of $\mathcal{A}$ on T, there is $\mathsf{R} \in \mathcal{R}$ such that $\mathsf{R} \preceq \mathsf{R}'$.

Definition 20. *Given a tree* T *and a* D*-augmented tree automaton* $\mathcal{A}$*, an* $\mathcal{A}$-type *of* T *is a set of features of the form* $[\mathsf{T}, \mathsf{R}]$*, where* R *ranges over some complete set* $\mathcal{R}$ *of runs of* $\mathcal{A}$ *on* T*.*

In the above definition, the $\mathcal{A}$-type $\big\{[\mathsf{T}, \mathsf{R}] \,:\, \mathsf{R} \in \mathcal{R}\big\}$ is said to be *major* (resp., *minor*) if $\mathcal{R}$ is chosen to be maximal (resp., minimal) among the complete sets of runs of $\mathcal{A}$ on T. Clearly, the maximal complete set of runs is the set of all runs, hence the major $\mathcal{A}$-type is unique. Analogously, since complete sets of runs are closed under intersections, the minor $\mathcal{A}$-type is unique as well.

We can easily extend the notion of $\mathcal{A}$-type to any possibly empty or non-full tree T, under the proviso that the input alphabet of the automaton $\mathcal{A}$ contains the dummy symbol $\perp$. More precisely, if T is the empty tree or a non-full tree, then the $\mathcal{A}$-type of T is defined as the $\mathcal{A}$-type of the completion $\mathsf{T}_\perp$ of T.

Hereafter, given a D-augmented tree automaton $\mathcal{A}$, we denote by $\mathscr{T}_A$ the set of the *minor* $\mathcal{A}$-types of all possible D-augmented trees. Since $\mathscr{T}_A$ is included in the finite set $\mathscr{P}\big(\mathsf{S} \times \mathscr{P}(\mathscr{P}(\mathsf{S})) \times \mathscr{P}(\mathsf{D} \times \mathsf{S} \times \mathscr{P}(\mathsf{S}))\big)$, there exist only finitely many different minor $\mathcal{A}$-types for any choice of the automaton $\mathcal{A}$.

3.2.2 Types and the Acceptance Problem

Now we prove that the acceptance problem for a tree T is equivalent to the problem of computing (and checking) one of its $\mathcal{A}$-types, for any given tree automaton $\mathcal{A}$.

Proposition 27. *Given a* D*-augmented tree automaton* $\mathcal{A}$ *and an* $\mathcal{A}$*-type* σ *of a (non-empty full)* D*-augmented tree* T*, one can decide whether* $\mathsf{T} \in \mathscr{L}(\mathcal{A})$*.*

Proof. Let $\mathcal{A} = (\mathsf{A}, \mathsf{C}, \mathsf{D}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$ and let σ be the set:

$$\left\{ \begin{pmatrix} r_h \\ \{F_{h,i} \,:\, i \in I_h\} \\ \{(d_{h,j},\, r_{h,j},\, G_{h,j}) \,:\, j \in J_h\} \end{pmatrix} : h \in H \right\}$$

where H is a finite set of indices (each one representing a set of computations characterized by the same feature) and, for all $h \in H$, I_h and J_h are finite sets of indices, r_h and $r_{h,j}$, with $j \in J_h$, are states from S, $d_{h,j}$, with $j \in J_h$, are labels from D, and $F_{h,i}$, with $i \in I_h$, and $G_{h,j}$, with $j \in J_h$, are sets of states from S.

Suppose that $\mathcal{A}$ accepts T. This implies the existence of a successful run R of $\mathcal{A}$ whose feature $[\mathsf{T}, \mathsf{R}]$ belongs to σ, that is, there is $h \in H$ such that

- $r_h \in \mathcal{I}$;
- $F_{h,i} \in \mathcal{F}$ for all $i \in I_h$;
- $(d_{h,j}, r_{h,j}) \in \mathcal{G}$ for all $j \in J_h$.

The above conditions can be easily checked on the given $\mathcal{A}$-type σ. The converse implication holds symmetrically: if there exists an index in H that satisfies the above conditions, the automaton $\mathcal{A}$ accepts T. □

Proposition 28. *Given a* D*-augmented tree* T*, a* D*-augmented tree automaton* $\mathcal{A}$*, and a decision procedure for* Acc_{T}*, one can build the minor and the major* $\mathcal{A}$*-types of* T*.*

Proof. Given a D-augmented tree T, suppose Acc_{T} to be decidable. Let $\mathcal{A} = (\mathsf{A}, \mathsf{C}, \mathsf{D}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$ be a D-augmented tree automaton. We show how to build the minor $\mathcal{A}$-type σ of T by exploiting the decision procedure for Acc_{T} (a similar argument can be used for the major $\mathcal{A}$-type of T). We take advantage of the partial order $\leqslant$ between features induced by the preorder $\preceq$ between runs of $\mathcal{A}$ on T.

The construction of σ is based on an automaton-driven selection of the candidate features, starting from the minimal (with respect to $\leqslant$) ones. For any candidate feature t of the form

$$\begin{pmatrix} r \\ \{F_i \,:\, i \in I\} \\ \{(d_j,\, r_j,\, G_j) \,:\, j \in J\} \end{pmatrix}$$

we define the D-augmented tree automaton $\mathcal{A}_t = (A, C, D, S_t, \Delta_t, \mathcal{I}_t, \mathcal{F}_t, \mathcal{G}_t)$ as follows:

- $S_t = S \times \mathscr{P}(S)$;
- for every state $q = (s, Y) \in S_t$ and every symbol $c \in C$, $\Delta_t(q, c)$ is the formula obtained from $\Delta(s, c)$ by replacing every atom of the form $\langle a, s' \rangle$ with $\langle a, q' \rangle$, where $q' = (s', Y')$ and $Y' = Y \cup \{s'\}$;
- $\mathcal{I}_t$ is the singleton $\{(r, \{r\})\}$;
- $\mathcal{F}_t$ consists of all sets of the form $\{(s_1, Y), ..., (s_k, Y)\}$ such that $s_1, ..., s_k \in S$, $Y \subseteq S$, and $\{s_1, ..., s_k\} = F_i$ for some $i \in I$;
- $\mathcal{G}_t$ consists of all tuples (d_j, q_j), where $q_j = (r_j, G_j)$, for j ranging over J.

It is easy to check that $\mathcal{A}_t$ accepts T if and only if there is a feature t' of $\mathcal{A}$ on T such that $t' \leqslant t$. In general, t is a feature of σ if and only if $T \in \mathscr{L}(\mathcal{A}_t)$ and, for every feature $t'' \neq t$, with $t'' \leqslant t$, $\mathcal{A}_{t''}$ rejects T. Therefore, the minor $\mathcal{A}$-type σ can be build by selecting only the minimal features t that are accepted by the corresponding automaton $\mathcal{A}_t$. Such a selection can be performed using the decision procedure for Acc_T. $\square$

From Propositions (26-28), it follows that: (i) one can compute $\mathcal{A}$-types of regular trees and (ii) pairs of trees T, T' having a common $\mathcal{A}$-type are indistinguishable by the automaton $\mathcal{A}$, that is, $T \in \mathscr{L}(\mathcal{A})$ iff $T' \in \mathscr{L}(\mathcal{A})$. Furthermore, by looking at the proof of Proposition 28, we know the set of all trees T that have the same designated minor $\mathcal{A}$-type σ is a rational tree language, namely, it is a language recognized by a suitable tree automaton.

Summing up, we have shown that the model checking problem, the acceptance problem, and the problem of computing the minor (or the major) type of a tree T are inter-reducible. In the next section, we shall prove that, when dealing with such problems, one can safely replace T with a retraction of it, that is, a suitable tree-shaped structure that collects the minor types of the 'factors' of T.

3.2.3 From Trees to Their Retractions

We now show how minor $\mathcal{A}$-types can actually be exploited to solve non-trivial instances of the acceptance problem. To this end, we introduce the notion of factorization, which allows us to decompose a tree T into its basic components. Each component, called factor, is obtained by selecting the elements of T that lie in between some distinguished vertices. Taking advantage of the notion of factorization, we define the corresponding retraction of T, which is a tree-shaped arrangement of the minor $\mathcal{A}$-types of the factors of T. Then, we prove that the acceptance problem for T can be reduced to the acceptance problem for (a suitable encoding of) the retraction of T.

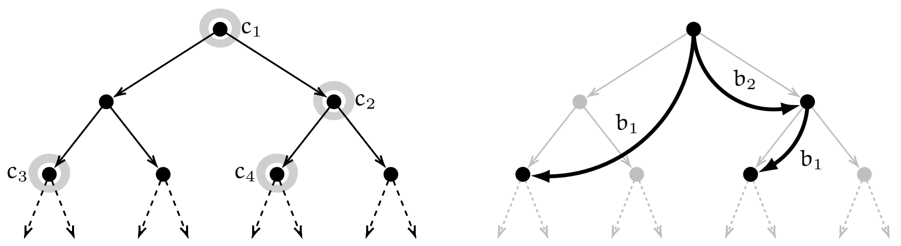


Fig. 3.2. An example of factorization

Definition 21. *Given a* D*-augmented tree* T*, a* factorization *of* T *is a (possibly non-deterministic) labeled uncolored tree* Π *such that*

- $\mathcal{D}om(\Pi)$ *is a subset of* $\mathcal{D}om(T)$ *that contains the root of* T*;*
- *for every pair of distinct vertices* u, v *of* Π*,* (u, v) *is an edge of* Π *iff* u *is an ancestor of* v *in* T *and there exists no other vertex* $v' \in \mathcal{D}om(\Pi)$ *that occurs along the path from* u *to* v *in* T*;*
- *the labels of the edges of* Π *are arbitrarily chosen in a designated finite set* B*.*

We can graphically represent a factorization of a tree by first identifying its vertices (e.g., circled nodes in the left-hand side of Figure 3.2) and then drawing the resulting edges together with the chosen labels (e.g., the bold arrows in the right-hand side of Figure 3.2).

The (marked) factors of a tree T with respect to a factorization Π of T are defined as follows. Let u be a vertex of Π and *Succ*(u) the set of all successors of u in Π. The *unmarked factor* of T rooted at u, denoted $T_{\Pi}[u]$, is the tree obtained by selecting all descendants of u in T which are not proper descendants of any vertex $v \in$ *Succ*(u) in T. For every $v \in$ *Succ*(u), we define the *marker* of v, denoted $m_{\Pi}[v]$, as the label $b \in B$ of the (unique) edge of Π having v as target vertex. The *marked factor* of T rooted at u, denoted $T^{+}_{\Pi}[u]$, is defined as the tree obtained from $T_{\Pi}[u]$ by recoloring each leaf v with the corresponding marker $m_{\Pi}[v]$.

Clearly, the marked factors of a D-*augmented* tree T with respect to a B-*labeled* factorization Π are $(B \cup D)$-*augmented* trees. For the rest of this section, in order to simplify the notation, we assume that the set B of the edge labels of a factorization Π *includes* the set D of the markers of the leaves of T.

Definition 22. *Let* T *be a* D*-augmented tree,* Π *a* B*-labeled factorization of* T*, with* $D \subseteq B$*, and* $\mathcal{A}$ *be a* B*-augmented tree automaton. The* retraction *of* T *with respect to* $\mathcal{A}$ *and* Π *is the* B-labeled $\mathscr{T}_{\mathcal{A}}$-colored *(*$\mathscr{T}_{\mathcal{A}}$*-augmented) tree* $\widetilde{T}$ *such that*

- $\mathcal{D}om(\widetilde{T}) = \mathcal{D}om(\Pi)$*;*
- *for every* $b \in B$*,* (u, v) *is a* b*-labeled edge in* $\widetilde{T}$ *iff* (u, v) *is a* b*-labeled edge in* Π*;*
- *each vertex* u *of* $\widetilde{T}$ *is colored with the minor* $\mathcal{A}$*-type of the corresponding marked factor* $T^{+}_{\Pi}[u]$*.*

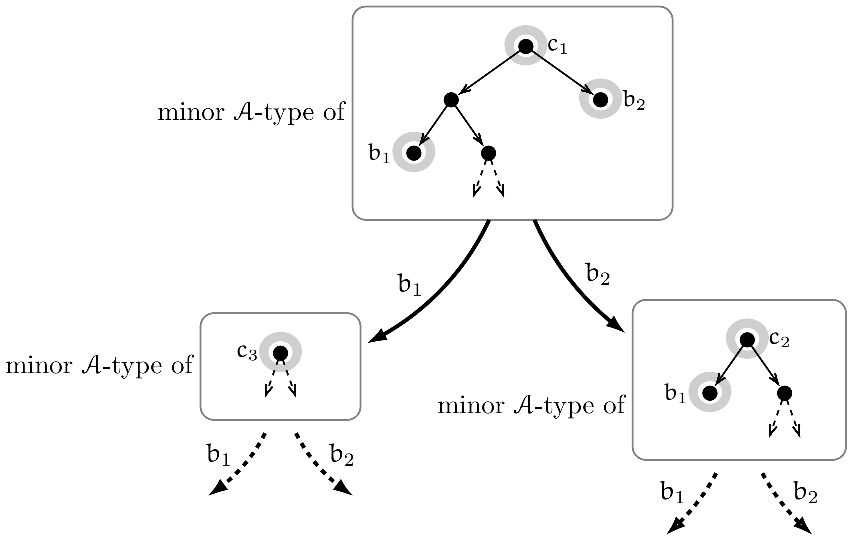

minor $\mathcal{A}$-type of
c_1
b_2
b_1
b_1
b_2
minor $\mathcal{A}$-type of
c_3
c_2
minor $\mathcal{A}$-type of
b_1
b_1
b_2
b_1
b_2

Fig. 3.3. An example of retraction

As an example, Figure 3.3 depicts the retraction of the tree of Figure 3.2 with respect to a given tree automaton $\mathcal{A}$.

In general, the retraction $\widetilde{\mathsf{T}}$, as well as the factorization Π, may be a non-deterministic tree, possibly having vertices with unbounded (or even infinite) out-degree. However, since tree automata operate on deterministic trees, we must restrict ourselves to retractions which are either deterministic trees or bisimilar to deterministic trees. As an example, the latter case occurs when for every pair of edges (u, v) and (u, v') in $\widetilde{\mathsf{T}}$ labeled with the same symbol, the subtrees of $\widetilde{\mathsf{T}}$ rooted at v and v' are isomorphic. Formally, we say that a tree $\overrightarrow{\mathsf{T}}$ *encodes* a retraction $\widetilde{\mathsf{T}}$ if $\overrightarrow{\mathsf{T}}$ is an *infinite complete deterministic* tree bisimilar to the infinite complete tree obtained from $\widetilde{\mathsf{T}}$ by adding $\perp$-colored vertices. Such an encoding of a retraction (if exists) is unique up to isomorphisms and it can be provided as input to a suitable tree automaton.

As a matter of fact, one may define different factorizations of the same tree T for different tree automata. This allows one to vary the (encoding of the) retraction from one automaton to the other. However, it is not known whether such a flexibility really improves the strength of the method. As a matter of fact, all trees we shall take into consideration feature a single factorization such that, for any given automaton, the corresponding retraction is encoded by a suitable infinite complete deterministic tree.

Now, we show how, given a tree T, a B-labeled factorization Π of T, and a B-augmented tree automaton $\mathcal{A}$, one can build a suitable tree automaton $\overrightarrow{\mathcal{A}}$ (which only depends on $\mathcal{A}$) such that $\mathcal{A}$ accepts T iff $\overrightarrow{\mathcal{A}}$ accepts the encoding of the retraction of T with respect to $\mathcal{A}$ and Π. Intuitively, the automaton $\overrightarrow{\mathcal{A}}$

mimics the behavior of $\mathcal{A}$ at a 'coarser' level. Its input alphabet is the set $\mathscr{T}_{\mathcal{A}}$ of all minor $\mathcal{A}$-types plus the additional symbol $\perp$; its states encode the finite amount of information processed by $\mathcal{A}$ during its computations up to a certain point; its transitions compute the new states which extend information given by the current state with information provided by the input symbol, i.e., the minor $\mathcal{A}$-type of a marked factor or the dummy symbol $\perp$.

Definition 23. *For every* B*-augmented tree automaton* $\mathcal{A} = (\mathsf{A}, \mathsf{C}, \mathsf{B}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$*, we define the* retraction automaton $\overrightarrow{\mathcal{A}} = (\mathsf{B}, \mathscr{T}_{\mathcal{A}} \cup \{\perp\}, \overrightarrow{\mathsf{S}}, \overrightarrow{\Delta}, \overrightarrow{\mathcal{I}}, \overrightarrow{\mathcal{F}})$ *as follows:*

- $\overrightarrow{\mathsf{S}}$ *consists of all subsets* X *of* S*, all triples* $(\mathsf{b}, \mathsf{s}, \mathsf{Y}) \in \mathsf{B} \times \mathsf{S} \times \mathscr{P}(\mathsf{S})$*, and all quadruples* $(\mathsf{b}, \mathsf{s}, \mathsf{Y}, \mathsf{Z}) \in \mathsf{B} \times \mathsf{S} \times \mathscr{P}(\mathsf{S}) \times \mathscr{P}(\mathsf{S})$[4]*;*
- *for every state* $\vec{\mathsf{s}} \in \mathscr{P}(\mathsf{S}) \cup (\mathsf{B} \times \mathsf{S} \times \mathscr{P}(\mathsf{S}))$ *and every input symbol* $\sigma \in \mathscr{T}_{\mathcal{A}} \cup \{\perp\}$*,*
$$\overrightarrow{\Delta}(\vec{\mathsf{s}}, \sigma) = \bigwedge_{\mathsf{b}' \in \mathsf{B}} \langle \mathsf{b}', \vec{\mathsf{s}} \rangle;$$
- *for every state* $\vec{\mathsf{s}} = (\mathsf{b}, \mathsf{s}, \mathsf{Y}, \mathsf{Z}) \in \mathsf{B} \times \mathsf{S} \times \mathscr{P}(\mathsf{S}) \times \mathscr{P}(\mathsf{S})$*,*
$$\overrightarrow{\Delta}(\vec{\mathsf{s}}, \perp) = \bigwedge_{\mathsf{b}' \in \mathsf{B}} \langle \mathsf{b}', \vec{\mathsf{s}}\,' \rangle,$$
where $\vec{\mathsf{s}}\,' = (\mathsf{b}, \mathsf{s}, \mathsf{Y})$*;*
- *for every state* $\vec{\mathsf{s}} = (\mathsf{b}, \mathsf{s}, \mathsf{Y}, \mathsf{Z}) \in \mathsf{B} \times \mathsf{S} \times \mathscr{P}(\mathsf{S}) \times \mathscr{P}(\mathsf{S})$ *and every minor* $\mathcal{A}$*-type* σ *of the form*
$$\left\{ \begin{pmatrix} \mathsf{r}_{\mathsf{h}} \\ \{\mathsf{F}_{\mathsf{h},\mathsf{i}} \,:\, \mathsf{i} \in \mathsf{I}_{\mathsf{h}}\} \\ \{(\mathsf{b}_{\mathsf{h},\mathsf{j}},\, \mathsf{r}_{\mathsf{h},\mathsf{j}},\, \mathsf{G}_{\mathsf{h},\mathsf{j}}) \,:\, \mathsf{j} \in \mathsf{J}_{\mathsf{h}}\} \end{pmatrix} \,:\, \mathsf{h} \in \mathsf{H} \right\}$$

[4] The subsets X of S store information about the states of $\mathcal{A}$ that occur infinitely often along infinite paths of previously visited factors, while the triples of the form $(\mathsf{b}, \mathsf{s}, \mathsf{Y})$ appear as soon as a $\perp$-colored vertex in $\overrightarrow{\mathsf{T}}$ is visited and they describe, for some leaf of T, the marker b of the leaf, the state s at the leaf, and the set Y of states that occurred at least once along the access path to the leaf. When the automaton $\overrightarrow{\mathcal{A}}$ reaches one of the above states, it loops forever on it. Quadruples of the form $(\mathsf{b}, \mathsf{s}, \mathsf{Y}, \mathsf{Z})$ are used to store information about the computation of $\mathcal{A}$ up to a certain vertex ν of Π. The first component identifies the label b of the edge of Π that reaches ν (if ν is the root of Π, it is an arbitrary label), the second component identifies the state s at ν, and the third (resp., the fourth) component gives the set Y of states that occurred at least once along the whole access path to ν (resp., the set Z of states that occurred along the suffix of the access path of ν that lies entirely inside the last visited factor).

we have

$$\vec{\Delta}(\vec{s}, \sigma) = \bigvee_{\substack{h \in H \\ \textit{with } r_h = s}} \Big(\bigwedge_{\substack{i \in I_h \\ b' \in B}} \langle b', F_{h,i} \rangle \;\wedge\; \bigwedge_{j \in J_h} \langle b_{h,j}, \vec{s}_{h,j} \rangle \Big)$$

where $\vec{s}_{h,j} = (b_{h,j},\, r_{h,j},\, Y \cup G_{h,j},\, G_{h,j})$
(if H *is empty,* $\vec{\Delta}(\vec{s}, \sigma) = \texttt{false}$ *and thus there is no valid run of* $\vec{\mathcal{A}}$*);*

- $\vec{\mathcal{I}}$ *consists of all quadruples* $(b, s, \{s\}, \{s\})$*, with* $b \in B$ *and* $s \in \mathcal{I}$*;*
- $\vec{\mathcal{F}}$ *consists of all sets of one of the following forms:*
  i) $\big\{X\big\}$*, with* $X \in \mathcal{F}$*,*
  ii) $\big\{(b, s, Y)\big\}$*, with* $(b, s) \in \mathcal{G}$ *and* $Y \subseteq S$*,*
  iii) $\big\{(b_1, s_1, Y_1, Z_1), ..., (b_k, s_k, Y_k, Z_k)\big\}$*, with* $k > 0$ *and* $\bigcup_{1 \leqslant l \leqslant k} Z_l \in \mathcal{F}$*.*

The next theorem reduces the acceptance problem for a tree T to the acceptance problem for the encoding of a retraction of T (the proof is quite long and technical and thus moved to the appendix).

Theorem 5. *Let* T *be a* D*-augmented tree,* Π *be a* B*-labeled factorization of* T*, with* $D \subseteq B$*, and* $\mathcal{A}$ *be a* B*-augmented tree automaton. We have that*

$$T \in \mathscr{L}(\mathcal{A}) \qquad \textit{iff} \qquad \vec{T} \in \mathscr{L}(\vec{\mathcal{A}}),$$

where $\vec{T}$ *denotes the encoding of the retraction of* T *with respect to* $\mathcal{A}$ *and* Π *and* $\vec{\mathcal{A}}$ *denotes the retraction automaton of* $\mathcal{A}$*.*

The upshot of Theorem 5 is that, given a tree T and an automaton $\mathcal{A}$, if we are able to find a retraction $\widetilde{T}$ of T with respect to $\mathcal{A}$ whose encoding has a decidable acceptance problem, then we can decide whether or not $\mathcal{A}$ accepts T. Moreover, if two trees T, T' have bisimilar retractions $\widetilde{T}, \widetilde{T}'$ with respect to a given automaton $\mathcal{A}$, then they are indistinguishable by $\mathcal{A}$.

Theorem 5 can be further generalized to allow one to compute the minor $\mathcal{A}$-type σ of a tree T from the minor $\vec{\mathcal{A}}$-type $\vec{\sigma}$ of the encoding $\vec{T}$ of a retraction of T with respect to $\mathcal{A}$. As a matter of fact, there is no easy way to lift such a result to major types, even if we accordingly adapt the definition of retraction. We defer the proof of the following theorem to the appendix.

Theorem 6. *Let* T *be a* D*-augmented tree,* Π *be a* B*-labeled factorization of* T*, with* $D \subseteq B$*, and* $\mathcal{A}$ *be a* B*-augmented tree automaton. The minor* $\mathcal{A}$*-type of* T *can be computed from the minor* $\vec{\mathcal{A}}$*-type of the encoding* $\vec{T}$ *of the retraction of* T *with respect to* $\mathcal{A}$ *and* Π*, where* $\vec{\mathcal{A}}$ *is the retraction automaton of* $\mathcal{A}$*.*

Below, we provide an intuitive account of the main ingredients of the proofs of Theorem 5 and Theorem 6. The key element of both proofs is a two-way correspondence between (the features of) the runs of $\mathcal{A}$ on T and (the features of) the runs of $\vec{\mathcal{A}}$ on $\vec{T}$, defined on the basis of the following 'mimicking' relation.

Definition 24. *We say that a run* $\overrightarrow{\mathsf{R}}$ *of* $\overrightarrow{\mathcal{A}}$ *on* $\overrightarrow{\mathsf{T}}$ mimics *a run* R *of* $\mathcal{A}$ *of* T *iff the following conditions are satisfied:*

(C1) the state at the root of $\overrightarrow{\mathsf{R}}$ *is of the form* $(\mathsf{b}, \mathsf{s}, \{\mathsf{s}\}, \{\mathsf{s}\})$*, where* s *is the state at the root of* R*;*

(C2) for every infinite path $\vec{\pi}$ *in* $\overrightarrow{\mathsf{R}}$ *such that* $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ *is a singleton of the form* $\{\mathsf{X}\}$*, with* $\mathsf{X} \subseteq \mathsf{S}$*, there exists an infinite path* π *in* R *such that* $\mathcal{I}nf(\mathsf{R}|\pi) = \mathsf{X}$*;*

(C3) for every infinite path $\vec{\pi}$ *in* $\overrightarrow{\mathsf{R}}$ *such that* $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ *is a set of the form* $\{(\mathsf{b}_1, \mathsf{s}_1, \mathsf{Y}, \mathsf{Z}_1), ..., (\mathsf{b}_k, \mathsf{s}_k, \mathsf{Y}, \mathsf{Z}_k)\}$*, there exists an infinite path* π *in* R *such that* $\mathcal{I}nf(\mathsf{R}|\pi) = \bigcup_{1 \leqslant l \leqslant k} \mathsf{Z}_l$*;*

(C4) for every infinite path $\vec{\pi}$ *in* $\overrightarrow{\mathsf{R}}$ *such that* $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ *is a singleton of the form* $\{(\mathsf{b}, \mathsf{s}, \mathsf{Y})\}$*, there exists a leaf* ν *of* R *such that* $\mathsf{T}(\downarrow_1 \mathsf{R}(\nu)) = \mathsf{b}$*,* $\downarrow_2 \mathsf{R}(\nu) = \mathsf{s}$*, and* $\mathcal{I}mg(\mathsf{R}|\pi_\nu) = \mathsf{Y}$*, where* π_ν *is the access path of* ν *in* R*;*

(C5) for every infinite path π *in* R*, there exists an infinite path* $\vec{\pi}$ *in* $\overrightarrow{\mathsf{R}}$ *that satisfies one of the following two conditions:*

a) $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ *is a singleton of the form* $\{\mathsf{X}\}$*, with* $\mathsf{X} = \mathcal{I}nf(\mathsf{R}|\pi)$*,*

b) $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ *is a set of the form* $\{(\mathsf{b}_1, \mathsf{s}_1, \mathsf{Y}, \mathsf{Z}_1), ..., (\mathsf{b}_k, \mathsf{s}_k, \mathsf{Y}, \mathsf{Z}_k)\}$*, with* $\bigcup_{1 \leqslant l \leqslant k} \mathsf{Z}_l = \mathcal{I}nf(\mathsf{R}|\pi)$*;*

(C6) for every leaf ν *of* R*, there exists an infinite path* $\vec{\pi}$ *in* $\overrightarrow{\mathsf{R}}$ *such that* $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ *is a singleton of the form* $\{(\mathsf{b}, \mathsf{s}, \mathsf{Y})\}$*, with* $\mathsf{b} = \mathsf{T}(\downarrow_1 \mathsf{R}(\nu))$*,* $\mathsf{s} = \downarrow_2 \mathsf{R}(\nu)$*, and* $\mathsf{Y} = \mathcal{I}mg(\mathsf{R}|\pi_\nu)$*, where* π_ν *is the access path of* ν *in* R*.*

The above conditions guarantee that the feature $[\mathsf{T}, \mathsf{R}]$ is uniquely determined by the feature $[\overrightarrow{\mathsf{T}}, \overrightarrow{\mathsf{R}}]$.

The proofs of the two theorems rest on the following properties, which are respectively proved by Lemma 15, Lemma 16, and Lemma 17 given in Appendix A.1:

(P1) *For every run* $\overrightarrow{\mathsf{R}}$ *of* $\overrightarrow{\mathcal{A}}$ *on* $\overrightarrow{\mathsf{T}}$ *such that the state at the root of* $\overrightarrow{\mathsf{R}}$ *is a quadruple of the form* $(\mathsf{b}, \mathsf{s}, \{\mathsf{s}\}, \{\mathsf{s}\})$*, there is a run* R *of* $\mathcal{A}$ *on* T *which is mimicked by* $\overrightarrow{\mathsf{R}}$*.*

This property is proved by first extracting from $\overrightarrow{\mathsf{R}}$ suitable runs of $\mathcal{A}$ on the marked factors of T and then combining these runs to form a valid run of $\mathcal{A}$ on T, which satisfies Conditions C1–C6.

(P2) *For every run* R *of* $\mathcal{A}$ *on* T*, there is a run* R' *of* $\mathcal{A}$ *on* T *such that* $\mathsf{R}' \preceq \mathsf{R}$ *and there is a run* $\overrightarrow{\mathsf{R}}$ *of* $\overrightarrow{\mathcal{A}}$ *on* $\overrightarrow{\mathsf{T}}$ *that mimics* R'. (Note that there may exist no run $\overrightarrow{\mathsf{R}}$ of $\overrightarrow{\mathcal{A}}$ on $\overrightarrow{\mathsf{T}}$ that directly mimics R.)

This requires a three-step construction. First, given a run R of $\mathcal{A}$ on T, one builds a run $\overrightarrow{\mathsf{R}}$ of $\overrightarrow{\mathcal{A}}$ on $\overrightarrow{\mathsf{T}}$ by choosing suitable transitions of $\overrightarrow{\mathcal{A}}$ that

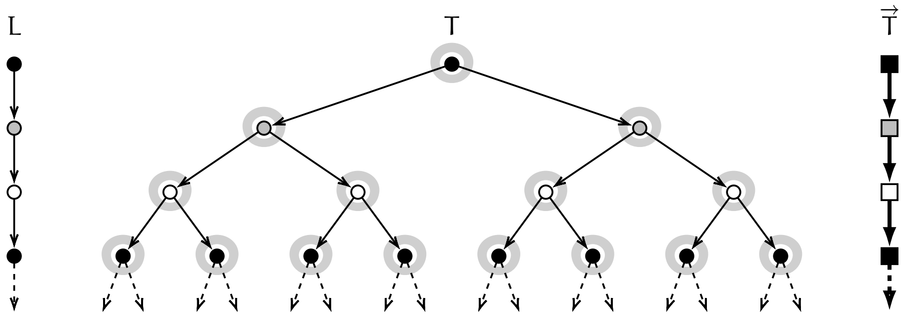


Fig. 3.4. An example of application of Theorem 5

match those of R. Then, by exploiting Property P1, one builds a new run R' of $\mathcal{A}$ on T which is mimicked by $\overrightarrow{\mathsf{R}}$. Finally, one verifies that $\mathsf{R}' \preceq \mathsf{R}$.

(P3) *Let* R_1 *and* R_2 *be two runs of* $\mathcal{A}$ *on* T *and let* $\overrightarrow{\mathsf{R}}_1$ *and* $\overrightarrow{\mathsf{R}}_2$ *be two runs of* $\overrightarrow{\mathcal{A}}$ *on* $\overrightarrow{\mathsf{T}}$. *If* $\overrightarrow{\mathsf{R}}_1$ *mimics* R_1, $\overrightarrow{\mathsf{R}}_2$ *mimics* R_2, *and* $\overrightarrow{\mathsf{R}}_1 \preceq \overrightarrow{\mathsf{R}}_2$, *then* $\mathsf{R}_1 \preceq \mathsf{R}_2$.
This easily follows from the definition of the mimicking relation.

It is easy to show that the above properties allow one to build a complete set of runs of $\mathcal{A}$ on T from a given complete set of runs of $\overrightarrow{\mathcal{A}}$ on $\overrightarrow{\mathsf{T}}$.

3.2.4 An Example

We conclude the section with a simple example of application of Theorem 5.

Example 3. Let us consider an infinite word $w : \mathbb{N} \to \mathsf{C}$. It can be thought of as an expanded linear structure $\mathsf{L} = (\mathbb{N}, \mathsf{E}, (\mathsf{P}_c)_{c \in \mathsf{C}})$, where $(i, j) \in \mathsf{E}$ iff $j = i + 1$, and $i \in \mathsf{P}_c$ iff $w(i) = c$. Let T be the infinite complete C-colored binary tree, where every vertex belonging to $i+1$-th level of the tree is colored by $w(i)$. Formally, if we label the edges of T with elements from the set $\mathsf{A} = \{a_1, a_2\}$, we have that $\mathsf{T}(v) = w(|v|)$ for every $v \in \mathsf{A}^*$ (see Figure 3.4). It is well-known that if the MSO theory of L is decidable, then that of T is decidable as well. This can be proved by showing that T is nothing but the unfolding of the graph G obtained from L via an MSO-definable interpretation that replaces each edge e of L with two distinct copies e_1 and e_2 labeled, respectively, with a_1 and a_2. By exploiting the MSO-compatibility of the unfolding operation [94, 21, 111, 112], one can reduce the model checking problem of T to that of G, which can in its turn be reduced to the model checking problem of L. Equivalently, one can prove that the MSO theory of T is decidable by exploiting the correspondence between the runs of non-deterministic tree automata on T are the runs of a alternating sequential automata on L. Our method provides an alternative proof of the decidability of the MSO theory of T, which is independent from the MSO-compatibility of

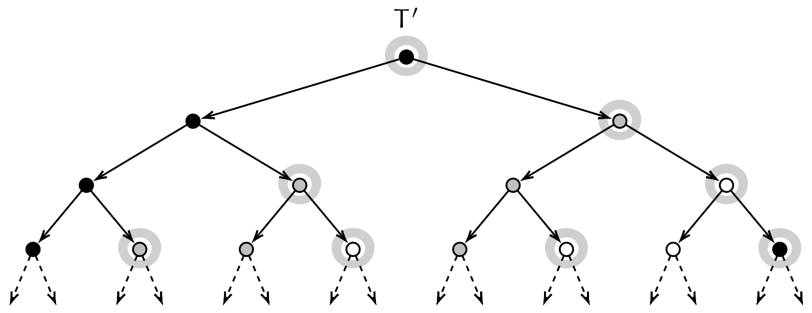


Fig. 3.5. A distorted tree from Example 3

the unfolding operation and from the equivalence between non-deterministic and alternating automata. Let $B = \{b\}$ and let Π be the B-labeled factorization of T such that $\mathcal{D}om(\Pi) = \mathcal{D}om(T)$. Let us consider a generic B-augmented tree automaton $\mathcal{A} = (A, C, B, S, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$ running on T. The retraction $\widetilde{T}$ of T with respect to $\mathcal{A}$ and Π is defined as follows:

- $\mathcal{D}om(\widetilde{T}) = \mathcal{D}om(\Pi)$;
- for every vertex v of $\widetilde{T}$, $\widetilde{T}(v)$ is the minor $\mathcal{A}$-type of the marked factor of T rooted at v (notice that such a marked factor coincides with the finite B-augmented tree $c\langle b, b\rangle$, where $c = w(|v|)$).

If we denote by Ω the (computable) function that maps each color $c \in C$ to the minor $\mathcal{A}$-type of the B-augmented tree $c\langle b, b\rangle$, then we easily see that $\widetilde{T}$ is bisimilar to the linear structure $\overrightarrow{T} = \Omega(L)$, which can be obtained from L via an MSO-definable interpretation that replaces every color $c \in C$ with the corresponding minor $\mathcal{A}$-type $\Omega(c)$. By Theorem 5, the decidability of the acceptance problem for T (and thus the decidability of the MSO theory of T) follows immediately from the decidability of the acceptance problem for L (equivalently, from the decidability of the MSO theory of L).

Even though tree automata are intimately related to MSO formulas evaluated over deterministic colored trees, we do not know whether the above example can be dealt with by means of classical compositional results from logic (see, for instance, [95, 106, 89]). Precisely, we found it difficult to directly map (without using MSO-compatibility of the unfolding operation) MSO formulas over T to equivalent MSO formulas over L. Moreover, the arguments used in the above example can be easily generalized to several 'distortions' of the tree T, e.g., to the following trees:

- the tree T' defined by $\mathcal{D}om(T') = A^*$ and $T'(v) = w(|v|_{a_2})$, for all $v \in \mathcal{D}om(T')$, where $|v|_{a_2}$ denotes the number of occurrences of the label a_2 along the access path of v (see Figure 3.5);

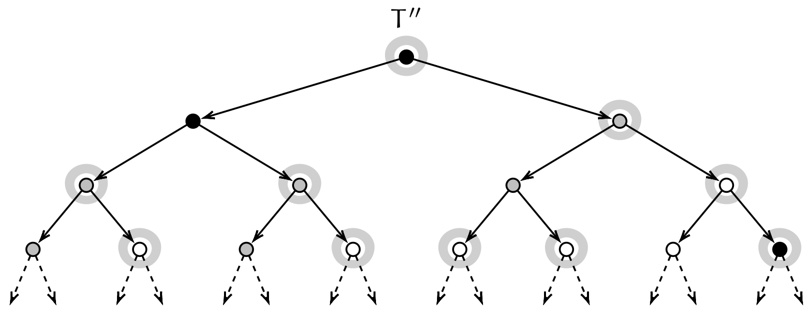


Fig. 3.6. Another distorted tree from Example 3

- the tree T'' defined by $\mathcal{D}om(\mathsf{T}'') = \mathsf{A}^*$ and $\mathsf{T}''(\nu) = w\left(\left\lfloor \frac{|\nu|+|\nu|_{a_2}}{2} \right\rfloor\right)$, for all $\nu \in \mathcal{D}om(\mathsf{T}'')$ (see Figure 3.6).

Finally, it is worth noticing that the decidability results involving the 'distortions' T' and T'' can be equivalently obtained by exploiting the correspondence between the runs of non-deterministic tree automata on T' (resp., T'') and the runs of alternating tree automata with ε-transitions on L.

3.3 Tree Transformations

This section describes and compares several natural transformations on trees, namely, recolorings, substitutions, transductions, inverse substitutions, and unfoldings. Some of them, such as transductions and unfoldings, are pretty standard, other ones, e.g., tree insertions, are less common. In the following, we provide a self-contained uniform presentation of all of them. In the next section, we shall introduce a large class of trees with a decidable acceptance problem and we prove closure properties for such a class with respect to the considered tree transformations.

3.3.1 Tree Recolorings

We distinguish three kinds of tree recoloring: memoryless recoloring, finite-state recoloring (without lookahead), and finite-state recoloring with rational lookahead. In the following, we assume D to be disjoint from C and we restrict our attention to A-labeled C-colored $(\mathsf{C} \cup \mathsf{D})$-augmented trees.

Definition 25. *A* memoryless recoloring *is a function* $\Omega : \mathsf{C} \to \mathsf{C}'$. *The memoryless recoloring of an* A*-labeled* C*-colored* $(\mathsf{C} \cup \mathsf{D})$*-augmented tree* T *is the* A*-labeled* C'*-colored* $(\mathsf{C}' \cup \mathsf{D})$*-augmented tree* $\Omega(\mathsf{T})$ *such that*

- *$\mathcal{D}om(\Omega(\mathsf{T})) = \mathcal{D}om(\mathsf{T})$;*
- *for all C-colored vertices ν of T, $\Omega(\mathsf{T})(\nu) = \Omega(\mathsf{T}(\nu))$;*
- *for all D-colored leaves ν of T, $\Omega(\mathsf{T})(\nu) = \mathsf{T}(\nu)$.*

A memoryless recoloring is a *local transformation* that only processes the C-colored vertices of an input tree, leaving the D-colored leaves unchanged. A local transformation is an operation that can be performed on each vertex ν of an input tree without taking into account the colors of its ancestors or descendants. A typical example of local transformation is the operation of second-order tree substitution. On the contrary, non-local tree recolorings use states and transitions to make the recoloring of a vertex of the input tree dependent on the current state.

A finite-state recoloring without lookahead is specified by a top-down deterministic Mealy tree automaton. Such an automaton is a deterministic tree automaton devoid of an acceptance condition and provided with an output function that, given the current state and color, produces a new color. Unlike the case of memoryless recoloring, the recoloring computed by a Mealy tree automaton $\mathcal{M}$ can change the color of D-colored leaves in the following restricted way: the new color assigned to each d-colored leaf ν of T, with $\mathsf{d} \in \mathsf{D}$, is the pair (q, d), where q is the state of $\mathcal{M}$ at the leaf ν. Such a restriction does not involve any loss of generality and it eases the study of compositional properties of trees obtained from non-local tree transformations.

Definition 26. *A (deterministic top-down)* Mealy tree automaton *is a tuple $\mathcal{M} = (\mathsf{A}, \mathsf{C}, \mathsf{D}, \mathsf{C}', \mathsf{Q}, \delta, \Omega, \mathsf{q}_0)$, where*

- *Q is a finite set of states;*
- *$\delta : \mathsf{Q} \times \mathsf{C} \times \mathsf{A} \to \mathsf{Q}$ is a transition function;*
- *$\Omega : \mathsf{Q} \times \mathsf{C} \to \mathsf{C}'$ is a recoloring function;*
- *q_0 is the initial state.*

The Mealy tree automaton $\mathcal{M}$ maps any A-labeled C-colored $(\mathsf{C}\cup\mathsf{D})$-augmented tree to a suitable A-labeled C'-colored $(\mathsf{C}' \cup \mathsf{D}')$-augmented tree, where $\mathsf{D}' = \mathsf{Q} \times \mathsf{D}$. The (unique) *run* of $\mathcal{M}$ on an A-labeled C-colored $(\mathsf{C} \cup \mathsf{D})$-augmented tree T is the A-labeled Q-colored (Q-augmented) tree R such that

- $\mathcal{D}om(\mathsf{R}) = \mathcal{D}om(\mathsf{T})$;
- $\mathsf{R}(\varepsilon) = \mathsf{q}_0$;
- for every a-labeled edge (ν, ν') in R, with $\mathsf{a} \in \mathsf{A}$, $\mathsf{R}(\nu') = \delta(\mathsf{R}(\nu), \mathsf{T}(\nu), \mathsf{a})$.

The *finite-state recoloring* of T via $\mathcal{M}$ is the A-labeled C'-colored $(\mathsf{C}' \cup \mathsf{D}')$-augmented tree $\mathcal{M}(\mathsf{T})$ such that

- $\mathcal{D}om(\mathcal{M}(\mathsf{T})) = \mathcal{D}om(\mathsf{T})$;
- for all C-colored vertices ν of T, $\mathcal{M}(\mathsf{T})(\nu) = \Omega(\mathsf{R}(\nu), \mathsf{T}(\nu))$;
- for all D-colored leaves ν of T, $\mathcal{M}(\mathsf{T})(\nu) = (\mathsf{R}(\nu), \mathsf{T}(\nu))$.

Example 4. A *rational marking* [14] is a particular form of finite-state recoloring. It is specified by a rational language L consisting of finite words over the alphabet A and it maps any A-labeled C-colored (C-augmented) tree T to the A-labeled C'-colored (C'-augmented) tree T', with $\mathsf{C}' = \mathsf{C} \times \{0, 1\}$, such that, for every vertex ν of T',

$$\mathsf{T}'(\nu) = \begin{cases} (\mathsf{T}(\nu), 1) & \text{if } \nu \in \mathsf{L}, \\ (\mathsf{T}(\nu), 0) & \text{otherwise.} \end{cases}$$

Such a recoloring can be performed by a top-down deterministic Mealy tree automaton as follows. Let us denote by $\mathcal{A} = (\mathsf{A}, \mathsf{S}, \delta, \mathsf{s}_0, \mathsf{F})$ a *deterministic* finite-state automaton that recognizes L, where δ is a function from $\mathsf{S} \times \mathsf{A}$ to S, s_0 is the initial state S, and F is the set of accepting states. We define the Mealy tree automaton $\mathcal{M} = (\mathsf{A}, \mathsf{C}, \emptyset, \mathsf{C}', \mathsf{S}, \delta', \Omega, \mathsf{s}_0)$, where

- $\delta'(\mathsf{s}, \mathsf{c}, \mathsf{a}) = \delta(\mathsf{s}, \mathsf{a})$ for all $\mathsf{s} \in \mathsf{S}$, $\mathsf{c} \in \mathsf{C}$, and $\mathsf{a} \in \mathsf{A}$;
- $\Omega(\mathsf{s}, \mathsf{c}) = (\mathsf{c}, 1)$ for all $\mathsf{s} \in \mathsf{F}$ and $\mathsf{c} \in \mathsf{C}$;
- $\Omega(\mathsf{s}, \mathsf{c}) = (\mathsf{c}, 0)$ for all $\mathsf{s} \notin \mathsf{F}$ and $\mathsf{c} \in \mathsf{C}$.

The reader can easily check that, for every tree T, $\mathcal{M}(\mathsf{T})$ coincides with the rational marking of T with respect to the language L.

Example 5. Let $(\mathsf{C}, \cdot)$ be a finite (*multiplicative*) *semigroup*, namely, let C be a finite set and let $\cdot$ be an associative binary operator over C. Furthermore, suppose that C contains the identity element 1 such that $1 \cdot \mathsf{c} = \mathsf{c} \cdot 1 = \mathsf{c}$ for every $\mathsf{c} \in \mathsf{C}$. The structure $(\mathsf{C}, \cdot)$ is said to be a (finite) *multiplicative monoid*. Let us now consider the transformation that maps any C-colored (C-augmented) tree T to the C-colored (C-augmented) tree T' such that $\mathcal{D}om(\mathsf{T}') = \mathcal{D}om(\mathsf{T})$ and $\mathsf{T}'(\nu) = \mathsf{T}(\nu_0) \cdot \mathsf{T}(\nu_1) \cdot \ldots \cdot \mathsf{T}(\nu_n)$, where $\nu_0\, \nu_1 \ldots \nu_n$ identifies the access path of ν in T. Such a transformation is computed by the Mealy tree automaton $\mathcal{M} = (\mathsf{A}, \mathsf{C}, \emptyset, \mathsf{C}, \mathsf{Q}, \delta, \Omega, \mathsf{q}_0)$, where

- $\mathsf{Q} = \mathsf{C}$;
- $\delta(\mathsf{q}, \mathsf{c}, \mathsf{a}) = \mathsf{q} \cdot \mathsf{c}$ for every $\mathsf{q} \in \mathsf{Q}$, $\mathsf{c} \in \mathsf{C}$, and $\mathsf{a} \in \mathsf{A}$;
- $\Omega(\mathsf{q}, \mathsf{c}) = \mathsf{q} \cdot \mathsf{c}$ for every $\mathsf{q} \in \mathsf{Q}$ and $\mathsf{c} \in \mathsf{C}$;
- q_0 is the identity element of $(\mathsf{C}, \cdot)$.

Finite-state recolorings with rational lookahead are specified by Mealy tree automata with the ability of inspecting the subtree issued from the current position before processing it ('lookahead').

Definition 27. *A* Mealy tree automaton with rational lookahead *is a tuple* $\mathcal{M} = (\mathsf{A}, \mathsf{C}, \mathsf{D}, \mathsf{C}', \mathcal{L}, \mathsf{Q}, \delta, \Omega, \mathsf{q}_0)$*, where*

- $\mathcal{L} = \{\mathsf{L}_1, \ldots, \mathsf{L}_k\}$ *is a set of rational languages consisting of* A*-labeled* C*-colored* $(\mathsf{C} \cup \mathsf{D})$*-augmented trees (namely, each language* L_e*, with* $1 \leqslant e \leqslant k$*, is recognized by a* $(\mathsf{C} \cup \mathsf{D})$*-augmented tree automaton);*
- Q *is a finite set of states;*

- $\delta : Q \times \mathscr{P}(\{1, ..., k\}) \times A \rightarrow Q$ *is a transition function;*
- $\Omega : Q \times \mathscr{P}(\{1, ..., k\}) \rightarrow C'$ *is a recoloring function;*
- q_0 *is the initial state.*

Both the transition function and the recoloring function take into account the set of the indices of the languages L_e that contain the subtree rooted at the current position. The (unique) *run* of $\mathcal{M}$ on an A-labeled C-colored $(C \cup D)$-augmented tree T is the A-labeled Q-colored (Q-augmentd) tree R such that

- $\mathcal{D}om(R) = \mathcal{D}om(T)$;
- $R(\varepsilon) = q_0$;
- for every a-labeled edge (v, v') in R, with $a \in A$, $R(v') = \delta(R(v), E, a)$, where $E \subseteq \{1, ..., k\}$ is the set of all indices e such that L_e contains the subtree of T rooted at v ($T^{\downarrow v}$ for short).

The *finite-state recoloring with rational lookahead* of T via $\mathcal{M}$ is the A-labeled C'-colored $(C' \cup D')$-augmented tree $\mathcal{M}(T)$, with $D' = Q \times D$, such that

- $\mathcal{D}om(\mathcal{M}(T)) = \mathcal{D}om(T)$;
- for all C-colored vertices v of T, $\mathcal{M}(T)(v) = \Omega(R(v), E)$, where $E \subseteq \{1, ..., k\}$ is the set of all indices e such that $T^{\downarrow v} \in L_e$;
- for all D-colored leaves v of T, $\mathcal{M}(T)(v) = (R(v), T(v))$.

It is immediate to see that memoryless recoloring, finite-state recoloring, and finite-state recoloring with rational lookahead form a strictly increasing hierarchy of tree transformations.

3.3.2 Tree Substitutions

We focus our attention on tree morphisms and tree insertions, both of which are derived from the general notion of second-order tree substitution. Intuitively, a second-order tree substitution replaces all c-colored vertices in a tree T by a designated tree F_c, for all colors $c \in C$. Even though second-order tree substitutions are usually applied to *ranked* trees (i.e., trees where the out-degree of each vertex is uniquely determined by the arity of the vertex), here we adapt the standard definitions to the case of deterministic unranked trees. Hereafter, we assume that the domains of the trees are prefix-closed languages over the set of the edge labels.

As a preliminary step, we introduce the notion of first-order tree substitution.

Definition 28. *Given an* A*-labeled* C*-colored* $(C \cup D)$*-augmented tree* T*, a prefix-free language* $L \subseteq \mathcal{D}om(T)$*, and, for each* $v \in L$*, an* A'*-labeled* C'*-colored* $(C' \cup D)$*-augmented tree* F_v*, the* first-order tree substitution *produces an* $A \cup A'$*-labeled* $(C \cup C')$*-colored* $(C \cup C' \cup D)$*-augmented tree* $T[F_v/v]_{v \in L}$ *defined as follows:*

- $\mathcal{D}om\big(\mathsf{T}[\mathsf{F}_v/v]_{v\in \mathsf{L}}\big) = \big(\mathcal{D}om(\mathsf{T}) \setminus (\mathsf{L}\,A^*)\big) \cup \big\{v\,w : v \in \mathsf{L},\, w \in \mathcal{D}om(\mathsf{F}_v)\big\}$;
- $\mathsf{T}[\mathsf{F}_v/v]_{v\in \mathsf{L}}(u) = \begin{cases} \mathsf{T}(u) & \textit{if } u \in \mathcal{D}om(\mathsf{T}) \setminus (\mathsf{L}\,A^*), \\ \mathsf{F}_v(w) & \textit{if } u = v\,w,\, v \in \mathsf{L},\, w \in \mathcal{D}om(\mathsf{F}_v). \end{cases}$

First-order tree substitutions are usually applied to the D-colored leaves of a tree T (in such a case, L is a subset of the frontier of T). By a slight abuse of notation, we shall write $\mathsf{T}[\mathsf{F}_\mathsf{d}/\mathsf{d}]_{\mathsf{d}\in\mathsf{D}}$ to denote the first-order tree substitution in T of all d-colored leaves by F_d for all markers $\mathsf{d} \in \mathsf{D}$ (we shall take advantage of this notation in Definition 29).

Unlike first-order tree substitutions, second-order substitutions do not force one to remove the subtrees rooted at the replaced nodes. Intuitively, the subtrees rooted at the 1-st, 2-nd, ..., k-th successor of a replaced c-colored vertex v in T are attached to the replacing tree F_c as follows: we mark some leaves of F_c (if any) with elements from the label set A, thus making F_c a $(\mathsf{C}'\cup\mathsf{D})$-augmented tree, with $\mathsf{D} \supseteq A$, and we use these markers as placeholders for the subtrees of T to be attached to F_c. Notice that, unlike first-order ones, second-order tree substitutions can occur simultaneously at vertices belonging to the same path of a tree T.

Second-order tree substitutions can be formalized as follows. As in the case of tree recolorings, let us assume the sets C and D to be disjoint. According to Courcelle [20], we first define second-order tree substitutions over *finite* trees and then we show how to extend them to the *infinite* case by using a limit argument (precisely, we shall exploit the fact that second-order tree substitutions are monotone in their arguments with respect to a suitable ω-complete partial order).

Definition 29. *Given a finite A-labeled C-colored $(\mathsf{C}\cup\mathsf{D})$-augmented tree T and, for each color $\mathsf{c} \in \mathsf{C}$, a (possibly infinite) A'-labeled C'-colored $(\mathsf{C}'\cup\mathsf{D})$-augmented tree F_c, the* second-order tree substitution *produces an A'-labeled C'-colored $(\mathsf{C}'\cup\mathsf{D})$-augmented tree $\mathsf{T}[\![\mathsf{F}_\mathsf{c}/\mathsf{c}]\!]_{\mathsf{c}\in\mathsf{C}}$ inductively defined as follows:*

$$\mathsf{T}[\![\mathsf{F}_\mathsf{c}/\mathsf{c}]\!]_{\mathsf{c}\in\mathsf{C}} = \begin{cases} \emptyset & \textit{if } \mathsf{T} = \emptyset, \\ \mathsf{F}_\mathsf{c}[\mathsf{T}^{(a)}/a]_{a\in A} & \textit{if } \mathsf{T} \neq \emptyset \textit{ and } \mathsf{T}(\varepsilon) = \mathsf{c} \in \mathsf{C}, \\ \mathsf{d} & \textit{if } \mathsf{T} \neq \emptyset \textit{ and } \mathsf{T}(\varepsilon) = \mathsf{d} \in \mathsf{D}, \end{cases}$$

where, for every label $a \in A$, $\mathsf{T}^{(a)} = \mathsf{T}^{\downarrow a}[\![\mathsf{F}_\mathsf{c}/\mathsf{c}]\!]_{\mathsf{c}\in\mathsf{C}}$ ($\mathsf{T}^{\downarrow a}$ is the subtree of T rooted at a).

Figure 3.7 depicts the result of the second-order tree substitution of c-colored vertices in T by F_c.

The above definition can be generalized to infinite trees as follows. Let us denote by `Trees` (resp., `Trees`$'$) the set of all possibly infinite A-labeled C-colored $(\mathsf{C}\cup\mathsf{D})$-augmented trees (resp., the set of all possibly infinite A'-labeled C'-colored $(\mathsf{C}'\cup\mathsf{D})$-augmented trees) and by `FinTrees` the subset

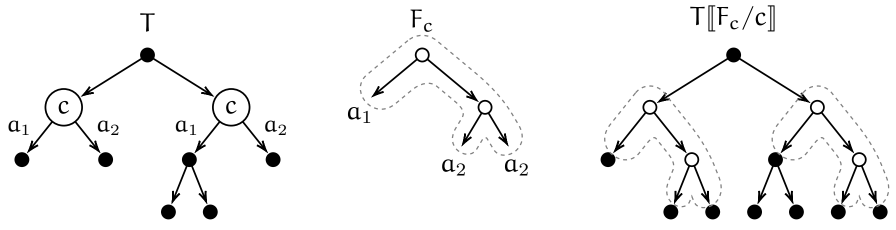


Fig. 3.7. An example of second-order tree substitution

of `Trees` consisting of *finite* trees only. The operation of second-order tree substitution can be viewed as a function from $\mathtt{FinTrees}\times(\mathtt{Trees}')^{C}$ to $\mathtt{Trees}'$, which maps any *finite* A-labeled C-colored $(C\cup D)$-augmented tree T and any tuple $\bar{F} = (F_c)_{c\in C}$ of A'-labeled C'-colored $(C' \cup D)$-augmented trees to the A'-labeled C'-colored $(C' \cup D)$-augmented tree $T[\![F_c/c]\!]_{c\in C}$. A partial order $\sqsubseteq$ over the set of all trees can be naturally defined: for every pair of trees T_1 and T_2, we let $T_1 \sqsubseteq T_2$ iff $\mathcal{D}om(T_1) \subseteq \mathcal{D}om(T_2)$ and, for every vertex v of T_1, $T_1(v) = T_2(v)$. Intuitively, $T_1 \sqsubseteq T_2$ holds iff T_1 can be obtained from T_2 by pruning some of its subtrees. It is easy to see that the empty tree is the least element of $\sqsubseteq$; moreover, $\sqsubseteq$ turns out to be an ω-complete partial order. This means that every $\sqsubseteq$-directed set X of trees has a least upper bound $Sup(X)$, which is defined as follows:

- $\mathcal{D}om(Sup(X)) = \bigcup_{T\in X} \mathcal{D}om(T)$;
- for every vertex v of $Sup(X)$, $Sup(X)(v) = T(v)$, where T is any tree in X that contains the vertex v (notice that $Sup(X)(v)$ is well-defined).

The operation of second-order tree substitution is *monotone* in its arguments. This implies (see Proposition 2.4.2 in [20]) that there is only one way to extend second-order tree substitution to an ω-continuous function from $\mathtt{Trees} \times (\mathtt{Trees}')^{C}$ to $\mathtt{Trees}'$.

Notice that, since second-order tree substitutions are local as memoryless recolorings, we can combine sequences of second-order tree substitutions, e.g., $T\left[\!\left[F_c[\![F'_{c'}/c']\!]_{c'\in C'}/c\right]\!\right]_{c\in C}$.

Finally, it is possible to show that second-order tree substitutions are associative.

Lemma 10. *Second-order tree substitutions are homomorphisms with respect to first-order tree substitutions on* D*-colored leaves, namely, for every tree* T *and every pair of tuples* $(F_d)_{d\in D}$ *and* $(F_c)_{c\in C}$ *of trees, we have*

$$T[F_d/d]_{d\in D}[\![F_c/c]\!]_{c\in C} = T[\![F_c/c]\!]_{c\in C}\left[F_d[\![F_c/c]\!]_{c\in C}/d\right]_{d\in D}.$$

A proof of Lemma 10 can be found in Section 3.5 of [20].

Proposition 29. *Second-order tree substitutions are associative, namely, for every* A*-labeled* C*-colored* $(\mathsf{C} \cup \mathsf{D})$*-augmented tree* T*, every tuple* $(\mathsf{F}_c)_{c \in \mathsf{C}}$ *of* A'*-labeled* C'*-colored* $(\mathsf{C}' \cup \mathsf{D})$*-augmented trees, and every tuple* $(\mathsf{F}'_{c'})_{c' \in \mathsf{C}'}$ A''*-labeled* C''*-colored* $(\mathsf{C}'' \cup \mathsf{D})$*-augmented trees, we have*

$$\mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in \mathsf{C}}[\![\mathsf{F}'_{c'}/c']\!]_{c' \in \mathsf{C}'} = \mathsf{T}\Big[\!\!\Big[\mathsf{F}_c[\![\mathsf{F}'_{c'}/c']\!]_{c' \in \mathsf{C}'}/c\Big]\!\!\Big]_{c \in \mathsf{C}}.$$

Proof. From previous arguments, it is sufficient to prove the claim for any *finite* tree T (the infinite case follows from ω-continuity). We prove it by induction on the height of T (the cases $\mathsf{T} = \emptyset$ and $\mathsf{T} = c$ are trivial). Suppose that T has at least one internal node. By exploiting Lemma 10 and the definition of second-order tree substitution, we obtain

$$\begin{aligned}
&\mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in \mathsf{C}}[\![\mathsf{F}'_{c'}/c']\!]_{c' \in \mathsf{C}'} \\
&\qquad = \mathsf{F}_{\mathsf{T}(\varepsilon)}\left[\mathsf{T}^{\downarrow a}[\![\mathsf{F}_c/c]\!]_{c \in \mathsf{C}}/a\right]_{a \in \mathsf{A}}[\![\mathsf{F}'_{c'}/c']\!]_{c' \in \mathsf{C}'} \\
&\qquad = \mathsf{F}_{\mathsf{T}(\varepsilon)}[\![\mathsf{F}'_{c'}/c']\!]_{c' \in \mathsf{C}'}\left[\mathsf{T}^{\downarrow a}[\![\mathsf{F}_c/c]\!]_{c \in \mathsf{C}}[\![\mathsf{F}'_{c'}/c']\!]_{c' \in \mathsf{C}'}/a\right]_{a \in \mathsf{A}} \\
&\qquad = \mathsf{F}_{\mathsf{T}(\varepsilon)}[\![\mathsf{F}'_{c'}/c']\!]_{c' \in \mathsf{C}'}\left[\mathsf{T}^{\downarrow a}[\![\mathsf{F}_c[\![\mathsf{F}'_{c'}/c']\!]_{c' \in \mathsf{C}'}/c]\!]_{c \in \mathsf{C}}/a\right]_{a \in \mathsf{A}} \\
&\qquad = \mathsf{T}\Big[\!\!\Big[\mathsf{F}_c[\![\mathsf{F}'_{c'}/c']\!]_{c' \in \mathsf{C}'}/c\Big]\!\!\Big]_{c \in \mathsf{C}}.
\end{aligned}$$

Tree morphisms and tree insertions can be viewed as special cases of second-order tree substitutions. As already pointed out, a second-order tree substitution is a function that maps a tree T and a tuple $\bar{\mathsf{F}} = (\mathsf{F}_c)_{c \in \mathsf{C}}$ of replacing trees to the tree $\mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in \mathsf{C}}$. Tree morphisms (resp., insertions) are second-order tree substitutions where the second argument $\bar{\mathsf{F}}$ (resp., first argument T) is fixed.

A *tree morphism* is a function μ, specified by a tuple $(\mathsf{F}_c)_{c \in \mathsf{C}}$ of A'-labeled C'-colored $(\mathsf{C}' \cup \mathsf{D})$-augmented trees, which maps any A-labeled C-colored D-augmented tree T to the A'-labeled C'-colored $(\mathsf{C}' \cup \mathsf{D})$-augmented tree

$$\mu(\mathsf{T}) = \mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in \mathsf{C}}.$$

The notion of tree morphism is a natural generalization of that of word morphism [12]. In particular, we have that, given $\mathsf{A} = \{a_1, ..., a_k\}$, every tree morphism μ is *uniquely determined* by the images $\mu(c\langle a_1, ..., a_k\rangle)$, for c ranging over C (this follows from the monotonicity of μ with respect to the partial order $\sqsubseteq$). A tree morphism μ is said to be *regular* if F_c is a regular tree for every color $c \in \mathsf{C}$. At the top of Figure 3.8, we depict a regular tree morphism for the set of edge labels $\mathsf{A} = \{a_1, a_2\}$.

The notion of regular tree morphism naturally leads to that of morphic trees, viewed as the limits of n-fold iterations of suitable regular tree morphisms. For each color $c \in \mathsf{C}$, let F_c a regular A-labeled C-colored $(\mathsf{C} \cup \mathsf{D})$-augmented tree, and let μ be the corresponding regular tree morphism. Moreover, let $\tilde{c}$ be a

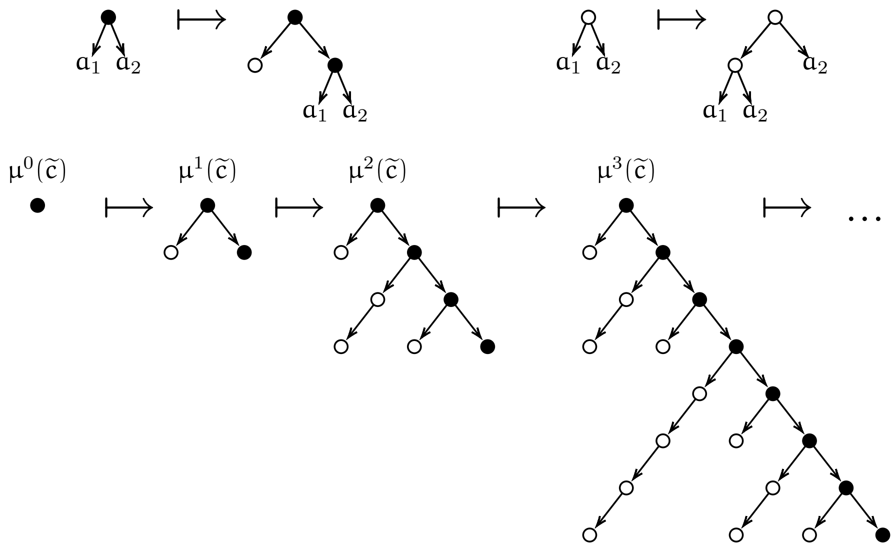


Fig. 3.8. An example of regular tree morphism and its n-fold iteration

distinguished color in C, called *seed*. We consider the n-fold iteration of the tree morphism μ starting from the seed $\widetilde{c}$:

$$\mu^n(\widetilde{c}) = \begin{cases} \widetilde{c} & \text{if } n = 0, \\ \mu^{n-1}(\mu(\widetilde{c})) & \text{if } n > 0. \end{cases}$$

If the color of the root of the replacing tree $F_{\widetilde{c}}$ is exactly the seed $\widetilde{c}$, then it is immediate to see that the sequence $\mu^0(\widetilde{c})$, $\mu^1(\widetilde{c})$, $\mu^2(\widetilde{c})$, ... is (weakly) increasing with respect to the partial order $\sqsubseteq$, namely, $\mu^i(\widetilde{c}) \sqsubseteq \mu^j(\widetilde{c})$ holds for all $i \leqslant j$. Since $\sqsubseteq$ is an ω-complete partial order, the above-defined sequence has a least upper bound, which we shortly denote by $\mu^\omega(\widetilde{c})$. Any tree which is obtained in this way is called *morphic*.

Example 6. Let μ be the tree morphism depicted at the top of Figure 3.8. The sequence of trees obtained from n-fold iterations of μ is shown at the bottom of Figure 3.8. The limit of such a sequence is the morphic tree $\mu^\omega(\widetilde{c})$, whose domain is the set of all words of the form a_2^n or $a_2^n\, a_1^m$, where n ranges over $\mathbb{N}$ and m is less than or equal to the highest power of 2 that divides $n+1$.

A *tree insertion* is a function ν, specified by a tuple of distinct colors $c_1, ..., c_n$ and by an A-labeled C-colored $(C \cup D)$-augmented tree T, which maps any n-tuple $(F_1, ..., F_n)$ of A'-labeled C'-colored $(C' \cup D)$-augmented trees to the A'-labeled C'-colored $(C' \cup D)$-augmented tree

$$\nu(F_1, ..., F_n) = T[\![F_1/c_1, ..., F_n/c_n]\!].$$

Notice that we allow $\{c_1, ..., c_n\}$ to be a proper subset of the set C of all colors (in such a case, the vertices of T which are associated with colors outside $\{c_1, ..., c_n\}$ are not modified by the second-order tree substitution). The tree insertion ν is said to be *regular* if T is regular. As a matter of fact, it comes natural to extend the notion of tree insertion by applying substitutions in more than one tree. Precisely, given an n'-tuple $\bar{\nu}$ of tree insertions $\nu_1, ..., \nu_{n'}$, we can think of $\bar{\nu}$ as function that maps the n-tuple $(F_1, ..., F_n)$ to the n'-tuple $\big(\nu_1(F_1, ..., F_n), ..., \nu_{n'}(F_1, ..., F_n)\big)$. In such a case, we say that $\bar{\nu}$ is a tree insertion *with dimension* (n, n'). By Proposition 29, (regular) tree insertions with matching dimensions can be composed together, namely, for every (regular) tree insertion ν with dimension (n, n') and every (regular) tree insertion ν' with dimension (n', n''), $\nu \circ \nu'$ is a (regular) tree insertion with dimension (n, n'').

3.3.3 Tree Transducers

We now introduce another family of tree transformations, which are specified by top-down deterministic tree transducers [37, 38]. By a slight abuse of terminology, any transformation defined by a tree transducer is called *tree transduction* (in fact, according to [16], these transformations are equivalent to the MSO-definable transductions that preserve bisimilarity of graphs).

Top-down deterministic tree transducers are finite-state machines that process a tree in a top-down fashion and replace the vertex in the current position with a regular tree, which may depend on the current state and color. At each computation step, different states can be spread along different (copies of the) successors of the current vertex. Hence, unlike second-order tree substitutions, a tree transducer may exhibit different behaviors on different copies of the same subtree. In order to distinguish between such different behaviors, we mark some leaves in each replacing tree with pairs belonging to the finite set $B = Q \times A$, making the replacing tree a $(C' \cup D \cup B)$-augmented tree. At each step, the transducer first chooses a replacing tree F on the ground of the state and color of the current vertex v; then, for each $q \in Q$ and each a-successor v_a of v in the input tree, it attaches a copy of the subtree rooted at v_a to every (q, a)-colored leaf of F and it recursively processes that copy starting from state q.

A tree transducer can be formally defined as follows (different, but equivalent, definitions can be found in the literature [37, 38]). Hereafter, given a set Q of states, we use B as a shorthand for the set $Q \times A$ of markers for the leaves of the replacing trees and we denote by $\mathtt{Trees}$, $\mathtt{Trees}'$, and $\mathtt{RegTrees}$ the set of all possibly infinite A-labeled C-colored $(C \cup D)$-augmented trees, the set of all possibly infinite A'-labeled C'-colored $(C' \cup D')$-augmented (where $D' = Q \times D$, as in the case of finite-state recoloring), and the set of all *regular* A'-labeled C'-colored $(C' \cup D \cup B)$-augmented trees, respectively.

Definition 30. *A (top-down deterministic)* tree transducer *is a tuple* $\mathcal{T} = (A, C, D, A', C', Q, \Omega, q_0)$*, where*

- Q *is a finite set of states;*
- $\Omega : Q \times C \rightarrow \texttt{RegTrees}$ *is a replacement function;*
- q_0 *is the initial state.*

The tree transducer $\mathcal{T}$ maps A-labeled C-colored $(C \cup D)$-augmented trees to A'-labeled C'-colored $(C' \cup D')$-augmented trees. In the case of infinite trees, the output of $\mathcal{T}$ is defined as follows, according to [16].

For every $n \geqslant 0$, we recursively define the function $\mathcal{T}^n$ from $Q \times \texttt{Trees}$ to $\texttt{Trees}'$ as follows:

- if $n = 0$ or $T = \emptyset$, then we set $\mathcal{T}^n(q, T) = \emptyset$;
- if $n > 0$, $T \neq \emptyset$, and $T(\varepsilon) \in C$, then we define $\mathcal{T}^n(q, T) = \Omega(q, T(\varepsilon))$ $[T^{(b)}/b]_{b \in B}$, where, for all $b = (r, a) \in B$, $T^{(b)} = \mathcal{T}^{n-1}(r, T^{\downarrow a})$;
- if $n > 0$ and T is the singleton tree d, with $d \in D$, then we let $\mathcal{T}^n(q, T)$ be the singleton D'-colored tree (q, d).

Let us consider now the usual ω-complete partial order $\sqsubseteq$ on the set of all trees. It is easy to verify, by exploiting induction on n, that, for every tree $T \in \texttt{Trees}$, the sequence $\mathcal{T}^0(q_0, T)$, $\mathcal{T}^1(q_0, T)$, $\mathcal{T}^2(q_0, T)$, ... is (weakly) increasing with respect to $\sqsubseteq$. This allows us to define the output $\mathcal{T}(T)$ of the transducer as the limit $\mathcal{T}^\omega(q_0, T)$ of the sequence $\mathcal{T}^0(q_0, T)$, $\mathcal{T}^1(q_0, T)$, $\mathcal{T}^2(q_0, T)$, ... The function that maps a tree T to the tree $\mathcal{T}(T)$ is called *tree transduction.*

Example 7. Let us fix $A = \{1, 2\}$ and $C = \{\wedge, \vee, \neg, p_1, p_2, p_3\}$. Furthermore, let $\mathcal{T} = (A, C, \emptyset, A', C', Q, \Omega, q_0)$ be a tree transducer, where $Q = \{q_0, q_1\}$ and $\Omega(q, c)$ id defined as follows:

$$\Omega(q, c) = \begin{cases} c\langle (q_0, 1), (q_0, 2) \rangle & \text{if } q = q_0 \text{ and } c \in \{\wedge, \vee\}, \\ \vee\langle (q_1, 1), (q_1, 2) \rangle & \text{if } q = q_1 \text{ and } c = \wedge, \\ \wedge\langle (q_1, 1), (q_1, 2) \rangle & \text{if } q = q_1 \text{ and } c = \vee, \\ (q_1, 1) & \text{if } q = q_0 \text{ and } c = \neg, \\ (q_0, 1) & \text{if } q = q_1 \text{ and } c = \neg, \\ c & \text{if } q = q_0 \text{ and } c \in \{p_1, p_2, p_3\}, \\ \neg\langle c, \emptyset \rangle & \text{if } q = q_1 \text{ and } c \in \{p_1, p_2, p_3\}. \end{cases}$$

Intuitively, the transducer $\mathcal{T}$ processes an input tree T by pushing the negation symbol $\neg$ to the leaves through the boolean connectives $\wedge$ and $\vee$. Figure 3.9 shows the computation of the transducer $\mathcal{T}$ on an A-labeled C-colored input tree T that represents the formula $\neg\big(p_1 \vee \neg(p_2 \wedge p_3)\big)$.

Similarly to finite-state recolorings, tree transducers can be enriched with the facility of rational lookahead.

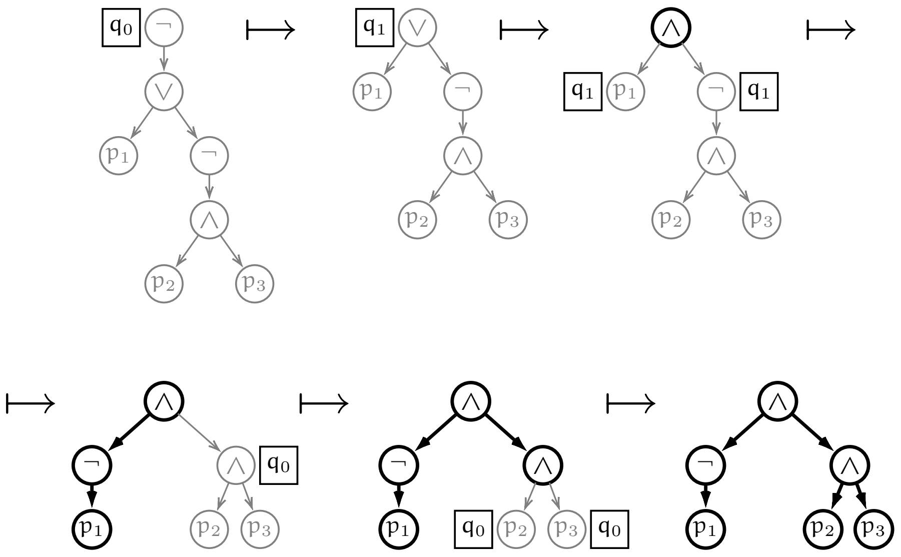


Fig. 3.9. An example of tree transduction

Definition 31. *A* tree transducer with rational lookahead *is a tuple of the form* $\mathcal{T} = (A, C, D, A', C', \mathcal{L}, Q, \Omega, q_0)$*, where*

- $\mathcal{L} = \{L_1, ..., L_k\}$ *is a set of rational tree languages consisting of* A*-labeled* C*-colored* $C \cup D$*-augmented trees;*
- Q *is a finite set of states;*
- Ω *is a replacement function from* $Q \times \mathscr{P}(\{1, ..., k\})$ *to* $\texttt{RegTrees}$*;*
- q_0 *is the initial state.*

For every $n \in \mathbb{N}$, we define the function $\mathcal{T}^n : Q \times \texttt{Trees} \to \texttt{Trees}'$ as follows:

- if $n = 0$ or $T = \emptyset$, then we set $\mathcal{T}^n(q, T) = \emptyset$;
- if $n > 0, T \neq \emptyset$, and $T(\varepsilon) \in C$, then we set $\mathcal{T}^n(q, T) = \Omega(q, E)[T^{(b)}/b]_{b \in B}$, where $E \subseteq \{1, ..., k\}$ is the set of all indices e such that $T \in L_e$;
- if $n > 0$ and T is the singleton tree d, with $d \in D$, then we let $\mathcal{T}^n(q, T)$ be the singleton tree (q, d).

The output $\mathcal{T}(T)$ of the transducer $\mathcal{T}$ on a tree T is defined as the limit $\mathcal{T}^\omega(q_0, T)$ of the $\sqsubseteq$-directed sequence $\mathcal{T}^0(q_0, T)$, $\mathcal{T}^1(q_0, T)$, $\mathcal{T}^2(q_0, T)$, ... The function that maps any tree T to the tree $\mathcal{T}(T)$ is called *tree transduction with rational lookahead*[5].

[5] As pointed out in [37, 38], tree transductions, with or without rational lookahead, are not closed under functional composition, that is, the operation resulting from the composition of two tree transductions is not, in general, a tree transduction. Closure under functional composition can be recovered by imposing suitable restrictions on the replacement function, as in the cases of total and of non-deleting tree transductions [91, 104].

The relationships between tree transductions, finite-state recolorings, and regular tree morphisms are captured by the following proposition.

Proposition 30. *Regular tree morphisms are special cases of tree transductions without lookahead; moreover, finite-state recolorings without lookahead (resp., with rational lookahead) are special cases of tree transductions without lookahead (resp., with rational lookahead). Conversely, any tree transduction without lookahead (resp., with rational lookahead) can be viewed as the composition of a regular tree morphism, a finite-state recoloring without lookahead (resp., with rational lookahead), and another regular tree morphism.*

Proof. We restrict our attention to finite-state recolorings and tree transductions without lookahead (the claims for transformations with rational lookahead can be proved in a similar way).

The proof that tree transductions without lookahead subsume both finite-state recolorings without lookahead and regular tree morphisms is straightforward, and thus omitted.

Let $\mathcal{T} = (A, C, D, A', C', Q, \Omega, q_0)$ be a tree transducer and let $B = Q \times A$. We show that the output of $\mathcal{T}$ on any A-labeled C-colored $(C \cup D)$-augmented tree T can be obtained from T by applying suitable regular tree morphisms and finite-state recolorings. The proof can be decomposed in three steps: (i) we build a regular tree morphism μ_1 that maps T to a B-labeled C-colored $(C \cup D)$-augmented tree $\mu_1(T)$ which is bisimilar to T (up to a relabeling of the edges); (ii) we build a Mealy tree automaton $\mathcal{M}$ that associates with each vertex of $\mu_1(T)$ the corresponding state of the run of $\mathcal{T}$ on T; (iii) we exploit a second regular tree morphism μ_2 to map the tree $\mathcal{M}(\mu_1(T))$ to the tree $\mathcal{T}(T)$.

Let $\{b_1, ..., b_h\}$ be an arbitrary enumeration of all elements of B $(= Q \times A)$, where each b_i is a pair of the form (q_i, a_i), with $q_i \in Q$ and $a_i \in A$, and, for every color $c \in C$, let F_c be the B-labeled C-colored A-augmented tree $c\langle a_1, ..., a_h\rangle$. Furthermore, let μ_1 be the regular tree morphism specified by the tuple $(F_c)_{c \in C}$. Intuitively, such a morphism transforms the tree T into a B-labeled C-colored $(C \cup D)$-augmented tree $\mu_1(T)$ by replacing every a-labeled edge in T with a (q, a)-labeled copy of it, for each $q \in Q$. As a consequence, the tree T is bisimilar to a suitable relabeling of the tree $\mu_1(T)$. Thus, on the ground of the bisimulation relation between T and $\mu_1(T)$, we can associate with each (c-colored) vertex v of $\mu_1(T)$ a corresponding (c-colored) vertex v_A of T. It follows that the tree transducer $\mathcal{T}' = (B, C, D, A', C', Q, \Omega', q_0)$, which is obtained from $\mathcal{T}$ by replacing the label set A with B and the function Ω with Ω' such that $\Omega'(q, c) = \Omega(q, c)[(q', b)/b]_{b=(q',a) \in B}$, satisfies $\mathcal{T}'(\mu_1(T)) = \mathcal{T}(T)$.

Let us now introduce a fresh copy $\widetilde{q}$ of each state $q \in Q$ and we denote by $\widetilde{Q}$ the set of all such states. Furthermore, let $\widetilde{C} = \widetilde{Q} \times C$ and $\widetilde{D} = Q \times D$ (notice that these two sets are disjoint). We define the Mealy tree automaton $\mathcal{M} = (B, C, D, \widetilde{C}, \widetilde{Q}, \widetilde{\delta}, \widetilde{\Omega}, \widetilde{q}_0)$ as follows:

- $\widetilde{\delta}(\widetilde{q}, c, b) = \delta(q, c, b)$ for all $q \in Q$, $c \in C$, and $b \in B$;
- $\widetilde{\Omega}(\widetilde{q}, c) = (\widetilde{q}, c)$ for all $q \in Q$ and $c \in C$.

The Mealy tree automaton $\mathcal{M}$ maps the B-labeled C-colored $(C \cup D)$-augmented tree $\mu_1(T)$ to a suitable B-labeled $\widetilde{C}$-colored $(\widetilde{C} \cup \widetilde{D})$-augmented tree $\mathcal{M}(\mu_1(T))$ by associating to each vertex of $\mu_1(T)$ the corresponding state of the run of $\mathcal{T}'$ on $\mu_1(T)$.

Finally, let μ_2 the regular tree morphism specified by the tuple $\big(\widetilde{F}_{\widetilde{c}}\big)_{\widetilde{c} \in \widetilde{C}}$, where $\widetilde{F}_{\widetilde{c}}$ is the A'-labeled C'-colored $(C' \cup D)$-augmented tree $\Omega(q, c)$, for every $\widetilde{c} = (\widetilde{q}, c) \in \widetilde{C}$, with $q \in Q$ and $c \in C$. On the grounds of the above definitions, for every A-labeled C-colored $(C \cup D)$-augmented tree T, we have

$$\mathcal{T}(T) \;=\; \mathcal{T}'(\mu_1(T)) \;=\; \mathcal{M}(\mu_1(T))\,[\![\widetilde{F}_{\widetilde{c}}/\widetilde{c}]\!]_{\widetilde{c} \in \widetilde{C}} \;=\; \mu_2(\mathcal{M}(\mu_1(T)))\,. \qquad \square$$

3.3.4 Inverse Substitutions

We now consider tree transformations resulting from inverse substitutions followed by unfoldings. First, we recall some basic definitions and then we prove some relevant properties of inverse (rational) substitutions and unfoldings.

An inverse substitution (also called an inverse mapping) [13, 75] is a special form of interpretation, which specifies the edges of the resulting structure by means of suitable path expressions. Intuitively, the application of an inverse substitution to an A-labeled graph G results in a A'-labeled graph G', whose domain coincides with that of G and where (v, v') is an a'-labeled edge of G' iff there exists a path from v to v' in G labeled with a word in a designated language. In the general case, we allow paths to traverse edges in either direction. Thus, given a label $a \in A$, we use the symbol a (resp., the symbol $\bar{a}$) to denote an edge traversed in forward (resp., backward) direction and we introduce a disjoint copy $\bar{A}$ of A, which contains a symbol $\bar{a}$ for any $a \in A$. Hereafter, we denote by $\overleftrightarrow{A}$ the set $A \cup \bar{A}$. Given a finite sequence of vertices $v_1, v_2, v_3, ...$ in a graph G, we say that $v_1, v_2, v_3, ...$ is a *traversal* of G labeled with a word $w \in \overleftrightarrow{A}^*$ iff, for every $1 \leqslant i < n$, one of the following two conditions holds:

1. (v_i, v_{i+1}) is an a-labeled edge of G and $w(i) = a$;
2. (v_{i+1}, v_i) is an a-labeled edge of G and $w(i) = \bar{a}$.

As an example, the word $\bar{a}_1\, a_2$ describes a traversal of a graph G that visits some vertices v_1, v_2, v_3, where (v_2, v_1) is an a_1-labeled edge of G and (v_2, v_3) is an a_2-labeled edge of G. Sometimes, inverse substitutions can also take into account the colors of the traversed vertices. In such a case, traversals are described by words over the expanded alphabet $\overleftrightarrow{A} \cup C$. For the sake of simplicity, we do not consider such a general case.

Let A (resp., A') be the set of edge labels of the input (resp., output) graph of inverse substitutions.

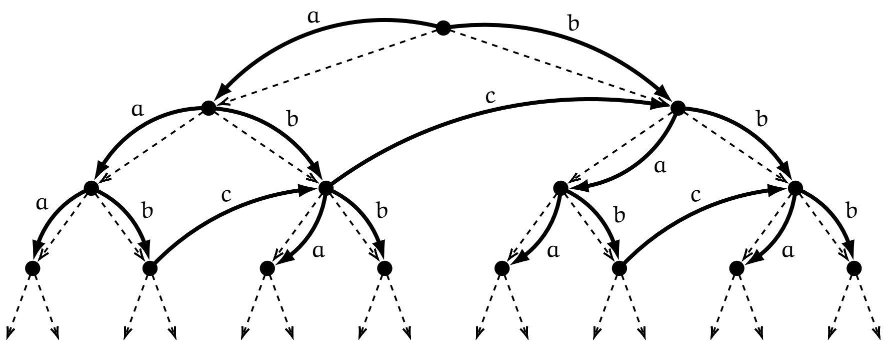


Fig. 3.10. An example of an inverse rational substitution

Definition 32. *Given a graph* $\mathsf{G} = (\mathsf{V}, (\mathsf{E}_a)_{a \in \mathsf{A}})$ *and a function* h *from a set of labels* A' *to the set* $\mathscr{P}(\overleftrightarrow{\mathsf{A}}^*)$*, the* inverse substitution h^{-1} *of* G *is an* A'*-labeled (rooted) graph defined as follows:*

- *the domain of* $h^{-1}(\mathsf{G})$ *is* V*;*
- *if* G *is a rooted graph with root* v_0*, then* v_0 *is the root of* $h^{-1}(\mathsf{G})$ *as well;*
- *for every pair of vertices* v, v' *of* G *and every label* $a' \in \mathsf{A}'$*,* (v, v') *is an* a'*-labeled edge of* $h^{-1}(\mathsf{G})$ *iff* G *contains a traversal from* v *to* v' *labeled with a word* $w \in h(a')$*.*

Since both inverse substitutions and unfoldings preserve graph bisimilarity, it is easy to show that they commute up to bisimulation, namely, for every (rooted) graph G, $\mathcal{U}nf(h^{-1}(\mathsf{G}))$ and $h^{-1}(\mathcal{U}nf(\mathsf{G}))$ are bisimilar.

We say that the function h of Definition 32 is *rational* if, for every label $a' \in \mathsf{A}'$, the language $h(a')$ is rational. Such a notion can be naturally extended to inverse substitutions. As an example, consider the infinite complete $\{1,2\}$-labeled tree T and the rational function $h : \{a, b, c\} \rightarrow \mathscr{P}(\{1, 2, \bar{1}, \bar{2}\}^*)$ defined by $h(a) = \{1\}$, $h(b) = \{2\}$, and $h(c) = \{\bar{2}\,\bar{1}\,2\}$. Figure 3.10 depicts the $\{a, b, c\}$-labeled rooted graph $h^{-1}(\mathsf{T})$ generated from T via the inverse substitution h^{-1} (dashed arrows denote the edges of the underlying tree T, bold arrows denote the edges of $h^{-1}(\mathsf{T})$).

We now focus our attention on particular forms of inverse substitutions, precisely, the inverse flip substitutions and the inverse forward substitutions. Below, # denotes a fresh symbol, not belonging to $\overleftrightarrow{\mathsf{A}}$, and $\overleftrightarrow{\mathsf{A}}_\#$ denotes the set $\overleftrightarrow{\mathsf{A}} \cup \{\#\}$.

The *inverse flip substitution* h^{-1}_{A} transforms a given A-labeled tree T into an $\overleftrightarrow{\mathsf{A}}_\#$-labeled rooted graph by adding, for each a-labeled edge (v, v') of T, one backward $\bar{a}$-labeled edge (v', v) and, for each vertex v of T, one #-labeled loop (v, v). Formally, the inverse flip substitution can be obtained from Definition 32 by letting $\mathsf{A}' = \overleftrightarrow{\mathsf{A}}_\#$, $h(\#) = \{\varepsilon\}$, $h(a) = \{a\}$ for all $a \in \mathsf{A}$, and $h(\bar{a}) = \{\bar{a}\}$ for all $\bar{a} \in \bar{\mathsf{A}}$.

Hereafter, we denote by $\mathcal{F}lip\mathcal{U}nf$ the operation of *unfolding with backward edges and loops* (also called two-way unfolding), which maps any A-labeled tree T to the $\overleftrightarrow{\mathsf{A}}_{\#}$-labeled tree $\mathcal{U}nf(\mathsf{h}_{\mathsf{A}}^{-1}(\mathsf{T}))$ (i.e., the unfolding of the graph $\mathsf{h}_{\mathsf{A}}^{-1}(\mathsf{T})$).

Inverse A*-forward substitutions* are all and only those inverse substitutions where path expressions can only use forward edges and must avoid the empty word. It is immediate to see that inverse substitutions in this restricted class map trees to acyclic graphs.

The following lemma shows that any inverse (rational) substitution is equivalent to an inverse flip substitution followed by an inverse (rational) forward substitution.

Lemma 11. *For every inverse (rational) substitution* h^{-1}*, there is a (rational)* $\overleftrightarrow{\mathsf{A}}_{\#}$*-forward substitution* $\vec{\mathsf{h}}^{-1}$ *such that* $\mathsf{h}^{-1} = \vec{\mathsf{h}}^{-1} \circ \mathsf{h}_{\mathsf{A}}^{-1}$.

Proof. We define the function $\vec{\mathsf{h}}$ as follows: for every $\mathsf{a}' \in \mathsf{A}'$, we set $\vec{\mathsf{h}}(\mathsf{a}') = \{w\# : w \in \mathsf{h}(\mathsf{a}')\}$. It is immediate to see that

$$\mathsf{h}_{\mathsf{A}}\big(\vec{\mathsf{h}}(\mathsf{a}')\big) \;=\; \mathsf{h}_{\mathsf{A}}\big(\{w\# : w \in \mathsf{h}(\mathsf{a}')\}\big) \;=\; \big\{\mathsf{h}_{\mathsf{A}}(w\#) : w \in \mathsf{h}(\mathsf{a}')\big\} \;=\; \mathsf{h}(\mathsf{a}').$$

This proves that $\mathsf{h}^{-1} = \vec{\mathsf{h}}^{-1} \circ \mathsf{h}_{\mathsf{A}}^{-1}$.
Clearly, $\vec{\mathsf{h}}^{-1}$ is rational if and only if h^{-1} is rational. □

By exploiting Lemma 11, we obtain the following result.

Proposition 31. *For every inverse (rational) substitution* h^{-1}*, there is an inverse (rational)* $\overleftrightarrow{\mathsf{A}}_{\#}$*-forward substitution* $\vec{\mathsf{h}}^{-1}$ *such that, for every* A*-labeled tree* T*, the tree* $\mathcal{U}nf(\mathsf{h}^{-1}(\mathsf{T}))$ *and the graph* $\vec{\mathsf{h}}^{-1}(\mathcal{F}lip\mathcal{U}nf(\mathsf{T}))$ *are bisimilar.*

Proof. Given an inverse (rational) substitution h^{-1}, by Lemma 11, there is an inverse (rational) forward substitution $\vec{\mathsf{h}}^{-1}$ such that, for every A-labeled tree T, $\mathsf{h}^{-1}(\mathsf{T}) = \vec{\mathsf{h}}^{-1}(\mathsf{h}_{\mathsf{A}}^{-1}(\mathsf{T}))$. Since inverse (rational) forward substitutions and unfoldings commute up to bisimulation, the tree $\mathcal{U}nf\big(\vec{\mathsf{h}}^{-1}(\mathsf{h}_{\mathsf{A}}^{-1}(\mathsf{T}))\big)$ is bisimilar to the graph $\vec{\mathsf{h}}^{-1}\big(\mathcal{U}nf(\mathsf{h}_{\mathsf{A}}^{-1}(\mathsf{T}))\big)$ $(= \vec{\mathsf{h}}^{-1}(\mathcal{F}lip\mathcal{U}nf(\mathsf{T})))$. □

We conclude the section by showing that inverse rational forward substitutions can be implemented by tree transducers with rational lookahead.

Proposition 32. *For every inverse rational* A*-forward substitution* h^{-1}*, there is a tree transducer* $\mathcal{T}$ *with rational lookahead such that, for every* A*-labeled tree* T*, if* $\mathsf{h}^{-1}(\mathsf{T})$ *is a deterministic (acyclic) graph, then* $\mathcal{T}(\mathsf{T})$ *is the (unique) deterministic tree bisimilar to* $\mathsf{h}^{-1}(\mathsf{T})$.

Proof. First, observe that any inverse rational forward substitution can be viewed as a particular form of MSO-definable interpretation which preserves

$$\begin{tikzcd}
 & \begin{array}{c}\text{finite-state recolorings}\\ \text{+ regular tree morphisms}\end{array} \arrow[d] & \begin{array}{c}\text{finite-state recolorings}\\ \text{w. rational lookahead}\\ \text{+ regular tree morphisms}\end{array} \arrow[l] \arrow[d] & \\
\begin{array}{c}\text{regular tree}\\ \text{morphisms}\end{array} \arrow[d] & \begin{array}{c}\text{tree}\\ \text{transductions}\end{array} \arrow[l] \arrow[d] & \begin{array}{c}\text{tree transductions}\\ \text{w. rational lookahead}\end{array} \arrow[l] \arrow[r] \arrow[d] & \begin{array}{c}\text{inverse rational}\\ \text{forward}\\ \text{substitutions}\end{array} \\
\begin{array}{c}\text{memoryless}\\ \text{recolorings}\end{array} & \begin{array}{c}\text{finite-state}\\ \text{recolorings}\end{array} \arrow[l] & \begin{array}{c}\text{finite-state recolorings}\\ \text{w. rational lookahead}\end{array} \arrow[l] &
\end{tikzcd}$$

Fig. 3.11. Relationships between the various tree transformations

bisimilarity of graphs [108]. Indeed, the existence of a path labeled by a word in a designated rational language can be expressed by a suitable formula in the monadic second-order logic. Therefore, the claim of the proposition follows from a result by Colcombet and Löding [16], which shows that any MSO-definable interpretation which preserves bisimilarity is equivalent (up to bisimulation) to a suitable tree transduction with rational lookahead.

3.3.5 A Summary

In Figure 3.11, we graphically represent the relationships between the various pairs of tree transformations. An arrow departing from a (set of) operation(s) X and reaching a (set of) operation(s) Y means that Y can be expressed in terms of (a composition of operations of) X, or, shortly, X subsumes Y.

3.4 The Class of Reducible Trees

In this section, we introduce the notion of rank, which can be thought of as the number of iterated retractions which are sufficient to reduce a given tree to a regular one, and we prove some closure properties for the classes of trees having rank n, for $n \in \mathbb{N}$.

Given the results of Section 3.2, we know that the acceptance problem for a tree T is reducible to the acceptance problem for the encoding $\vec{T}$ of a retraction of it. As a matter of fact, the encoding $\vec{T}$ may be a regular tree and hence enjoy a decidable acceptance problem. On the one hand, it comes natural to study how many iterated retractions are sufficient to reduce a given tree T to a regular one. On the other hand, if we do not restrict the form of the factorizations which are used to define retractions of trees, we easily see that any tree is reducible to a regular retraction in a single (trivial) step. Given a tree T and an automaton $\mathcal{A}$ running on T, we can indeed denote by Π the factorization of T such that $\mathcal{D}om(\Pi) = \{\varepsilon\}$. Such a factorization induces a retraction $\widetilde{T}$ of T that consists of a single vertex colored with the minor $\mathcal{A}$-type of T. The encoding $\overrightarrow{T}$ of $\widetilde{T}$ is obviously a regular tree, hence enjoying a decidable acceptance problem. Such an attempt to identify a class of decidable trees is interesting neither from an algorithmic point of view nor from a combinatorial point of view, since the decidability of the acceptance problem is reduced to the computability of a singleton retraction. In order to find a reasonable notion of rank for trees, we need to restrict the form of the factorizations in such a way that the minor types of the marked factors are guaranteed to be computable. The natural choice is to enforce the condition that the marked factors resulting from a factorization of a tree are *regular*. In such a case, we say that the factorization is regular.

Definition 33. *Let T be a D-augmented tree. We say that T has rank 0 if it is regular. For $n > 0$, we say that T has rank n if there is a set $B \supseteq D$ and a B-labeled regular factorization Π of T such that, for every B-augmented tree automaton $\mathcal{A}$, the encoding of the retraction of T with respect to $\mathcal{A}$ and Π has rank $n - 1$. We say that T is* reducible *if it has a finite rank $n \in \mathbb{N}$.*

From Definition 33 and Theorem 6, the acceptance problem for any reducible tree turns out to be decidable, provided that there is an effective way to compute (a finite representation) of the encoding of an iterated retraction of T. Let the *footprint* of a tree T be the minimum amount of information that must be provided to make the reduction from T to the encoding of its retraction effective. Such a footprint can be inductively defined as follows. Given a D-augmented tree T, if T has rank 0, then a footprint of T is a finite rooted graph whose unfolding is isomorphic to T. If T has rank $n > 0$, then a footprint of T is a pair $\xi = (B, f)$, where $B \supseteq D$ and f is a computable function that maps any B-augmented tree automaton $\mathcal{A}$ to a footprint of a suitable B-labeled $(\mathscr{T}_{\mathcal{A}} \cup \{\perp\})$-colored tree $\overrightarrow{T}$ which has rank $n-1$ and encodes a retraction of T with respect to $\mathcal{A}$. Equivalently, a footprint of a D-augmented tree with rank $n > 0$ can be finitely presented as a pair consisting of a set $B \supseteq D$ and a program that, given any B-augmented tree automaton $\mathcal{A}$, computes a footprint of a suitable B-labeled $(\mathscr{T}_{\mathcal{A}} \cup \{\perp\})$-colored tree $\overrightarrow{T}$ which has rank $n - 1$ and encodes a retraction of T with respect to $\mathcal{A}$.

As pointed out in Section 3.2, factorizations may depend on automata. To keep the notation as simple as possible, for every reducible tree we fix a single factorization, which does not depend on the automata running on it. Whenever necessary, however, such a restriction can be safely removed by adopting more general notions of reducible tree and footprint. Hereafter, we restrict ourselves to reducible trees which are modeled according to any suitable representation system that allows the computation of their footprints. Under such a restriction, we have the following result.

Theorem 7. *Reducible trees have decidable acceptance problems.*

Proof. Let T be a tree having rank $n > 0$ and let $\xi = (B, f)$ be its footprint. For every B-augmented tree automaton $\mathcal{A}$, we define (i) a sequence of sets of markers $B_0, B_1, ..., B_n$, (ii) a sequence of augmented tree automata $\mathcal{A}_0, \mathcal{A}_1, ..., \mathcal{A}_n$, and (iii) a sequence of footprints $\xi_0, \xi_1, ..., \xi_n$ as follows:

- $B_0 = B$, $\mathcal{A}_0 = \mathcal{A}$, and $\xi_0 = \xi$;
- for every $0 \leqslant i < n$, B_{i+1} is the first component of the footprint ξ_i;
- for every $0 \leqslant i < n$, $\xi_{i+1} = f_i(\mathcal{A}_i)$, where f_i is the second component of the footprint ξ_i;
- $\mathcal{A}_1$ is the retraction automaton of $\mathcal{A}_0$ and, for every $1 \leqslant i < n$, $\mathcal{A}_{i+1}$ is the retraction automaton of the B_i-augmented tree automaton $\mathcal{A}_i[B_i]$ obtained by expanding the retraction automaton $\mathcal{A}_i$ (recall that $\mathcal{A}_i$ is a 'bare' alternating tree automaton running on an infinite complete tree) with the set of markers B_i and with the set $\mathcal{G} = \emptyset$.

For every $0 \leqslant i \leqslant n$, ξ_i is a footprint of a tree having rank $n - i$. In particular, ξ_n is a footprint of a regular tree (that is, a finite rooted graph). Moreover, from Theorem 5, T is accepted by $\mathcal{A}$ iff the unfolding of ξ_n is accepted by $\mathcal{A}_n$. Finally, from Proposition 26, the problem of deciding whether the unfolding of ξ_n is accepted by $\mathcal{A}_n$ is decidable and this concludes the proof. $\square$

It is easy to see that the class of all reducible trees properly includes the class of all profinitely (or residually) ultimately periodic words [12]. Similarly, it is possible to prove that the class of all reducible trees properly includes the class of profinitely (or residually) ultimately periodic trees [72]. The latter inclusion shows the generality of the notion of factorization proposed in Section 3.2.3, which allows reductions not only to linear structures, but also to branching ones. More precisely, let us call a factorization that satisfies the definition given in [72] a *linear factorization.* A linear factorization of a tree T can be viewed as a factorization Π of T (according to Definition 21), which further satisfies the following condition: for every pair of paths π and π' in Π, labeled with the words u and u', respectively, if u and u' have the same length and terminate with the same symbol, then the marked factors of T associated with u and u' coincide. Moreover, the definition proposed in [72] associates with each linear factorization an ordinal n, called *level*, which takes into account the form of its factors. For instance, all factors in a level 1 linear factorization must be regular trees, while those in a level 2 linear factorization must be decomposable into level 1 linear factorizations. It can be shown that there exist simple trees which do not admit any linear factorization (of any level). Consider, for instance, the A-labeled C-colored tree T, with $A = \{a, b, c\}$ and $C = \{0, 1\}$, defined as follows:

- the domain of T is the set of all finite words over the alphabet A;
- for every vertex v of T, $T(v) = 1$ iff v is a word of the form $u\,(c\,u)^n$, for some $u \in \{a, b\}^*$ and $n \in \mathbb{N}$.

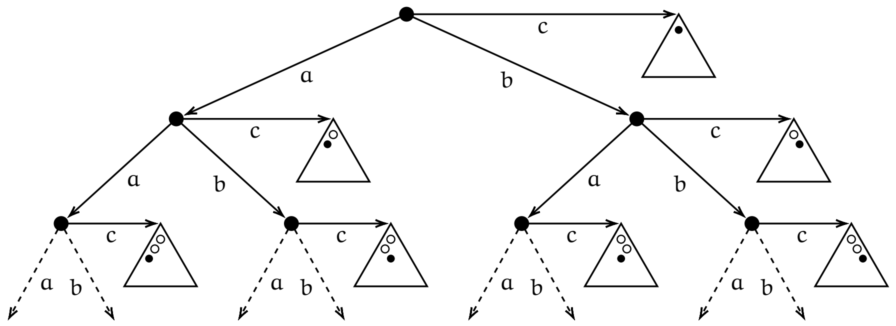


Fig. 3.12. A reducible tree which is not profinitely ultimately periodic

Figure 3.12 depicts a portion of the tree T, where the white-colored circles and the black-colored circles represent, respectively, the 0-colored vertices and the 1-colored vertices of T. Clearly, all subtrees of T rooted at the vertices $u \in \{a, b\}^*$ are pairwise non-isomorphic. Due to such a particular structure, there exists no linear factorization of T of any level. However, by exploiting the compositional properties of types provided in the next section, one can easily prove that T has rank 1. As a matter of fact, one can also prove that T belongs to the third level of the Caucal hierarchy.

The inductive definition of rank of a tree induces a hierarchical structure on the class of all reducible trees. Establishing whether or not such a hierarchy is *strictly increasing*, namely, whether or not there exists an upper bound $n \in \mathbb{N}$ on the minimum rank of all reducible trees, is an open (and difficult) problem. As a matter of fact, from the results given in [90], it is clear that every *unary* tree, i.e., every word, with a decidable MSO theory has rank 1. We expect that this does not hold anymore for the class of binary trees with a decidable MSO theory. Clearly, the above mentioned problem is also related to the problem of determining the minimum rank of any given tree. For such a reason, whenever we say that a tree T has rank n, we actually mean that its minimum rank is less than or equal to n.

3.4.1 Compositional Properties of Types

In the following, we prove a composition theorem for types with respect to second-order tree substitutions. More precisely, we relate the types of a tree T and the types of a tuple $(F_c)_{c\in C}$ of replacing trees to the types of the tree $T[\![F_c/c]\!]_{c\in C}$. From now on, for any given automaton $\mathcal{A}$, we shortly denote by $\tau_{\mathcal{A}}$ the function that maps any tree to its minor $\mathcal{A}$-type.

As in the definition of second-order tree substitution, we fix a set A (resp., A') for the edge labels of the input tree (resp., the output tree), a set C (resp., C') for the colors of the internal vertices of the input tree (resp., the output tree), and a set $C \cup D$ (resp., $C' \cup D$) for the colors of the leaves of the input tree (resp., the output tree). As usual, we assume that C is disjoint from D and $\mathsf{D} \supseteq \mathsf{A}$. Moreover, by a slight abuse of terminology, given a D-augmented tree automaton $\mathcal{A}$ and a tuple $\bar{\mathsf{F}} = (\mathsf{F}_1, ..., \mathsf{F}_n)$ of A'-labeled C'-colored $(\mathsf{C}' \cup \mathsf{D})$-augmented replacing trees, we say that $\bar{\sigma} = (\sigma_1, ..., \sigma_n)$ is the minor $\mathcal{A}$-type of $\bar{\mathsf{F}}$, if, for every $1 \leqslant i \leqslant n$, σ_i is the minor $\mathcal{A}$-type of F_i; then we accordingly extend the function $\tau_{\mathcal{A}}$.

The following theorem resembles the composition theorems for MSO logics (see [95, 106, 89]). Unlike them, however, it involves the operation of second-order tree substitution instead of simpler operations like the juxtaposition of words [95, 106] or the appending of trees at the root [89].

Theorem 8. *Let $\mathcal{A}'$ be a $(\mathsf{C}' \cup \mathsf{D})$-augmented tree automaton and let $\bar{\sigma} = (\sigma_c)_{c \in C}$ be a tuple of minor $\mathcal{A}'$-types. One can compute a $(\mathsf{C} \cup \mathsf{D})$-augmented tree automaton $\mathcal{A}$ such that, for every tree T and every tuple $\bar{\mathsf{F}} = (\mathsf{F}_c)_{c \in C}$ of replacing trees, the minor $\mathcal{A}'$-type σ' of $\mathsf{T}' = \mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in C}$ is uniquely determined by (and computable from) the minor $\mathcal{A}$-type σ of T and the minor $\mathcal{A}'$-type $\bar{\sigma}$ of $\bar{\mathsf{F}}$.*

We describe the key ingredients of the proof of Theorem 8 and we refer the interested reader to Appendix A.2 for the technical details. To start with, we distinguish between two kinds of second-order tree substitution, namely, *erasing* and *non-erasing* ones. A second-order tree substitution $\mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in C}$ is said to be erasing if there is at least one color $c \in \mathsf{C}$ such that the A'-labeled C'-colored $(\mathsf{C}' \cup \mathsf{D})$-augmented tree F_c is either empty or consists of a single vertex marked with an element $a \in \mathsf{A}$. In such a case, the color c (resp., the replacing tree F_c) is said to be an *erasing color* (resp., an *erasing tree*) of the second-order tree substitution. Given an *erasing* second-order tree substitution $\mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in C}$, we then denote by C^- the set of all erasing colors in C and by C^+ the set of all other colors. If we further have that $\mathsf{A}' = \mathsf{A} = \{a_1, ..., a_k\}$, $\mathsf{C}' = \mathsf{C}$, and, for every non-erasing color $c \in \mathsf{C}^+$, $\mathsf{F}_c = c\langle a_1, ..., a_k\rangle$, namely, if the substitution preserves all non-erasing colors without adding new nodes, then say that the substitution is *shrinking*. It is easy to see that for every (possibly erasing) second-order tree substitution $\mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in C}$, if we define

$$\mathsf{F}^-_c = \begin{cases} \mathsf{F}_c & \text{if } c \in \mathsf{C}^- \\ c\langle a_1, ..., a_k\rangle & \text{if } c \in \mathsf{C}^+ \end{cases} \qquad \text{and} \qquad \mathsf{F}^+_c = \begin{cases} \mathsf{F}_c & \text{if } c \in \mathsf{C}^+ \\ c\langle a_1, ..., a_k\rangle & \text{if } c \in \mathsf{C}^- \end{cases},$$

then we have

$$\mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in C} = \mathsf{T}[\![\mathsf{F}^-_c/c]\!]_{c \in C}[\![\mathsf{F}^+_c/c]\!]_{c \in C}. \tag{3.1}$$

The first substitution in the right-hand side of Equation 3.1 is shrinking and the second one is non-erasing. Such a property allows us to prove Theorem 8 separately for *shrinking* substitutions (see Lemma 18 in Appendix A.2) and for *non-erasing* ones (see Lemma 19 in Appendix A.2). As a matter of fact, the proof of the compositional property for non-erasing second-order tree substitutions is a rather easy application of Theorem 6. Theorem 8 can thus be thought of as a sort of generalization of Theorem 6.

The following corollary relates the minor $\mathcal{A}'$-types of the input trees of a regular tree insertion to the minor $\mathcal{A}'$-types of the output trees.

Corollary 1. *Let $\mathcal{A}'$ be a $(C' \cup D)$-augmented tree automaton and let ν be a regular tree insertion with dimension (n, n'). One can compute a function $\vec{\nu} : \mathscr{T}^{n}_{\mathcal{A}'} \to \mathscr{T}^{n'}_{\mathcal{A}'}$ such that, for every n-tuple $\bar{F}$ of replacing trees,*

$$\vec{\nu}\big(\tau_{\mathcal{A}'}(\bar{F})\big) = \tau_{\mathcal{A}'}\big(\nu(\bar{F})\big).$$

Proof. The claim follows trivially from Theorem 8. The regular tree insertion ν is indeed specified by an n-tuple of colors $c_1, ..., c_n$ and by a *regular* tree T. Therefore, given any $(C' \cup D)$-augmented tree automaton $\mathcal{A}'$, we can use Theorem 8 to compute the $(C \cup D)$-augmented tree automaton $\mathcal{A}$ and then Proposition 26 and Proposition 28 to compute the minor $\mathcal{A}$-type σ of T. From previous results, we know that the function $\vec{\nu}$ is computable. □

The function $\vec{\nu}$ of Corollary 1 is called the $\mathcal{A}'$-*abstraction* of the regular tree insertion ν. Let us denote by $\mathcal{A}'$-`Abstractions` the set of all $\mathcal{A}'$-abstractions of regular tree insertions with dimension $(1, 1)$ and let us equip it with the associative operation of functional composition $\circ$. It turns out that ($\mathcal{A}'$-`Abstractions`, $\circ$) is a *finite monoid*:

i) the set $\mathscr{T}_{\mathcal{A}'}$ (and hence the set $\mathcal{A}'$-`Abstractions`) is finite;
ii) from Proposition 29, it easily follows that $\mathcal{A}'$-abstractions of regular tree insertions are closed under functional composition;
iii) there exists a regular tree insertion ν_{id} which maps any tree F to itself and hence the $\mathcal{A}'$-abstraction $\vec{\nu}_{id}$ plays the role of the identity in ($\mathcal{A}'$-`Abstractions`, $\circ$).

In the following, we consider iterated applications of regular tree insertions. In particular, we focus on properties of the so-called profinitely ultimately periodic functions, which include those functions $f : \mathbb{N} \to \mathbb{N}$ such that, for every $\mathcal{A}$-abstraction $\vec{\nu}$ of a regular tree insertion, the sequence $\vec{\nu}^{f(0)}, \vec{\nu}^{f(1)}, \vec{\nu}^{f(2)}, ...$ turns out to be ultimately periodic. Roughly speaking, profinitely ultimately periodic functions exhibit a (computable) repeating pattern whenever they are projected into any finite monoid, e.g., ($\mathcal{A}$-`Abstractions`, $\circ$). Examples of such functions are n^2, 2^n, $2^n - n^2$, n^n, $n!$, and the exponential tower $2^{2^{\cdot^{\cdot^{\cdot^2}}}}$. Hereafter, for every $p \geqslant 0$, every $q > 0$, and every $n \geqslant 0$, we denote by $[n]_{p,q}$ either the value n or the value $((n - p) \bmod q) + p$, depending on whether $n < p$ or $n \geqslant p$.

Definition 34. *A function $f : \mathbb{N} \to \mathbb{N}$ is said to be* profinitely ultimately periodic *if, given $l \geqslant 0$ and $r > 0$, one can compute $p \geqslant 0$ and $q > 0$ such that, for every $n \geqslant 0$,*

$$\big[f(n)\big]_{l,r} = \big[f([n]_{p,q})\big]_{l,r}. \tag{3.2}$$

Every function f that satisfies Equation 3.2 can be characterized in terms of the periodicity of the sequences of the form $(e^{f(n)})_{n \in \mathbb{N}}$, where e is any element of a finite (multiplicative) monoid. This is formally stated in Proposition 33. As a matter of fact, such a characterization implies that the notion of profinitely ultimately periodic function is actually equivalent to (i) the notion of profinitely ultimately periodic sequence (see [12]), and (ii) the notion of ultimately periodic function with respect to finite semigroups/monoids (see [114, 115, 72, 73]).

Proposition 33. *A function* $f : \mathbb{N} \to \mathbb{N}$ *satisfies Definition 34 iff, given any finite (multiplicative) monoid* $(X, \cdot)$ *and any element* $e \in X$*, one can compute* $p \geqslant 0$ *and* $q > 0$ *such that, for all* $n \in \mathbb{N}$*,*

$$e^{f(n)} = e^{f([n]_{p,q})},$$

namely, the sequence $(e^{f(n)})_{n \in \mathbb{N}}$ *is (effectively) ultimately periodic.*

Proof. Let us assume that f satisfies Definition 34. Given any finite monoid $(X, \cdot)$ and any element $e \in X$, by exploiting the Pigeonhole Principle, one can compute two natural numbers $l \geqslant 0$ and $r > 0$ such that $e^n = e^{[n]_{l,r}}$ for every $n \in \mathbb{N}$. Since f satisfies Definition 34, one can compute two natural numbers $p \geqslant 0$ and $q > 0$ such that, for all $n \in \mathbb{N}$, $[f(n)]_{l,r} = [f([n]_{p,q})]_{l,r}$. Thus, we have

$$e^{f(n)} = e^{[f(n)]_{l,r}} = e^{[f([n]_{p,q})]_{l,r}} = e^{f([n]_{p,q})}.$$

For the converse implication, let $f : \mathbb{N} \to \mathbb{N}$ be a function such that, for every finite monoid $(X, \cdot)$ and every element $e \in X$, one can compute $p \geqslant 0$ and $q > 0$ such that, for all $n \in \mathbb{N}$, $e^{f(n)} = e^{f([n]_{p,q})}$. Let us fix two natural numbers $l \geqslant 0$ and $r > 0$. One can choose a finite monoid $(X, \cdot)$ and an element $e \in X$ such that r turns out to be the *least* integer such that e^r is the identity of X. Thus, one can compute two natural numbers $p \geqslant 0$ and $q > 0$ such that, for every $n \in \mathbb{N}$, $e^{f(n)} = e^{f([n]_{p,q})}$. Notice that, by construction, for any $i, j \in \mathbb{N}$, $e^i = e^j$ implies $i \bmod r = j \bmod r$, whence $[i]_{l,r} = [j]_{l,r}$. Therefore, for every $n \in \mathbb{N}$, we have

$$\big[f(n)\big]_{l,r} = \big[f([n]_{p,q})\big]_{l,r}. \qquad \square$$

The following proposition identifies a number of ways to obtain profinitely ultimately periodic functions, starting from a set of basic functions. It extends previous results from [12]. Hereafter, we use $i = j \pmod m$ as a shorthand for $i \bmod m = j \bmod m$. Moreover, we say that a function $f : \mathbb{N} \to \mathbb{N}$ has *unbounded infimum* if $\liminf_{n \to \infty} f(n) = \infty$ (in such a case, we tacitly assume that for any given $l \in \mathbb{N}$, one can compute an index n_0 such that, for every $n \geqslant n_0$, $f(n) \geqslant l$ holds).

Proposition 34. *Let* f *and* g *be two profinitely ultimately periodic functions. The following functions are profinitely ultimately periodic as well:*

1. **(Sum)** $h = f + g$, *defined by* $h(n) = f(n) + g(n)$*;*
2. **(Product)** $h = f \cdot g$, *defined by* $h(n) = f(n) \cdot g(n)$*;*
3. **(Difference)** $h = f - g$, *defined by* $h(n) = f(n) - g(n)$, *provided that* h *has unbounded infimum;*
4. **(Quotient)** $h = \left\lfloor \frac{f}{d} \right\rfloor$, *defined by* $h(n) = \left\lfloor \frac{f(n)}{d} \right\rfloor$, *where* d *is any positive constant;*
5. **(Exponentiation)** $h = f^g$, *defined by* $h(n) = \big(f(n)\big)^{g(n)}$, *provided that* h *has unbounded infimum;*
6. **(Exponential tower)** h *defined by* $h(0) = 1$ *and* $h(n+1) = b^{h(n)}$, *where* b *is any positive constant;*
7. **(Fibonacci numbers)** h *defined by* $h(0) = h(1) = 1$ *and* $h(n + 2) = h(n) + h(n + 1)$*;*
8. **(Generalized sum)** h *defined by* $h(n) = \sum_{i=0}^{n-1} f(i)$*;*
9. **(Generalized product)** h *defined by* $h(n) = \prod_{i=0}^{n-1} f(i)$*;*
10. **(Composition)** $h = f \circ g$, *defined by* $h(n) = g(f(n))$.

In order to not interrupt the flow of the presentation, the rather straightforward proof of Proposition 34 is postponed to Appendix A.3.

We conclude the section by showing how to apply Proposition 34 to a tree, which does not belong to the Caucal hierarchy.

Example 8. Let *tow* be the tower of exponentials, defined by $tow(0) = 1$ and $tow(n + 1) = 2^{tow(n)}$ for all $n \in \mathbb{N}$. We denote by T the $\{a_1, a_2\}$-labeled tree whose domain consists of all and only the finite words of the form $a_2^n\, a_1^m$, with $n \in \mathbb{N}$ and $m \leqslant tow(n)$ (see the upper left part of Figure 3.13). We assume that all vertices of T are colored over a singleton $\{c\}$. In [11], Carayol and Wöhrle show that such a tree enjoys a decidable MSO theory, even though it does not belong to the Caucal hierarchy. Here, we give an alternative (automaton-based) proof of the decidability of the MSO theory of T. We define the B-labeled factorization Π of T, with $B = \{a_2\}$, as follows:

- the domain of Π is the set of all words of the form a_2^n, with $n \in \mathbb{N}$ (the vertices of Π are represented by circled nodes in Figure 3.13)';
- (u, v) is an a_2-labeled edge of Π iff u is a word of the form a_2^n, for some $n \in \mathbb{N}$, and v is the word a_2^{n+1}.

Now, let F be the singleton tree c, let G be the $\{c, x\}$-colored $\{a_2\}$-augmented tree $c\langle x, a_2\rangle$ (see the upper right part of Figure 3.13), and let v be the regular tree insertion specified by the tree G (x is the color to be replaced by the input of v). Notice that, for every $n \in \mathbb{N}$, the marked factor of T rooted at the vertex a_2^n is isomorphic to the tree $v^{tow(n)}(F)$, i.e., the $tow(n)$-fold application

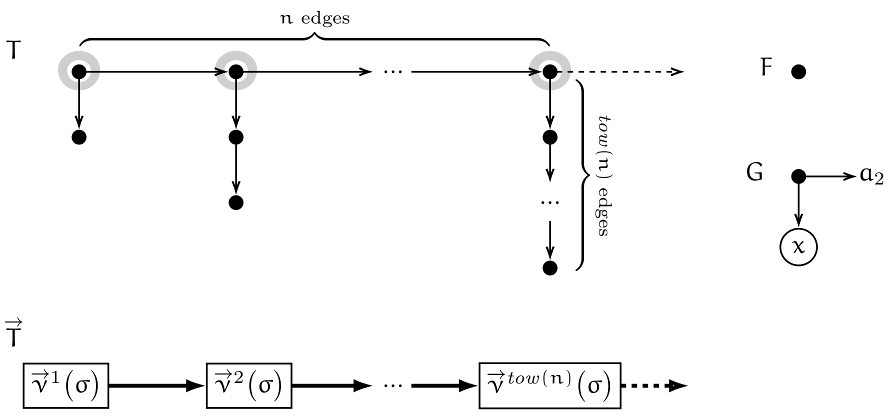


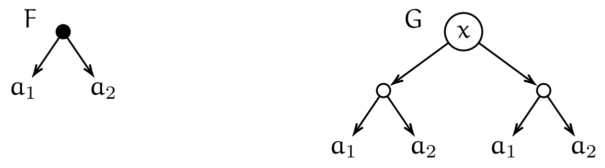
Fig. 3.13. An example of tree outside the Caucal hierarchy

of v to F. Then, given any B-augmented tree automaton $\mathcal{A}$, let $\vec{\mathsf{v}}$ be the $\mathcal{A}$-abstraction of v and σ be the minor $\mathcal{A}$-type of F. From Corollary 1, we have that $\vec{\mathsf{v}}^{tow(\mathsf{n})}(\sigma)$ is the minor $\mathcal{A}$-type of the marked factor of T rooted at the vertex $\mathsf{a}_2^{\mathsf{n}}$. This implies that the infinite complete B-labeled $\mathscr{T}_{\mathcal{A}}$-colored tree $\vec{\mathsf{T}}$ defined by

$$\vec{\mathsf{T}}(\mathsf{a}_2^{\mathsf{n}}) = \vec{\mathsf{v}}^{tow(\mathsf{n})}(\sigma),$$

depicted at the bottom of Figure 3.13, encodes the retraction of T with respect to $\mathcal{A}$ and Π. Moreover, from Proposition 34, the function *tow* is profinitely ultimately periodic and hence $\vec{\mathsf{T}}$ is a regular tree. Therefore, the tree T has rank 1 and it enjoys a decidable MSO theory.

As a matter of fact, the argument exploited in Example 8 can be easily generalized to several trees outside the Caucal hierarchy, like, for instance, the tree T' depicted in Figure 3.14. In such a case, one can show that the tree T' has rank 1 by first defining two basic trees F and G as follows



and then identifying some marked factors F_n inside T' that satisfy the equation $\mathsf{F}_\mathsf{n} = \mathsf{v}^{tow(\mathsf{n})-1}(\mathsf{F})$ for all $\mathsf{n} \in \mathbb{N}$, where v is the regular tree insertion specified by G.

3.4.2 Closure Properties

For every $\mathsf{n} \in \mathbb{N}$, we say that a class $\mathscr{C}$ of trees of rank n is effectively closed under a family $\mathscr{F}$ of tree transformations if, for every tree $\mathsf{T} \in \mathscr{C}$ and every

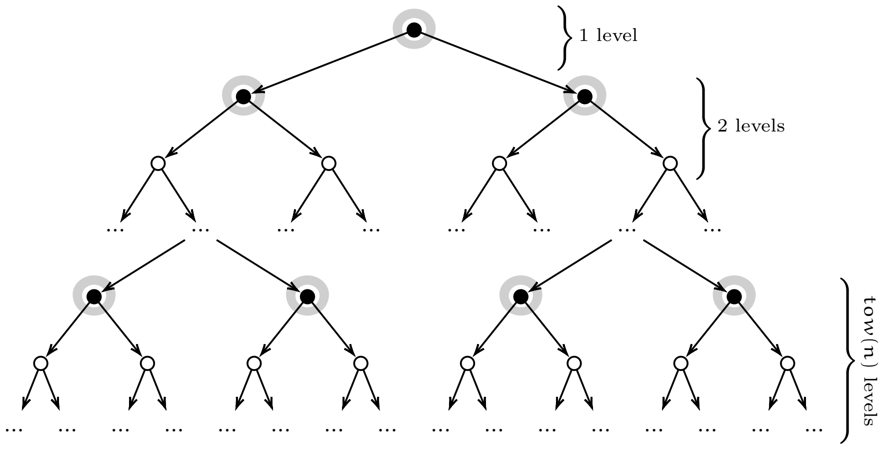


Fig. 3.14. Another example of tree outside the Caucal hierarchy

transformation $f \in \mathscr{F}$, the application of f to T results in a tree $T' \in \mathscr{C}$, whose footprint is computable on the grounds of the footprint of T. In this section, we prove that, for every $n \in \mathbb{N}$, the class of trees of rank n is effectively closed under tree transductions with rational lookahead. We also prove that the class of all reducible trees is effectively closed with respect to unfoldings with backward edges and loops.

First of all, we give an intuitive account of the proof of the closure property with respect to tree transductions with rational lookahead. It is by induction on the rank of the involved trees and it exploits the fact that any tree transduction with rational lookahead is equivalent to a functional composition of suitable regular tree morphisms and finite-state recolorings with rational lookahead (see Proposition 30). The class of all trees of rank 0, i.e., regular trees, is easily proved to be closed under both regular tree morphisms and finite-state recolorings with rational lookahead. As for the inductive step, we fix a tree T of rank $n > 0$ and a regular tree morphism μ (resp., a finite-state recoloring with rational lookahead $\mathcal{M}$) and we let $T' = \mu(T)$ (resp., $T' = \mathcal{M}(T)$). Then, given an automaton $\mathcal{A}'$ running on T', we build (i) an automaton $\mathcal{A}$ running on T, (ii) a tree $\overrightarrow{T}$ of rank $n-1$ and encoding a retraction of T with respect to the automaton $\mathcal{A}$, and (iii) a regular tree morphism $\overrightarrow{\mu}$ (resp., a tree transduction with rational lookahead $\overrightarrow{\mathcal{T}}$) that maps the infinite complete tree $\overrightarrow{T}$ to a suitable infinite complete tree $\overrightarrow{T}'$, which encodes a retraction of T' with respect to $\mathcal{A}'$. By exploiting the inductive hypothesis, it turns out that $\overrightarrow{T}'$ has rank $n-1$ and hence T' has rank n. As a matter of fact, such an argument allows us to interpret the MSO-compatibility of tree transductions with rational lookahead, as well as several other transformations of trees, in terms of computability of footprints.

For the sake of simplicity, in this section we restrict our attention to infinite complete trees (we can easily get rid of such a restriction, because, for every $n \in \mathbb{N}$, the class of trees of rank n is effectively closed under completions with $\perp$-colored vertices).

Lemma 12. *For every* $n \in \mathbb{N}$*, the class of all trees of rank* n *is effectively closed under regular tree morphisms.*

Proof. The proof of the base case ($n = 0$) can be found in Section 4.1 of [20].

As for the inductive case, let us fix (i) a positive natural number n, (ii) an infinite complete A-labeled C-colored tree T of rank n, (iii) a footprint $\xi = (B, f)$ of T, and (iv) a regular tree morphism μ specified by a tuple $\bar{F} = (F_c)_{c \in C}$ of A'-labeled C'-colored A-augmented trees. Without loss of generality, we can assume that $B \supseteq A$. We first show that the infinite complete A'-labeled C'-colored tree $T' = \mu(T)$ has rank n. Suppose that we are given a B-augmented tree automaton $\mathcal{A}'$ and the minor $\mathcal{A}'$-type $\bar{\sigma} = (\sigma_c)_{c \in C}$ of $\bar{F}$. From Theorem 8, we can compute a B-augmented tree automaton $\mathcal{A}$ and a function $\vec{\mu} : \mathscr{T}_{\mathcal{A}} \rightarrow \mathscr{T}_{\mathcal{A}'}$ such that, for every A-labeled C-colored B-augmented tree F, if σ is the minor $\mathcal{A}$-type of F, then $\vec{\mu}(\sigma)$ is the minor $\mathcal{A}'$-type of $\mu(F)$. We then partition the set of all A-labeled C-colored B-augmented trees into $2 + |B|$ different classes:

1. the language L_0 consisting of all B-augmented trees whose image under the regular tree morphism μ is the empty tree;
2. the language L_1 consisting of all B-augmented trees whose image under the regular tree morphism μ is a non-empty non-singleton tree;
3. the language L_b, for each $b \in B$, consisting of all B-augmented trees whose image under the regular tree morphism μ is the singleton tree b.

All the above languages are rational; moreover, they can be recognized by suitable *deterministic* B-augmented tree automata. Without loss of generality, we can assume that the automaton $\mathcal{A}$ is able to distinguish between the above languages of trees, namely, for every pair of B-augmented trees F and F' having the same minor $\mathcal{A}$-type, $F \in L_i$ iff $F' \in L_i$, for all $i \in \{0,1\} \cup B$ (for any given automaton $\mathcal{A}$, which does not satisfy such a condition, we can indeed produce another automaton that satisfies it, which can be obtained from $\mathcal{A}$ by properly refining its state space).

It easily follows that we can compute a function $\vec{\gamma} : \mathscr{T}_{\mathcal{A}} \rightarrow \{0,1\} \cup B$ such that, for every B-augmented tree F, if σ is the minor $\mathcal{A}$-type of F, then $\vec{\gamma}(\sigma)$ is the index $i \in \{0,1\} \cup B$ of the (unique) language L_i that contains F. Now, we extend the function $\vec{\mu}$ to a regular tree morphism which maps infinite complete B-labeled $(\mathscr{T}_{\mathcal{A}} \cup \{\perp\})$-colored trees to infinite complete B-labeled $(\mathscr{T}_{\mathcal{A}'} \cup \{\perp\})$-colored trees as follows. Let $B = \{b_1, ..., b_h\}$ and let $T_\emptyset$ be the infinite complete B-labeled $\perp$-colored tree. For every symbol $\sigma \in \mathscr{T}_{\mathcal{A}} \cup \{\perp\}$, we define

$$\vec{\mu}(\sigma\langle b_1, ..., b_h\rangle) = \begin{cases} T_\emptyset & \text{if } \sigma = \perp \text{ or } \vec{\gamma}(\sigma) = 0, \\ \vec{\mu}(\sigma)\langle b_1, ..., b_h\rangle & \text{if } \sigma \neq \perp \text{ and } \vec{\gamma}(\sigma) = 1, \\ b & \text{if } \sigma \neq \perp \text{ and } \vec{\gamma}(\sigma) = b \in B \end{cases}$$

Let us show now that the regular tree morphism $\vec{\mu}$ maps the encoding of a B-labeled retraction of T with respect to $\mathcal{A}$ to the encoding of a suitable B-labeled retraction of T' with respect to $\mathcal{A}'$. Let Π be a B-labeled factorization of T such that the encoding $\vec{T}$ of the retraction of T with respect to $\mathcal{A}$ and Π has rank $n{-}1$. First, we have to provide a suitable B-labeled factorization Π' of T' such that each marked factor of T' with respect to Π' can be viewed as the image, under the regular tree morphism μ, of a corresponding marked factor of T with respect to Π. The factorization Π' is a straightforward generalization of the notion of *induced* factorization for a given non-erasing second-order tree substitution (see the proof of Lemma 19 in Appendix A.2). Precisely, for every vertex v of T, we recursively define the set V_v of vertices of T' that correspond to v as follows:

$$V_v = \begin{cases} \{\varepsilon\} & \text{if } v = \varepsilon, \\ V_u\,\{w \in \mathcal{F}r(F_c) \,:\, F_c(w) = a\} & \text{if } v \,=\, u\,a \text{ and } T(u) = c. \end{cases}$$

We define Π' as follows:

- the domain of Π' is the union, over all vertices v of Π, of the sets V_v;
- (u', v') is a b-labeled edge of Π' iff Π contains a b-labeled edge (u, v) such that $u' \in V_u$ and $v' \in V_v$.

For every vertex v' of Π', there is at least one vertex v of Π such that $v' \in V_v$. Moreover, for every pair of distinct vertices u, v of Π, the sets V_u and V_v are either disjoint or coincide. In the latter case, u is either an ancestor or a descendant of v in T and, for every color c that occurs along the path from u to the predecessor of v in T, or from v to the predecessor of u in T, F_c is a singleton A-augmented tree. Thus, for every vertex v' of Π', we can define the corresponding vertex v of Π as the *least* vertex (according to the order given by the ancestor relation of Π) among all vertices u of Π such that $v' \in V_u$ (by the previous argument, all such vertices lie on the same path). We let the reader check that, for every vertex v' of Π', if v is the vertex of Π that corresponds to v' and F (resp., F') is the marked factor of T (resp., T') rooted at v (resp., v'), then $F' = \mu(F)$ and hence $\tau_{\mathcal{A}'}(F') = \vec{\mu}(\tau_{\mathcal{A}}(F))$. Given the encoding $\vec{T}$ of the retraction of T with respect to $\mathcal{A}$ and Π, the tree $\vec{T}' = \vec{\mu}(\vec{T})$ encodes the corresponding retraction of T' with respect to $\mathcal{A}'$ and Π'. Moreover, by the inductive hypothesis, we have that $\vec{T}'$ has rank $n{-}1$ and hence T' has rank n.

To conclude the proof, we must show how to compute a footprint ξ' of T' on the grounds of the given footprint $\xi = (B, f)$ of T. We simply let $\xi' = (B, f')$, where f' is the function that maps any given B-augmented tree automaton $\mathcal{A}'$ to a footprint $\vec{\xi}'$ of $\vec{T}'$, which, by the inductive hypothesis, is computable on the grounds of the footprint $\vec{\xi} = f(\mathcal{A})$ of $\vec{T}$. □

Lemma 13. *For every $n \in \mathbb{N}$, the class of all trees of rank n is effectively closed under finite-state recolorings with rational lookahead.*

Proof. We first prove the base case ($n = 0$). Let T be an infinite complete A-labeled C-colored regular tree and let (G, v_0) be a finite rooted graph whose unfolding is isomorphic to T. Furthermore, let $\mathcal{M} = (A, C, , C', \mathcal{L}, Q, \delta, \Omega, q_0)$ be a Mealy tree automaton with rational lookahead, where $\mathcal{L} = \{L_1, ..., L_k\}$. We have to build a finite rooted graph (G', v'_0) whose unfolding produces the infinite complete A-labeled C'-colored tree $T' = \mathcal{M}(T)$. We define the graph G' as follows:

- the domain of G' consists of all pairs of the form (v, q), with v being a vertex of G and q being a state of $\mathcal{M}$;
- for every c-colored vertex v of G, with $c \in C$, and every state $q \in Q$, we let $\Omega(q, E)$ be the color of the vertex (v, q) of G', where E is the set of all indices e such that the language L_e contains the tree $\mathcal{U}nf(G, v)$;
- for every pair of vertices v, v' of G, every pair of states q, q' of $\mathcal{M}$, and every label $a \in A$, we let (v', q') be an a-successor of (v, q) in G' iff (i) (v, v') is an a-labeled edge of G and (ii) $\delta(q, E, a) = q'$, where E is the set of all indices e such that the language L_e contains the tree $\mathcal{U}nf(G, v)$.

The unfolding of G' from the vertex $v'_0 = (v_0, q_0)$ is isomorphic to the tree T'.

As for the inductive step, let us fix (i) a positive natural number n, (ii) an infinite complete A-labeled C-colored tree T of rank n, (iii) a footprint $\xi = (B, f)$ of T, and (iv) a Mealy tree automaton with rational lookahead $\mathcal{M} = (A, C, \emptyset, C', \mathcal{L}, Q, \delta, \Omega, q_0)$. To show that the infinite complete A-labeled C'-colored tree $T' = \mathcal{M}(T)$ has rank n, we follow the same path of the proof of Lemma 12: first, given an automaton $\mathcal{A}'$ running on T', we build a suitable automaton $\mathcal{A}$ running on T; then, we show how to obtain the encoding $\vec{T}'$ of a retraction of T' with respect to $\mathcal{A}'$ (and some factorization Π') by applying a suitable transformation to the encoding $\vec{T}$ of a retraction of T with respect to $\mathcal{A}$ (and some factorization Π). However, two major difficulties arise with such a proof. First, since the transformation computed by the Mealy tree automaton $\mathcal{M}$ is not local, we have that isomorphic factors of T may be transformed by $\mathcal{M}$ into non-isomorphic factors of T'. Thus, the transformation that maps $\vec{T}$ to $\vec{T}'$ may not preserve the out-degree of the vertices (for such a reason, it will be defined as a tree transduction, that is, as the composition of a regular tree morphism and a finite-state recoloring with rational lookahead). Second, the Mealy tree automaton $\mathcal{M}$ is equipped with the facility of rational lookahead, which makes the color of each vertex in the output tree T' dependent on the colors of the descendants of the corresponding vertex in the input tree T. This makes the definition of the automaton $\mathcal{A}$ rather involved, since $\mathcal{A}$ must correctly mimic any computation of $\mathcal{A}'$ on T', while running on the tree T. To this end, we allow the automaton $\mathcal{A}$ to guess the colors of the vertices of T' and to subsequently check whether the guess was correct (this requires the use of alternation in the transition function of $\mathcal{A}$).

Let P denote the (unique) run of $\mathcal{M}$ on T. For any B-labeled factorization Π of T, we define a corresponding B'-labeled factorization Π' of T', where $B' = Q \times B$, as follows:

- $\mathcal{D}om(\Pi) = \mathcal{D}om(\Pi')$;
- (u, v) is a (q, b)-labeled edge of Π' iff (u, v) is a b-labeled edge of Π and $q = P(v)$, namely, q is the state of $\mathcal{M}$ at the vertex v of T.

Let $\mathcal{A}' = (A, C', B', S', \Delta', \mathcal{I}', \mathcal{F}', \mathcal{G}')$ be a generic B'-augmented tree automaton, which runs on the marked factors of T'. For the sake of simplicity, we assume that the set $\mathcal{L} = \{L_1, ..., L_k\}$, containing the lookahead languages of $\mathcal{M}$, forms a partition of the set of all infinite complete A-labeled C-colored trees (relaxing such a simplification is trivial). Let $E = \{1, ..., k\}$ and, for each index $e \in E$, let $\mathcal{A}_e = (A, C, B, S_e, \Delta_e, \mathcal{I}_e, \mathcal{F}_e, \mathcal{G}_e)$ be a B-augmented tree automaton that recognizes L_e (we assume that the state sets S_e, for all $e \in E$, are disjoint). We define the B-augmented tree automaton $\mathcal{A} = (A, C, B, S, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$, which runs on the marked factors of T, as follows:

- $S = (S' \times Q) \,\cup\, \bigcup_{e \in E} S_e$;
- for every state $s = (s', q) \in S' \times Q$ and every color $c \in C$,

$$\Delta(s, c) = \bigvee_{e \in E} (\Phi_{s,c,e} \,\wedge\, \Psi_{c,e})$$

  where $\Phi_{s,c,e}$ is obtained by replacing every atom of the form $\langle a, s'' \rangle$ in $\Delta'(s', \Omega(q, e))$ with $\langle a, (s'', \delta(q, e, a)) \rangle$ and $\Psi_{c,e}$ is the disjunction, over all $s_0 \in \mathcal{I}_e$, of $\Delta_e(s_0, c)$;
- for every index $e \in E$, every state $s \in S_e$, and every color $c \in C$,

$$\Delta(s, c) = \Delta_e(s, c);$$

- $\mathcal{I} = S' \times Q$;
- $\mathcal{F} = \mathscr{P}(S' \times Q) \,\cup\, \bigcup_{e \in E} \mathcal{F}_e$;
- $\mathcal{G} = \mathscr{P}(B \times (S' \times Q)) \,\cup\, \bigcup_{e \in E} \mathcal{G}_e$.

Notice that the acceptance condition of $\mathcal{A}$ envisages only the occurrences of those states that belong to S_e, for some $e \in E$ (the other states are used to mimic the computations of $\mathcal{A}'$ on T'). Moreover, it is easy to see that the (unique) run P of $\mathcal{M}$ on T can be obtained from *any successful* run R of $\mathcal{A}$ on T that starts from a state of the form (s, q_0) by first selecting the branches of R that involve only the states in $S' \times Q$ and then projecting them into their second component. Similarly, every run R' of $\mathcal{A}'$ on T' can be obtained from a suitable successful run R of $\mathcal{A}$ on T by first selecting the branches of

R that involve only the states in $S' \times Q$ and then projecting then into their first component.

In the following, we show how to lift the above properties to the level of retractions. More precisely, we want to find an effective correspondence between the minor $\mathcal{A}$-types of the marked factors of T and the minor $\mathcal{A}'$-types of the marked factors of T'. Since the transformation computed by $\mathcal{M}$ is non-local, such a correspondence must consider both the positions of the marked factors in T and the $\mathcal{L}$-classification of the subtrees rooted at their leaves. Let us denote by $\vec{E}$ the set $\{\perp\} \cup (S \times \mathscr{P}(\mathscr{P}(S)) \times \mathscr{P}(B \times S \times \mathscr{P}(S)))$, which contains the dummy symbol $\perp$ and all possible features of the automaton $\mathcal{A}$. For every $t \in \vec{E}$, we define the language $\vec{L}_t$ as follows:

1. if $t = \perp$, then $\vec{L}_t$ is the language that consists of a single infinite complete B-labeled $\{\perp\}$-colored tree;
2. if $t \neq \perp$, then $\vec{L}_t$ is the language of all infinite complete B-labeled $(\mathscr{T}_{\mathcal{A}} \cup \{\perp\})$-colored trees $\vec{H}$ such that (i) $t \in \vec{H}(\varepsilon)$, namely, t is a feature contained in the minor $\mathcal{A}$-type $\vec{H}(\varepsilon)$, and (ii) the retraction automaton $\vec{\mathcal{A}}$ of $\mathcal{A}$ accepts the tree $\vec{H}_t$ obtained from $\vec{H}$ by recoloring its root with the singleton $\{t\}$.

Notice that the above languages are rational. Now, let $\vec{T}$ (resp., $\vec{T}'$) be the encoding of the retraction of T (resp., T') with respect to $\mathcal{A}$ and Π (resp., $\mathcal{A}'$ and Π'). For every vertex $v \in \mathcal{D}om(\Pi)(= \mathcal{D}om(\Pi'))$, we denote by $\vec{v}$ (resp., $\vec{v}'$) the corresponding vertex of $\vec{T}$ (resp., $\vec{T}'$), which is recursively defined by $\vec{v} = \vec{v}' = \varepsilon$ for $v = \varepsilon$ and $\vec{v} = \vec{u}\,b$ (resp., $\vec{v}' = \vec{u}'\,b'$) for every b-labeled edge (u, v) of Π (resp., for every b'-labeled edge (u, v) of Π'). It is routine to verify that, for every vertex v of Π, if

$$t = \begin{pmatrix} s \\ \{F_i \;:\; i \in I\} \\ \{(b_j,\, s_j,\, G_j) \;:\; j \in J\} \end{pmatrix}$$

is a feature of $\mathcal{A}$ on the marked factor of T rooted at v (equivalently, if $t \in \vec{T}(\vec{v})$) that satisfies

i) $s \in S' \times Q$;

ii) $\downarrow_2 s = P(v)$, namely, the state appearing in the second component of s coincides with the state at the vertex v of the run P of $\mathcal{M}$ on T;

iii) $\vec{T}^{\downarrow \vec{v}} \in \vec{L}_t$, namely, the language $\vec{L}_t$ contains the subtree of $\vec{T}$ rooted at $\vec{v}$,

then the triple

$$t' = \begin{pmatrix} \downarrow_1 s \\ \{\downarrow_1 F_i \;:\; i \in I, F_i \subseteq S' \times Q\} \\ \{((\downarrow_2 s_j, b_j),\, \downarrow_1 s_j,\, \downarrow_1 G_j) \;:\; j \in J, G_j \subseteq S' \times Q\} \end{pmatrix}$$

is a feature of $\mathcal{A}'$ on the marked factor of T' rooted at v, that is, $t' \in \vec{T}'(\vec{v}'))$. In addition, we have that the following two conditions hold:

i) for every $j \in J$ such that $G_j \subseteq S' \times Q$, the marked factor of T rooted at v contains a leaf w such that $P(vw) = \downarrow_2 s_j$;

ii) for every leaf w of the marked factor of T rooted at v, there is an index $j \in J$ such that $G_j \subseteq S' \times Q$ and $\downarrow_2 s_j = P(vw)$.

To complete the proof, it suffices to provide a suitable tree transducer with rational lookahead $\vec{\mathcal{T}}$ that implements the above correspondence between features $t \in \vec{T}(\vec{v})$ and features $t' \in \vec{T}'(\vec{v}')$. Such a transducer, which only depends on $\mathcal{A}'$ (and $\mathcal{M}$), is defined as the tuple $(B, \mathscr{T}_{\mathcal{A}} \cup \{\perp\}, \emptyset, B', \mathscr{T}_{\mathcal{A}'} \cup \{\perp\}, \vec{\mathcal{L}}, \vec{Q}, \vec{\Omega}, \vec{q}_0)$, where

- $\vec{\mathcal{L}}$ is the set of all languages $\vec{L}_t$, for t ranging over $\vec{E}$;
- $\vec{Q} = Q \cup \{q_\perp\}$, where $q_\perp$ is a fresh state;
- for every state $\vec{q} \in \vec{Q}$ and every subset $\vec{E}'$ of $\vec{E}$,

$$\vec{\Omega}(\vec{q}, \vec{E}') = \begin{cases} \perp\langle m_{q,b}\rangle_{(q,b)\in B'} & \text{if } \perp \in \vec{E}' \text{ or } \vec{q} = q_\perp, \\ \sigma'\langle m_{q,b}\rangle_{(q,b)\in B'} & \text{if } \perp \notin \vec{E}' \text{ and } \vec{q} \neq q_\perp, \end{cases}$$

  where

  i) for every $(q, b) \in B'$, $m_{q,b}$ is either the pair (q, b) or the pair $(q_\perp, b)$, depending on whether or not $\vec{E}'$ contains a feature of the form $t = (s, \{F_i\}_{i\in I}, \{(b_j, s_j, G_j)\}_{j\in J})$, with $b_j = b$ and $\downarrow_2 s_j = q$ for some $j \in J$,

  ii) σ' is the set of all triples

$$\left(\begin{array}{c} \downarrow_1 s \\ \big\{ \downarrow_1 F_i \ : \ i \in I, F_i \subseteq S' \times Q \big\} \\ \big\{ \big((\downarrow_2 s_j, b_j),\ \downarrow_1 s_j,\ \downarrow_1 G_j \big) \ : \ j \in J, G_j \subseteq S' \times Q \big\} \end{array} \right)$$

  for $t = (s, \{F_i\}_{i\in I}, \{(b_j, s_j, G_j)\}_{j\in J})$ ranging over $\vec{E}'$ and satisfying $s \in S' \times Q$ and $\downarrow_2 s = \vec{q}$;
- $\vec{q}_0 = q_0$.

It is easy to show that the transduction is correct, that is, that $\vec{\mathcal{T}}$ actually computes the encoding $\vec{T}'$ of a retraction of T', with respect to $\mathcal{A}'$ and Π', from the encoding of a retraction $\vec{T}$ of T, with respect to $\mathcal{A}$ and Π.

From the inductive hypothesis and from Lemma 12 and Proposition 30, we have that $\vec{T}' = \vec{\mathcal{T}}(\vec{T})$ has rank $n - 1$ (and thus T' has rank n), whenever $\vec{T}$ has rank $n - 1$. It remains to show how to compute a footprint ξ' of T' on the grounds of the given footprint $\xi = (B, f)$ of T. We simply let $\xi' = (B', f')$, where f' is the function that maps any given B'-augmented tree automaton

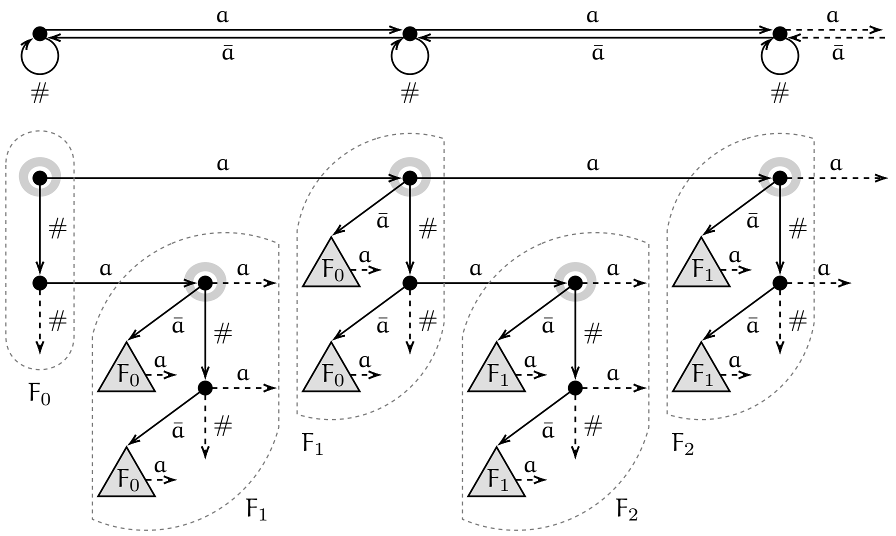


Fig. 3.15. The two-way unfolding of the semi-infinite line

$\mathcal{A}'$ to a footprint $\vec{\xi}'$ of $\vec{\mathsf{T}}'$, which, by the inductive hypothesis (plus Lemma 12 and Proposition 30), is computable from the footprint $\vec{\xi} = \mathsf{f}(\mathcal{A})$ of $\vec{\mathsf{T}}$. □

The following theorem states the general closure property for the class of trees of rank n.

Theorem 9. *For every $n \in \mathbb{N}$, the class of all trees of rank n is effectively closed under tree transductions with rational lookahead.*

Proof. This is a consequence of Lemma 12, Lemma 13, and Proposition 30. □

We conclude the section by showing that the class of all reducible trees is effectively closed under unfoldings with backward edges and loops. We start with the well-known example of the unfolding of the semi-infinite line with backward edges and loops (see, for instance, [7]). Such an example, which uses the compositional results given in Section 3.4.1, should provide a rough idea of how the general closure property is proved.

Example 9. Let $\mathsf{L} = (\mathbb{N}, \mathsf{E}_a, \mathsf{E}_{\bar{a}}, \mathsf{E}_{\#})$ be the semi-infinite line equipped with a-labeled forward edges, $\bar{a}$-labeled backward edges, and #-labeled loops (see the upper part of Figure 3.15). Let T be the unfolding of L from the leftmost vertex. The lower part of Figure 3.15 depicts the tree T, where, for each $n \in \mathbb{N}$, F_n denotes the unfolding from the rightmost vertex of the subgraph L_n obtained by restricting L to set of vertices $\{0, ..., n\}$. We give an alternative proof of the decidability of the acceptance problem (and hence the decidability of the MSO theory) of T. The basic idea is to recursively define the components

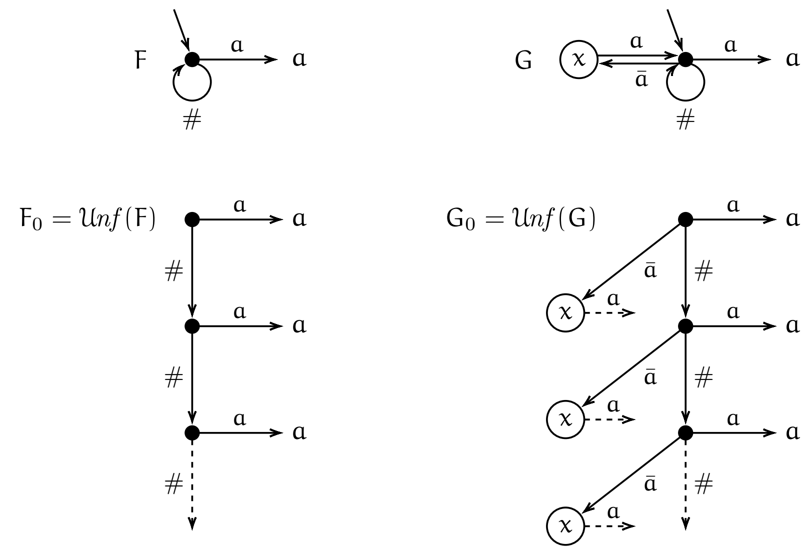


Fig. 3.16. Marked factors for the two-way unfolding of the semi-infinite line

$F_0, F_1, F_2, \ldots$ of T and then exploit such a characterization to build regular retractions of T. By construction, every vertex v of T corresponds to a unique path π_v in L. We denote by u_v the last vertex along the path π_v in L and we define the B-labeled factorization Π of T, with $B = \{a\}$, as follows:

- the domain of Π is the set of all vertices v of T such that there is no proper ancestor v' of v for which $u_v = u_{v'}$ (the set $\mathcal{D}om(\Pi)$ is represented in Figure 3.15 by circled nodes);
- the resulting edges of Π are labeled with the unique symbol a.

Notice that the vertices of Π have an infinite out-degree. However, for every pair of vertices u, u' in Π, if the access path of u and that of u' in Π have the same length, then the marked factor of T rooted at u and the marked factor of T rooted at u' are isomorphic. This allows us to identify access paths in Π having the same length. Hence, for any given B-augmented tree automaton $\mathcal{A}$, the retraction of T with respect to $\mathcal{A}$ and Π turns out to be bisimilar to a regular deterministic B-labeled $\mathscr{T}_{\mathcal{A}}$-colored tree. More precisely, we can denote by F and G the two graphs depicted in the upper part of Figure 3.16 and by F_0 and G_0 their unfoldings, which are depicted in the lower part. Furthermore, we denote by ν the regular tree insertion specified by the tree G_0 (x is the color to be replaced by the input of ν). It is easy to see that, for every vertex u of Π at distance n from the root, the marked factor of T rooted at u is isomorphic to the tree $\nu^n(F_0)$ (i.e., the n-fold application of ν to F_0). Now, given any

B-augmented tree automaton $\mathcal{A}$, we denote by $\vec{\mathsf{v}}$ the $\mathcal{A}$-abstraction of v and we define the infinite complete B-labeled $\mathscr{T}_{\mathcal{A}}$-colored tree $\vec{\mathsf{T}}$ as follows:

$$\vec{\mathsf{T}}(\mathsf{u}) = \begin{cases} \tau_{\mathcal{A}}(\mathsf{F}_0) & \text{if } \mathsf{u} = \varepsilon, \\ \vec{\mathsf{v}}(\vec{\mathsf{T}}(\mathsf{u}')) & \text{if } \mathsf{u} = \mathsf{u}'\, \mathsf{a}. \end{cases}$$

From Corollary 1, we know that, for every vertex v of Π at distance n from the root, $\vec{\mathsf{T}}(\mathsf{a}^{\mathsf{n}})$ is the minor $\mathcal{A}$-type of the marked factor of T rooted at v and hence $\vec{\mathsf{T}}$ encodes the retraction of T with respect to $\mathcal{A}$ and Π. Moreover, since $\mathscr{T}_{\mathcal{A}}$ is a finite set, from the Pigeonhole Principle, we have that $\vec{\mathsf{T}}$ is a regular tree. This shows that T has rank 1 and hence it enjoys a decidable acceptance problem.

The following theorem is the natural generalization of the above example to the class of all reducible trees.

Theorem 10. *For every tree T of rank n, the tree $\mathcal{FlipUnf}(\mathsf{T})$, i.e., the unfolding of T with backward edges and loops, has rank $\mathsf{n}+1$ and hence the class of all reducible trees is effectively closed under $\mathcal{FlipUnf}$.*

Proof. Example 9 basically shows that any retraction of the unfolding of the semi-infinite line L with backward edges and loops is encoded by a suitable regular tree, which results from the application of a finite-state recoloring to L (hence, from Theorem 9, it has rank 0). Not surprisingly, the same proof technique can be applied to any tree T of rank n in order to show that its unfolding with backward edges and loops has rank $\mathsf{n}+1$. Let us fix (i) a positive natural number n and (ii) an infinite complete A-labeled C-colored tree T of rank n. Furthermore, let G be the $(\mathsf{A} \cup \bar{\mathsf{A}} \cup \{\#\})$-labeled C-colored rooted graph obtained by adding $\bar{\mathsf{A}}$-labeled backward edges and $\#$-labeled loops to T and let $\mathsf{T}' = \mathcal{Unf}(\mathsf{G}) = \mathcal{Unf}(\mathsf{h}_{\bar{\mathsf{A}}}^{-1}(\mathsf{T})) = \mathcal{FlipUnf}(\mathsf{T})$, namely, T' be the unfolding of G from its root. Notice that every vertex v' of T' identifies a path in G (equivalently, a traversal in T) that starts from the root and reaches a vertex v of G (equivalently, a vertex v of T). We express such a relationship by saying that the vertex v corresponds to v'. We now define an A-labeled factorization Π' of T' as follows:

- $\mathcal{D}om(\Pi')$ is the set of all vertices v' of T' for which there is no proper ancestor u' of v' in T' such that $\mathsf{u} = \mathsf{v}$, where u and v are the vertices of T that corresponds, respectively, to u' and v';
- $(\mathsf{u}', \mathsf{v}')$ is an a-labeled edge of Π' iff (u, v) is an a-labeled edge of T, where u and v are the vertices of T that corresponds, respectively, to u' and v'.

Even though the vertices of Π' have infinite out-degree, for every pair of vertices u', v' of Π', if the sequences of labels along the access paths of u' and v' in Π' are the same, then, since the corresponding traversals in T lead to the same vertex, we have that the two subtrees $(\mathsf{T}')^{\downarrow \mathsf{u}'}$ and $(\mathsf{T}')^{\downarrow \mathsf{v}'}$ are actually

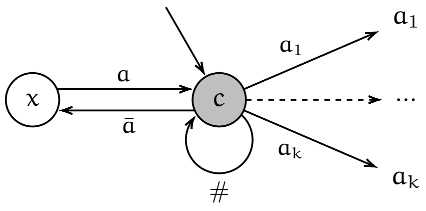


Fig. 3.17. The (c, a)-slice

isomorphic and hence the two marked factors of T' rooted at u' and v' are isomorphic as well.

In the following, we prove that, for any given A-augmented tree automaton $\mathcal{A}$, the encoding of the retraction of T' with respect to $\mathcal{A}$ and Π' has rank n, which immediately implies that T' has rank $n+1$. As in the case of the semi-infinite line, the basic idea is to recursively characterize the form of the marked factor of T' rooted at a vertex v' of Π' on the grounds of the marked factor of T' rooted at the parent u' of v' in Π'. We introduce a fresh color x, not belonging to C, and, for every $c \in C$ and $a \in A$, we define the finite rooted A-labeled $(A \cup C \cup \{x\})$-colored graph $G_{c,a}$, called (c, a)-*slice*, as follows (see Figure 3.17):

- $G_{c,a}$ contains a single c-colored vertex v_c, a single x-colored vertex v_x, and one a-colored vertex v_a for each $a \in A$;
- the root of $G_{c,a}$ is the vertex v_c;
- $G_{c,a}$ contains a single $\#$-labeled loop (v_c, v_c), two a-labeled edges (v_x, v_c) and (v_c, v_a), a single $\bar{a}$-labeled edge (v_c, v_x), one a'-labeled edge $(v_c, v_{a'})$ (but no $\bar{a}'$-labeled edges) for each label $a' \neq a$ in A.

For the sake of brevity, we denote by $A' = A \cup \bar{A} \cup \{\#\}$ the set of the edge labels of T'. Each (c, a)-slice $G_{c,a}$ defines a regular tree insertion $\nu_{c,a}$ that maps any A'-labeled C-colored A-augmented tree F to the second-order tree substitution of all x-colored vertices in the unfolding of $G_{c,a}$ by F (notice that the result of such a substitution is an A'-labeled C-colored A-augmented tree). Clearly, if $T(\varepsilon) = c$, then, for any choice of $a \in A$, the marked factor of T' associated with the root of Π' is isomorphic to the tree $\nu_{c,a}(\emptyset)$, where $\emptyset$ denotes the empty tree. Moreover, if (u', v') is an a-labeled edge of the factorization Π', v is the vertex of T that corresponds to v', and $T(v) = c$, then the marked factor of T' rooted at v' is the result of the application of $\nu_{c,a}$ to the marked factor of T' rooted at u'. This basically means that, for every vertex v' of Π', the marked factor of T' rooted at v' only depends on the sequence of edge labels along the access path of v' in Π' and on the sequence of vertex colors along the corresponding path in T. Now, let $\mathcal{A}'$ be an A-augmented tree automaton. By Corollary 1, for every $c \in C$ and $a \in A$, we can denote by $\vec{\nu}_{c,a}$ the $\mathcal{A}'$-abstraction of $\nu_{c,a}$, which maps the minor $\mathcal{A}'$-type of any

A-augmented tree F to the minor $\mathcal{A}'$-type of the tree $\nu_{\mathsf{c},\mathsf{a}}(\mathsf{F})$. This shows that the infinite complete A-labeled $(\mathscr{T}_{\mathcal{A}'} \cup \{\perp\})$-colored tree $\vec{\mathsf{T}}'$, defined by:

- $\vec{\mathsf{T}}'(\varepsilon) = \vec{\nu}_{\mathsf{c},\mathsf{a}}(\tau_{\mathcal{A}'}(\emptyset))$, where $\mathsf{c} = \mathsf{T}(\varepsilon)$ and a is an arbitrary label from A;
- for every vertex $\nu \in \mathsf{A}^*$ and every label $\mathsf{a} \in \mathsf{A}$, $\vec{\mathsf{T}}'(\nu\,\mathsf{a}) = \vec{\nu}_{\mathsf{c},\mathsf{a}}(\vec{\mathsf{T}}'(\nu))$, where $\mathsf{c} = \mathsf{T}(\nu\,\mathsf{a})$,

encodes the retraction of T' with respect to $\mathcal{A}'$ and Π'.

It remains to prove that $\vec{\mathsf{T}}'$ has rank n, namely, that $\vec{\mathsf{T}}'$ can be obtained from T via a finite-state recoloring (without lookahead). The Mealy tree automaton $\mathcal{M}$ that produces $\vec{\mathsf{T}}'$ from input T is defined as follows. We let $\mathcal{M} = (\mathsf{A}, \mathsf{C}, \emptyset, \mathscr{T}_{\mathcal{A}'} \cup \{\perp\}, \mathsf{Q}, \delta, \Omega, \mathsf{q}_0)$, where

- $\mathsf{Q} = \mathscr{T}_{\mathcal{A}'} \times \mathsf{A}$;
- for every state $\mathsf{q} = (\sigma, \mathsf{a}) \in \mathsf{Q}$, every input symbol $\mathsf{c} \in \mathsf{C}$, and every label $\mathsf{a}' \in \mathsf{A}$,
$$\delta(\mathsf{q}, \mathsf{c}, \mathsf{a}') = (\vec{\nu}_{\mathsf{c},\mathsf{a}}(\sigma), \mathsf{a}');$$
- for every state $\mathsf{q} = (\sigma, \mathsf{a}) \in \mathsf{Q}$ and every input symbol $\mathsf{c} \in \mathsf{C}$,
$$\Omega(\mathsf{q}, \mathsf{c}) = \vec{\nu}_{\mathsf{c},\mathsf{a}}(\sigma);$$
- $\mathsf{q}_0 = (\sigma_0, \mathsf{a}_0)$, where σ_0 is the minor $\mathcal{A}'$ of the empty tree and a is any arbitrary label chosen from A.

We let the reader check that $\mathcal{M}(\mathsf{T})$ is the encoding $\vec{\mathsf{T}}'$ of the retraction of T' with respect to $\mathcal{A}'$ and Π'. To conclude the proof, we must build the footprint ξ' of T' on the grounds of a given footprint ξ of T. We simply set $\xi' = (\mathsf{A}, \mathsf{f}')$, where f' is the function that maps any A-augmented tree automaton $\mathcal{A}'$ to the footprint $\vec{\xi}'$ of $\vec{\mathsf{T}}'$ ($= \mathcal{M}(\mathsf{T})$), which, by Theorem 9, is computable from the footprint ξ. □

3.5 Effectiveness of the Contraction Method

In this section, we show the effectiveness of the closure properties for reducible trees through a number of meaningful application examples, namely, the identification of suitable upper bounds to the ranks of the deterministic trees in the Caucal hierarchy, a characterization of the languages recognized by the so-called two-way alternating Muller tree automata, and the decidability of the acceptance problem for morphic trees. These results, besides showing the robustness of the notion of rank of a tree, provide a neat framework to reason on retractions of trees and to easily transfer decidability results.

3.5.1 Reducible Trees and the Caucal Hierarchy

In the following, we focus our attention on the deterministic trees of the Caucal hierarchy and we provide an alternative characterization of them in terms of tree transductions with rational lookahead and unfoldings with backward edges and loops. Moreover, by exploiting the closure properties established in Section 3.4.2, we show that every deterministic tree in the level n of the Caucal hierarchy has rank n.

First of all, we recall that, according to Definition 18, every deterministic tree in the Caucal hierarchy can be obtained from some finite graph by repeatedly applying MSO-definable interpretations and unfoldings. Moreover, as shown in the proof of Proposition 32, any inverse *rational* substitution can be viewed as a special form of MSO-definable interpretation. On the contrary, there exist MSO-definable interpretations which are not expressible as inverse rational substitutions. Even though inverse rational substitutions are less expressive than MSO-definable interpretations, one can generate all and only the graphs in the Caucal hierarchy either by means of MSO-definable interpretations and unfoldings [108] or by means of inverse rational substitutions and unfoldings [13], starting from finite graphs. In the following, we shall use this latter characterization of the Caucal hierarchy. Precisely, we assume that, for every $n > 0$, the level n trees of the Caucal hierarchy can be obtained from the level $n-1$ trees by applying an inverse rational substitution followed by an unfolding operation.

From [11], we know that for each level of the Caucal hierarchy, there exists a representative graph, called *graph generator*, from which all other graphs belonging to that level can be obtained via MSO-definable interpretations. For a given $n \in \mathbb{N}$, the generator G_n for the level n graphs of the Caucal hierarchy is defined as the n-fold application of the treegraph operation [112] to the infinite complete binary tree [11]. Graph generators are closely related to some special trees which have been introduced in [7] to simulate games on higher order pushdown systems. These trees, which we call *Cachat tree generators*, are obtained from the infinite complete binary tree via n-fold applications of $\mathcal{F}lip\mathcal{U}nf$. Let us denote by C_n, with $n \in \mathbb{N}$, the tree obtained from the n-fold applications of $\mathcal{F}lip\mathcal{U}nf$ to the infinite complete binary tree. It is easy to see that each graph generator G_n can be obtained from the tree C_n via a suitable MSO-definable interpretation that first restricts the domain to those vertices/words that do not contain any occurrence of substrings of the form $a\,\bar{a}$ or $\bar{a}\,a$, with $a \in A$, and then reverses all $\bar{a}$-labeled edges. It thus follows that Cachat trees are generators of the graphs of the Caucal hierarchy via MSO-definable interpretations.

Here, we define a new class of tree generators, which contains all Cachat tree generators and generates all *deterministic* trees in the Caucal hierarchy via inverse rational forward substitutions, that is, restricted forms of MSO-definable interpretations. On the one hand, such a result is weaker than those by Carayol and Wöhrle and by Cachat, because it allows one to generate only the proper subset of the Caucal hierarchy consisting of all deterministic trees; moreover, to this end, it makes use of a set of tree generators, rather a single graph (resp., tree) generator, for each level. On the other, it is stronger than them, because it exploits a transformation which weaker than MSO interpretation.

Definition 35. *For every* $\mathsf{n} \in \mathbb{N}$*, a* level n tree generator *is any tree of the form* $\mathsf{T_n} = \mathit{FlipUnf}^{\mathsf{n}}(\mathsf{T})$*, where* T *is a deterministic regular tree.*

From Theorem 10, we immediately have that every level n tree generator has rank n. Below, we show that inverse rational forward substitutions applied to level n tree generators yields all level n trees of the Caucal hierarchy (up to bisimulation).

Lemma 14. *Every level* n *tree* T *of Caucal hierarchy is bisimilar to a tree of the form* $\overrightarrow{\mathsf{h}}_{\mathsf{n}}^{-1}(\mathsf{T_n})$*, where* $\mathsf{T_n}$ *is a level* n *tree generator, labeled over a set* $\mathsf{A_n}$*, and* $\overrightarrow{\mathsf{h}}_{\mathsf{n}}^{-1}$ *is an inverse rational* $\mathsf{A_n}$*-forward substitution. Moreover, if* T *is deterministic, then* $\mathsf{T} = \overrightarrow{\mathsf{h}}_{\mathsf{n}}^{-1}(\mathsf{T_n})$*.*

Proof. We prove the claim by using induction on the level n of the tree T and Proposition 31. The case $\mathsf{n} = 0$ is trivial. Let $\mathsf{n} > 0$ and let A be the set of edge labels of T. By construction, there exist a level $\mathsf{n}-1$ tree T' of the Caucal hierarchy and an inverse rational substitution h^{-1} such that $\mathsf{T} = \mathit{Unf}(\mathsf{h}^{-1}(\mathsf{T}'))$. From the inductive hypothesis, there exist a level $\mathsf{n}-1$ tree generator $\mathsf{T_{n-1}}$, labeled over a set $\mathsf{A_{n-1}}$, and an inverse rational $\mathsf{A_{n-1}}$-forward substitution $\overrightarrow{\mathsf{h}}_{\mathsf{n-1}}^{-1}$ such that T' is bisimilar to the tree $\overrightarrow{\mathsf{h}}_{\mathsf{n-1}}^{-1}(\mathsf{T_{n-1}})$. Thus, we have

$$\mathsf{T} = \mathit{Unf}\big(\mathsf{h}^{-1}(\overrightarrow{\mathsf{h}}_{\mathsf{n-1}}^{-1}(\mathsf{T_{n-1}}))\big).$$

Then, by applying Proposition 31 to the inverse rational substitution $\mathsf{h}^{-1} \circ \overrightarrow{\mathsf{h}}_{\mathsf{n-1}}^{-1}$, we have that there exist a level n tree generator $\mathsf{T_n} = \mathit{FlipUnf}(\mathsf{T_{n-1}})$, labeled over a set $\mathsf{A_n}$, and an inverse rational $\mathsf{A_n}$-forward substitution $\overrightarrow{\mathsf{h}}_{\mathsf{n}}^{-1}$ such that T is bisimilar to the tree $\overrightarrow{\mathsf{h}}_{\mathsf{n}}^{-1}(\mathsf{T_n})$. Moreover, if T is deterministic, then the two trees are also isomorphic. □

The following theorem provides an alternative characterization of the class of all deterministic trees of the Caucal hierarchy in terms of unfoldings with backward edges and loops and tree transductions with rational lookahead.

Theorem 11. *For every* $\mathsf{n} \in \mathbb{N}$*, the level* n *deterministic trees of the Caucal hierarchy are all and only the trees of the form*

$$\mathcal{T}\big(\mathit{FlipUnf}^{\mathsf{n}}(\mathsf{T})\big),$$

where T *is a regular tree and* $\mathcal{T}$ *is a tree transduction with rational lookahead.*

Proof. This follows from Lemma 14 and Proposition 32. □

Corollary 2. *For every* $\mathsf{n} \in \mathbb{N}$*, the level* n *deterministic trees of the Caucal hierarchy have rank* n*.*

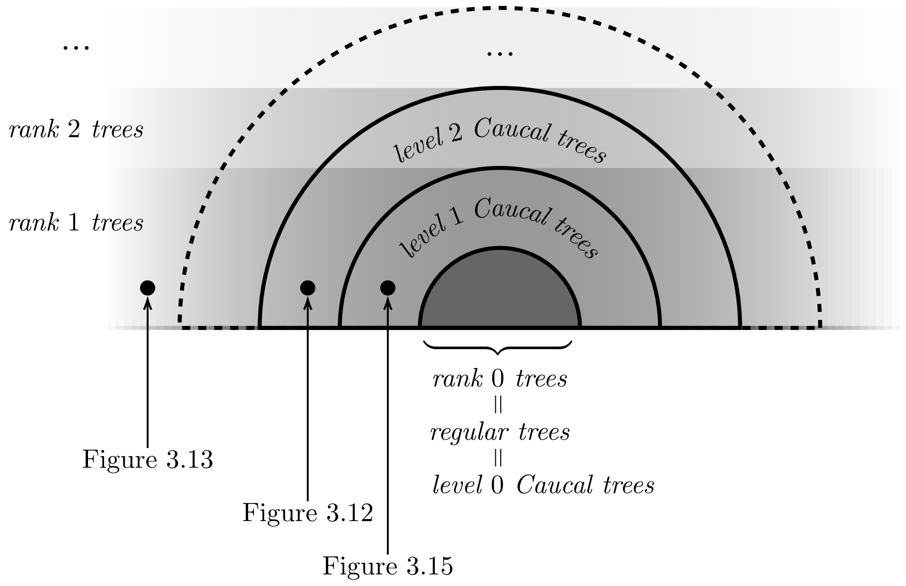


Fig. 3.18. Relationships between reducible trees and Caucal trees

Proof. This follows from Theorem 11 and Theorem 9, since level n tree generators have rank n. □

It is worth remarking that the converse of Corollary 2 does not hold for $n > 0$. As an example, the tree depicted in Figure 3.12 is known to belong to the third level of the Caucal hierarchy; however, by exploiting compositional properties of types, it is easy to prove that it has rank 1. Moreover, we already know that there exist several deterministic trees (see, for instance, Figure 3.13) that do not belong to the Caucal hierarchy but still have rank n, for some $n \in \mathbb{N}$. This basically means that the class of all reducible trees properly include the class of all deterministic trees of the Caucal hierarchy. Figure 3.18 gives and intuitive account of the relationships between the classes of trees of rank n and the classes of the level n deterministic trees of the Caucal hierarchy, for n ranging over $\mathbb{N}$.

3.5.2 Two-Way Alternating Tree Automata

Two-way alternating parity tree automata have been introduced by Vardi in [109] in order to solve the satisfiability problem for the so-called two-way modal μ-calculus, i.e., the extension of propositional modal μ-calculus [57] with backward modalities $\Diamond^-$ and $\Box^-$. In that paper, two-way alternating parity tree automata are also shown to be expressively equivalent to nondeterministic parity tree automata. Here, we consider two-way alternating tree automata equipped with Muller acceptance condition (hereafter called simply two-way tree automata) and we exploit Theorem 5 and Theorem 10 to translate any given two-way alternating (Muller) tree automaton into an equivalent (one-way) alternating (Muller) tree automaton.

Intuitively, while an alternating tree automaton can only spread its states along the successors of the current vertex, a two-way alternating tree automaton is allowed to move along the successors and the predecessor of the current vertex, as well as to stay at the current vertex (and possibly change the state). As in the definition of inverse substitutions (see Section 3.3.4), we distinguish between edges traversed in forward direction and edges traversed in backward direction. Let A be a set of edge labels and C be a set of vertex colors. Moreover, we denote by $\overleftrightarrow{A}$ the set $A \cup \bar{A} \cup \{\#\}$, where $\bar{A}$ is a disjoint copy of A and $\#$ is a fresh symbol not belonging to $A \cup \bar{A}$. For the sake of simplicity, hereafter we restrict our attention to infinite complete trees only.

Definition 36. *A* two-way alternating (Muller) tree automaton *is a tuple* $\mathcal{A} = (A, C, S, \Delta, \mathcal{I}, \mathcal{F})$*, where*

- S *is a finite set of states;*
- $\Delta : S \times C \rightarrow \mathcal{B}^+(\overleftrightarrow{A} \times S)$ *is a transition function;*
- $\mathcal{I} \subseteq S$ *is a set of initial states;*
- $\mathcal{F} \subseteq \mathscr{P}(S)$ *is a family of accepting sets.*

Given an infinite complete (A-labeled C-colored) tree T, a *run* of $\mathcal{A}$ on T is an unlabeled ($\mathcal{D}om(T) \times S$)-colored tree R such that

- the root of R is colored with a pair (ε, s), where ε is the root of T and $s \in S$;
- for every vertex u of R colored with (v, s), if $\Delta(s, T(v)) = \varphi$, then (i) there is a minimal set $M = \{(a_1, s_1), ..., (a_k, s_k)\}$, with $a_1, ..., a_k \in \overleftrightarrow{A}$ and $s_1, ..., s_k \in S$, that satisfies φ, (ii) R contains exactly k successors $u_1, ..., u_k$ of u, and (iii) for all $1 \leqslant i \leqslant k$,

$$R(u_i) = \begin{cases} (v\, a_i,\ s_i) & \text{if } a_i \in A, \\ (v',\ s_i) & \text{if } a_i = \bar{a} \in A \text{ and } v\ =\ v'\, a, \text{ for some } a \in A, \\ (v,\ s_i) & \text{if } a_i = \#. \end{cases}$$

The run R is said to be successful, and hence $\mathcal{A}$ accepts T, if it further satisfies the following two conditions:

i) the state associated with the root of R is an initial state from $\mathcal{I}$;
ii) for every infinite path π in R, the set of states that occur infinitely often along π is an accepting set from $\mathcal{F}$.

The following theorem shows that any two-way alternating tree automaton $\mathcal{A}$ is equivalent to a suitable (one-way) alternating tree automaton $\widetilde{\mathcal{A}}$.

Theorem 12. *For any given two-way alternating tree automaton $\mathcal{A}$, one can build an alternating tree automaton $\widetilde{\mathcal{A}}$ such that, for every infinite complete tree* T*,*

$$\mathsf{T} \in \mathscr{L}(\mathcal{A}) \qquad \textit{iff} \qquad \mathsf{T} \in \mathscr{L}(\widetilde{\mathcal{A}}).$$

Moreover, the size of $\widetilde{\mathcal{A}}$ is doubly exponential in the size of $\mathcal{A}$.

Proof. Let $\mathcal{A} = (\mathsf{A}, \mathsf{C}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F})$ be a two-way alternating tree automaton. By definition of run of $\mathcal{A}$, we have that

$$\mathsf{T} \in \mathscr{L}(\mathcal{A}) \qquad \text{iff} \qquad \mathsf{T}' \in \mathscr{L}(\mathcal{A}')$$

where $\mathsf{T}' = \mathcal{F}lip\mathcal{U}nf(\mathsf{T})$, namely, T' is the two-way unfolding of T, and $\mathcal{A}'$ is the A-augmented alternating tree automaton $(\mathsf{A}', \mathsf{C}, \mathsf{A}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$, with $\mathsf{A}' = \overleftrightarrow{\mathsf{A}}$ and $\mathcal{G} = \emptyset$. Moreover, we can define the A-labeled factorization Π' of T' as in the proof of Theorem 10 and then denote by $\overrightarrow{\mathsf{T}}'$ the encoding of the retraction of T' with respect to $\mathcal{A}'$ and Π' and by $\overrightarrow{\mathcal{A}}' = (\mathsf{A}, \mathscr{T}_{\mathcal{A}'} \cup \{\bot\}, \overrightarrow{\mathsf{S}}, \overrightarrow{\Delta}, \overrightarrow{\mathcal{I}}, \overrightarrow{\mathcal{F}})$ the retraction automaton of $\mathcal{A}'$. Hence, by Theorem 5, we have

$$\mathsf{T}' \in \mathscr{L}(\mathcal{A}') \qquad \text{iff} \qquad \overrightarrow{\mathsf{T}}' \in \mathscr{L}(\overrightarrow{\mathcal{A}}').$$

Again, according to the argument of the proof of Theorem 10, we know that the infinite complete A-labeled $(\mathscr{T}_{\mathcal{A}'} \cup \{\bot\})$-colored tree $\overrightarrow{\mathsf{T}}'$ can be obtained from T via a suitable finite-state recoloring (without lookahead). We can then 'synchronize' the retraction automaton $\overrightarrow{\mathcal{A}}'$ with such a finite-state recoloring, thus obtaining an alternating tree automaton $\widetilde{\mathcal{A}}$ equivalent to $\mathcal{A}$. More precisely, we denote by $\mathcal{M} = (\mathsf{A}, \mathsf{C}, \emptyset, \mathscr{T}_{\mathcal{A}'} \cup \{\bot\}, \mathsf{Q}, \delta, \Omega, \mathsf{q}_0)$ the Mealy tree automaton that produces $\overrightarrow{\mathsf{T}}'$ on input T and we define the alternating tree automaton $\widetilde{\mathcal{A}} = (\mathsf{A}, \mathsf{C}, \widetilde{\mathsf{S}}, \widetilde{\Delta}, \widetilde{\mathcal{I}}, \widetilde{\mathcal{F}})$ as follows:

- $\widetilde{\mathsf{S}} = \mathsf{S} \times \mathsf{Q}$;
- for every state $\widetilde{\mathsf{s}} = (\mathsf{s}, \mathsf{q}) \in \widetilde{\mathsf{S}}$ and every color $\mathsf{c} \in \mathsf{C}$, $\widetilde{\Delta}(\widetilde{\mathsf{s}}) = \Phi_{\mathsf{s},\mathsf{q},\mathsf{c}}$, where $\Phi_{\mathsf{s},\mathsf{q},\mathsf{c}}$ is the formula obtained from $\overrightarrow{\Delta}(\mathsf{s}, \Omega(\mathsf{q}, \mathsf{c}))$ by replacing every atom of the form $\langle \mathsf{a}, \mathsf{s}' \rangle$ with $\langle \mathsf{a}, (\mathsf{s}', \delta(\mathsf{q}, \mathsf{c}, \mathsf{a})) \rangle$;
- $\widetilde{\mathcal{I}}$ consists of all pairs of the form $(\mathsf{s}, \mathsf{q}_0)$, with $\mathsf{s} \in \mathcal{I}$;
- $\widetilde{\mathcal{F}}$ consists of all subsets F of $\widetilde{\mathsf{S}}$ such that $\downarrow_1 \mathsf{F} \in \mathcal{F}$.

It is immediate to prove that

$$\overrightarrow{\mathsf{T}}' \in \mathscr{L}(\overrightarrow{\mathcal{A}}') \qquad \text{iff} \qquad \mathsf{T} \in \mathscr{L}(\widetilde{\mathcal{A}}).$$

Finally, from the definitions of the retraction automaton $\overrightarrow{\mathcal{A}}$ and the Mealy tree automaton $\mathcal{M}$, it easily follows that the size of $\widetilde{\mathcal{A}}$ is doubly exponential in the size of $\mathcal{A}$. □

3.5.3 Morphic Trees

We conclude the section by proving that any morphic tree, namely, any tree obtained as the limit of n-fold applications of a regular tree morphism (see Section 3.3.2), has rank 1. Below, we recall the definition of morphic tree.

Definition 37. *A* morphic tree *is any tree of the form*

$$\mu^{\omega}(\widetilde{c}) = \sup\big\{\mu^0(\widetilde{c}) \sqsubseteq \mu^1(\widetilde{c}) \sqsubseteq \dots\big\},$$

where μ is a regular tree morphism, $\widetilde{c}$ is a seed, namely, a color such that $\mu(\widetilde{c})(\varepsilon) = \widetilde{c}$, and $\sqsubseteq$ is the usual ω-complete partial order on trees.

Roughly speaking, the proof that all morphic trees have rank 1 is based on the fact that, given any finite (multiplicative) semigroup $(X, \cdot)$, there only exist finitely many different functions that map trees to elements of X and are compatible with first-order tree substitutions, that is, 'concatenations' at the leaves. Such a property can be easily explained by referring to the simpler case of words. Recall that a word homomorphism is a function f from the set of all finite words over an alphabet C to a (multiplicative) semigroup $(X, \cdot)$ such that, for every pair of words $w_1, w_2 \in C^*$, $f(w_1\,w_2) = f(w_1)\,f(w_2)$ holds. Such a definition immediately implies that any word homomorphism f is uniquely determined by the images $f(c)$ of the symbols $c \in C$ and hence there only exist finitely many different word homomorphisms to a given finite semigroup. These notions and results can be immediately lifted to trees. For the sake of simplicity, we restrict our attention to 'tree homomorphisms' of the form $\tau_{\mathcal{A}'} \circ \mu$, where $\mathcal{A}'$ is a $(C' \cup D)$-augmented tree automaton and μ is a regular tree morphism mapping A-labeled C-colored $(C \cup D)$-augmented trees to A'-labeled C'-colored $(C' \cup D)$-augmented trees. In the sequel, we assume that $A = \{a_1, \dots, a_k\}$.

Proposition 35. *Let $\mathcal{A}'$ be a $(C' \cup D)$-augmented tree automaton and let μ_1 and μ_2 be two regular tree morphisms. We have that $\tau_{\mathcal{A}'} \circ \mu_1 = \tau_{\mathcal{A}'} \circ \mu_2$ iff, for every color $c \in C$, $\tau_{\mathcal{A}'}\big(\mu_1(c\langle a_1, \dots, a_k\rangle)\big) = \tau_{\mathcal{A}'}\big(\mu_2(c\langle a_1, \dots, a_k\rangle)\big)$.*

Proof. The left to right implication is trivial. We prove the converse implication by exploiting Theorem 8. Let $\bar{F}_1$ (resp., $\bar{F}_2$) be the tuple of regular replacing trees that specifies the tree morphism μ_1 (resp., μ_2) and let $\bar{\sigma}_1$ (resp., $\bar{\sigma}_2$) be the minor $\mathcal{A}'$-type of $\bar{F}_1$ (resp., $\bar{F}_2$). It suffices to prove that, whenever $\bar{\sigma}_1 = \bar{\sigma}_2$ holds, $\tau_{\mathcal{A}'}(\mu_1(T)) = \tau_{\mathcal{A}'}(\mu_2(T))$ holds for every tree T. We denote by $\mathcal{A}$ the $(C' \cup D)$-augmented tree automaton obtained from Theorem 8 by letting $\bar{\sigma} = \bar{\sigma}_1$ $(= \bar{\sigma}_2)$. Furthermore, let T be an A-labeled C-colored $(C \cup D)$-augmented tree and σ its minor $\mathcal{A}$-type. Theorem 8 states that the minor $\mathcal{A}'$-type of the tree $\mu_1(T)$ (resp., $\mu_2(T)$) is uniquely determined by the minor $\mathcal{A}$-type of T and the minor $\mathcal{A}'$-type $\bar{\sigma}_1$ $(= \bar{\sigma}_2)$. This shows that $\tau_{\mathcal{A}'}(\mu_1(T)) = \tau_{\mathcal{A}'}(\mu_2(T))$. □

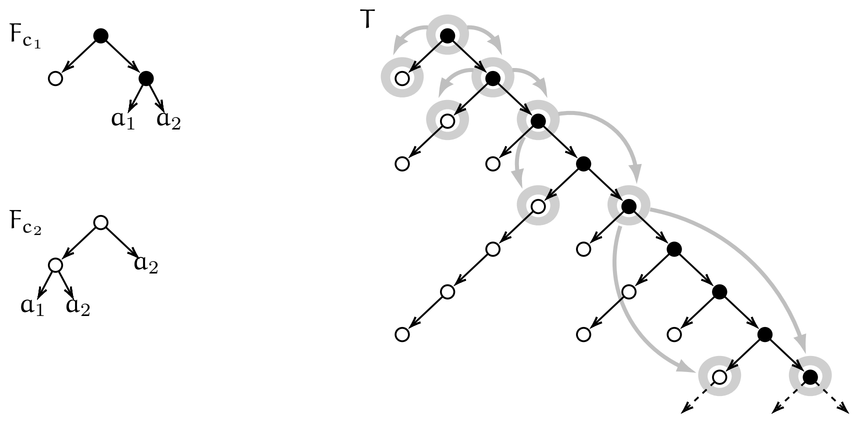


Fig. 3.19. The factorization of a morphic tree

As a consequence of Proposition 35, we have that any function of the form $\tau_{\mathcal{A}'} \circ \mu$, where μ is a regular tree morphism, is uniquely determined by the images of the basic trees $c\langle a_1, ..., a_k\rangle$, for c ranging over C. Since these images range over the finite set $\mathscr{T}_{\mathcal{A}'}$, there only exist finitely many different functions of the form $\tau_{\mathcal{A}'} \circ \mu$, for any choice of the $(C' \cup D)$-augmented tree automaton $\mathcal{A}'$. In particular, by applying the Pigeonhole Principle, we have that, for every regular tree morphism μ, the sequence $\tau_{\mathcal{A}'} \circ \mu^n$, for $n = 0, 1, 2, ...$, is ultimately periodic.

In the following example, we take advantage of Proposition 35 to provide a regular retraction of a given morphic tree.

Example 10. Let $A = \{a_1, a_2\}$, $C = \{c_1, c_2\}$ (c_1-colored vertices and c_2-colored vertices will be graphically represented by black-colored circles and white-colored circles, respectively), and $D = A$. Furthermore, let μ be the regular tree morphism of Example 6 specified by the two replacing trees $F_{c_1} = c_1\langle c_2, c_1\langle a_1, a_2\rangle\rangle$ and $F_{c_2} = c_2\langle c_2\langle a_1, a_2\rangle, a_2\rangle$ (left part of Figure 3.19). We consider the morphic tree $T = \mu^\omega(\widetilde{c})$, which is obtained as the limit of n-fold applications of μ to the seed $\widetilde{c} = c_1$ (right part of Figure 3.19). We can define an A-labeled factorization Π of T as follows. First, for every $n \in \mathbb{N}$, the tree $\mu^n(\widetilde{c}\langle a_1, a_2\rangle)$ is an expansion of $\mu^n(\widetilde{c})$ with A-colored vertices. We thus say that a vertex v of T is an a-extension of T, with $a \in A$, if there is an index $n \in \mathbb{N}$ such that v is an a-colored leaf of $\mu^n(\widetilde{c}\langle a_1, a_2\rangle)$. We define the domain of the factorization Π of T as the set of all a-extensions of T, for $a \in A$ (plus the root of T). Moreover, we label each edge (u, v) of Π with $a \in A$ if v is an a-extension of T. Figure 3.19 (right part) depicts the factorization Π (which is only accidentally a deterministic tree).

Now, let G_{a_1} and G_{a_2} be, respectively, the left subtree and the right subtree of $F_{\widetilde{c}}$. It is easy to see that, for every vertex v of Π at distance $n > 0$ from the root, if v is an a-extension of T, then the marked factor of T rooted at

ν is isomorphic to the tree $\mu^{n-1}(G_a)$. Thus, for any given A-augmented tree automaton $\mathcal{A}$, the infinite complete A-labeled $(\mathscr{T}_{\mathcal{A}} \cup \{\perp\})$-colored tree $\vec{T}$ such that

$$\vec{T}(u) = \begin{cases} \tau_{\mathcal{A}}(F_{\tilde{c}}) & \text{if } u = \varepsilon, \\ \tau_{\mathcal{A}}(\mu^n(G_{a_1})) & \text{if } u = a_2^n\, a_1, \text{ for some } n \in \mathbb{N}, \\ \tau_{\mathcal{A}}(\mu^n(G_{a_2})) & \text{if } u = a_2^n\, a_2, \text{ for some } n \in \mathbb{N}, \\ \perp & \text{otherwise} \end{cases}$$

encodes the retraction of T with respect to $\mathcal{A}$ and Π. Moreover, from Proposition 35, we know that $\vec{T}$ is a regular tree and hence T has rank 1, which proves that T enjoys a decidable acceptance problem.

The following theorem is a straightforward generalization of Example 10.

Theorem 13. *All morphic trees have rank* 1.

Proof. Let $\bar{F} = (F_c)_{c \in C}$ be a tuple of A'-labeled C'-colored $(C' \cup D)$-augmented regular replacing trees, with $D \supseteq A$ and let μ be the corresponding regular tree morphism, which maps A-labeled C-colored $(C \cup D)$-augmented trees to A'-labeled C'-colored $(C' \cup D)$-augmented trees. Furthermore, let $\tilde{c}$ be a seed of μ, namely, a color $\tilde{c} \in C$ such that $F_{\tilde{c}}(\varepsilon) = \tilde{c}$. Finally, let $A = \{a_1, ..., a_k\}$. For every $n \in \mathbb{N}$, we denote by T_n (resp., T_n^+) the A'-labeled C'-colored $(C' \cup D)$-augmented tree $\mu^n(\tilde{c})$ (resp., $\mu^n(\tilde{c}\langle a_1, ..., a_k\rangle)$). For every $n \in \mathbb{N}$, both $T_n \sqsubseteq T_{n+1}$ and $T_n \sqsubseteq T_n^+$ hold. We have to show that the morphic tree $T = \sup\{T_0 \sqsubseteq T_1 \sqsubseteq ...\}$ has rank 1. We first define an A-labeled factorization Π of T (following the construction we outlined in Example 10). We say that a vertex ν of T is an a-extension of T, with $a \in A$, if there is an index $n \in \mathbb{N}$ such that ν is an a-colored leaf of T_n^+. We define the domain of the factorization Π as the set of all a-extensions of T for $a \in A$, plus the root of T. Moreover, we label any edge (u, ν) of Π by a iff ν is an a-extension of T. Notice that, by construction, ν is a vertex of Π at distance $n > 0$ from the root iff ν is an a-colored leaf of T_{n-1}^+.

We now focus our attention on the marked factors of T with respect to Π. Let ν be a vertex of Π at distance $n > 0$ from the root and let a be the label of the edge of Π that has ν as its target vertex. If we denote by G_a the subtree of $F_{\tilde{c}}$ rooted at the a-successor of $F_{\tilde{c}}$, for every $a \in A$, we have

$$\begin{aligned} T_n^+ &= \mu^n\big(\tilde{c}\langle a_1, ..., a_k\rangle\big) \\ &= \mu^{n-1}\big(\mu(\tilde{c}\langle a_1, ..., a_k\rangle)\big) \\ &= \mu^{n-1}\big(F_{\tilde{c}}\big) \\ &= \mu^{n-1}\big(\tilde{c}\langle a_1, ..., a_k\rangle[G_{a_i}/a_i]_{a_i \in A}\big) \\ &= \mu^{n-1}\big(\tilde{c}\langle a_1, ..., a_k\rangle\big)\big[\mu^{n-1}(G_{a_i})/a_i\big]_{a_i \in A} \\ &= T_{n-1}^+[\mu^{n-1}(G_{a_i})/a_i]_{a_i \in A} \end{aligned}$$

Since by construction every successor of v in Π is a leaf of T_n^+, we have that the marked factor of T rooted at v is isomorphic to the A'-labeled C'-colored $(C' \cup D)$-augmented tree $\mu^{n-1}(G_a)$ and hence $\tau_{\mathcal{A}} \circ \mu^{n-1}(G_a)$ is its minor $\mathcal{A}$-type, for any given $(C' \cup D)$-augmented tree automaton $\mathcal{A}$. From Proposition 35, we have that there exist only finitely many different functions of the form $\tau_{\mathcal{A}} \circ \mu^n$, for n ranging over $\mathbb{N}$. It immediately follows that the retraction of T with respect to $\mathcal{A}$ and Π is encoded by a suitable regular tree $\overrightarrow{T}$. We now show how such a tree can be defined as the result of the application of a finite-state recoloring (without lookahead) to the infinite complete A-labeled $\perp$-colored tree. More precisely, let us denote by Γ the *finite* set of all functions of the form $\tau_{\mathcal{A}} \circ \mu^n$, for n ranging over $\mathbb{N}$. We define a Mealy tree automaton $\mathcal{M} = (A, \{\perp\}, \emptyset, \mathscr{T}_{\mathcal{A}} \cup \{\perp\}, Q, \delta, \Omega, q_0)$, which only depends on $\mathcal{A}$, such that

- Q consists of two distinguished states q_0 and $q_\perp$, plus all pairs of the form (γ, a), with $\gamma \in \Gamma$ and $a \in A$;
- for every state $q \in Q$ and every label $a \in A$,

$$\delta(q, \perp, a) = \begin{cases} (\tau_{\mathcal{A}}, a) & \text{if } q = q_0 \text{ and } F_{\widetilde{c}} \text{ has some } a\text{-colored leaf,} \\ (\gamma \circ \mu, a) & \text{if } q = (\gamma, a') \text{ and } G_{a'} \text{ has some } a\text{-colored leaf,} \\ q_\perp & \text{otherwise;} \end{cases}$$

- for every state $q \in Q$,

$$\Omega(q, \perp) = \begin{cases} \tau_{\mathcal{A}}(\widetilde{c}\langle a_1, ..., a_k\rangle) & \text{if } q = q_0, \\ \gamma(G_a) & \text{if } q = (\gamma, a), \\ \perp & \text{otherwise.} \end{cases}$$

It is clear that the (unique) output of the Mealy tree automaton $\mathcal{M}$ is the A-labeled $(\mathscr{T}_{\mathcal{A}} \cup \{\perp\})$-colored *regular* tree $\overrightarrow{T}$ that encodes the retraction of T with respect to $\mathcal{A}$ and Π. This shows that T has rank 1. Moreover, one can effectively build a finite rooted graph $\overrightarrow{G}$ representing the tree $\overrightarrow{T}$ on the grounds of the given tree morphism μ and the given $(C' \cup D)$-augmented tree automaton $\mathcal{A}$. Thus, the footprint of T can be defined as the pair (B, f), where $B = C' \cup D$ and f is the function that maps any B-augmented tree automaton $\mathcal{A}$ to the corresponding graph $\overrightarrow{G}$. □

It is worth pointing out the existence of a natural connection between regular tree morphisms and (higher-order) recursive program schemes. Higher-order recursive program schemes were first introduced by Damm in [29, 31] and are a hot topic of current research in theoretical computer science (see, for instance, [54, 1, 55, 84, 48, 47, 10]). At the bottom (first-order) level, these schemes consist of simple term rewriting rules, whose semantics can be thought of as a substitution of non-terminal symbols in a tree, that is, as a second-order tree substitution. Therefore, the class of morphic trees seems to be closely related to that of recursive program schemes of level 1, even though, at the moment, we are not able to provide translations from one class to the other. More generally, it would be interesting to study suitable classes of 'higher-order' morphic trees by possibly comparing them with corresponding levels in the hierarchy of recursive program schemes.

3.5.4 Layered Temporal Structures

In this section, we consider layered temporal structures, which have been originally introduced by Montanari et al. in [66, 67, 71] to model finite and infinite hierarchies of time granularities. The focus is on three kinds of layered temporal structures: the k-refinable n-layered structure (n-LS for short), which consists of a fixed finite number n of temporal layers such that each time point can be refined into k time points of the immediately finer layer, if any, the k-refinable downward-unbounded layered temporal structure (DULS for short), which consists of an infinite number of arbitrarily fine layers, and the k-refinable upward-unbounded layered temporal structure (UULS for short), which consists of an infinite number of arbitrarily coarse layers.

In their original formulation, layered temporal structures were equipped with suitable ordering relations and viewed as tree-shaped structures. As an example, the k-refinable DULS can be viewed as an infinite ordered sequence of infinite k-ary trees, while the k-refinable UULS can be viewed as a complete k-ary infinite tree generated from the leaves or, equivalently, as an infinite ordered sequence of finite increasing k-ary trees.

The MSO theories of layered temporal structures have been shown to be expressive enough to capture meaningful temporal properties of reactive systems (such as *'the event p occurs at all time points k^i, with $i \in \mathbb{N}$'* or *'the event p occurs densely over a given time interval'*) and moreover decidable. Originally, the decidability of the model checking problems for the k-refinable n-LS, DULS, and UULS has been proved by reducing each of these problems to the decidability of the MSO theory of a suitable 'collapsed' structure. In particular, the MSO theory of the k-refinable n-LS is reduced to the MSO theory of the semi-infinite line, the MSO theory of the k-refinable DULS is translated into the MSO theory of the infinite complete k-ary tree, and that of the k-refinable UULS into the MSO theory of the k-ary systolic tree [69, 73, 68, 71].

Let us fix a natural number $k \geqslant 2$ and let us define the set $A = \{a_1, ..., a_k\}$ of edge labels of k-refinable layered temporal structures. Below, we give a generic definition of k-refinable layered temporal structure. Subsequently, we instantiate such a definition with specific forms of layered temporal structure, which have been considered in the literature.

Definition 38. *A* k-refinable layered temporal structure *is a graph* $\mathsf{G} = \big(\mathsf{V}, <, (\mathsf{E}_a)_{a\in\mathsf{A}}\big)$*, where*

- *the set* V *is a union of the form* $\bigcup_{i\in I} \mathsf{L}_i$*, where* I *is a subset of* $\mathbb{Z}$ *and, for every* $i \in I$*,* $\mathsf{L}_i = \big\{(i, n) : n \in \mathbb{N}\big\}$ *(hereafter, each set* L_i *is called a* layer*);*
- $<$ *is a total order on* V*;*
- *for each label* $a_j \in \mathsf{A}$*,* E_{a_j} *is a partial function, called* j-th projection relation*, that maps a vertex* $(i, n) \in \mathsf{L}_i$ *to the vertex* $(i+1, kn+j-1) \in \mathsf{L}_{i+1}$*, provided that* $i+1 \in I$*.*

For the sake of simplicity, for now on, we assume $k = 2$, namely, we restrict our attention to 2-refinable layered temporal structures only (the general case of k-refinable layered temporal structures can be dealt with by a similar approach). We denote by $\prec_0$, $\prec_1$, and $\prec_2$ the ordering relations defined, respectively, by the pre-order, in-order, and post-order visits of the vertices in a binary tree (for arbitrary k-ary trees, it is straightforward to generalize these definitions by using $k+1$ distinct ordering relations $\prec_0, ..., \prec_k$). Given a layered temporal structure $\mathsf{G} = \big(\mathsf{V}, <, (\mathsf{E}_a)_{a\in\Lambda_k}\big)$, we call *subtree* of G any subgraph of G which is induced by the set of vertices reachable from a designated source vertex $(i, n) \in \mathsf{V}$ through the projection relations $\mathsf{E}_{a_1}, ..., \mathsf{E}_{a_k}$.

We now describe the set I and the total order $<$ of Definition 38 for the various specific forms of layered temporal structure, precisely, for to the upward-unbounded and the downward-unbounded layered temporal structures. We also introduce a third form of layered temporal structure, called totally-unbounded layered temporal structure (TULS for short), which can be viewed as the composition of the DULS and the UULS.

- The (2-refinable) DULS is obtained by defining $I = \mathbb{N}$ and $<$ as the total order which is induced by $\prec_0$, for the elements that belong to a common subtree, and by the linear order $<_0$ of the top layer L_0 (i.e., $(0,0) <_0 (0,1) <_0 (0,2) <_0 ...$), for the elements that belong to distinct subtrees.
- The (2-refinable) UULS is obtained by defining $I = -\mathbb{N}$ (i.e., the set of all non-positive integers) and $<$ as the total order induced by $\prec_1$.
- The (2-refinable) TULS is obtained by defining $I = \mathbb{Z}$ and $<$ as the total order induced by $\prec_1$.

Figures 3.20, 3.21, and 3.22 depict (parts of) the 2-refinable DULS, UULS, and TULS (the projection relations are represented by $\{a_1, a_2\}$-labeled vertical edges, while the transitive reduction of the ordering relation $<_0$ is represented by $<_0$-labeled horizontal edges).

We denote by DULS$^{\not<}$ (resp., UULS$^{\not<}$, TULS$^{\not<}$) the downward-unbounded (resp., upward-unbounded, totally-unbounded) layered temporal structure devoid of the ordering relation $<$. On the one hand, it turns out that the order $<$ is MSO-definable in UULS$^{\not<}$ and in TULS$^{\not<}$ (this follows from the fact that the relation $\prec_1$ induced by the in-order visit of a tree structure is MSO-definable in the structure itself). On the other hand, the order $<$ is not MSO-definable

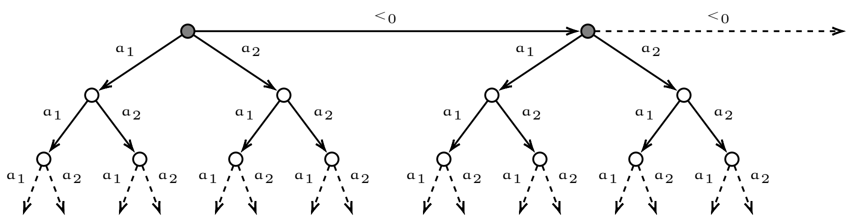


Fig. 3.20. The 2-refinable DULS

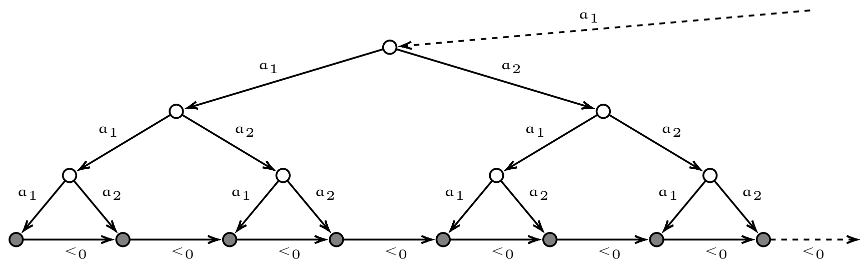


Fig. 3.21. The 2-refinable UULS

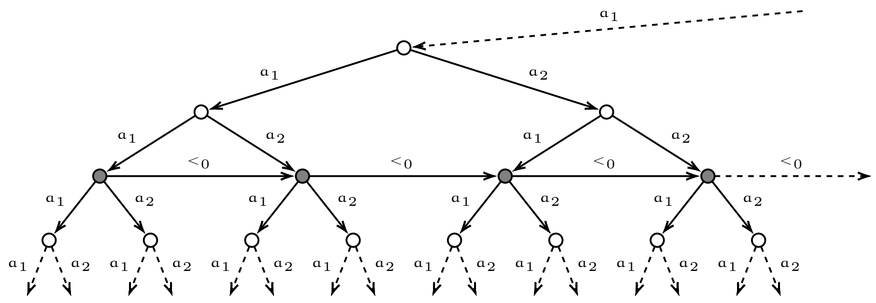


Fig. 3.22. The 2-refinable TULS

in DULS$^{\not<}$ (this follows from the fact that the partial order $<_0$ allows one to distinguish between infinitely many isomorphic subtrees of DULS$^{\not<}$). However, the order $<$ turns out to be MSO-definable in DULS$^{\not<}$, provided that we augment the structure with the partial order $<_0$ on the top layer.

Moreover, both downward-unbounded and upward-unbounded layered temporal structures, endowed with the ordering relation $<$, are MSO-definable in the structure TULS$^{\not<,\mathrm{L}_0}$, which denotes the totally-unbounded layered temporal structure devoid of the ordering relation $<$ and expanded with the unary predicate L_0 (such an extension is needed in order to make it possible to identify the top and the bottom layers of the DULS and the UULS, respectively).

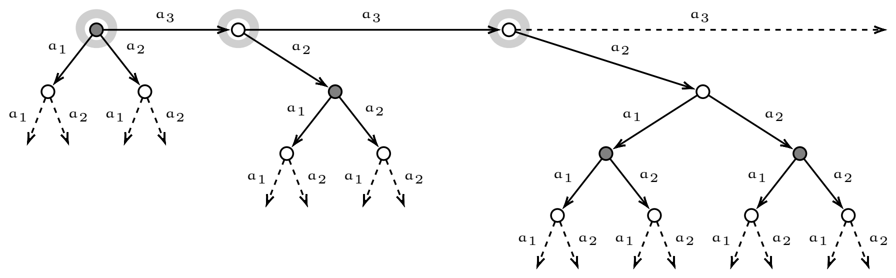


Fig. 3.23. The ternary colored tree embedding $\mathrm{TULS}^{\not<,\mathsf{L}_0}$

On the grounds of these results, we can restrict our attention to the structure $\mathrm{TULS}^{\not<,\mathsf{L}_0}$ only.

In the sequel, we use the contraction method for tree automata to prove that the MSO theory of $\mathrm{TULS}^{\not<,\mathsf{L}_0}$ is decidable. Note that, by exploiting the MSO-compatibility of MSO-definable interpretations, one can transfer such a decidability result to the other layered temporal structures, thus generalizing previous results in the literature. Later on, we proof original decidability results for the chain fragment of MSO logic interpreted over the TULS expanded with either an equi-level predicate or an equi-column predicate.

As a preliminary step, we show that the structure $\mathrm{TULS}^{\not<,\mathsf{L}_0}$ can be obtained via an MSO-definable interpretation (indeed an inverse rational substitution) from a suitable ternary colored tree. Such an MSO-definable interpretation allows us to move from the setting of layered temporal structures to the more standard framework of deterministic colored trees and, therefore, exploit the results presented in Section 3.2.

Let us denote by T the $\{\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3\}$-labeled $\{0, 1\}$-colored tree defined by

- $\mathcal{D}om(\mathsf{T}) = \mathsf{A}^* \cup \big(\{\mathsf{a}_3\}^+ \{\mathsf{a}_2\} \mathsf{A}^*\big)$,
- for every vertex v of T, if v is the empty word ε or v is a word in $\{\mathsf{a}_3\}^n \{\mathsf{a}_2\} \mathsf{A}^n$, then $\mathsf{T}(\mathsf{v}) = 1$, otherwise $\mathsf{T}(\mathsf{v}) = 0$.

Figure 3.23 depicts the tree T. For the sake of brevity, hereafter, we denote by $\mathsf{E}'_{\mathsf{a}_1}$, $\mathsf{E}'_{\mathsf{a}_2}$, and $\mathsf{E}'_{\mathsf{a}_3}$ the successor relations of T and by P the set of all 1-colored vertices of T.

Below, we prove that T has rank 1 and hence, by Theorem 7, it has a decidable MSO theory.

Proposition 36. *The tree* T *that embeds* $\mathit{TULS}^{\not<,\mathsf{L}_0}$ *has rank* 1.

Proof. We prove that T has rank 1 by providing a suitable factorization of it and by showing that, for any given tree automaton $\mathcal{A}$, the corresponding retraction is regular. The factorization Π of T is defined by letting $\mathcal{D}om(\Pi) = \{\mathsf{a}_3\}^*$ and by labeling the induced edges with a single symbol b (see the circled vertices in Figure 3.23). We denote by F_0 the infinite complete A-labeled $\{0, 1\}$-colored tree such that $\mathsf{F}_0(\varepsilon) = 1$ and $\mathsf{F}_0(\mathsf{v}) = 0$ for all $\mathsf{v} \in \mathsf{A}^+$. Then, for every

$n > 0$, we recursively define the tree F_n as $F_n = 0\langle F_{n-1}, F_{n-1}\rangle$. Since for all $n > 0$, F_n is obtained from F_{n-1} via a suitable regular tree insertion, from compositional properties of types (see Corollary 1), we know that, for any given $\{b\}$-augmented tree automaton $\mathcal{A}$, there is a computable function $\overrightarrow{\gamma}$ that maps the minor $\mathcal{A}$-type of F_{n-1} to the minor $\mathcal{A}$-type of F_n. Moreover, it is easy to see that, for every vertex v of Π of the form a_3^{n+1}, the marked factor $T_{\Pi}^{+}[v]$ is isomorphic to the tree $0\langle\emptyset, F_n, b\rangle$. Therefore, one can compute an ultimately periodic sequence $\sigma_0, \sigma_1, \sigma_2, \ldots$ such that, for every $n \geqslant 0$, σ_n is the minor $\mathcal{A}$-type of $T_{\Pi}^{+}[a_3^n]$. This implies that the retraction of T with respect to $\mathcal{A}$ is encoded by a regular $\{b\}$-labeled tree $\overrightarrow{T}$, whose footprint can be effectively build on the grounds of the given automaton $\mathcal{A}$. Therefore, T has rank 1.

Now, let $G = \big(V, (E_a)_{a\in A}, L_0\big)$ be the structure TULS$^{\not<, L_0}$. It is easy to see that G is the result of an MSO-definable interpretation of T that maps vertices in P to vertices in L_0, reverses all a_3-labeled edges, and renames them by a_1. It thus follows that G has a decidable MSO theory. As a matter of fact, an alternative proof of such a decidability result stems from the fact that both T and G belong to the second level of the Caucal hierarchy.

Corollary 3. *The structure TULS$^{\not<, L_0}$ enjoys a decidable MSO theory.*

Proof. The result follows immediately from Proposition 36, Theorem 7, and MSO-compatibility of MSO-definable interpretations.

As previously mentioned, Corollary 3 subsumes previous results about the decidability of MSO theories of DULS and UULS. In [43], model checking problems for (fragments of) MSO logic over n-LS, DULS, and UULS expanded with the binary *equi-level* and *equi-column* predicates L and C have been also considered. The equi-level predicate L allows one to check whether two given elements belong to the same layer of the structure, while the equi-column predicate C allows one to check whether two given elements are at the same distance from the origin of the layer they belong to. Formally, if $V = \big\{(i, n) : i \in I, n \in \mathbb{N}\big\}$ is the domain of the layered temporal structure, with I being a subset of $\mathbb{Z}$, then

$$L = \big\{\big((i, n), (i, n')\big) : i \in I, n, n' \in \mathbb{N}\big\}$$
$$C = \big\{\big((i, n), (i', n)\big) : i, i' \in I, n \in \mathbb{N}\big\}.$$

In [43], it has been shown that the (weak) MSO theories of DULS^{L}, UULS^{L}, DULS^{C}, UULS^{C} (i.e., the extensions of DULS and UULS with the equi-level and the equi-column predicates) are not decidable. Such results have been achieved by reducing several undecidable problems (e.g., the tiling problem over the two-dimensional infinite grid) to the model checking problems for the corresponding structures. Moreover, in the same paper, the authors prove the decidability of the model checking problem for the chain fragment of MSO logic interpreted over DULS^{L}, UULS^{L}, and UULS^{C}, but they leave open the problem for DULS^{C}.

We conclude the section by showing that the model checking problem for the chain fragment of MSO logic[6] interpreted over the structure $\text{TULS}^{\not<,\mathsf{L}_0}$ expanded with either the equi-level or the equi-column predicate is decidable. As a matter of fact, since MSO-definability of DULS and UULS with respect to $\text{TULS}^{\not<,\mathsf{L}_0}$ holds even if restricting to quantifications over chains, Theorem 14 and Theorem 15 below generalize previous results in the literature. Moreover, Theorem 15 answers positively to the open problem for $\text{DULS}^{\not<,\mathsf{C}}$ [43].

The following proofs are partly based on a method introduced by Thomas in [105], which allows one to reduce the monadic chain logic interpreted over a tree structure to the full MSO logic interpreted over a linear structure. As usual, we consider, for simplicity, 2-refinable layered temporal structures.

Theorem 14. *The model checking problem for monadic chain logic interpreted over the expanded structure $\mathit{TULS}^{\not<,\mathsf{L}_0,\mathsf{L}}$, namely, $\mathit{TULS}^{\not<,\mathsf{L}_0}$ equipped with the equi-level predicate, is decidable.*

Proof. The idea is to translate a given monadic chain sentence φ interpreted over the expanded structure $\text{TULS}^{\not<,\mathsf{L}_0,\mathsf{L}}$ to an equi-satisfiable MSO sentence $\vec{\varphi}$ interpreted over the linear order $(\mathbb{Z},<)$ expanded with the constant 0. Without loss of generality, we can assume that φ only uses existential quantifications over *non-empty* chains and atoms of the following forms:

- $\mathsf{X} \subseteq \mathsf{Y}$, meaning that 'the chain X is included in the chain Y',
- $\mathsf{E}_{a_j}(\mathsf{X},\mathsf{Y})$, meaning that '$\mathsf{X},\mathsf{Y}$ are singletons of the form $\{x\},\{y\}$, with y being the successor of x with respect to the j-th projection relation',
- $\mathsf{L}_0(\mathsf{X})$, meaning that 'X is a singleton of the form $\{x\}$, with $x \in \mathsf{L}_0$',
- $\mathsf{L}(\mathsf{X},\mathsf{Y})$, meaning that '$\mathsf{X},\mathsf{Y}$ are singletons of the form $\{x\},\{y\}$, with x and y belonging to the same layer'.

The translation from φ to $\vec{\varphi}$ is achieved by encoding each set variable X appearing in φ with a pair of variables $\mathsf{Z}_\mathsf{X}, \mathsf{W}_\mathsf{X}$, to be instantiated by subsets of $\mathbb{Z}$, as follows. Let $\mathsf{V} = \bigcup_{i\in\mathbb{Z}} \mathsf{L}_i$ be the domain of the structure $\text{TULS}^{\not<,\mathsf{L}_0,\mathsf{L}}$. As a preliminary remark, notice that for every non-empty chain H and for every index $i \in \mathbb{Z}$, there is *at most one* natural number n such that $(i,n) \in \mathsf{H}$. We say that a subset P of V is a *cover* of H if P is a maximal path (consisting of A-labeled edges) that includes H, namely, if $\mathsf{H} \subseteq \mathsf{P}$ and for every $i \in \mathbb{Z}$, there is exactly one $n \in \mathbb{N}$ such that $(i,n) \in \mathsf{P}$. We denote by P_H the leftmost cover of H, that is, the (unique) cover P_H such that, if i is the greatest integer satisfying $\exists\, n \in \mathbb{N}.\ (i,n) \in \mathsf{H}$, then every descendant of (i,n) along P_H is of

[6] Recall that the chain fragment of MSO logic obtained by allowing quantifications over chains only, namely, subsets of paths.

the form $\big(i', 2^{i'-i}n\big)$, with $i' \geqslant i$. We then define Z_H and W_H as the unique sets of integers such that

i) $i \in Z_H$ iff there is a (unique) *odd* natural number n satisfying $(i, n) \in P_H$, namely, (i, n) is the target of an a_2-*labeled* edge along the path P_H,

ii) $i \in W_H$ iff $(i, n) \in H$ for some $n \in \mathbb{N}$, namely, H intersects the layer L_i.

Notice that the encoding (Z_H, W_H) uniquely determines the non-empty chain H. Moreover, we can translate the above construction in the logic. Precisely, we introduce two set variables Z_X and W_X for each chain variable X in φ and we define $\vec{\varphi}$ recursively on the structure of φ as follows:

- if φ is of the form $X \subseteq Y$, then we let $\vec{\varphi}$ be $W_X \subseteq W_Y \;\wedge\; \big(Z_X = Z_Y \;\vee\; \exists\, w \in W_X.\ (\forall\, w' \in W_X.\ w \geqslant w' \;\wedge\; \forall\, z \leqslant w.\ z \in Z_X \leftrightarrow z \in Z_Y)\big)$ (meaning that 'W_X is included in W_Y and either $Z_X = Z_Y$ holds or W_X has a greatest element w and $Z_X \cap \{z \leqslant w\} = Z_Y \cap \{z \leqslant w\}$');
- if φ is of the form $E_{a_1}(X, Y)$, then we let $\vec{\varphi}$ be $\exists\, w.\ Z_X = Z_Y \;\wedge\; W_X = \{w\} \;\wedge\; W_Y = \{w+1\}$;
- if φ is of the form $E_{a_2}(X, Y)$, then we let $\vec{\varphi}$ be $\exists\, w.\ Z_X \cup \{w+1\} = Z_Y \;\wedge\; W_X = \{w\} \;\wedge\; W_Y = \{w+1\}$;
- if φ is of the form $L_0(X)$, then we let $\vec{\varphi}$ be $W_X = \{0\}$;
- if φ is of the form $L(X, Y)$, then we let $\vec{\varphi}$ be $\exists\, w.\ W_X = W_Y = \{w\}$;
- if φ is of the form $\varphi_1 \;\vee\; \varphi_2$, then we let $\vec{\varphi}$ be $\vec{\varphi}_1 \;\vee\; \vec{\varphi}_2$;
- if φ is of the form $\neg\varphi'$, then we let $\vec{\varphi}$ be $\neg\vec{\varphi}'$;
- if φ is of the form $\exists\, X.\ \varphi'$, then we let $\vec{\varphi}$ be $\exists\, Z_X.\ \exists\, W_X.\ \vec{\varphi}' \;\wedge\; W_X \neq \emptyset \;\wedge\; \forall\, w \in W_X.\ (\forall\, w' \in W_X.\ w \geqslant w') \;\rightarrow\; (\forall\, z \in Z_X.\ w \geqslant z)$ (meaning that 'there exist Z_X and $W_X \neq \emptyset$ satisfying $\vec{\varphi}'$ and, whenever W_X has a greatest element w, w is greater than or equal to every element of Z_X', namely, the cover corresponding to the encoding of the chain X is the leftmost one).

It is routine to check that φ holds in the structure TULS$^{\not<, L_0, L}$ iff $\vec{\varphi}$ holds in $(\mathbb{Z}, <)$. From the decidability of the MSO theory of $(\mathbb{Z}, <)$ [2, 87], it follows that the model checking problem for the monadic chain logic interpreted over the expanded structure TULS$^{\not<, L_0, L}$ is decidable.

Theorem 15. *The model checking problem for monadic chain logic interpreted over the expanded structure TULS$^{\not<, L_0, C}$, namely, TULS$^{\not<, L_0}$ equipped with the equi-column predicate, is decidable.*

Proof. In analogy to the previous proof, the idea is to translate a given monadic chain sentence φ interpreted over the expanded structure TULS$^{\not<, L_0, C}$ to an equi-satisfiable MSO sentence $\vec{\varphi}$ interpreted over $(\mathbb{Z} \cup \{\infty\}, <, neg)$, where ∞ denotes a special element not belonging to $\mathbb{Z}$ and *neg* denotes the sign-flipping relation $\big\{(z, -z) : z \in \mathbb{Z}\big\}$. As in the previous proof, we restrict ourselves to a setting where formulas are build up from atoms of the form $X \subseteq Y$,

$E_{a_j}(X,Y)$, $L_0(X)$, and $C(X,Y)$, via boolean connectives and existential quantifications over *non-empty* chains. Let $V = \bigcup_{i\in\mathbb{Z}} L_i$ be the domain of the structure $\text{TULS}^{\not<, L_0, C}$. Here, we encode non-empty chains in a slightly different way, which makes it possible to check whether two vertices lies on the same diagonal. Precisely, we encode any non-empty chain H with an integer s_H and three subsets Z_H, W_H, Q_H of $\mathbb{N}$ defined as follows. We denote by P_H the rightmost cover of H, formally, the superset of H that contains exactly one element (i,n) for each $i \in \mathbb{Z}$ and such that, whenever i is the greatest integer satisfying $\exists\, n \in \mathbb{N}.\ (i,n) \in H$, then every descendant of (i,n) along P_H is of the form $\big(i', 2^{i'-i}(n+1)-1\big)$, with $i' \geqslant i$. Then, we distinguish between two cases: either P_H coincides with the leftmost branch of $\text{TULS}^{\not<, L_0, C}$ (this happens if H is a downward infinite chain lying entirely on the leftmost branch), or there is a greatest index $i \in \mathbb{Z}$ such that $(i,0) \in P_H$. In the former case, we set $s_H = \infty$, $Z_H = \emptyset$, $W_H = \big\{i \in \mathbb{N} : (i,0) \in H\big\}$, and $Q_H = \big\{i \in \mathbb{N}_{>0} : (-i,0) \in H\big\}$. In the latter case, we let s_H be the greatest index $i \in \mathbb{Z}$ such that $(i,0) \in P_H$ and we define $Z_H, W_H, Q_H \subseteq \mathbb{N}$ as the unique sets of integers such that

i) $i \in Z_H$ iff there is a (unique) *odd* natural number n satisfying $(s_H+i,n) \in P_H$, (s_H+i,n) is the target of an a_2-*labeled* edge along the path P_H,
ii) $i \in W_H$ iff $(s_H+i,n) \in H$ for some $n \in \mathbb{N}$, H intersects the layer L_{s_H+i},
iii) $i \in Q_H$ iff $i > 0$ and $(s_H - i, 0) \in H$, namely, H intersects the layer L_{s_H-i}.

Notice that, in both cases, the encoding (s_H, Z_H, W_H, Q_H) uniquely determines the non-empty chain H. Switching to logic, we introduce, for each chain variable X, a vertex variable s_X and three set variables Z_X, W_X, and Q_X. Then, we define $\vec{\varphi}$ recursively on the structure of φ as follows:

- if φ is of the form $X \subseteq Y$, then we let $\vec{\varphi}$ be $W_X \subseteq W_Y \ \wedge\ Q_X \subseteq Q_Y \ \wedge\ \big((s_X = s_Y = \infty) \ \vee\ (s_X = s_Y \neq \infty \ \wedge\ Z_X = Z_Y) \ \vee\ (s_X = s_Y \neq \infty \ \wedge\ \exists\, w \in W_X.\ (\forall\, w' \in W_X.\ w \geqslant w' \ \wedge\ \forall\, z \leqslant w.\ z \in Z_X \leftrightarrow z \in Z_Y))\big)$;
- if φ is of the form $E'_{a_1}(X,Y)$, then we let $\vec{\varphi}$ be $\exists\, w.\ (s_X = w \ \wedge\ s_Y = w{+}1 \wedge Z_X = Z_Y \wedge W_X = W_Y = \{0\} \wedge Q_X = Q_Y = \emptyset) \vee \exists\, w.\ (s_X = s_Y \neq \infty \wedge Z_X = Z_Y \cup \{w+1\} \wedge W_X = \{w\} \wedge W_Y = \{w+1\} \wedge Q_X = Q_Y = \emptyset)$;
- if φ is of the form $E'_{a_2}(X,Y)$, then we let $\vec{\varphi}$ be $s_X = s_Y \neq \infty \ \wedge\ Z_X = Z_Y \ \wedge\ \exists\, w.\ W_X = \{w\} \ \wedge\ W_Y = \{w+1\} \ \wedge\ Q_X = Q_Y = \emptyset$;
- if φ is of the form $L_0(X)$, then we let $\vec{\varphi}$ be $W_X = \{\mathit{neg}(s_X)\} \ \wedge\ Q_X = \emptyset$;
- if φ is of the form $C(X,Y)$, then we let $\vec{\varphi}$ be $s_X \neq \infty \ \wedge\ s_Y \neq \infty \ \wedge\ Z_X = Z_Y \ \wedge\ \exists\, w.\ W_X = W_Y = \{w\} \ \wedge\ Q_X = Q_Y = \emptyset$;
- if φ is of the form $\varphi_1 \ \vee\ \varphi_2$, then we let $\vec{\varphi}$ be $\vec{\varphi}_1 \ \vee\ \vec{\varphi}_2$;
- if φ is of the form $\neg\varphi'$, then we let $\vec{\varphi}$ be $\neg\vec{\varphi}'$;
- if φ is of the form $\exists\, X.\ \varphi'$, then we let $\vec{\varphi}$ be the formula $\exists\, s_X.\ \exists\, Z_X, W_X, Q_X \subseteq \mathbb{N}.\ \vec{\varphi}' \ \wedge\ W_X \cup Q_X \neq \emptyset \ \wedge\ \big(s_X = \infty \ \wedge\ Z_H = \emptyset\big) \ \vee\ \big(s_X \neq \infty \wedge \forall\, w \in W_X.\ (\forall\, w' \in W_X.\ w \geqslant w') \ \rightarrow\ (\forall\, z > w.\ z \in Z_X)\big)$ (meaning that 'there exist s_X, Z_X, W_X, Q_X, with $W_X \cup Q_X \neq \emptyset$, satisfying $\vec{\varphi}'$ and

such that either $s_X = \infty$ and $Z_H = \emptyset$, or $s_X \neq \infty$, and, whenever W_X has a greatest element w, Z_X contains all elements greater than w', namely, either the chain X is bounded from below, and in that case $Z_H = \emptyset$, or the cover corresponding to the encoding of X is the rightmost one).

We let the reader check that φ holds in the expanded structure $\mathrm{TULS}^{\not<, L_0, C}$ iff $\vec{\varphi}$ holds in $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$. It remains to prove that $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$ has a decidable MSO theory. As a matter of fact, the relational structure $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$ can be defined inside the expanded linear order $(\mathbb{N}, <, P_{even}, P_{odd})$, where $P_{even} = \{2n : n \in \mathbb{N}\}$ and $P_{odd} = \{2n + 1 : n \in \mathbb{N}\}$, via an MSO-definable interpretation that maps

i) 0 to the element ∞ of $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$,

ii) the even numbers $2, 4, 6, ...$ to the non-negative numbers $0, 1, 2, ...$ of $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$,

iii) the odd numbers $1, 3, 5, ...$ to the negative numbers $-1, -2, -3, ...$ of $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$,

iv) the pairs of natural numbers x, y satisfying $\big(x = 1 \wedge y = 2\big) \vee \big(x \neq 0 \wedge P_{even}(x) \wedge P_{even}(y) \wedge x < y\big) \vee \big(P_{odd}(x) \wedge P_{odd}(y) \wedge y < x\big)$ to the $<$-labeled edges of $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$,

v) the pairs of natural numbers x, y satisfying $\big(x = 2 \wedge y = 2\big) \vee \big(P_{even}(x) \wedge x = y + 3\big) \vee \big(P_{odd}(x) \wedge y = x + 3\big)$ to the *neg*-labeled edges of $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$.

Since the predicates P_{even} and P_{odd} are MSO-definable in the linear order $(\mathbb{N}, <)$ and since MSO-definable interpretations preserve the decidability of MSO theories, we have that $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$ enjoys a decidable MSO theory and $\mathrm{TULS}^{\not<, L_0, C}$ as well.

3.6 Discussion

In this chapter, we described in full detail the contraction method for tree automata.

By taking advantage of a suitable notion of tree indistinguishability with respect to alternating Muller tree automata, we showed that the acceptance problem for a given colored tree can be reduced to the acceptance problem for a suitable retraction of it. Such a technique allowed us to give a uniform proof of the decidability of the MSO theories of a large class of tree structures, including the deterministic trees of the Caucal hierarchy and several trees outside it.

As mentioned in the beginning of this chapter, the contraction method for tree automata presents some similarities with Shelah's composition method [95], which directly exploits a notion of indistiguishability of relational structures with respect to monadic second-order formulas. As a matter of fact, we believe it possible to generalize both the contraction method and the composition method in order to effectively deal with model checking problems for generic relational structures.

We also introduced a suitable notion of rank for deterministic colored trees, which can be thought of as the number of iterated retractions which are sufficient to reduce a given tree to a regular one. An interesting problem, which is left open, consists of establishing whether the hierarchy of rank n trees, for n ranging over the set of natural numbers, is strictly increasing or not. Such a problem is related to the problem of determining the *minimum* rank of any given tree.

Subsequently, we analyzed some natural tree transformations, such as second-order tree substitutions, tree transductions, and unfoldings with backward edges and loops, and we disloced notable relationships between them. We then investigated compositional and closure properties of reducible (rank n) trees. Precisely, we proved (i) a composition theorem for (minor) types of trees with respect to second-order tree substitutions, (ii) the closure of the class of rank n trees with respect to tree transductions with rational lookahead (thus solving an open problem in [74]), and (iii) the closure of the class of reducible trees with respect to the operation of unfolding with backward edges and loops.

We finally provided meaningful applications of the contraction method. In particular, we proved that the languages recognized by two-way alternating Muller tree automata are rational, namely, recognizable by alternating Muller tree automata, and we proved that any morphic tree, namely, the limit of n-fold applications of any regular tree morphism, has rank 1.

In the last part of the chapter, we considered the model checking problem for (fragments of) MSO logics interpreted over the so-called layered temporal structures. First, we introduced a new notion of layered temporal structure, which we called totally unbounded layered temporal structure (TULS for short) and which generalizes previous definitions proposed in the literature. We proved that such a kind of structure (extended with a suitable coloring predicate) embeds both downward-unbounded and upward-unbounded layered temporal structures. Then, by exploiting the contraction method for tree automata, we established the decidability of the MSO theory of the TULS. Finally, we proved an original decidability result for the chain fragment of MSO logic interpreted over the TULS equipped with either the equi-level or the equi-column predicate.

4

Summary

The main contributions of the book can be summarized as follows:

- We explored the automaton-based approach to time granularity in full detail. In particular, we introduced suitable notions of single-string automata (possibly extended with counters), which allow one to compactly represent and efficiently manipulate single (ultimately periodic) time granularities. We provided effective procedures to solve the crucial problems of granule conversion, equivalence, and optimization of automaton-based representations of time granularities.
- We dealt with the problem of managing (possibly infinite) sets of time granularities. We characterized the subclass of Büchi automata that recognize exactly the rational ω-languages consisting of ultimately periodic words only (these languages represent possibly infinite sets of ultimately periodic time granularities) and we provided efficient solutions to several basic problems (the emptiness problem, the acceptance problem, the equivalence problem, the inclusion problem, and the state optimization problem), as well as to the granularity comparison problem.
- We addressed the model checking problem for monadic second-order logics interpreted over deterministic vertex-colored trees. We generalized Elgot and Rabin's contraction method to deal with such a problem and we introduced a large class of trees, called reducible trees, which naturally extends that of regular trees and for which the model checking problem turns out to be decidable.
- We identified several natural operations on trees (e.g., tree morphisms, tree transductions, unfoldings with backward edges and loops) and we proved closure properties of the class of reducible trees with respect to such operations. Then, we exploited these closure properties to prove that the class of reducible trees includes all deterministic trees in the Caucal hierarchy as well as several trees outside it. Moreover, we gave various application examples of the contraction method. In particular, we

exploited it to show the equivalence between one-way and two-way alternating Muller tree automata and to prove the decidability of the model checking problem for morphic trees.

- We introduce the class of totally-unbounded layered temporal structures, whose theories subsume the theories of previously known layered temporal structures (i.e., n-layered structures, downward-unbounded and upward-unbounded layered structures). We proved the decidability of the monadic second-order theories of these structures and the decidability of the model checking problem for the chain fragment of monadic second-order logic interpreted over the totally-unbounded layered temporal structures extended with either the equi-level or the equi-column predicate. These results subsume and extend previous results in the literature.

Finally, some open problems deserve further investigations:

- To either find a deterministic polynomial-time algorithm that solves the equivalence problem for nested counter single-string automata or prove that such a problem is co-NP-complete (the current solution is based on a polynomial-time non-deterministic algorithm that tests the non-equivalence of two given nested counter single-string automata).
- To improve the algorithms that optimize the complexity and the number of states of nested counter single-string automata (currently, the proposed algorithms take polynomial time with respect to the given string-based specifications, but they may require exponential time with respect to equivalent automaton-based specifications).
- To establish whether the hierarchy of rank n trees is strictly increasing or not.
- To extend the notion of tree retraction in order to deal with higher-order morphic trees and trees generated by higher-order recursive schemes.
- To generalize the contraction method in order to effectively deal with the model checking problem for monadic second-order logic interpreted over generic relational structures.

A

Technical Proofs

A.1 Proofs of Theorem 5 and Theorem 6

This section is devoted to prove Theorem 5 and Theorem 6. For the rest of this section, we fix the following objects:

- a (non-empty full) D-augmented tree T;
- a B-labeled factorization Π of T, with $\mathsf{B} \supseteq \mathsf{D}$;
- a B-augmented tree automaton $\mathcal{A} = (\mathsf{A}, \mathsf{C}, \mathsf{B}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$;
- the retraction $\widetilde{\mathsf{T}}$ of T with respect to $\mathcal{A}$ and Π;
- the encoding $\overrightarrow{\mathsf{T}}$ of $\widetilde{\mathsf{T}}$;
- the retraction automaton $\overrightarrow{\mathcal{A}} = (\mathsf{B}, \mathscr{T}_{\mathcal{A}} \cup \{\bot\}, \overrightarrow{\mathsf{S}}, \overrightarrow{\Delta}, \overrightarrow{\mathcal{I}}, \overrightarrow{\mathcal{F}})$.

Before entering the technical details of the proofs of the two main theorems, we provide some preliminary definitions and results about tree decompositions. To easily understand the contents of this section, the reader should keep in mind that factorizations, as well as unmarked and marked factors, can be defined over arbitrary trees, e.g., over uncolored trees, non-deterministic trees, and unlabeled trees. As a matter of fact, given a tree T and a run R of the automaton $\mathcal{A}$ on T, every factorization Π of T induces a corresponding factorization Γ of R, defined as follows:

- the vertices of Γ are all and only the vertices v of R such that $\downarrow_1 \mathsf{R}(v)$ is a vertex of Π;
- (v, v') is a b-labeled edge in Γ iff v is an ancestor of v' in Π' and $(\downarrow_1 \mathsf{R}(v)$, $\downarrow_1 \mathsf{R}(v'))$ is a b-labeled edge in Π.

Accordingly, we denote by $\mathsf{R}_\Gamma[v]$ the unmarked factor of R rooted at v with respect to Γ (note that v is a vertex of Γ).

In order to keep the notation simple, we shall use some natural shorthands, which should be clear from the context. As an example, given a run R of an alternating tree automaton and a vertex v of R, we shall use the simplified notation $\mathcal{I}mg(\mathsf{R}|\nu)$, in place of $\mathcal{I}mg(\mathsf{R}|\pi_\nu)$, to denote the set of all states that occur at least once along the access path π_ν of ν in R. Similarly, given a path π in R, we shall use $\mathsf{R}(\pi)$, in place of $\mathsf{R}(\nu)$, to denote the color of the vertex ν of R whose access path is π.

Definition 39. *Let T be a generic tree and Π a factorization of it. For every vertex ν of a tree T, the* normal decomposition *of ν with respect to Π is the (unique) sequence*

$$\nu_0 \xrightarrow{\pi_0} \nu_1 \xrightarrow{\pi_1} \dots \xrightarrow{\pi_{n-1}} \nu_n \xrightarrow{\pi_n} \nu$$

such that

- $\nu_0\, \nu_1 \dots \nu_n$ *identifies a finite path in Π that starts from the root;*
- *each π_i, with $0 \leqslant i < n$, is a finite path inside the unmarked factor $\mathsf{T}_\Pi[\nu_i]$ that reaches a leaf of $\mathsf{T}_\Pi[\nu_i]$;*
- π_n *is a finite path inside the unmarked factor $\mathsf{T}_\Pi[\nu_n]$ that reaches a leaf of $\mathsf{T}_\Pi[\nu_n]$ iff ν is a leaf of T;*
- *for all $0 \leqslant i < n$, the sequence $\pi_0 \cdot \pi_1 \cdot \dots \cdot \pi_i$ yields the access path of the vertex ν_{i+1} in T;*
- *the sequence $\pi_0 \cdot \pi_1 \cdot \dots \cdot \pi_n$ yields the access path of the vertex ν in T.*

Since every vertex in a tree is identified by its access path, we can extend the notion of normal decomposition to finite paths in the obvious way.

It is easy to see that, for every vertex ν of T having normal decomposition $\nu_0 \xrightarrow{\pi_0} \nu_1 \xrightarrow{\pi_1} \dots \xrightarrow{\pi_{n-1}} \nu_n \xrightarrow{\pi_n} \nu$, we have:

i) for all $0 \leqslant i < n$, $\mathsf{T}^+_\Pi[\nu_i](\pi_i)$ is the label of the edge (ν_i, ν_{i+1}) of Π;
ii) $\mathsf{T}^+_\Pi[\nu_n](\pi_n) = \mathsf{T}(\nu)$.

Moreover, given a run R, a factorization Γ of R, and a vertex ν of R with normal decomposition $\nu_0 \xrightarrow{\pi_0} \nu_1 \xrightarrow{\pi_1} \dots \xrightarrow{\pi_{n-1}} \nu_n \xrightarrow{\pi_n} \nu$, we have

$$\mathcal{I}mg(\mathsf{R}|\nu) = \bigcup_{0 \leqslant i \leqslant n} \mathcal{I}mg(\mathsf{R}_\Gamma[\nu_i]|\pi_i).$$

An analogous notion of decomposition can be given for the infinite paths of a tree T. Below, we distinguish between paths that traverse finitely many vertices of Π and paths that traverse infinitely many vertices of Π.

Definition 40. *For every infinite path π in T that traverses only finitely many vertices of Π, the* normal decomposition *of π with respect to Π is the (unique) finite sequence*

$$\nu_0 \xrightarrow{\pi_0} \nu_1 \xrightarrow{\pi_1} \dots \xrightarrow{\pi_{n-1}} \nu_n \xrightarrow{\pi_n}$$

such that

- $v_0\, v_1 \ldots v_n$ *identifies a finite path in* Π *that starts from the root;*
- *each* π_i*, with* $0 \leqslant i < n$*, is a finite path inside the unmarked factor* $T_\Pi[v_i]$ *that reaches a leaf of* $T_\Pi[v_i]$*;*
- π_n *is an infinite path inside the unmarked factor* $T_\Pi[v_n]$*;*
- *for all* $0 \leqslant i < n$*, the sequence* $\pi_0 \cdot \pi_1 \cdot \ldots \cdot \pi_i$ *yields the access path of the vertex* v_{i+1} *in* T*;*
- *the sequence* $\pi_0 \cdot \pi_1 \cdot \ldots \cdot \pi_n$ *yields the infinite path* π *in* T*.*

Given a run R and a factorization Γ of R, for every infinite path π in R that traverses only finitely many vertices of Π and has normal decomposition $v_0 \xrightarrow{\pi_0} v_1 \xrightarrow{\pi_1} \ldots \xrightarrow{\pi_{n-1}} v_n \xrightarrow{\pi_n}$, we have

$$\mathcal{I}nf(R|\pi) = \mathcal{I}nf(R_\Gamma[v_n]|\pi_n).$$

Definition 41. *For every infinite path* π *in* T *that traverses infinitely many vertices of* Π*, the* normal decomposition *of* π *with respect to* Π *is the (unique) infinite sequence*

$$v_0 \xrightarrow{\pi_0} v_1 \xrightarrow{\pi_1} v_2 \xrightarrow{\pi_2} \ldots$$

such that

- $v_0\, v_1\, v_2 \ldots$ *identifies an infinite path in* Π *that starts from the root;*
- *each* π_i*, with* $i \geqslant 0$*, is a finite path inside the unmarked factor* $T_\Pi[v_i]$ *that reaches a leaf of* $T_\Pi[v_i]$*;*
- *for all* $i \geqslant 0$*, the sequence* $\pi_0 \cdot \pi_1 \cdot \ldots \cdot \pi_i$ *yields the access path of the vertex* v_{i+1} *in* T*;*
- *the sequence* $\pi_0 \cdot \pi_1 \cdot \pi_2 \cdot \ldots$ *yields the infinite path* π *in* T*.*

Given a run R and a factorization Γ of R, for every infinite path π in R that traverses infinitely many vertices of Π and has normal decomposition $v_0 \xrightarrow{\pi_0} v_1 \xrightarrow{\pi_1} v_2 \xrightarrow{\pi_2} \ldots$, we have

$$\mathcal{I}nf(R|\pi) = \big\{ s \in S \,:\, \exists^\omega\, i \geqslant 0.\ s \in \mathcal{I}mg(R_\Gamma[v_i]|\pi_i) \big\}.$$

Now, we can prove Properties P1–P3, which were first stated in Section 3.2.3. Subsequently, we shall prove Theorem 5 and Theorem 6.

Lemma 15 (Property P1). *For every run* $\vec{R}$ *of* $\vec{\mathcal{A}}$ *on* $\vec{T}$ *such that the state at the root of* $\vec{R}$ *is a quadruple of the form* $(b, s, \{s\}, \{s\})$*, there is a run* R *of* $\mathcal{A}$ *on* T *which is mimicked by* $\vec{R}$*.*

Proof. Let us fix a run $\vec{R}$ of $\vec{\mathcal{A}}$ on $\vec{T}$ satisfying the hypothesis of the lemma. To start with, we build a suitable run R of $\mathcal{A}$ on T and, at the same time, we define the factorization Γ of R induced by Π and we associate with each vertex $v \in \mathcal{D}om(\Gamma) \cup \mathcal{F}r(R)$ a corresponding vertex $\vec{v}$ of $\vec{R}$. These objects are defined by exploiting mutual induction, starting from the root of $\vec{R}$, as follows.

- Let us denote by $\vec{v}$ the root of $\vec{R}$ and let $\vec{s} = \downarrow_2(\vec{R}(\vec{v}))$ be the state associated with it in $\vec{R}$. Furthermore, let v be the root of R, which obviously belongs to the factorization Γ and corresponds to the vertex $\vec{v}$ of $\vec{R}$.

  By hypothesis, we know that the state $\vec{s}$ is a quadruple of the form $(b, s, \{s\}, \{s\})$. Hence, we can define $R(v) = (\varepsilon, s)$ (intuitively, the first component ε of $R(v)$ represents the root of T, while the second component s of $R(v)$ represents the state of the automaton $\mathcal{A}$ at the root of T).
- Suppose, by inductive hypothesis, that R is defined over a vertex $v \in \mathcal{D}om(\Gamma)$, $R(v) = (u, s)$, $\vec{v}$ is the corresponding vertex of $\vec{R}$, and $\vec{R} = (\vec{u}, \vec{s})$ (note that u is a vertex of T, s is a state of the automaton $\mathcal{A}$, $\vec{u}$ is a vertex of $\vec{T}$, and $\vec{s}$ is a state of the automaton $\vec{\mathcal{A}}$ of the form (b, s, Y, Z), with $b \in B$ and $Y, Z \subseteq S$).

  We have to define the successors of v in Γ that correspond to the successors of $\vec{v}$ in $\vec{R}$. Moreover, we have to describe the portion of the tree R which is delimited by the vertex v and its successors in Γ. First, notice that the vertex $\vec{v}$ of $\vec{R}$ has finite out-degree. Since the vertex $\vec{u}$ of $\vec{T}$ is colored with an $\mathcal{A}$-type σ, we know that all successors of $\vec{v}$ in $\vec{R}$ are marked with states of the form X $(\subseteq S)$ or (b, s, Y, Z) $(\in B \times S \times \mathscr{P}(S) \times \mathscr{P}(S)$. Thus, we can denote by $\vec{v}_i$, for i ranging over a suitable finite set I of indices, all and only the successors of the vertex $\vec{v}$ in $\vec{R}$ that are marked with states of the form $\vec{s}_i = X_i$. Similarly, we can denote by $\vec{v}'_j$, for j ranging over a suitable finite set J, all and only the successors of the vertex $\vec{v}$ in $\vec{R}$ that are marked with states of the form $\vec{s}'_j = (b_j, s_j, Y_j, Z_j)$.

  Now, from the definition of the transition function of $\vec{\mathcal{A}}$, we know that there exists a run $R_{\vec{v}}$ of $\mathcal{A}$ on the marked factor $T^+_\Pi[u]$ such that

  i) the state at the root of $R_{\vec{v}}$ coincides with the state at the vertex v of R;
  ii) the sets $\mathcal{I}nf(R_{\vec{v}}|\pi)$, where π ranges over all infinite paths of $R_{\vec{v}}$, are all and only the sets X_i, where i ranges over I;
  iii) the quadruples $(T^+_\Pi[u](u_w),\ s_w,\ \mathcal{I}mg(R|v) \cup \mathcal{I}mg(R_{\vec{v}}|w),\ \mathcal{I}mg(R_{\vec{v}}|w))$, where $u_w = \downarrow_1 R_{\vec{v}}(w)$, $s_w = \downarrow_2 R_{\vec{v}}(w)$, and w ranges over all leaves of $R_{\vec{v}}$, are all and only the quadruples (b_j, s_j, Y_j, Z_j), where j ranges over J.

  Therefore, the portion of the tree R which is delimited by the vertex v and its successors in Γ is obtained by appending the run $R_{\vec{v}}$ to the vertex v of R. More precisely, given a vertex w of $R_{\vec{v}}$, we shortly denote by u_w the vertex $\downarrow_1 R_{\vec{v}}(w)$ of T and by s_w the state $\downarrow_2 R_{\vec{v}}(w)$ associated with the vertex w of $R_{\vec{v}}$. Then, for every vertex w of $R_{\vec{v}}$, we define $R(v \cdot w) = (u \cdot u_w, s_w)$.

  Accordingly, for each leaf w of $R_{\vec{v}}$, we choose an index $j \in J$ such that

  i) $b_j = T^+_\Pi[u](u_w)$,
  ii) $s_j = \downarrow_2 s_w$,

iii) $Y_j = \mathcal{I}mg(R|v) \cup \mathcal{I}mg(R_{\vec{v}}|w)$,

iv) $Z_j = \mathcal{I}mg(R_{\vec{v}}|w)$,

and we let $v \cdot w$ be the vertex of Γ that corresponds to the successor $\vec{v}'_j$ of $\vec{v}$ in $\vec{R}$. Finally, if $u \cdot u_w$ is a vertex of Π, then we let $(v, v \cdot w)$ be a b_j-labeled edge in Γ.

It is easy to verify that the tree R defined above is a valid run of $\mathcal{A}$ on T and that Γ is the factorization of R induced by Π.

It remains to show that $\vec{R}$ mimics R, namely, that Conditions C1–C6 of Definition 24 hold. It is immediate to see that Condition C1 holds.

To prove that Condition C2 holds, we consider an infinite path $\vec{\pi}$ in $\vec{R}$ such that $\mathcal{I}nf(\vec{R}|\vec{\pi})$ is a singleton of the form $\{X\}$, with $X \subseteq S$. From the definition of the transition function of $\vec{\mathcal{A}}$, we know that $\vec{\pi}$ contains an edge $(\vec{v}, \vec{v}')$ such that (i) the state $\vec{s}$ that appears at $\vec{v}$ is a quadruple in $B \times S \times \mathscr{P}(S) \times \mathscr{P}(S)$ and (ii) the state $\vec{s}'$ that appears at $\vec{v}'$ coincides with X. Moreover, by construction, there exist a vertex v of Γ, which corresponds to $\vec{v}$, and an infinite path τ in $R_\Gamma[v]$ such that $\mathcal{I}nf(R_\Gamma[v]|\tau) = X$. Therefore, we conclude that R contains an infinite path π, which is obtained by concatenating the access path of v and the infinite path τ, such that $\mathcal{I}nf(R|\pi) = \mathcal{I}nf(R_\Gamma[v]|\tau)$ $(= X)$.

As for Condition C3, let us consider an infinite path $\vec{\pi}$ in $\vec{R}$ such that $\mathcal{I}nf(\vec{R}|\vec{\pi})$ is a set of the form $\{(b_1, s_1, Y, Z_1), ..., (b_k, s_k, Y, Z_k)\}$. This means that $\vec{\pi}$ traverses infinitely many vertices $\vec{v}_0, \vec{v}_1, \vec{v}_2, ...$ in $\vec{R}$, which are marked by states in $B \times S \times \mathscr{P}(S) \times \mathscr{P}(S)$. By construction, there exists an infinite sequence of vertices $v_0, v_1, v_2, ...$ of Γ, which correspond to $\vec{v}_0, \vec{v}_1, \vec{v}_2, ...$, that describes an infinite path π in R having normal decomposition $v_0 \xrightarrow{\pi_0} v_1 \xrightarrow{\pi_1} v_2 \xrightarrow{\pi_2} ...$. Thus, we have $\mathcal{I}nf(R|\pi) = \big\{s \in S : \exists^\omega\, i \geqslant 0.\ s \in \mathcal{I}mg(R_\Gamma[v_i]|\pi_i)\big\}$. Moreover, it is easy to see that a state s belongs to $\mathcal{I}mg(R_\Gamma[v_i]|\pi_i)$ for infinitely many indices $i \geqslant 0$ iff s belongs to Z_l, for some $1 \leqslant l \leqslant k$. Thus, we can conclude that $\mathcal{I}nf(R|\pi) = \bigcup_{1 \leqslant l \leqslant k} Z_l$.

As for Condition C4, we consider an infinite path $\vec{\pi}$ in $\vec{R}$ such that $\mathcal{I}nf(\vec{R}|\vec{\pi})$ is a singleton of the form $\{(b, s, Y)\}$. From the definition of the transition function of $\vec{\mathcal{A}}$, we know that $\vec{\pi}$ contains an edge $(\vec{v}, \vec{v}')$ such that (i) the state $\vec{s}$ that appears at $\vec{v}$ is a quadruple in $B \times S \times \mathscr{P}(S) \times \mathscr{P}(S)$ and (ii) the state $\vec{s}'$ that appears at $\vec{v}'$ coincides with the triple (b, s, Y). Therefore, there exists a leaf v' of R, which corresponds to $\vec{v}'$, that satisfies $\vec{s}' = (T(\downarrow_1 R(v)),\ \downarrow_2 R(v),\ \mathcal{I}mg(R|v))$.

We omit the proofs for Condition C5 and Condition C6, which can be obtained by using symmetric arguments. □

Lemma 16 (Property P2). *For every run R of $\mathcal{A}$ on T, there is a run R' of $\mathcal{A}$ on T such that $R' \preceq R$ and there is a run $\vec{R}$ of $\vec{\mathcal{A}}$ on $\vec{T}$ that mimics R'.*

Proof. Let us fix a run R of $\mathcal{A}$ on T and let us denote by Γ the factorization of R induced by T. First of all, we have to build a run $\vec{\mathsf{R}}$ of $\vec{\mathcal{A}}$ on $\vec{\mathsf{T}}$ and, for each vertex $\vec{v}$ of $\vec{\mathsf{R}}$ with an associated state of the form (b, s, Y, Z), we have to identify a corresponding vertex v in $\mathcal{D}om(\Gamma) \cup \mathcal{F}r(\mathsf{T})$ such that $\downarrow_2 \mathsf{R}(v) = s$.

- Let us denote by v the root of R and let $s = \downarrow_2 \mathsf{R}(v)$ be the state associated with it in R. Furthermore, let $\vec{v}$ be the root of $\vec{\mathsf{R}}$, which obviously corresponds to v.

  We define $\vec{\mathsf{R}}(\vec{v}) = (\varepsilon, \vec{s})$, where $\vec{s} = (b, s, \{s\}, \{s\})$ and b is an arbitrary label from the set B (intuitively, the first component ε of $\vec{\mathsf{R}}(\vec{v})$ represents the root of T, while the second component $\vec{s}$ of $\vec{\mathsf{R}}(\vec{v})$ represents the state of the automaton $\vec{\mathcal{A}}$ at the root of T).

- Suppose, by inductive hypothesis, that $\vec{\mathsf{R}}$ is defined over a vertex $\vec{v}$, $\vec{\mathsf{R}}(\vec{v}) = (\vec{u}, \vec{s})$, v is the designated vertex in $\mathcal{D}om(\Pi) \cup \mathcal{F}r(\mathsf{T})$ that corresponds to $\vec{v}$, and $\mathsf{R}(v) = (u, s)$, with u being a suitable vertex in T.

  If the state $\vec{s}$ that appears at $\vec{v}$ is of the form X ($\subseteq S$) or of the form (b, s, Y) ($\in B \times S \times \mathscr{P}(S)$), then, for each label $b' \in B$, we let $\vec{v}_{b'}$ be a successor of $\vec{v}$ in $\vec{\mathsf{R}}$ and we define $\vec{\mathsf{R}}(\vec{v}_{b'}) = (\vec{u} \cdot b', \vec{s})$.

  Otherwise, the state $\vec{s}$ must be of the form (b, s, Y, Z) ($\in B \times S \times \mathscr{P}(S) \times \mathscr{P}(S)$). In such a case, we know that the vertex v ($\in \mathcal{D}om(\Pi) \cup \mathcal{F}r(\mathsf{T})$), which, by inductive hypothesis, corresponds to $\vec{v}$, satisfies $\downarrow_2 \mathsf{R}(v) = s$. Recall that the first component $\vec{u}$ of $\vec{\mathsf{R}}(\vec{v})$ represents a vertex of $\vec{\mathsf{T}}$. If $\vec{\mathsf{T}}(\vec{u}) = \perp$, then, for each label $b' \in B$, we let $\vec{v}_{b'}$ be a successor of $\vec{v}$ in $\vec{\mathsf{R}}$ and we define $\vec{\mathsf{R}}(\vec{v}_{b'}) = (\vec{u} \cdot b', \vec{s}\,')$, where $\vec{s}\,' = (b, s, Y)$. Otherwise, if $\vec{\mathsf{T}}(\vec{u})$ is an $\mathcal{A}$-type σ, then we know that there exists a feature $t \in \sigma$ of the form
$$\begin{pmatrix} r \\ \{F_i \ : \ i \in I\} \\ \{(b_j,\, r_j,\, G_j) \ : \ j \in J\} \end{pmatrix}$$
  that satisfies the following conditions:

  i) $r = \downarrow_2 \mathsf{R}(v)$ $(= s)$;
  ii) the sets F_i, for i ranging over I, are all and only the sets $\mathcal{I}nf(\mathsf{R}_\Gamma[v]|\pi)$, for π ranging over all infinite paths in $\mathsf{R}_\Gamma[v]$;
  iii) the triples (b_j, r_j, G_j), for j ranging over J, are all and only the triples $(\mathsf{T}^+_\Pi[u](u_w),\ s_w,\ \mathcal{I}mg(\mathsf{R}_\Gamma[v]|w))$, where $u_w = \downarrow_1 \mathsf{R}_\Gamma(w)$ and $s_w = \downarrow_2 \mathsf{R}_\Gamma(w)$, for w ranging over all leaves of $\mathsf{R}_\Gamma[v]$.

  Now, we can define the successors of the vertex $\vec{v}$ in $\vec{\mathsf{R}}$. For each label $b' \in B$ and for each index $i \in I$, we let $\vec{v}_{b',i}$ be a successor of $\vec{v}$ in $\vec{\mathsf{R}}$ and we define $\vec{\mathsf{R}}(\vec{v}_{b',i}) = (\vec{u} \cdot b', \vec{s}_i)$, where $\vec{s}_i = F_i$. Similarly, for each label $b' \in B$ and for each index $j \in J$ such that $b_j = b'$, we let $\vec{v}\,'_{b',j}$

be a successor of $\vec{v}$ in $\vec{R}$ and we define $\vec{R}(\vec{v}'_{b',j}) = (\vec{u} \cdot b', \vec{s}'_{b',j})$, where $\vec{s}'_{b',j} = (b_j, r_j, Y \cup G_j, G_j)$.

It remains to identify the vertices in $\mathcal{D}om(\Gamma) \cup \mathcal{F}r(T)$ that correspond to the above defined successors $\vec{v}'_{b',j}$ of $\vec{v}$. For each label $b' \in B$ and for each index $j \in J$ such that $b_j = b'$, we denote by $w_{b',j}$ an arbitrary leaf of $R_\Gamma[v]$ that satisfies (i) $T^+_\Pi[u](u_{b',j}) = b_j$, where $u_{b',j} = \downarrow_1 R_\Gamma[v](w_{b',j})$, (ii) $\downarrow_2 R_\Gamma[v](w_{b',j}) = r_j$, and (iii) $\mathcal{I}mg(R_\Gamma[v]|w_{b',j}) = G_j$. Finally, we let $v'_{b',j} = v \cdot w_{b',j}$ be the vertex in $\mathcal{D}om(\Gamma) \cup \mathcal{F}r(T)$ that corresponds to $\vec{v}'_{b',j}$.

It is routine to check that the tree $\vec{R}$ is a valid run of $\vec{\mathcal{A}}$ on $\vec{T}$. Moreover, the root of $\vec{R}$ is marked with a state of the form $(b, s, \{s\}, \{s\})$. Hence, by Lemma 15, there is another run R' of $\mathcal{A}$ on T which is mimicked by $\vec{R}$.

To complete the proof, we have to show that $R' \preceq R$, namely, that the following properties hold:

i) the state that appears at the root of R' coincides with the state that appears at the root of R;

ii) for every infinite path π' in R', there is an infinite path π in R such that $\mathcal{I}nf(R'|\pi') = \mathcal{I}nf(R|\pi)$;

iii) for every leaf v' of R', there is a leaf v of R such that $T(\downarrow_1 R'(v')) = T(\downarrow_1 R(v))$, $\downarrow_2 R'(v') = \downarrow_2 R(v)$, and $\mathcal{I}mg(R'|v') = \mathcal{I}mg(R|v)$.

The first property holds trivially by construction.

As for the second property, let us consider an infinite path π' in R'. Since $\vec{R}$ mimics R', there is an infinite path $\vec{\pi}$ in $\vec{R}$ that satisfies one of the following conditions:

1. $\mathcal{I}nf(\vec{R}|\vec{\pi})$ is a singleton of the form $\{X\}$ such that $X = \mathcal{I}nf(R'|\pi')$;
2. $\mathcal{I}nf(\vec{R}|\vec{\pi})$ is a set of the form $\{(b_1, s_1, Y, Z_1), ..., (b_k, s_k, Y, Z_k)\}$ that satisfies $\bigcup_{1 \leqslant l \leqslant k} Z_l = \mathcal{I}nf(R'|\pi')$.

In the former case, we know, by construction, that $\vec{\pi}$ contains an edge $(\vec{v}, \vec{v}\,')$ such that (i) the state associated with $\vec{v}$ is a quadruple in $B \times S \times \mathscr{P}(S) \times \mathscr{P}(S)$ and (ii) the state associated with $\vec{v}\,'$ coincides with the set X. This implies that there exist a vertex v of Γ, which corresponds to $\vec{v}$, and an infinite path τ in $R_\Gamma[v]$ such that $\mathcal{I}nf(R_\Gamma[v]|\tau) = X$. Hence, R contains an infinite path π, which is obtained by concatenating the access path of v and the infinite path τ, that satisfies $\mathcal{I}nf(R|\pi) = \mathcal{I}nf(R_\Gamma[v]|\tau)$ $(= X = \mathcal{I}nf\big(R'|\pi'\big))$. In the latter case, we know that $\vec{\pi}$ traverses infinitely many vertices $\vec{v}_0, \vec{v}_1, \vec{v}_2, ...$ in $\vec{R}$, which are marked with states from $B \times S \times \mathscr{P}(S) \times \mathscr{P}(S)$. Let $v_0, v_1, v_2, ...$ be the infinite sequence of vertices of Γ that correspond to $\vec{v}_0, \vec{v}_1, \vec{v}_2, ...$. Such a sequence describes an infinite path π in R, which has normal decomposition $v_0 \xrightarrow{\pi_0} v_1 \xrightarrow{\pi_1} v_2 \xrightarrow{\pi_2} ....$ Hence, from properties of normal decompositions, we know that $\mathcal{I}nf(R|\pi) = \big\{s \in S : \exists^\omega\, i \geqslant 0.\ s \in \mathcal{I}mg(R_\Gamma[v_i]|\pi_i)\big\}$. We thus conclude that $\mathcal{I}nf(R|\pi) = \bigcup_{1 \leqslant l \leqslant k} Z_l$ $(= \mathcal{I}nf(R'|\pi'))$.

As for the last property, let us consider a leaf ν' of R'. Since $\overrightarrow{\mathsf{R}}$ mimics R', there is an infinite path $\vec{\pi}$ in $\overrightarrow{\mathsf{R}}$ such that $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ is a singleton of the form $\{(\mathsf{b}, \mathsf{s}, \mathsf{Y})\}$, where $\mathsf{b} = \mathsf{T}(\downarrow_1 \mathsf{R}'(\nu'))$, $\mathsf{s} = \downarrow_2 \mathsf{R}'(\nu')$, and $\mathsf{Y} = \mathcal{I}mg(\mathsf{R}'|\nu')$. From the definition of the transition function of $\vec{\mathcal{A}}$, we know that $\vec{\pi}$ must contain an edge $(\vec{\nu}, \vec{\nu}\,')$ such that (i) the state associated with $\vec{\nu}$ is a quadruple from $\mathsf{B} \times \mathsf{S} \times \mathscr{P}(\mathsf{S}) \times \mathscr{P}(\mathsf{S})$ and (ii) the state associated with $\vec{\nu}\,'$ coincides with the triple $(\mathsf{b}, \mathsf{s}, \mathsf{Y})$. This implies that there is a leaf ν of R, which corresponds to $\vec{\nu}$, such that $\mathsf{T}(\downarrow_1 \mathsf{R}(\nu)) = \mathsf{b}$ $(= \mathsf{T}(\downarrow_1 \mathsf{R}'(\nu')))$, $\downarrow_2 \mathsf{R}(\nu) = \mathsf{s}$ $(= \downarrow_2 \mathsf{R}'(\nu'))$, and $\mathcal{I}mg(\mathsf{R}|\nu) = \mathsf{Y}$ $(= \mathcal{I}mg(\mathsf{R}'|\nu'))$. □

Lemma 17 (Property P3). *Let* R_1 *and* R_2 *be two runs of* $\mathcal{A}$ *on* T *and let* $\overrightarrow{\mathsf{R}}_1$ *and* $\overrightarrow{\mathsf{R}}_2$ *be two runs of* $\vec{\mathcal{A}}$ *on* $\vec{\mathsf{T}}$. *If* $\overrightarrow{\mathsf{R}}_1$ *mimics* R_1, $\overrightarrow{\mathsf{R}}_2$ *mimics* R_2, *and* $\overrightarrow{\mathsf{R}}_1 \preceq \overrightarrow{\mathsf{R}}_2$, *then* $\mathsf{R}_1 \preceq \mathsf{R}_2$.

Proof. Since $\overrightarrow{\mathsf{R}}_1 \preceq \overrightarrow{\mathsf{R}}_2$, $\overrightarrow{\mathsf{R}}_1$ mimics R_1, and $\overrightarrow{\mathsf{R}}_2$ mimics R_2, we have

$$\begin{aligned}\downarrow_2 \overrightarrow{\mathsf{R}}_1(\varepsilon) &= \big(\mathsf{b}, \downarrow_2\mathsf{R}_1(\varepsilon), \{\downarrow_2\mathsf{R}_1(\varepsilon)\}, \{\downarrow_2\mathsf{R}_1(\varepsilon)\}\big)\\ &= \big(\mathsf{b}, \downarrow_2\mathsf{R}_2(\varepsilon), \{\downarrow_2\mathsf{R}_2(\varepsilon)\}, \{\downarrow_2\mathsf{R}_2(\varepsilon)\}\big) = \downarrow_2 \overrightarrow{\mathsf{R}}_2(\varepsilon)\end{aligned}$$

from which $\downarrow_2\mathsf{R}_1(\varepsilon) = \downarrow_2\mathsf{R}_2(\varepsilon)$ follows.

Now, let π_1 be an infinite path in R_1. Since $\overrightarrow{\mathsf{R}}_1$ mimics R_1 (see Condition C5), there exists and infinite path $\vec{\pi}_1$ in $\overrightarrow{\mathsf{R}}_1$ that satisfies one of the following conditions:

1. $\mathcal{I}nf(\overrightarrow{\mathsf{R}}_1|\vec{\pi}_1)$ is a singleton of the form $\{\mathsf{X}\}$ such that $\mathsf{X} = \mathcal{I}nf(\mathsf{R}_1|\pi_1)$;
2. $\mathcal{I}nf(\overrightarrow{\mathsf{R}}_1|\vec{\pi}_1)$ is a set of the form $\{(\mathsf{b}_1, \mathsf{s}_1, \mathsf{Y}, \mathsf{Z}_1), ..., (\mathsf{b}_k, \mathsf{s}_k, \mathsf{Y}, \mathsf{Z}_k)\}$, that satisfies $\bigcup_{1 \leqslant l \leqslant k} \mathsf{Z}_l = \mathcal{I}nf(\mathsf{R}_1|\pi_1)$.

In the former case, since $\overrightarrow{\mathsf{R}}_1 \preceq \overrightarrow{\mathsf{R}}_2$, there exists an infinite path $\vec{\pi}_2$ in $\overrightarrow{\mathsf{R}}_2$ such that $\mathcal{I}nf(\overrightarrow{\mathsf{R}}_2|\vec{\pi}_2) = \mathcal{I}nf(\overrightarrow{\mathsf{R}}_1|\vec{\pi}_1)$. Now, since $\overrightarrow{\mathsf{R}}_2$ mimics R_2, by Condition C2, we know that there exists an infinite path π_2 in R_2 such that $\mathcal{I}nf(\mathsf{R}_2|\pi_2) = \mathsf{X}$ $(= \mathcal{I}nf(\mathsf{R}_1|\pi_1))$. An analogous argument holds for the latter case.

As for the last condition, let ν_1 be a leaf of R_1. Since $\overrightarrow{\mathsf{R}}_1$ mimics R_1 (see Condition C6), there exists an infinite path $\vec{\pi}_1$ in $\overrightarrow{\mathsf{R}}_1$ such that $\mathcal{I}nf(\overrightarrow{\mathsf{R}}_1|\vec{\pi}_1)$ is a singleton of the form $\{(\mathsf{b}, \mathsf{s}, \mathsf{Y})\}$, with $\mathsf{b} = \mathsf{T}(\downarrow_1\mathsf{R}_1(\nu_1))$, $\mathsf{s} = \downarrow_2\mathsf{R}_1(\nu_1)$, and $\mathsf{Y} = \mathcal{I}mg(\mathsf{R}_1|\nu_1)$. Again, since $\overrightarrow{\mathsf{R}}_1 \preceq \overrightarrow{\mathsf{R}}_2$, we know that there exists an infinite path $\vec{\pi}_2$ in $\overrightarrow{\mathsf{R}}_2$ such that $\mathcal{I}nf(\overrightarrow{\mathsf{R}}_2|\vec{\pi}_2) = \mathcal{I}nf(\overrightarrow{\mathsf{R}}_1|\vec{\pi}_1)$. Finally, since $\overrightarrow{\mathsf{R}}_2$ mimics R_2, from Condition C4 we devise the existence of a leaf ν_2 of R_1 such that (i) $\mathsf{T}(\downarrow_1 \mathsf{R}_2(\nu_2)) = \mathsf{b}$ $(= \mathsf{T}(\downarrow_1 \mathsf{R}_1(\nu_1)))$, (ii) $\downarrow_2 \mathsf{R}_2(\nu_2) = \mathsf{s}$ $(= \downarrow_2 \mathsf{R}_1(\nu_1))$, and (iii) $\mathcal{I}mg(\mathsf{R}_2|\nu_2) = \mathsf{Y}$ $(= \mathcal{I}mg(\mathsf{R}_1|\nu_1))$. This proves that $\mathsf{R}_1 \preceq \mathsf{R}_2$. □

Theorem 5. *Let* T *be a* D*-augmented tree,* Π *be a* B*-labeled factorization of* T, *with* $\mathsf{D} \subseteq \mathsf{B}$, *and* $\mathcal{A}$ *be a* B*-augmented tree automaton. We have that*

$$\mathsf{T} \in \mathscr{L}(\mathcal{A}) \qquad \textit{iff} \qquad \vec{\mathsf{T}} \in \mathscr{L}(\vec{\mathcal{A}}),$$

where $\overrightarrow{\mathsf{T}}$ *denotes the encoding of the retraction of* T *with respect to* $\mathcal{A}$ *and* Π *and* $\overrightarrow{\mathcal{A}}$ *denotes the retraction automaton of* $\mathcal{A}$*.*

Proof. Let $\mathcal{A} = (\mathsf{A}, \mathsf{C}, \mathsf{D}, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$ and $\overrightarrow{\mathcal{A}} = (\mathsf{B}, \mathscr{T}_{\mathcal{A}} \cup \{\bot\}, \overrightarrow{\mathsf{S}}, \overrightarrow{\Delta}, \overrightarrow{\mathcal{I}}, \overrightarrow{\mathcal{F}})$. We first prove the left to right implication.

Let us consider a successful run R of $\mathcal{A}$ on T. By Lemma 16, we know that there exist a run R' of $\mathcal{A}$ on T such that $\mathsf{R}' \preceq \mathsf{R}$ and a run $\overrightarrow{\mathsf{R}}$ of $\overrightarrow{\mathcal{A}}$ on $\overrightarrow{\mathsf{T}}$ that mimics R'. We have to prove that $\overrightarrow{\mathsf{R}}$ is a successful run of $\overrightarrow{\mathcal{A}}$. First of all, since $\overrightarrow{\mathsf{R}}$ mimics R' (see Condition C1), we have that $\downarrow_2 \overrightarrow{\mathsf{R}}(\varepsilon)$ is a set of the form $(\mathsf{b}, \mathsf{s}, \{\mathsf{s}\}, \{\mathsf{s}\})$, with $\mathsf{s} = \downarrow_2 \mathsf{R}'(\varepsilon)$. Then, since $\mathsf{R}' \preceq \mathsf{R}$ and R is successful, we have $\mathsf{s} \in \mathcal{I}$, which immediately implies $\downarrow_2 \overrightarrow{\mathsf{R}}(\varepsilon) \in \overrightarrow{\mathcal{I}}$.

Now, let $\vec{\pi}$ be an infinite path in $\overrightarrow{\mathsf{R}}$. We distinguish between the following three cases:

1. $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ is a singleton of the form $\{\mathsf{X}\}$;
2. $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ is is a set of the form $\{(\mathsf{b}_1, \mathsf{s}_1, \mathsf{Y}, \mathsf{Z}_1), ..., (\mathsf{b}_k, \mathsf{s}_k, \mathsf{Y}, \mathsf{Z}_k)\}$;
3. $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi})$ is a singleton of the form $\{(\mathsf{b}, \mathsf{s}, \mathsf{Y})\}$.

In the first case, since $\overrightarrow{\mathsf{R}}$ mimics R' (see Condition C2), there exists an infinite path π' in R' such that $\mathcal{I}nf(\mathsf{R}'|\pi') = \mathsf{X}$. Moreover, since $\mathsf{R}' \preceq \mathsf{R}$, there is an infinite path π in R such that $\mathcal{I}nf(\mathsf{R}|\pi) = \mathcal{I}nf(\mathsf{R}'|\pi')$. Finally, since R is successful, we have $\mathcal{I}nf(\mathsf{R}|\pi) \in \mathcal{F}$, whence $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi}) \in \overrightarrow{\mathcal{F}}$.

In the second case, since $\overrightarrow{\mathsf{R}}$ mimics R' (see Condition C3), there exists an infinite path π' in R' such that $\mathcal{I}nf(\mathsf{R}'|\pi') = \bigcup_{1 \leqslant l \leqslant k} \mathsf{Z}_l$. Moreover, since $\mathsf{R}' \preceq \mathsf{R}$, there exists an infinite path π in R such that $\mathcal{I}nf(\mathsf{R}|\pi) = \mathcal{I}nf(\mathsf{R}'|\pi')$. Finally, since R is successful, we have $\mathcal{I}nf(\mathsf{R}|\pi) \in \mathcal{F}$, whence $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi}) \in \overrightarrow{\mathcal{F}}$.

In the third case, since $\overrightarrow{\mathsf{R}}$ mimics R' (see Condition C4), there exists a leaf ν' of R' such that $\mathsf{T}\big(\downarrow_1 \mathsf{R}'(\nu')\big) = \mathsf{b}$, $\downarrow_2 \mathsf{R}'(\nu') = \mathsf{s}$, and $\mathcal{I}mg(\mathsf{R}'|\nu') = \mathsf{Y}$. Moreover, since $\mathsf{R}' \preceq \mathsf{R}$, there exists a leaf ν of R such that $\mathsf{T}(\downarrow_1 \mathsf{R}(\nu)) = \mathsf{b}$, $\downarrow_2 \mathsf{R}(\nu) = \mathsf{s}$, and $\mathcal{I}mg(\mathsf{R}|\nu) = \mathsf{Y}$. Finally, since R is successful, we have $(\mathsf{b}, \mathsf{s}, \mathsf{Y}) \in \mathcal{G}$, whence $\mathcal{I}nf(\overrightarrow{\mathsf{R}}|\vec{\pi}) \in \overrightarrow{\mathcal{F}}$.

The above arguments show that $\overrightarrow{\mathsf{R}}$ is a successful run of $\overrightarrow{\mathcal{A}}$ on $\overrightarrow{\mathsf{T}}$.

As for the converse implication, let us consider a successful run $\overrightarrow{\mathsf{R}}$ of $\overrightarrow{\mathcal{A}}$ on $\overrightarrow{\mathsf{T}}$. By Lemma 15, there is a run R of $\mathcal{A}$ on T which is mimicked by $\overrightarrow{\mathsf{R}}$. By using arguments analogous to the previous ones, we can easily verify that (i) the state at the root of R belongs to $\mathcal{I}$, (ii) every infinite path π in R satisfies $\mathcal{I}nf(\mathsf{R}|\pi) \in \mathcal{F}$, and (iii) every leaf ν of R satisfies $(\mathsf{T}(\downarrow_1 \mathsf{R}(\nu)), \downarrow_2 \mathsf{R}(\nu), \mathcal{I}mg(\mathsf{R}|\nu)) \in \mathcal{G}$. This shows that R is a successful run of $\mathcal{A}$ on T. □

Theorem 6. *Let* T *be a* D*-augmented tree,* Π *be a* B*-labeled factorization of* T*, with* $\mathsf{D} \subseteq \mathsf{B}$*, and* $\mathcal{A}$ *be a* B*-augmented tree automaton. The minor* $\mathcal{A}$*-type of* T *can be computed from the minor* $\overrightarrow{\mathcal{A}}$*-type of the encoding* $\overrightarrow{\mathsf{T}}$ *of the retraction of* T *with respect to* $\mathcal{A}$ *and* Π*, where* $\overrightarrow{\mathcal{A}}$ *is the retraction automaton of* $\mathcal{A}$*.*

Proof. Suppose that the minor $\vec{\mathcal{A}}$-type $\vec{\sigma}$ of $\vec{T}$ is of the form

$$\left\{ \begin{pmatrix} \downarrow_2 \vec{R}(\varepsilon) \\ \{\mathcal{I}nf(\vec{R}|\vec{\pi}) \ : \ \vec{\pi} \in \mathcal{B}ch(\vec{R})\} \\ \emptyset \end{pmatrix} \ : \ \vec{R} \in \vec{\mathcal{R}} \right\}$$

where $\vec{\mathcal{R}}$ is a complete set of runs of $\vec{\mathcal{A}}$ on $\vec{T}$ (notice that any run of $\vec{\mathcal{A}}$ on $\vec{T}$ has no leaves). By Lemma 15, we know that, for every run $\vec{R} \in \vec{\mathcal{R}}$, if the state that appears at the root of $\vec{R}$ is a quadruple of the form $(b, s, \{s\}, \{s\})$, then there is a run R of $\mathcal{A}$ on T which is mimicked by $\vec{R}$. Let $\mathcal{R}$ be the set of all and only the runs of $\mathcal{A}$ which are mimicked, in virtue of Lemma 15, by some run $\vec{R} \in \vec{\mathcal{R}}$. We have to show that $\mathcal{R}$ is a *complete* set of runs. Let R_3 be a generic run of $\mathcal{A}$ on T. By Lemma 16, there exist a run R_2 of $\mathcal{A}$ on T such that $R_2 \preceq R_3$ and a run $\vec{R}_2$ of $\vec{\mathcal{A}}$ on $\vec{T}$ that mimics R_2. Now, since $\vec{\mathcal{R}}$ is a complete set of runs, there exists a run $\vec{R}_1 \in \vec{\mathcal{R}}$ such that $\vec{R}_1 \preceq \vec{R}_2$. Moreover, by construction, there exists a run $R_1 \in \mathcal{R}$ which is mimicked by $\vec{R}_1$. Summing up, we have

i) $\vec{R}_1$ mimics R_1;

ii) $\vec{R}_2$ mimics R_2;

iii) $\vec{R}_1 \preceq \vec{R}_2$.

By Lemma 17, this implies that $R_1 \preceq R_2$ and, by transitivity, $R_1 \preceq R_3$. This shows that $\mathcal{R}$ is a complete set of runs of $\mathcal{A}$ on T.

Now, we rewrite $\vec{\sigma}$ as follows:

$$\left\{ \begin{pmatrix} \vec{r}_h \\ \{\vec{F}_{h,i} \ : \ i \in \vec{I}_h\} \\ \emptyset \end{pmatrix} \ : \ h \in \vec{H} \right\}$$

where $\vec{H}$ is a suitable finite set of indices and, for all $h \in \vec{H}$, $\vec{I}_h$ is another finite set of indices, $\vec{r}_h$ is a state of $\mathcal{A}$, and $\vec{F}_{h,i}$, with $i \in \vec{I}_h$, is a set of states of $\mathcal{A}$. Let H be the subset of $\vec{H}$ that contains all and only the indices h such that $\vec{r}_h$ is a quadruple of the form $(b_h, s_h, \{s_h\}, \{s_h\})$. Moreover, for every $h \in H$, we write $\vec{I}_h$ as the union of three disjoint sets I^1_h, I^2_h, and I^3_h, defined as follows:

- I^1_h consists of all and only the indices $i \in \tilde{I}_h$ such that $\vec{F}_{h,i}$ is a singleton of the form $\{X_{h,i}\}$;
- I^2_h consists of all and only the indices $i \in \tilde{I}_h$ such that $\vec{F}_{h,i}$ is a set of the form $\{\vec{r}_{h,i,1}, ..., \vec{r}_{h,i,k_{h,i}}\}$, with each $\vec{r}_{h,i,l}$ being a state in $B \times S \times \mathscr{P}(S) \times \mathscr{P}(S)$;

- I^3_h consists of all and only the indices $i \in \widetilde{I}_h$ such that $\overrightarrow{F}_{h,i}$ is a singleton of the form $\{(b_{h,i}, s_{h,i}, Y_{h,i})\}$.

From previous results, we know that an $\mathcal{A}$-type of T can be obtained by collecting all triples of the form

$$\begin{pmatrix} s_h \\ \{X_{h,i} \,:\, i \in I^1_h\} \;\cup\; \{\bigcup_{1\leqslant l\leqslant k_{h,i}} \downarrow_4 \overrightarrow{r}_{h,i,l} \,:\, i \in I^2_h\} \\ \{(b_{h,i},\, s_{h,i},\, Y_{h,i}) \,:\, i \in I^3_h\} \end{pmatrix}$$

where h ranges over H. Furthermore, one can easily discard from such an $\mathcal{A}$-type the redundant features, if any. This proves that the minor $\mathcal{A}$-type σ of T can be effectively computed from the given (minor) $\overrightarrow{\mathcal{A}}$-type $\overrightarrow{\sigma}$ of $\overrightarrow{T}$. $\square$

A.2 Proof of Theorem 8

This section is devoted to prove Theorem 8. As explained in Section 3.4.1, we shall give separate proofs for shrinking second-order tree substitutions and non-erasing second-order tree substitutions.

Let us first consider the case of a shrinking substitution. We assume that $A = A' = \{a_1, ..., a_k\}$ and $C = C'$ and we let $\bar{F} = (F_c)_{c\in C}$ be a tuple of A-labeled C-colored $(C\cup D)$-augmented replacing trees to be used in a shrinking second-order tree substitution, where $D \supseteq A$. Notice that, for every $(C\cup D)$-augmented tree automaton $\mathcal{A}'$ and every color $c \in C$, one of the following conditions holds:

1. $F_c = \emptyset$,
2. $F_c = a'$, for some $a' \in A$,
3. $F_c = c\langle a_1, ..., a_k\rangle$.

Moreover, one can detect whether case 1., 2., or 3. holds by looking at the minor $\mathcal{A}'$-type of F_c. Indeed, if $F_c = \emptyset$, then the minor $\mathcal{A}'$-type of F_c consists only of triples of the form $(r, \{F_i\}_{i\in I}, \emptyset)$ (recall that the minor $\mathcal{A}'$-type of the empty tree is defined as the minor $\mathcal{A}'$-type of the infinite complete $\perp$-colored tree). If $F_c = a'$, for some $a' \in A$, then the minor $\mathcal{A}'$-type of F_c consists only of triples of the form $(r, \emptyset, \{(a', r, \{r\})\})$ (notice that the third component of these triples is a singleton). Finally, if $F_c = c\langle a_1, ..., a_k\rangle$, then the minor $\mathcal{A}'$-type of F_c consists of triples of the form $(r, \emptyset, \{(a'_j, r_j, G_j)\}_{j\in J})$, where $\{a'_j\}_{j\in J} = \{a_1, ..., a_k\}$.

On the grounds of previous arguments, in the case of a shrinking substitution, Theorem 8 can be rephrased as follows.

Lemma 18. *Let $\mathcal{A}'$ be a $(C \cup D)$-augmented tree automaton and let $\bar{F} = (F_c)_{c\in C}$ be a tuple of replacing trees of a shrinking second-order tree substitution. One can compute a $(C\cup D)$-augmented tree automaton $\mathcal{A}$ and a function*

$\mathsf{f} : \mathscr{T}_{\mathcal{A}} \to \mathscr{T}_{\mathcal{A}'}$ *such that, for every tree* T*, if* σ *is the minor* $\mathcal{A}$*-type of* T*, then* $\mathsf{f}(\sigma)$ *is the minor* $\mathcal{A}'$*-type of* $\mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in C}$.

Proof. Let $\mathcal{A}' = (A, C, C \cup D, S, \Delta, \mathcal{I}, \mathcal{F}, \mathcal{G})$ be a $(C \cup D)$-augmented tree automaton, let T be a generic A-labeled C-colored $(C \cup D)$-augmented tree, and let $\mathsf{T}' = \mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in C}$. For the sake of simplicity, we assume that T and T' are non-empty full trees (if this is not the case, we simply replace both trees with their completions and we let $\mathsf{F}_\perp = \perp\langle a_1, ..., a_k\rangle$). Below, we define the $(C \cup D)$-augmented tree automaton $\mathcal{A}$, which runs on T, and mimics the computations of the automaton $\mathcal{A}'$ on T'. The states of $\mathcal{A}$ are the triples of the form $\widetilde{s} = (x, c, s)$, with $x \in \{0, 1\}$, $c \in C$, and $s \in S$. Intuitively, when the first component x of $\widetilde{s}$ is set to 1, the automaton $\widetilde{\mathcal{A}}$ lies at an erased vertex of $\widetilde{\mathsf{T}}$ and hence its computation is somehow 'disabled'; when $x = 0$, the second component c of $\widetilde{s}$ stores the last (non-erased) color read by the automaton $\mathcal{A}'$ and the third component s of $\widetilde{s}$ stores the current state of $\mathcal{A}'$. Formally, we define $\mathcal{A} = (A, C, C \cup D, \widetilde{S}, \widetilde{\Delta}, \widetilde{\mathcal{I}}, \widetilde{\mathcal{F}})$, where

- $\widetilde{S} = \{0, 1\} \times C \times S$;
- for every state $\widetilde{s} = (x, c, s) \in \widetilde{S}$, with $x = 1$, and every color $c' \in C$,
$$\widetilde{\Delta}(\widetilde{s}, c') = \bigwedge_{a \in A} \langle a, \widetilde{s}\rangle;$$
- for every state $\widetilde{s} = (x, c, s) \in \widetilde{S}$, with $x = 0$, and every color $c' \in C$, with $\mathsf{F}_{c'} = \emptyset$,
$$\widetilde{\Delta}(\widetilde{s}, c') = \bigwedge_{a \in A} \langle a, \widetilde{s}'\rangle,$$
where $\widetilde{s}' = (1, c', s)$;
- for every state $\widetilde{s} = (x, c, s) \in \widetilde{S}$, with $x = 0$, and every color $c' \in C$, with $\mathsf{F}_{c'} = a'$ for some $a' \in A$,
$$\widetilde{\Delta}(\widetilde{s}, c') = \langle a', \widetilde{s}'\rangle,$$
where $\widetilde{s}' = (0, c', s)$;
- for every state $\widetilde{s} = (x, c, s) \in \widetilde{S}$, with $x = 0$, and every color $c' \in C$, with $\mathsf{F}_{c'} = c'\langle a_1, ..., a_k\rangle$,
$$\widetilde{\Delta}(\widetilde{s}, c') = \Phi_{s,c'},$$
where $\Phi_{s,c'}$ is the formula obtained from $\Delta(s, c')$ by replacing every atom of the form $\langle a', s'\rangle$ with $\langle a', (0, c', s')\rangle$;
- the sets $\widetilde{\mathcal{I}}$ and $\widetilde{\mathcal{F}}$ are not relevant here.

It is easy to verify that, for every run R' of $\mathcal{A}'$ on T', there is a corresponding run R of $\mathcal{A}$ on T, with $\downarrow_2 \mathsf{R}(\varepsilon)$ being of a triple of the form $(0, c, s)$ (and, vice versa, for every run R of $\mathcal{A}$ on T, with $\downarrow_2 \mathsf{R}(\varepsilon)$ being a triple of the form

$(0, c, s)$, there is a corresponding run R' of $\mathcal{A}'$ on T') that satisfies the following conditions:

i) $\downarrow_2 R(\varepsilon)$ is of the form $(0, c, s)$, with $s = \downarrow_2 R'(\varepsilon)$;

ii) for every infinite path π in R such that $\mathcal{I}nf(R|\pi)$ is a set of the form $\{(0, c_1, s_1), ..., (0, c_n, s_n)\}$, there exists an infinite path π' in R' such that $\mathcal{I}nf(R'|\pi') = \{s_1, ..., s_n\}$;

iii) for every leaf ν of R such that $T(\downarrow_1 R(\nu)) = d$, with $d \in C^+ \cup D$ (namely, d is a non-erased color), $\downarrow_2 R(\nu)$ is a triple of the form $(0, c, s)$, and $\mathcal{I}mg(R|\nu)$ is a set of the form $\{(0, c_1, s_1), ..., (0, c_n, s_n)\}$, there exists a leaf ν' of R' such that $T'(\downarrow_1 R'|\nu') = d$, $\downarrow_2 R'(\nu') = s$, and $\mathcal{I}mg(R'|\pi_{\nu'}) = \{s_1, ..., s_n\}$;

iv) for every leaf ν of R such that $T(\downarrow_1 R(\nu)) = d$, with $d \in C^-$ (namely, d is an erased color), $\downarrow_2 R(\nu)$ is a triple of the form $(0, c, s)$, and $\mathcal{I}mg(R|\nu)$ is a set of the form $\{(0, c_1, s_1), ..., (0, c_n, s_n)\}$, there exists a leaf ν' of R' such that $T'(\downarrow_1 R'(\nu')) = c$, $\downarrow_2 R'(\nu') = s$, and $\mathcal{I}mg(R'|\nu') = \{s_1, ..., s_n\}$;

v) for every infinite path π' in R', there exists an infinite path π in R such that $\mathcal{I}nf(R|\pi)$ is a set of the form $\{(0, c_1, s_1), ..., (0, c_n, s_n)\}$, with $\{s_1, ..., s_n\} = \mathcal{I}nf(R'|\pi')$;

vi) for every leaf ν' of R', there exists a leaf ν of R such that $\downarrow_2 R(\nu)$ is a state of the form $(0, c, s)$, with $s = \downarrow_2 R'(\nu')$, $\mathcal{I}mg(R|\nu)$ is a set of the form $\{(0, c_1, s_1), ..., (0, c_n, s_n)\}$, with $\{s_1, ..., s_n\} = \mathcal{I}mg(R'|\pi_{\nu'})$, and $T(\downarrow_1 R(\nu))$ is a color d either in the set $C^+ \cup D$, in which case $d = T'(\downarrow_1 R'(\nu'))$, or in the set C^-, in which case $c = T'(\downarrow_1 R'(\nu'))$.

On the grounds of the above arguments, one can compute the minor $\mathcal{A}'$-type of the tree T' from the given minor $\mathcal{A}$-type of the tree T. More precisely, let the minor $\mathcal{A}$-type of T be a set of the form

$$\left\{ \begin{pmatrix} \widetilde{s}_h \\ \{\widetilde{F}_{h,i} \ : \ i \in \widetilde{I}_h\} \\ \{(d_{h,j}, \ \widetilde{s}_{h,j}, \ \widetilde{G}_{h,j}) \ : \ j \in \widetilde{J}_h\} \end{pmatrix} \ : \ h \in \widetilde{H} \right\}$$

We denote by H the subset of $\widetilde{H}$ that contains all and only the indices h such that $\widetilde{s}_h$ is a triple of the form $(0, c_h, s_h)$. Then, for each $h \in H$, we let I_h be the subset of $\widetilde{I}_h$ that consists of all and only the indices $i \in \widetilde{I}_h$ such that the first component of every triple in $\widetilde{F}_{h,i}$ is 0 and we view $\widetilde{J}_h$ as the union of the two disjoint sets J^1_h and J^2_h satisfying the following conditions:

- J^1_h consists of all and only the indices $j \in \widetilde{J}_h$ such that the second component of $\widetilde{s}_{h,j}$ is a color belonging to the set $C^+ \cup D$ (i.e., a non-erased color);
- J^2_h consists of all and only the indices $j \in \widetilde{J}_h$ such that the second component of $\widetilde{s}_{h,j}$ is a color belonging to the set C^- (i.e., an erased color).

It is clear that the set σ' of all triples of the form

$$\left(\begin{array}{c} \downarrow_3\widetilde{s}_h \\ \left\{\downarrow_3\widetilde{F}_{h,i} \,:\, i \in I_h\right\} \\ \left\{(d_{h,j},\, \downarrow_3\widetilde{s}_{h,j},\, \downarrow_3\widetilde{G}_{h,j}) \,:\, j \in J^1_h\right\} \cup \left\{(\downarrow_2\widetilde{s}_{h,j},\, \downarrow_3\widetilde{s}_{h,j},\, \downarrow_3\widetilde{G}_{h,j}) \,:\, j \in J^2_h\right\} \end{array}\right)$$

for h ranging over H, is the minor $\mathcal{A}'$-type of the tree T'. □

We now consider the case of a non-erasing second-order tree substitution. In such a case, Theorem 8 is rephrased as follows.

Lemma 19. *Let $\mathcal{A}'$ be a $(C' \cup D)$-augmented tree automaton and let $\bar{\sigma} = (\sigma_c)_{c\in C}$ be a tuple of minor $\mathcal{A}'$-types. One can compute a tree automaton $\widetilde{\mathcal{A}}$ such that, for every tree T and every tuple $\bar{F} = (F_c)_{c\in C}$ of non-erasing replacing trees, the minor $\mathcal{A}'$-type σ' of $T' = T[\![F_c/c]\!]_{c\in C}$ is uniquely determined by (and computable from) the minor $\widetilde{\mathcal{A}}$-type $\widetilde{\sigma}$ of the infinite complete $(C \cup D \cup \{\perp\})$-colored tree $\widetilde{T}$, which obtained from T by adding $\perp$-colored vertices, and the minor $\mathcal{A}'$-type $\bar{\sigma}$ of $\bar{F}$.*

Proof. Let T be generic A-labeled C-colored $(C \cup D)$-augmented tree, let $\bar{F} = (F_c)_{c\in C}$ be a generic tuple of A'-labeled C'-colored $(C' \cup D)$-augmented non-erasing replacing trees, and let $T' = T[\![F_c/c]\!]_{c\in C}$. Furthermore, let us denote by $\widetilde{T}$ the infinite complete A-labeled $(C \cup D \cup \{\perp\})$-colored tree obtained from T by adding $\perp$-colored vertices. For the sake of brevity, we expand the tuple $\bar{F}$ with one (singleton) replacing tree $F_d = d$, for each $d \in D$, and we assume that T' is a non-empty full tree. For every vertex v of T, we denote by V_v the set of vertices of T' that correspond to v. Formally, the set V_v is recursively defined as follows:

$$V_v = \begin{cases} \{\varepsilon\} & \text{if } v = \varepsilon, \\ V_u \cdot \{w \in \mathcal{F}r(F_c) \,:\, F_c(w) = a\} & \text{if } v = u \cdot a \text{ and } T(u) = c. \end{cases}$$

Then, we define the *induced factorization* Π' of T' as follows:

- the domain of Π' is the union, over all vertices v of T, of the sets V_v;
- (u', v') is an a-labeled edge of Π', with $a \in A$, iff T contains an a-labeled edge (u, v) such that $u' \in V_u$ and $v' \in V_v$.

Since, the replacing tree F_c, for every $c \in C$, is neither empty nor singleton, we have that the sets V_v, where v ranges over $\mathcal{D}om(T)$, are pairwise disjoint. Therefore, for every vertex v' of Π', we can denote by v the (unique) vertex of T that corresponds to v, namely, such that $v' \in V_v$. It turns out that, for any vertex v' of Π', if v is the vertex of T that corresponds to v' and $T(v) = c$, then the marked factor of T' rooted at v' is isomorphic to the tree F_c (note that $c \in C \cup D$).

We can now prove the lemma by directly exploiting Theorem 6. Let $\mathcal{A}'$ be a given $(\mathsf{C}' \cup \mathsf{D})$-augmented tree automaton and let $\bar{\sigma} = (\sigma_c)_{c \in C \cup D}$ be the minor $\mathcal{A}'$-type of the tuple $\bar{\mathsf{F}}$ of non-erasing replacing trees. We need to define a suitable tree automaton $\widetilde{\mathcal{A}}$, which only depends on $\mathcal{A}'$ and $\bar{\sigma}$, that mimics $\mathcal{A}'$. Notice that the encoding $\overrightarrow{\mathsf{T}}'$ of the retraction of T' with respect to $\mathcal{A}'$ and Π' can be obtained from the infinite complete A-labeled $(\mathsf{C} \cup \mathsf{D} \cup \{\perp\})$-colored tree $\widetilde{\mathsf{T}}$ by replacing each color $\mathsf{c} \in \mathsf{C} \cup \mathsf{D}$ with the corresponding minor $\mathcal{A}'$-type σ_c. Thus, we can denote by $\overrightarrow{\mathcal{A}} = (\mathsf{C} \cup \mathsf{D}, \mathscr{T}_{\mathcal{A}'} \cup \{\perp\}, \overrightarrow{\mathsf{S}}, \overrightarrow{\Delta}, \overrightarrow{\mathfrak{I}}, \overrightarrow{\mathfrak{F}})$ the retraction automaton of $\mathcal{A}'$ and then define $\widetilde{\mathcal{A}} = (\mathsf{A}, \mathsf{C} \cup \mathsf{D} \cup \{\perp\}, \widetilde{\mathsf{S}}, \widetilde{\Delta}, \widetilde{\mathfrak{I}}, \widetilde{\mathfrak{F}})$, where

- $\widetilde{\mathsf{S}} = \overrightarrow{\mathsf{S}}$, namely, the states of $\widetilde{\mathcal{A}}$ are exactly those of $\overrightarrow{\mathcal{A}}$;
- for every state $\widetilde{s} \in \widetilde{\mathsf{S}}$ and every input symbol $\mathsf{c} \in \mathsf{C} \cup \mathsf{D} \cup \{\perp\}$, we set

$$\widetilde{\Delta}(\widetilde{s}, c) = \begin{cases} \overrightarrow{\Delta}(\widetilde{s}, \sigma_c) & \text{if } c \in \mathsf{C} \cup \mathsf{D}, \\ \overrightarrow{\Delta}(\widetilde{s}, \perp) & \text{if } c = \perp; \end{cases}$$

- the sets $\widetilde{\mathfrak{I}}$ and $\widetilde{\mathfrak{F}}$ are not relevant here.

Clearly, every run $\widetilde{\mathsf{R}}$ of $\widetilde{\mathcal{A}}$ on $\widetilde{\mathsf{T}}$ is also a run of $\overrightarrow{\mathcal{A}}$ on $\overrightarrow{\mathsf{T}}'$, and vice versa. This basically means that the minor $\widetilde{\mathcal{A}}$-type of $\widetilde{\mathsf{T}}$ coincides with the minor $\overrightarrow{\mathcal{A}}$-type of $\overrightarrow{\mathsf{T}}'$. Therefore, by Theorem 6, one can compute the minor $\mathcal{A}'$-type σ' of the tree T' from the minor $\widetilde{\mathcal{A}}$-type $\widetilde{\sigma}$ of $\widetilde{\mathsf{T}}$ and the minor $\mathcal{A}'$-type $\bar{\sigma}$ of $\bar{\mathsf{F}}$. □

Finally, we can prove Theorem 8 (for the sake of clearness, we renamed some of the objects appearing in the statement of the theorem).

Theorem 8. *Let $\mathcal{A}''$ be a $(\mathsf{C}' \cup \mathsf{D})$-augmented tree automaton and let $\bar{\sigma} = (\sigma_c)_{c \in C}$ be a tuple of minor $\mathcal{A}''$-types. One can compute a $(\mathsf{C} \cup \mathsf{D})$-augmented tree automaton $\mathcal{A}$ such that, for every tree T and every tuple $\bar{\mathsf{F}} = (\mathsf{F}_c)_{c \in C}$ of replacing trees, the minor $\mathcal{A}''$-type σ'' of $\mathsf{T}'' = \mathsf{T}[\![\mathsf{F}_c/c]\!]_{c \in C}$ is uniquely determined by (and computable from) the minor $\mathcal{A}$-type σ of T and the minor $\mathcal{A}''$-type $\bar{\sigma}$ of $\bar{\mathsf{F}}$.*

Proof. Let $\mathcal{A}''$ be a $(\mathsf{C}' \cup \mathsf{D})$-augmented tree automaton, let T be an A-labeled C-colored $(\mathsf{C} \cup \mathsf{D})$-augmented tree, and let $\bar{\mathsf{F}} = (\mathsf{F}_c)_{c \in C}$ be a tuple of A'-labeled C'-colored $(\mathsf{C}' \cup \mathsf{D})$-augmented replacing trees. Recall that C^- (resp., C^+) denotes the set of all erased (resp., non-erased) colors of C. We define

$$\mathsf{F}^-_c = \begin{cases} \mathsf{F}_c & \text{if } c \in \mathsf{C}^- \\ c\langle a_1, ..., a_k \rangle & \text{if } c \in \mathsf{C}^+ \end{cases} \qquad \mathsf{F}^+_c = \begin{cases} \mathsf{F}_c & \text{if } c \in \mathsf{C}^+ \\ c\langle a_1, ..., a_k \rangle & \text{if } c \in \mathsf{C}^- \end{cases}$$

$$\mathsf{T}' = \mathsf{T}[\![\mathsf{F}^-_c/c]\!]_{c \in C} \qquad \mathsf{T}'' = \mathsf{T}'[\![\mathsf{F}^+_c/c]\!]_{c \in C}.$$

Clearly, T' is an A-labeled C^+-colored $(\mathsf{C}^+ \cup \mathsf{D})$-augmented tree resulting from a shrinking second-order tree substitution. Similarly, T'' is an A'-labeled C'-colored $(\mathsf{C}' \cup \mathsf{D})$-augmented tree resulting from a non-erasing second-order tree substitution.

Now, let $\widetilde{\mathsf{T}}'$ be the infinite complete A-labeled $(\mathsf{C}^+ \cup \mathsf{D} \cup \{\perp\})$-colored tree obtained from T' by adding $\perp$-colored vertices. We prove the theorem in three steps. First, given the minor $\mathcal{A}''$-type $\bar{\sigma}$ of the tuple $\bar{\mathsf{F}} = (\mathsf{F}^+_\mathsf{c})_{\mathsf{c}\in\mathsf{C}}$, we exploit Lemma 19 to compute a tree automaton $\widetilde{\mathcal{A}}'$ in such a way that the minor $\mathcal{A}''$-type σ'' of T'' is uniquely determined by (and computable from) the minor $\widetilde{\mathcal{A}}'$-type $\widetilde{\sigma}'$ of $\widetilde{\mathsf{T}}'$ and the minor $\mathcal{A}''$-type $\bar{\sigma}$ of $\bar{\mathsf{F}}$. Then, we define a $(\mathsf{C}^+ \cup \mathsf{D})$-augmented tree automaton $\mathcal{A}'$ from $\widetilde{\mathcal{A}}$ and we show how to compute the minor $\widetilde{\mathcal{A}}'$-type $\widetilde{\sigma}'$ of $\widetilde{\mathsf{T}}'$ on the grounds of the minor $\mathcal{A}'$-type σ' of T'. Finally, we exploit Lemma 18, to compute a $(\mathsf{C} \cup \mathsf{D})$-augmented tree automaton $\mathcal{A}$ in such a way that the minor $\mathcal{A}'$-type σ' of T' is uniquely determined by (and computable from) the minor $\mathcal{A}$-type σ of T.

Let us consider the non-erasing substitution $\mathsf{T}'' = \mathsf{T}'[\![\mathsf{F}^+_\mathsf{c}/\mathsf{c}]\!]_{\mathsf{c}\in\mathsf{C}}$. Assume that $\bar{\sigma}$ is the minor $\mathcal{A}''$-type of $\bar{\mathsf{F}} = (\mathsf{F}^+_\mathsf{c})_{\mathsf{c}\in\mathsf{C}}$. By Lemma 19, given a $(\mathsf{C}' \cup \mathsf{D})$-augmented tree automaton $\mathcal{A}''$, we can obtain a tree automaton $\widetilde{\mathcal{A}}'$ in such a way that the minor $\mathcal{A}''$-type of T'' is uniquely determined by (and computable from) the minor $\widetilde{\mathcal{A}}'$-type $\widetilde{\sigma}'$ of $\widetilde{\mathsf{T}}'$ and the minor $\mathcal{A}''$-type $\bar{\sigma}$ of $\bar{\mathsf{F}}$.

Now, let $\mathcal{A}'$ be the $(\mathsf{C}^+ \cup \mathsf{D})$-augmented tree automaton obtained by expanding the above defined tree automaton $\widetilde{\mathcal{A}}'$ with the set of markers $\mathsf{C}^+ \cup \mathsf{D}$ (the acceptance conditions at the leaves are not relevant here). We show how to compute the minor $\widetilde{\mathcal{A}}'$-type of $\widetilde{\mathsf{T}}'$ on the grounds of the minor $\mathcal{A}'$-type of T. For every $\mathsf{d} \in \mathsf{C}^+ \cup \mathsf{D}$, we denote by T_d the infinite complete A-labeled $(\mathsf{C}^+ \cup \{\perp\})$-colored tree defined by

$$\mathsf{T}_\mathsf{d}(v) = \begin{cases} \mathsf{d} & \text{if } v = \varepsilon, \\ \perp & \text{otherwise.} \end{cases}$$

We further denote by σ_d the minor $\widetilde{\mathcal{A}}'$-type of T_d (note that σ_d is computable since T_d is a regular tree). We know that every infinite path in $\widetilde{\mathsf{T}}'$ is either entirely contained in T' or it traverses an $(\mathsf{C}^+ \cup \mathsf{D})$-colored leaf of T'. From this it is easy to see that, if the minor $\mathcal{A}'$-type σ' of T' is a set of the form

$$\left\{ \begin{pmatrix} r_h \\ \{\mathsf{F}_{h,i} \,:\, i \in I_h\} \\ \{(\mathsf{d}_{h,j},\, r_{h,j},\, \mathsf{G}_{h,j}) \,:\, j \in J_h\} \end{pmatrix} : h \in H \right\}$$

and, for all $\mathsf{d} \in \mathsf{C}^+ \cup \mathsf{D}$, the minor $\mathcal{A}'$-type σ_d of T_d is a set of the form

$$\left\{ \begin{pmatrix} r'_{\mathsf{d},k} \\ \{\mathsf{F}'_{\mathsf{d},k,l} \,:\, l \in L_{\mathsf{d},k}\} \\ \emptyset \end{pmatrix} : k \in K_\mathsf{d} \right\}$$

then the set $\widetilde{\sigma}'$ of all triples of the form

$$\left(\begin{array}{c} r_h \\ \Big\{ F_{h,i} \,:\, i \in I_h \Big\} \;\cup\; \left\{ F'_{d,k,l} \,:\, \begin{array}{l} j \in J_h,\ k \in K_b,\ l \in L_{d,k}, \\ d = d_{h,j},\ r'_{d,k} = r_{h,j} \end{array} \right\} \\ \emptyset \end{array} \right)$$

for h ranging over H, turns out to be the minor $\widetilde{\mathcal{A}}'$-type of $\widetilde{T}'$.

It remains to show how to compute the minor $\mathcal{A}'$-type σ' of T' from the minor $\mathcal{A}$-type σ of T. Let us consider the shrinking substitution $T' = T[\![F^-_c/c]\!]_{c\in C}$. By Lemma 18, one can compute a $(C \cup D)$-augmented tree automaton $\mathcal{A}$ in such a way that the minor $\mathcal{A}'$-type σ' of T' is uniquely determined by (and computable from) the minor $\mathcal{A}$-type σ of T and (the minor $\mathcal{A}'$-type of) the tuple $(F^-_c)_{c\in C}$. □

A.3 Proof of Proposition 34

Here, we briefly recall the statement of Proposition 34 and we give a detailed proof of it.

Proposition 34. *Let* f *and* g *be two profinitely ultimately periodic functions. The following functions are profinitely ultimately periodic as well:*

1. **(Sum)** $h = f + g$, *defined by* $h(n) = f(n) + g(n)$*;*
2. **(Product)** $h = f \cdot g$, *defined by* $h(n) = f(n) \cdot g(n)$*;*
3. **(Difference)** $h = f - g$, *defined by* $h(n) = f(n) - g(n)$, *provided that* h *has unbounded infimum;*
4. **(Quotient)** $h = \lfloor \frac{f}{d} \rfloor$, *defined by* $h(n) = \lfloor \frac{f(n)}{d} \rfloor$, *where* d *is any positive constant;*
5. **(Exponentiation)** $h = f^g$, *defined by* $h(n) = \big(f(n)\big)^{g(n)}$, *provided that* h *has unbounded infimum;*
6. **(Exponential tower)** h *defined by* $h(0) = 1$ *and* $h(n+1) = b^{h(n)}$, *where* b *is any positive constant;*
7. **(Fibonacci numbers)** h *defined by* $h(0) = h(1) = 1$ *and* $h(n+2) = h(n) + h(n+1)$*;*
8. **(Generalized sum)** h *defined by* $h(n) = \sum_{i=0}^{n-1} f(i)$*;*
9. **(Generalized product)** h *defined by* $h(n) = \prod_{i=0}^{n-1} f(i)$*;*
10. **(Composition)** $h = f \circ g$, *defined by* $h(n) = g(f(n))$.

Proof. As for the functions defined in *1.* and *2.*, it suffices to note that the operator $[\ ]_{l,r}$ respects sums and products, namely, $\big[i+j\big]_{l,r} = \big[[i]_{l,r}+[j]_{l,r}\big]_{l,r}$ and $\big[i \cdot j\big]_{l,r} = \big[[i]_{l,r} \cdot [j]_{l,r}\big]_{l,r}$ for every $i, j \in \mathbb{N}$.

Similarly, one can prove that $[\ ]_{l,r}$ respects differences, namely, $\big[i-j\big]_{l,r} = \big[[i]_{l,r} - [j]_{l,r}\big]_{l,r}$, under the proviso that $i - j \geqslant l$. It thus follows that the difference function $h = f - g$ defined in *3.* is profinitely ultimately periodic whenever it has unbounded infimum.

As for the function $h = \big\lfloor \frac{f}{d} \big\rfloor$ defined in *4.*, note that for every $l \geqslant 0$ and every $r > 0$, $\big[h(i)\big]_{l,r}$ is either 0 or $\big[[h(i-d)]_{l,r}+1\big]_{l,r}$, depending on whether $i < d$ or $i \geqslant d$. Thus, by letting $\big(h_1(i), ..., h_d(i)\big) = \big([h(i)]_{l,r}, ..., [h(i+d-1)]_{l,r}\big)$, we obtain

$$\big(h_1(i+1), ..., h_d(i+1)\big) \;=\; \begin{cases} (0, ..., 0) & \text{if } i = 0, \\ \big(h_2(i), ..., h_d(i), [h_1(i)+1]_{l,r}\big) & \text{if } i > 0. \end{cases}$$

Since each value $h_j(i)$ ranges over the finite domain $\{0, ..., l+r-1\}$, one can exploit the Pigeonhole Principle to compute two integers $p \geqslant 0$ and $q > 0$ such that $h_j(i) = h_j(i+q)$ for every $i \geqslant p$ and every $0 \leqslant j < d$. This proves that $\big[h(i)\big]_{l,r} = \big[h([i]_{p,q})\big]_{l,r}$ and hence h is a profinitely ultimately periodic function.

As for the function $h = f^g$ defined in *5.*, we preliminarily recall the definition of the 'Euler totient function' $\phi : \mathbb{N}_{>0} \;\to\; \mathbb{N}_{>0}$

$$\phi(i) \;=\; i \prod_{p \text{ prime dividing } i} \left(1 - \frac{1}{p}\right)$$

and the following two properties (d is a positive constant):

$$\begin{aligned} e^n &= e^n + \textstyle\sum_{i=1}^{n} \binom{n}{i} e^{n-i} \cdot 0 \\ &= e^n + \textstyle\sum_{i=1}^{n} \binom{n}{i} e^{n-i} \cdot m^i \\ &= (e+m)^n \qquad\qquad (\text{mod } m), \end{aligned}$$

$$\begin{aligned} e^n &= e^n \cdot 1 \\ &= e^n \cdot e^{\phi(m)} \\ &= e^{n+\phi(m)} \qquad\qquad (\text{mod } m). \end{aligned}$$

Since f and g are profinitely ultimately periodic functions, one can compute $p, p' \geqslant 0$ and $q, q' \geqslant 0$ such that $\big[f(i)\big]_{0,m} = \big[f([i]_{p,q})\big]_{0,m}$ and $\big[g(i)\big]_{0,\phi(m)} = \big[g([i]_{p',q'})\big]_{0,\phi(m)}$. Now, by letting $r = \max(p, p')$ and $s = \text{lcm}(q, q')$, we obtain

$$\begin{aligned} \big[f(i)^{g(i)}\big]_{0,m} &= \Big[\big([f(i)]_{0,m}\big)^{[g(i)]_{0,\phi(m)}}\Big]_{0,m} \\ &= \Big[\big([f([i]_{r,s})]_{0,m}\big)^{[g([i]_{r,s})]_{0,\phi(m)}}\Big]_{0,m} \\ &= \Big[f\big([i]_{r,s}\big)^{g([i]_{r,s})}\Big]_{0,m}. \end{aligned}$$

We can further generalize the above result for any $l \geqslant 0$. Let $\gamma(l)$ be the least integer i such that $i \geqslant r$ and $f(i)^{g(i)} \geqslant l$ (such a value exists and it is computable by hypothesis). Then, for every $i \geqslant \gamma(l)$, we have

$$\begin{aligned}\left[f(i)^{g(i)}\right]_{l,m} &= \left[f(i)^{g(i)} - l\right]_{0,m} + l \\ &= \left[\left[f(i)^{g(i)}\right]_{0,m} - l\right]_{0,m} + l \\ &= \left[\left[f\left([i]_{\gamma(l),s}\right)^{g([i]_{\gamma(l),s})}\right]_{0,m} - l\right]_{0,m} + l \\ &= \left[f\left([i]_{\gamma(l),s}\right)^{g([i]_{\gamma(l),s})}\right]_{l,m}.\end{aligned}$$

This proves that h $(= f^g)$ is a profinitely ultimately periodic function.

We now consider the exponential tower h defined in *6.*. We have $h(0) = 1$ and $h(i) = b^{h(i-1)}$ for every $i > 0$. Let $r > 0$ and, for every $l \geqslant 0$, let $\gamma(l) = \lceil \log_b(l) \rceil$. We prove, by induction on j, that, for every $0 \leqslant j \leqslant r$, every $l \geqslant 0$, and every $i \geqslant \gamma(l)$, the following equation holds

$$\left[h(i+j)\right]_{l,\phi^{r-j}(r)} = \left[h(i+j+1)\right]_{l,\phi^{r-j}(r)}.$$

The case $j = 0$ is almost trivial. Since $\phi(i)$ is strictly decreasing for $i > 1$, we have $\phi^r(r) = 1$. This implies that $[h(i)]_{l,\phi^r(r)} = l = [h(i+1)]_{l,\phi^r(r)}$ holds for every $i \geqslant \gamma(l)$. Now, let $j > 0$. For every $l \geqslant 0$ and every $i \geqslant \gamma(l)$, we have

$$\begin{aligned}\left[h(i+j)\right]_{l,\phi^{r-j}(r)} &= \left[b^{h(i+(j-1))}\right]_{l,\phi^{r-j}(r)} \\ &= \left[b^{[h(i+(j-1))]_{\gamma(l),\phi^{r-(j-1)}(r)}}\right]_{l,\phi^{r-j}(r)} \\ &= \left[b^{[h(i+(j-1)+1)]_{\gamma(l),\phi^{r-(j-1)}(r)}}\right]_{l,\phi^{r-j}(r)} \\ &= \left[b^{h(i+(j-1)+1)}\right]_{l,\phi^{r-j}(r)} \\ &= \left[h(i+j+1)\right]_{l,\phi^{r-j}(r)}.\end{aligned}$$

In particular, by letting $j = r$, we obtain that, for every $l \geqslant 0$ and every $i \geqslant \gamma(l)$,

$$\left[h(i+r)\right]_{l,r} = \left[h(i+r+1)\right]_{l,r}.$$

Therefore, we can conclude that, for every $l \geqslant 0$, every $r > 0$, and every $i \geqslant 0$, $\left[h(i)\right]_{l,r} = \left[h([i]_{\gamma(l)+r,1}\right]_{l,r}$ and hence h is a profinitely ultimately periodic function.

If h is the Fibonacci function, as defined in *7.*, then we have $\left[h(i)\right]_{l,r} = \left[[h(i-2)]_{l,r} + [h(i-1)]_{l,r}\right]_{l,r}$ whenever $i \geqslant 2$. Thus, by letting $\left(h_1(i), h_2(i)\right) = \left([h(i)]_{l,r}, [h(i+1)]_{l,r}\right)$, we obtain

$$\left(h_1(i+1), h_2(i+1)\right) = \begin{cases} \left([1]_{l,r}, [1]_{l,r}\right) & \text{if } i = 0, \\ \left(h_2(i), [h_1(i) + h_2(i)]_{l,r}\right) & \text{if } i > 0. \end{cases}$$

Since the values $h_1(i)$ and $h_2(i)$ range over the finite domain $\{0, ..., l+r-1\}$, one can exploit the Pigeonhole Principle to compute two integers $p \geqslant 0$ and $q > 0$ such that $\big(h_1(i), h_2(i)\big) = \big(h_1(i+q), h_2(i+q)\big)$ for every $i \geqslant p$. This shows that $\big[h(i)\big]_{l,r} = \big[h([i]_{p,q})\big]_{l,r}$ and hence h is a profinitely ultimately periodic function.

We now consider the function h defined in *8.*. Let $l \geqslant 0$ and $r > 0$. Since f is a profinitely ultimately periodic function, one can compute two integers p, q such that $\big[f(i)\big]_{l,r} = \big[f([i]_{p,q})\big]_{l,r}$. We now define $S = \sum_{j=p+1}^{p+q} f(j)$ and we observe that for every $n \geqslant 0$,

$$\left[\sum_{j=p+1}^{p+nq} f(j)\right]_{l,r} = \big[n \cdot S\big]_{l,r}.$$

By exploiting the Pigeonhole Principle, one can compute two integers $p' \geqslant 0$ and $q' > 0$ such that $\big[p' \cdot n \cdot S\big]_{l,r} = \big[(p'+q') \cdot n \cdot S\big]_{l,r}$. Moreover, for any given index $i \geqslant p + (q \cdot q')$, one can compute an integer n_i such that $i = \big[i\big]_{p,q\cdot q'} + n_i \cdot q \cdot q'$. Therefore, for every $i \geqslant p + (q \cdot q')$, we have

$$\begin{aligned}
\big[h(i)\big]_{l,r} &= \Big[\textstyle\sum_{j=0}^{i} f(j)\Big]_{l,r} \\
&= \Big[\textstyle\sum_{j=0}^{[i]_{p,q\cdot q'}} f(j) + \sum_{j=[n]_{p,q\cdot q'}+1}^{i} f(j)\Big]_{l,r} \\
&= \Big[\textstyle\sum_{j=0}^{[i]_{p,q\cdot q'}} f(j) + q' \cdot n_i \cdot S\Big]_{l,r} \\
&= \Big[\textstyle\sum_{j=0}^{[i]_{p,q\cdot q'}} f(j) + q' \cdot [n_i]_{p',q'} \cdot S\Big]_{l,r} \\
&= \Big[h([i]_{p,q\cdot q'})\Big]_{l,r}.
\end{aligned}$$

This shows that h is a profinitely ultimately periodic function.

By analogous arguments, one can prove that the function h defined in *9.* is profinitely ultimately periodic.

Finally, the function h defined in *10.* is easily proved to be profinitely ultimately periodic by noticing that, given $l \geqslant 0$ and $r > 0$, one can compute $p, p' \geqslant 0$ and $q, q' > 0$ satisfying

$$\big[g(f(i))\big]_{l,r} = \big[g(\big[f(i)\big]_{p,q})\big]_{l,r} = \big[g([f([i]_{p',q'})]_{p,q})\big]_{l,r} = \big[g(f([i]_{p',q'}))\big]_{l,r}. \square$$

References

1. Aehlig, K., de Miranda, J.G., Ong, C.-H.L.: The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 39–54. Springer, Heidelberg (2005)
2. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proceedings of the International Congress for Logic, Methodology and Philosophy of Science, pp. 1–11. Stanford University Press (1962)
3. Bettini, C., Jajodia, S., Wang, X.S.: Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer, Heidelberg (2000)
4. Booth, K.S.: Lexicographically least circular substrings. Information Processing Letters 10(4-5), 240–242 (1980)
5. Bresolin, D., Montanari, A., Puppis, G.: Time granularities and ultimately periodic automata. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 513–525. Springer, Heidelberg (2004)
6. Brzozowski, J.A.: Canonical regular expressions and minimal state graphs for definite events. Mathematical Theory of Automata 12, 529–561 (1962)
7. Cachat, T.: Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 556–569. Springer, Heidelberg (2003)
8. Calbrix, H., Nivat, M., Podelski, A.: Ultimately periodic words of rational ω-languages. In: Main, M.G., Melton, A.C., Mislove, M.W., Schmidt, D., Brookes, S.D. (eds.) MFPS 1993. LNCS, vol. 802, pp. 554–566. Springer, Heidelberg (1994)
9. Campeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
10. Carayol, A., Hague, M., Meyer, A., Ong, C.-H.L., Serre, O.: Winning regions of higher-order pushdown games. In: Proceedings of the 23rd IEEE Symposium on Logic in Computer Science (LICS), pp. 193–204. IEEE Computer Society, Los Alamitos (2008)
11. Carayol, A., Wöhrle, S.: The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 112–123. Springer, Heidelberg (2003)

12. Carton, O., Thomas, W.: The monadic theory of morphic infinite words and generalizations. Information and Computation 176(1), 51–65 (2002)
13. Caucal, D.: On infinite terms having a decidable monadic theory. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 165–176. Springer, Heidelberg (2002)
14. Caucal, D.: On infinite transition graphs having a decidable monadic theory. Theoretical Computer Science 290, 79–115 (2003)
15. Clifford, J., Rao, A.: A simple general structure for temporal domains. In: Temporal Aspects of Information Systems, pp. 17–28. Elsevier Science Publishers, Amsterdam (1988)
16. Colcombet, T., Löding, C.: On the expressiveness of deterministic transducers over infinite trees. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 428–439. Springer, Heidelberg (2004)
17. Combi, C., Franceschet, M., Peron, A.: Representing and reasoning about temporal granularities. Journal of Logic and Computation 14, 51–77 (2004)
18. Comon, H., Cortier, V.: Flatness is not a weakness. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 262–276. Springer, Heidelberg (2000)
19. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms, 2nd edn. McGraw-Hill, New York (2001)
20. Courcelle, B.: Fundamental properties of infinite trees. Theoretical Computer Science 25, 95–169 (1983)
21. Courcelle, B.: The monadic second-order theory of graphs IX: Machines and their behaviors. Theoretical Computer Science 151, 125–162 (1995)
22. Courcelle, B.: The expression of graph properties and graph transformations in monadic second-order logic. In: Handbook of Graph Grammars and Computing by Graph Transformation. Foundations, vol. I, pp. 313–400. World Scientific, Singapore (1997)
23. Courcelle, B., Knapix, T.: The evaluation of first-order substitution is monadic second-order compatible. Theoretical Computer Science 281(1-2), 177–206 (2002)
24. Courcelle, B., Walukiewicz, I.: Monadic second-order logic, graph coverings, and unfoldings of transition systems. Annals of Pure and Applied Logic 92, 35–62 (1998)
25. Dal Lago, U., Montanari, A.: Calendars, time granularities, and automata. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, pp. 279–298. Springer, Heidelberg (2001)
26. Dal Lago, U., Montanari, A., Puppis, G.: Towards compact and tractable automaton-based representations of time granularity. In: Blundo, C., Laneve, C. (eds.) ICTCS 2003. LNCS, vol. 2841, pp. 72–85. Springer, Heidelberg (2003)
27. Dal Lago, U., Montanari, A., Puppis, G.: Compact and tractable automaton-based representations for time granularities. Theoretical Computer Science 373(1-2), 115–141 (2007)
28. Dal Lago, U., Montanari, A., Puppis, G.: On the equivalence of automaton-based representations of time granularities. In: Proceedings of the 14th International Symposium on Temporal Representation and Reasoning (TIME), pp. 82–93. IEEE Computer Society, Los Alamitos (2007)
29. Damm, W.: Languages defined by higher type program schemes. In: Salomaa, A., Steinby, M. (eds.) ICALP 1977. LNCS, vol. 52, pp. 164–179. Springer, Heidelberg (1977)

30. Damm, W.: An algebraic extension of the Chomsky-hierarchy. In: Becvar, J. (ed.) MFCS 1979. LNCS, vol. 74, pp. 266–276. Springer, Heidelberg (1979)
31. Damm, W.: The IO- and OI-hierarchies. Theoretical Computer Science 20, 95–207 (1982)
32. Damm, W., Goerdt, A.: An automata-theoretical characterization of the OI-hierarchy. Information and Control 71(1), 1–32 (1986)
33. Demri, S.: LTL over integer periodicity constraints. Theoretical Computer Science 360(1-3), 96–123 (2006)
34. Dyreson, C.E., Evans, W.S., Lin, H., Snodgrass, R.T.: Efficiently supporting temporal granularities. IEEE Transactions on Knowledge and Data Engineering 12(4), 568–587 (2000)
35. Elgot, C.C., Rabin, M.O.: Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. Journal of Symbolic Logic 31(2), 169–181 (1966)
36. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science. Formal models and semantics, vol. B, pp. 995–1072. Elsevier/MIT Press (1990)
37. Engelfriet, J.: Bottom-up and top-down tree transformations - a comparison. Mathematical Systems Theory 9(3), 198–231 (1975)
38. Engelfriet, J.: Top-down tree transducers with regular look-ahead. Mathematical Systems Theory 10, 289–303 (1977)
39. Engelfriet, J.: Iterated stack automata and complexity classes. Information and Computation 95(1), 21–75 (1991)
40. Euzenat, J., Montanari, A.: Time granularity. In: Handbook of Temporal Reasoning in Artificial Intelligence, pp. 59–118. Elsevier Science Publishers, Amsterdam (2005)
41. Fine, N.J., Wilf, H.S.: Uniqueness theorems for periodic functions. Proceedings of the American Mathematical Society 16, 109–114 (1965)
42. Franceschet, M., Montanari, A.: Time granularities in databases, data mining, and temporal reasoning, by C. Bettini, S. Jajodia, and S.X. Wang (book review). The Computer Journal 45(6), 683–685 (2002)
43. Franceschet, M., Montanari, A., Peron, A., Sciavicco, G.: Definability and decidability of binary predicates for time granularity. In: Proceedings of the 10th International Symposium on Temporal Representation and Reasoning (TIME) and the 4th International Conference on Temporal Logic (ICTL), pp. 192–202. IEEE Computer Society, Los Alamitos (2003)
44. Gécseg, F., Steinby, M.: Tree automata. Akadémiai Kiodó (1984)
45. Ginsburg, S., Spanier, E.: Semigroups, Presburger formulas and languages. Pacific Journal of Mathematics 16(2), 285–296 (1966)
46. Goralwalla, I., Leontiev, Y., Özsu, M.T., Szafron, D., Combi, C.: Temporal granularity for unanchored temporal data. In: Proceedings of the 7th International Conference on Information and Knowledge Management (CIKM), pp. 414–423. Association for Computing Machinery (1998)
47. Hague, M., Murawski, A.S., Ong, C.-H.L., Serre, O.: Collapsible pushdown automata and recursion schemes. In: Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 452–461. IEEE Computer Society, Los Alamitos (2008)
48. Hague, M., Ong, C.-H.L.: Symbolic backwards-reachability analysis for higher-order pushdown systems. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, pp. 213–227. Springer, Heidelberg (2007)

49. Henzinger, T.A., Majumdar, R.: A classification of symbolic transition systems. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 13–34. Springer, Heidelberg (2000)
50. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley Longman Publishing Co., Inc., Amsterdam (2001)
51. Janin, D., Walukiewicz, I.: On the expressive completeness of the propositional μ-calculus with respect to monadic second-order logic. In: Sassone, V., Montanari, U. (eds.) CONCUR 1996. LNCS, vol. 1119, pp. 263–277. Springer, Heidelberg (1996)
52. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM Journal on Computing 22(6), 1117–1141 (1993)
53. Kameda, T., Weiner, P.: On the state minimization of nondeterministic finite automata. IEEE Transactions on Computers 19(7), 617–627 (1970)
54. Knapik, T., Niwiński, D., Urzyczyn, P.: Deciding monadic theories of hyperalgebraic trees. In: Abramsky, S. (ed.) TLCA 2001. LNCS, vol. 2044, pp. 253–267. Springer, Heidelberg (2001)
55. Knapik, T., Niwinski, D., Urzyczyn, P., Walukiewicz, I.: Unsafe grammars and panic automata. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1450–1461. Springer, Heidelberg (2005)
56. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal on Computing 6, 323–350 (1977)
57. Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science 27, 333–354 (1983)
58. Löding, C.: Optimal bounds for the transformation of omega-automata. In: Pandu Rangan, C., Raman, V., Sarukkai, S. (eds.) FST TCS 1999. LNCS, vol. 1738, pp. 97–109. Springer, Heidelberg (1999)
59. Löding, C., Thomas, W.: Alternating automata and logics over infinite words. In: Watanabe, O., Hagiya, M., Ito, T., van Leeuwen, J., Mosses, P.D. (eds.) TCS 2000. LNCS, vol. 1872, pp. 521–535. Springer, Heidelberg (2000)
60. Leban, B., McDonald, D., Foster, D.: A representation for collections of temporal intervals. In: Proceedings of the AAAI National Conference on Artificial Intelligence, vol. 1, pp. 367–371. AAAI Press, Menlo Park (1986)
61. Loma Linda International Heart Institute. Pediatric heart transplantation protocol. Technical report, International Heart Institute, Loma Linda University Medical Center, Loma Linda, CA (2002), `http://www.llu.edu/ihi/pedproto.pdf`
62. Matz, O., Potthoff, A.: Computing small nondeterministic automata. In: Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). BRICS Notes Series, pp. 74–88 (1995)
63. Michel, M.: Complementation is more difficult with automata on infinite words. CNET, Paris (1988) (manuscript)
64. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)
65. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. Theoretical Computer Science 32, 321–330 (1984)
66. Montanari, A.: Metric and Layered Temporal Logic for Time Granularity. ILLC Dissertation Series. Institute for Logic, Language and Computation, University of Amsterdam (1996)

67. Montanari, A., Peron, A., Policriti, A.: Theories of ω-layered metric temporal structures: Expressiveness and decidability. Logic Journal of the Interest Group in Pure and Applied Logic 7(1), 79–102 (1999)
68. Montanari, A., Peron, A., Policriti, A.: The taming (timing) of the states. Logic Journal of the Interest Group in Pure and Applied Logic 8(5) (2000)
69. Montanari, A., Peron, A., Policriti, A.: Extending Kamp's theorem to model time granularity. Journal of Logic and Computation 12(4), 641–678 (2002)
70. Montanari, A., Peron, A., Puppis, G.: On the relationships between theories of time granularity and the monadic second-order theory of one successor. Applied Non-classical Logics 16(3-4), 433–455 (2006)
71. Montanari, A., Policriti, A.: Decidability results for metric and layered temporal logics. Notre Dame Journal of Formal Logic 37(2), 260–282 (1996)
72. Montanari, A., Puppis, G.: Decidability of MSO theories of tree structures. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 434–446. Springer, Heidelberg (2004)
73. Montanari, A., Puppis, G.: Decidability of the theory of the totally unbounded ω-layered structure. In: Proceedings of the 11th International Symposium on Temporal Representation and Reasoning (TIME), pp. 156–160. IEEE Computer Society, Los Alamitos (2004)
74. Montanari, A., Puppis, G.: A contraction method to decide MSO theories of deterministic trees. In: Proceedings of the 22nd Symposium on Logic in Computer Science (LICS), pp. 141–150. IEEE Computer Society, Los Alamitos (2007)
75. Morvan, C.: On rational graphs. In: Tiuryn, J. (ed.) FOSSACS 2000. LNCS, vol. 1784, pp. 252–266. Springer, Heidelberg (2000)
76. Muller, D., Schupp, P.: The theory of ends, pushdown automata, and second-order logics. Theoretical Computer Science 37, 51–75 (1985)
77. Muller, D.E.: Infinite sequences and finite machines. In: Proceedings of the 4th Symposium on Switching Circuit Theory and Logical Design. LNCS, pp. 3–16. IEEE Computer Society, Los Alamitos (1963)
78. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. Theoretical Computer Science 141(1-2), 69–107 (1995)
79. Knapik, T., Niwiński, D., Urzyczyn, P.: Higher-order pushdown trees are easy. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, pp. 205–222. Springer, Heidelberg (2002)
80. Niezette, M., Stevenne, J.: An efficient symbolic representation of periodic time. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 161–168. Association for Computing Machinery (1992)
81. Ning, P., Jajodia, S., Wang, X.S.: An algebraic representation of calendars. Annals of Mathematics and Artificial Intelligence 36, 5–38 (2002)
82. Ohlbach, H.J.: Calendar logic. In: Temporal Logic: Mathematical Foundations and Computational Aspects, vol. 2. Oxford University Press, Oxford (2000)
83. Ohlbach, H.J., Gabbay, D.M.: Calendar logic. Applied Non-classical Logics 8(4), 291–324 (1999)
84. Ong, C.-H.L.: On model-checking trees generated by higher-order recursion schemes. In: Proceedings of the 21st Symposium on Logic in Computer Science (LICS), pp. 81–90. IEEE Computer Society, Los Alamitos (2006)

85. Paige, R., Tarjan, R.E., Bonic, R.: A linear time solution to the single function coarsest partition problem. Theoretical Computer Science 40, 67–84 (1985)
86. Papadimitriou, C.M.: Computational Complexity. Addison-Wesley Longman Publishing Co., Inc., Amsterdam (1994)
87. Perrin, D., Schupp, P.E.: Automata on integers, recurrence distinguishability, and the equivalence and decidability of monadic theories. In: Proceedings of the Symposium on Logic in Computer Science (LICS), pp. 301–304. IEEE Computer Society, Los Alamitos (1986)
88. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Transactions of the American Mathematical Society 141, 1–35 (1969)
89. Rabinovich, A.: Composition theorems for generalized sum and recursively defined types. In: Proceedings of the 11th Workshop on Logic, Language, Information and Computation (WoLLIC 2004). Electronic Notes in Theoretical Computer Science, vol. 123, pp. 209–211 (2005)
90. Rabinovich, A., Thomas, W.: Decidable theories of the ordering of natural numbers with unary predicates. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 562–574. Springer, Heidelberg (2006)
91. Rounds, W.C.: Mappings and grammars on trees. Mathematical Systems Theory 4, 257–287 (1970)
92. Safra, S.: On the complexity of ω-automata. In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science, pp. 319–327. IEEE Computer Society, Los Alamitos (1988)
93. Safra, S.: Complexity of Automata on Infinite Objects. PhD thesis, Weizmann Institute of Science, Rehovot, Israel (1989)
94. Semenov, A.L.: Decidability of monadic theories. In: Chytil, M.P., Koubek, V. (eds.) MFCS 1984. LNCS, vol. 176, pp. 162–175. Springer, Heidelberg (1984)
95. Shelah, S.: The monadic theory of order. Annals of Mathematics 102, 379–419 (1975)
96. Shiloach, Y.: Fast canonization of circular strings. Journal of Algorithms 2(2), 107–121 (1981)
97. Siefkes, D.: Decidable extensions of monadic second-order successor arithmetic. In: Automatentheorie und Formale Sprachen, pp. 441–472. B.I. Hochschultaschenbücher, Mannheim (1970)
98. Sistla, A., Clarke, E.: The complexity of propositional linear temporal logic. Journal of the Association for Computing Machinery 32, 733–749 (1985)
99. Sistla, A.P., Vardi, M.Y., Wolper, P.: Reasoning about infinite computation paths. In: Proceedings of the 24th Annual Symposium on Foundations of Computer Science (SFCS), pp. 185–194. IEEE Computer Society, Los Alamitos (1983)
100. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. Theoretical Computer Science 49, 217–237 (1987)
101. Skiena, S.S.: The Algorithm Design Manual. Springer, Heidelberg (1998)
102. Snodgrass, R.T. (ed.): The TSQL2 Temporal Query Language. Kluwer Academic Publishers, Dordrecht (1995)
103. Stockmeyer, L.: The Complexity of Decision Problems in Automata and Logic. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (1974)
104. Thatcher, J.W.: Generalized sequential machine maps. Journal of Computer and System Sciences 4, 339–367 (1970)

105. Thomas, W.: Infinite trees and automaton definable relations over ω-words. In: Choffrut, C., Lengauer, T. (eds.) STACS 1990. LNCS, vol. 415, pp. 263–277. Springer, Heidelberg (1990)
106. Thomas, W.: Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In: Mycielski, J., Rozenberg, G., Salomaa, A. (eds.) Structures in Logic and Computer Science. LNCS, vol. 1261, pp. 118–143. Springer, Heidelberg (1997)
107. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages, vol. 3, pp. 389–455. Springer, Heidelberg (1997)
108. Thomas, W.: Constructing infinite graphs with a decidable MSO-theory. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 113–124. Springer, Heidelberg (2003)
109. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)
110. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proceedings of the 1st IEEE Symposium on Logic in Computer Science (LICS), pp. 332–344. IEEE Computer Society, Los Alamitos (1986)
111. Walukiewicz, I.: Monadic second-order logic on tree-like structures. In: Puech, C., Reischuk, R. (eds.) STACS 1996. LNCS, vol. 1046, pp. 401–413. Springer, Heidelberg (1996)
112. Walukiewicz, I.: Monadic second-order logic on tree-like structures. Theoretical Computer Science 275, 311–346 (2002)
113. Wijsen, J.: A string-based model for infinite granularities. In: Proceedings of the AAAI Workshop on Spatial and Temporal Granularities, pp. 9–16. AAAI Press, Menlo Park (2000)
114. Zhang, G.: Periodic functions for finite semigroups (1998) (unpublished)
115. Zhang, G.: Automata, boolean matrices, and ultimate periodicity. Information and Computation 152(1), 138–154 (1999)

Notation

$\mathscr{L}(\mathcal{A})$	language recognized by an automaton $\mathcal{A}$ 12, 93
$\mathscr{L}^{\omega}(\mathcal{A})$	language recognized by a Büchi automaton $\mathcal{A}$ 13
$\lvert\mathcal{A}\rvert$	size of an automaton $\mathcal{A}$ 13, 23, 33
$\downarrow_i$	i-th element of a tuple 19, 94
$[\delta, \gamma]$	global transition function of an (N)CSSA 21, 27
IPC^{++}	fragment of Presburger Arithmetic 23
PLTL$^{\mathrm{mod}}$	fragment of Presburger LTL 24
$\Gamma_{\mathcal{A}}$	nesting relation of an NCSSA $\mathcal{A}$ 28
$u_s^{\mathcal{A}}$, $v_s^{\mathcal{A}}$	subwords recognized by an NCSSA $\mathcal{A}$ 29
$\xrightarrow{w}$	existence of a partial run that recognizes w between two given configurations 29, 67
$\lvert w\rvert_a$	number of occurrences of a in w 31, 82
$\lVert\mathcal{A}\rVert$	complexity measure of an NCSSA $\mathcal{A}$ 32
$n(\mathcal{A})$	number of states of an automaton $\mathcal{A}$ 40
U_{A}	set of ultimately periodic words over A 58
$\mathcal{UP}(\mathsf{L})$	set of ultimately periodic words in L 58
$\mathcal{D}om(\mathsf{G})$	domain of a graph G 91
$\mathcal{I}nf(\mathsf{R}\vert\pi)$	set of states that occur infinite often along π 93
Acc_{T}	acceptance problem for a tree T 100
$\mathsf{T}_{\perp}$	completion of T 100

$\mathcal{F}r(R)$	set of all leaves of R 102
$\mathfrak{Unf}(\mathsf{G})$	unfolding of a rooted graph G 98
$\mathfrak{Bch}(\mathsf{R})$	set of all infinite paths in R 102
π_v	access path of v 102
$\mathfrak{Img}(\mathsf{R}\vert\pi)$	set of states along π 102
$[\mathsf{T},\mathsf{R}]$	feature of R on T 102
$\preceq$	preorder relation between runs 103
$\mathscr{T}_{\mathcal{A}}$	set of minor $\mathcal{A}$-types 103
$T_{\Pi}[u]$	unmarked factor of T w.r.t. Π 106
$T^{+}_{\Pi}[u]$	marked factor of T w.r.t. Π 106
$T^{\downarrow v}$	subtree of T rooted at vertex v 116
$\mathsf{T}[\mathsf{F}_{\mathsf{d}}/\mathsf{d}]$	first order substitution in T of d-colored leaves by F_d 117
$\mathsf{T}[\![\mathsf{F}_{\mathsf{d}}/\mathsf{d}]\!]$	second-order substitution in T of d-colored vertices by F_d 117
$\sqsubseteq$	ω-complete partial order on trees 117
h^{-1}_{A}	inverse flip substitution 126
$\mathfrak{Flip}\,\mathfrak{Unf}(\mathsf{T})$	unfolding of T with backward edges and loops 126

Index