Università degli Studi di Udine
Dipartimento di Matematica e Informatica
Dottorato di Ricerca in Informatica

Thesis

Automata for Branching and Layered Temporal Structures

Gabriele Puppis

March 31, 2006

Dipartimento di Matematica e Informatica
Università degli Studi di Udine
Via delle Scienze, 206
33100 Udine
Italia

Abstract

This thesis deals with properties of infinite systems by means of automaton-based representations. The leit-motif underlying the results provided in this thesis is that, once we identify a finite set of properties to be tested, any infinite complex system can be often reduced to a simpler one, which satisfies the same properties and presents strong regularities (e.g., periodicity) in its structure. As a consequence, checking properties of such a system can be done by considering only a finite number of components. The most simple example is the notion of single-string automata, which allow one to represent and reason on any infinite ultimately periodic word by means of a finite number of states and transitions. Such a notion can then be refined by introducing counters in order to compactly represent repeated occurrences of the same substring in an infinite word. Thus, algorithms working on such a kind of representation exploit suitable abstractions of the configuration space of the automaton (these abstractions must take into account the indistinguishability of the valuations of the counters with respect to the transition function). A similar approach can be adopted when dealing with the acceptance problem for tree automata. In such a case, abstractions are defined according to a suitable indistinguishability relation on colored trees with respect to the involved tree automaton. By exploiting such a notion of indistinguishability, one can reduce several instances of the model checking problem for monadic second-order logics (interpreted over non-regular trees) to equivalent instances of the acceptance problem for tree automata over regular trees, which are known to be decidable.

Contents

List of Figures

List of Tables

Introduction

The aim of this thesis is to exploit (different classes of) automata for modeling and reasoning on infinite complex systems. It focuses on two different notions of automata: sequential automata and tree automata.

The first class of automata turns out to be useful when one wants to represent temporal information and to deal 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. 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, for 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 granularities to make their representation as compact as possible.

In Chapter 2 we exploit the notion of sequential automaton, possibly extended with counters, to devise suitable formalisms for the representation and the management of time granularities. In developing these formalisms, we focus our attention on the crucial problems of equivalence (to decide whether two given representations define the same time granularities), granule conversion (to relate granules of a given granularity to those of another one), granularity comparison (to decide whether, for two given sets $\mathcal{G}$ and $\mathcal{H}$ of time granularities, represented by means of suitable automata, there exist granularities $G \in \mathcal{G}$ and $H \in \mathcal{H}$ that satisfy a given relation), and optimization (to compute compact and/or tractable representations of time granularities). We also provide some real-world applications of automaton-based representations of time-granularities.

Chapter 2 is organized as follows. In Section 2.1 we review the most important contributions in this field. In Section 2.2 we formally define time granularities and we introduce some standard relationships between them. In Section 2.3 we describe the algebraic formalism of Calendar Algebra, which allows one to produce compact and user-friendly representations of a large class of time granularities, including finite and infinite (periodical) ones. In Section 2.4 we describe Wijsen's string-based approach to time granularities and we provide an efficient (linear-time) algorithm for solving the equivalence problem for string-based representations of time granularities. We then compare the expressiveness of the string-based formalism with that of Calendar Algebra and we provide suitable algorithms that map Calendar Algebra expressions to equivalent string-based representations. By taking advantage of such a correspondence and by exploiting the proposed solution to the equivalence problem for string-based representations, one can solve the problem of deciding whether two given expressions of Calendar Algebra are equivalent (such a problem has been overlooked in the original presentation of the formalism of Calendar Algebra). In Section 2.5 we introduce the automaton-based approach, we define single-string automata (which recognize single ultimately periodic words), and we show that the resulting framework is as expressive as the formalism of Calendar Algebra and Wijsen's string-based model. We then extend single-string automata with counters in order to allow compact representations of time granularities. In Section 2.6 we refine the notion of single-string automaton extended with counters by introducing suitable restrictions on the structure of the transition functions. These restrictions are then exploited to devise efficient algorithms for the problems of granule conversion and equivalence. We also consider two kinds of optimization problems: the size-optimization problem (which consists in computing the most compact representation of a given time granularity) and the complexity-optimization problem (which consists in computing a representation of a given time granularity on which crucial algorithms, such as granule conversion algorithms, run fastest). In Section 2.7 we focus on the problem of representing and reasoning on (possibly infinite) sets of periodical time granularities, rather than single time granularities. We give a characterization of the subclass of Büchi automata that recognize exactly the regular ω-languages consisting of ultimately periodic words only and we give efficient solutions to several basic problems

in automata theory, precisely, the emptiness problem, the membership problem, the equivalence problem, and the size-optimization problem. Finally, we deal with the granularity comparison problem by reducing it to the emptiness problem.

As for the notion of tree automata, they come into play in the automatic verification of properties of infinite-state 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 Kripke structure. Thus, 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. If we choose monadic second-order (MSO) logic to specify properties of a tree-shaped structure, then the model checking problem can be reduced to the acceptance problem for Rabin tree automata [60], 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 for establishing the decidability of the model checking problem for MSO 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 complex non-regular trees to regular ones. We show that such an automaton-based approach works effectively for a large class of trees, called reducible trees, which naturally extend the notion of regular tree and include meaningful relational structures inside and outside the so-called Caucal hierarchy. We also prove closure properties for the class of reducible trees with respect to several natural operations on trees. Finally, we consider the model checking problem for the chain fragment of MSO logics interpreted over the so-called multi-layered temporal structures, which are tree-shaped structures suited for modeling and reasoning about temporal relations at different 'grain levels'.

Chapter 3 is organized as follows. In Section 3.2 we preliminary review some commonly-used operations on colored trees, such as finite-state recolorings, tree substitutions, tree transductions, etc. Then, we compare these operations and we provide characterizations for some of them. In Section 3.3 we explain our automaton-based method to establish the decidability of MSO theories of colored tree structures. First, by taking advantage of well-known results from automata theory, we reduce the model checking problem for MSO logic to the acceptance problem for Muller tree automata, which is decidable for the class of regular colored trees. Then, we extend such a result to non-regular trees by exploiting a suitable notion of indistinguishability of trees with respect to tree automata. Such a notion allows the replacing of a tree by a 'retraction' of it (i.e., a suitable abstraction of the original tree). By exploiting such a method, one can reduce, in a uniform way, instances of the model checking problem (over non-regular trees) to equivalent instances of the acceptance problem for tree automata over regular trees. In Section 3.4 we show that such an automaton-

based approach to the decidability of MSO logics works effectively for a large class of complex (possibly non-regular) trees, which we call reducible trees. We prove closure properties for the class of regular trees with respect to several natural transformations of trees (for instance, finite-state recolorings with bounded lookahead, regular tree morphisms) and then we extend these properties to the class of reducible trees. Such results, besides showing the robustness of the class of reducible trees, provide a neat framework to reason on retractions of trees and to easily transfer decidability results. In Section 3.5 we compare the expressiveness of our approach with that of other frameworks proposed in the literature. Precisely, we show that the class of reducible trees includes several interesting decidable structures, like the unfoldings of the deterministic context-free graphs, the algebraic trees, and the tree generators for the levels of the Caucal hierarchy. Moreover, we consider the model checking problem for MSO logics interpreted over the so-called multi-layered temporal structures. In order to generalize previous results in the literature, we introduce a new notion of multi-layered structure, called k-refinable totally unbounded ω-layered structure, and we show that the MSO theory of such a kind of structure (extended with a suitable coloring predicate) subsumes the MSO theories of both downward unbounded and upward unbounded ω-layered structures. Then, by exploiting the automaton-based approach described in Section 3.3, we establish the decidability of MSO theories of totally unbounded ω-layered structures. Finally, we give new decidability results for the chain fragment of MSO logic interpreted over the totally unbounded ω-layered structure extended with suitable binary predicates that pair vertices belonging to either the same level or the same column.

The results of Section 2.4 that deal with the equivalence problem for Wijsen's string-based specifications and the mapping of expressions of Calendar Algebra are based on [40]. The results of Section 2.6 on restricted labeled single-string automata have been originally presented in [41] and later refined in [40, 42]. The results of Section 2.7 have been published in [12, 13]. Finally, a preliminary version of the results presented in Chapter 3 has been published in [82, 83, 84]. The current version appeared in [85].

1

Preliminaries

In this chapter, we revise some basic notation and terminology about words, languages, graphs, automata, logics, and periodicity and we recall/establish noticeable properties for them.

From now on, for any given $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, ..., n\}$ and by $[\omega]$ the set $\{1, 2, ...\}$ of the positive natural numbers. Given an n-tuple t and an index $i \in [n]$, we denote by $\mathcal{P}rj_i(t)$ the i-th element of t. Moreover, if f is a generic function from a set A to another set B, then, by a little abuse of notation, we extend f to sequences and sets of elements belonging to A. For instance, if A is a set of n-tuples, then $\mathcal{P}rj_1(A)$ denotes the set of all elements that occur in the first position of some n-tuple $t \in A$.

1.1 Words and languages

A finite word over an alphabet Σ is a mapping w from a set $[n]$, where $n \in \mathbb{N}$, to Σ. The parameter n is said to be the length of the word w, shortly denoted $|w|$, and for every index $i \in [n]$, $w[i]$ denotes the i-th character of w. The symbol ε denotes the empty word (hence, $|\varepsilon| = 0$). An infinite word over Σ is a mapping w from $[\omega]$ to Σ and we assume that its length $|w|$ is ω.

Given any (finite or infinite) word w, and two indices $i, j \in [|w|]$, we denote by $w[i, j]$ the substring $w[i]w[i+1]...w[j-1]w[j]$ (if $j < i$, then $w[i, j] = \varepsilon$). If $i = 1$, then $w[i, j]$ is said to be a prefix of w. If $j = |w|$, then $w[i, j]$ is said to be a suffix of w. The concatenation operation $\cdot$ maps a pair of finite words u, v to the finite word $u \cdot v$ having length $|u| + |v|$ and such that $(u \cdot v)[i] = u[i]$ for every $i \in |u|$ and $(u \cdot v)[i + |u|] = v[i]$ for every $i \in |v|$. Such an operation is then extended to infinite words in the natural way (e.g., $u \cdot v = u$ whenever u is an infinite word). The k-repetition v^k of a finite word v is recursively defined as follows: $v^0 = \varepsilon$ and $v^{k+1} = v^k \cdot v$. Similarly, the ω-repetition v^ω of a non-empty finite word v is the infinite word $v \cdot v \cdot v \cdot ....$

A language (resp., an ω-language) over the alphabet Σ is a set of finite (resp., infinite) words over Σ. If L is a language, then we denote by L^k the set of all finite words of the form $u_1 \cdot ... \cdot u_k$, where each u_i is a word from L (note that $L^0 = \{\varepsilon\}$). We further denote by L^* the language $\bigcup_{k \in \mathbb{N}} L^k$ and by L^ω the ω-language $\{v_1 \cdot v_2 \cdot ... : \forall\, i > 0.\ v_i \in L \setminus \{\varepsilon\}\}$.

1.2 Graphs and trees

We use the term *label* (resp., *color*) to mean a symbol, usually taken from a finite set Λ (resp., Σ), marking an edge (resp., a vertex) of a graph. A Λ-labeled (directed simple[1]) graph is a tuple $G = (S, (E_a)_{a\in\Lambda})$, where S (also denoted $\mathcal{D}om(G)$) is a countable set of vertices and $(E_a)_{a\in\Lambda}$ are binary relations defining the edges and their labels.

Given an edge $e = (v, v') \in E_a$, v is called the *source* of e and v' is called the *target* of e. A vertex v' is said to be *adjacent* to v if $(v, v') \in E_a$ or $(v', v) \in E_a$ for some $a \in \Lambda$ (namely, if v and v' are ends of the same edge). A vertex v is an *a-predecessor* of v' (or, equivalently, v' is an *a-successor* of v) in G if G contains an a-labeled edge from v to v'. We say that d is the *out-degree* of a vertex $v \in \mathcal{D}om(G)$ if there are exactly d successors of v in G. Furthermore, we say that the graph is *deterministic* if, for every $v \in \mathcal{D}om(G)$ and for every $a \in \Lambda$, there is at most one a-successor of v in G. Clearly, under the proviso that the set of labels is finite, the vertices in any deterministic graph have uniformly bounded out-degree, namely, there is k $(= |\Lambda|)$ such that for every vertex $v \in \mathcal{D}om(G)$, the out-degree of v does not exceed k.

Given a graph $G = (S, (E_a)_{a\in\Lambda})$ and a set $V \subseteq S$ of vertices, the *subgraph of G induced by V* is the graph $G|_V = (V, (E'_a)_{a\in\Lambda})$, where, for each $a \in \Lambda$, $E'_a = E_a \cap (V \times V)$. An *expanded graph* is a graph equipped with a tuple $\bar{V}$ of unary predicates, namely, a structure of the form $(S, (E_a)_{a\in\Lambda}, \bar{V})$, where $\bar{V} = (V_1, ..., V_m)$ and $V_i \subseteq S$ for all $i \in [m]$. Any expanded graph $(G, \bar{V})$ is canonically represented by a Σ-*colored* graph $G_{\bar{V}} = (G, C)$, called the *canonical representation* of $(G, \bar{V})$, where $\Sigma = \mathscr{P}([m])$ and $C : \mathcal{D}om(G) \rightarrow \Sigma$ is a coloring function mapping a vertex $v \in \mathcal{D}om(G)$ to the set of all indices $i \in [m]$ such that $v \in V_i$. Hereafter, we shortly denote the color of a vertex v in $G_{\bar{V}}$ by $G_{\bar{V}}(v)$.

A finite *path* in G from v to v' is a finite sequence $\pi = e_1e_2...e_n$ of edges such that (i) the source of e_1 is v, (ii) the target of e_n is v', and (iii) for all $i \in [n-1]$, the target of e_i is the source of e_{i+1}. The *length* $|\pi|$ of the path π is the number of its edges. We similarly define infinite paths, namely, infinite sequences of edges the form $\pi = e_1e_2e_3...$ (in such a case, we assume that $|\pi| = \omega$). For any finite (resp., infinite) path π and for any index $0 \leq i \leq |\pi|$ (resp., $i \geq 0$), we denote by π_i the $i+1$-th vertex along π, i.e., the source of the first edge in π if $i = 0$ or the target of the i-th edge in π if $i > 0$. For instance, if π is a finite path, then $\pi_{|\pi|}$ denotes the last vertex along π. Furthermore, given a finite (resp., infinite) path π, we say that $w \in \Lambda^*$ (resp., $w \in \Lambda^\omega$) is the sequence of labels along π if $|w| = |\pi|$ and, for all $i \in [|\pi|]$ (resp., for all $i \geq 1$), $w[i]$ is the label of the i-th edge $\pi[i]$ in π. Notice that, if the graph G is deterministic, then, given a pair of vertices v, v' in G and given a (finite or infinite) word w over Λ, there is at most one path π from v to v' such that w is the sequence of labels along π. Similarly, given a finite (resp., infinite) path π in a Σ-colored graph G, we say that $w \in \Sigma^*$ (resp., $w \in \Sigma^\omega$) is the sequence of colors along π, and we shortly denote it by $G|\pi$, if $|w| = |\pi| + 1$ (resp., $|w| = \omega$) and, for all $0 \leq i \leq |\pi|$ (resp., for all $i \geq 0$), $w[i+1]$ is the color of the vertex π_i.

[1] A directed graph is said to be simple if distinct edges cannot have the same source vertex, the same target vertex, and the same label.

Given two graphs G and G', an *isomorphism* from G to G' is a bijection f from $\mathcal{D}om(G)$ to $\mathcal{D}om(G')$ such that (v, v') is an a-labeled edge of G iff $(f(v), f(v'))$ is an a-labeled edge of G'. If G and G' are colored graphs, we further require that v is a c-colored vertex of G iff $f(v)$ is a c-colored vertex of G'. Intuitively, an isomorphism is simply a renaming of the vertices of a graph. If there exists an isomorphism from a (colored) graph G to a (colored) graph G', then we say that G and G' isomorphic. Note that the relation of isomorphism is an equivalence, namely, it transitive, reflexive, and symmetrical. Unless strictly necessary, we will not distinguish between isomorphic graphs. We now recall the notion of *bisimilarity*. Two graphs G and G' are said to be bisimilar if there is a relation $\sim \subseteq \mathcal{D}om(G) \times \mathcal{D}om(G')$, called bisimulation, such that for every pair of vertices $v \in \mathcal{D}om(G)$ and $v' \in \mathcal{D}om(G')$ and for every label $a \in \Lambda$, if $v \sim v'$, then for every a-successor w of v in G, there is an a-successor w' of v' in G' (and vice versa, for every a-successor w' of v', there is an a-successor w of v) such that $w \sim w'$. If G and G' are colored graphs, we further require that $v \sim v'$ implies that $G(v) = G'(v')$. Notice that the relation of bisimilarity is an equivalence relation. It is however coarser than the isomorphism relation, since there exist some non-isomorphic bisimilar graphs, but isomorphic graphs are always bisimilar.

We shall also consider (unranked rooted) *trees*, namely, graphs such that for every vertex v, there exists a unique path, called *access path*, from a distinguished source vertex (called *root*) to v. We say that v is an *ancestor* of v' (or, equivalently, v' is a *descendant* of v) in a tree if there is a (possibly empty) path from v to v'. Notice that the ancestor relation induces a partial left-linear order in the domain of the tree. Hence, we can identify each vertex in a tree (resp., in a deterministic tree) with its unique access path (resp., with the unique sequence of labels along its access path). In particular, we can view any *deterministic Λ-labeled Σ-colored tree* as a partial function T from Λ^* to Σ, whose domain $\mathcal{D}om(T)$ is a *prefix-closed*[2] language over the set Λ of edge labels. In this perspective, we shall denote the color of a vertex v in T by $T(v)$ and, whenever T is well understood, the a-successor of v in T by $v \cdot a$. Unless otherwise stated, we assume that trees are deterministic, labeled over a finite set Λ, colored over a finite alphabet Σ, and identified by a partial function from Λ^* to Σ. Sometimes, if there is a well understood ordering of the labels in the set Λ, we can also use (unranked) terms to denote trees; for instance, if $\Lambda = \{a_1 < a_2 < a_3\}$, 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 targets of two edges labeled respectively by a_1 and a_2.

The *leaves* of a tree T are all and only the vertices $v \in \mathcal{D}om(T)$ such that for every $a \in \Lambda$, $v \cdot a \notin \mathcal{D}om(T)$; the other vertices are called *internal vertices*. The *frontier* of a tree T, denoted $\mathcal{F}r(T)$, is the set of all leaves of T; note that this is a *prefix-free*[3] language. A *branch* is a maximal path, namely, a path that is not a proper prefix of any other path in $\mathcal{D}om(T)$. We denote by $\mathcal{B}ch(T)$ the set of all *infinite* branches of T. For any vertex $v \in \mathcal{D}om(T)$, the subtree of T rooted at v, denoted $T^{\downarrow v}$, is the tree T' such that

[2] A language L is prefix-closed if for every $u \in L$ and for every prefix v of u, $v \in L$.

[3] A language L is prefix-free if for every $u \in L$ and for every proper prefix v of u, $v \notin L$.

- $\mathcal{D}om(T') = \{w : v \cdot w \in \mathcal{D}om(T)\}$,
- $T'(w) = T(v \cdot w)$ for every $w \in \mathcal{D}om(T^{\downarrow v})$.

Moreover, given a subset L of $\mathcal{D}om(T)$ closed under the ancestor relation of T, we denote by $T|_L$ the tree obtained from T by restricting the domain to L, formally, the tree T' defined by

- $\mathcal{D}om(T') = \mathcal{D}om(T) \cap L$,
- $T'(v) = T(v)$ for every $v \in \mathcal{D}om(T')$.

A *full* tree is a deterministic tree such that, whenever $(u, u \cdot a) \in E_a$ holds for some $a \in \Lambda$, then $(u, u \cdot a') \in E_{a'}$ holds for every $a' \in \Lambda$. An *infinite complete* tree is a deterministic tree T such that for every word $w \in \Lambda^*$, there is an access path from the root of T labeled with w. A deterministic tree T is said to be *regular* if T contains only finitely many non-isomorphic subtrees.

We now recall the notion of *unfolding* of a graph. Given a (colored) graph G and a source vertex $v_0 \in \mathcal{D}om(G)$, the unfolding of G from v_0, denoted $\mathcal{U}nf(G, v_0)$ is the (colored) tree whose domain consists of all and only the finite paths from v_0 to a vertex $v \in \mathcal{D}om(G)$, and where the a-labeled edges are all and only the pairs of paths (π, π') such that π' extends π with an a-labeled edge of G. Note that, if every vertex of G is reachable from v_0, then the unfolding $\mathcal{U}nf(G, v_0)$ is a tree bisimilar to G. Moreover, if G is deterministic, then $\mathcal{U}nf(G, v_0)$ is deterministic as well and this implies that $\mathcal{U}nf(G, v_0)$ is the unique tree (up to isomorphisms) which is bisimilar to G (to see this, simply notice that bisimilar deterministic trees are isomorphic).

In Chapter 3, an important role is played by the so-called (Σ-colored) Δ-*augmented* trees. These trees have internal nodes colored over Σ and leaves colored over $\Sigma \cup \Delta$, with Δ being a finite set disjoint from Σ. Even though a Σ-colored Δ-augmented tree can be viewed as a $\Sigma \cup \Delta$-colored tree, it is obviously not true that a $\Sigma \cup \Delta$-colored tree is a Σ-colored Δ-augmented tree. On the other hand, a Σ-colored tree can be always thought of as a Σ-colored Δ-augmented tree, for any given Δ. We call the Δ-colored leaves in a Δ-augmented tree *markers*, *placeholders*, or *variables*, depending on the context and the kind of operations we are interested in.

1.3 Monadic second-order logic

Roughly speaking, 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 fix a signature $\mathcal{R} = (\mathtt{R}, \rho)$, namely, a pair consisting of a finite set $\mathtt{R}$ of relational symbols and a ranking function ρ from $\mathtt{R}$ to $\mathbb{N}^+$. MSO formulas over the signature $\mathcal{R}$ are built up starting from atoms of the form $x = y$, $x \in Y$, and $\mathtt{r}(x_1, ..., x_{\rho(\mathtt{r})})$, for each relational symbol $\mathtt{r} \in \mathtt{R}$. Basic formulas can be combined by means of the Boolean connectives $\vee$ and $\neg$ and the existential quantifications $\exists\, x.$ and $\exists\, X.$ over first-order variables (denoted by lowercase letters) and second-order variables (denoted by uppercase letters). We say that a variable occurring in a formula φ is *free* if it is not bounded by an existential quantifier and we write $\varphi(x_1, ..., x_n, X_1, ..., X_m)$ to mean that the free variables in φ are only $x_1, ..., x_n, X_1, ..., X_m$. In order to define the semantics of a MSO formula, we

briefly review the notion of *relational structure*: this is a tuple $\mathcal{S}$ consisting of a countable (possibly infinite) set S of elements and a $\rho(\mathtt{r})$-ary relation r on S, for each relational symbol $\mathtt{r} \in \mathtt{R}$. We then define an *assignment* for an MSO formula $\varphi(x_1, ..., x_n, X_1, ..., X_m)$ as a tuple $\sigma = (s_1, ..., s_n, S_1, ..., S_m)$, where $s_i \in S$ for all $i \in [n]$ and $S_j \subseteq S$ for all $j \in [m]$. We say that $\varphi(x_1, ..., x_n, X_1, ..., X_m)$ *holds* in the relational structure $\mathcal{S}$ with the assignment $\sigma = (s_1, ..., s_n, S_1, ..., S_m)$, and we shortly write $\mathcal{S} \vDash \varphi[s_1, ..., s_n, S_1, ..., S_m]$, if one of the following conditions holds:

- φ is of the form $x_i = x_j$ and $s_i = s_j$;
- φ is of the form $x_i \in X_j$ and $s_i \in S_j$;
- φ is of the form $\mathtt{r}(x_{i_1}, ..., x_{i_{\rho(r)}})$ and $(s_{i_1}, ..., s_{i_{\rho(r)}}) \in r$;
- 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 $\mathcal{S} \vDash \varphi_1[s_1, ..., s_n, S_1, ..., S_m]$ or $\mathcal{S} \vDash \varphi_2[s_1, ..., s_n, S_1, ..., S_m]$;
- φ is of the form $\neg\varphi'(x_1, ..., x_n, X_1, ..., X_m)$ and $\mathcal{S} \nvDash \varphi'[s_1, ..., s_n, S_1, ..., S_m]$;
- φ is of the form $\exists\, x_{n+1}.\ \varphi'(x_1, ..., x_{n+1}, X_1, ..., X_m)$ and there is $s_{n+1} \in S$ such that $\mathcal{S} \vDash \varphi'[s_1, ..., s_{n+1}, S_1, ..., S_m]$
- φ is of the form $\exists\, X_{m+1}.\ \varphi'(x_1, ..., x_n, X_1, ..., X_{m+1})$ and there is $S_{m+1} \subseteq S$ such that $\mathcal{S} \vDash \varphi'[s_1, ..., s_n, S_1, ..., S_{m+1}]$.

We say that an n-ary relation $r \subseteq S^n$ is *MSO-definable* in the relational structure $\mathcal{S}$ if there is an MSO formula $\varphi(x_1, ..., x_n)$ such that $(s_1, ..., s_n) \in r$ iff $\mathcal{S} \vDash \varphi[s_1, ..., s_n]$. Note that, in such a case, the extension of the logic with the relational symbol $\mathtt{r}$, corresponding to the relation r, does not increase the expressive power (this is because every occurrence of the relational symbol $\mathtt{r}$ can be replaced by the equivalent formula φ). As an example, if $\mathtt{r}$ is the symbol corresponding to a binary relation $r \subseteq S^2$, then the transitive and reflexive closure of r, denoted r^*, is MSO-definable in $\mathtt{R}$, precisely, by the formula $\varphi(x, y) = \forall\, Z.\ \big(Z(x) \ \wedge \ \forall\, z_1.\ \forall\, z_2.\ (Z(z_1) \ \wedge \ \mathtt{r}(z_1, z_2)) \ \rightarrow \ Z(z_2)\big) \ \rightarrow \ Z(y)$. Following this idea, one can define a new relational structure $\mathcal{S}'$, over a signature $\mathcal{R}' = (\mathtt{R}', \rho')$ by using MSO formulas over the signature $\mathcal{R} = (\mathtt{R}, \rho)$. Formally, an *MSO-definable interpretation* of $\mathcal{R}'$ into $\mathcal{R}$ is a family $\mathcal{I} = \big(\psi(x), (\varphi_{\mathtt{r}}(x_1, ..., x_{\rho'\mathtt{r}})_{\mathtt{r} \in \mathtt{R}'}\big)$ of MSO formulas, which gives rise to the relational structure $\mathcal{S}'$, whose domain is the set of all elements $s \in S$ such that $\mathcal{S} \vDash \psi[s]$ and where, for each $\mathtt{r} \in \mathtt{R}'$, the relation r corresponding to $\mathtt{r}$ is the set of all and only the $\rho'(\mathtt{r})$-tuples of elements $(s_1, ..., s_{\rho'(\mathtt{r})})$ such that $\mathcal{S} \vDash \varphi[s_1, ..., s_{\rho'(\mathtt{r})}]$. Henceforth, by a slight abuse of notation, we will not distinguish between relational symbols in a signature $\mathcal{R}$ and the corresponding relations in a structure $\mathcal{S}$.

The *model checking* problem for a given relational structure $\mathcal{S}$ is the problem of deciding, given an MSO sentence φ (i.e., a formula with no free variables), whether $\mathcal{S} \vDash \varphi$. The set of all sentences that hold in the structure $\mathcal{S}$ is called the *MSO theory* of $\mathcal{S}$ and it is denoted by $MTh(\mathcal{S})$. Thus, we say that $MTh(\mathcal{S})$ is recursive (or decidable) if the model checking problem for $\mathcal{S}$ is decidable. Note that graphs can be thought of as particular relational structures, where all relations are binary. Similarly, an expanded graph $(G, V_1, ..., V_m)$ (and hence its canonical representation $G_{\bar{V}}$) can be thought of as a relational structure endowed with the assignment $(V_1, ..., V_m)$ for some free second-order variables $X_1, ..., X_m$. In this case, we say that the model checking for the colored graph $G_{\bar{V}}$ is the problem of deciding, for any given MSO formula $\varphi(X_1, ..., X_m)$, whether $G \vDash \varphi[V_1, ..., V_m]$. Accordingly, the MSO theory

of $G_{\bar{V}}$ is the set of all the formulas with m free variables that holds in G under the assignment $\bar{V}$. If a (colored) graph G has a decidable MSO theory, then any (colored) graph $\mathcal{I}(G)$ obtained from G via an MSO-definable interpretation $\mathcal{I}$ has a decidable MSO theory as well (to see this, notice that any formula φ over $\mathcal{I}(G)$ can be translated into an equi-satisfiable formula φ' over G). It can be also proved that the decidability of the MSO theory of the unfolding $\mathcal{U}nf(G, v_0)$ of a (colored) graph G from an MSO-definable vertex v_0 is reducible to the decidability of the MSO theory of G itself. More generally, we say that a transformation t, which maps a relational structure to another relational structure, is *MSO-compatible* if, given any MSO formula φ over $t(\mathcal{S})$, one can compute a formula φ' over $\mathcal{S}$ such that $t(\mathcal{S}) \vDash \varphi$ iff $\mathcal{S} \vDash \varphi'$. Clearly, MSO-compatible transformations preserve the decidability of MSO theories. In [37] Courcelle proved that the unfolding of a *deterministic* graph is MSO-compatible. Subsequently, in [100, 115, 116] the result has been extended to a more powerful transformation, which is called tree-iteration. As a matter of fact, in [38], the unfolding of a graph G is shown to be MSO-definable in the tree-iteration of G, thus generalizing previous results of Courcelle.

By exploiting the above mentioned results, it is possible to generate a number of interesting structures enjoying a decidable MSO theory. For instance, one can start with finite graphs, which are obviously decidable, and then apply unfoldings to them, thus obtaining (decidable) regular trees. In succession, one can apply MSO-definable interpretations to regular trees to obtain other (decidable) graphs (the so-called prefix-recognizable graphs [23, 25]), and so on. Such an alternation of unfoldings and MSO-definable interpretations yields a hierarchy of decidable graphs and trees, called Caucal hierarchy [24, 114]. Moreover, in [18] the graphs belonging to this hierarchy have been characterized in terms of configuration graphs of high-order pushdown automata (see also [44, 45, 46]). From the strictness of the hierarchy of the languages recognized by high-order pushdown automata (see [43, 53]), it follows that the Caucal hierarchy is strictly increasing as well.

1.4 Sequential and tree automata

Here we define sequential automata and tree automata, which are used to represent sets of (finite or infinite) words and sets of trees, respectively. We also recall some well-known results from automata theory (for a complete discussion on sequential and tree automata, we refer to [113]).

Definition 1.4.1 *A* sequential (finite-state) automaton *is a tuple* $M = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$, *where*

- S *is a finite set of* states,
- Σ *is a finite* alphabet,
- $\delta \subseteq S \times \Sigma \times S$ *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 M over a finite (resp., infinite) word $w \in \Sigma^*$ (resp., $w \in \Sigma^\omega$) is a finite (resp., infinite) sequence of states ρ such that

- $|\rho| = |w| + 1$ ($|\rho| = |w| = \omega$ if w is infinite),
- for every $i \in [|\rho| - 1]$, $(\rho[i], w[i], \rho[i+1]) \in \delta$.

Acceptance conditions can be defined for both finite and infinite words. If w is a finite word and ρ is a run of M on w such that $\rho[1] \in \mathcal{I}$ and $\rho[|\rho|] \in \mathcal{F}$, then we say that ρ is a *successful run* on w. The finite word w is *accepted* by M iff there exists a successful run of M on w. The language *recognized* by M, denoted $\mathscr{L}(M)$, is the set of all and only the finite words that are accepted by M. The languages recognized by a finite-state sequential automaton are called *regular* languages.

When considering languages of infinite words, the automaton M is called (sequential) *Büchi automaton* and a different kind of acceptance condition is adopted (here we only consider Büchi acceptance conditions, even though different, but equivalent, acceptance conditions can be found in the literature). Given a run ρ of M on an infinite word w, we say that ρ is *successful* iff $\rho[1] \in \mathcal{I}$ and there is at least one final state $s \in \mathcal{F}$ that occurs infinitely often in ρ. In such a case w is *accepted* by M. The ω-language *recognized* by a Büchi automaton M, denoted $\mathscr{L}^\omega(M)$, is the set of all and only the infinite words that are accepted by M. The Büchi-recognizable languages are also called *regular* ω-languages.

A finite-state automaton (resp., a Büchi automaton) is said to be *deterministic* if for every pair of transitions of the form (s, a, s') and (s, a, s''), we have $s' = s''$. It is known that, while deterministic finite-state automata are expressively equivalent to non-deterministic finite-state automata (cf. [64]), deterministic Büchi automata are less expressive than non-deterministic Büchi automata (cf. [113]). Moreover, it is possible to extend the notions of finite-state and Büchi automata by introducing ε-transitions, namely, transitions which allow an automaton to change its control state without reading any symbol from the input word. However, the resulting notions have been proved to be expressively equivalent to the ones we introduced above (cf. [64, 113]).

We briefly recall the following fundamental results due to Büchi.

Proposition 1.4.2 (Büchi [5]) *Büchi-recognizable languages are effectively closed under union, intersection, and complementation, namely, given two Büchi automata M and N, one can compute a Büchi automaton $M \cup N$ (resp., $M \cap N$, $\bar{M}$) recognizing the ω-language $\mathscr{L}^\omega(M) \cup \mathscr{L}^\omega(N)$ (resp., $\mathscr{L}^\omega(M) \cap \mathscr{L}^\omega(N)$, $\Sigma^\omega \setminus \mathscr{L}^\omega(M)$).*

By viewing any infinite word $w \in \Sigma^\omega$ as an infinite linear structure $(\mathbb{N}, <, C)$, where $C(i) = w[i+1]$ for every $i \in \mathbb{N}$, the following characterization of Büchi-recognizable languages follows.

Theorem 1.4.3 (Büchi [5]) *An ω-language L is Büchi-recognizable iff it is MSO-definable in $(\mathbb{N}, <)$, namely, there is an MSO-formula φ whose linear models are all and only the infinite words belonging to L.*

This makes it possible to reduce the model checking problem for MSO logic interpreted over $(\mathbb{N}, <)$ to the emptiness problem for Büchi automata, which is known to be decidable.

Corollary 1.4.4 (Büchi [5]) *The MSO theory of* $(\mathbb{N}, <)$ *is decidable.*

We now introduce tree automata, namely, automata recognizing sets of (possibly infinite) colored trees.

Definition 1.4.5 *A* tree automaton *is a tuple* $M = (S, \Sigma, \Lambda, \delta, \mathcal{I}, \mathcal{A})$*, where*

- S *is a finite set of* states*,*
- Σ *is a finite* alphabet*, namely a finite set of colors for the vertices of the input tree,*
- Λ *is a finite set of labels,*
- $\delta \subseteq S \times \Sigma \times S^{\Lambda}$ *is a* transition relation*,*
- $\mathcal{I} \subseteq S$ *is a set of* initial states*,*
- $\mathcal{A} \subseteq S^{\omega}$ *is an* acceptance condition*, namely, a set consisting of infinite sequences of states.*

Given an *infinite complete* Λ-labeled Σ-colored tree T, a *run* of the automaton M on T is any infinite complete Λ-labeled S-colored tree R such that for every $v \in \mathcal{D}om(R)$,

$$\big(R(v), T(v), (R(v \cdot a))_{a \in \Lambda}\big) \in \delta$$

where $v \cdot a$ denotes the a-successor of v in R. We say that R is *successful*, and hence T is *accepted* by M, if $R(\varepsilon) \in \mathcal{I}$ and for every infinite path π in R, $R|\pi$ belongs to $\mathcal{A}$ (namely, the sequence of states occurring along π satisfies the acceptance condition). The language $\mathscr{L}(M)$ *recognized* by M is the set of all and only the (infinite complete) trees accepted by M.

Like Büchi automata, one can use different, but equivalent, acceptance conditions, usually envisaging the occurrences of states in a sequence. Given an infinite sequence $\alpha \in S^{\omega}$, we denote by $\mathcal{I}nf(\alpha)$ the set of all elements that occur infinitely often in α. We further denote by $\mathcal{I}mg(\alpha)$ the set of all elements that occur at least once in a (finite or infinite) sequence α. *Rabin acceptance conditions* are specified by a set $\mathcal{A}$ of the form $\{\alpha \in S^{\omega} : \exists\, i \in [n].\, \mathcal{I}nf(\alpha) \cap E_i = \emptyset \ \wedge\ \mathcal{I}nf(\alpha) \cap F_i \neq \emptyset\}$, where $(E_i, F_i)_{i \in [n]}$ is a finite set of acceptance pairs. Other options are given by *Rabin chain acceptance conditions*, which are Rabin acceptance conditions where $E_1 \subsetneq F_1 \subsetneq E_2 \subsetneq F_2 \subsetneq \ldots \subsetneq E_n \subsetneq F_n$, *parity acceptance conditions*, which are sets of the form $\mathcal{A} = \{\alpha \in S^{\omega} : max(\mathcal{I}nf(\Omega(\alpha)))$ is even$\}$, where Ω is a function mapping states to integers and $\Omega(\alpha)$ is the natural extension to infinite sequences of states, or *Muller acceptance conditions*, which are sets of the form $\mathcal{A} = \{\alpha \in S^{\omega} : \mathcal{I}nf(\alpha) \in \mathcal{F}\}$, where $\mathcal{F} \subseteq \mathscr{P}(S)$ is a family of sets of states. As a matter of fact, it is easy to see that Rabin chain acceptance conditions and parity acceptance conditions are equivalent, namely, given a set of acceptance pairs $(E_i, F_i)_{i \in [n]}$, with $E_i \subsetneq F_i$ for all $i \in [n]$ and $F_i \subsetneq E_{i+1}$ for all $i \in [n-1]$, one can set $\Omega(s) = 2i - 1$ for all $s \in E_i \setminus F_{i-1}$ and $\Omega(s) = 2i$ for all $s \in F_i \setminus E_i$. Conversely, given a function $\Omega : S \to [2n]$ such that $\forall\, i \in [2n].\ \exists\, s \in S.\ \Omega(s) = i$ (if this is not the case, one can extend the automaton with new states and possibly shift the values of Ω), one can set $E_i = \{s \in S : \Omega(s) \leq 2i - 1\}$ and $F_i = \{s \in S : \Omega(s) \leq 2i\}$. In fact, it can be proved that Rabin tree automata, Rabin chain tree automata, parity tree automata, and Muller

tree automata are equivalent in the sense that, given any of such automata, one can compute Rabin, Rabin chain, parity, and Muller tree automata recognizing the same language [60, 113, 86, 6]. From now on, we shall use the term tree automaton to mean a Muller tree automaton, keeping in mind that the results presented in this thesis are applicable to any other equivalent notion of tree automaton.

In [96] Rabin generalizes the results of Büchi for infinite words to the case of infinite complete colored trees.

Proposition 1.4.6 (Rabin [96]) *The languages recognized by Rabin tree automata are effectively closed under union, intersection, and complementation.*

Theorem 1.4.7 (Rabin [96]) *A language consisting of infinite complete colored trees is recognized by a Rabin tree automaton iff it is MSO-definable in the infinite complete tree, namely, there is an MSO-formula φ whose infinite complete tree models are all and only the colored trees belonging L.*

Corollary 1.4.8 (Rabin [96]) *The MSO theory of the infinite complete k-ary tree is decidable.*

We can further modify the notion of tree automaton in order to allow computations over *non-complete* trees. To do that, it is sufficient to extend the input alphabet of the automaton with a fresh symbol $\perp \notin \Sigma$ and assume that, whenever $v \notin \mathcal{D}om(T)$, M reads $\perp$ on the vertex v.

1.5 Repetitions and ultimately periodic words

In this section we establish some fundamental properties of repeating patterns of finite and ultimately periodic words. In particular, we describe an efficient (linear-time) algorithm that computes the shortest repeating pattern of a given (finite or ultimately periodic word). To start with, we introduce the notions of period, partial period, and border.

Definition 1.5.1 *A finite (resp., infinite) word w has* a period p *if, for some $k > 0$ (resp., for $k = \omega$), we have $w = w[1, p]^k$.* The period *of w is its minimum period. If the period of w is equal to $|w|$ (namely, $w = w[1, p]^k$ implies $p = |w|$ and $k = 1$), then we say that w is* primitive. *By analogy, we say that p is* a partial period *of a finite word w if w is a prefix of $w[1, p]^\omega$. Finally, a* border *of a finite word w is a word $v \neq w$ such that v is both a prefix and a suffix of w.*

The following lemma relates distinct (partial) periods of words. It is a straightforward generalization of the well-known Fine-Wilf's Lemma [57].

Lemma 1.5.2 *For any finite non-empty word w, if p and q are partial periods of w and $|w| \geq p + q$, then $gcd(p, q)$ is a partial period of w as well.*

Proof. We prove the claim by induction on $p + q$. We assume that $p < q$ and we denote by r the value $q - p$. Since p and q are both partial periods of w, for every $i \in [1, |w| - q]$, we have $w[i] = w[i + q] = w[i + q - p] = w[i + r]$. Similarly, for every $i \in [|w| - q + 1, |w| - r]$, $w[i] = w[i - p]$ (since $|w| \geq p + q$) and $w[i - p] = w[i + q - p] = w[i + r]$ hold. Thus r is partial period of w. Since $p + r < p + q$ and $gcd(p, q) = gcd(p, r)$ hold, we conclude, by inductive hypothesis, that $gcd(p, q)$ is a partial period of w. ■

We now show that for every finite word w, the periods of all substrings of w can be efficiently computed in time $\Theta(|w|^2)$. The algorithm rests on noticeable properties of partial periods and borders and it is somehow related to the way Knuth, Morris, and Pratt define the prefix function of a word in the context of string-matching algorithms [70]. From now on, we shall use the abbreviations $v \dashv w$ and $v \dashv\!\!\dashv w$ to respectively say that 'v is a border of w' and 'v is the maximum border of w'. We start by showing distinctive properties of partial periods and borders. The following proposition establishes a correspondence between (maximum) borders and (minimum) partial periods.

Proposition 1.5.3 *Given a finite word w, $w[1, q]$ is a (maximum) border of w iff $|w| - q$ is a (minimum) partial period of w.*

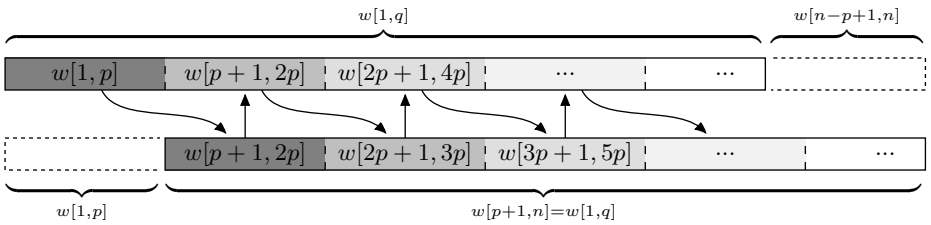

Figure 1.1: Relationship between partial periods and borders.

Proof. Let w be a finite word having length n and let q be a natural number such that $w[1, q] = w[n - q + 1, n]$. We define $p = n - q$ and we show, by induction on k, that for every $i \in [1, p]$, $kp + i \leq n$ implies $w[i] = w[kp + i]$ (see Figure 1.1). For $k = 0$, the property holds trivially. For $k > 0$ and $i \leq n - kp$, by inductive hypothesis, we have that $w[i] = w[(k - 1)p + i]$ and, since (i) $w[1, q]$ is a border of w, (ii) $kp+i \geq n-q+1$, and (iii) $(k-1)p+i \leq (k-1)p+n-kp = q$, $w[i] = w[(k-1)p+i] = w[(k - 1)p + i + (n - q)] = w[kp + i]$ holds. Hence, w has a partial period $p = n - q$. For the converse, let w be a finite word having length n and let p be a partial period of w. We know that for every $i \in [1, p]$, $kp + i \leq n$ implies $w[i] = w[kp + i]$. Now, by letting $q = n - p$, we have $w[1, q] = w[p + 1, p + q] = w[n - q + 1, n]$. Therefore, $w[1, q]$ is a border of w. Finally, the maximum border is the border having maximum length q, hence $p = n - q$ is the minimum partial period of w, and vice versa. ■

Let us now focus our attention on the computation of the maximum border of each prefix of a given word w. We preliminarily establish some interesting properties of the relations 'border of' and 'maximum border of'. These properties will allow us to devise an algorithm that computes the partial periods of all the prefixes of a given finite word w in linear time with respect to $|w|$. Taking advantage of such an algorithm, we shall be able to compute the partial periods of all substrings of w in time $\Theta(|w|^2)$ by simply iterating the computation on each suffix of w. Since w contains exactly $\frac{|w|(|w|+1)}{2}$ substrings, the resulting algorithm turns out to be asymptotically optimal. The following lemma determines the relation between the borders of a given word w and the borders of the extended word $w \cdot a$, by showing that $v \cdot a$ is a border of $w \cdot a$ only if v is a border of w.

Lemma 1.5.4 *The relation $\dashv$ respects the extension of words to the right, that is, $(v \cdot a) \dashv (w \cdot a)$ holds iff both $v \dashv w$ and $w[|v|+1] = a$ hold (see Figure 1.2).*

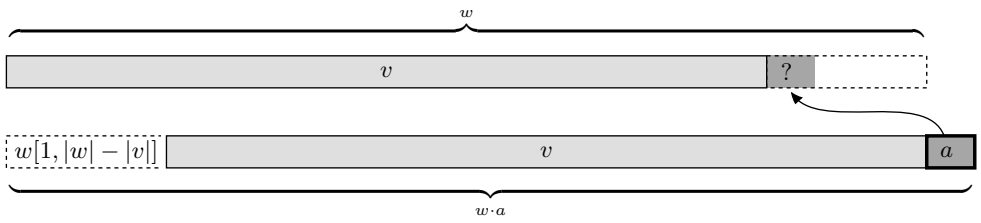

Figure 1.2: Right extensions of borders.

Proof. The proof is almost trivial. If $v \cdot a$ is a border of $w \cdot a$, then $v = w[1, |v|] = w[|w| - |v| + 1, |w|]$ and $w[|v| + 1] = a$ hold. For the converse, if v is a border of w and $w[|v| + 1] = a$ holds, then $v \cdot a = w[1, |v| + 1] = w[|w| - |v| + 1, |w|] \cdot a$ and hence $v \cdot a$ is a border of $w \cdot a$. □ ■

From Lemma 1.5.4, we can easily devise a dynamic-programming-oriented algorithm that computes all borders of all prefixes of a given word w: for each border $w[1, q]$ of some prefix $w[1, j]$, check whether $w[q + 1] = w[j + 1]$ (this suffices to establish whether $w[1, q+1]$ is a border of $w[1, j+1]$). From the lemma it follows that, given the borders of $w[1, j]$, one can easily compute the maximum border of $w[1, j+1]$. In fact, it is not necessary to store *all* borders of all prefixes in order to compute the *maximum* borders of all prefixes, as the following lemma shows (cf. Figure 1.3, where we depict the transitive reduction of a simple instance of the relation 'border of').

Lemma 1.5.5 *The relation $\dashv$ is linear to the left, that is, whenever $v \dashv w$ and $u \dashv w$ hold, then we have $v = u$ or $v \dashv u$ or $u \dashv v$.*

Proof. Let w, v, and u be three finite words of length n, m, and r, respectively. From $v \dashv w$, $v = w[1, m] = w[n - m + 1, n]$ follows, and from $u \dashv w$, $u = w[1, r] =$

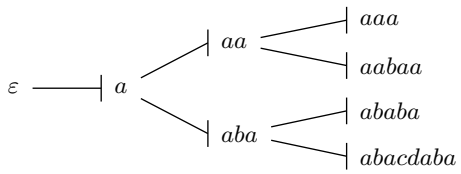

Figure 1.3: Left linearity of the relation 'border of'.

$w[n-r+1,n]$ follows. If $r < m$, then we have $u = w[1,r] = w[1,m][1,r] = v[1,r]$ and $u = w[n-r+1,n] = w[n-m+1,n][m-r+1,m] = v[m-r+1,m]$. Hence, u is a border of v. The other cases can be proved in a similar way. ■

From Lemma 1.5.5 it follows that, whenever v is a border of $w[1,j]$, then v is either the maximum border of $w[1,j]$ or a border of the maximum border of $w[1,j]$. Such a property can be exploited to prove the following corollary.

Corollary 1.5.6 *Let w be a finite word and let $w_1, ..., w_n$ be the (unique) sequence of finite words such that $\varepsilon = w_1 \dashv\!\dashv ... \dashv\!\dashv w_n \dashv\!\dashv w$. If $v \dashv w$, then there is $k \in [n]$ such that $v = w_k$.*

Proof. The proof is by induction on n. The case $n = 1$ is trivial, since $v \dashv w$ implies $v = \varepsilon = w_1$. For $n > 1$, we distinguish two cases: either v is a maximum border of w, or v is not a maximum border of w. In the former case, we simply let $k = n$. In the latter case, by Lemma 1.5.5, we know that there is at least one word u such that $v \dashv u$ and $u \dashv w$. Let u be the longest of such words, so that $u = w_n \dashv\!\dashv w$. From the inductive hypothesis, we immediately obtain $v = w_k$, for some $k \in [n]$. ■

The upshot of Lemma 1.5.4 and Corollary 1.5.6 is that, in order to compute the maximum border of $w[1,j+1]$, it is sufficient to recursively determine the maximum border of each proper prefix of $w[1,j+1]$ and then descend the chain of relations $\dashv\!\dashv$, searching for the longest (i.e., the first) border whose extension matches with $w[j+1]$. As an example, consider the chain of relations $\varepsilon \dashv\!\dashv a \dashv\!\dashv aba \dashv\!\dashv abacdaba$; the maximum border of the word $w = abacdaba \cdot b$ is $a \cdot b$, which is precisely the word obtained by extending with b the rightmost word v in the sequence ε, a, aba such that $w[|v|+1] = b$ (if any). The above argument is formally stated by the next theorem. Subsequently, we shall show that, even if some steps of the computation of a maximum border may take linear time, the total time needed to compute the maximum borders of all prefixes of w is still linear in $|w|$.

Theorem 1.5.7 *Let w be a finite word and let $w_1, ..., w_n$ be the (unique) sequence of finite words such that $\varepsilon = w_1 \dashv\!\dashv ... \dashv\!\dashv w_n \dashv\!\dashv w$. For any given v, the following two conditions are equivalent:*

1. *$(v \cdot a) \dashv\!\dashv (w \cdot a)$,*
2. *there exist $k \in [n]$ such that $w_k = v$, $w[|w_k|+1] = a$, and $w[|w_h|+1] \neq a$ for all $h > k$.*

Proof. As for the implication from *1* to *2*, if $v \cdot a$ is the maximum border of $w \cdot a$, then, by Lemma 1.5.4, v is a border of w. From Corollary 1.5.6, we know that $v = w_k$ holds for a suitable $k \in [n]$, and, again by Lemma 1.5.4, $w[|w_k|+1] = a$. Furthermore, for every $h > k$, w_h is longer than w_k and hence $w_h \cdot a$ cannot be a border of $w \cdot a$. Since w_h is a border of w and $w_k \cdot a$ is the maximum border of $w \cdot a$, this implies that $w[|w_h| + 1] \neq a$. Conversely, let k be the largest index such that $w[|w_k| + 1] = a$. Clearly $w_k \cdot a$ is a border of $w \cdot a$. If there would exist a border $v \cdot a$ of $w \cdot a$ with $|v \cdot a| > |w_k \cdot a|$, then, by Lemma 1.5.4, v would be a border of w and, by Corollary 1.5.6, there would be an integer h such that $w_h = v$. Moreover, from Lemma 1.5.5, we know that w_k is a border of v, and hence h should be greater that k. This implies that $w[|w_h|+1] = a$, which is against the hypothesis of k being the largest index such that $w[|w_k| + 1] = a$. ∎

Let us provide now an algorithm that computes the minimum partial period of each prefix $w[1, j]$ of a given finite word w. By denoting with $p(j)$ (resp., $q(j)$) the minimum partial period (resp., the length of the maximum border) of $w[1, j]$, the recurrence equations

$$q(1) = 0, \tag{1.5.1}$$

$$q(j+1) = max\big\{0, r+1 \,:\, w[r+1] = w[j+1] \ \wedge\ \exists\, i > 0.\ (r = q^i(j))\big\}, \tag{1.5.2}$$

follow directly from Theorem 1.5.7.
The algorithm *PartialPeriodsOfAllPrefixes* uses the above equations to compute $q(j)$ and $p(j)$, for every $j \in [|w|]$.

PartialPeriodsOfAllPrefixes(w)

```
1: n ← |w|
2: q(1) ← 0
3: for j = 2...n do
4:    r ← q(j − 1)
5:    while w[r + 1] ≠ w[j] and r > 0 do
6:       r ← q(r)
7:    end while
8:    if w[r + 1] = w[j] then
9:       q(j) ← r + 1
10:   else
11:      q(j) ← 0
12:   end if
13: end for
14: for j = 1...n do
15:   p(j) ← j − q(j)
16: end for
17: return (p(j))_{j∈[1,n]}
```

It remains to show that the execution of *PartialPeriodsOfAllPrefixes*(w) takes time linear in $n = |w|$. Note that $r = q(j-1)$ holds just before entering the 'while' loop at

lines 5–7. Furthermore, at lines 8–12, either $r+1$ or 0 is assigned to $q(j)$. Since the variable r decreases at least by 1 at each iteration of the inner loop, for each value of j the number of iterations is bounded by $q(j-1)-(q(j)-1)$. Hence, the computation of the length $q(j)$ of the maximum border of $w[1,j]$, with $j \in [2,n]$, takes time proportional to $q(j-1)-(q(j)-1)$. Therefore, the total time required to execute *PartialPeriodsOfAllPrefixes*(w) is proportional to $\sum_{j\in[2,n]}(q(j-1)-(q(j)-1)) = \Theta(n)$. Since there clearly exists a linear lower bound on the complexity of the problem solved by procedure *PartialPeriodsOfAllPrefixes*, the proposed algorithm is asymptotically optimal.

Finally, we provide an asymptotically optimal algorithm for computing the periods of all substrings of w. The algorithm rests on the following proposition, which connects periods to partial periods.

Proposition 1.5.8 *For every finite non-empty word w and for every positive natural number $p < |w|$, p is the minimum period of w iff p divides $|w|$ and it is the minimum partial period of w.*

Proof. The right-to-left implication is trivial. Conversely, if p is a period of w, then p is a partial period of w as well. Thus it remains to show that p is the minimum partial period of w, provided that p ($< |w|$) is the minimum period of w. Suppose that p ($< |w|$) is the minimum period of w. We have that p is a partial period of w. Now, if w had a partial period $q < p$, then $p + q < 2p \leq |w|$ and then, by Lemma 1.5.2, $gcd(p,q)$ would be another partial period of w. Since p divides $|w|$ and $gcd(p,q)$ divides p, $gcd(p,q)$ would be a period of $|w|$, and hence p would not be the minimum period of w. This is a contradiction and thus p must be the minimum partial period of w. ■

The following algorithm computes the periods of all substrings of a word w (we use $p(i,j)$ and $P(i,j)$ to denote, respectively, the minimum partial period and the minimum period of a substring $w[i,j]$).

PeriodsOfAllSubstrings(w)

```
 1: n ← |w|
 2: for i = 1...n do
 3:    (p(i,j))_{j∈[i,n]} ← PartialPeriodsOfAllPrefixes(w[i,n])
 4: end for
 5: for i = 1...n do
 6:    for j = 1...n do
 7:       if (j − i + 1) mod p(i,j) = 0 then
 8:          P(i,j) ← p(i,j)
 9:       else
10:          P(i,j) ← j − i + 1
11:       end if
12:    end for
13: end for
14: return (P(i,j))_{i∈[1,n],j∈[i,n]}
```

We conclude the section by defining ultimately periodic words and their canonical representations in terms of pairs consisting of a minimum prefix and a minimum repeating pattern.

Definition 1.5.9 *An* ultimately periodic word *is an infinite word of the form $w = u \cdot v^\omega$, where u and v are finite words and $v \neq \varepsilon$. The word u is said to be* a prefix *of w and the word v is said to be* a repeating pattern *of w. If u (resp., v) is the shortest prefix (resp., the shortest repeating pattern) of w, then u (resp., v) is said to be* the prefix *(resp.,* the repeating pattern*) of w.*

It is clear that any ultimately periodic word w can be finitely represented as a pair (u, v) of finite words such that $w = u \cdot v^\omega$. Note that, if u if *the* prefix of w and v is a repeating pattern of w, then $u[|u|] \neq v[|v|]$ (otherwise, $u[1, |u| - 1]$ would be a prefix of w shorter than u). Similarly, if v is *the* repeating pattern of w, then v is primitive (otherwise, there would exist a proper prefix $v[1, p]$ of v which is a repeating pattern of w). This yields a straightforward algorithm that computes the canonical (i.e., minimal and unique) representation of a given ultimately periodic word $w = u \cdot v^\omega$ in linear time with respect to the lengths of u and v.

CanonicalRepresentation(u, v)

1: $(p(j))_{j \in [1,|v|]} \leftarrow$ *PartialPeriodsOfAllPrefixes*(v))
2: **if** $|v| \bmod p(|v|) = 0$ **then**
3: $\quad q \leftarrow p(|v|)$
4: **else**
5: $\quad q \leftarrow |v|$
6: **end if**
7: $i \leftarrow 0$
8: $j \leftarrow 0$
9: **while** $i < |u|$ **and** $u[|u| - i] = v[q - j]$ **do**
10: $\quad i \leftarrow i + 1$
11: $\quad j \leftarrow (j + 1) \bmod q$
12: **end while**
13: $u' \leftarrow u[1, |u| - i]$
14: $v' \leftarrow v[q - i + 1, q] \cdot v[1, q - i]$
15: **return** (u', v')

1.6 Periodicity with respect to finite monoids

Here we define ultimately periodic functions with respect to finite monoids and we prove some relevant properties of them. Such a notion of function is found, with different but equivalent definitions, in many areas of computer science (e.g., group theory, combinatorics of words, and automata theory). Roughly speaking, ultimately periodic functions with respect to finite monoids are functions over the natural numbers that manifest a repeating pattern whenever projected into any finite monoid. Examples of such functions are i^2, 2^i, $2^i - i^2$, i^i, $i!$, and the exponential tower $2^{2^{\cdot^{\cdot^{2}}}}$.

Henceforth, for some given $p \geq 0$, $q > 0$, and $i \geq 0$, we denote by $[i]_{p,q}$ either the value i or the value $((i-p) \bmod q) + p$, depending on whether $i < p$ or $i \geq p$ holds.

Definition 1.6.1 *A function $f : \mathbb{N} \to \mathbb{N}$ is said to be* ultimately periodic with respect to finite monoids *if, given $k \geq 0$ and $r > 0$, one can compute $p \geq 0$ and $q > 0$ such that for every $i \geq 0$,*

$$\big[f(i)\big]_{k,r} = \big[f([i]_{p,q})\big]_{k,r}. \tag{1.6.1}$$

The reason for the name 'ultimately periodic function with respect to finite monoids', comes from the fact that every function f satisfying Equation 1.6.1 can be characterized in terms of the periodicity of the sequences of the form $(e^{f(i)})_{i\geq 0}$, where e is an element of a finite (multiplicative) monoid $(M, \cdot)$. This is formally stated in Proposition 1.6.2. As a matter of fact, such a characterization implies that the notion of ultimately periodic function with respect to finite monoids is in fact equivalent to (i) the notion of residually (or profinitely) ultimately periodic sequence (cf. [19, 20]), (ii) the notion of ultimately periodic function with respect to finite semigroups (cf. [119, 120]), and (iii) the notion of ultimately periodic function with respect to finite groupoids (cf. [82, 83, 84]).

Proposition 1.6.2 *A function $f : \mathbb{N} \to \mathbb{N}$ satisfies Equation 1.6.1 iff, given any finite (multiplicative) monoid* [4] *$(S, \cdot)$ and given any element $e \in S$, one can compute $p \geq 0$ and $q > 0$ such that, for all $i \geq 0$,*

$$e^{f(i)} = e^{f([i]_{p,q})},$$

namely, the sequence $(e^{f(i)})_{i\geq 0}$ is (effectively) ultimately periodic.

Proof. Let assume that f satisfies Equation 1.6.1 and fix an arbitrary pair of integers $k \geq 0$ and $r > 0$. Then, there are $p \geq 0$ and $q > 0$, computable from k and r, such that $\big[f(i)\big]_{k,r} = \big[f([i]_{p,q})\big]_{k,r}$ holds for all $i \geq 0$. Now, given any finite monoid $(S, \cdot)$ and any element $e \in S$, we know, from the Pigeonhole Principle, that there exist two positive integers $k \geq 0$ and $r > 0$, computable from $(S, \cdot)$ and e, such that $e^j = e^{[j]_{k,r}}$ holds for every $j \geq 0$. Thus, we have $e^{f(i)} = e^{[f(i)]_{k,r}} = e^{[f([i]_{p,q})]_{k,r}} = e^{f([i]_{p,q})}$.
For the converse implication, suppose that, given $f : \mathbb{N} \to \mathbb{N}$, for every finite monoid $(S, \cdot)$ and for every element $e \in M$, we can compute $p \geq 0$ and $q > 0$ such that, for all $i \geq 0$, $e^{f(i)} = e^{f([i]_{p,q})}$ holds. Let fix two integers $k \geq 0$ and $r > 0$. We can chose (compute) a finite monoid $(S, \cdot)$ and an element $e \in S$ such that r turns out to be the *least* integer such that e^r is the identity of S. Note that, by construction, for any pair of integers i and j, $a^i = a^j$ implies $i \bmod r = j \bmod r$, whence $[i]_{k,r} = [j]_{k,r}$. From the hypothesis on f, we can compute $p \geq 0$ and $q > 0$ such that for every $i \geq 0$, $e^{f(i)} = e^{f([i]_{p,q})}$ holds. Hence, for every $i \geq 0$, we have $[f(i)]_{k,r} = [f([i]_{p,q})]_{k,r}$. ∎

In the following, we describe a number of ways to obtain ultimately periodic functions with respect to finite monoids, starting from a set of basic functions. Such a result extends previous results from [103, 19, 20].

[4] A finite monoid is a finite set S equipped with a binary associative operation $\cdot$ and containing the identity element (i.e., the element e_1 such that for every $e \in S$, $e \cdot e_1 = e_1 \cdot e = e$).

Hereafter, we use $i = j \pmod m$ as a shorthand for $i \bmod m = j \bmod m$. We say that a function $f : \mathbb{N} \to \mathbb{N}$ has *unbounded infimum* if $lim\ inf_{i \to \infty} f(i) = \infty$. In such a case, we understand that, for any given $l \in \mathbb{N}$, we can compute i_0 such that $f(i) \geq l$ holds, for all $i \geq i_0$.

Proposition 1.6.3 *Let f and g be two ultimately periodic functions with respect to finite monoids. The following functions are also ultimately periodic with respect to finite monoids:*

1. **(Sum)** *$h = f + g$, defined by $h(i) = f(i) + g(i)$;*
2. **(Product)** *$h = f * g$, defined by $h(i) = f(i) * g(i)$;*
3. **(Difference)** *$h = f - g$, defined by $h(i) = f(i) - g(i)$, provided that h has unbounded infimum;*
4. **(Quotient)** *$h = \lfloor \frac{f}{d} \rfloor$, defined by $h(i) = \lfloor \frac{f(i)}{d} \rfloor$, where $d > 0$;*
5. **(Exponentiation)** *$h = f^g$, defined by $h(i) = (f(i))^{g(i)}$, provided that h has unbounded infimum;*
6. **(Exponential tower)** *h defined by $h(0) = 1$ and $h(i+1) = b^{h(i)}$, where $b > 0$;*
7. **(Fibonacci numbers)** *h defined by $h(0) = h(1) = 1$ and $h(i+2) = h(i)+h(i+1)$;*
8. **(Generalized sum)** *h defined by $h(i) = \sum_{j=0}^{i-1} f(j)$;*
9. **(Generalized product)** *h defined by $h(i) = \prod_{j=0}^{i-1} f(j)$;*
10. **(Substitution)** *$h = f \circ g$, defined by $h(i) = g(f(i))$.*

Proof. As for cases *1.* and *2.*, it suffices to note that the operator $[\ ]_{l,r}$ respects sums and products, namely, $[i + j]_{l,r} = [[i]_{l,r} + [j]_{l,r}]_{l,r}$ and $[i * j]_{l,r} = [[i]_{l,r} * [j]_{l,r}]_{l,r}$, for every $i, j \in \mathbb{N}$.

Similarly, one can show that $[\ ]_{l,r}$ respects differences, namely, $[i - j]_{l,r} = [[i]_{l,r} - [j]_{l,r}]_{l,r}$, provided that $i - j \geq l$. Thus, case *3.* follows immediately if we assume that $h = f - g$ has unbounded infimum.

Case *4.* is proved by noticing that for every $l \geq 0$ and $r > 0$, $[h(i)]_{l,r}$ is either 0 or $[[h(i - d)]_{l,r} + 1]_{l,r}$, depending on whether $i < d$ or $i \geq d$. Thus, by defining $(h_1(i), ..., h_d(i)) = ([h(i)]_{l,r}, ..., [h(i + d - 1)]_{l,r})$, we obtain

$$(h_1(i+1), ..., h_d(i+1)) = \begin{cases} (0, ..., 0) & \text{if } i = 0, \\ (h_2(i), ..., h_d(i), [h_1(i) + 1]_{l,r}) & \text{if } i > 0. \end{cases}$$

Since each value $h_j(i)$ ranges over the finite domain $\{0, ..., l + r - 1\}$, we can apply the Pigeonhole Principle and claim that there are two (computable) integers $p \geq 0$ and $q > 0$ such that $h_j(i) = h_j(i + q)$, for every $i \geq p$ and $j \in [d]$. This proves that $[h(i)]_{l,r} = [h([i]_{p,q})]_{l,r}$.

As for case *5.*, we preliminarily recall the definition of the 'Euler totient function' ϕ

$$\phi(i) = i \prod_{p \text{ prime dividing } i} \left(1 - \frac{1}{i}\right)$$

and the following two properties:

$$b^a = b^a + \sum_{i=1}^{a}\big(\tbinom{a}{i} b^{a-i} * 0\big) = b^a + \sum_{i=1}^{a}\big(\tbinom{a}{i} b^{a-i} * m^i\big) = (b+m)^a \pmod m,$$
$$b^a = b^a * 1 = b^a * b^{\phi(m)} = b^{a+\phi(m)} \pmod m.$$

Since f and g satisfy Equation 1.6.1, one can compute $p, p' \geq 0$ and $q, q' \geq 0$ such that $[f(i)]_{0,m} = [f([i]_{p,q})]_{0,m}$ and $[g(i)]_{0,\phi(m)} = [g([i]_{p',q'})]_{0,\phi(m)}$. Now, by letting $r = max(p, p')$ and $s = lcm(q, q')$, 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 $l \geq 0$. Let $\sigma(l)$ be the least integer i such that $i \geq r$ and $f(i)^{g(i)} \geq l$ (such a value exists and it is computable by hypothesis). Then, for every $i \geq \sigma(l)$, we have

$$\begin{aligned}\Big[f\,(i)^{g(i)}\Big]_{l,m} &= \Big[f\,(i)^{g(i)} - l\Big]_{0,m} + l = \Big[\big[f\,(i)^{g(i)}\big]_{0,m} - l\Big]_{0,m} + l \\ &= \Big[\big[f\,\big([i]_{\sigma(l),s}\big)^{g([i]_{\sigma(l),s})}\big]_{0,m} - l\Big]_{0,m} + l = \Big[f\big([i]_{\sigma(l),s}\big)^{g([i]_{\sigma(l),s})}\Big]_{l,m}.\end{aligned}$$

This proves that $h = f^g$ satisfies Equation 1.6.1.
We now prove case *6.*. We have $h(0) = 1$ and $h(i) = b^{h(i-1)}$, for every $i > 0$. Let $r > 0$ and, for every $l \geq 0$, let $\sigma(l) = \lceil \log_b(l) \rceil$. We prove, by induction on j, that for every $0 \leq j \leq r$, every $l \geq 0$, and every $i \geq \sigma(l)$, the following equation holds

$$\big[h(i+j)\big]_{l,\phi^{r-j}(r)} = \big[h(i+j+1)\big]_{l,\phi^{r-j}(r)}.$$

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 $i \geq \sigma(l)$. Now, let $j > 0$. For every $l \geq 0$ and for every $i \geq \sigma(l)$, we have

$$\begin{aligned}\Big[h(i+j)\Big]_{l,\phi^{r-j}(r)} &= \Big[b^{h(i+(j-1))}\Big]_{l,\phi^{r-j}(r)} = \Big[b^{[h(i+(j-1))]_{\sigma(l),\phi^{r-(j-1)}(r)}}\Big]_{l,\phi^{r-j}(r)} \\ &= \Big[b^{[h(i+(j-1)+1)]_{\sigma(l),\phi^{r-(j-1)}(r)}}\Big]_{l,\phi^{r-j}(r)} \\ &= \Big[b^{h(i+(j-1)+1)}\Big]_{l,\phi^{r-j}(r)} = \Big[h(i+j+1)\Big]_{l,\phi^{r-j}(r)}.\end{aligned}$$

In particular, by letting $j = r$, we have that for every $l \geq 0$ and every $i \geq \sigma(l)$,

$$\big[h(i+r)\big]_{l,r} = \big[h(i+r+1)\big]_{l,r}.$$

Therefore, we can conclude that $[h(i)]_{l,r} = [h([i]_{\sigma(l)+r,1}]_{l,r}$ for every $l \geq 0$, $r > 0$, and $i \geq 0$.

In case *7.*, we have $[h(i)]_{l,r} = [[h(i-2)]_{l,r} + [h(i-1)]_{l,r}]_{l,r}$ whenever $i \geq 2$. By defining $(h_1(i), h_2(i)) = ([h(i)]_{l,r}, [h(i+1)]_{l,r})$, we have

$$(h_1(i+1), h_2(i+1)) = \begin{cases} ([1]_{l,r}, [1]_{l,r}) & \text{if } i = 0, \\ (h_2(i), [h_1(i) + h_2(i)]_{l,r}) & \text{if } i > 0. \end{cases}$$

Since the values $h_1(i)$ and $h_2(i)$ range over the finite domain $\{0, ..., l + r - 1\}$, we can apply the Pigeonhole Principle and claim that there are two (computable) integers $p \geq 0$ and $q > 0$ such that $(h_1(i), h_2(i)) = (h_1(i+q), h_2(i+q))$, for every $i \geq p$. This implies that $[h(i)]_{l,r} = [h([i]_{p,q})]_{l,r}$.

We now prove case *8.* (case *9.* follows similarly). Given a function f satisfying Equation 1.6.1, we fix two integers l, r and we denote by p, q the (computable) integers such that $[f(i)]_{l,r} = [f([i]_{p,q})]_{l,r}$. We further define $S = \sum_{j=p+1}^{p+q} f(j)$ and we notice that $\left[\sum_{j=p+1}^{p+nq} f(j)\right]_{l,r} = \left[nS\right]_{l,r}$ holds for every integer $n \geq 0$. From the Pigeonhole Principle, there are suitable integers p', q' such that $[p'nS]_{l,r} = [(p'+q')nS]_{l,r}$. Moreover, for every $i \geq p + qq'$, one can compute an integer n_i such that $i = [i]_{p,qq'} + n_i qq'$. Thus, for every $i \geq p + qq'$, we obtain

$$\begin{aligned} \Big[h(i)\Big]_{l,r} &= \Big[\sum_{j=0}^{i} f(j)\Big]_{l,r} = \Big[\sum_{j=0}^{[i]_{p,qq'}} f(j) + \sum_{j=[n]_{p,qq'}+1}^{i} f(j)\Big]_{l,r} \\ &= \Big[\sum_{j=0}^{[i]_{p,qq'}} f(j) + q' n_i S\Big]_{l,r} = \Big[\sum_{j=0}^{[i]_{p,qq'}} f(j) + q'[n_i]_{p',q'} S\Big]_{l,r} = \Big[h([i]_{p,qq'})\Big]_{l,r}. \end{aligned}$$

It remains to show case *10.*. This is immediately proved by noticing that, given $l \geq 0$ and $r > 0$, one can compute $p, p' \geq 0$ and $q, q' > 0$ satisfying

$$\big[g(f(i))\big]_{l,r} = \big[g([f(i)]_{p,q})\big]_{l,r} = \big[g([f([i]_{p',q'})]_{p,q})\big]_{l,r} = \big[g(f([i]_{p',q'}))\big]_{l,r}.$$

■

2

Sequential Automata and Time Granularities

Different approaches to time granularity have been proposed in the literature, based on algebraic, logical, string-based, and automaton-based formalisms. In this chapter, the automaton-based approach to time granularity is explored in full detail. We review the notion of single-string (sequential) automaton (cf. [39]), we consider extensions with counters in order to make representations of granularities more compact, we show that single-string automata (possibly extended with counters) are as expressive as the formalism of Calendar Algebra and Wijsen's string-based models, and we provide effective solutions to the problems of equivalence, granule conversion, and optimization of time granularity representations. The decidability of the equivalence problem implies the possibility of effectively testing the semantic equivalence of two different granularity specifications, thus making it possible to use smaller, or more tractable, representations in place of bigger, or less tractable, ones. The granule conversion problem is the problem of relating the granules of a given granularity to those of another one. The relevance of such a problem has been pointed out by several authors (see [8, 90, 106, 49, 73]). As regards optimization problems, we identify two possible ways of optimizing automaton-based representations. According to the first one, optimizing means computing the smallest representation of a given time granularity; according to the second one, optimizing means computing the most tractable representation of a given granularity, that is, the one on which crucial algorithms (e.g., conversion algorithms) run fastest. These two criteria are clearly not equivalent, since the smallest representation is not necessarily the most tractable one, and vice versa. Furthermore, it is worth mentioning that, in the case of automata extended with counters, both problems yield non-unique solutions. We then focus on the problem of representing and reasoning on (possibly infinite) sets of periodical time granularities. In this respect, we define a proper subclass of Büchi automata, recognizing exactly the regular ω-languages consisting of ultimately periodic words, and we provide efficient solutions to several paradigmatic problems in automata theory, precisely, the emptiness problem, the membership problem, the equivalence problem, and the size-optimization problem. We also deal with the granularity comparison problem, namely, the problem of establishing, given two sets of granularities $\mathcal{G}$ and

$\mathcal{H}$, whether there exist $G \in \mathcal{G}$ and $H \in \mathcal{H}$ such that $G \sim H$, where $\sim$ is one of the commonly-used relations between granularities (e.g, partition, group, sub-granularity, aligned refinement, etc.).

2.1 Background knowledge

A number of formalisms to systematically deal with time granularities have been proposed in the literature. Most of them follow the set-theoretic approach to time granularity originally outlined by Clifford and Rao [28] as a suitable way of structuring information with a temporal dimension, independently of any particular calendric system. The proposed temporal structure consists of a finite, totally ordered set of temporal domains built upon some base, discrete, totally ordered, infinite set, whose elements are the smallest observable/interesting time units, through suitable intervallic partitions. Such an approach has been later extended and refined by several authors (a comprehensive survey can be found in [8]).

The most significant contributions are the formalism of collection expressions [72], that of slice expressions [89], and Calendar Algebra [90]. All these formalisms make it possible to express time granularities of practical interest, including infinite periodic granularities. They are based on the same idea: one can represent time granularities by means of symbolic terms built up from a finite set of basic granularities using a finite set of operators. The different sets of algebraic operators provided by the three formal systems and their relationships are investigated in [7], where it is proved that Calendar Algebra actually subsumes the other two systems. A common limitation of all these formalisms is that they focus on expressiveness issues, and almost completely ignore some basic problems of obvious theoretical and practical importance [58].

An approach that uses linear temporal logic (LTL) to represent and to reason about time granularities has been advocated by Combi et al. [30]. Time granularities are defined as discrete linear temporal structures, properly labeled with propositional symbols marking the starting and ending points of granules. The models of LTL-formulas are then used to represent possibly infinite sets of time granularities. The expressiveness of LTL makes it possible to capture a large set of regular granularities, such as, for instance, repeating patterns that can start at an arbitrary temporal point (unanchored granularities). 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 [104, 109].

A somehow similar, but less expressive, framework has been developed by Ohlbach and Gabbay in [92, 91], where a propositional logic extended with time interval modalities (i.e., $[2000, \mathit{year}]$, $\langle 2000, \mathit{year} \rangle$) is used to specify temporal relationships between events. The resulting logic, called Calendar Logic, is then translated into propositional logic, thus showing that the problem of checking consistency of granularity specifications is decidable. However, 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.

Recently, in [48], Demri proposed an original logical framework for dealing with integer periodicity constraints and, in particular, with time granularities. His approach is based on a fragment of Presburger LTL, denoted PLTL$^{\mathrm{mod}}$, which is obtained by extending LTL with past-time operators and by letting atoms to be formulas from a suitable first-order constraint language (a strict fragment of Presburger arithmetic). Such a combination of logical languages provides the capability of expressing quantitative temporal constraints as well as to compactly represent periodicity constraints for time granularities. Moreover, like plain LTL (but differently from full Presburger LTL, which is known to be highly undecidable), PLTL$^{\mathrm{mod}}$ is shown to enjoy 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 [39]) turn out to be in PSPACE.

Another approach to the representation and manipulation of time granularities has also been proposed by Wijsen in [118]. According to such an approach, time granularities are modeled as infinite strings over a suitable finite alphabet. Whenever a granularity is (ultimately) periodic, it can be finitely represented as an ordered pair, whose first element is a finite prefix and the second element is a finite repeating pattern. The resulting string-based model was then used to formally state and solve some important problems that have not been addressed satisfactorily by other formalisms, such as the equivalence problem and the minimization problem (given a granularity specification, find the shortest equivalent specification).

An original automaton-based approach that revises and extends Wijsen's one has been proposed by Dal Lago and Montanari in [39]. The resulting framework views granularities as strings generated by a specific class of sequential automata, called single-string automata (SSA), thus making it possible to (re)use well-known results from automata theory for dealing with time granularities. The authors also considered the possibility of extending SSA with counters to exploit regularities of modeled granularities, so as to make the structure of SSA more compact as well as to deal with those granularities which have a quasi-periodic structure.

2.2 A framework for time granularities

Different definitions of time granularity have been proposed in the literature. All these definitions use partitions of a fixed temporal domain to represent temporal structures. In this section we give a formal definition of time granularity capturing a reasonably large class of temporal structures. Then, we specialize the notion of granularity in order to allow finite representation and to ease manipulation of associated data.

Definition 2.2.1 *Given a set T of temporal instants and a total order $<$ on T, a* labeled time granularity *on the domain $(T, <)$ is a function $G : \mathbb{Z} \to \mathscr{P}(T)$ such that for every pair of integers $x < y$ and for every $t_x \in G(x)$ and $t_y \in G(y)$, $t_x < t_y$ holds.*

Each non-empty set $G(x)$, with $x \in \mathbb{Z}$, is said to be a *granule* and the set $\{x \in \mathbb{Z} : G(x) \neq \emptyset\}$ is called the *label set* of G. A labeled time granularity G *covers* the temporal instant t if there is a label x such that $t \in G(x)$. Note that Definition 2.2.1 captures a large set of time granularities over any kind of linear temporal domain (e.g., finite, infinitely countable, or uncountable). Such granularities can either cover the entire temporal domain (for instance, `Day`, `Week`, `Month`), or have gaps within and between granules (for instance, `BusinessDay`, `BusinessWeek`, `BusinessMonth`). Figure 2.1 depicts some of these granularities (the temporal domain is represented by a straight line, some segments of which are continuous, some others are dashed, depending on whether the corresponding time points are covered or not by some granule).

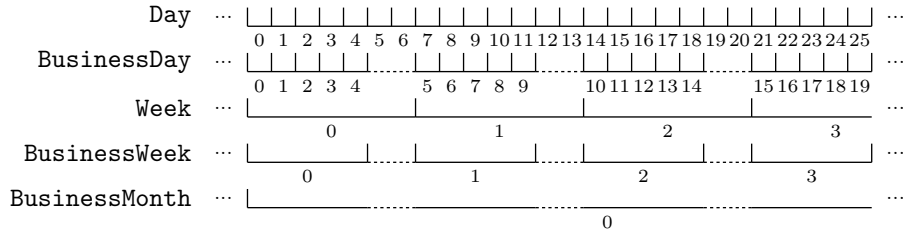

Figure 2.1: Some examples of labeled time granularities.

According the Definition 2.2.1, a time granularity is a labeled partition of a subset of a temporal domain containing countably many non-overlapping equivalence classes. In this sense, the definition of time granularity is similar to the one proposed by Bettini et al. [8, 90]. However, other definitions of time granularity have been adopted in the literature; for instance, the notion of *full-integer labeled granularity*, which forces the label set to be $\mathbb{Z}$, the notion of *simple granularity*, which assumes that the labels of the granules are contiguous (namely, whenever $G(i), G(j) \neq \emptyset$ and $i < k < j$, then $G(k) \neq \emptyset$ follows), and the notion of *unlabeled time granularity* [118], where granules are devoid of their labels (hence an unlabeled time granularity is a partition G of a subset T' of the temporal domain T such that for every pair of sets $g, g' \in G$, either $\forall\, t \in g.\, \forall\, t' \in g'.\, (t < t')$ or $\forall\, t \in g.\, \forall\, t' \in g'.\, (t' < t)$ holds). Besides slight variations on the notion of time granularity, several different hypothesis on the structure of the underlying temporal domain can be adopted. In particular, the choice of the temporal domain is typically between dense (e.g., $(\mathbb{Q}, <)$) and discrete (e.g., $(\mathbb{Z}, <)$). For instance, the former option is well suited if one is interested in representing arbitrarily fine granularities, while the latter one is more appropriate if the application under consideration requires a finite system of granularities which constitute coarser abstractions of a fixed bottom granularity. Another direction along which one can specialize the notion of temporal domain is that of choosing either a finite, semi-infinite, or bi-infinite linear structure. Clearly, the last option is the more general one and makes it possible to avoid the introduction of distinguished initial and final granules and to arbitrarily move backward and forward with respect to the reference granule. On the other hand, one may argue that in many practical situations the

second option (and sometimes also the first one) is sufficient, since we can assume the existence of an initial date (resp., the existence of an initial date and a final date far ahead in the future).

In the following we briefly summarize the most relevant relationships between time granularities.

Group into

A granularity G *groups into* a granularity H (equivalently, H *groups* G or $G \trianglelefteq H$), if for every integer x, there is a (possibly infinite) set $S_x \subseteq \mathbb{Z}$ such that $H(x) = \bigcup_{y \in S_x} G(y)$. Figure 2.2 represents two granularities G and H such that $G \trianglelefteq H$. Note that the granule $G(3)$ and the granule $G(5)$ are not included in any granule of H (indeed, the set S_z could be a non-convex interval of $\mathbb{Z}$).

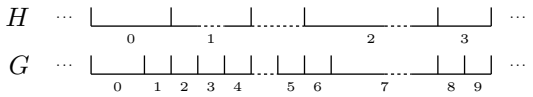

Figure 2.2: An instance of 'group into' relation.

Finer than

A granularity G *is finer than* a granularity H (or $G \preceq H$), if for every integer x, there is an integer y such that $G(x) \subseteq H(y)$. Figure 2.3 represents two granularities G and H such that $G \preceq H$. Note that, in this case, every granule of G must be included in a granule of H, while there may exist some granules of H (e.g., $H(0)$, $H(1)$, and $H(3)$) that contain temporal instants not covered by G.

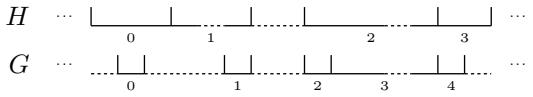

Figure 2.3: An instance of 'finer than' relation.

Label-aligned refinement

A granularity G *is a label-aligned refinement* of a granularity H, if for every integer x, $G(x) \subseteq H(x)$. Figure 2.4 represents two granularities G and H such that G is a label-aligned refinement of H.

Partition

If both $G \trianglelefteq H$ and $G \preceq H$ hold for two given granularities G and H, then we say that G *partitions* H. Figure 2.5 represents two granularities G and H such that $G \trianglelefteq H$ and $G \preceq H$.

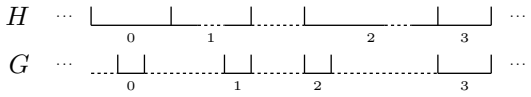

Figure 2.4: An instance of the 'label-aligned refinement' relation.

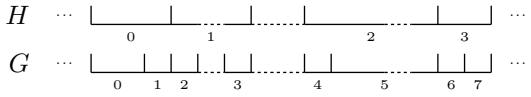

Figure 2.5: An instance of 'partition' relation.

Group periodically into

A granularity G *groups periodically into* H (equivalently, H *periodically groups* G) if G groups into H and there are two positive integers r and p, with r being less than the number of the granules of H, such that for every integer x and for every sequence of labels $y_0, y_1, ..., y_{n-1}$, whenever $H(x) = \bigcup_{0 \leq i < n} G(y_i)$ and $H(x + r) \neq \emptyset$ hold, $H(x + r) = \bigcup_{0 \leq i < n} G(y_i + p)$ follows. The definition of the relation 'group periodically into' may appear involved, but it is actually quite simple: it ensures that G groups into H and, whenever $H(x + r)$ is not empty, then $H(x + r)$ groups the granules of G in the same way (i.e., with the same 'pattern') as $H(x)$ does. Figure 2.6 represents two granularities G and H such that G groups periodically into H (with parameters $r = 2$ and $p = 7$).

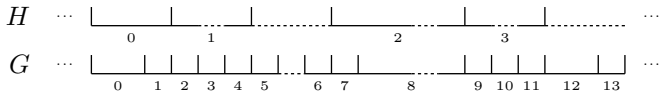

Figure 2.6: An instance of 'group periodically into' relation.

Sub-granularity

A granularity G is a *sub-granularity* of H (or $G \sqsubseteq H$), if, for every granule $G(x)$, there is a label y such that $G(x) = H(y)$. Figure 2.7 represents two granularities G and H such that $G \sqsubseteq H$.

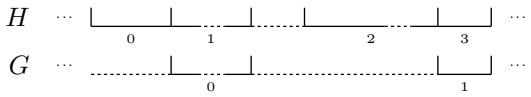

Figure 2.7: An instance of 'sub-granularity' relation.

Label-aligned sub-granularity

A granularity G is a *label-aligned sub-granularity* of H if, for every integer x, $G(x) \neq \emptyset$ implies $G(x) = H(x)$. Figure 2.8 represents two granularities G and H such that G is a label-aligned sub-granularity of H.

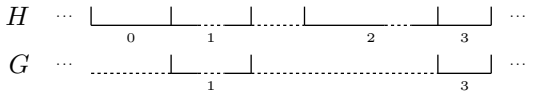

Figure 2.8: An instance of 'label-aligned sub-granularity' relation.

2.3 The Calendar Algebra

Given an infinite temporal domain, the set of all functions that satisfy Definition 2.2.1 is clearly uncountable. As a consequence, it is not possible to deal with all time granularities by means of a finitary formalism. In the following, we explain how the problem of providing a representation for time granularities can be tackled by restricting to structures resulting from the application of a finite set of simple transformations, starting from some known basic granularities. The idea underlying the design of such a symbolic formalism is the observation that temporal structures contained in most granularity systems are not isolated, but closely related. As a matter of fact, such an idea underlies several natural formalisms for representing time granularities (cf. [8, 72, 89]). In this section we focus our attention on a well-known formalism, called Calendar Algebra [8, 90], which allow one to represent a broad class of granularities in a natural and compact way. Calendar Algebra uses symbolic operations, called *calendar operations*, which capture significant relationships between time granularities. Calendar operations, once they are applied to some known granularities, form symbolic expressions representing new granularities in terms of existing ones. As an example, the granularity `Week` can be generated by applying the operation $Group_7$ to the granularity of days. Since these operations reflect the ways people define new granularities from existing ones, Calendar Algebra turns out to be a natural and intuitive way to represent user-defined granularities. Moreover, granularity representations are typically built, directly or indirectly, from a fixed single *bottom granularity* (e.g., `Second`, `Hour`, `Day`, depending on the accuracy required in the application context).

In the sequel, we formally define the calendar operations. We distinguish between two kinds of operations: *grouping-oriented* and *granule-oriented* ones. Grouping-oriented operations combine granules of the operands to form coarser granularities, while granule-oriented operations generate new granularities by selecting granules according to certain relationships.

2.3.1 Grouping-oriented operations

Grouping operation

For every full-integer labeled granularity G and for every positive natural number m, the expression $Group_m(G)$ represents the (full-integer labeled) granularity H such that for every $y \in \mathbb{Z}$,

$$H(y) = \bigcup_{my \leq x < m(y+1)} G(x).$$

For instance, `Week` is represented by $Group_7$(`Day`) (as shown in Figure 2.9), assuming that the day labeled with 0 starts a week.

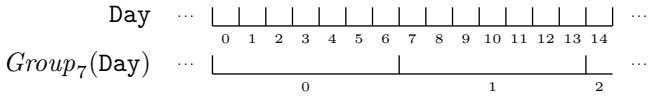

Figure 2.9: The grouping operation.

Altering-tick operation

Let fix two full-integer labeled granularities G and H such that G partitions H. We preliminarily define the *separator function* for G and H: we denote by $sep_{G,H}$ the function that maps a label $y \in \mathbb{Z}$ to the label of the first granule of G that is included in $H(y)$ (note that such a granule always exists). Clearly, for every $y \in \mathbb{Z}$, we have

$$H(y) = \bigcup_{sep_{G,H}(y) \leq x < sep_{G,H}(y+1)} G(x).$$

Now, let $l, k, m \in \mathbb{N}$, with $m > 0$. The expression $Alter^m_{l,k}(G, H)$ represents the (full-integer labeled) granularity J such that for every $y \in \mathbb{Z}$,

$$J(y) = \bigcup_{sep'_{G,H}(y) \leq x < sep'_{G,H}(y+1)} G(x),$$

where $sep'_{G,H}$ is the function defined by $sep'_{G,H}(y) = sep_{G,H}(y) + k\lfloor \frac{max(0,y-l)}{m} \rfloor$. Intuitively, the goal of the altering-tick operation is to modify H in such a way that the $l+1$-th granule in each group of m granules of H has k additional granules (or $-k$ fewer granules, if $k < 0$) of G (recall that G partitions H). An extension of such an operation is also used: when the parameter m is ω, the operation $Alter^m_{l,k}(G, H)$ expands or shrinks (depending on the value of k) the single granule $H(l)$. As an example, Figure 2.10 represents the granularity generated by $Alter^2_{0,-3}$(`Day`, `Week`). The following is a more interesting example. Let `31Days` be a granularity having intervals of 31-days as granules and suppose we want to shrink by 3 days the second granule in each group of 12 granules; then we may perform $Alter^{12}_{1,-3}$(`Day`, `31Days`).

We can iterate such a construction, thus obtaining a representation of the granularity `Month` of the Gregorian calendar:

$$\begin{aligned}\texttt{PseudoMonth} &= \mathit{Alter}^{12}_{10,-1}(\texttt{Day}, \mathit{Alter}^{12}_{8,-1}(\texttt{Day}, \mathit{Alter}^{12}_{5,-1}(\texttt{Day},\\ &\quad \mathit{Alter}^{12}_{3,-1}(\texttt{Day}, \mathit{Alter}^{12}_{1,-3}(\texttt{Day}, \texttt{31Days})))))\\ \texttt{Month} &= \mathit{Alter}^{12*400}_{1+12*399,1}(\texttt{Day}, \mathit{Alter}^{12*100}_{1+12*399,-1}(\texttt{Day},\\ &\quad \mathit{Alter}^{12*4}_{1+12*3,1}(\texttt{Day}, \texttt{PseudoMonth})))\end{aligned}$$

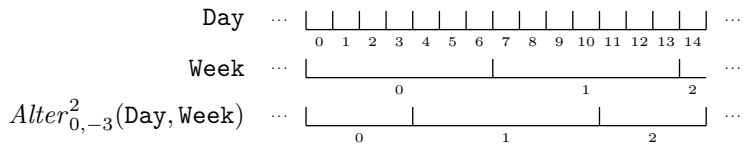

Figure 2.10: The altering-tick operation.

Shifting operation

For every full-integer labeled granularity G and for every $m \in \mathbb{Z}$, the expression $\mathit{Shift}_m(G)$ represents the (full-integer labeled) granularity H such that for every $x \in \mathbb{Z}$, $H(x) = G(x + m)$. Thus, the shifting operation generates a new granularity from a given one by simply shifting the labels of the granules.
For instance, since the hours of US Pacific Time are 3 hours later than those of US Eastern Time, one can represent the granularity of hours of US Pacific Time by performing $\mathit{Shift}_{-3}(\texttt{USEastHour})$, where `USEastHour` stands for the granularity of hours of US Eastern Time.

Combining operation

Let G and H be two granularities. The expression $\mathit{Combine}(G, H)$ denotes the granularity J such that for every $x \in \mathbb{Z}$,

$$J(y) = \bigcup_{x \text{ such that } G(x) \subseteq H(y)} G(x).$$

Intuitively, the operation $\mathit{Combine}(G, H)$ generates the granularity J by grouping all granules of G that are included in some granule of H. For instance, $\texttt{BusinessWeek} = \mathit{Combine}(\texttt{BusinessDay}, \texttt{Week})$ (see Figure 2.11).

Anchored grouping operation

We first define the *successor function* for a given granularity G: we denote by succ_G the function that maps a label $y \in \mathbb{Z}$ of a granule of G to the label of the first granule

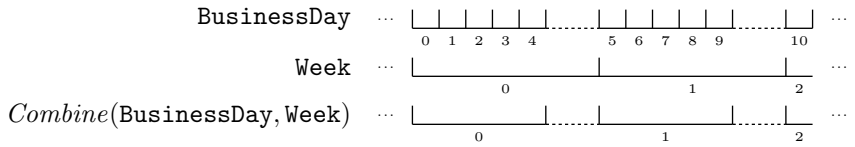

Figure 2.11: The combining operation.

of G that follows $G(y)$ (if such a granule does not exist, then the function returns ω); formally, we have $succ_G(y) = min\{x : x = \omega \vee (x \in \mathbb{Z} \wedge x > y \wedge G(x) \neq \emptyset)\}$. Now, let G and H be two granularities such that G is a label-aligned sub-granularity of H and H is a full-integer labeled granularity. The expression $AnchoredGroup(G, H)$ represents the granularity J such that for every $y \in \mathbb{Z}$,

$$J(y) = \begin{cases} \bigcup_{y \leq x < succ_G(y)} H(x) & \text{if } G(y) \neq \emptyset, \\ \emptyset & \text{otherwise.} \end{cases}$$

As an example, suppose that each fiscal year at a company begins in October and ends in the next September; then the granularity corresponding to these fiscal years is represented by $\texttt{FiscalYear} = AnchoredGroup(\texttt{Month}, \texttt{October})$.

2.3.2 Granule-oriented operations

Subset operation

For every granularity G and for every $m, n \in \mathbb{Z}$ such that $m < n$, the operation $Subset_m^n(G)$ generates a new granularity H such that for every $x \in \mathbb{Z}$,

$$H(x) = \begin{cases} G(x) & \text{if } m \leq x \leq n, \\ \emptyset & \text{otherwise.} \end{cases}$$

We further allow m to be $-\omega$ and n to be ω, with the obvious meaning. As an example, the granularity of years after the 20-th century is represented by $Subset_{2001}^{\omega}(\texttt{Year})$.

Selection operations

The selection operations are *SelectUp*, *SelectDown*, and *SelectByIntersect*, each one generating a new granularity by selecting granules from the first operand according to a designated relation that must be satisfied by the granules of the second operand. For every pair of granularities G and H, the operation $SelectUp(G, H)$ generates the granularity J such that, for every $x \in \mathbb{Z}$,

$$J(x) = \begin{cases} G(x) & \text{if } \exists\, y \in \mathbb{Z}.\ H(y) \subseteq G(x), \\ \emptyset & \text{otherwise.} \end{cases}$$

Intuitively, J is obtained by selecting those granules of G that contain one or more granules of H. For instance, given the granularity `Thanksgiving` of the fourth Thursdays of all Novembers, *SelectUp*(`Week`, `Thanksgiving`) represents those weeks that contain Thanksgiving days.
In order to simplify the description for the last two operations, we introduce the following notation: for every finite set $S = \{x_1 < x_2 < x_3 < ... < x_n\}$ of integers and for every pair of positive natural numbers k, l, we define $\Delta_k^l(S)$ as the subset $\{x_i : k \leq i < k + l\}$ of S, consisting of the first l integers following the $k-1$-th one. It is also possible to extend the semantics of the operator Δ_k^l by allowing the parameter k to be a negative value; in this case, we assume that $\Delta_k^l(S) = \{x_i : n + k - l + 2 \leq i < n + k\}$. For instance, $\Delta_3^2(\{1,2,3,4,5,6,7\}) = \{3,4\}$ and $\Delta_{-3}^2(\{1,2,3,4,5,6,7\}) = \{4,5\}$.
For every pair of granularities G and H and for every $k \neq 0$ and $l > 0$, the expression $\mathit{SelectDown}_k^l(G,H)$ represents the granularity J such that for every $x \in \mathbb{Z}$,

$$J(x) = \begin{cases} G(x) & \text{if } \exists\, y \in \mathbb{Z}.\ x \in \Delta_k^l(\{z \in \mathbb{Z} : \emptyset \neq G(z) \subseteq H(y)\}), \\ \emptyset & \text{otherwise.} \end{cases}$$

Intuitively, the operation $\mathit{SelectDown}_k^l(G,H)$ generates a new granularity J by selecting l granules, starting from the k-th one, in each set of granules of G that are contained in one granule of H. For instance, `Monday` $= \mathit{SelectDown}_1^1$(`Day`, `Week`), `Tuesday` $= \mathit{SelectDown}_2^1$(`Day`, `Week`), etc.
For every pair of granularities G and H and for every $k \neq 0$ and $l > 0$, the expression $\mathit{SelectByIntersect}_k^l(G,H)$ represents the granularity J such that for every $x \in \mathbb{Z}$,

$$J(x) = \begin{cases} G(x) & \text{if } \exists\, y \in \mathbb{Z}.\ x \in \Delta_k^l(\{z \in \mathbb{Z} : G(z) \cap H(y) \neq \emptyset\}), \\ \emptyset & \text{otherwise.} \end{cases}$$

Intuitively, the operation $\mathit{SelectByIntersect}_k^l(G,H)$ generates a new granularity J by selecting l granules, starting from the k-th one, in each set of granules of G that intersect one granule of H. For instance, `FirstWeekOfMonth` $= \mathit{SelectByIntersect}_1^1$(`Week`, `Month`).

Set operations

We introduce a last series of operations, which come from viewing granularities as sets of granules. In order to guarantee that the result of a set operation is always a valid granularity, we require that set operations are defined on two granularities G and H only if there exists a granularity I such that both G and H are label-aligned sub-granularities of I.
The union operation $G \cup H$ generates the granularity J such that for every $x \in \mathbb{Z}$,

$$J(x) = \begin{cases} G(x) & \text{if } G(x) \neq \emptyset, \\ H(x) & \text{if } G(x) = \emptyset \text{ and } H(x) \neq \emptyset, \\ \emptyset & \text{otherwise.} \end{cases}$$

For instance, $\texttt{WeekendDay} = \texttt{Saturday} \cup \texttt{Sunday}$.
The intersection operation $G \cap H$ generates the granularity J such that for every $x \in \mathbb{Z}$,

$$J(z) = \begin{cases} G(z) & \text{if } G(z), H(z) \neq \emptyset, \\ \emptyset & \text{otherwise.} \end{cases}$$

For instance, given the granularity `Monday` and the granularity `FullMoonDay`, which includes the days when there is a full moon, the granularity of the Mondays in which there is a full moon can be generated by the expression $\texttt{Monday} \cap \texttt{FullMoonDay}$.
The difference operation $G - H$ generates the granularity J such that for every $x \in \mathbb{Z}$,

$$J(x) = \begin{cases} G(x) & \text{if } G(x) \neq \emptyset \text{ and } H(x) = \emptyset, \\ \emptyset & \text{otherwise.} \end{cases}$$

For instance, $\texttt{BusinessDay} = \texttt{Day} - \texttt{WeekendDay} - \texttt{FederalHoliday}$.

2.3.3 Accommodating conditions on calendar operations

Some calendar operations are defined under the proviso that suitable relationships between the operands hold. For instance, set operations are only applicable to pairs of granularities that are label-aligned sub-granularities of a common one. Moreover, in several cases, the resulting granularity satisfies particular relationships with the input granularities. For instance, the result of a set operation is always a label-aligned sub-granularity of I, provided that both operands are label-aligned sub-granularities of I. Table 2.1 summarizes the preconditions on the operands and the relationships with the resulting granularity for each calendar operation.

Clearly, checking preconditions of calendar operations is, in general, an undecidable problem. In order to make the check of preconditions feasible, Bettini et. al. [8, 90] assume the existence of a fixed full-integer labeled *bottom granularity* $G_\perp$ covering the whole temporal domain and then they require that every granularity must be derived from $G_\perp$ by applying (maybe repeatedly) calendar operations. They argue that, in such a way, it is possible to effectively test whether preconditions for calendar operations are satisfied. However, such a task may still be computationally expensive (especially for those granularities that have complex repeating patterns, such as `PseudoMonth` and `Month`). To overcome this problem, Bettini et al. introduce further (syntactical) restrictions in the way calendar expressions are built. Precisely, they exploit the relationships derived from the operations themselves in order to classify granularities. As an example, the altering-tick operation can be performed on a pair of granularities G, H if H is the result of a grouping, altering-tick, or shifting operation having G as first operand. Similarly, the anchored grouping operation is applicable to a pair of granularities G, H if G is the result of (a repeated application of) subset, selection, and set operations having H as first operand. From such a kind of restrictions, it is also possible to devise a classification of granularities according to the properties they satisfy. For instance, repeated applications of grouping, altering-tick, and shifting operations starting from the bottom granularity result in a class

Calendar operation	Preconditions	Relationships
$Group_m(G)$	G is a full-integer labeled granularity	the result is a full-integer labeled granularity and G partitions the result
$Alter^m_{l,k}(G,H)$	G,H are full-integer labeled granularities and G partitions H	the result is a full-integer labeled granularity and G partitions the result
$Shift_m(G)$	G is a full-integer labeled granularity	the result is a full-integer labeled granularity partitioning G and G partitions the result
$Combine(G,H)$	none	none
$AnchoredGroup(G,H)$	G is a label-aligned sub-granularity of H and H is a full-integer labeled granularity	none
$Subset^n_m(G)$	none	the result is a label-aligned sub-granularity of G
$SelectUp(G,H)$	none	the result is a label-aligned sub-granularity of G
$SelectDown^l_k(G,H)$	none	the result is a label-aligned sub-granularity of G
$SelectByIntersect^l_k(G,H)$	none	the result is a label-aligned sub-granularity of G
$G \cup H$	both G and H are label-aligned sub-granularities of I	the result is a label-aligned sub-granularity of I
$G \cap H$	both G and H are label-aligned sub-granularities of I	the result is a label-aligned sub-granularity of I
$G - H$	both G and H are label-aligned sub-granularities of I	the result is a label-aligned sub-granularity of I

Table 2.1: Preconditions and relationships for calendar operations.

of full-integer labeled granularities which cover the whole temporal domain. As a matter of fact, Bettini et al. suggest a classification into three layers: layer 1 consists of full-integer labeled granularities that cover the temporal domain, layer 2 consists of granularities having no gaps within granules, and layer 3 consists of granularities with possible gaps within and between granules. For a comprehensive description of how syntactic restrictions can be exploited in order to classify granularities of Calendar Algebra we refer to [8].

2.4 A string-based model for time granularities

In [118], Wijsen proposes an original string-based framework for describing (semi-infinite) time granularities. Representations are given in terms of infinite words over a fixed finite alphabet consisting of three symbols, namely, ■ (filler), □ (gap), and ≀ (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 and to define the labels of the granules. A typical example is the infinite (ultimately periodic) word ■■■■■□□≀■■■■■□□≀..., which represents the granularity `BusinessWeek` over the temporal domain of days or, alternatively, the relation `Days` $\trianglelefteq$ `BusinessWeek`. Here, it is worth noticing that the restriction to a *left-bounded discrete* temporal domain has been made. As a matter of fact, most problems of practical interest involve granularities that are ultimately periodic with respect to a fixed bottom granularity and left-bounded (that is, they have an initial granule). Indeed, one of the main motivations for the proposal of the string-based model was the observation that ultimately periodic left-bounded granularities can be equivalently defined on the temporal domain $(\mathbb{N}, <)$. Moreover, by viewing $(\mathbb{Z}, <)$ as the disjoint union of $(\mathbb{N}, <)$ and its reverse order, it is in general not difficult to extend any given formalism from the case of left-bounded granularities to the case of bi-infinite granularities (the interested reader could read [95] for an application of such an idea in the field of formal languages and automata). Another peculiarity of Wijsen's approach is that the notion of *unlabeled* time granularity has been adopted. For reasons of coherence with the rest of the paper, we adapt Wijsen's formalism to the more general notion of labeled granularity.

From now on, we restrict to granularities defined over the temporal domain $(\mathbb{N}, <)$ and labeled over a subset of $\mathbb{N}$ (namely, for every granularity G and for every $x \in \mathbb{Z}$, we assume that $G(x) = \emptyset$ whenever $x < 0$ and $G(x) \subseteq \mathbb{N}$ whenever $x \geq 0$). Moreover, we shortly denote by $|w|_a$ the number of occurrences of a symbol a in a (finite or infinite) word w.

Definition 2.4.1 *Given an infinite word $w \in \{■, □, ≀\}^\omega$, we say that w* represents a granularity *G if for every $t, x \in \mathbb{N}$, we have $t \in G(x)$ iff $w[t + x + 1] = ■$ and $|w[1, t + x]|_≀ = x$ (namely, $w[1, t + x]$ contains exactly x occurrences of the separator symbol).*

In order to finitely model infinite time granularities, Wijsen introduces the notion of *granspec*. A granspec is an ordered pair (u, v) of finite strings over the alphabet $\{■, □, ≀\}$ such that v contains at least one occurrence of a non-separator symbol (i.e., ■ or □). A granspec (u, v) encodes the ultimately periodic word $w = u \cdot v^\omega$, which, according to Definition 2.4.1, represents a well-defined granularity. As an example, the granspec $(\varepsilon, ■■■■■□□≀)$ represents the granularity `BusinessWeek` in terms of days. From now on, we shortly denote by $[\![\alpha]\!]$ the ultimately periodic word $u \cdot v^\omega$ encoded by a given granspec $\alpha = (u, v)$.

2.4.1 Equivalence of string-based representations

Although it is common practice to let the repeating pattern coincide with granule boundaries, Wijsen's formalism allows repeating patterns starting or ending in the middle of a granule. For instance, an equivalent representation of `BusinessWeek` is (■■, ■■■≀□□■■). Thus, the same granularity can be represented by different granspecs. This gives rise to some interesting problems like the equivalence one (namely, the problem of establishing whether two given representations define the same granularity) and the minimization one (namely, the problem of computing, given a representation of a granularity, an equivalent representation having the minimum number of symbols). Wijsen provides a solution to both problems by introducing aligned and canonical forms of granspecs. Intuitively, in an aligned granspec separators are forced to occur immediately after an occurrence of ■ or ≀. In such a way a one-to-one correspondence between words and granularities is guaranteed. Canonical granspecs are nothing but minimal representations of granspecs in aligned form, namely, pairs of the form (u, v) where $u \cdot v^\omega$ satisfies the alignment condition and u and v have minimum length.

Definition 2.4.2 *An* aligned *granspec α is a granspec that satisfies the following two conditions:*

- *for every $i > 0$, if $[\![\alpha]\!][i] = \wr$, then either $i = 1$ or $[\![\alpha]\!][i-1] \neq \square$,*
- *for every $i > 0$, if $[\![\alpha]\!][i] = \wr$, then there is $j > i$ such that $[\![\alpha]\!][j] = \blacksquare$.*

Definition 2.4.3 *A granspec $\alpha = (u, v)$ is in* canonical form *iff the following conditions are satisfied:*

- *α is an aligned granspec,*
- *$u[|u|] \neq v[|v|]$,*
- *v is primitive, namely, $v = z^k$ implies $k = 1 \ \wedge \ z = v$.*

The following two theorems imply that for every given granspec α, there is a unique canonical granspec β which is equivalent to α.

Theorem 2.4.4 (Wijsen [118]) *For every granspec α, there is an equivalent aligned granspec β. Moreover, if α and β are equivalent aligned granspecs, then $[\![\alpha]\!] = [\![\beta]\!]$.*

Theorem 2.4.5 (Wijsen [118]) *For every granspec α, there is an equivalent canonical granspec β. Moreover, if α and β are equivalent canonical granspecs, then $\alpha = \beta$.*

Given the above results, the equivalence problem can be reduced to the problem of computing canonical forms of granspecs. Moreover, the following properties holds, where $\alpha \equiv \beta$ means that α and β are equivalent granspecs:

i) $(x \cdot y \cdot z, v) \equiv (x \cdot \wr^m \cdot \square^n \cdot z, v)$, provided that $|y|_\blacksquare = 0$, $|y|_\square = n$, $|y|_\wr = m$,

ii) $(u, x \cdot y \cdot z) \equiv (u, x \cdot \wr^m \cdot \square^n \cdot z)$, provided that $|y|_\blacksquare = 0$, $|y|_\square = n$, $|y|_\wr = m$,

iii) $(u, y_1 \cdot x \cdot y_2) \equiv (u \cdot y_1, x \cdot y_2 \cdot y_1)$, provided that $|y_1|_\blacksquare + |y_2|_\blacksquare = 0$, $|y_1|_\square + |y_2|_\square = n$, $|y_1|_\wr + |y_2|_\wr = m$,

iv) $(u, y) \equiv (u, \square)$, provided that $|y|_\blacksquare = 0$.

As a matter of fact, the above equivalences, read from left to right, can be used as rewriting rules in order to bring a given granspec α in an aligned form. Note that, if we always choose y (resp., y_1, y_2) in the left-hand side terms to be maximal substrings containing no occurrences of ■, then, after $\mathcal{O}(|\alpha|)$ reduction steps (where $|\alpha|$ denotes the size of the original granspec), we end up with a granspec in aligned form. Moreover, it is easy to see that each reduction step takes at most linear time in the size of the original granspec.

Similarly, in order to compute the canonical form of an aligned granspec, one can exploit the following two equivalences:

i) $(x \cdot y, z \cdot y) \equiv (x, y \cdot z)$,

ii) $(u, v) \equiv (u, z)$, provided that $v = z^k$.

In this case, each reduction rule can be applied at most once if, starting from a given granspec $\alpha = (u, v)$, we respectively choose y to be the longest common suffix of u and v and we choose z to be the shortest repeating pattern of v (refer to Section 1.5 for an algorithm that computes the shortest repeating pattern of a word v in time $\Theta(|v|)$). As a consequence of these results, the equivalence problem for granspecs turns out to be solvable in quadratic time with respect to the size of the input granspecs.

In the following, we present a more efficient (linear-time) solution to the equivalence problem for granspecs. We start by defining a 'matching' relationship between (finite or infinite) words over the alphabet $\{■, □, \wr\}$.

Definition 2.4.6 *Two (finite or infinite) words u and v* match *iff all occurrences of ■ lie at the same positions in u and in v and they are preceded by the same number of occurrences of $\wr$.*

For instance, the words ■□$\wr\wr$□□■ and ■$\wr$□□$\wr$□■ match, while ■□$\wr\wr$□□■ and ■□$\wr\wr$□$\wr$■ do not match.

Proposition 2.4.7 *Two infinite words u and v represent the same granularity iff they match.*

Proof. Let G and H be the granularities represented by u and v in terms of a bottom granularity $G_\perp$. We prove the left-to-right implication. Suppose that u and v do not match and let i be the position of the first occurrence of ■ in u such that either $i \neq j$ or $i = j \ \wedge \ x \neq y$ holds, where j is the position of the $|u[1,i]|_■$-th occurrence of ■ in v (if it exists, otherwise $G \neq H$ follows trivially), $x = |u[1, i-1]|_\wr$, and $y = |u[1, j-1]|_\wr$. By definition, if we let $z = i - x - 1$ and $z' = j - y - 1$, we have that $G_\perp(z) \subseteq G(x)$ and $G_\perp(z') \subseteq H(y)$. Now, if $i \neq j$, then either $v[i, j-1]$ or $v[j+1, i]$ is a non-empty substring containing only occurrences of □ and $\wr$ and hence $G_\perp(z) \not\subseteq H(x)$. Otherwise, if $i = j \ \wedge \ x \neq y$, then $G(x) \cap H(y) \neq \emptyset$. In both cases, $G \neq H$ follows.

For the converse implication, we assume that u and v match and we prove that, for every $z, x \in \mathbb{N}$, $G_\perp(z) \subseteq G(x)$ iff $G_\perp(z) \subseteq H(x)$. Suppose that $G_\perp(z) \subseteq G(x)$ for some $z, x \in \mathbb{N}$. Clearly, $u[z+x+1] = ■$ and $|u[1, z+x]|_\wr = x$ hold. From the hypothesis, we know that $v[z+x+1] = ■$ and $|v[1, z+x]|_\wr = x$, which implies that $G_\perp(z) \subseteq H(x)$. ■

According to Definition 2.4.6, checking whether two given infinite words match or not may require an infinite number of tests to be performed. However, by exploiting the periodicity of the words represented by granspecs, the number of such tests can be bounded with respect to the size of the involved granspecs.

Proposition 2.4.8 *Two ultimately periodic words $w = u{\cdot}v^{\omega}$ and $w' = u'{\cdot}(v')^{\omega}$ match iff their n-character prefixes match, where $n = max(|u|+|v|, |u'|+|v'|)+lcm(|v|,|v'|)$.*

Proof. Clearly, if w and w' match, then also their n-character prefixes do. As for the converse implication, assume that $w[1,n]$ and $w'[1,n]$ match, where $n = max(|u| + |v|, |u'| + |v'|) + lcm(|v|,|v'|)$. Now, let i be the position of an occurrence of ■ in w. We distinguish between two cases: $i \leq n$ and $i > n$. In the former case, we clearly have $v[i] =$ ■ and $|w[1,i-1]|_{\wr} = |w'[1,i-1]|_{\wr}$. In the latter case, we define $r = max(|u|,|u'|)$, $p = max(|u| + |v|, |u'| + |v'|)$, $q = lcm(|v|,|v'|)$, and i_0 as the position in $w[1,p+q]$ (or, equivalently, in $w'[1,p+q]$ since $n = p+q$ and the two words $w[1,p+q]$ and $w'[1,p+q]$ match) of the first occurrence of ■ after the r-th position. Notice that $p < r < i_0 \geq p+q$. This means that the filler symbol at position i_0 occurs both in the repeating pattern of w and in the repeating pattern of w'. Moreover, since the words $w[1,p+q]$ and $w'[1,p+q]$ match, for every symbol $a \in \{$■, □, $\wr\}$, we have:

i) $\big|w[1,i_0]\big|_a = \big|w'[1,i_0]\big|_a$,

ii) $\big|w[i_0+1,i_0+q]\big|_a = \big|w'[i_0+1,i_0+q]\big|_a$,

iii) $\big|w[i_0+1,i]\big|_a = k\big|w[i_0+1,i_0+q]\big|_a + \big|w[i_0+1,i_0+h]\big|_a = k\big|w'[i_0+1,i_0+q]\big|_a + \big|w'[i_0+1,i_0+h]\big|_a = \big|w'[i_0+1,i]\big|_a$, where $k = \lfloor \frac{i-i_0-1}{q} \rfloor$ and $h = (i-i_0-1) \bmod q$.

In particular, this implies that $w'[i] =$ ■ and $|w'[1,i-1]|_{\wr} = |w[1,i-1]|_{\wr}$. Thus we must conclude that the two infinite words w and w' match. ■

Proposition 2.4.8 is not sufficient to achieve a linear-time solution to the equivalence problem yet. The following proposition shows that two ultimately periodic words $w = u \cdot v^{\omega}$ and $w' = u' \cdot (v')^{\omega}$ match iff there is a suitable ultimately periodic word $w'' = u'' \cdot (v'')^{\omega}$ that match with both w and w' and it has a repeating pattern of length $gcd(|v|,|v'|)$.

Proposition 2.4.9 *Two ultimately periodic words $w = u{\cdot}v^{\omega}$ and $w' = u'{\cdot}(v')^{\omega}$ match iff $w'' = u''{\cdot}(u'')^{\omega}$ match with both w and w', where $u'' = w[1,p]$, $v'' = w[p+1,p+q]$, p is the position in w of the first occurrence of ■ after the position $max(|u|,|u'|)$ (if such an occurrence does not exist, then we set $p = max(|u|,|u'|)$), and $q = gcd(|v|,|v'|)$.*

Proof. The right-to-left implication follows trivially from the transitivity property of the matching relation. As for the left-to-right implication, suppose that $w = u{\cdot}v^{\omega}$ and $w' = u'{\cdot}(v')^{\omega}$ match. We can assume, without loss of generality, that v and v' contains at least one occurrence of ■ (if this is not the case, then the claim holds trivially). Let $r = max(|u|,|u'|)$ and let p and q be defined as in the proposition. Note that $r < p \leq r + min(|v|,|v'|)$. This means that the filler symbol at position p occurs both

in the repeating pattern of w and in the repeating pattern of w'. Now, consider the two sets $I = \{p + k|v| : k \in \mathbb{N}\}$ and $I' = \{p + k'|v'| : k' \in \mathbb{N}\}$. Since w and w' match, it follows that, for every $k, k' \in \mathbb{N}$, $w[p + k|v|] = w[p] = \blacksquare = w'[p] = w'[p + k'|v'|]$ and hence for every $i \in I \cup I'$, $w[i] = w'[i] = \blacksquare$. Moreover, for every $i \in I \cup I'$, the words $w[1, i]$ and $w'[1, i]$ match and thus, since $w[i] = w'[i] = \blacksquare$, $w[i + 1, i + q]$ and $w'[i + 1, i + q]$ match as well. By definition of I, I', and q, for every $0 \leq h < \frac{|v|}{q}$, there exists $i_h \in I'$ such that $(i_h - p - 1) \bmod |v| = hq$. Hence, by exploiting the reflexive and transitive property of the matching relation, we know that the words $v'' = w[p + 1, p + q]$, $w'[p + 1, p + q] = w'[i_h + 1, i_h + q]$, and $w[i_h + 1, i_h + q] = w[p + hq + 1, p + (h + 1)q]$ match two by two. This means that v'' matches with every substring of the form $w[p+hq+1, p+(h+1)q]$ and, by a similar argument, with every substring of the form $w'[p + hq + 1, p + (h + 1)q]$. This proves that the ultimately periodic word $w'' = u'' \cdot (v'')^\omega$, where $u'' = w[1, p]$, matches with both w and w'. ■

Theorem 2.4.10 *The equivalence problem for granspecs is solvable in linear time.*

Proof. The claim follows trivially from Proposition 2.4.8 and Proposition 2.4.9. ■

Below we report a simple algorithm that solves the equivalence problem in linear time with respect to the size of the input granspecs.

TestEquivalence(α, β)

let $\alpha = (u, v)$
let $\beta = (u', v')$
if $|v| \bmod |v'| \neq 0$ **then**
  if $|v|_{\blacksquare} > 0$ **then**
    $p \leftarrow min\{i > max(|u|, |u'|) : (u \cdot v)[i] = \blacksquare\}$
  else
    $p \leftarrow max(|u|, |u'|)$
  end if
  $q \leftarrow gcd(|v|, |v'|)$
  $u'' \leftarrow (u \cdot v)[1, p]$
  $v'' \leftarrow v[p + 1 - |u|, |v|] \cdot v[1, p - |u|]$
  $\gamma \leftarrow (u'', v'')$
  return *TestEquivalence*(α, γ) **and** *TestEquivalence*(β, γ)
end if
$n \leftarrow max(|u| + |v|, |u'| + |v'|) + lcm(|v|, |v'|)$
$x \leftarrow 0$
$y \leftarrow 0$
for $i = 1...n$ **do**
  $a \leftarrow (u \cdot v)[((i - |u| - 1) \bmod |v|) + |u| + 1]$
  $b \leftarrow (u' \cdot v')[((i - |u'| - 1) \bmod |v'|) + |u'| + 1]$
  if $a = \wr$ **then**
    $x \leftarrow x + 1$
  end if
  if $b = \wr$ **then**

```
25:       y ← y + 1
26:     end if
27:     if (a = ■ or b = ■) and (x ≠ y or a ≠ ■ or b ≠ ■) then
28:       return false
29:     end if
30: end for
31: return true
```

2.4.2 A string-based account of Calendar Algebra

In [8] (Theorem 2.6.1) Bettini et al. show that Calendar Algebra captures all finite and ultimately periodic granularities. Not surprisingly, the string-based model as well captures all and only the finite and ultimately periodic granularities. In this section we prove that the formalism of granspecs actually subsumes Calendar Algebra, thus showing that the expressiveness of the two formalisms is the same. More precisely, under the assumption of a left-bounded discrete temporal domain, we develop some algorithms that map calendar expressions to granspecs that equivalently represent the same granularity in terms of the bottom one, denoted $G_\perp$ (note that any granularity generated by repeated applications of calendar operations, starting from $G_\perp$, groups $G_\perp$). It is worth remarking that Calendar Algebra gives no means to decide whether two given expressions represent the same granularity. Having solved the equivalence problem for the string-based model and having developed some algorithms that map calendar expressions to string-based specifications of time granularities, give us an effective way to test the equivalence of the expressions of Calendar Algebra. In this perspective, the string-based approach (as well as the automaton-based one, which is intimately related to it) can be viewed as a 'low-level' framework into which 'high-level' and 'user-friendly' formalisms like Calendar Algebra can be mapped. Moreover, recall that several calendar operations are defined only if particular preconditions on operands hold. To avoid checking these preconditions, Bettini et al. introduced syntactic restrictions on the way the calendar expressions are formed. The expressiveness results provided in this section hold even if we do not choose to enforce such syntactical restrictions.

In order to simplify the notation, given an ultimately periodic word w, we shall denote by p_w, q_w, respectively, the length of a prefix and the length of a repeating pattern of w, namely, $p_w = |u|$ and $q_w = |v|$ for suitable u, v such that $w = u \cdot v^\omega$. Given p_w, q_w, we then define the values $\widetilde{p}_w = |w[1, p_w]|_{\wr}$, $\widetilde{q}_w = |w[p_w + 1, p_w + q_w]|_{\wr}$, $\bar{p}_w = |w[1, p_w]|_{\blacksquare,\square}$, and $\bar{q}_w |w[p_w + 1, p_w + q_w]|_{\blacksquare,\square}$. Clearly, for every ultimately periodic word w over the alphabet $\{\blacksquare, \square, \wr\}$, $p_w = \widetilde{p}_w + \bar{p}_w$ and $q_w = \widetilde{q}_w + \bar{q}_w$. As an example, given the ultimately periodic word $w = (\blacksquare\blacksquare\blacksquare\blacksquare\square\square\wr)^\omega$, we can have $p_w = 0$, $q_w = 8$, $\widetilde{p}_w = 0$, $\widetilde{q}_w = 1$, $\bar{p}_w = 0$, $\bar{q}_w = 7$. Furthermore, in order to shortly describe the algorithms, we exploit some auxiliary procedures that process symbol occurrences in the ultimately periodic word represented by a given granspec. Table 2.2 reports the heading and a short description of the behavior of such basic procedures. Note that several special cases should be taken into account in order to implement

such procedures. As an example, if $i = 0$, then $Occurrence(\alpha, i, A)$ returns 0 and if $[\![\alpha]\!]$ contains less than i occurrences of symbols from A, then $Occurrence(\alpha, i, A)$ returns ω; similarly, if $[\![\alpha]\!]$ contains no occurrences of symbols from A before the i-th occurrence of a symbol from B, then $OccurrenceLastBefore(\alpha, i, A, B)$ returns 0 and if $[\![\alpha]\!]$ contains less than i occurrences of symbols from B, then $OccurrenceLastBefore(\alpha, i, A, B)$ returns the position of the last occurrence of a symbol from A (possibly ω if there are infinitely many of such occurrences). We can also use similar procedures when dealing with finite words.

Algorithm	Behavior
$SymbolAt(\alpha, i)$	Returns the symbol at position i in $[\![\alpha]\!]$.
$Substring(\alpha, i, j)$	Returns the finite substring $[\![\alpha]\!][i, j]$.
$CountOccurrences(\alpha, i, j, A)$	Returns the number of occurrences in $[\![\alpha]\!][i, j]$ of symbols from A.
$Occurrence(\alpha, i, A)$	Returns the position in $[\![\alpha]\!]$ of the i-th occurrence of a symbol from A.
$OccurrenceFirstAfter(\alpha, i, A, B)$	Returns the position in $[\![\alpha]\!]$ of the first occurrence of a symbol from A after the i-th occurrence of a symbol from B.
$OccurrenceLastBefore(\alpha, i, A, B)$	Returns the position in $[\![\alpha]\!]$ of the last occurrence of a symbol from A before the i-th occurrence of a symbol from B.

Table 2.2: Some auxiliary procedures for string-based representations.

The presentation of the algorithms is organized as follows: for each calendar operation, we first build, on the grounds of the ultimately periodic word(s) representing the operand(s), an ultimately periodic word representing the resulting granularity, then we relate the lengths of the prefixes and the repeating patterns of the input word(s) to those of the output word, and finally we provide an algorithm that mimics the construction of the output word directly on granspecs. It is routine to check that each of such algorithms takes at most quadratic time with respect to the size of the input granspec(s). Moreover, it is worth remarking that the algorithms are not guaranteed to generate *minimum* granspecs. Nevertheless, it is always possible to post-process the generated granspecs in order to reduce their size or bring them in canonical form. From now on, we understand that the label set of any time granularity is a subset of $\mathbb{N}$. Accordingly, we call full-integer labeled granularity any granularity whose label set coincides with $\mathbb{N}$.

Grouping operation

Proposition 2.4.11 *Let $m > 1$, let u be an infinite word representing the full-integer labeled granularity G, and let v be the infinite word obtained from u by removing every i-th occurrence of $\wr$, where i is not a multiple of m. Then, v represents the*

granularity $Group_m(G)$. *Moreover, we have* $p_v = \bar{p}_u + \lfloor \frac{\widetilde{p}_u}{m} \rfloor$, *and* $q_v = k\bar{q}_u + \frac{k\widetilde{q}_u}{m}$, *where* $k = \frac{m}{gcd(m,\widetilde{q}_u)}$.

Proof. Let $H = Group_m(G)$. By definition of H, for every $z, y \in \mathbb{N}$, $G_\perp(z) \subseteq H(y)$ iff there is $x \in \mathbb{N}$ such that $my \leq x < m(y+1)$ and $G_\perp(z) \subseteq G(x)$. The latter condition $G_\perp(z) \subseteq G(x)$ holds iff $u[z+x+1] = \blacksquare$ and $|u[1, z+x]|_\wr = x$. Moreover, by definition of v, we have $u[z+x+1] = \blacksquare \wedge |u[1, z+x]|_\wr = x$ iff $v[z+y'+1] = \blacksquare \wedge |v[1, z+y']|_\wr = y'$, where $y' = \lfloor \frac{x}{m} \rfloor$. Since $my \leq x < m(y+1)$ holds iff $y = \lfloor \frac{x}{m} \rfloor$, we can conclude that $y' = y$ and $G_\perp(t) \subseteq H(y)$ iff $v[z+y+1] = \blacksquare \wedge |v[1, z+y]|_\wr = y$, which proves that v represents H in terms of $G_\perp$.
It remains to show that v is ultimately periodic and to calculate the length of a prefix and the length of a repeating pattern of v. Let p_u and q_u be the length of a prefix and the length of a repeating pattern of u, respectively. Since G is a full-integer labeled granularity, u contains infinitely many occurrences of $\wr$. Hence $\widetilde{q}_u > 0$ and we can define $k = \frac{m}{gcd(m,\widetilde{q}_u)}$. Clearly, kq_u is the length of a repeating pattern of u and a multiple of m. Therefore, since v is obtained from u by removing $m-1$ occurrences out of m, we easily see that $v = v[1, p_v] \cdot v[p_v + 1, p_v + q_v]^\omega$, where $p_v = \bar{p}_u + \lfloor \frac{\widetilde{p}_u}{m} \rfloor$ and $q_v = k\bar{q}_u + \frac{k\widetilde{q}_u}{m}$. ■

The following algorithm is based on Proposition 2.4.11 and it computes, given a granspec α representing G and given a parameter $m > 0$, a new granspec β representing $Group_m(G)$.

MapGroup(α, m)

```
1: let α = (u_1, u_2)
2: p_u ← |u_1|
3: q_u ← |u_2|
4: p̃_u ← CountOccurrences(α, 1, p_u, {≀})
5: q̃_u ← CountOccurrences(α, p_u + 1, p_u + q_u{≀})
6: p̄_u ← CountOccurrences(α, 1, p_u, {■, □})
7: q̄_u ← CountOccurrences(α, p_u + 1, p_u + q_u{■, □})
8: k ← m/gcd(m, q̃_u)
9: p_v ← p̄_u + ⌊p̃_u/m⌋
10: q_v ← k * q̄_u + k * q̃_u/m
11: x ← 1
12: i ← 1
13: v ← ε
14: while |v| < p_v + q_v do
15:    if SymbolAt(α, i) ∈ {■, □} then
16:       v ← v · SymbolAt(α, i)
17:    else
18:       if x mod m = 0 then
19:          v ← v · SymbolAt(α, i)
20:       end if
21:       x ← x + 1
```

22: **end if**
23: $i \leftarrow i + 1$
24: **end while**
25: **return** $(v[1, p_v], v[p_v + 1, p_v + q_v])$

Altering-tick operation

Let fix two full-integer labeled granularities G and H such that G partitions H. Before describing how the altering-tick operation is mapped in the string-based model, we solve a simpler problem, namely, that of computing the separator function $sep_{G,H}$ (cf. Section 2.3.1).

Proposition 2.4.12 *Let u, v be two ultimately periodic words representing, respectively, G and H in terms of $G_\perp$. For every $y \in \mathbb{N}$, we have $sep_{G,H}(y) = \left|u[1, i]\right|_\wr$, where*

- *i is the position in u of the j-th occurrence of non-separator symbols,*
- *j is the number of occurrences in $v[1, k]$ of non-separator symbols,*
- *r is the position in v of the first occurrence of $\blacksquare$ after the y-th occurrence of $\wr$.*

Proof. Since G partitions H, u and v contain the same occurrences of non-separator symbols in the same order. Furthermore, two occurrences of non-separator symbols in v are interleaved by $\wr$ only if the corresponding occurrences in u are interleaved by at least one occurrence of $\wr$. Now, let fix $y \in \mathbb{N}$ and let r, j, i be the values defined in the proposition. We have to verify that $x = |u[1, i]|_\wr$ is the label of the first granule of G that is included in $H(y)$. Clearly, since $H(y) \neq \emptyset$, $v[j + y + 1] = \blacksquare \ \wedge \ |v[1, j + y]|_\wr = y$ and hence $G_\perp(j) \subseteq H(y)$. Similarly, since G partitions H, $u[j + x + 1]] = \blacksquare \ \wedge \ |u[1, j + x]|_\wr = x$ and hence x is the label of a granule of G that is included in $H(y)$. It remains to show that x is the *least* label of those granules of G that are included in $H(y)$. From the previous observations, if there existed a label $x' < x$ such that $\emptyset \neq G(x') \subseteq H(y)$, then there would exist also a label $j' < j$ such that $G_\perp(j') \subseteq G(x') \subseteq H(y)$ and hence v would contain an occurrence of $\blacksquare$ before the position r but after the y-th occurrence of $\wr$, which is against the definition of r. Thus, we can conclude that $sep_{G,H}(y) = x$. ∎

The following algorithm receives in input two string-based representations α and β the granularities for G and H and a label $y \in \mathbb{Z}$, and it returns in output the label of the first granule of G that is included in $H(y)$.

Sep(α, β, y)
1: $g \leftarrow$ *OccurrenceFirstAfter*$(\beta, y, \{\blacksquare\}, \{\wr\})$
2: $j \leftarrow$ *CountOccurrences*$(\beta, 1, g, \{\blacksquare, \square\})$
3: $i \leftarrow$ *Occurrence*$(\alpha, j, \{\blacksquare, \square\})$
4: **return** *CountOccurrences*$(\alpha, 1, i, \{\wr\})$

Now, we are able to provide a construction of an ultimately periodic word representing the granularity $Alter^m_{l,k}(G,H)$. In the sequel, $sep'_{G,H}$ denotes the function that maps a label of a granule of H to $sep_{G,H}(y) + k\lfloor\frac{max(0,y-l)}{m}\rfloor$, where l, k, m are some fixed parameters. Note that such a function is computable given the representation α and β for G and H.

Proposition 2.4.13 *Let $l, k, m \in \mathbb{N}$, with $m > 0$, and let u and v be two ultimately periodic words representing, respectively, G and H in terms of $G_\perp$. Further let $u = u_0 \cdot u_1 \cdot u_2 \cdot ...$, where each factor u_y of u ends at the position of the $sep'_{G,H}(y+1)$-th occurrence of $\wr$ in u. If we denote by w_y the word obtained from u_y by removing every occurrence of $\wr$ except the last one, then the infinite word $w = w_0 \cdot w_1 \cdot w_2 \cdot ...$ represents the granularity $Alter^m_{l,k}(G,H)$ in terms of $G_\perp$. Moreover, if p_u and q_u (resp., p_v and q_v) are the length of a prefix and the length of a repeating pattern of u (resp., v), then we can assume, without loss of generality, that*

i) $\widetilde{p}_v \geq l$ (if this is not the case, then we can prolong the prefix of v since v contains infinitely many occurrences of $\wr$),

ii) $\bar{p}_u = \bar{p}_v$, (if this is not the case, then we can prolong either the prefix of u or the prefix of v),

iii) $\widetilde{q}_v$ is a multiple of $m\widetilde{q}_u$ (if this is not the case, then we can take a suitable repetition of the pattern of v),

iv) there exist $h_u, h_v > 0$ such that $h_u\bar{q}_u = h_v\bar{q}_v$.

Under these assumptions, the word w is ultimately periodic and it has a prefix of length $p_w = \widetilde{p}_v + |u[1,n]|_{\blacksquare,\square}$, where n is the position in u of the $sep'_{G,H}(\widetilde{p}_v)$-th occurrence of $\wr$, and a repeating pattern of length $q_w = h_v\widetilde{q}_v + h_u\bar{q}_u + k\frac{h_v\widetilde{q}_v}{m\widetilde{q}_u}\bar{q}_u$.

Proof. We preliminary notice the following facts (they can be easily proved by exploiting induction on j and the definition of w):

i) for every $j \in \mathbb{N}$, if $u[j+1] = \blacksquare$, $|u[1,j]|_\wr = x$ and $sep'_{G,H}(y) \leq x < sep'_{G,H}(y+1)$, then $w[j-x+y+1] = \blacksquare$ and $|w[1,j-x+y]|_\wr = y$,

ii) for every $j \in \mathbb{N}$, if $w[j+1] = \blacksquare$ and $|w[1,j]|_\wr = y$, then there exist $x \in \mathbb{N}$ such that $sep'_{G,H}(y) \leq x < sep'_{G,H}(y+1)$, $u[j+x-y+1] = \blacksquare$, and $|u[1,j+x-y]|_\wr = x$.

Let $J = Alter^m_{l,k}(G,H)$. For every $z, y \in \mathbb{N}$, $G_\perp(z) \subseteq J(y)$ iff there is $x \in \mathbb{N}$ such that $sep'_{G,H}(y) \leq x < sep'_{G,H}(y+1)$ and $G_\perp(z) \subseteq G(x)$. Since u represents G in terms of $G_\perp$, $G_\perp(z) \subseteq G(x)$ iff $u[z+x+1] = \blacksquare \ \wedge \ |u[1,z+x]|_\wr = x$. From the above properties, we know that $G_\perp(z) \subseteq G(x)$ iff $w[z+y+1] = \blacksquare \wedge |w[1,z+y-1]|_\wr = y$. This proves that w represents J in terms of $G_\perp$.

It remains to show that the word w is ultimately periodic. Let p_u, q_u, p_v, and q_v be the lengths of a prefix of u, a repeating pattern of u, a prefix of v, and a repeating pattern of v satisfying the conditions given in the proposition. For every $y \in \mathbb{N}$, we define the following values:

- r_y is the position in v of the first occurrence of $\blacksquare$ after the $y+1$-th occurrence of $\wr$,
- j_y is the number of occurrences in $v[1,r_y]$ of non-separator symbols,
- i_y is the position in u of the j_y-th occurrence of non-separator symbols,

- s_y is the number of occurrences in $u[1, i_y]$ of $\wr$,
- $s'_y = s_y + k\lfloor\frac{max(0,y+1-l)}{m}\rfloor$,
- $t'_{-1} = 0$ and t'_y is the position in u of the s'_y-th occurrence of $\wr$.

Notice that, by Proposition 2.4.12, $sep_{G,H}(y+1) = s_y$ and $sep'_{G,H}(y+1) = s'_y$ hold. Moreover, $u_y = u[t'_{y-1} + 1, t'_y]$. Now, we characterize the above series in terms of generalized arithmetic progressions. Precisely, we have:

i) $r_y \leq p_v + q_v$ holds for every $y < \widetilde{p}_v$ and, since $h_v q_v$ is the length of a repeating pattern of v, $r_{y+h_v\widetilde{q}_v} = r_y + h_v q_v$ holds for every $y \geq \widetilde{p}_v$,
ii) $j_y < \widetilde{p}_v + \widetilde{q}_v$ holds for every $y \leq \widetilde{p}_v$ and $j_{y+h_v\widetilde{q}_v} = j_y + h_v\bar{q}_v$ holds for every $y \geq \widetilde{p}_v$,
iii) since $\bar{p}_u = \bar{p}_v$ and $h_u\bar{q}_u = h_v\bar{q}_v$, $i_y \leq p_u + h_u q_u$ holds for every $y < \widetilde{p}_v$ and and $i_{y+h_v\widetilde{q}_v} = i_y + h_u q_u$ holds for every $y \geq \widetilde{p}_v$,
iv) $s_{y+h_v\widetilde{q}_v} = s_y + h_u\widetilde{q}_u$ holds for every $y \geq \widetilde{p}_v$,
v) since $\widetilde{p}_v \geq l$ and $h_v\widetilde{q}_v$ is multiple of m, $s'_{y+h_v\widetilde{q}_v} = s'_y + h_u\widetilde{q}_u + k\frac{h_v\widetilde{q}_v}{m}$ holds for every $y \geq \widetilde{p}_v$,
vi) $t_{y+h_v\widetilde{q}_v} = t_y + h_u q_u$ holds for every $y \geq \widetilde{p}_v$,
vii) since $k\frac{h_v\widetilde{q}_v}{m}$ is multiple of $\widetilde{q}_u$, $t'_{y+h_v\widetilde{q}_v} = t'_y + h_u q_u + k\frac{h_v\widetilde{q}_v}{m\widetilde{q}_u}q_u$ holds for every $y \geq \widetilde{p}_v$.

Since $h_u q_u + k\frac{h_v\widetilde{q}_v}{m\widetilde{q}_u q_u}q_u$ is the length of a repeating pattern of u, for every $y \geq \widetilde{p}_v$ we have $u_{y+h_v\widetilde{q}_v} = u_y$ and hence $w_{y+h_v\widetilde{q}_v} = w_y$. Thus, we can conclude that w has a prefix of length $p_w = \widetilde{p}_v + |u[1,n]|_{\blacksquare,\square}$, where n is the position in u of the $sep'_{G,H}(\widetilde{p}_v)$-th occurrence of $\wr$, and a repeating pattern of length $q_w = h_v\tilde{q}_v + h_u\bar{q}_u + k\frac{h_v\tilde{q}_v}{m\tilde{q}_u}\bar{q}_u$. ■

Before describing the algorithm that maps the altering-tick operation, we first show how to accommodate the hypothesis of Proposition 2.4.13 on the prefixes and repeating patterns of u and v and how to compute the length of a prefix and the length of a repeating pattern of w. The following algorithm receives in input two granspecs α and β, with $[\![\alpha]\!] = u$ and $[\![\beta]\!] = v$, and some parameters l, k, m, and it returns in output the length of a prefix and the length of a repeating pattern of w.

AlterPrefixAndRepeatingPattern(α, β, l, k, m)

1: **let** $\alpha = (u_1, u_2)$
2: **let** $\beta = (v_1, v_2)$
3: $p_u \leftarrow |u_1|$
4: $q_u \leftarrow |u_2|$
5: $p_v \leftarrow |v_1|$
6: $q_v \leftarrow |v_2|$
7: **if** *CountOccurrences*$(\beta, 1, p_v, \{\wr\}) < l$ **then**
8: $\quad p_v \leftarrow$ *Occurrence*$(\beta, l, \{\wr\})$
9: **end if**
10: $\bar{p}_u \leftarrow$ *CountOccurrences*$(\alpha, 1, p_u, \{\blacksquare, \square\})$
11: $\bar{p}_v \leftarrow$ *CountOccurrences*$(\beta, 1, p_v, \{\blacksquare, \square\})$
12: **if** $\bar{p}_v > \bar{p}_u$ **then**
13: $\quad p_u \leftarrow$ *Occurrence*$(\alpha, \bar{p}_v, \{\blacksquare, \square\})$
14: **else if** $\bar{p}_u > \bar{p}_v$ **then**

15: $p_v \leftarrow Occurrence(\beta, \bar{p}_u, \{\blacksquare, \square\})$
16: **end if**
17: $\widetilde{p}_v \leftarrow CountOccurrences(\beta, 1, p_v, \{\wr\})$
18: $n \leftarrow Occurrence(\alpha, Sep'(\alpha, \beta, \widetilde{p}_v), \{\wr\})$
19: $p_w \leftarrow \widetilde{p}_v + CountOccurrences(\alpha, 1, n, \{\blacksquare, \square\})$
20: $\widetilde{q}_u \leftarrow CountOccurrences(\alpha, p_u + 1, p_u + q_u, \{\wr\})$
21: $\widetilde{q}_v \leftarrow CountOccurrences(\beta, p_v + 1, p_v + q_v, \{\wr\})$
22: $q_v \leftarrow q_v * m * \widetilde{q}_u / gcd(\widetilde{q}_v, m * \widetilde{q}_u)$
23: $\bar{q}_u \leftarrow CountOccurrences(\alpha, p_u + 1, p_u + q_u, \{\blacksquare, \square\})$
24: $\bar{q}_v \leftarrow CountOccurrences(\beta, p_v + 1, p_v + q_v, \{\blacksquare, \square\})$
25: $h_u \leftarrow lcm(\bar{q}_u, \bar{q}_v)/\bar{q}_u$
26: $h_v \leftarrow lcm(\bar{q}_u, \bar{q}_v)/\bar{q}_v$
27: $q_w \leftarrow h_v * \tilde{q}_v + h_u * \bar{q}_u + k * (h_v * \widetilde{q}_v)/(m * \tilde{q}_u) * \bar{q}_u$
28: **return** (p_w, q_w)

Now, following the construction of the ultimately periodic word w given in Proposition 2.4.13, we provide an algorithm that computes a granspec representing the granularity $Alter^m_{l,k}(G, H)$ on the grounds of the input granspecs representing G and H and of the parameters l, k, m.

MapAlter(α, β, l, k, m)
1: $(p_w, q_w) \leftarrow AlterPrefixAndRepeatingPattern(\alpha, \beta, l, k, m)$
2: $x \leftarrow 0$
3: $y \leftarrow 0$
4: $i \leftarrow 1$
5: $w \leftarrow \varepsilon$
6: **while** $|w| < p_w + q_w$ **do**
7: **if** $SymbolAt(\alpha, i) \in \{\blacksquare, \square\}$ **then**
8: $w \leftarrow w \cdot SymbolAt(\alpha, i)$
9: **else**
10: $x \leftarrow x + 1$
11: **if** $x = Sep'_{G,H}(\alpha, \beta, y + 1)$ **then**
12: $w \leftarrow w \cdot SymbolAt(\alpha, i)$
13: $y \leftarrow y + 1$
14: **end if**
15: **end if**
16: $i \leftarrow i + 1$
17: **end while**
18: **return** $(w[1, p_w], w[p_w + 1, p_w + q_w])$

Shifting operation

The shifting operation generates a new granularity by shifting the labels of the granules of the input granularity. Since granspecs represent only granularities whose labels are non-negative integers, we have to slightly modify the definition of such an

operation, in order to guarantee that the label set of the resulting granularity is a subset of $\mathbb{N}$. From now on, given a full-integer labeled granularity G, we identify the expression $\mathit{Shift}_m(G)$ with the granularity H such that for every $x \geq \mathit{max}(m, 0)$, $H(x) = G(x - m)$ and for every $x < \mathit{max}(m, 0)$, $H(x) = \emptyset$. Note that, H always partitions G. However, if $m > 0$, then H is not a full-integer granularity (since $H(x) = \emptyset$ for every $0 \leq x < m$); on the other hand, if $m < 0$, then G does not partition H.

Proposition 2.4.14 *Let $m \in \mathbb{Z}$, let u be an ultimately periodic word representing the full-integer labeled granularity G in terms of $G_\perp$, and let v be the word defined as follows:*

- *if $m \geq 0$, then $v = \wr^m$,*
- *if $m < 0$, then $v = \square^{|u_1|}{}_{\blacksquare,\square} \cdot u_2$, where $u = u_1 \cdot u_2$ and u_1 ends at the position of the $(-m)$-th occurrence $\wr$ in u.*

Then, v represents the granularity $\mathit{Shift}_m(G)$ in terms of $G_\perp$.

Proof. Let $H = \mathit{Shift}_m(G)$. For every $x \geq \mathit{max}(m, 0)$, $H(x) = G(x - m)$ and hence, for every $x \geq \mathit{max}(m, 0)$ and for every $z \in \mathbb{N}$, $G_\perp(z) \subseteq H(x)$ iff $G_\perp(z) \subseteq G(x - m)$. The latter condition holds iff $u[z+x-m+1] = \blacksquare \wedge |u[1, z + x - m]|_\wr = x-m$, hence iff $v[z + x + 1] = \blacksquare \wedge |v[1, z + x]|_\wr = x$. Moreover, if $m \geq 0$, every occurrence of $\blacksquare$ in v is preceded by at least m occurrences of $\wr$, thus implying that $v[z + x + 1] = \square$ whenever $|v[1, z + x]|_\wr = x < m$. This shows that v represents H in terms of $G_\perp$. ■

The following algorithm receives in input a granspec α representing G and a parameter $m \in \mathbb{N}$ and it returns in output a granspec β that represents $\mathit{Shift}_m(G)$.

MapShift(α, m)

```
1: let α = (u1, u2)
2: p_u ← |u1|
3: q_u ← |u2|
4: if m ≥ 0 then
5:    β ← (℩^m · u1, u2)
6: else
7:    i ← Occurrence(α, −m, {℩})
8:    k ← CountOccurrences(α, 1, i, {■, □})
9:    if i ≤ p_u then
10:      β ← (□^k · u1[i + 1, p_u], u2)
11:   else
12:      β ← (□^k, u2[i − p_u, q_u] · u2[1, i − p_u − 1])
13:   end if
14: end if
15: return β
```

Combining operation

Before describing how the combining operation is mapped in the string-based model, we focus on the problem of computing, given two granularities G and H and a label $x \in \mathbb{N}$, the label $y \in \mathbb{N}$ of the granule of H (if exists) that includes the granule $G(x)$.

Proposition 2.4.15 *Let u and v be two words representing, respectively, the granularities G and H in terms of $G_\perp$ and let p_u, q_u, p_v, and q_v be, respectively, the lengths of a prefix of u, a repeating pattern of u, a prefix of v, and a repeating pattern of v satisfying $\bar{p}_u = \bar{p}_v$ and $\bar{q}_u = \bar{q}_v$. Given $x \in \mathbb{N}$, we further denote by z_x (resp., z'_x) the number of occurrences in u of non-separator symbols before the x-th occurrence (resp., the $x+1$-th occurrence) of $\wr$ (if u contains less than x occurrences of $\wr$, then $z_x = z'_x = \omega$, if u contains exactly x occurrences of $\wr$, then $z'_x = max(z_x, \bar{p}_u) + 2\bar{q}_u$). Then, $G(x) \subseteq H(y)$ holds iff, for every z such that $z_x < z \leq z'_x$, $u[z+x+1] = \blacksquare \wedge |u[1, z+x]|_\wr = x$ implies $v[z+y+1] = \blacksquare \wedge |v[1, z+y]|_\wr = y$.*

Proof. Suppose that $G(x) \subseteq H(y)$. Let z be any integer such that $z_x < z \leq z'_x$, $u[z+x+1] = \blacksquare$, and $|u[1, z+x]|_\wr = x$. Clearly, $G_\perp(z) \subseteq G(x)$ holds, which implies $G_\perp(z) \subseteq H(y)$. Hence, since v represents H in terms of $G_\perp$, $v[z+y+1] = \blacksquare \wedge |v[1, z+y]|_\wr = y$ follows. Conversely, if $G(x) \not\subseteq H(y)$, then there exist $z \in \mathbb{N}$ such that $G_\perp(z) \subseteq G(x)$ and $G_\perp(z) \not\subseteq H(y)$. This implies that $u[z+x+1] = \blacksquare \wedge |u[1, z+x]|_\wr = x$ holds but $v[z+y+1] = \blacksquare \wedge |v[1, z+y]|_\wr = y$ does not. If u contains more than x occurrences of $\wr$ or $z \leq z'_x$, then the thesis follows trivially. Otherwise, if u contains exactly x occurrences of $\wr$ and $z > z'_x$ then we have $\widetilde{p}_u = x$, $\widetilde{q}_u = 0$, $z > \bar{p}_u + 2\bar{q}_u$, and $z > z_x + 2q_u$. Let k be the least integer such that $k\bar{q}_u \geq z - z'_x$ and let $z_1 = z - k\bar{q}_u$ and $z_2 = z - (k+1)\bar{q}$. By exploiting the definition of k, it is easy to verify that $z_x < z_1, z_2 \leq z'_x$ and $z_1, z_2 > \bar{p}_u$. Therefore, since p_u and q_u (resp., p_v and q_v) are the lengths of a prefix and a repeating pattern of u (resp., v), we have $u[z_1 + x + 1] = u[z_2 + x + 1] = u[z + x + 1] = \blacksquare$, $|u[1, z_1 + x]|_\wr = |u[1, z_2 + x]|_\wr = |u[1, z + x]|_\wr = x$, $v[z_1 + y + 1] = v[z_2 + y + 1] = v[z + y + 1]$, $|u[1, z_1 + y]|_\wr = |v[1, z + y]|_\wr - k\widetilde{q}_v$, and $|v[1, z_2 + y]|_\wr = |v[1, z + y]|_\wr - (k+1)\widetilde{q}_v$. We now distinguish between the following three cases: $v[z + y + 1] \neq \blacksquare$, $\widetilde{q}_v = 0 \wedge |v[1, z + y]|_\wr \neq y$, and $\widetilde{q}_v > 0$. In the first case, we immediately obtain $v[z_1 + y + 1] = v[z + y + 1] \neq \blacksquare$, from which the thesis follows. In the second case, we have $|v[1, z_1 + y]|_\wr = |v[1, z + y]|_\wr \neq y$, which, again, implies the thesis. In the third case, we have $|v[1, z_1 + y]|_\wr \neq |v[1, z_2 + y]|_\wr$, which implies that either $|v[1, z_1 + y]|_\wr \neq y$ or $|v[1, z_2 + y]|_\wr \neq y$ holds. ■

Below, we provide an algorithm that, given two granspecs α and β for the granularities G and H, computes the lengths p_u, q_u, p_v, and q_v of suitable prefixes and repeating patterns of u and v satisfying the hypothesis of Proposition 2.4.15.

CombinePrefixesAndRepeatingPatterns(α, β)

1: **let** $\alpha = (u_1, u_2)$
2: **let** $\beta = (v_1, v_2)$

3: $p_u \leftarrow |u_1|$
4: $q_u \leftarrow |u_2|$
5: $p_v \leftarrow |v_1|$
6: $q_v \leftarrow |v_2|$
7: $\bar{p}_u \leftarrow \mathit{CountOccurrences}(\alpha, 1, p_u, \{\blacksquare, \square\})$
8: $\bar{p}_v \leftarrow \mathit{CountOccurrences}(\beta, 1, p_v, \{\blacksquare, \square\})$
9: **if** $\bar{p}_v > \bar{p}_u$ **then**
10: $\quad p_u \leftarrow \mathit{Occurrence}(\alpha, \bar{p}_v, \{\blacksquare, \square\})$
11: **else if** $\bar{p}_u > \bar{p}_v$ **then**
12: $\quad p_v \leftarrow \mathit{Occurrence}(\beta, \bar{p}_u, \{\blacksquare, \square\})$
13: **end if**
14: $\bar{q}_u \leftarrow \mathit{CountOccurrences}(\alpha, p_u + 1, p_u + q_u, \{\blacksquare, \square\})$
15: $\bar{q}_v \leftarrow \mathit{CountOccurrences}(\beta, p_v + 1, p_v + q_v, \{\blacksquare, \square\})$
16: $q_u \leftarrow q_u * \mathit{lcm}(\bar{q}_u, \bar{q}_v)/\bar{q}_u$
17: $q_v \leftarrow q_v * \mathit{lcm}(\bar{q}_u, \bar{q}_v)/\bar{q}_v$
18: **return** (p_u, q_u, p_v, q_v)

In succession, we report an algorithm, based on Proposition 2.4.15, that computes the label of the granule of H (if exists) that includes the granule $G(x)$ of G.

IncludingGranule(α, β, x)

1: $(p_u, q_u, p_v, q_v) \leftarrow \mathit{CombinePrefixesAndRepeatingPatterns}(\alpha, \beta)$
2: $\bar{p}_u \leftarrow \mathit{CountOccurrences}(\alpha, 1, p_u, \{\blacksquare, \square\})$
3: $\bar{q}_u \leftarrow \mathit{CountOccurrences}(\alpha, p_u + 1, p_u + q_u, \{\blacksquare, \square\})$
4: $i_x \leftarrow \mathit{Occurrence}(\alpha, x, \{\wr\})$
5: $i'_x \leftarrow \mathit{Occurrence}(\alpha, x + 1, \{\wr\})$
6: $z_x \leftarrow \mathit{CountOccurrences}(\alpha, 1, i_x - 1, \{\blacksquare, \square\})$
7: $z'_x \leftarrow \mathit{CountOccurrences}(\alpha, 1, i'_x - 1, \{\blacksquare, \square\})$
8: **if** $z_x = \omega$ **then**
9: $\quad$ **return** $\perp$
10: **end if**
11: **if** $z'_x = \omega$ **then**
12: $\quad z'_x \leftarrow \mathit{max}(z_x, \bar{p}_u) + 2 * \bar{q}_u$
13: **end if**
14: $j \leftarrow \mathit{Occurrence}(\beta, z_x, \{\blacksquare, \wr\})$
15: $y \leftarrow \mathit{CountOccurrences}(\beta, 1, j, \{\wr\})$
16: **for** $z = z_x + 1 ... z'_x$ **do**
17: $\quad s \leftarrow \mathit{CountOccurrences}(\alpha, 1, z + x, \{\wr\})$
18: $\quad t \leftarrow \mathit{CountOccurrences}(\beta, 1, z + y, \{\wr\})$
19: $\quad$ **if** $\mathit{SymbolAt}(\alpha, z + x + 1) = \blacksquare$ **and** $s = x$ **then**
20: $\quad\quad$ **if** $\mathit{SymbolAt}(\beta, z + y + 1) \neq \blacksquare$ **or** $t \neq x$ **then**
21: $\quad\quad\quad$ **return** $\perp$
22: $\quad\quad$ **end if**
23: $\quad$ **end if**
24: **end for**
25: **return** y

It remains to provide an effective construction of a string-based representation of the granularity resulting from a combining operation.

Proposition 2.4.16 *Let u and v be two ultimately periodic words representing G and H in terms of $G_\perp$ and let p_u, q_u, p_v, and q_v be the lengths of some prefixes and repeating patterns of u and v satisfying $\bar{p}_u = \bar{p}_v$ and $\bar{q}_u = \bar{q}_v$. Furthermore, for every $x \in \mathbb{N}$, let*

- *i_x be the position in u of the x-th occurrence of $\wr$ (we assume $i_0 = 0$ and, if u contains less than x occurrences of $\wr$, then $i_x = \omega$),*
- *y_x be either the label of the (unique) granule of H that includes $G(x)$ (if it exists) or y_{x-1}, where y_{-1} is assumed to be 0,*
- $w_x = \begin{cases} \wr^{y_x - y_{x-1}} \cdot u[i_x + 1, i_{x+1} - 1] & \text{if } G(x) \subseteq H(y_x) \\ \square^{i_{x+1} - i_x - 1} & \text{if } G(x) \not\subseteq H(y_x). \end{cases}$

It follows that the word $w = w_0 \cdot w_1 \cdot w_2 \cdot \ldots$ represents the granularity $Combine(G, H)$ in terms of $G_\perp$. Moreover, w has a prefix of length $p_v + 2q_v$ and a repeating pattern of length either q_u or q_v, depending on whether $\widetilde{q}_u = 0$ or not.

Proof. Let $J = Combine(G, H)$. Suppose that $G_\perp(z) \subseteq J(y)$; we show that $w[z + y + 1] = \blacksquare$ and $|w[z + y]|_\wr = y$ hold. If $G_\perp(z) \subseteq J(y)$, then there is (a unique) $x \in \mathbb{N}$ such that $G_\perp(z) \subseteq G(x) \subseteq H(y)$ and, moreover, $u[z + x + 1] = \blacksquare$, $|u[z + x]|_\wr = x$, $v[z + y + 1] = \blacksquare$, and $|v[1, z + y]|_\wr = y$ hold. We prove the claim by induction on the label x (note that $y = y_x$). If $x = 0$, then $w_0 = \wr^y \cdot u[1, i_1 - 1]$ and hence $w[z+y+1] = w_0[z + y + 1] = u[z + 1] = u[z + x + 1] = \blacksquare$ and $|w[1, z + y]|_\wr = |w_0[1, z + y]|_\wr = y + |u[1, z]|_\wr = y$. If $x > 0$, then for every $0 \leq j \leq x$, $w_j = \wr^{y_j - y_{j-1}} \cdot u[i_j + 1, i_{j+1} - 1]$ holds and hence

$$\begin{aligned} |w_0 \cdot \ldots \cdot w_{x-1}| &= \sum_{0 \leq j < x} |w_j| = \sum_{0 \leq j < x} (y_j - y_{j-1}) + \sum_{0 \leq j < x} (i_{j+1} - i_j - 1) \\ &= (y_{x-1} - y_{-1}) + (i_x - i_0) + x = y_{x-1} + i_x - x. \end{aligned}$$

Consequently, $w[z+y+1] = w_x[z+y+1-y_{x-1}-i_x+x] = u[i_x+z+1-i_x+x] = u[z+x+1] = \blacksquare$ and $\big|w[1, z + y]\big|_\wr = |w_0 \cdot \ldots \cdot w_{x-1}|_\wr + \big|w_x[1, z + y - y_{x-1} - i_x + x]\big|_\wr = (y_{x-1} - y_{-1}) + (y - y_{x-1}) = y$. Conversely, suppose that $w[z + y + 1] = \blacksquare$ and $|w[1, z + y]|_\wr = y$ hold for some $z, y \in \mathbb{N}$. We have to prove that $G_\perp(z) \subseteq J(y)$, namely, that there is $x \in \mathbb{N}$ such that $G_\perp(z) \subseteq G(x)$ and $G(x) \subseteq J(y)$. Let i be the position in u of the $z+1$-th occurrence of non-separator symbols and let $x = |u[1, i]|_\wr$. Note that $i = z + x + 1$ and $i_x < i < i_{x+1}$ hold by definition. Moreover, since $w[z + y + 1] = \blacksquare$ by hypothesis, $y = y_x$ and $G(x) \subseteq J(y)$ follows trivially. Therefore,

we have

$$\begin{aligned} u[z+x+1] = u[i] = (u[i_x+1, i_{x+1}-1])[i-i_x] = w_x[(y_x - y_{x-1}) + i - i_x] \\ = w_x[y_x - y_{x-1} + (z+x+1) - i_x] \\ = w[(y_{x-1} + i_x - x) + y_x - y_{x-1} + z + x + 1 - i_x] \\ = w[z + y_x + 1] = w[z + y + 1] = \blacksquare, \\ \big|u[1, z+x]\big|_{\wr} = \big|u[1,i]\big|_{\wr} = x, \end{aligned}$$

which implies that $G_\perp(z) \subseteq G(x)$. This proves that w represents J in terms of $G_\perp$. It remains to show that w is an ultimately periodic word. If $\widetilde{q}_u = 0$, then, by using x as a shorthand for $\widetilde{p}_u$, we know that $i_x = \omega$ and w_x is either $\wr^{y_x - y_{x-1}} \cdot u[i_x+1, \omega]$ or $\square^\omega$, depending on whether $G(x) \subseteq H(y_x)$ or not. Therefore w has a prefix of length $|w_0 \cdot \ldots \cdot w_{x-1}| + (y_x - y_{x-1}) + \bar{q}_u = (y_{x-1} + i_x - x) + y_x - y_{x-1} + \bar{q}_u = y_x + (i_x - x) + \bar{q}_u \leq \widetilde{p}_v + \bar{p}_u + \bar{q}_u \leq p_v + 2q_v$ and a repeating pattern of length q_u. Otherwise, if $\widetilde{q}_u > 0$, then, by exploiting the periodicity of u and v, we can easily check that

i) $i_{x+\widetilde{q}_u} = i_x + q_u$, for every $x > \widetilde{p}_u$,
ii) $G(x + \widetilde{q}_u) \subseteq H(y + \widetilde{q}_v)$ iff $G(x) \subseteq H(y)$, for every $x > \widetilde{p}_u$ (cf. Proposition 2.4.15),
iii) $y_{x+\widetilde{q}_u} = y_x + \widetilde{q}_v$, for every $x > \widetilde{p}_u + \widetilde{q}_u$,
iv) $w_{x+\widetilde{q}_u} = w_x$, for every $x > \widetilde{p}_u + 2\widetilde{q}_u$.

Therefore, by using x as a shorthand for $\widetilde{p}_u + 2\widetilde{q}_u$, we know that w has a prefix of length $|w_0 \cdot \ldots \cdot w_{x-1}| = (y_{x-1}) + (i_x - x) = (\widetilde{p}_v + 2\widetilde{q}_v) + (\bar{p}_u + 2\bar{q}_u) \leq p_v + 2q_v$ and a repeating pattern of length $|w_x \cdot \ldots \cdot w_{x+\widetilde{q}_u-1}| = (y_{x+\widetilde{q}_u-1} + i_{x+\widetilde{q}} - x - \widetilde{q}_u) - (y_{x-1} + i_x - x) = \widetilde{q}_v + q_u - \widetilde{q}_u = (q_v - \bar{q}_v) + q_u - (q_u - \bar{q}_u) = q_v$. $\blacksquare$

The following algorithm computes a string-based representation of $\mathit{Combine}(H, G)$ from two given granspecs α and β representing the granularities G and H, respectively.

$\underline{\mathit{MapCombine}(\alpha, \beta)}$

```
(p_u, q_u, p_v, q_v) ← CombinePrefixesAndRepeatingPatterns(α, β)
p̃_u ← CountOccurrences(α, 1, p_u, {≀})
q̃_u ← CountOccurrences(α, p_u + 1, p_u + q_u, {≀})
p̃_v ← CountOccurrences(β, 1, p_v, {≀})
q̃_v ← CountOccurrences(β, p_v + 1, p_v + q_v, {≀})
p_w ← p_v + 2q_v
if q̃_u = 0 then
    q_w ← q_u
else
    q_w ← q_v
end if
x ← 0
i_0 ← 0
y_{-1} ← 0
w ← ε
while |w| < p_w + q_w do
    i_{x+1} ← Occurrence(α, x + 1, {≀})
```

18: $y_x \leftarrow \mathit{IncludingGranule}(\alpha, \beta, x)$
19: **if** $y_x \neq \perp$ **then**
20: **if** $i_{x+1} \neq \omega$ **then**
21: $w \leftarrow w \cdot \wr^{y_x - y_{x-1}} \cdot \mathit{Substring}(\alpha, i_x + 1, i_{x+1} - 1)$
22: **else**
23: $w \leftarrow w \cdot \wr^{y_x - y_{x-1}} \cdot \mathit{Substring}(\alpha, i_x + 1, i_x + (p_w + q_w - (|w| + y_x - y_{x-1})))$
24: **end if**
25: **else**
26: $y_x \leftarrow y_{x-1}$
27: **if** $i_{x+1} \neq \omega$ **then**
28: $w \leftarrow w \cdot \square^{i_{x+1} - i_x - 1}$
29: **else**
30: $w \leftarrow w \cdot \square^{p_w + q_w - |w|}$
31: **end if**
32: **end if**
33: $x \leftarrow x + 1$
34: **end while**
35: **return** $(w[1, p_w], w[p_w + 1, p_w + q_w])$

Anchored grouping operation

Recall that the anchored grouping operation $\mathit{AncoredGroup}(G, H)$ is defined on the grounds of the successor function succ_G, which maps a label $y \in \mathbb{N}$ of a granule of G to the label of the next granule of G (if exists, otherwise $\mathit{succ}_G(y) = \omega$). Given a string-based representation of the granularity G, the values of the function succ_G can be calculated as showed in the following proposition.

Proposition 2.4.17 *If u is an ultimately periodic word representing G in terms of $G_\perp$ and $y \in \mathbb{N}$ is the label of a granule of G, then $\mathit{succ}_G(y) = |u[1, i]|_\wr$, where i is the position in u of the first occurrence of $\blacksquare$ after the $y + 1$-th occurrence of $\wr$ (if such an occurrence does not exists, then $\mathit{succ}_G(y) = \omega$).*

Proof. We assume that $G(y)$ is not the last granule of G (the other case is trivial). Let i be the position in u of the first occurrence of $\blacksquare$ after the $y + 1$-th occurrence of $\wr$ and let y' be the number of occurrences of $\wr$ in $u[1, i]$. Clearly, $u[1, i] = \blacksquare$ and $|u[1, i - 1]|_\wr = y'$. Hence, by letting $z = i - 1 - y'$, we immediately obtain $G_\perp(z) \subseteq G(y')$. Moreover, y' is greater than y, which shows that $G(y')$ is a granule following $G(y)$. It remains to prove that y' is indeed the index of the first granule following $G(y)$. By contradiction, suppose that there exists $y < y'' < y'$ and $z \in \mathbb{N}$ such that $G_\perp(z) \subseteq G(y'')$. This would imply that $u[1, i'] = \blacksquare$ and $|u[1, i' - 1]|_\wr = y''$ hold, where $i' = z + y'' + 1$, which violates the definition of i. Hence we must conclude that $y' = \mathit{succ}_G(y)$. ∎

The following algorithm is based on Proposition 2.4.17. It receives in input a granspec α representing the granularity G and a label $y \in \mathbb{N}$ of a granule of G, and it

returns in output the label of the next granule of G (if it exists, otherwise it returns ω). Note that, if $y = -1$ is given in input to such an algorithm, then the output is the label of the first granule of G.

Succ(α, y)

```
1: if y ≥ 0 then
2:    i ← OccurrenceFirstAfter(α, y + 1, {■}, {≀})
3: else
4:    i ← OccurrenceFirst(α, {■})
5: end if
6: if i = ω then
7:    return ω
8: end if
9: y' ← CountOccurrences(α, 1, i, {≀})
10: return y'
```

We now provide a construction of a string-based representation of the granularity *AncoredGroup*(G, H).

Proposition 2.4.18 *Let u and v be two infinite words representing respectively G and H in terms of $G_\perp$, where G is a label-aligned sub-granularity of H and H is a full-integer labeled granularity. Further let y_i be the label of the $i+1$-th granule of G (if G has exactly i granules, then $y_i = \omega$) and let $v = v_0 \cdot v_1 \cdot v_2 \cdot ...$, where, for every $i \geq 0$, the factor v_i ends at the position of the y_i-th occurrence of $\wr$ (if $y_0 = 0$ then $v_0 = \varepsilon$, if $y_i = \omega$ then v_i is an infinite suffix of v). If we define $w_0 = \square^{|v_0|-y_0} \cdot \wr^{y_0}$ and, for every $i > 0$, w_i as the word obtained from v_i by moving all occurrences of $\wr$ in the right-most position (if v_i is infinite, then w_i is obtained from v_i by simply removing the occurrences of $\wr$), then the infinite word $w = w_0 \cdot w_1 \cdot w_2 \cdot ...$ represents the granularity AnchoredGroup(G, H) in terms of $G_\perp$. Moreover, if p_u and q_u (resp., p_v and q_v) are the lengths of a prefix and a repeating pattern of u (resp., v), then we can assume, without loss of generality, that $\widetilde{p}_v \geq \widetilde{p}_u$ and $\widetilde{q}_u$ divides $\widetilde{q}_v$ whenever $\widetilde{q}_u > 0$. Under these assumptions, the word w is ultimately periodic and it has a prefix of length $p_w = p_v + q_v$ and a repeating pattern of length either $q_w = q_v$ or $q_w = q_v - \widetilde{q}_v$, depending on whether $\widetilde{q}_u > 0 \ \wedge \ |u[p_u + 1, p_u + q_u]|_{\blacksquare} > 0$ holds or not.*

Proof. We preliminary notice the following facts (they can be easily proved by exploiting induction on j):

i) for every $j \in \mathbb{N}$, if $|v[1, j]|_\wr = x < y_0$, then $w[j - x + 1] = \square$ and $|w[1, j - x]|_\wr = 0$,
ii) for every $j \in \mathbb{N}$, if $v[j + 1] = \blacksquare$ and $|v[1, j]|_\wr = x \geq y_0$, then there exists $i \in \mathbb{N}$ such that $y_i \leq x < y_{i+1}$, $w[j - x + y_i + 1] = \blacksquare$ and $|w[1, j - x + y_i]|_\wr = y_i$,
iii) for every $j \in \mathbb{N}$, if $w[j + 1] = \blacksquare$ and $|w[1, j]|_\wr = y$, then there exist $i, x \in \mathbb{N}$ such that $y = y_i \leq x < y_{i+1}$, $v[j + x - y_i + 1] = \blacksquare$, and $|v[1, j + x - y_i]|_\wr = x$.

Now, let $J = AnchoredGroup(G, H)$ and assume that, for some $y, z \in \mathbb{N}$, $G_\perp(z) \subseteq J(y)$ holds. By definition of J, there is $x \in \mathbb{N}$ such that $y \leq x < succ_G(y)$, $G_\perp(z) \subseteq H(x)$, and $G(y) \neq \emptyset$. From $G_\perp(z) \subseteq H(x)$, we know that $v[z + x + 1] = \blacksquare$ and

$|v[1, z+x]|_{\wr} = x \geq y_0$, and hence, from the above properties, there is $i \in \mathbb{N}$ such that $y_i \leq x < y_{i+1}$ (thus $y_i = y$), $w[z+y+1] = w[(z+x) - x + y_i + 1] = \blacksquare$, and $|w[1, z+y]|_{\wr} = |w[1, (z+x) - x + y_i]|_{\wr} = y$. Conversely, assume that, for some $y, z \in \mathbb{N}$, $w[z+y+1] = \blacksquare$ and $|w[1, z+y]|_{\wr} = y$. Then, there exist $i, x \in \mathbb{N}$ satisfying $y = y_i \leq x < y_{i+1}$, $v[z+x+1] = v[(z+y) + x - y_i + 1] = \blacksquare$, and $|v[1, z+x]|_{\wr} = |v[1, (z+y) + x - y_i]|_{\wr} = x$, which proves that $G_{\perp}(z) \subseteq H(x)$. It remains to show that $G(y) \subseteq H(y)$, but this follows immediately from $G(y) \neq \emptyset$ (which holds since $y = y_i$ is the label of a granule of G) and from the fact that G is a label-aligned sub-granularity of H. This proves that w represents J in terms of $G_{\perp}$.

As for the proof that w is ultimately periodic, we distinguish between two cases: the case where G has only finitely many granules and the case where G has infinitely many granules. If G has a finite number $n > 1$ of granules (the cases $n = 0$ and $n = 1$ are trivial), then, by construction, $w = w_0 \cdot ... \cdot w_{n-1} \cdot w_n$, where w_n is an infinite word. Moreover, we can prove that the position in v of the y_{n-1}-th occurrence of $\wr$ does not exceed $p_v + q_v$. Indeed, if this were not the case, then, since $\widetilde{p}_v \geq \widetilde{p}_u$, the position in u of the y_{n-1}-th occurrence of $\wr$ would be greater than p_u. This would imply that $\widetilde{q}_u > 0$ and hence $\widetilde{q}_u$ divides $\widetilde{q}_v$. Moreover, since G is a label-aligned sub-granularity of H, the position in u of the y_{n-1}-th occurrence of $\wr$ would be greater than $p_u + q_u$, from which it follows that the repeating pattern $u[p_u + 1, p_u + q_u]$ of u contains at least one occurrence of $\wr$ and one occurrence of $\blacksquare$, against the hypothesis that G contains only finitely many granules. Therefore, we know that $|w_0 \cdot ... \cdot w_{n-1}| \leq p_v + q_v$. Furthermore, w_n is an ultimately periodic word having a prefix of length $max(p_v - |w_0 \cdot ... \cdot w_{n-1}|)$ and a repeating pattern of length $q_v - \widetilde{q}_v$. In this way we proved that, under the proviso that G has only finitely many granules (which implies $\widetilde{q}_u = 0 \vee |u[p_u + 1, p_u + q_u]|_{\blacksquare} = 0$), w is an ultimately periodic word having a prefix of length $p_v + q_v$ and a repeating pattern of length $q_v - \widetilde{q}_v$. We now consider the case of G having infinitely many granules. Note that, in such a case, the repeating pattern $u[p_u + 1, p_u + q_u]$ must contain at least one occurrence of $\wr$ and one occurrence of $\blacksquare$ ($\widetilde{q}_u > 0 \wedge |u[p_u + 1, p_u + q_u]|_{\blacksquare} > 0$). From the assumptions on p_u, q_u, p_v, and q_v, we also know that $\widetilde{p}_v \geq \widetilde{p}_u$ and $\widetilde{q}_u$ divides $\widetilde{q}_v$. Now, for every $j \geq 0$, we define

- s_j as the position in u of the j-th occurrence of $\wr$ (if $j = 0$, then $s_j = 0$),
- k_j as the position in u of the first occurrence of $\blacksquare$ after the position s_j,
- g_j as the number of occurrences in $u[1, k_j]$ of $\wr$ (note that $g_0, g_1, g_2, ...$ are all and only the labels of the granules of G, possibly repeated several times),
- t_j as the position in v of the g_i-th occurrence of $\wr$ (if $g_i = 0$, then $t_j = 0$).

The above series can be characterized in terms of generalized arithmetic progressions:

i) $s_j \leq p_u$ holds for every $j \leq \widetilde{p}_u$ and $s_{j+\widetilde{q}_u} = s_j + q_u$ holds for every $j > \widetilde{p}_u$,
ii) $k_j \leq p_u + q_u$ holds for every $j \leq \widetilde{p}_u$ and $k_{j+\widetilde{q}_u} = k_j + q_u$ holds for every $j > \widetilde{p}_u$,
iii) $g_j \leq \widetilde{p}_u + \widetilde{q}_u$ holds for every $j \leq \widetilde{p}_u$ and $g_{j+\widetilde{q}_u} = g_j + \widetilde{q}_u$ holds for every $j > \widetilde{p}_u$,
iv) since $\widetilde{p}_v \geq \widetilde{p}_u$ and $\widetilde{q}_v > 0$ is multiple of $\widetilde{q}_u$, $t_j \leq p_v + q_u$ holds for every $j \leq \widetilde{p}_v$ and $t_{j+\widetilde{q}_v} = t_j + q_v$ holds for every $j > \widetilde{p}_v$.

Moreover, the label set $Y = \{g_0, g_1, g_2, ...\}$ of G can be written as follows: $Y = Y_1 \cup \{y + h\widetilde{q}_u : y \in Y_2, h \in \mathbb{N}\}$, where $Y_1 = \{g_0, ..., g_{\widetilde{p}_v}\}$ and $Y_2 = \{g_{\widetilde{p}_v+1}, ..., g_{\widetilde{p}_v+\widetilde{q}_v}\}$.

This implies that for every $i > 0$, if j is the least integer such that $g_j = y_i$, then $v_i = v[t_{j-1} + 1, t_j]$. Therefore, for every $y > |Y_1|$, we have $v_{i+|Y_2|} = v_i$ and hence $w_{i+|Y_2|} = w_i$. Finally, since $max(Y_1) = g_{\widetilde{p}_v}$ and $max(Y_2) = g_{\widetilde{p}_v+\widetilde{q}_v}$, it turns out that w has a prefix of length $|w_0 \cdot ... \cdot w_{|Y_1|}| = |v_0 \cdot ... \cdot v_{|Y_1|}| = t_{\widetilde{p}_v} \leq p_v + q_v$ and a repeating pattern of length $|w_{|Y_1|+1} \cdot ... \cdot w_{|Y_1|+|Y_2|}| = |v_{|Y_1|+1} \cdot ... \cdot v_{|Y_1|+|Y_2|}| = t_{\widetilde{p}_v+\widetilde{q}_v} - t_{\widetilde{p}_v} = q_v$. ■

We now show how to accommodate the hypothesis of Proposition 2.4.18 on the prefixes and repeating patterns of u and v and how to compute the length of a prefix and the length of a repeating pattern of w. The following algorithm receives in input two granspecs α and β, with $[\![\alpha]\!] = u$ and $[\![\beta]\!] = v$, and it returns in output the length of a prefix and the length of a repeating pattern of w.

AnchoredGroupPrefixAndRepeatingPattern(α, β)

1: **let** $\alpha = (u_1, u_2)$
2: **let** $\beta = (v_1, v_2)$
3: $p_u \leftarrow |u_1|$
4: $q_u \leftarrow |u_2|$
5: $p_v \leftarrow |v_1|$
6: $q_v \leftarrow |v_2|$
7: $\widetilde{p}_u \leftarrow$ *CountOccurrences*$(\alpha, 1, p_u, \{\wr\})$
8: $\widetilde{p}_v \leftarrow$ *CountOccurrences*$(\beta, 1, p_v, \{\wr\})$
9: **if** $\widetilde{p}_v < \widetilde{p}_u$ **then**
10: $\quad p_v \leftarrow$ *Occurrence*$(\beta, \widetilde{p}_u, \{\wr\})$
11: **end if**
12: $p_w \leftarrow p_v + q_v$
13: $\widetilde{q}_u \leftarrow$ *CountOccurrences*$(\alpha, p_u + 1, p_u + q_u, \{\wr\})$
14: $\widetilde{q}_v \leftarrow$ *CountOccurrences*$(\beta, p_v + 1, p_v + q_v, \{\wr\})$
15: **if** $\widetilde{q}_u > 0$ **then**
16: $\quad q_v \leftarrow q_v * \widetilde{q}_u / gcd(\widetilde{q}_v, \widetilde{q}_u)$
17: $\quad \widetilde{q}_v \leftarrow lcm(\widetilde{q}_v, \widetilde{q}_u)$
18: **end if**
19: **if** $\widetilde{q}_u > 0$ **and** *CountOccurrences*$(\alpha, p_u + 1, p_u + q_u, \{\blacksquare\}) > 0$ **then**
20: $\quad q_w \leftarrow q_v$
21: **else**
22: $\quad q_w \leftarrow q_v - \widetilde{q}_v$
23: **end if**
24: **return** (p_w, q_w)

The following algorithm is based on Proposition 2.4.13 and it computes a granspec representing the granularity $J = AnchoredGroup(G, H)$, on the grounds of two string-based representations α and β for G and H, respectively.

MapAnchoredGroup(α, β)

1: $(p_w, q_w) \leftarrow$ *AnchoredGroupPrefixAndRepeatingPattern*(α, β)
2: $x \leftarrow 0$

3: $i \leftarrow 0$
4: $j \leftarrow 1$
5: $y_0 \leftarrow Succ(\alpha, -1)$
6: $w \leftarrow \varepsilon$
7: **while** $|w| < p_w + q_w$ **do**
8: **if** $SymbolAt(\beta, j) \in \{\blacksquare, \square\}$ **then**
9: **if** $x < y_0$ **then**
10: $w \leftarrow w \cdot \square$
11: **else**
12: $w \leftarrow w \cdot SymbolAt(\alpha, j)$
13: **end if**
14: **else**
15: $x \leftarrow x + 1$
16: **if** $x = y_i$ **then**
17: **if** $i = 0$ **then**
18: $w \leftarrow w \cdot \wr^{y_0}$
19: **else**
20: $w \leftarrow w \cdot \wr^{y_i - y_{i-1}}$
21: **end if**
22: $i \leftarrow i + 1$
23: $y_i \leftarrow Succ(\alpha, y_{i-1})$
24: **end if**
25: **end if**
26: $j \leftarrow j + 1$
27: **end while**
28: **return** $(w[1, p_w], w[p_w + 1, p_w + q_w])$

Subset operation

Proposition 2.4.19 *Let $m, n \in \mathbb{N}$ such that $m \leq n$ and let $u = u_1 \cdot u_2 \cdot u_3$ be an ultimately periodic word representing G in terms $G_\perp$, where*

- *u_1 ends at the position in u of the m-th occurrence of $\wr$ in u (if $m = 0$, then $u_1 = \varepsilon$, if u contains less than m occurrences of $\wr$, then $u_1 = u$ and $u_2 = u_3 = \varepsilon$),*
- *u_2 ends at the position in u of the $n+1$-th occurrence of $\wr$ in u (if u contains less than n occurrences of $\wr$, then u_2 is a suffix of u and $u_3 = \varepsilon$).*

Then, the word $v = \square^k \cdot \wr^m \cdot u_2 \cdot \square^\omega$ ($v = \square^\omega$ if $u_1 = u$), where $k = |u_1|_{\blacksquare,\square}$, represents the granularity $Subset_m^n(G)$ in terms of $G_\perp$.

Proof. Let $H = Subset_m^n(G)$. For every $m \leq x \leq n$, we have $H(x) = G(x)$ and hence, for every $z \in \mathbb{N}$, $G_\perp(z) \subseteq H(x)$ iff $G_\perp(z) \subseteq G(x)$, iff $u[z+x+1] = \blacksquare \wedge |u[1, z+x]|_\wr = x$, iff $v[z+x+1] = \blacksquare \ \wedge \ |v[1, z+x]|_\wr = x$. Moreover, for every $x < m$ and every $x > n$, $H(x) = \emptyset$ holds and $|v[1, z+x]|_\wr = x$ implies $v[z+x+1] = \square$. This shows that v represents H in terms of $G_\perp$. ■

The following algorithm receives in input a granspec α representing G in terms of $G_\perp$ and two parameters $m, n \in \mathbb{Z}$, and it returns in output a granspecs β representing $H = Subset^n_m(G)$ in terms of $G_\perp$.

MapSubset(α, m, n)

let $\alpha = (u_1, u_2)$
$p_u \leftarrow |u_1|$
$q_u \leftarrow |u_2|$
$i \leftarrow Occurrence(\alpha, m, \{\wr\})$
$j \leftarrow Occurrence(\alpha, n+1, \{\wr\})$
$k \leftarrow CountOccurrences(\alpha, 1, i, \{\blacksquare, \square\})$
if $i = \omega$ **then**
  $\beta \leftarrow (\varepsilon, \square)$
else if $j = \omega$ **then**
  if $i \leq p_u$ **then**
    $\beta \leftarrow (\wr^m \cdot \square^k \cdot u_1[i+1, p_u], u_2)$
  else
    $\beta \leftarrow (\wr^m \cdot \square^k, u_2[i - p_u, q_u] \cdot u_2[1, i - p_u - 1])$
  end if
else
  $\beta \leftarrow (\wr^m \cdot \square^k \cdot Substring(\alpha, i+1, j), \square)$
end if
return β

Selection operations

Here we provide an algorithm that maps two given granspecs representing G and H and two parameters k, l to a granspec representing $SelectDown^l_k(G, H)$. The algorithms for the other selection operations *SelectUp* and *SelectByIntersect* are slight variations of this one. For the sake of simplicity, we further restrict ourselves to the case of k being a positive natural number (the case $k < 0$ can be handled in a similar way).

Proposition 2.4.20 *Let $u = u_0 \cdot u_1 \cdot u_2 \cdot ...$ and v be two ultimately periodic words that represent respectively G and H in terms of $G_\perp$, where each factor u_x of u ends at the position of the $x+1$-th occurrence of $\wr$ in u (if u contains x occurrences of $\wr$, then u_x is an infinite suffix of u). For every $y \in \mathbb{N}$, we denote by X_y the set $\{x \in \mathbb{N} : \emptyset \neq G(x) \subseteq H(y)\}$ and for every $x \in \mathbb{N}$, we define*

$$w_x = \begin{cases} u_x & \text{if } \exists\, y \in \mathbb{N}.\ x \in \Delta^l_k(X_y) \\ \square^{|u_x|-1} \cdot \wr & \text{otherwise.} \end{cases}$$

Clearly, the word $w = w_0 \cdot w_1 \cdot x_2 \cdot ...$ represents the granularity $SelectDown^l_k(G, H)$ in terms of $G_\perp$. Moreover, if p_u and q_u (resp., p_v and q_v) are the lengths of a prefix and a repeating pattern of u (resp., v), then we can assume, without loss of generality,

that $\bar{p}_u = \bar{p}_v$ and $\bar{q}_u = \bar{q}_v$. Under these assumptions, the word w is ultimately periodic and it has a prefix of length $p_w = p_u + q_u$ and a repeating pattern of length $q_w = q + u$.

Proof. Let $J = \mathit{SelectDown}_k^l(G, H)$. For every $z, x \in \mathbb{N}$, $G_\perp(z) \subseteq J(x)$ iff $G_\perp(z) \subseteq G(x)$ and there exists $y \in \mathbb{N}$ such that $x \in \Delta_k^l(X_y)$, iff $u[z + x + 1] = \blacksquare \ \wedge \ |u[1, z + x]|_\wr = x$ and $\exists\, y \in \mathbb{N}.\ x \in \Delta_k^l(S_y)$. Thus, by definition of w, we immediately obtain that $G_\perp(z) \subseteq J(x)$ iff $w[z + x + 1] = \blacksquare \ \wedge \ |w[1, z + x]|_\wr = x$ and this proves that w represents the granularity J in terms of $G_\perp$.
It remains to show that w is an ultimately periodic word. Let p_u, q_u, p_v, and q_v be the lengths of some prefixes and repeating patterns of u and v satisfying the hypothesis of the proposition. We distinguish between the following two cases: $\widetilde{q}_u = 0$ and $\widetilde{q}_u > 0$. If $\widetilde{q}_u = 0$, then the last factor $u_{\widetilde{p}_u}$ of u is an ultimately periodic word having a prefix of length $p_u - |u_0 \cdot ... \cdot u_{\widetilde{p}-1}|$ and a repeating pattern of length q_u. This implies that w is an ultimately periodic word having a prefix of length $p_u \leq p_u + q_u$ and a repeating pattern q_u. Now, assume that $\widetilde{q}_u > 0$ and consider the infinite sequence of sets $X_0, X_1, X_2, ...$, where $X_y = \{x \in \mathbb{N} : \emptyset \neq G(x) \subseteq H(y)\}$. From the periodicity of the word u representing G, we know that for every $x > \widetilde{p}_u$, $G(x) \neq \emptyset$ iff $G(x + \widetilde{q}_u) \neq \emptyset$. Moreover, from Proposition 2.4.15 and from the hypothesis on the prefixes and repeating patterns of u and v, for every $x > \widetilde{p}_u$ and for every $u > \widetilde{p}_v$, $G(x) \subseteq H(y)$ iff $G(x + \widetilde{q}_u) \subseteq H(y + \widetilde{q}_v)$. Therefore, for every $y > \widetilde{p}_v$, $X_{y+\widetilde{q}_v} = \{x + \widetilde{q}_u : x \in X_y\}$ follows and hence $w_{x+\widetilde{q}_u} = w_x$, for every $x > \widetilde{p}_u$. Therefore, we can conclude that w has a prefix of length $|w_0 \cdot ... \cdot w_{\widetilde{p}_u}| = |u_0 \cdot ... \cdot u_{\widetilde{p}_u}| \leq p_u + q_u$ and a repeating pattern of length $|w_{\widetilde{p}_u+1} \cdot ... \cdot w_{\widetilde{p}_u+\widetilde{q}_u}| = |u_{\widetilde{p}_u+1} \cdot ... \cdot u_{\widetilde{p}_u+\widetilde{q}_u}| = q_u$. ■

On the grounds of Proposition 2.4.20, we can devise the following algorithm. It receives in input two granspecs α and β representing G and H in terms of $G_\perp$ and two parameters $k, l > 0$ and it returns in output a granspec representing the granularity $\mathit{SelectDown}_k^l(G, H)$. The computation of the lengths of the prefixes and repeating patterns of $u = [\![\alpha]\!]$ and $v = [\![\beta]\!]$ is demanded to the procedure *CombinePrefixesAndRepeatingPatterns*, which has been introduced in Section 2.3.1. Moreover, the procedure *IncludingGranule* is used to determine which granules of G are included in some granule of H.

MapSelectDown(α, β, k, l)

1: $(p_u, q_u, p_v, q_v) \leftarrow \mathit{CombinePrefixesAndRepeatingPatterns}(\alpha, \beta)$
2: $\widetilde{p}_u \leftarrow \mathit{CountOccurrences}(\alpha, 1, p_u, \{\wr\})$
3: $\widetilde{q}_u \leftarrow \mathit{CountOccurrences}(\alpha, p_u + 1, p_u + q_u, \{\wr\})$
4: $\widetilde{p}_v \leftarrow \mathit{CountOccurrences}(\beta, 1, p_v, \{\wr\})$
5: $\widetilde{q}_v \leftarrow \mathit{CountOccurrences}(\beta, p_v + 1, p_v + q_v, \{\wr\})$
6: $p_w \leftarrow p_u + q_u$
7: $q_w \leftarrow q_u$
8: $x \leftarrow 0$
9: $i_0 \leftarrow 0$
10: $y_{-1} \leftarrow \perp$
11: $w \leftarrow \varepsilon$

12: **while** $|w| < p_w + q_w$ **do**
13: $i_{x+1} \leftarrow \mathit{Occurrence}(\alpha, x+1, \{\wr\})$
14: **if** $\mathit{CountOccurrences}(\alpha, i_x + 1, i_{x+1} - 1, \{\blacksquare\}) > 0$ **then**
15: $y_x \leftarrow \mathit{IncludingGranule}(\alpha, \beta, x)$
16: **else**
17: $y_x \leftarrow \perp$
18: **end if**
19: **if** $y_x \neq \perp$ **and** $y_x \neq y_{x-1}$ **then**
20: $n \leftarrow 1$
21: **else if** $y_x = y_{x-1}$ **then**
22: $n \leftarrow n + 1$
23: **end if**
24: **if** $y_x \neq \perp$ **and** $k \leq n < k + l$ **then**
25: **if** $i_{x+1} \neq \omega$ **then**
26: $w \leftarrow w \cdot \mathit{Substring}(\alpha, i_x + 1, i_{x+1})$
27: **else**
28: $w \leftarrow w \cdot \mathit{Substring}(\alpha, i_x + 1, i_x + (p_w + q_w - |w|))$
29: **end if**
30: **else**
31: **if** $i_{x+1} \neq \omega$ **then**
32: $w \leftarrow w \cdot \square^{i_{x+1} - i_x} \cdot \wr$
33: **else**
34: $w \leftarrow w \cdot \square^{p_w + q_w - |w|}$
35: **end if**
36: **end if**
37: $x \leftarrow x + 1$
38: **end while**
39: **return** $(w[1, p_w], w[p_w + 1, p_w + q_w])$

Set operations

In the following, we show how to map the union operation of Calendar Algebra into the string-based formalism (the difference and the intersection operations can be handled in a similar way). For the sake of simplicity, we restrict to granspecs in aligned form (recall that such a form can be computed in quadratic time by exploiting the algorithm sketched in Section 2.4.1). Note that the construction described below may result in an ultimately periodic word w containing only finitely many occurrences of non-separator symbols. In such a case, in order to let w be a well-defined granspec, we need to replace the repeating pattern $\wr$ with $\square$.

Proposition 2.4.21 *Let u and v be two ultimately periodic words in aligned form representing G and H in terms of $G_\perp$, where G and H are label-aligned sub-granularities of a common granularity I. Furthermore, for every $x \in \mathbb{N}$, let*

- *i_x (resp., j_x) be the number of occurrences in u (resp., v) of non-separator symbols before $x+1$-th occurrence of $\wr$, if u (resp., v) contains less than $x+1$ occurrences*

of $\wr$*, then we assume that* i_x *(resp.,* j_x*) is* 1 *plus the number of occurrences in* u *(resp.,* v*) of non-separator symbols before the last occurrence of* $\blacksquare$ *(possibly* $i_x = \omega$ *if* u *contains infinitely many occurrences of* $\blacksquare$ *and* $j_x = \omega$ *if* v *contains infinitely many occurrences of* $\blacksquare$*),*

- $k_x = \begin{cases} i_x & \text{if } i_x > i_{x-1}, \\ j_x & \text{if } i_x = i_{x-1} \text{ and } j_x > j_{x-1}, \\ k_{x-1} & \text{otherwise,} \end{cases}$

where i_{-1}*,* j_{-1}*, and* k_{-1} *are assumed to be* 0*,*

- $w_x = \begin{cases} u[k_{x-1}+x+1, k_x+x]\cdot\wr & \text{if } i_x > i_{x-1}, \\ v[k_{x-1}+x+1, k_x+x]\cdot\wr & \text{if } i_x = i_{x-1} \text{ and } j_x > j_{x-1}, \\ \wr & \text{otherwise.} \end{cases}$

The word $w = w_0 \cdot w_1 \cdot w_2 \cdot \ldots$ *represents the granularity* $G \cup H$ *in terms of* $G_\perp$*. Moreover, if* p_u *and* q_u *(resp.,* p_v *and* q_v*) are the lengths of a prefix and a repeating pattern of* u *(resp.,* v*), then we can assume, without loss of generality, that* $\bar{p}_u = \bar{p}_v$ *and* $\bar{q}_u = \bar{q}_v$*. Under these assumptions,* w *is an ultimately periodic word satisfying the following conditions:*

i) if $\widetilde{q}_u > 0$ *and* $\widetilde{q}_v > 0$*, then* $\widetilde{q}_u = \widetilde{q}_v$*,* $q_u = q_v$*, and* w *has a prefix of length* $max(p_u, p_v) + q_u$ *and a repeating pattern of length* q_u*,*

ii) if $\widetilde{q}_u > 0$ *and* $\widetilde{q}_v = 0$*, then* $|v[p_v+1, p_v+q_v]|_\blacksquare = 0$ *and* w *has a prefix of length* $max(p_u, p_v) + q_u$ *and a repeating pattern of length* q_u*,*

iii) if $\widetilde{q}_v > 0$ *and* $\widetilde{q}_u = 0$*, then* $|u[p_u+1, p_u+q_u]|_\blacksquare = 0$ *and* w *has a prefix of length* $max(p_u, p_v) + q_v$ *and a repeating pattern of length* q_v*,*

iv) if $\widetilde{q}_u = \widetilde{q}_v = 0$ *and* $|u[p_u+1, p_u+q_u]|_\blacksquare > 0$*, then* $\widetilde{p}_v \leq \widetilde{p}_u$*,* $p_v \leq p_u$*, and* w *has a prefix of length* p_u *a repeating pattern of length* q_u*,*

v) if $\widetilde{q}_u = \widetilde{q}_v = 0$*,* $|u[p_u+1, p_u+q_u]|_\blacksquare = 0$ *and* $|v[p_v+1, p_v+q_v]|_\blacksquare > 0$*, then* $\widetilde{p}_u \leq \widetilde{p}_v$*,* $p_u \leq p_v$*, and* w *has a prefix of length* p_v *a repeating pattern of length* q_v*,*

vi) if $\widetilde{q}_u = \widetilde{q}_v = 0$ *and* $|u[p_u+1, p_u+q_u]|_\blacksquare = |v[p_v+1, p_v+q_v]|_\blacksquare = 0$*, then* w *has a prefix of length* $max(p_v, q_v)$ *and a repeating pattern of length* 1*.*

Proof. First of all, notice that for every $x \in \mathbb{N}$, $|w_x|_\wr = 1$ and $|w_0 \cdot \ldots \cdot w_x| = k_x + x + 1$ (this is easy to prove by induction on x). Now, let $J = G \cup H$ and suppose that, for some given $z, x \in \mathbb{N}$, $w[z+x+1] = \blacksquare$ and $|w[1, z+x]|_\wr = x$ hold. We distinguish between two cases: $i_x > i_{x-1}$ and $i_x = i_{x-1} \wedge j_x > j_{x-1}$ (note the case $i_x = i_{x-1} \wedge j_x = j_{x-1}$ is not possible). If $i_x > i_{x-1}$, then $k_{x-1} + x + 1 \leq z + x \leq k_x + x$. Moreover, from the above properties, we obtain $\blacksquare = w[z+x+1] = \big(u[k_{x-1}+x+1, k_x+x]\big)[z+x+1-(k_{x-1}+x+1)] = u[z+x+1]$, $|u[k_{x-1}+x+1, z+x]|_\wr = 0$, and, since $i_x > i_{x-1}$, $|u[1, k_{x-1}+x]|_\wr = x$. This shows that $G_\perp(z) \subseteq G(x)$ and hence $G_\perp(z) \subseteq J(x)$. The case $i_x = i_{x-1} \wedge j_x > j_{x-1}$ can be dealt in a similar way, with the additional property that $i_x = i_{x-1}$ implies $G(x) = \emptyset$. This shows that $G_\perp(z) \subseteq J(x)$. As for the converse implication, suppose that $G_\perp(z) \subseteq J(x)$. By definition, either $G_\perp(z) \subseteq G(x)$ or $G(x) = \emptyset \wedge G_\perp(z) \subseteq H(x)$ holds. In the former case, we have $u[z+x+1] = \blacksquare \wedge |u[1, z+x]|_\wr = x$. Moreover, since $G(x) \neq \emptyset$, $i_x > i_{x-1}$ follows and hence $w[z+x+1] = \big(u[k_{x-1}+x+1, k_x+x]\big)[z+x+1-(k_{x-1}+x+1)] = u[z+x+1] = \blacksquare$

and $|w[1, z+x]|_{\wr} = |u[1, k_{x-1}+x]|_{\wr} + |u[k_{x-1}+x+1, z+x]|_{\wr} = x$. In the latter case, we have $v[z+x+1] = \blacksquare \wedge |v[1, z+x]|_{\wr} = x$. Moreover, since u is in aligned form, $i_x = i_{x-1}$ holds, from which it follows $w[z+x+1] = \big(v[k_{x-1}+x+1, k_x+x]\big)[z+x+1-(k_{x-1}+x+1)] = v[z+x+1] = \blacksquare$ and $|w[1, z+x]|_{\wr} = |v[1, k_{x-1}+x]|_{\wr} + |v[k_{x-1}+x+1, z+x]|_{\wr} = x$.

It remains to prove that w is an ultimately periodic word and that the conditions i)-vi) hold. Consider the first condition, and assume that $\widetilde{q}_u > 0$ and $\widetilde{q}_v > 0$. Note that $\widetilde{q}_u = \widetilde{q}_v$ and hence $q_u = q_v$ necessarily follows (otherwise, since both G and H contains infinitely many granules, G and H would not be label-aligned sub-granularities of the same granularity I). Moreover, by exploiting the periodicity of u and v, it is easy to verify that

- $i_{x+\widetilde{q}_u} = i_x + \bar{q}_u$ holds for every $x \geq \widetilde{p}_u$,
- $i_{x+\widetilde{q}_u} > i_{x-1+\widetilde{q}_u}$ iff $i_x > i_{x-1}$, for every $x > \widetilde{p}_u$,
- $j_{x+\widetilde{q}_u} = j_x + \bar{q}_u$ holds for every $x \geq \widetilde{p}_v$ (recall that $\widetilde{q}_u = \widetilde{q}_v$ and $\bar{q}_u = \bar{q}_v$),
- $j_{x+\widetilde{q}_u} > j_{x-1+\widetilde{q}_u}$ iff $j_x > j_{x-1}$, for every $x > \widetilde{p}_v$,
- $k_{x+\widetilde{q}_u} = k_x + \bar{q}_u$ holds for every $x \geq max(\widetilde{p}_u, \widetilde{p}_v)$,
- $k_x + x + 1 \geq \bar{p}_u + max(\widetilde{p}_u, \widetilde{p}_v) + 1 = max(p_u, p_v) + 1$ holds for every $x \geq max(\widetilde{p}_u, \widetilde{p}_v)$ (note that, if $k_x = k_{x-1}$, then $i_{x-1} = j_{x-1} = k_{x-1}$),
- $w_{x+\widetilde{q}_u} = w_x$ holds for every $x > max(\widetilde{p}_u, \widetilde{p}_v)$.

Therefore, if let $\widetilde{p} = max(\widetilde{p}_u, \widetilde{p}_v)$, w is an ultimately periodic word having a prefix of length $|w_0 \cdot \ldots \cdot w_{\widetilde{p}}| = k_{\widetilde{p}} + \widetilde{p} + 1 \leq max(p_u, p_v) + q_u$ and a repeating pattern of length $|w_{\widetilde{p}+1} \cdot \ldots \cdot w_{\widetilde{p}+\widetilde{q}_u}| = \big(k_{\widetilde{p}+\widetilde{q}_u} + \widetilde{p} + \widetilde{q}_u + 1\big) - \big(k_{\widetilde{p}} + \widetilde{p} + 1\big) = \bar{q}_u + \widetilde{q}_u = q_u$. As for the other conditions, they can be easily proved by using similar arguments, after noticing that

ii) if $\widetilde{q}_u > 0$ and $\widetilde{q}_v = 0$, then $|v[p_v + 1, p_v + q_v]|_{\blacksquare} = 0$ follows, since, otherwise, $G(x) \cap H(\widetilde{p}_v) \neq \emptyset$ would hold for infinitely many $x > \widetilde{p}_u$, thus contradicting the hypothesis that G and H are labeled-aligned sub-granularities of I,

iii) if $\widetilde{q}_v > 0$ and $\widetilde{q}_u = 0$, then $|u[p_u + 1, p_u + q_u]|_{\blacksquare} = 0$ follows, since, otherwise, $H(x) \cap G(\widetilde{p}_u) \neq \emptyset$ would hold for infinitely many $x > \widetilde{p}_v$, thus contradicting the hypothesis that G and H are labeled-aligned sub-granularities of I,

iv) if $\widetilde{q}_u = \widetilde{q}_v = 0$ and $|u[p_u + 1, p_u + q_u]|_{\blacksquare} > 0$, then $\widetilde{p}_v \leq \widetilde{p}_u$, since, otherwise, $H(x) \cap G(\widetilde{p}_u) \neq \emptyset$ would hold for some $x \geq \widetilde{p}_v > \widetilde{p}_u$, thus contradicting the hypothesis that G and H are labeled-aligned sub-granularities of I,

v) if $\widetilde{q}_u = \widetilde{q}_v = 0$ and $|v[p_v + 1, p_v + q_v]|_{\blacksquare} > 0$, then $\widetilde{p}_u \leq \widetilde{p}_v$, since, otherwise, $G(x) \cap H(\widetilde{p}_v) \neq \emptyset$ would hold for some $x \geq \widetilde{p}_u > \widetilde{p}_v$, thus contradicting the hypothesis that G and H are labeled-aligned sub-granularities of I,

vi) if $\widetilde{q}_u = \widetilde{q}_v = 0$ and $|v[p_v + 1, p_v + q_v]|_{\blacksquare} > 0$, then $\widetilde{p}_u \leq \widetilde{p}_v$.

$\blacksquare$

The following algorithm receives in input two granspecs α and β representing G and H in terms of $G_{\perp}$, where G and H are label-aligned sub-granularities of a common granularity I, and it returns in output a granspec representing the granularity $G \cup H$. Like in the case of selecting operations, the computation of the lengths of the prefixes and repeating patterns of $u = [\![\alpha]\!]$ and $v = [\![\beta]\!]$ is demanded to the procedure

CombinePrefixesAndRepeatingPatterns.

MapUnion(α, β)

1: $(p_u, q_u, p_v, q_v) \leftarrow \mathit{CombinePrefixesAndRepeatingPatterns}(\alpha, \beta)$
2: $\widetilde{p}_u \leftarrow \mathit{CountOccurrences}(\alpha, 1, p_u, \{\wr\})$
3: $\widetilde{q}_u \leftarrow \mathit{CountOccurrences}(\alpha, p_u + 1, p_u + q_u, \{\wr\})$
4: $\widetilde{p}_v \leftarrow \mathit{CountOccurrences}(\beta, 1, p_v, \{\wr\})$
5: $\widetilde{q}_v \leftarrow \mathit{CountOccurrences}(\beta, p_v + 1, p_v + q_v, \{\wr\})$
6: $r_u \leftarrow \mathit{CountOccurrences}(\alpha, p_u + 1, p_u + q_u, \{\blacksquare\})$
7: $r_v \leftarrow \mathit{CountOccurrences}(\alpha, p_v + 1, p_v + q_v, \{\blacksquare\})$
8: **if** $\widetilde{q}_u > 0$ **and** $\widetilde{q}_v > 0$ **then**
9: $\quad p_w \leftarrow \mathit{max}(p_u, p_v) + q_u$
10: $\quad q_w \leftarrow q_u$
11: **else if** $\widetilde{q}_u > 0$ **and** $\widetilde{q}_v = 0$ **then**
12: $\quad p_w \leftarrow \mathit{max}(p_u, p_v) + q_u$
13: $\quad q_w \leftarrow q_u$
14: **else if** $\widetilde{q}_v > 0$ **and** $\widetilde{q}_u = 0$ **then**
15: $\quad p_w \leftarrow \mathit{max}(p_u, p_v) + q_v$
16: $\quad q_w \leftarrow q_v$
17: **else if** $\widetilde{q}_u = \widetilde{q}_v = 0$ **and** $r_u > 0$ **then**
18: $\quad p_w \leftarrow p_u$
19: $\quad q_w \leftarrow q_u$
20: **else if** $\widetilde{q}_u = \widetilde{q}_v = 0$ **and** $r_v > 0$ **then**
21: $\quad p_w \leftarrow p_v$
22: $\quad q_w \leftarrow q_v$
23: **else**
24: $\quad p_w \leftarrow \mathit{max}(p_u, p_v)$
25: $\quad q_w \leftarrow 1$
26: **end if**
27: $x \leftarrow 0$
28: $i_{-1} \leftarrow 0$
29: $j_{-1} \leftarrow 0$
30: $k_{-1} \leftarrow 0$
31: $w \leftarrow \varepsilon$
32: **while** $|w| < p_w + q_w$ **do**
33: $\quad s \leftarrow \mathit{Occurrence}(\alpha, x + 1, \{\wr\})$
34: $\quad$ **if** $s = \omega$ **then**
35: $\quad\quad s \leftarrow \mathit{OccurrenceLastBefore}(\alpha, x + 1, \{\blacksquare\}, \{\wr\})$
36: $\quad$ **end if**
37: $\quad i_x \leftarrow \mathit{CountOccurrences}(\alpha, 1, s, \{\blacksquare, \square\})$
38: $\quad t \leftarrow \mathit{Occurrence}(\beta, x + 1, \{\wr\})$
39: $\quad$ **if** $t = \omega$ **then**
40: $\quad\quad t \leftarrow \mathit{OccurrenceLastBefore}(\beta, x + 1, \{\blacksquare\}, \{\wr\})$
41: $\quad$ **end if**
42: $\quad j_x \leftarrow \mathit{CountOccurrences}(\beta, 1, t, \{\blacksquare, \square\})$

43: **if** $i_x > i_{x-1}$ **then**
44: $k_x \leftarrow i_x$
45: **if** $k_x \neq \omega$ **then**
46: $w \leftarrow w \cdot \mathit{Substring}(\alpha, k_{x-1} + x + 1, k_x + x) \cdot \wr$
47: **else**
48: $w \leftarrow w \cdot \mathit{Substring}(\alpha, k_{x-1} + x + 1, k_{x-1} + x + 1 + (p_w + q_w - |w|))$
49: **end if**
50: **else if** $j_x > j_{x-1}$ **then**
51: $k_x \leftarrow j_x$
52: **if** $k_x \neq \omega$ **then**
53: $w \leftarrow w \cdot \mathit{Substring}(\beta, k_{x-1} + x + 1, k_x + x) \cdot \wr$
54: **else**
55: $w \leftarrow w \cdot \mathit{Substring}(\beta, k_{x-1} + x + 1, k_{x-1} + x + 1 + (p_w + q_w - |w|))$
56: **end if**
57: **else**
58: $k_x \leftarrow k_{x-1}$
59: $w \leftarrow w \cdot \wr$
60: **end if**
61: $x \leftarrow x + 1$
62: **end while**
63: **if** $w[p_w + 1, p_w + q_w] = \wr$ **then**
64: $w[p_w + 1, p_w + q_w] \leftarrow \square$
65: **end if**
66: **return** $(w[1, p_w], w[p_w + 1, p_w + q_w])$

2.5 From strings to automata

The idea of viewing granularities as ultimately periodic words naturally connects time granularity to the fields of formal languages and automata, because any regular ω-language (i.e., a language recognized by a sequential Büchi automaton) is uniquely determined by its ultimately periodic words [16]. An automaton-based approach to time granularity, which generalizes the string-based one in several respects, was originally proposed in [39]. The basic idea underlying the automaton-based approach to time granularity is as follows: we take a sequential Büchi automaton M recognizing a *single* word $w \in \{\blacksquare, \square, \wr\}^\omega$ (hence the name single-string automaton) and we say that M represents the granularity G iff w represents G. Below, we formally define single-string automata.

Definition 2.5.1 *A* single-string automaton *(*SSA *for short) is a quadruple* $M = (S, \Sigma, \delta, s_0)$*, where*

- S *is a finite set of* states*,*
- Σ *is a finite* alphabet *(e.g.,* $\{\blacksquare, \square, \wr\}$*),*
- δ *is a (total)* transition function *from* S *to* $\Sigma \times S$*,*
- $s_0 \in S$ *is a distinguished* initial state*.*

A *run* of an SSA $M = (S, \Sigma, \delta, s_0)$ is a pair $(\mathbf{s}, w) \in S^\omega \times \Sigma^\omega$ such that $\mathbf{s}[1] = s_0$ and for every $i > 0$, $\delta\big(\mathbf{s}[i]\big) = \big(w[i], \mathbf{s}[i+1]\big)$. Since δ is a single-valued function, the run of M is uniquely determined. Thus, given the SSA M, we say that w is *the* infinite word recognized by M if its (unique) run is of the form $(\mathbf{s}, w)$. It immediate to see that such a word w is an ultimately periodic word. Conversely, for every ultimately periodic word, there is an SSA recognizing it. Thus, SSA capture all and only the finite or ultimately periodic granularities, namely, those granularities that can be represented either by granspecs or by Calendar Algebra. Figure 2.12 depicts an SSA representing the granularity `BusinessWeek` (the states of the automaton are represented by circles, the transitions are represented by labeled arrows, and the initial state is identified by a short incoming arrow).

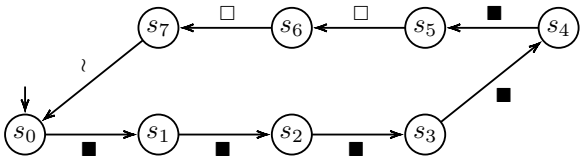

Figure 2.12: An SSA representing `BusinessWeek`.

For the sake of simplicity, from now on we restrict our attention to *unlabeled* time granularities. We say that x is the *index* of a granule g of an unlabeled time granularity G over the temporal domain $(\mathbb{N}, <)$ if g is the $x+1$-th granule of G according to the ordering relation induced by $<$. Note that, in such a case, contiguous occurrences of the symbol $\wr$ in a string-based representation can be replaced by a single occurrence of $\wr$. Moreover, we assume that the string-based representations are in *aligned* form, that is, each occurrence of $\wr$ must be preceded by either the symbol $\blacksquare$ or the symbol $\wr$. This means that we can encode maximal substrings of the form $\blacksquare\wr...\wr$ by a new symbol $\blacktriangleleft$. Note that such a kind of encoding enforces a one-to-one correspondence between strings and (unlabeled) time granularities. As a matter of fact, this simplified framework allows us to reduce the equivalence problem for representations of time granularities to the automata equivalence problem, namely, the problem of establishing whether two given automata recognize the same language/word. As for SSA, the equivalence problem can be easily solved in linear time with respect to the size (i.e., number of states) of the involved automata. Precisely, if M and M' are two SSA recognizing the ultimately periodic words $w = u \cdot v^\omega$ and $w' = u' \cdot (v')^\omega$, respectively, and if $p = max(|u|, |u'|)$ and $q = gcd(|v|, |v'|)$, then, by Fine-Wilf's Lemma [57], we have $w = w'$ iff the following conditions hold:

i) $w[1, p] = w'[1, p]$,

ii) $w[p+1, p+|v|] = w[p+1, p+q]^{\frac{|v|}{q}}$,

iii) $w'[p+1, p+|v'|] = w[p+1, p+q]^{\frac{|v'|}{q}}$,

This leads to a straightforward solution to the equivalence problem for SSA: first compute the lengths of some prefixes and repeating patterns for two given SSA M and M' and then test whether the above conditions hold by simulating the transitions

of M and M'.

2.5.1 Exploiting counters

We saw that granspecs and SSA are well suited notions for modeling time granularities of common periodic phenomena. However, a major limitation of both formalisms is that, whenever the granularity to be represented has a long prefix and/or a long repeating pattern, they produce lengthy representations. As an example, recall that leap years recurs with the same exact order 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 be greater than 10^5. In such cases, computations on representations of time granularities may become rather expensive. In the following, we extend and refine the automaton-based approach by introducing counters in order to compactly encode redundancies of temporal structures. Precisely, we exploit the possibility of activating different transitions from the same (control) state and we rule them through guards envisaging the values of the counters. Such an idea is somehow related to the notion of timed automaton [3], but, differently from that notion, here no synchronization between counters is performed (instead, suitable operators are used to explicitly update the values of the counters when a transition is activated). In the following, we define extended single-string automata.

Definition 2.5.2 *An* extended single-string automaton *(*ESSA *for short) is a tuple* $M = (S, I, \Sigma, \delta, \gamma, s_0, c_0)$*, where*

- S *is a finite set of* control states*,*
- I *is a finite set of* counters *(their* valuations *are functions belonging to the set* $\mathcal{C}_I = \mathbb{N}^I$*),*
- Σ *is a finite alphabet,*
- δ *is a (total)* primary transition function *from* S *to* ${\mathcal{C}_I}^{\mathcal{C}_I} \times \Sigma \times S$*,*
- γ *is a (partial)* secondary transition function *from* S *to* $L_I \times {\mathcal{C}_I}^{\mathcal{C}_I} \times \Sigma \times S$*, with* L_I *being a suitable logical language interpreted over* $\mathbb{N}$ *with free variables belonging to* I*,*
- $s_0 \in S$ *is an* initial state*,*
- $c_0 \in \mathcal{C}_I$ *is an* initial valuation*.*

Like SSA, the run of an ESSA is unique. In order to formally define it, we need to introduce the notion of configuration. A *configuration* for M is a pair state-valuation (s, c), where $s \in S$ and $c \in \mathcal{C}_I$. The transitions of M are taken according to a (total) function $\Delta_M : S \times \mathcal{C}_I \rightharpoonup \Sigma \times S \times \mathcal{C}_I$ such that

- if $\gamma(s) = (\varphi, \sigma, a, r)$ and $c \vDash \varphi$, then $\Delta_M(s, c) = (a, r, \sigma(c))$ (namely, the secondary transition is activated whenever its guard is satisfied by the valuation c),
- if $\gamma(s)$ is not defined or $c \nvDash \mathcal{P}rj_1(\gamma(s))$, and if $\delta(s) = (\sigma, a, r)$, then $\Delta_M(s, c) = (a, r, \sigma(c))$ (namely, the primary transition is activated whenever no secondary transition is defined or the guard of the secondary transition is not satisfied by the valuation c).

The (unique) *run* of an ESSA M, is then defined as the triple $(\mathbf{s}, \mathbf{c}, w) \in S^\omega \times \mathcal{C}_I^\omega \times \Sigma^\omega$ such that

- $\mathbf{s}[1] = s_0$,
- $\mathbf{c}[1] = c_0$,
- for every $i > 0$, $\Delta_M(\mathbf{s}[i], \mathbf{c}[i]) = (w[i], \mathbf{s}[i+1], \mathbf{c}[i+1])$.

Given an ESSA M and its (unique) run $(\mathbf{s}, \mathbf{c}, w)$, we say that w is the word recognized by M. As an example, Figure 2.13 depicts an ESSA representing the granularity `BusinessWeek`. Control states are represented by circles; transitions are represented by arrows annotated with the update operators (e.g., $i \leftarrow i + 1$) and the recognized symbol (e.g., ■); primary and secondary transitions are identified by continuous and dashed arrows, respectively; guards (e.g., $i = 4$) are also specified for each secondary transition; the initial valuation c_0 is assumed to satisfy $c_0(i) = 0$. Note that the granularity `BusinessWeek` is represented by using 2 control states and 1 counter only.

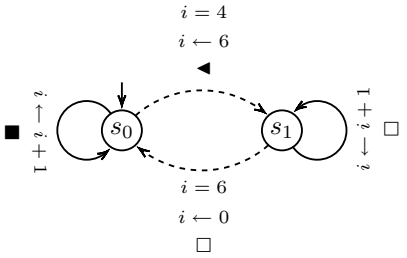

Figure 2.13: An ESSA representing `BusinessWeek`.

A more interesting example is given in Figure 2.14, where an ESSA representing the granularity `Year` in terms of days is depicted. The automaton uses three counters, i, j, and k, to store the index of the current day, month, and year, respectively (each counter is initialized to 0).

From the above examples it is clear how ESSA can be exploited to compactly encode redundancies of temporal structures. However, it is worth noticing that the proposed notion is still too general. In particular, if we do not restrict the set of admissible formulas and update operators for primary and secondary transitions, several paradigmatic problems over automata are not guaranteed to be decidable. As an example, if we allow guards of the form $x = 0$ and update operators of the form $x \leftarrow 0$ and $x \leftarrow x + 1$, then the halting problem for Minsky (two-counters) machines (cf. [76]) can be easily reduced to the equivalence problem for ESSA, thus showing that the latter problem is undecidable. Thus, a specialization of the notion of ESSA is needed. In [39], Dal Lago and Montanari suggest to

i) restrict to guards which are conjunctions of atomic formulas of the form $t_1 = t_2$ or $t_1 \neq t_2$, where each of t_1 and t_2 is either an integer constant or a term of the form $i \bmod d$, with $i \in I$ being a variable and $d > 0$ being an integer constant;

ii) restrict to update operators which are functional compositions of the basic operators $i \leftarrow 0$ and $i \leftarrow i + 1$, where i ranges in I; the operator $i \leftarrow 0$ (resp., $i \leftarrow i + 1$) denotes the function that maps a valuation c to the valuation $c[0/i]$ (resp., $c[c(i) + 1/i]$), where $c[x/i]$ denotes the valuation such that $c[x/i](i) = x$ and $c[x/i](j) = c(j)$ for every $j \neq i$.

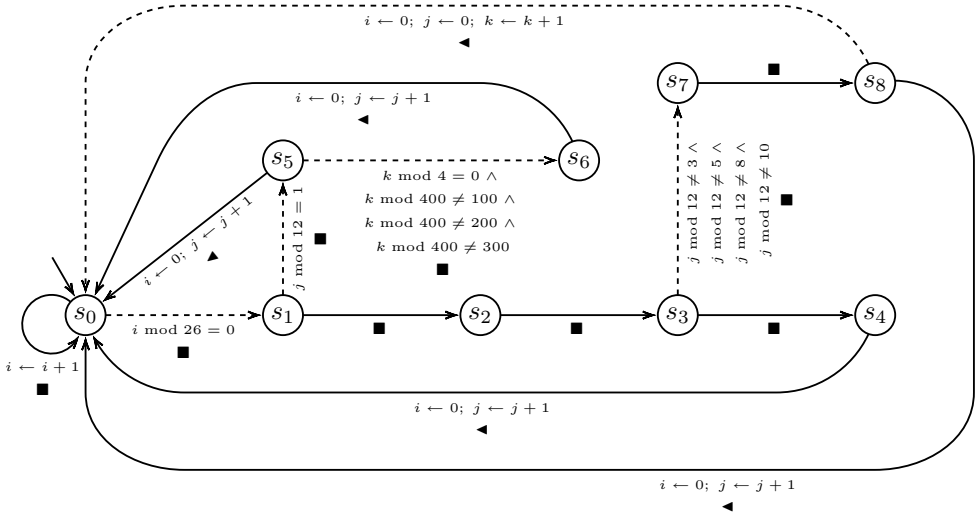

Figure 2.14: An ESSA representing `Year`.

The resulting class of automata, called reducible extended single-string automata, is expressive enough to compactly encode granularities of practical interest (as an example, the automata depicted in Figure 2.13 and in Figure 2.14 are reducible ESSA) and well behaved, namely, they guarantee decidability results for a number of paradigmatic problems over automata. As a matter of fact, by exploiting the above defined restrictions, one can effectively map a reducible ESSA to an equivalent SSA, hence proving that reducible ESSA are as expressive as (but more compact than) SSA. Such a correspondence between the expressiveness of reducible ESSA and the expressiveness of SSA comes from the possibility of defining, for any given reducible ESSA M, an abstraction relation over the configurations of M, which turns out to be an equivalence of finite index compatible with the transition function Δ_M (cf. [3] and [48] for similar constructions).

Formally, we say that a relation $\sim$ over a (possibly infinite) set X is compatible with a function $f : X \rightarrow X$ iff, for every $x, x' \in X$, $x \sim x'$ implies $f(x) \sim f(x')$. As a matter of fact, if $M = (S, I, \Sigma, \delta, \gamma, s_0, c_0)$ is a generic reducible ESSA and, for each counter $i \in I$, d_i is the least common multiple of all constants d that appear inside terms of the form $i \bmod d$ in $\gamma(S)$, then, we can define the relation $\cong_M$ over the set $S \times \mathcal{C}_I$ of configurations of M in such a way that $(s, c) \cong_M (s', c')$ iff $s = s'$ and, for all $i \in I$, $c(i) = c'(i) \pmod{d_i}$[1]. Clearly, $\cong_M$ is an equivalence of finite index. Moreover, it is compatible with the transition function Δ_M, as shown in the following proposition.

Proposition 2.5.3 (Dal Lago and Montanari [39]) *For every reducible ESSA $M = (S, I, \Sigma, \delta, \gamma, s_0, c_0)$, the corresponding equivalence relation $\cong_M$ is compatible with*

[1] $x = y \pmod{d}$ is a shorthand for $x \bmod d = y \bmod d$.

the transition function Δ_M.

Proof. Suppose that $(s, c) \cong_M (s, c')$, for some $s \in S$ and $c, c' \in \mathcal{C}_I$, and consider the case where $\gamma(s) = (\varphi, \sigma, a, r)$ and $c \vDash \varphi$. Then we have $\Delta_M(s, c) = (a, r, \sigma(c))$. Since φ is a conjunction of atoms like $t_1 = t_2$ or $t_1 \neq t_2$ and since the valuations of each term t_j according to c and to c' coincide, we have that $c' \vDash \varphi$ and hence $\Delta_M(s', c) = (a, r, \sigma(c'))$. Moreover, since σ is a functional composition of basic operators like $i \leftarrow 0$ and $i \leftarrow i + 1$, it is easy to see that $\sigma(c)(i) \equiv_{d_i} \sigma(c')(i)$. Therefore, we can conclude that $(r, \sigma(c)) \cong_M (r, \sigma(c'))$. The cases where $\gamma(s)$ is not defined or $c \nvDash \mathcal{P}rj_1(\gamma(s))$ can be handled in a similar way. ■

Proposition 2.5.3 also shows that reducible ESSA belong to the first class of labeled transition systems, according to the classification introduced by Henzinger and Majumdar [63, 62], and hence they are expressively equivalent to SSA.

Theorem 2.5.4 (Dal Lago and Montanari [39]) *Given any reducible ESSA, one can compute an equivalent SSA.*

Proof. Let $M = (S, I, \Sigma, \delta, \gamma, s_0, c_0)$ be a reducible ESSA. We define an equivalent SSA $M' = (Q, \Sigma, \delta', q_0)$ by letting $Q = \big\{[(s, c)]_{\cong_M} : s \in S, c \in \mathcal{C}_I\big\}$ be the set of all equivalence classes of $\cong_M$, $\delta'\big([(s, c)]_{\cong_M}\big) = \big(a, [(r, c')]_{\cong_M}\big)$ whenever $(r, c') = \Delta_M(a, s, c)$ (note that δ' is well defined by Proposition 2.5.3), and $q_0 = [(s_0, c_0)]_{\cong_M}$. Clearly, M and M' recognize the same word. ■

Below, we sketch a simple algorithm, based on Theorem 2.5.4, which computes an SSA M' equivalent to a given reducible ESSA M.

ReduceESSAtoSSA(M)

let $M = (S, I, \Sigma, \delta, \gamma, s_0, c_0)$
let $I = \{1, ..., k\}$
for all $i \in I$ **do**
  $d_i \leftarrow lcm\big\{d \in \mathbb{N} : \exists\, s \in S.\ i \bmod d \text{ is a term appearing in } \mathcal{P}rj_1(\gamma(s))\big\}$
end for
$s \leftarrow s_0$
$c \leftarrow c_0$
$q_0 \leftarrow \big(s, c(1) \bmod d_1, ..., c(k) \bmod d_k\big)$
$q \leftarrow q_0$
$Q \leftarrow \emptyset$
while $q \notin Q$ **do**
  $Q \leftarrow Q \cup \{q\}$
  $(a, r, c') \leftarrow \Delta_M(s, c)$
  $q' \leftarrow \big(r, c'(1) \bmod d_1, ..., c'(k) \bmod d_k\big)$
  $\delta'(q) \leftarrow (a, q')$
  $s \leftarrow r$
  $c \leftarrow c'$
  $q \leftarrow q'$

19: **end while**
20: $M' = (Q, \Sigma, \delta', q_0)$
21: **return** M'

Note that each valuation (line 13) of the transition function Δ_M takes at most linear time with respect to the number $|I|$ of counters of M. Thus, the overall complexity of the algorithm *ReduceESSAtoSSA* is $\mathcal{O}(|I||M'|)$, where $|M'|$ denotes the number of states of the output SSA M'. Moreover, notice that the size of the output automaton M' is at most *exponential* with respect to the size of M, which is defined as the number of states of M plus the size of the guard of each secondary transition. This further implies that the equivalence problem for reducible ESSA can be solved in exponential time with respect to the sizes of the input automata, by first computing equivalent SSA and then testing their equivalence. In fact, such an upper bound to the complexity of the equivalence problem was later refined by Demri in [48, 47].

Corollary 2.5.5 *The equivalence problem for reducible ESSA is in EXPTIME.*

Proof. Trivial from previous results and from the complexity of the equivalence problem for SSA. ■

2.5.2 The logical analogue of reducible ESSA

As mentioned in the introductory section, in [48] Demri describes a logical framework that allows one to express, in a concise way, integer periodicity constraints over a linear temporal domain. The formalism is based on a fragment of Presburger linear temporal logic, denoted PLTL$^{\text{mod}}$. More precisely, the logical language is obtained by combining PLTL (i.e., linear temporal logic with past-time operators) with a suitable first-order constraint language IPC^{++}, whose formulas are built via standard Boolean connectives and existential quantifications, starting from basic atoms 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 where $d, k, d_1, d_2, ...$ are integer constants. Given a valuation $c : \{x, y, ...\} \rightarrow \mathbb{Z}$ for the variables $x, y, ...$, the semantics of an atomic formula is the obvious one: $c \vDash (x = d)$ 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 $c(x) \equiv_k d$, and $c \vDash (x \equiv_k y + [d_1, d_2])$ iff $c(x) \equiv_k c(y) + d$ for some $d_1 \leq d \leq d_2$. Note that the constraint language IPC^{++} is a strict fragment of Presburger arithmetic [61]. The combined language PLTL$^{\text{mod}}$ is obtained by allowing atoms of PLTL to be formulas of the form $\varphi[\mathtt{X}^{i_1} x_{j_1}, ..., \mathtt{X}^{i_k} x_{j_k}]$, where φ is an IPC^{++}-formula with k free variables $x_1, ..., x_k$ and each $\mathtt{X}^{i_l} x_{j_l}$ is a term that replaces all free occurrences of x_l in φ ($\mathtt{X}$ is the usual next-time operator, for instance, $\mathtt{X}x_i$ denotes the value of x_i at the next time point). Atomic formulas of PLTL$^{\text{mod}}$ can then be combined by means of Boolean connectives and temporal modalities $\mathtt{X}$, $\mathtt{X}^{-1}$, $\mathtt{U}$, with the usual meaning. Thus, a model of a PLTL$^{\text{mod}}$-formula is an infinite sequence of valuations, namely, a function of the form $\mathbf{c} : \mathbb{N} \times \{x, y, ...\} \rightarrow \mathbb{Z}$. As an example, we report the encoding of some granularities `Second`, `Minute`, ... of the Gregorian calendar, as defined in [48] (these

granularities are modeled as infinite sequences of valuations for the corresponding integer variables *sec*, *min*, ...):

- $sec \equiv_{60} 0 \wedge \Box(0 \leq sec < 60) \wedge \mathtt{X}sec \equiv_{60} sec + 1$,
- $min \equiv_{60} 0 \wedge \Box(0 \leq min < 60 \wedge (sec = 59 \rightarrow \mathtt{X}min \equiv_{60} min + 1) \wedge (sec \neq 59 \rightarrow \mathtt{X}min = min))$,
- $hour \equiv_{24} 0 \wedge \Box(0 \leq hour < 24 \wedge (min = 59 \wedge sec = 59 \rightarrow \mathtt{X}hour \equiv_{24} hour + 1) \wedge (min \neq 59 \vee sec \neq 59 \rightarrow \mathtt{X}hour = hour))$,
- $day \equiv_7 0 \wedge \Box(0 \leq day < 7 \wedge (hour = 23 \wedge min = 59 \wedge sec = 59 \rightarrow \mathtt{X}day \equiv_7 day + 1) \wedge (hour \neq 23 \vee min \neq 59 \vee sec \neq 59 \rightarrow \mathtt{X}day = day))$,
- as regards the granularities `Month` and `Year` one can encode them by fixing some end dates far ahead in the time line (such an assumption is necessary since we cannot use constraints like $\mathtt{X}year = year + 1$ without incurring in undecidability [31]).

Moreover, notice that, differently from the approach of Combi et al. [30], no propositional variables appear in the logical language. However, any propositional variable P can be easily encoded by an IPC^{++}-formula of the form $x_P = 1$, where x_P is a fresh variable associated with P. As a consequence, PLTL$^{\mathrm{mod}}$ turns out to be a well suited logical language for expressing both qualitative and quantitative temporal constraints. Finally, Demri shows that, like plain LTL but differently from full Presburger LTL, PLTL$^{\mathrm{mod}}$ enjoys a PSPACE-complete satisfiability problem [48, 47]. Such a result is achieved by first defining suitable automaton-based representations for (abstracted) models of PLTL$^{\mathrm{mod}}$-formulas and then by reducing the satisfiability problem to the emptiness problem for these automata.

As for the connection with reducible ESSA, notice that the guards used to define secondary transitions of reducible ESSA can be viewed as Boolean combinations of formulas like $x \equiv_k d$ and $\exists\, z.\ (x \equiv_k z \wedge y \equiv_{k'} z)$. Thus, guards of reducible ESSA belongs to a (strict) fragment of IPC^{++}, denoted IPC*. The corresponding Presburger LTL fragment, obtained by combining PLTL and IPC*, is then denoted by PLTL*. Building on these ideas, Demri shows that the equivalence problem for restricted ESSA is reducible to the satisfiability problem for PLTL*-formulas (or, equivalently, to the emptiness problem for suitable Büchi automata, where the input symbols are atomic IPC*-formulas). The size of the formulas corresponding to a given instance of the equivalence problem for restricted ESSA is shown to be polynomially bounded with respect to the size of the automata, thus proving that the equivalence problem for reducible ESSA is in PSPACE. Notice that such a result improves the previously known EXPTIME upper bound. Moreover, a converse reduction from the satisfiability problem of quantified boolean formulas to the equivalence problem of reducible ESSA is also given, thus showing that the equivalence problem for reducible ESSA is indeed PSPACE-complete.

2.6 Restricted labeled automata

In this section we introduce a new class of automata, called restricted labeled single-string automata (RLA for short), which are an attempt to find a trade-off between

the handiness of SSA and the compactness of (reducible) ESSA. In this perspective, RLA are similar to reducible ESSA, since they exploit counters to compactly encode repeating patterns of time granularities. However, the distinctive feature of this class of automata lies in the structure of the transition functions, which is now more restricted (for instance, update operators are fixed for each counter). By exploiting such restrictions, we will be able to devise improved algorithms for several problems on time granularities.

Before formalizing the notion of RLA, we give an intuitive description of their structure and behavior. In order to simplify the notation and the formalization of useful properties, RLA label states instead of transitions. The set of control states is partitioned into two groups, respectively denoted by S_Σ and S_ε. S_Σ is the set of states where the labeling function is defined, while S_ε is the set of states where it is not defined. Furthermore, like in the case of ESSA, there are two kinds of transitions, respectively called primary and secondary transitions, and, at any point of the computation, at most one (primary or secondary) transition is taken according to an appropriate rule envisaging the state at which the automaton lies and the value of the counter associated with that state. Primary transition functions can be defined in any state, while secondary transition functions are only defined in non-labeled states. Moreover, a primary transition can be taken in a non-labeled state $s \in S_\varepsilon$ only once the secondary transition associated with s has been consecutively taken $c_0(s)$ times, where $c_0(s)$ is the initial valuation for the counter associated with s.

Figure 2.15 depicts an RLA recognizing the word $(\blacktriangleleft\square^6)^\omega$, which represents Mondays in terms of days. States in S_Σ are represented by Σ-labeled circles, while states in S_ε are represented by triangles. Primary and secondary transitions are represented by continuous and dashed arrows, respectively. The (initial values of) counters are associated with states in S_ε (for the sake of readability, we depict them as labels of the secondary transitions exiting states in S_ε).

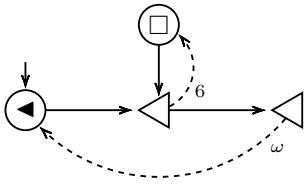

Figure 2.15: An RLA that represents Mondays in terms of days.

In the sequel, we deal with counters ranging over the discrete domain $\mathbb{N} \cup \{\omega\}$. Counters can be either set to their initial value or decremented (we tacitly assume that $\omega - 1 = \omega$). We now proceed with the formal definitions.

Definition 2.6.1 *A* restricted labeled (single-string) automaton *(*RLA *for short) is a tuple* $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$*, where*

- S_Σ *and* S_ε *are disjoint finite sets of (control) states (hereafter, we shall denote by* S *the set* $S_\Sigma \cup S_\varepsilon$*),*
- Σ *is a finite alphabet,*

- $\Omega : S_\Sigma \to \Sigma$ *is a (total)* labeling function,
- $\delta : S \rightharpoonup S$ *is a (partial)* primary transition function *whose transitive closure* δ^+ *is irreflexive (namely, it never happens that* $(s, s) \in \delta^+$*),*
- $\gamma : S_\varepsilon \to S$ *is a (total)* secondary transition function *such that for every* $s \in S_\varepsilon$*,* $(\gamma(s), s) \in \delta^+$*;*
- $s_0 \in S$ *is an* initial state,
- $c_0 : S_\varepsilon \to \mathbb{N}^+ \cup \{\omega\}$ *is an* initial valuation.

Let us denote by $\mathcal{C}_{S_\varepsilon}$ the set of all valuations of the form $c : S_\varepsilon \to (\mathbb{N} \cup \{\omega\})$ for the counters of an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$. A *configuration* for M is a pair (s, c), where $s \in S$ and $c \in \mathcal{C}_{S_\varepsilon}$. The transitions of M are taken according to a (partial) function $\Delta_M : S \times \mathcal{C}_{S_\varepsilon} \rightharpoonup S \times \mathcal{C}_{S_\varepsilon}$ satisfying the following conditions:

i) if $s \in S_\Sigma$ and $\delta(s)$ is defined, then $\Delta_M(s, c) = (\delta(s), c)$ (namely, if the automaton lies in a labeled state and there is an exiting primary transition, then it takes the primary transition, which does not change the valuation),

ii) if $s \in S_\varepsilon$ and $c(s) > 0$, then $\Delta_M(s, c) = (\gamma(s), c')$, where $c' = c[c(s) - 1/s]$ (namely, if the automaton lies in a non-labeled state whose counter has a positive value, then it takes the secondary transition and it decrements the counter by 1),

iii) if $s \in S_\varepsilon$, $c(s) = 0$, and $\delta(s)$ is defined, then $\Delta_M(s, c) = (\delta(s), c')$, where $c' = c[c_0(s)/s]$ (namely, if the automaton lies in a non-labeled state whose counter has value 0 and if there is an exiting primary transition, then it takes the primary transition and it re-initializes the counter),

iv) if none of the above conditions holds, then $\Delta_M(s, c)$ is undefined.

Note that, since Δ_M may be not defined on some configurations, the run of an RLA may be finite. In the sequel, we shall denote by Δ_M^+ (resp., by Δ_M^*) the transitive closure of Δ_M (resp., the reflexive and transitive closure of Δ_M) and we shall adopt the infix notation for such relations. Moreover, we shall denote by Σ^∞ the set of all (finite and infinite) words over Σ, namely $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

The *run* of an RLA M is defined as the pair $(\mathbf{s}, \mathbf{c}) \in S^\infty \times \mathcal{C}_{S_\varepsilon}^\infty$ of maximum (possibly infinite) sequences of states and valuations such that

- $\mathbf{s}[1] = s_0$,
- $\mathbf{c}[1] = c_0$,
- for all $1 \leq i < |\mathbf{s}|$ $(= |\mathbf{c}|)$, $\Delta_M(\mathbf{s}[i], \mathbf{c}[i]) = (\mathbf{s}[i+1], \mathbf{c}[i+1])$.

Given the RLA M and its run $(\mathbf{s}, \mathbf{c})$, one can extract a (finite or infinite) sequence of labeled states $\mathbf{s}_\Sigma \in S_\Sigma^\infty$ by discarding the valuations and the non-labeled states. Such a sequence is said to be the *labeled run* of M. We say that M recognizes the word w iff $w = \Omega(\mathbf{s}_\Sigma)$ (here Ω is extended from states to sequences of states in the natural way). Notice that Definition 2.6.1 allows situations where states and transitions of an RLA form an unconnected (directed) graph. We can overcome these clumsy situations by discarding useless states and transitions. Since counters always range over a finite domain, it is immediate to see that RLA recognize either finite or ultimately periodic words. In the sequel, we provide a finer characterization of the words recognized by RLA. It is based on the notions of δ-*degree* and γ-*degree* of states. The δ-*degree* of a state $s \in S$ is the (unique) natural number n such that $\delta^n(s)$ is defined, but $\delta^{n+1}(s)$ is not. For each non-labeled state $s \in S_\varepsilon$, the γ-*degree* of s is the least $n \in \mathbb{N}$

such that $(\gamma(s), s) \in \delta^n$. We then define a binary relation Γ_M over the set S_ε as follows: $(r, s) \in \Gamma_M$ iff $r = \delta^i(\gamma(s))$, where i is less than the γ-degree of s. Note that the reflexive and transitive closure Γ_M^* is antisymmetric. Thus, Γ_M^* can be given the status of a partial order over the set of non-labeled states. Such a partial order immediately suggests an induction principle, called *γ-induction*, which may be used in both formal definitions and proofs. As an example, if we denote by s_0 the initial state of the RLA of Figure 2.15, by s_1 its successor, by s_2 the top-most state, and by s_3 the right-most state, we have that

- the δ-degree of s_0 (respectively, s_1, s_2, s_3) is 2 (respectively, 1, 2, 0),
- the γ-degree of s_1 is 1 and the γ-degree of s_3 is 2,
- $\Gamma_M = \{(s_1, s_3)\}$ and Γ_M^* consists the pair in Γ_M plus the pairs (s_1, s_1) and (s_3, s_3).

Now, suppose that $(\mathbf{s}, \mathbf{c}) \in S^n \times \mathcal{C}_{S_\varepsilon}^n$ is a finite sequence of n states and valuations satisfying $\Delta_M(\mathbf{s}[i], \mathbf{c}[i]) = (\mathbf{s}[i+1], \mathbf{c}[i+1])$ for every $1 \leq i < n$. Then, we shall write $(\mathbf{s}[1], \mathbf{c}[1]) \rightarrow^w (\mathbf{s}[n], \mathbf{c}[n])$, where $w = \Omega(\mathbf{s}_\Sigma)$. Analogously, if $(\mathbf{s}, \mathbf{c}) \in S^\omega \times \mathcal{C}_{S_\varepsilon}^\omega$ is an infinite sequence of states and valuations satisfying $\Delta_M(\mathbf{s}[i], \mathbf{c}[i]) = (\mathbf{s}[i+1], \mathbf{c}[i+1])$ for every $i \geq 1$, then we shall write $(\mathbf{s}[1], \mathbf{c}[1]) \rightarrow^w$, where $w = \Omega(\mathbf{s}_\Sigma)$. In the following, we denote by σ_s^M the sequence of symbols inductively defined as follows[2]:

- $\Omega(s)$, if $s \in S_\Sigma$,
- $\left(\sigma^M_{\gamma(s)} \cdot \sigma^M_{\delta(\gamma(s))} \cdot \ldots \cdot \sigma^M_{\delta^{m-1}(\gamma(s))}\right)^{c_0(s)}$, if $s \in S_\varepsilon$ and m is the γ-degree of s.

Moreover, for any $s \in S_\varepsilon$, we denote by ρ_s^M the word $\sigma^M_{\gamma(s)} \cdot \sigma^M_{\delta(\gamma(s))} \cdot \ldots \cdot \sigma^M_{\delta^{m-1}(\gamma(s))}$.

The following lemma relates the nesting structure of the transition relations of an RLA M with the occurrences of states in the run of M. It intuitively claims that a non-labeled state s of M can be reached at a certain time only if, for every non-labeled state r such $(r, s) \in \Gamma_M^+$, the value of the counter associated to r at that time coincides with the value $c_0(r)$ of the initial valuation.

Lemma 2.6.2 *Let $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$ be an RLA, $s \in S_\varepsilon$ a non-labeled state, and $c \in \mathcal{C}_{S_\varepsilon}$ a valuation such that $c(r) = c_0(r)$ whenever $(r, s) \in \Gamma_M^+$. Then exactly one of the following conditions holds:*

1. *$c(s) = 0$,*
2. *$(s, c) \rightarrow^w (s, c')$, where $w = \rho_s^M$ and $c' = c[c(s) - 1/s]$, moreover, if $w = u \cdot v$, with $u, v \neq \varepsilon$, and $(s, c) \rightarrow^u (r, c'') \rightarrow^v (s, c')$, with r being a non-labeled state, then we have $(r, s) \in \Gamma_M^+$,*
3. *$(s, c) \rightarrow^w$, where $w = \rho_s^M$; moreover, if $(s, c) \rightarrow^u (r, c')$ holds, with r being a non-labeled state, then we have $(r, s) \in \Gamma_M^+$.*

Proof. We prove the lemma by γ-induction on s. Let m be the γ-degree of s, let $r_i = \delta^i(\gamma(s))$, where i ranges over $\{0, ..., m\}$, and let $c(s) > 0$ (otherwise condition *1.* of lemma trivially holds). If s is a minimal non-labeled state with respect to the ordering given by Γ_M^*, then we know that every state r_i, with $0 \leq i < m$, is a labeled state and hence condition *2.* of lemma follows. Otherwise, if s is not a minimal element with respect to Γ_M^*, by inductive hypothesis, we can distinguish between two cases:

[2]Note that the well-definedness of σ_s^M directly follows from the principle of γ-induction.

i) either every non-labeled state r_i, with $0 \leq i < m$, satisfies condition *1.* or condition *2.* of lemma,
ii) or there is a non-labeled state r_i, with $0 \leq i < m$, satisfying condition *3.* of lemma.

Let first consider case i). We further distinguish between two sub-cases.

- If for every $i \in \{0, ..., m-1\}$, $c(r_i) \neq \omega$, then we let $c' = c[c(s) - 1/s]$ and we verify that

$$(s, c) \rightarrow^{\varepsilon} (r_0, c') \rightarrow^{w_0} (r_1, c') \rightarrow^{w_1} ... \rightarrow^{w_{m-2}} (r_{m-1}, c') \rightarrow^{w_{m-1}} (s, c')$$

  holds, where $w_i = \sigma^M_{r_i}$. Indeed, if $r_i \in S_\Sigma$, then $(r_i, c') \rightarrow^{\Omega(r_i)} (r_{i+1}, c')$ follows, by definition. Otherwise, if $r_i \in S_\varepsilon$, then, by inductive hypothesis, we have

$$(r_i, c') \rightarrow^{v_i} (r_i, c''_1) \rightarrow^{v_i} (r_i, c''_2) \rightarrow^{v_i} ... \rightarrow^{v_i} (r_i, c''_{n_i}) \rightarrow^{\varepsilon} (r_{i+1}, c')$$

  where $v_i = \rho^M_{r_i}$, $n_i = c'(r_i)$, and, for all $j \in \{1, ..., n_i\}$, $c''_j = c'[c'(r_i) - j/r_i]$. Now, suppose that $(s, c) \rightarrow^u (r, c'') \rightarrow^v (s, c')$ holds, with $w = u \cdot v$, $u, v \neq \varepsilon$, and $r \in S_\varepsilon$. Clearly, there is $j \in \{0, ..., m-1\}$ satisfying either $r = r_j$ or, by inductive hypothesis, $(r_j, r) \in \Gamma^+_M$. In both cases, $(r, s) \in \Gamma^+_M$ follows and s satisfies condition *2.* of lemma.
- If there is an index $k \in \{0, ..., m-1\}$ such that $c(r_k) = \omega$, then we can assume, without loss of generality, that k is the least of such indices. Now, since every non-labeled state r_i, with $0 \leq i < k$, satisfies $c(r_i) \neq \omega$ and either condition *1.* or condition *2.* of lemma holds, we can exploit an argument similar to the previous one to show that

$$(s, c) \rightarrow^{\varepsilon} (r_0, c') \rightarrow^{w_0} (r_1, c') \rightarrow^{w_1} ... \rightarrow^{w_{k-1}} (r_k, c')$$

  holds, where $w_i = \sigma^M_{r_i} \neq \varepsilon$ and $c' = c[c(s) - 1/s]$. Again, by inductive hypothesis, since $c'(r_k) = c(r_k) = \omega$, we know that $(r_k, c') \rightarrow^{v_k} (r_k, c')$, where $v_k = \rho^M_{r_k}$. Thus, by letting $w_k = \sigma^M_{r_k} \in \Sigma^\omega$, $(r_k, c') \rightarrow^{w_k}$ follows and hence $(s, c) \rightarrow^w$, where $w = w_0 \cdot w_1 \cdot ... \cdot w_k = \sigma^M_s$. Now, let $(s, c) \rightarrow^u (r, c')$. Clearly, there is $j \in \{0, ..., k\}$ satisfying either $r = r_j$ or $(r, r_j) \in \Gamma^+_M$ and in both cases $(r, s) \in \Gamma^+_M$ follows and s satisfies condition *3.* of lemma.

Let now consider case ii), namely, let i be the least index from $\{0, ..., m-1\}$ such that r_i is a non-labeled state satisfying condition *3.* of lemma. Clearly, for every $j \in \{0, ..., i-1\}$, r_j satisfies either condition *1.* or condition *2.*. Thus, we can proceed, exactly as in the previous case, by distinguishing between two sub-cases, depending on whether there is $k \in \{0, ..., i-1\}$ such that $c(r_k) = \omega$ or not. It is easy to verify that in both sub-cases s satisfies condition *3.* of lemma. ■

Now, we can state and prove the following proposition.

Proposition 2.6.3 *The word recognized by an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$ is of the form $\sigma^M_{s_0} \cdot \sigma^M_{\delta(s_0)} \cdot ... \cdot \sigma^M_{\delta^n(s_0)}$, where n is the δ-degree of s_0,*

Proof. This is a direct consequence of Lemma 2.6.2. ■

Proposition 2.6.3 basically states that the word recognized by any RLA can be represented by expressions like $(\blacksquare^4\blacktriangleleft\square^2)^\omega$, $\blacksquare^6((\blacksquare^2\square)^2\square^2)^\omega$, ..., which include nested repetitions. An an example, let us consider the case of the RLA of Figure 2.15. According to Proposition 2.6.3, the recognized word is $u_0^1 \cdot u_1^6 \cdot u_2^\omega$, where $u_0 = \blacktriangleleft$, $u_1 = \square$, and $u_2 = \blacktriangleleft \cdot \square^6$. The words u_0, u_1, and u_2 are recognized by the RLA M_0, M_1, and M_2 of Figure 2.16, respectively.

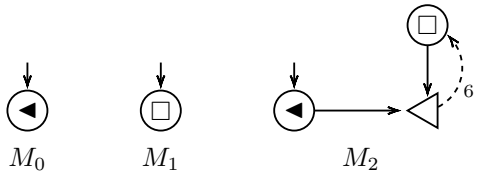

Figure 2.16: A decomposition of an RLA.

2.6.1 Granule conversion problems

The nesting structure of RLA can be exploited to efficiently solve some fundamental problems for time granularities. In particular, the notion of RLA is well suited to cope with the granule conversion problem, namely, the problem of relating granules of a given granularity to those of another one. The relevance of this a problem has been advocated by several authors [8, 90, 106, 49, 73]. Nevertheless, in many approaches it has been only partially worked out in a rather intricate way. In this section we present some algorithms to solve granule conversion problems for RLA-based representations of time granularities. In Section 2.6.3 we shall describe in detail some optimization techniques that make it possible to considerably improve the proposed algorithms for granule conversion.

Hereafter, given an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$, a non-labeled state $s \in S_\varepsilon$, and a symbol $a \in \Sigma$, we shall denote by $|\rho_s^M|$ and by $|\rho_s^M|_a$ respectively the length of ρ_s^M and the number of occurrences of a in ρ_s^M. Furthermore, we denote by $|M|$ the number of states of M. Note that the values $|\rho_s^M|$ and $|\rho_s^M|_a$, for each $a \in \Sigma$, can be pre-computed in quadratic time with respect to $|M|$. We assume that these values are stored into appropriate data structures for M.

To explain our solution to the granule conversion problem, we first address a simpler problem, which arises very often when dealing with time granularities as well as with infinite words in general, namely, the problem of finding the n-th occurrence of a given symbol in a word. Such a problem can be easily solved in linear time *with respect to the number of transitions* needed to reach the symbol occurrence: it suffices to follow the transitions of the automaton until the n-th occurrence of the symbol is recognized. We can improve this straightforward solution by taking advantage of the structure of RLA. For instance, if we are searching for an occurrence of a symbol $a \in \Sigma$ in a word w recognized by an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$ and we have that $s_0 \in S_\varepsilon$ and $\rho_{s_0}^M$ contains no occurrences of the symbol a, then we can avoid

processing the first $c_0(s_0)\cdot|\rho^M_{s_0}|$ symbols in w. Similarly, if $s_0 \in S_\varepsilon$, $\gamma(s_0) \in S_\varepsilon$, and $\rho^M_{s_0}$ contains at least one occurrence of a, but $\rho^M_{\gamma(s_0)}$ does not, then we can start searching for an occurrence of a in w from the position $c_0(\gamma(s_0))|\rho^M_{\gamma(s_0)}|$. By applying the same argument to any state of M, we can devise an algorithm, called *SeekAtOccurrence*, which returns the configuration reached by simulating transitions of M from a given configuration (s, c) until the n-th occurrence of a symbol belonging to a distinguished set $A \subseteq \Sigma$ has been read. As a side effect, *SeekAtOccurrence*$(M, s, c, A, n, counter)$ returns in $counter[a]$ the number of processed occurrences of each symbol $a \in \Sigma$.

SeekAtOccurrence$(M, s, c, A, n, counter)$

```
1: let M = (S_Σ, S_ε, Σ, Ω, δ, γ, s_0, c_0)
2: for all a ∈ Σ do
3:    counter[a] ← 0
4: end for
5: i ← 0
6: while i < n do
7:    if s = ⊥ then
8:       fail
9:    end if
10:   if s ∈ S_Σ then
11:      if Ω(s) ∈ A then
12:         i ← i + 1
13:      end if
14:      counter[Ω(s)] ← counter[Ω(s)] + 1
15:      s ← δ(s)
16:   else
17:      q ← Σ_{a∈A} |ρ^M_s|_a
18:      r ← s
19:      if i + q * c(s) ≤ n then
20:         if |ρ^M_s| * c(s) = ω then
21:            fail
22:         else
23:            l ← c(s)
24:            c(s) ← c_0(s)
25:            s ← δ(s)
26:         end if
27:      else
28:         l ← (n − i) div q
29:         c(s) ← c(s) − l
30:         s ← γ(s)
31:      end if
32:      i ← i + l * q
33:      for all a ∈ Σ do
34:         counter[a] ← counter[a] + |ρ^M_r|_a * l
35:      end for
```

36: **end if**
37: **end while**
38: **return** (s, c)

In spite of the simplicity of the idea, the formal analysis of the complexity of the above algorithm is rather involved. However, it is not difficult to see that *SeekAtOccurrence*$(M, s, c, A, n, counter)$ runs in worst-case *linear* time with respect to $\|M\|$, where $\|M\|$ is a measure of the *complexity* of M in terms of the nesting structure of its transition relations, defined in the following way. For every state s of M and for every integer n, let $\mathbf{C}^M_{s,n}$ be defined as follows[3]:

- 0, if $n < 0$;
- 1, if $n \geq 0$, $s \in S_\Sigma$, and $\delta(s)$ is undefined;
- $1 + \mathbf{C}^M_{\delta(s),n-1}$, if $n \geq 0$, $s \in S_\Sigma$, and $\delta(s)$ is defined;
- $1 + \mathbf{C}^M_{\gamma(s),m-1}$, where m is the γ-degree of s, if $n \geq 0$, $s \in S_\varepsilon$, and $\delta(s)$ is undefined;
- $1 + max\big(\mathbf{C}^M_{\delta(s),n-1}, \mathbf{C}^M_{\gamma(s),m-1}\big)$, where m is the γ-degree of s, if $n \geq 0$, $s \in S_\varepsilon$, and $\delta(s)$ is defined.

The complexity $\|M\|$ is then defined as

$$\|M\| = \mathbf{C}^M_{s_0,n},$$

where s_0 is the initial state of M and n is the δ-degree of s_0. Notice that $\|M\| \leq |M|^2$.

As for the relationships between the complexities of automaton-based and string-based representations, there exist several cases that account for the compactness and tractableness of RLA with respect to granspecs. As an example, it is not difficult to produce an RLA representing the granularity `Month` in terms of days and having complexity 520, which is significantly less than the size of any equivalent granspec.

It turns out that the running time of many other algorithms working on RLA can be expressed in terms of the complexities of the involved automata. In particular, one can provide simple algorithms, akin to those presented in Section 2.4.1 working on granspecs, that look for occurrences of symbols in the word recognized by a given RLA and use *SeekAtOccurrence* as an auxiliary subroutine. It is also worth pointing out that, in general, the complexity of such algorithms is sub-linear with respect to the number of transitions needed to reach the addressed symbol occurrence. As an example, let w be the word recognized by an RLA M. The following algorithm computes the position of the last occurrence of the symbol a in w that precedes the first occurrence of the symbol b.

OccurrenceLastBeforeFirst(M, a, b)
1: **let** $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$
2: $(s, c) \leftarrow (s_0, c_0)$
3: $SeekAtOccurrence(M, s, c, \{b\}, 1, counter_1)$
4: $(s, c) \leftarrow (s_0, c_0)$

[3]Here we use double induction on s and n, where the ordering for the first, dominant, argument is given by the relation Γ^*_M.

5: $SeekAtOccurrence(M, s, c, \{a\}, counter_1[a], counter_2)$
6: **return** $\sum_{c \in \Sigma} counter_2[c]$

In its most common formulation, the problem of granule conversion is viewed as the problem of determining a set of granules of a granularity H which are in some specific relation with a set of granules of a coarser/finer granularity G [56]. 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. Here, we consider the cases of the relations cover, covered-by, and intersect (the other relations can be dealt with in a similar way). The relation *cover* holds between a set R of granules of G and a set S of granules of H if S is the largest set such that $\bigcup_{g \in R} g \supseteq \bigcup_{h \in S} h$ (namely, the union of the granules in R includes the union of the granules in S). The relation *covered-by* is the converse of the relation cover and it holds between a set R of granules of G and a set S of granules of H if S is the smallest set such that $\bigcup_{g \in R} g \subseteq \bigcup_{h \in S} h$ (namely, the union of the granules in R is included in the union of the granules in S). Note that the relation cover defines a total function mapping a set of granules of G into a (possibly empty) set of granules of H, while the relation covered-by only defines a partial function, since it may happen that some sets of granules of G are not covered by any set of granules of H. Finally, the relation *intersect* holds between a set R of granules of G and a set S of granules of H if $S = \{h \in H : \exists\, g \in R.\, g \cap h \neq \emptyset\}$ (namely, S is the set of all and only the granules of H that intersect at least one granule of R). Notice that, if R is a set of granules of G, S and T are sets of granules of H, (R, S) is an instance of the relation covered-by and (R, T) is an instance of the relation intersect, then $S = T$. In some cases, however, given a set R of granules of G, there may not exist a set S of granules of H such that (R, S) is an instance of the relation covered-by, while there always exists a set T of granules of H such that (R, T) is an instance of the relation intersect.

For any possible instance of the granule conversion problem, we distinguish two variants of increasing complexity. In the simplest case (case 1), the involved granularities have no gaps within or between granules; in the second case (case 2), gaps may occur both within and between the granules of the involved granularities. Efficient automaton-based solutions for most common relations can be obtained in the first case, provided that we work with intervals of granules. In such case, the set of granules of the destination granularity H that correspond to an interval of granules of G, is an interval that can be dealt with as a whole. On the contrary, in case 2 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 conversion problems take advantage of some auxiliary functions, called downward conversions and upward conversions, which are quite similar to the conversion operators introduced by Snodgrass et al. in [49, 106]. Downward conversion receives a granularity G and a set (an interval, if we restrict to case 1) R of granules of G as input and it returns as output the set (respectively the interval) $T = \bigcup_{g \in R} g$ of time points. Upward conversion is the dual operation and it comes in three different variants:

- the *cover upward conversion* of a granularity G and a set/interval T of time points is the smallest set/interval S of granules of G such that $T \subseteq \bigcup_{g \in S} g$,
- the *covered-by upward conversion* of a granularity G and a set/interval T of time points is the largest set/interval S of granules of G such that $T \supseteq \bigcup_{g \in S} g$,
- the *intersect upward conversion* of a granularity G and a set/interval T of time points is the set/interval S of all granules g of G such that $g \cap T \neq \emptyset$.

We now provide the algorithms that compute downward and upward conversions for case 1 (no gaps allowed) and case 2 (gaps allowed). In case 1, since the input set R is assumed to be an interval of granules, we can use $min(R)$ and $max(R)$ to denote the least and the greatest index of the granules contained in R (if R contains infinitely many granules, then $max(R) = \omega$). As previously mentioned, since intervals can be dealt with as a whole, downward and upward conversions in case 1 can be implemented using only a finite number of calls to *SeekAtOccurrence* and thus their worst-case running time is linear with respect to $\|M\|$, where M is the RLA representing the involved granularity.

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

DownwardConversion1(M, R)

1: $i \leftarrow$ *OccurrenceFirstAfter*$(M, min(R), \{\blacksquare, \blacktriangleleft\}, \{\blacktriangleleft\})$
2: $j \leftarrow$ *Occurrence*$(M, max(R) + 1, \{\blacktriangleleft\})$
3: **return** $\{k : i \leq k \leq j\}$

The downward conversion for case 2 is performed by *DownwardConversion2*, which processes one granule of the input set R at a time. In particular, for each of such granules, *DownwardConversion2* first computes the smallest interval $[i, j]$ of time points covering that granule and then collects the time points $k \in [i, j]$ that belongs to the granule. The union T of all such time points gives the output of the algorithm[4].

DownwardConversion2(M, R)

1: $T \leftarrow \emptyset$
2: **for all** $x \in R$ **do**
3: $\quad i \leftarrow$ *OccurrenceFirstAfter*$(M, x, \{\blacksquare, \blacktriangleleft\}, \{\blacktriangleleft\})$
4: $\quad j \leftarrow$ *Occurrence*$(M, x + 1, \{\blacktriangleleft\})$

[4]It is worth noticing that termination is not guaranteed if the input set R is finite, but its last granule is infinite. In such a case, the smallest interval covering the set R of granules is infinite (procedure *Occurrence*$(M, x, \{\blacktriangleleft\})$ at line 4 assigns ω to j) and the algorithm cycles at lines 5–9. In order to guarantee termination, one can exploit the fact that M recognizes an ultimately periodic word $w = u \cdot v^{\omega}$ and then reason on the prefix and the repeating pattern of w. This allows one to detect a non-terminating loop and, accordingly, to return the (possibly infinite) set T of converted time points, which can be represented as a generalized arithmetic progression of the form $A \cup \{i + jq : i \in B, j \in \mathbb{N}\}$, where A and B are finite disjoint sets of indices and q is a positive natural number. A similar argument can be applied in the case R is an infinite set of granules, represented as a generalized arithmetic progression of indices. From now on, we shall not consider cases where infinite sets of granules or infinite sets of time points are involved (however, by reasoning on prefixes and repeating patterns of RLA-recognizable words, it is always possible to manage such cases in an effective way).

```
5:    for k = i...j do
6:      if SymbolAt(M, k) ≠ □ then
7:        T ← T ∪ {k}
8:      end if
9:    end for
10: end for
11: return T
```

In succession, we provide the algorithms for cover, covered-by, and intersect upward conversions, for both case 1 and case 2. Note that all algorithms for case 1 work in worst-case linear time with respect to $\|M\|$, where M is the RLA representing the involved granularity. The algorithms for case 2 are more general but less efficient, since they must process one element of the input set T at a time. The correctness of the first algorithm, *CoverUpwardConversion1*, stems from the following observation: if a granularity G is represented by the RLA M and t is a time point, then $CountOccurrences(M, 1, t, \{\blacktriangleleft\})$ is the index of the granule of G that includes t.

CoverUpwardConversion1(M, T)

```
1: x ← CountOccurrences(M, 1, min(T), {◀})
2: y ← CountOccurrences(M, 1, max(T), {◀})
3: return {z : x ≤ z ≤ y}
```

CoverUpwardConversion2(M, T)

```
1: S ← ∅
2: for all i ∈ T do
3:   if SymbolAt(M, i + 1) ≠ □ then
4:     S ← S ∪ {CountOccurrence(M, 1, i, {◀})}
5:   else
6:     fail
7:   end if
8: end for
9: return S
```

Covered-by upward conversions are computed by the following algorithms. Note that, in case 2, the covered-by upward conversion is computed by first collecting all granules of G (i.e., the granularity represented by the RLA M) that intersect the time points in T (lines 2–6), and then discarding those granules which are not entirely covered by T (lines 7–15).

CoveredByUpwardConversion1(M, T)

```
1: if min(T) = 0 then
2:   x ← 0
3: else
4:   x ← CountOccurrences(M, 1, min(T) − 1, {◀}) + 1
5: end if
6: y ← CountOccurrences(M, 1, max(T) + 1, {◀}) − 1
```

7: **return** $\{z : x \leq z \leq y\}$

CoveredByUpwardConversion2(M, T)

1: $S \leftarrow \emptyset$
2: **for all** $i \in T$ **do**
3: **if** $SymbolAt(M, i+1) \neq \square$ **then**
4: $S \leftarrow S \cup \{CountOccurrences(M, 1, i, \{\blacktriangleleft\})\}$
5: **end if**
6: **end for**
7: **for all** $x \in S$ **do**
8: $i \leftarrow OccurrenceFirstAfter(M, x, \{\blacksquare, \blacktriangleleft\}, \{\blacktriangleleft\})$
9: $j \leftarrow Occurrence(M, x+1, \{\blacktriangleleft\})$
10: **for** $k = i...j$ **do**
11: **if** $SymbolAt(M, k) \neq \square$ **and** $k \notin T$ **then**
12: $S \leftarrow S \setminus \{x\}$
13: **end if**
14: **end for**
15: **end for**
16: **return** S

Finally, intersect upward conversions are simplified versions of cover upward conversions (here we do not need to check for failure).

IntersectUpwardConversion1(M, T)

1: $x \leftarrow CountOccurrences(M, 1, min(T), \{\blacktriangleleft\})$
2: $y \leftarrow CountOccurrences(M, 1, max(T), \{\blacktriangleleft\})$
3: **return** $\{z : x \leq x \leq y\}$

IntersectUpwardConversion2(M, T)

1: $S \leftarrow \emptyset$
2: **for all** $i \in T$ **do**
3: **if** $SymbolAt(M, i+1) \neq \square$ **then**
4: $S \leftarrow S \cup \{CountOccurrences(M, 1, i, \{\blacktriangleleft\})\}$
5: **end if**
6: **end for**
7: **return** S

The above defined conversion operations are strictly connected to the relations introduced at the beginning of this section: each of them can be computed by performing a downward conversion followed by the corresponding upward conversion. As an example, the relation cover can be computed as follows.

Cover(M, N, R)

1: **return** $CoverUpwardConversion(N, DownConversion(M, R))$

Depending on the type of the involved granularities, we use different implementations of the conversion algorithms. In particular, if we restrict to granularities without gaps and to intervals (case 1), we can use more efficient implementations, that run in worst-case linear time with respect to $\|M\|$ and $\|N\|$, where M and N are the two RLA representing the involved granularities.

2.6.2 The equivalence problem

In this section we consider the equivalence problem for RLA-based representations of time granularities. As we previously mentioned, two single-string automata represent the same time granularity iff they accepts the same ultimately periodic word. We saw that the equivalence problem for reducible ESSA can be solved in polynomial space with respect to the size of the input automata. As for RLA, we devise a more efficient algorithm, which tests the (non-)equivalence of two given RLA in non-deterministic polynomial time. Our solution is based on a reduction of the equivalence problem to a number-theoretic problem, precisely, the problem of testing the satisfiability of linear diophantine equations, where variables are constrained by lower and upper bounds.

Intuitively, the idea underlying our solution to the equivalence problem is to represent the set of the positions of all occurrences of a labeled state in the labeled run of an RLA M by an expression of the form

$$\bigcup_{i \in [m]} (k_{i,1}x_{i,1} + ... + k_{i,n_i}x_{i,n_i})$$

where each variable $x_{i,j}$ ranges over a suitable interval of $\mathbb{Z}$ and where the values $m, n_1, ..., n_m$ are polynomially bounded with respect to the number of states of M. Thus, given two RLA M and N, one can decide whether M and N recognize the same infinite word by testing the emptiness of every set resulting from the intersection of two expressions E_1 and E_2, where E_1 represents the occurrence positions of an a-labeled state of M and E_2 represents the occurrence positions of a b-labeled state of N, with $a \neq b$. The latter problem can then be reduced to the problem of testing the *non*-satisfiability of some linear diophantine equations with lower and upper bounds on the variables. Even though the satisfiability problem for linear diophantine equations with bounds on variables is known be NP-complete, several solutions that perform well in practice (even on equations with thousands of variables) have been proposed in literature (see, for instance, [1]). Such a reduction shows that the equivalence problem for RLA is in Co-NP. Here, we do not prove the Co-NP-hardness of the equivalence problem for RLA. Instead, we conjecture that such a problem can be solved by a deterministic algorithm which takes polynomial time with respect to the size of input automata. Unfortunately, we are only able to provide a non-deterministic polynomial time algorithm for the (non-)equivalence problem of RLA.

Hereafter, we call *interval* any subset of $\mathbb{Z}$ of the form $[i, j] = \{x : i \leq x \leq j\}$, where $i \in \mathbb{Z} \cup \{-\omega\}$ and $j \in \mathbb{Z} \cup \{\omega\}$.

Definition 2.6.4 *Given a set $P \subseteq \mathbb{Z}$ and two positive integers p, q, we say that P is a p,q-*succinct linear progression *if there exist $m \leq p$, $n_1, ..., n_m \leq q$, $k_{1,1}, ..., k_{1,n_1}, ..., k_{m,1}, ..., k_{m,n_m} \in \mathbb{Z}$, and some intervals $I_{1,1}, ..., I_{1,n_1}, ..., I_{m,1}, ..., I_{m,n_m}$ such that*

$$P = \bigcup_{i \in [m]} \sum_{j \in [n_i]} k_{i,j} I_{i,j},$$

where $\sum_{j \in [n_i]} k_{i,j} I_{i,j} = \{k_{i,1} x_{i,1} + ... + k_{i,n_i} x_{i,n_i} : \forall\, j \in [n].\ x_{i,j} \in I_{i,j}\}$.

Now, we fix an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$ and we denote by $(\mathbf{s}, \mathbf{c})$ the (unique) run of M and by $\mathbf{s}_\Sigma$ the labeled run of M (i.e., the sequence obtained from $(\mathbf{s}, \mathbf{c})$ by discarding the valuations and the non-labeled states). Without loss of generality, we can temporarily assume that Ω is the identity function, which maps a labeled state s to itself (hence, we have $\Sigma = S_\Sigma$). Such an assumption allows us to think of ρ_s^M and σ_s^M (see definitions in Section 2.6) as sequences of labeled states, rather than sequences of symbols. Thus, from Proposition 2.6.3, we immediately have

$$\mathbf{s}_\Sigma = \sigma_{s_0}^M \cdot \sigma_{\delta(s_0)}^M \cdot ... \cdot \sigma_{\delta^n(s_0)}^M,$$

where n is the δ-degree of s_0. For every $s \in S_\Sigma$, every $r \in S$, and every $n \in \mathbb{Z}$ less than or equal to the δ-degree of s, we define the set $P_{s,r,n}$ containing the positions of all occurrences of s in the sequence $\sigma_r^M \cdot \sigma_{\delta(r)}^M \cdot ... \cdot \sigma_{\delta^n(r)}^M$. Clearly, the set of the positions of all occurrences of s in the labeled run $\mathbf{s}_\Sigma$ of M is $P_{s,s_0,n}$, where n is the δ-degree of s_0. Moreover, by exploiting the definition of σ_r^M, we can easily verify that

- $P_{s,r,n} = \emptyset$, if $n < 0$;
- $P_{s,r,n} = \emptyset$, if $n \geq 0$, $r \in S_\Sigma \setminus \{s\}$, and $\delta(r)$ is undefined;
- $P_{s,r,n} = \{1\}$, if $n \geq 0$, $r = s$, and $\delta(r)$ is undefined;
- $P_{s,r,n} = \{1\} + P_{s,\delta(r),n-1}$, if $n \geq 0$, $r \in S_\Sigma \setminus \{s\}$, and $\delta(r)$ is defined;
- $P_{s,r,n} = \{1\} \cup \big(1 + P_{s,\delta(r),n-1}\big)$, if $n \geq 0$, $r = s$, and $\delta(r)$ is defined;
- $P_{s,r,n} = P_{s,\gamma(r),m-1} + |\rho_r^M|[0, c_0(r) - 1]$, where m is the γ-degree of r, if $n \geq 0$, $r \in S_\varepsilon$, and $\delta(r)$ is undefined;
- $P_{s,r,n} = \big(P_{s,\gamma(r),m-1} + |\rho_r^M|[0, c_0(r) - 1]\big) \cup \big(|\sigma_r^M| + P_{s,\delta(r),n-1}\big)$, where m is the γ-degree of r, if $n \geq 0$, $r \in S_\varepsilon$, and $\delta(r)$ is defined.

The above equalities lead to a straightforward algorithm that computes the set of the positions of all occurrences of a given labeled state s in σ_r^M.

RLAPositions(M, s, r, n)

```
1: let M = (S_Σ, S_ε, Σ, Ω, δ, γ, s_0, c_0)
2: if n < 0 then
3:    return ∅
4: else if r ∈ S_Σ then
5:    if δ(r) = ⊥ then
6:       if r ≠ s then
7:          return ∅
8:       else
```

```
      return {1}
    end if
  else if r ≠ s then
    return 1 + RLAPositions(M, s, δ(r), n − 1)
  else
    return {1} ∪ (1 + RLAPositions(M, s, δ(r), n − 1))
  end if
else if δ(r) = ⊥ then
  m ← γ-degree(r)
  return RLAPositions(M, s, γ(r), m − 1) + |ρ_r^M| ∗ [0, c_0(r) − 1]
else
  m ← γ-degree(r)
  return (RLAPositions(M, s, γ(r), m − 1) + |ρ_r^M| ∗ [0, c_0(r) − 1]) ∪
         (|σ_r^M| + RLAPositions(M, s, δ(r), n − 1))
end if
```

The following lemma shows that the above defined set $P_{s,r,n}$ is indeed an $|S|^2, |S|^2$-succinct linear progression.

Lemma 2.6.5 *For every $s \in S_\Sigma$, every $r \in S$, and every $n \in \mathbb{Z}$ less than or equal to the δ-degree of r, $P_{s,r,n}$ is an $|S|^2, |S|^2$-succinct linear progression.*

Proof. First of all, notice that, if $P = \bigcup_{i\in[m]} \sum_{j\in[n_i]} k_{i,j} I_{i,j}$ is a p, q-succinct linear progression, k is an integer, and I is an interval, then $P' = \big(\bigcup_{i\in[m]} \sum_{j\in[n_i]} k_{i,j} I_{i,j}\big) + kI = \bigcup_{i\in[m]} \big(\sum_{j\in[n_i]} k_{i,j} I_{i,j} + kI\big)$ is a $p, q+1$-succinct linear progression. This implies that the above defined set $P_{s,r,n}$ can be written as $\bigcup_{i\in[m]} \sum_{j\in[n_i]} k_{i,j} I_{i,j}$, where $m \leq p_{r,n}$, $n_1, ..., n_m \leq q_{r,n}$, and $p_{r,n}$ and $q_{r,n}$ are recursively defined as follows:

- $p_{r,n} = 0$, if $n < 0$;
- $p_{r,n} = 1$, if $n \geq 0$, $r \in S_\Sigma$, and $\delta(r)$ is undefined;
- $p_{r,n} = 1 + p_{\delta r, n-1}$ if $n \geq 0$, $r \in S_\Sigma$, and $\delta(r)$ is defined;
- $p_{r,n} = p_{\gamma(r),m-1}$, where m is the γ-degree of r, if $n \geq 0$, $r \in S_\varepsilon$, and $\delta(r)$ is undefined;
- $p_{r,n} = p_{\gamma(r),m-1} + p_{\delta(r),n-1}$, where m is the γ-degree of r, if $n \geq 0$, $r \in S_\varepsilon$, and $\delta(r)$ is defined;
- $q_{r,n} = 0$, if $n < 0$;
- $q_{r,n} = 1$, if $n \geq 0$, $r \in S_\Sigma$, and $\delta(r)$ is undefined;
- $q_{r,n} = 1 + q_{\delta(r),n-1}$, if $n \geq 0$, $r \in S_\Sigma$, and $\delta(r)$ is defined;
- $q_{r,n} = 1 + q_{\gamma(r),m-1}$, where m is the γ-degree of r, if $n \geq 0$, $r \in S_\varepsilon$, and $\delta(r)$ is undefined;
- $q_{r,n} = 1 + max(q_{\delta(r),n-1}, q_{\gamma(r),m-1})$, where m is the γ-degree of r, if $n \geq 0$, $r \in S_\varepsilon$, and $\delta(r)$ is defined.

Moreover, by exploiting double induction on r and n, it is easy to verify that $p_{r,n} \leq \big|\big\{(t, \delta^i(r)) : 0 \leq i \leq n, t = \delta^i(r) \in S_\Sigma \vee (t, \delta^i(r)) \in \Gamma^*_M\big\}\big| \leq |S|^2$ and $q_{r,n} = \mathbf{C}^M_{r,n} \leq |S|^2$. ∎

As a consequence of Lemma 2.6.5, we have that the set of the positions of all occurrences of a labeled state in the labeled run of an RLA can be effectively represented as a succinct linear progression, whose size is polynomial in the size of the automaton. This also implies that $RLAPositions(M, s, s_0, n)$ takes polynomial time with respect to the size of the input RLA.

Proposition 2.6.6 *For every RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$ and for every labeled state $s \in S_\Sigma$, the set of the positions of all occurrences of s in $\mathbf{s}_\Sigma$ is a $|S|^2, |S|^2$-succinct linear progression.*

Proof. Given an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$, we define a new RLA $M' = (S_\Sigma, S_\varepsilon, S_\Sigma, \Omega', \delta, \gamma, s_0, c_0)$, where $\Omega'(s) = s$ for every $s \in S_\Sigma$. Clearly, the labeled run of M coincides with the labeled run of M'. Therefore, the claim follows trivially from Lemma 2.6.5. ■

We conclude the section by showing how to reduce the equivalence problem to the satisfiability problem for linear diophantine equations.

Theorem 2.6.7 *Two RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$ and $N = (S'_\Sigma, S'_\varepsilon, \Sigma, \Omega', \delta', \gamma', s'_0, c'_0)$ recognize the same infinite word iff for every $s \in S_\Sigma$ and $s' \in S'_\Sigma$, either $\Omega(s) = \Omega'(s')$ or $P_{s,s_0,n} \cap Q_{s',s'_0,n'} = \emptyset$ holds, where $P_{s,s_0,n}$ (resp., $Q_{s',s'_0,n'}$) is the set of the positions of all occurrences of s (resp., s') in the labeled run of M (resp., N).*

Proof. If M and N recognize the same infinite word w, then $w = \Omega(\mathbf{s}_\Sigma) = \Omega'(\mathbf{s}'_\Sigma)$, where $\mathbf{s}_\Sigma$ (resp., $\mathbf{s}'_\Sigma$) is the labeled run of M (resp., N). This implies that, if $s \in S_\Sigma$ and $s' \in S'_\Sigma$ with $\Omega(s) \neq \Omega'(s')$, then $P_{s,s_0,n} \cap Q_{s',s'_0,n'} = \emptyset$. Conversely, if M recognizes an infinite word w and N recognizes an infinite word w', with $w \neq w'$, then we let i be the first position such that $w[i] \neq w'[i]$ and we define $s = \mathbf{s}_\Sigma[i]$ and $s' = \mathbf{s}'_\Sigma$. Clearly, $\Omega(s) \neq \Omega(s')$ holds and i belongs to both $P_{s,s_0,n}$ and $Q_{s',s'_0,n'}$. ■

Finally, given two succinct linear progressions $P = \bigcup_{i\in[m]} \sum_{j\in[n_i]} k_{i,j} I_{i,j}$ and $Q = \bigcup_{i\in[m']} \sum_{j\in[n'_i]} h_{i,j} J_{i,j}$, $P \cap Q = \emptyset$ holds iff for every $i \in [m]$ and for every $i' \in [m']$, the following linear diophantine equation with bounds on variables is *not* satisfiable:

$$\begin{cases} k_{i,1}x_{i,1} + ... + k_{i,n_i}x_{i,n_i} = h_{i',1}y_{i',1} + ... + h_{i',n_i}y_{i',n'_{i'}} \\ \forall\, j \in [n_i].\ min(I_{i,j}) \leq x_{i,j} \leq max(I_{i,j}) \\ \forall\, j \in [n'_{i'}].\ min(J_{i',j}) \leq y_{i',j} \leq max(J_{i',j}) \end{cases}$$

Checking the satisfiability of a generic linear diophantine equation with bounds on variables is known to be an NP-complete problem. As for the NP-hardness, one can reduce the well-known *subset sum* problem (i.e., given a finite set Z of integers, decide whether there exists a subset Z' of Z that sums to exactly 0) to the satisfiability problem for linear diophantine equations. Precisely, given a finite set $Z = \{k_1, ..., k_n\}$ of integers, we define the linear diophantine equation $k_1 z_1 + ... + k_n z_n = 0$, where each variable z_i can be either 0 or 1. It clearly follows that the equation is satisfiable

iff Z is a positive instance of the subset sum problem. Even though the satisfiability problem for linear diophantine equations with bounds on variables is NP-complete, several efficient algorithms, based on non-trivial properties of rings and lattices, have been proposed in the literature (see, for instance, [1]). These algorithms can solve (systems of) linear diophantine equations with thousands of variables in a reasonable time and thus they can be effectively exploited to test the emptiness of sets resulting from the intersection of two succinct linear progressions.

The following algorithm solves the equivalence problem for RLA by reducing its instances to (a polynomial number of) instances of the satisfiability problem for linear diophantine equations.

RLAEquivalence(M, N)

1: **let** $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$
2: **let** $N = (S'_\Sigma, S'_\varepsilon, \Sigma, \Omega', \delta', \gamma', s'_0, c'_0)$
3: $n \leftarrow \delta$-degree(s_0)
4: **for all** $s \in S_\Sigma$ **do**
5: $\quad P[s] \leftarrow RLAPositions(M, s, s_0, n)$
6: **end for**
7: $n' \leftarrow \delta$-degree(s'_0)
8: **for all** $s' \in S'_\Sigma$ **do**
9: $\quad Q[s'] \leftarrow RLAPositions(N, s', s'_0, n')$
10: **end for**
11: **for all** $s \in S_\Sigma$ **do**
12: $\quad$ **for all** $s' \in S'_\Sigma$ **do**
13: $\quad\quad$ **if** $\Omega(s) \neq \Omega'(s')$ **and** $P[s] \cap Q[s'] \neq \emptyset$ **then**
14: $\quad\quad\quad$ **return false**
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: **end for**
18: **return true**

2.6.3 Optimality of automaton-based representations

In Section 2.6.1 we outlined some basic algorithms that compute granule conversions in worst-case linear time with respect to the complexities of the involved RLA. It immediately follows that, given an RLA M, it is worth to minimize its complexity $\|M\|$. Furthermore, there exists a widespread recognition of the fact that state minimization is an important problem in classical automata theory as well as in the theory of reactive systems, and thus another goal of practical interest is the minimization of $|M|$. The former problem is called *complexity-optimization problem*, while the latter is called *size-optimization problem*. Even though the size and complexity measures associated with an RLA are clearly related one to the other, they are not equivalent, and the same holds for the corresponding minimization problems. In particular, the size-optimization problem seems to be harder than the complexity-optimization one and only a partial solution to it will be given here. It is also worth to remark

that size- and complexity-optimal automata are not guaranteed to be unique (up to isomorphisms) as it happens, for instance, for deterministic finite automata. As an example, the three automata M, N, and O of Figure 2.17 recognize the same finite word ■■□■■□■■□. M and N are size-optimal automata ($|M| = |N| = 4$, while $|O| = 6$), and M and O are complexity-optimal automata ($\|M\| = \|O\| = 5$, while $\|N\| = 7$).

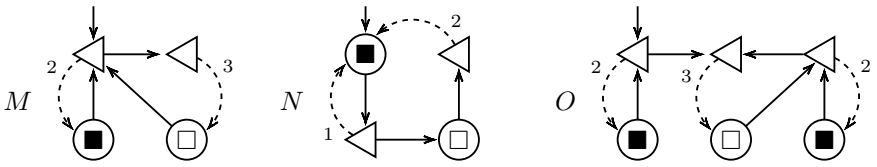

Figure 2.17: Size-optimal and complexity-optimal automata.

Automata optimization problems can be addressed in many different ways, e.g., by partitioning the state space or by exploiting noticeable relations between automata and expressions encoding the recognized words. In the following, we tackle both the complexity- and the size-optimization problems by using dynamic programming, namely, by computing an optimal automaton starting from smaller (optimal) ones in a bottom-up fashion. The key point of such a solution is the proof that the optimization problems enjoy *optimal-substructure properties*. In the following, we describe two operations on RLA and we prove closure properties for them; then, we compare the complexity and the size of compound automata with that of their components. In the last part of this section, we shall take advantage of closure properties of RLA to provide optimal substructure properties for the two optimization problems for RLA.

Closure properties

The class of RLA is effectively closed with respect to the operations of concatenation and repetition. Given two RLA M and N, recognizing u and v, respectively, we denote by $AppendChar(a, N)$ and by $AppendRepeat(M, k, N)$ respectively the *concatenation of a to N*, which recognizes the word $a \cdot v$, and the *concatenation of a k-repetition of M to N*, which recognizes the word $u^k \cdot v$, where $k \in \mathbb{N}^+ \cup \{\omega\}$ and u is assumed to be a finite word. The automata resulting from such operations can be computed as follows:

- the automaton $AppendChar(a, N)$ can be obtained from N by (i) adding a new a-labeled state s_0, (ii) linking it to the initial state of N, and (iii) giving it the status of initial state of the resulting automaton (see Figure 2.18);
- the automaton $AppendRepeat(M, k, N)$ can be obtained from M and N by (i) adding a new non-labeled state s_{loop}, (ii) introducing a secondary transition from s_{loop} to the initial state of N, a primary transition from the final state of N to s_{loop}, and a primary transition from s_{loop} to the initial state of M, and (iii) giving s_{loop} the status of initial state of the resulting automaton (see Figure 2.19).

We can actually give *AppendChar* and *AppendRepeat* the status of algorithms running in linear time.

Figure 2.18: The concatenation of a to N.

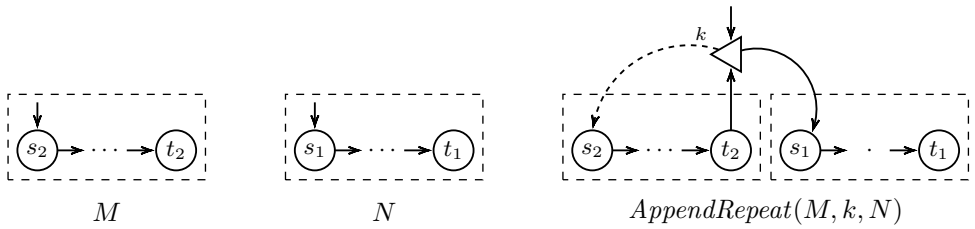

Figure 2.19: The concatenation of a k-repetition of M to N.

If the argument N in the above definitions is missing, then we obtain:

- the automaton $AppendChar(a)$, which recognizes a single character a,
- the automaton $AppendRepeat(M, k)$, which recognizes u^k.

Moreover, the size (resp., the complexity) of the resulting automata can be specified in terms of the size (resp., the complexity) of the component automata as follows:

- $AppendChar(a, N)$ has size $1 + |N|$ and complexity $1 + \|M\|$,
- $AppendRepeat(M, k, N)$ has size $1+|M|+|N|$ and complexity $1+max(\|M\|, \|N\|)$.

Now, let Σ be a finite alphabet and let $\mathcal{R}_\Sigma$ be the class of all RLA obtained from symbols in Σ by applying the operators *AppendChar* and *AppendRepeat*. Clearly, $\mathcal{R}_\Sigma$ is *properly included* in the class of all RLA, that is, there exist some RLA, including size-optimal and complexity-optimal ones (e.g., the automaton M in Figure 2.17), that cannot be generated via *AppendChar* and *AppendRepeat*. Nevertheless, it turns out that for every RLA M, $\mathcal{R}_\Sigma$ always contains at least one RLA which is equivalent to M and has the same *complexity*. Such a property can be exploited to generate a complexity-optimal automaton for a given (finite or infinite) word, starting from smaller (complexity-optimal) automata and using the above mentioned operators. Unfortunately, similar properties do not hold for size-optimal RLA.

Lemma 2.6.8 *For every RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, c_0)$, for every state $s \in S$, and for every natural number n not exceeding the δ-degree of s, there is an RLA $N_{s,n} \in \mathcal{R}_\Sigma$ that recognizes $w_{s,n} = \sigma^M_s \cdot \sigma^M_{\delta(s)} \cdot \ldots \cdot \sigma^M_{\delta^n(s)}$ and such that $\|N_{s,n}\| \leq \mathbf{C}^M_{s,n}$.*

Proof. The proof is by induction on $\mathbf{C}^M_{s,n}$. We distinguish some cases depending on s being an element of S_Σ or S_ε and on n being 0 or a positive integer.

- If $s \in S_\Sigma$ and $n = 0$, then we have $\mathbf{C}^M_{s,n} = 1$ and $w_{s,n} = \Omega(s)$. Hence, the thesis immediately follows, since $\|AppendChar(\Omega(s))\| = 1$.
- If $s \in S_\Sigma$ and $n > 0$, then we have $\mathbf{C}^M_{s,n} = 1+\mathbf{C}^M_{\delta(s),n-1}$ and $w_{s,n} = \Omega(s)\cdot w_{\delta(s),n-1}$. Hence, we can apply the inductive hypothesis to $\delta(s)$ and $n-1$, so that the thesis follows, since $\|AppendChar(\Omega(s), N_{\delta(s),n-1})\| = 1 + \|N_{\delta(s),n-1}\|$.
- If $s \in S_\varepsilon$ and $n = 0$, then we have $\mathbf{C}^M_{s,n} = 1 + \mathbf{C}^M_{\gamma(s),m-1}$ and $w_{s,n} = w^{c_0(s)}_{\gamma(s),m-1}$, with m being the γ-degree of s, under the assumption that $m > 0$ (the case $m = 0$ can be easily handled). By applying the inductive hypothesis to $\gamma(s)$ and $m-1$, we obtain an automaton $N_{\gamma(s),m-1}$ such that $N_{s,n} = AppendRepeat(N_{\gamma(s),m-1}, c_0(s))$ is the desired automaton.
- If $s \in S_\varepsilon$ and $n > 0$, then we have $\mathbf{C}^M_{s,n} = 1 + max\{\mathbf{C}^M_{\gamma(s),m-1}, \mathbf{C}^M_{\delta(s),n-1}\}$ and $w_{s,n} = w^{c_0(s)}_{\gamma(s),m-1} \cdot w_{\delta(s),n-1}$, with m being the γ-degree of s, under the assumption that $m > 0$ (the case $m = 0$ can be easily handled). From the inductive hypothesis, we obtain two RLA $N_{\gamma(s),m-1}$ and $N_{\delta(s),n-1}$ such that $N_{s,n} = AppendRepeat(N_{\gamma(s),m-1}, c_0(s), N_{\delta(s),n-1})$ is the desired automaton.

■

Proposition 2.6.9 *For every RLA M, there is an equivalent RLA $N \in \mathcal{R}_\Sigma$ such that $\|N\| \leq \|M\|$.*

Proof. This is trivial in view of Proposition 2.6.3 and Lemma 2.6.8. Let s_0 be the initial state of M and n its δ-degree. Consider the RLA $N_{s_0,n}$ obtained from Lemma 2.6.8. Clearly, $\|N_{s_0,n}\| \leq \|M\|$ and N recognizes the same word as M. ■

As an example, consider the automata M and O of Figure 2.17. They have the same complexity ($\|M\| = \|O\| = 5$), but $O \in \mathcal{R}_\Sigma$, while $M \notin \mathcal{R}_\Sigma$. It is easy to show that O can be obtained from M by applying the transformations given in Lemma 2.6.8.

Computing complexity-optimal automata

Here we exploit closure properties of RLA to devise a polynomial-time solution for the complexity-optimization problem. In virtue of Proposition 2.6.9, we have that, for any (finite or ultimately periodic) word $w \in \Sigma^\infty$, there exists a complexity-optimal automaton M that recognizes w and belongs to $\mathcal{R}_\Sigma$. As a matter of fact, we can prove that, for any word w, there exists one such M that is decomposable into complexity-optimal automata. As a preliminary result, we establish the following technical lemma.

Lemma 2.6.10 *Given an RLA M recognizing a (finite or infinite) word $w \in \Sigma^\infty$ and a natural number $1 \leq n \leq |w|$, there is an RLA in $\mathcal{R}_\Sigma$, denoted $Prefix(M, n)$, recognizing the prefix $w[1, n]$ and satisfying $\|Prefix(M, n)\| \leq \|M\|$.*

Proof. By Proposition 2.6.9, we have that there exists $N \in \mathcal{R}_\Sigma$ equivalent to M and such that $\|N\| \leq \|M\|$. We prove the thesis by induction on the structure of $N \in \mathcal{R}_\Sigma$. We only consider the non-trivial cases.

- Suppose that $N = \mathit{AppendChar}(a, L)$. We further distinguish between the following subcases:
  1. if $n = 1$, then the required automaton is $\mathit{AppendChar}(a)$;
  2. if $n > 1$, then the required automaton is $\mathit{AppendChar}(a, O)$, where O is the automaton obtained by applying the inductive hypothesis to L and $n - 1$.
- Suppose that $N = \mathit{AppendRepeat}(L, k, O)$, where k is a natural number. Let u (resp., v) be the word recognized by L (resp., O). We distinguish between the following subcases:
  1. if $n \leq |u|$, then the required automaton is obtained by simply applying the inductive hypothesis on L and n;
  2. if $|u| < n \leq k|u|$, let m and l be natural numbers such that $n = m|u| + l$, with $l < |u|$, and let P be the automaton obtained by applying the inductive hypothesis on L and l. Then, the required automaton is $\mathit{AppendRepeat}(L, m, P)$;
  3. if $n > k|u|$, then, by applying the inductive hypothesis on O and $n - k|u|$, we obtain P. The required automaton is $\mathit{AppendRepeat}(L, k, P)$.

■

The following two theorems are the basic ingredients of the solution to the complexity-minimization problem. They state optimal substructure properties for finite and ultimately periodic words, respectively.

Theorem 2.6.11 *Given a finite word w, one of the following conditions holds:*

1. $|w| = 1$ *and* $\mathit{AppendChar}(a)$ *is complexity-optimal for* w*;*
2. $|w| > 1$ *and* $\mathit{AppendChar}(a, M)$ *is complexity-optimal for* w*, whenever* M *is complexity-optimal for* $w[2, |w|]$*;*
3. $|w| > 1$ *and* $\mathit{AppendRepeat}(M, |w|/p)$ *is complexity-optimal for* w*, whenever* M *is complexity-optimal for* $w[1, p]$*, where* p *is the period of* w*;*
4. $|w| > 1$ *and there exists* $r < |w|$ *such that* $\mathit{AppendRepeat}(M, r/p, O)$ *is complexity-optimal for* w*, whenever* M *is complexity-optimal for* $w[1, p]$*, where* p *is the period of* $w[1, r]$ *and* O *is complexity-optimal for* $w[r + 1, |w|]$*.*

Proof. By Proposition 2.6.9, there exists a complexity-optimal automaton $N \in \mathcal{R}_\Sigma$ recognizing w. We prove the thesis by induction on the structure of N. We only consider the non-trivial cases.

- Suppose that $N = \mathit{AppendChar}(a, L)$ and let M be a complexity-optimal automaton recognizing $w[2, |w|]$. Then we have:

$$\begin{aligned} \|\mathit{AppendChar}(a, M)\| &= 1 + \|M\| \leq 1 + \|L\| = \\ &= \|\mathit{AppendChar}(a, L)\| = \|N\| \end{aligned}$$

  This implies that $\mathit{AppendChar}(a, M)$ is complexity-optimal.
- Suppose that $N = \mathit{AppendRepeat}(L, k, P)$, where $k \in \mathbb{N}$. Clearly, $k > 0$ holds, because otherwise N would not be complexity-optimal. Let u be the finite word recognized by L, let $r = k|u|$, and let p be the minimum period of $w[1, r]$ (and thus we have that $p \leq |u|$). Moreover, let M be a complexity-optimal automaton

for $w[1,p]$ and O be a complexity-optimal automaton for $w[r+1,|w|]$. We have (by possibly exploiting Lemma 2.6.10)

$$\begin{aligned} \|AppendRepeat(M, r/p, O)\| &= 1 + max\{\|M\|, \|O\|\} \leq \\ &\leq 1 + max\{\|L\|, \|P\|\} = \|N\|, \end{aligned}$$

which implies that $AppendRepeat(M, r/p, O)$ is complexity-optimal.

■

The case of ultimately periodic words is more problematic, because it may happen that a complexity-optimal automaton operates on a non-minimum prefix. Consider, for instance, the word $(abc)^2 \cdot ab \cdot (ce)^\omega$. Its minimum prefix length is 8 and its minimum period is 2. However, the complexity-optimal automata for such a word operate on the prefix $(abc)^3$ (having length 9) and on the repeating pattern ec (having length 2).

Theorem 2.6.12 *Given an ultimately periodic word u having minimum prefix length l and minimum period q, at least one of the following conditions holds:*

1. *$AppendChar(a, M)$ is complexity-optimal for w, whenever M is complexity-optimal for $w[2,\omega]$;*
2. *$AppendRepeat(M, \omega)$ is complexity-optimal for w, whenever M is complexity-optimal for $w[1,q]$;*
3. *There exists $r \leq 2l+2q$ such that $AppendRepeat(M, r/p, O)$ is complexity-optimal for w, whenever M is complexity-optimal for $w[1,p]$, where p is the period of $w[1,r]$, and O is complexity-optimal for $w[r+1,\omega]$.*

Proof. We proceed in the usual way. By Proposition 2.6.9, we know that there is a complexity-optimal automaton $N \in \mathcal{R}_\Sigma$ recognizing w and we prove the thesis by induction on the structure of N. We only consider the most complex case, that is, the case in which $N = AppendRepeat(L, k, P)$, where k is a *positive* natural number. We distinguish two cases: $k = 1$ and $k > 1$. Let assume $k = 1$ and let u be the finite word recognized by L. We define $r = min(|u|, ((|u| - l) \bmod q) + l)$. Note that r is always less than or equal to $|u|$ and it is strictly less than $l + q$. For every complexity-optimal automaton M that recognizes $w[1,r]$ and for every complexity-optimal automaton O that recognizes $w[r+1,\infty]$ $(= w[|u|+1,\omega])$, we have that $\|M\| \leq \|L\|$ (by Lemma 2.6.10) and $\|O\| \leq \|P\|$. Hence, $AppendRepeat(M, 1, O)$ is a complexity-optimal automaton that recognizes w. Now let assume $k > 1$. By proceeding as in the proof of Theorem 2.6.11, we can replace the automata L and P by equivalent complexity-optimal automata M and O, respectively. Let u be the finite word recognized by M, let $r = k|u|$, and let p be the minimum period of $w[1,r]$ (and thus we have $p \leq |u|$). To complete the proof, we need to show that $r \leq 2l + 2q$. Suppose, by contradiction, that $r > 2l + 2q$ and consider the substring $t = w[l+1,r]$ of w, which has partial periods q and $|u|$. We have that $r \geq max(2l + 2q, 2|u|)$ and thus $r \geq l + q + |u|$, whence $|t| \geq q + |u|$. By Lemma 1.5.2, this implies that t has partial period $m = gcd(q, |u|)$. Consider now the substring $w[2l+1, 2l+2q]$ of t. It has period q (since $2l+1$ is greater than l) and partial period m. Since m divides q, m is in fact a period. From the fact that q is the minimum period, we have that $q = m$.

It immediately follows that $|u|$ is a multiple of q $(= m)$, and thus u has period q. Moreover, from the minimality of l, it follows that $l = 0$ (if $l > 0$, then $w[l] \neq w[l+q]$, but, from $r > l + q$, we have that $w[l] = w[l + q]$). Hence, the repeating pattern of u having length q is equal to $w[r + 1, r + q]$ and we have

$$\begin{aligned} w &= w[1, r] \cdot w[r + 1, \omega] = u^k \cdot w[r + 1, r + q]^\omega = \\ &= w[1, q]^h \cdot w[r + 1, r + q]^\omega = w[r + 1, r + q]^\omega = w[r + 1, \omega], \end{aligned}$$

which contradicts the hypothesis that N is complexity-optimal (O recognizes $w[r + 1, \omega] = w$). ■

According to Theorem 2.6.11 and Theorem 2.6.12 there exists only a *finite* number of ways of building a complexity-optimal automaton for a word w, given some (optimal) automata for the substrings of w. However, if w is an infinite word, we must show that there is an upper bound on the number of possible applications of case *3.* of Theorem 2.6.12.

For every $n \in \mathbb{N}$, we denote by $\mathcal{F}_\Sigma(n)$ the restriction of $\mathcal{R}_\Sigma$ to automata recognizing finite words of length at most n. Moreover, for every n, r, m, we define, by induction on n, the subclass $\mathcal{T}_\Sigma(n, r, m)$ of $\mathcal{R}_\Sigma$, which contains automata recognizing ultimately periodic words with repeating patterns not longer than m.

$$\begin{aligned} \mathcal{T}_\Sigma(0, r, m) &= \{\mathit{AppendRepeat}(M, \omega) \,:\, M \in \mathcal{F}_\Sigma(m)\} \\ \mathcal{T}_\Sigma(n + 1, r, m) &= \mathcal{T}_\Sigma(n, r, m) \cup \{\mathit{AppendChar}(a, M) \,:\, M \in \mathcal{T}_\Sigma(n, r, m)\} \\ &\quad \cup \{\mathit{AppendRepeat}(M, k, O) \,:\, kp \leq r, \\ &\qquad M \in \mathcal{F}_\Sigma(p), O \in \mathcal{T}_\Sigma(n, r, m)\}. \end{aligned}$$

Proposition 2.6.13 *For every ultimately periodic word w having minimum prefix length l and minimum period q, there is an RLA $N \in \mathcal{T}_\Sigma(l + q, 2l + 2q, q)$ which is complexity-optimal for w.*

Proof. We can recursively apply Theorem 2.6.12 and end up with a complexity-optimal automaton N for u such that $N = N_n$, where

$$\begin{aligned} N_0 &= \mathit{AppendRepeat}(M_0, \omega) \\ \forall\, 1 \leq i \leq n.\ N_i &= \begin{cases} \text{either} & \mathit{AppendRepeat}(M_i, k_i, N_{i-1}) \\ \text{or} & \mathit{AppendChar}(a, N_{i-1}) \end{cases} \end{aligned}$$

for suitable natural numbers k_i and complexity-optimal automata M_i, with $0 \leq i \leq n$. It is easy to see that $n \leq l + q$. By contradiction, if $n > l + q$, then $\|N\| \geq l + q + 2$ follows, which implies that N is not complexity-optimal. Moreover, Theorem 2.6.12 implies that for every $1 \leq i \leq n$, $k_i|u_i| \leq 2l + 2q$, where u_i is the word recognized by M_i. From the same theorem we also have that $M_0 \in \mathcal{F}_\Sigma(q)$. Therefore, $N \in \mathcal{T}_\Sigma(l + q, q, 2l + 2q)$. ■

On the basis of the above results, we can devise a simple polynomial-time algorithm that solves the complexity-optimization problem for ultimately periodic words

(the algorithm for finite words is just a special case of it). Such an algorithm receives a pair (u, v) of finite words, where u and v are assumed to be of minimum length (recall that such words can be efficiently computed by the procedure *CanonicalRepresention*, given in Section 1.5), and it returns as output a complexity-optimal RLA that recognizes the ultimately periodic word $w = u \cdot v^{\omega}$. The algorithm uses the following data structures and procedures:

- a matrix $M_{fin}(i, j)$, where it stores the generated complexity-optimal automata recognizing the substrings $w[i, j]$ of w, for $1 \leq i \leq j \leq 3l + 3q$;
- an array $M_{inf}(i)$, where it stores the generated complexity-optimal automata recognizing the suffixes $w[i, \omega]$ of w, for $1 \leq i \leq l + q$ (we assume the array to be initialized with dummy automata of a very high complexity);
- a matrix $P(i, j)$, whose values are the periods of the substrings $w[i, j]$, for $1 \leq i \leq j \leq 3l + 3q$ (such periods are computed by the procedure *PeriodsOfAllSubstrings*, described in Section 1.5);
- an auxiliary procedure *BestComplexity*, which receives a finite set of RLA as input and it returns an RLA having minimum complexity;
- an auxiliary procedure $Substring(u, v, i, j)$, which returns the substring $w[i, j]$ of w;
- a routine $Normalize(i, l, q)$, which returns i if $i \leq l$ and $((i - l - 1) \bmod q) + l + 1$ otherwise (notice that, if w is an ultimately periodic word with prefix length l and period q, then, for every $i > 0$, $w[Normalize(i, l, q)] = w[i]$).

ComplexityOptimalRLA(u, v)

```
1: l ← |u|
2: q ← |v|
3: w ← Substring(u, v, 1, 3l + 3q)
4: (P(i, j))_{i∈[1,3l+3q], j∈[i,3l+3q]} ← PeriodsOfAllSubstrings(w)
5: for i = 1...3l + 3q do
6:    M_fin(i, i) ← AppendChar(w[i])
7: end for
8: for n = 2...3l + 3q do
9:    for i = 1...3l + 3q − n − 1 do
10:      N ← AppendChar(w[i], M_fin(i + 1, i + n − 1))
11:      p ← P(i, i + n − 1)
12:      if p ≠ n then
13:         N ← BestComplexity(N, AppendRepeat(M_fin(i, i + p − 1), n/p))
14:      end if
15:      for r = 1...n − 1 do
16:         p ← P(i, i + r − 1)
17:         N ← BestComplexity(N, AppendRepeat(M_fin(i, i + p − 1), r/p,
18:                                                M_fin(i + r, i + n − 1)))
19:      end for
20:      M_fin(i, i + n − 1) ← N
21:   end for
22: end for
```

23: **for** $i = l + 1...l + q$ **do**
24: $M_{inf}(i) \leftarrow AppendRepeat(M_{fin}(i, i + q - 1), \omega)$
25: **end for**
26: **for** $n = 1...l + q$ **do**
27: **for** $i = 1...l + q$ **do**
28: $N \leftarrow BestComplexity(M_{inf}(i),$
29: $AppendChar(w[i], M_{inf}(Normalize(i + 1, l, q))))$
30: **for** $r = 1...2l + 2q$ **do**
31: $p \leftarrow P(i, r)$
32: $N \leftarrow BestComplexity(N,$
33: $AppendRepeat(M_{fin}(i, i + p - 1), r/p,$
34: $M_{inf}(Normalize(i + r, l, q))))$
35: **end for**
36: $M_{inf}(i) \leftarrow N$
37: **end for**
38: **end for**
39: **return** $M_{inf}(1)$

Lines 5–22 (resp., 23–38) initialize the matrix M_{fin} (resp., M_{inf}). In particular, the cycle at lines 23–25 sets M_{inf} with complexity-optimal automata from $\mathcal{T}_\Sigma(0, q, 2l+2q)$, while the n-th iteration of the loop at lines 26–38 sets M_{inf} with complexity-optimal automata from $\mathcal{T}_\Sigma(n, q, 2l + 2q)$.
A straightforward implementation of the above algorithm requires time linear in the length of u and v to compute each automaton in M_{fin} and in M_{inf}. Therefore, $\mathcal{O}((|u| + |v|)^4)$ turns out to be an upper bound to the complexity of the problem. As a matter of fact, it is possible to compare the complexities of the generated automata without really building them. By exploiting definitions from Section 2.6.1 and by using suitable data structures, the complexity of each automaton can indeed be calculated in constant time, and thus we only need to explicitly generate the final complexity-optimal automaton for each substring $w[i, j]$. This allows us to conclude that a wiser implementation of the above algorithm would require time $\Theta((|u| + |v|)^3)$.

Computing size-optimal automata

Here we adapt the results we obtained for complexity-optimal automata to size-optimal ones. Unfortunately, we do not have an analogue of Proposition 2.6.9 for size-optimal automata. However, we can still find size-optimal automata with respect to the subclass $\mathcal{R}_\Sigma$. To do that we restrict the search space to $\mathcal{R}_\Sigma$ and we proceed exactly as in in the previous section. Hereafter, we define a size-optimal automaton for a given word w as an automaton in $\mathcal{R}_\Sigma$ having the minimum number of states (with respect to automata in $\mathcal{R}_\Sigma$) and recognizing u.

Theorem 2.6.14 *Given a finite word w, one of the following conditions holds:*
1. $|w| = 1$ *and* $AppendChar(a)$ *is size-optimal for w;*
2. $|w| > 1$ *and* $AppendChar(a, M)$ *is size-optimal for w, whenever M is size-optimal for $w[2, |w|]$;*

3. $|w| > 1$ *and* $AppendRepeat(M, |w|/p)$ *is size-optimal for* w*, whenever* M *is size-optimal for* $w[1, p]$*, where* p *is a period of* w*;*
4. $|w| > 1$ *and there exists* $r < |u|$ *such that* $AppendRepeat(M, r/p, O)$ *is size-optimal for* w*, whenever* M *is size-optimal for* $w[1, p]$*, where* p *is a period of* $w[1, r]$ *and* O *is size-optimal for* $w[r + 1, |w|]$*.*

Proof. Suppose that $N \in \mathcal{R}_\Sigma$ is a size-optimal automaton. We proceed by induction on the structure of N. We only consider the non-trivial cases (the remaining cases can be handled similarly).

- Suppose that $N = AppendChar(a, L)$ and let M be a size-optimal automaton recognizing $w[2, |u|]$. Then we have:

$$|AppendChar(a, M)| = 1 + |M| \leq 1 + |L| = |AppendChar(a, L)| = |N|$$

  This implies that $AppendChar(a, M)$ is size-optimal.
- Suppose that $N = AppendRepeat(L, k, P)$, where $k \in \mathbb{N}$. Let u be the finite word recognized by L, M be a size-optimal automaton for u, and O be a size-optimal automaton for $w[k|u|, |w|]$. We have:

$$|AppendRepeat(M, k, O)| = 1 + |M| + |O| \leq 1 + |L| + |P| = |N|$$

  and hence $AppendRepeat(M, k, O)$ is size-optimal.

■

Theorem 2.6.15 *Given an ultimately periodic word* w *having minimum prefix length* l *and minimum period* q*, at least one of the following conditions holds:*

1. $AppendChar(a, M)$ *is size-optimal for* w*, whenever* M *is size-optimal for* $w[2, \omega]$*;*
2. *there exists a multiple* r *of* q *such that* $AppendRepeat(M, \omega)$ *is size-optimal for* w*, whenever* M *is size-optimal for* $w[1, r]$*;*
3. *there exists* $r \leq 2l + 2q$ *such that* $AppendRepeat(M, r/p, O)$ *is size-optimal for* w*, whenever* M *is size-optimal for* $w[1, p]$*, where* p *is a period of* $w[1, r]$*, and* O *is size-optimal for* $w[r + 1, \omega]$*.*

Proof. Suppose that $N \in \mathcal{R}_\Sigma$ is a size-optimal automaton that recognizes w. We prove the thesis by induction on the structure of N. We only consider the most difficult case, that is, $N = AppendRepeat(L, k, P)$, with $k > 1$ (it is easy to show that k cannot be equal to 1 in a size-optimal automaton). First of all, we can replace the automata L and P by equivalent size-optimal automata M and O, respectively. Let u be the finite word recognized by M and let $r = k|u|$. We have to show that $r < 2l+2q$. Suppose, by contradiction, that $r \geq 2l + 2q$ and consider the substring $t = w[l + 1, r]$ of w, which has partial periods q and $|u|$. We have that $r \geq max(2l + 2q, 2|u|)$ and thus $r \geq l + q + |u|$, whence $|t| \geq q + |u|$. By Lemma 1.5.2, this implies that t has partial period $m = gcd(q, |u|)$. Consider now the substring $w[2l + 1, 2l + 2q]$ of t. It has period q (since $2l+1$ is greater than l) and partial period m. Since m divides q, m is in fact a period. From the fact that q is the minimum period, we have that $q = m$. It immediately follows that $|u|$ is a multiple of q $(= m)$, and thus u has period q.

Moreover, from the minimality of l, it follows that $l = 0$ (if $l > 0$, then $w[l] \neq w[l+q]$, but, from $r > l + q$, we have that $w[l] = w[l + q]$). Hence, the repeating pattern of u of length q is equal to $w[r + 1, r + q]$ and we have

$$\begin{aligned} w &= w[1, r] \cdot w[r + 1, \omega] = u^k \cdot w[r + 1, r + q]^\omega = \\ &= w[1, q]^h \cdot w[r + 1, r + q]^\omega = w[r + 1, r + q]^\omega = w[r + 1, \omega], \end{aligned}$$

which contradicts the hypothesis that N is size-optimal (O recognizes $w[r+1, \omega] = w$). ■

For every $n \in \mathbb{N}$ and for every $r, m, s \in \mathbb{N}^+$, we can define two classes of automata $\mathcal{S}_\Sigma(m, s), \mathcal{Q}_\Sigma(n, r, m, s) \subseteq \mathcal{R}_\Sigma$. These definitions allow us to reduce the search space to a finite set, when building size-optimal automata for infinite words. The definitions of $\mathcal{S}_\Sigma(m, s)$ and $\mathcal{Q}_\Sigma(n, r, m, s)$ are given by induction on m and n, respectively:

$$\begin{aligned} \mathcal{S}_\Sigma(1, s) &= \{AppendChar(a)\} \\ &\quad \cup \{AppendRepeat(M, k) \,:\, kp \leq s, M \in \mathcal{F}_\Sigma(p)\} \\ \mathcal{S}_\Sigma(m + 1, s) &= \mathcal{S}_\Sigma(m, s) \cup \{AppendChar(a, M) \,:\, M \in \mathcal{S}_\Sigma(m, s)\} \\ &\quad \cup \{AppendRepeat(M, k, O) \,:\, kp \leq s, \\ &\qquad M \in \mathcal{F}_\Sigma(p), O \in \mathcal{S}_\Sigma(m, s)\} \end{aligned}$$

$$\begin{aligned} \mathcal{Q}_\Sigma(0, r, m, s) &= \{AppendRepeat(M, \omega) \,:\, M \in \mathcal{S}_\Sigma(m, s)\} \\ \mathcal{Q}_\Sigma(n + 1, r, m, s) &= \mathcal{Q}_\Sigma(n, r, m, s) \\ &\quad \cup \{AppendChar(a, M) \,:\, M \in \mathcal{Q}_\Sigma(n, r, m, s)\} \\ &\quad \cup \{AppendRepeat(M, k, O) \,:\, kp \leq r, \\ &\qquad M \in \mathcal{F}_\Sigma(p), O \in \mathcal{Q}_\Sigma(n, r, m, s)\} \end{aligned}$$

As proved in the following proposition, we can assume that the parameters n, r, m, s are bounded by suitable functions linear in the prefix length and in the period of the given ultimately periodic word.

Proposition 2.6.16 *For every ultimately periodic word w having minimum prefix length l and minimum period q, there is an RLA $M \in \mathcal{Q}_\Sigma(l + q, 2l + 2q, q, 2q)$ which is size-optimal for w.*

Proof. We can recursively apply Theorem 2.6.15 and end up with a size-optimal automaton N for w such that $N = N_n$, where

$$\begin{aligned} N_0 &= AppendRepeat(M_0, \omega) \\ \forall\, 1 \leq i \leq n.\ N_i &= \begin{cases} \text{either} & AppendRepeat(M_i, k_i, N_{i-1}) \\ \text{or} & AppendChar(a, N_{i-1}) \end{cases} \end{aligned}$$

for suitable natural numbers k_i and size-optimal automata M_i, with $0 \leq i \leq n$. It is easy to see that $n \leq l + q$. By contradiction, if $n > l + q$, then $|N| \geq l + q + 2$,

which implies that N is not size-optimal. Moreover, Theorem 2.6.15 implies that for every $1 \leq i \leq n$, $k_i|u_i| \leq 2l + 2q$, where u_i is the string recognized by M_i. As for the automaton M_0, by Theorem 2.6.14, we can assume that $M_0 = Q_m$, where

$$Q_1 = \begin{cases} \text{either} & \mathit{AppendRepeat}(R_1, h_1) \\ \text{or} & \mathit{AppendChar}(a), \end{cases}$$

$$\forall\, 2 \leq i \leq m.\ Q_i = \begin{cases} \text{either} & \mathit{AppendRepeat}(R_i, h_i, Q_{i-1}) \\ \text{or} & \mathit{AppendChar}(a, Q_{i-1}). \end{cases}$$

Using an argument similar to the one we used to establish the bound $n \leq l + q$, one can easily show that $m \leq q$. It remains to show that for every $1 \leq i \leq m$, $h_i|v_i| \leq 2q$, where v_i is the string recognized by R_i. Suppose, by contradiction, that $h_i|v_i| > 2q$. First, notice that both q and $|v_i|$ are periods of $v_i^{h_i}$. Then, since $h_i \geq 2$, we have that $|v_i^{h_i}| \geq max(2q, 2|v_i|) \geq q + |v_i|$. Since q is the minimum period of w, Lemma 1.5.2 implies that $|v_i|$ is a multiple of q. Hence, the h_i-repetition of v_i is useless and $v_i^{h_i}$ can be replaced by v_i, which contradicts the hypothesis that N is size-optimal. This proves that $N \in \mathcal{Q}_\Sigma(l + q, 2l + 2q, q, 2q)$. ■

Putting all results together, we can solve the size-optimization problem as we solved the complexity-optimization one, provided that we restrict the search space to $\mathcal{R}_\Sigma$. We shall use an additional auxiliary procedure $\mathit{BestRepeat}(M_{\mathit{fin}}, P, i, j)$, that receives as input a matrix M_{fin}, which contains size-optimal automata recognizing substrings of a given word w, a matrix P, that contains the periods of the substrings of w, and two indices i, j, and it returns a pair (N, k), where N is a size-optimal automaton that recognizes a repeating pattern (not necessarily the minimum one) of $w[i, j]$ and h is the number of repetitions of that pattern in $w[i, j]$.

$\mathit{BestRepeat}(M_{\mathit{fin}}, P, i, j)$

```
1: p ← P(i, j)
2: N ← M_fin(i, i + p − 1)
3: k ← (j − i + 1)/p
4: for h = 2...(j − i + 1)/p do
5:    if (j − i + 1) mod (hp) = 0 and |N| > |M_fin(i, i + hp − 1)| then
6:       N ← M_fin(i, i + hp − 1)
7:       k ← (j − i + 1)/(hp)
8:    end if
9: end for
10: return (N, k)
```

Below we report the algorithm *SizeOptimalRLA*, which has almost the same structure as *ComplexityOptimalRLA*. The matrix $M_{rep}(m, i, j)$, where m, i, j range over the interval $[1, q]$, is used to store size-optimal automata from $\mathcal{S}_\Sigma(m, 2q)$ recognizing substrings of repeating patterns of w, namely, strings of the form $w[l + i, l + hq + j]$, with $h \in \mathbb{N}$. Furthermore, we assume M_{rep} to be initialized with dummy automata of very large size.

SizeOptimalRLA(u, v)

```
1: l ← |u|
2: q ← |v|
3: w ← Substring(u, v, 1, 3l + 3q)
4: (P(i, j))_{i∈[1,3l+3q],j∈[i,3l+3q]} ← PeriodsOfAllSubstrings(w)
5: for i = 1...3l + 3q do
6:    M_fin(i, i) ← AppendChar(w[i])
7: end for
8: for n = 2...3l + 3q do
9:    for i = 1...3l + 3q − n − 1 do
10:      N ← BestSize(AppendChar(w[i], M_fin(i + 1, i + n − 1)),
11:                   AppendRepeat(BestRepeat(M_fin, P, i, i + n − 1)))
12:      for r = 1...n − 1 do
13:         N ← BestSize(N, AppendRepeat(BestRepeat(M_fin, P, i, i + r − 1),
14:                                      M_fin(i + r, i + n − 1)))
15:      end for
16:      M_fin(i, i + n − 1) ← N
17:   end for
18: end for
19: for i = 1...q do
20:   M_rep(1, i, i) ← AppendChar(w[i])
21:   for s = 1...2q do
22:      j ← Normalize(i + s, 0, q)
23:      M_rep(1, i, j) ← BestSize(M_rep(1, i, j),
24:                                AppendRepeat(BestRepeat(M_fin, P, i, i + s)))
25:   end for
26: end for
27: for m = 2...q do
28:   for i = 1...q do
29:      for j = 1...q do
30:         h ← Normalize(i + 1, 0, q)
31:         M_rep(m, i, j) ← BestSize(M_rep(m − 1, i, j),
32:                                   AppendChar(w[i], M_rep(m − 1, h, j)))
33:      end for
34:      for s = 1...2q do
35:         C_new ← BestRepeat(M_fin, P, i, i + s)
36:         k ← Normalize(i + s + 1, 0, q)
37:         for h = 1...q do
38:            j ← Normalize(i + s + h, 0, q)
39:            M_rep(m, i, j) ← BestSize(M_rep(m, i, j),
40:                                      AppendRepeat(C_new, M_rep(m − 1, k, j)))
41:         end for
42:      end for
43:   end for
```

44: **end for**
45: **for** $i = 1...q$ **do**
46: $M_{inf}(l+i) \leftarrow AppendRepeat(M_{rep}(q, i, Normalize(i+q-1)), \omega)$
47: **end for**
48: **for** $n = 1...l+q$ **do**
49: **for** $i = 1...l+q$ **do**
50: $N \leftarrow BestSize(M_{inf}(i), AppendChar(w[i], M_{inf}(Normalize(i+1, l, q))))$
51: **for** $r = i...i+2l+2q-1$ **do**
52: $N \leftarrow BestSize(M_{inf}, AppendRepeat(BestRepeat(M_{fin}, P, i, i+r),$
53: $M_{inf}(Normalize(r+1, l, q))))$
54: **end for**
55: $M_{inf}(i) \leftarrow N$
56: **end for**
57: **end for**
58: **return** $M_{inf}(1)$

Lines 19–44 are used to fill the matrix M_{rep} with appropriate automata. This is done, in an inductive way, by first computing $M_{rep}(1, i, j)$ (lines 19–26) and then computing $M_{rep}(m, i, j)$, given $M_{rep}(m-1, i, j)$, (lines 27–44). The overall complexity is $\Theta((|u| + |v|)^4)$.

2.6.4 A simple example

In this section we give a simple, but meaningful, example that involves database relations, Calendar Algebra, algorithms mapping calendar expressions to equivalent string-based specifications, optimization algorithms, and granule conversion algorithms.

Let us suppose we want to register in a temporal database the rainfall amount for each day and support queries with generic user-defined granularities. Reasonably, the stored information can be expressed in terms of days, so we assume that there is a single temporal relation (or table) `Rainfall`, whose fields are `day` (the primary key) and `amount`, and a granularity system (or calendar), where `Hour` is the bottom granularity. Now, suppose a user wants to compute the rainfall amount during a given weekend. First of all, the system should be provided with a specification of the involved granularities. For this purpose, he may choose to use the formalism of Calendar Algebra. The following is a hypothetical specification of the granularities `Day`, `Week`, `WeekendDay`, and `Weekend` in terms of `Hour`:

1: add granularity `Day` as $Group_{24}$(`Hour`)
2: add granularity `Week` as $Group_7$(`Day`)
3: add granularity `WeekendDay` as $SelectDown_5^2$(`Day`, `Week`)
4: add granularity `Weekend` as $Combine$(`Week`, `WeekendDay`)

Once a specification of the involved granularities has been provided, an appropriate subsystem of the temporal database, which exploits the algorithms given in Section 2.4.2, can compute equivalent (low-level) string-based representations. In succession, by exploiting the results given in Section 2.6.3, the subsystem can compute

complexity-optimal RLA-based representations[5]. After that, the user is allowed to query the temporal database in order to obtain the rainfall amount during a given weekend, which is identified by an index $x \in \mathbb{N}$. This can be done, for instance, by using the SQL language extended with some granule comparison operators (hence, a TSQL-like language [106, 107]):

1: SELECT SUM(`amount`)
2: FROM `Rainfall`
3: WHERE `Day`(`day`) COVERED BY `WeekEnd`(x);

The above query can be efficiently evaluated by exploiting the (complexity-optimal) RLA-based representations for `Day` and `WeekEnd` and the granule conversion algorithms presented in Section 2.6.1. Note that, if M and N are two RLA representing, respectively, `Day` and `WeekEnd` in terms of `Second`, then each comparison between granules of `Day` and granules of `WeekEnd` takes at most linear time with respect to $\|M\|+\|N\|$ (if M and N are complexity-optimal RLA, then $\|M\| = 3$ and $\|N\| = 4$).

2.7 Reasoning on sets of granularities

In this section we generalize the automaton-based approach to deal with (possibly infinite) sets of ultimately periodic granularities, rather than single granularities. We first give a characterization of regular ω-languages consisting of ultimately periodic words only and then we exploit such a characterization to define a proper subclass of Büchi automata, called ultimately periodic automata (UPA), which consists of all and only the Büchi automata that recognize regular ω-languages of ultimately periodic words. UPA allow one to encode single granularities, sets of granularities which have the same repeating pattern and different prefixes, and sets of 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, we shall focus our attention on a number of basic problems involving sets of time granularities, precisely: the emptiness problem (that is to decide whether a given set of granularities is empty), the membership problem (that is to decide whether a given granularity belongs to a given set of granularities), the equivalence problem (that is to decide whether two given representations define the same set of granularities), the size-optimization problem (that is to compute compact representations of a given set of granularities), and the comparison of granularities (that is, given two sets of granularities $\mathcal{G}$ and $\mathcal{H}$, to decide whether there exist $G \in \mathcal{G}$

[5]It is worth noticing that the process of mapping calendar expressions to equivalent string-based representations and then to equivalent RLA-based representations is rather expensive, especially for granularities like `Month` and `Year`. An alternative choice may consist in mapping calendar expressions to RLA-based representations by modifying the algorithms of Section 2.6.3 in such a way that they can work directly on RLA (note that the auxiliary functions *SymbolAt*, *Substring*, *CountOccurrences*, etc. can be efficiently implemented on strings as well as on RLA). This choice significantly reduces the time to process calendar expressions, even though the resulting RLA-based representations are not guaranteed to be size-optimal.

and $H \in \mathcal{H}$ such that $G \sim H$, where $\sim$ is one of the commonly-used relations between granularities, e.g, partition, group, sub-granularity, aligned refinement, etc.).

2.7.1 Languages of ultimately periodic words

Here, we investigate on properties of regular ω-languages consisting of ultimately periodic words only. To start with, we recall a well-known characterization of Büchi-recognizable (i.e., regular) ω-languages.

Proposition 2.7.1 *An ω-language L is Büchi-recognizable iff it is a finite union of sets of the form $U \cdot V^\omega$, where U and V are regular languages of finite words.*

Hereafter, given a language $L \subseteq \Sigma^*$, we denote by $L^{\vec{\omega}}$ the set $\{v^\omega : v \in L \setminus \{\varepsilon\}\}$. Moreover, we denote by U_Σ the universal ω-language $\Sigma^* \cdot (\Sigma^*)^{\vec{\omega}}$, which consists of all and only the ultimately periodic words over Σ. Finally, for any given ω-language $L \subset \Sigma^\omega$, we denote by $\mathcal{U}p(L)$ the set $L \cap U_\Sigma$, which consists of all and only the ultimately periodic words belonging to L. Note that an ω-language L consists of only ultimately periodic words iff $L = \mathcal{U}p(L)$.

Proposition 2.7.2 (Calbrix et al. [16]) *Every non-empty regular ω-language contains at least one ultimately periodic word. Moreover, if L_1 and L_2 are two regular ω-languages, then $L_1 = L_2$ iff $\mathcal{U}p(L_1) = \mathcal{U}p(L_2)$.*

Proof. As for the first claim, by Proposition 2.7.1, any regular ω-language L can be written as $\bigcup_{i\in[n]} U_i \cdot V_i^\omega$. Since L is not empty, it contains a word of the form $w = u \cdot v_1 \cdot v_2 \cdot ...$, with $u \in U_i$ and $v_1, v_2, ... \in V_i$ for some index $i \in [n]$. Thus the ultimately periodic word $w' = u \cdot (v_1)^\omega$ belongs to L as well.
As for the second claim, let L_1 and L_2 be two regular 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 Büchi-recognizable languages (cf. Proposition 1.4.2), that $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is a regular ω-language which does not contain ultimately periodic words. Thus $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is empty and $L_1 = L_2$ follows. ■

Clearly, there exist non-regular ω-languages consisting of ultimately periodic words only: for instance, the ω-language U_Σ is not regular because $\mathcal{U}p(U_\Sigma) = \mathcal{U}p(\Sigma^\omega)$ holds and Σ^ω is regular (cf. Proposition 2.7.2). The rest of this section is devoted to find a characterization of regular ω-languages of ultimately periodic words, similar to that of Proposition 2.7.1.

Proposition 2.7.3 *The following closure properties hold:*

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

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

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

Proof. These properties follow trivially from closure properties of regular ω-languages. ■

As for the complementation of a regular ω-language of ultimately periodic words, it is clear that we need to adopt a slightly different notion of complementation, since the standard one introduces non-ultimately periodic words. We define the *restricted complementation* of an ω-language $L \subseteq \Sigma^\omega$ as the set $\bar{L} = U_\Sigma \setminus L$. Even though the restricted complementation $\bar{L}$ of an ω-language L contains only ultimately periodic words, this does not guarantee that $\bar{L}$ is regular whenever L is regular.

Proposition 2.7.4 *Regular ω-languages of ultimately periodic words are not closed under restricted complementation.*

Proof. A counterexample is given by the empty set, which is clearly a regular ω-language of ultimately periodic words, but its restricted complement is the universal ω-language U_Σ, which is not regular. As a matter of fact, it is easy to see that the restricted complement of *any* regular ω-language L of ultimately periodic words is not regular, since $U_\Sigma = L \cup \bar{L}$. ■

Given an ultimately periodic word $w = u \cdot v^\omega$, the set of its repeating patterns is clearly infinite (e.g., it contains v, v^2, v^3, ...), thus it is useful to define an appropriate equivalence relation that groups together the repeating patterns of the same word. Such a notion of equivalence is the starting point for a characterization of regular ω-languages of ultimately periodic words.

Definition 2.7.5 *We define the equivalence relation $\cong \subseteq \Sigma^* \times \Sigma^*$ such that $u \cong v$ iff the two infinite periodic words u^ω and v^ω share a common suffix, namely, there exist $x, y \in \Sigma^*$ and $z \in \Sigma^\omega$ such that $u^\omega = x \cdot z$ and $v^\omega = y \cdot z$.*

We say that two repeating patterns v and v' are *equivalent* if $v \cong v'$. Note that all repeating patterns of a given ultimately periodic word w are equivalent. In particular, they can be obtained by choosing different rotations and/or different repetitions of the primitive repeating pattern of w. 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.

We now investigate on properties of regular ω-languages of the form V^ω. In the sequel, we shall exploit these results to give a characterization of regular ω-languages of the form $U \cdot V^\omega$ consisting of ultimately periodic words only and then we generalize such a characterization to regular ω-languages of ultimately periodic words (recall that the latter ones are finite unions of regular ω-languages of the form $U \cdot V^\omega$). We say that an ω-language L *features* a repeating pattern v iff it contains an ultimately periodic word w having v as a repeating pattern.

Lemma 2.7.6 *Given a regular language V, for every repeating pattern v featured by V^ω, there exists an equivalent repeating pattern v' such that $(v')^\omega \in V^\omega$.*

Proof. Let M be a finite-state automaton recognizing V. We can assume, without loss of generality, that M has a unique initial state s_0. We define the Büchi automaton M' by (i) adding a new transition of the form (s, a, s_0) for each transition (s, a, s') of M, with s' being a final state of M, and (ii) letting s_0 be the unique final state of M'. Clearly, M' recognizes the ω-language V^ω. Let v be a repeating pattern featured by V^ω. By definition, V^ω contains an ultimately periodic word of the form $w = u \cdot v^\omega$. Let ρ be a successful run of M' on w. By definition of Büchi acceptance condition, there exist infinitely many occurrences of s_0 in ρ and hence we can define the position i of the first occurrence of s_0 in $\rho[|u|+1, \omega]$. Clearly, $\rho' = \rho[|u|+i, \omega]$ is a successful run of M' on the suffix $w[|u|+i, \omega]$ of w. Moreover, note that $w[|u|+i, \omega] = (v')^\omega$, where $v' = v[j+1, |v|] \cdot v[1, j]$ and $j = (i-1) \bmod |v|$. This proves that $(v')^\omega \in V^\omega$ and $v' \cong v$. ∎

Given a language V and a repeating pattern v featured by V^ω, we say that v is *V-aligned* if $v \in V^*$.

Lemma 2.7.7 *Given a regular language V, for every repeating pattern featured by V^ω, there exists an equivalent V-aligned repeating pattern.*

Proof. Let M be a finite-state automaton recognizing V and let M' be the Büchi automaton recognizing V^ω, defined as in the previous proof. We know that s_0 is the unique initial and final state of M'. Let v be a repeating pattern featured by V^ω. By Lemma 2.7.6, there is v' such that $(v') \in V^\omega$ and $v' \cong v$. Let ρ be a successful run of M' on $(v')^\omega$ and consider the infinite sequence $\rho_p = \rho[1]\rho[p+1]\rho[2p+1]...$, where $p = |v'|$. We then define:

- a state s of M that occurs infinitely often in ρ_p,
- two integers $i < j \in \mathbb{N}$ such that $\rho_p[i+1] = \rho_p[j+1] = s$ and $\rho[ip+1, jp]$ contains at least one occurrence of s_0 (recall that s_0 occurs infinitely often in ρ),
- the position k in $\rho[ip+1, jp]$ of an occurrence of s_0.

Now, by viewing M' as a finite-state automaton recognizing the language V^*, we have that $\rho[ip+1, jp+1]$ is a run of M' on the finite word $(v')^{j-i}$ and hence $\rho' = \rho[ip+k, jp+1] \cdot \rho[ip+1, ip+k]$ is a run of M' on the finite word $v'' = (v'[k, p] \cdot v'[1, k-1])^{j-i}$. Moreover ρ' is a successful run of M' since it starts and ends with the state s_0. This shows that $v'' \in V^*$ and $v'' \cong v' \cong v$. ∎

Proposition 2.7.8 *Given a regular ω-language $L = U \cdot V^\omega$, exactly one of the following conditions holds:*

1. *L features only equivalent repeating patterns,*
2. *L features infinitely many non-equivalent repeating patterns and it contains a word which is not ultimately periodic.*

Proof. Let $L = U \cdot V^\omega$ be a regular ω-language and assume that L features at least two non-equivalent repeating patterns v_1 and v_2. This implies that V^ω features the repeating patterns v_1 and v_2 as well. Hence, by Lemma 2.7.7, there are two V-aligned repeating patterns v_1' and v_2' such that $v_1' \cong v_1$ and $v_2' \cong v_2$. Since $v_1' \ncong v_2'$, it is easy to see that for every pair of distinct indices $i, j > 0$, the repeating patterns

$(v_1')^i \cdot (v_2')^i$ and $(v_1')^j \cdot (v_2')^j$ are non-equivalent. Moreover, the infinite word $w = (v_1') \cdot (v_2') \cdot (v_1')^2 \cdot (v_2')^2 \cdot (v_1')^3 \cdot (v_2')^3 \cdot ...$ is not ultimately periodic word and it belongs to V^ω. Hence L features infinitely many non-equivalent repeating patterns and it contains an infinite word which is not ultimately periodic. ■

Proposition 2.7.9 *Any regular ω-language $L = U \cdot V^\omega$ that features only equivalent repeating patterns can be written as $U \cdot \{v\}^\omega$ (hence it contains only ultimately periodic words).*

Proof. Let $L = U \cdot V^\omega$ be a regular ω-language featuring only equivalent repeating patterns. Thus, V^ω as well features only equivalent repeating patterns. We first show that for every pair words $v_1, v_2 \in V$, $v_1^\omega = v_2^\omega$ holds. Suppose that $v_1, v_2 \in V$. Since V^ω features only equivalent repeating patterns, we have that $v_1 \cong v_2$. We define two positive natural numbers k, h such that $k|v_1| = h|v_2|$ and we let $w_1 = v_1^k$ and $w_2 = v_2^k$. Note that $w_1 \cong w_2$ holds and hence there are two finite words x, y and an infinite word z such that $w_1 = x \cdot z$ and $w_2 = y \cdot z$. Now, suppose, by contradiction, that $w_1 \neq w_2$. Since $|w_1| = |w_2|$, we know that w_1 is a rotation of w_2, namely, $w_1 = u \cdot u'$ and $w_2 = u' \cdot u$, for suitable finite words u, u'. Hence, the infinite periodic word $(w_1 \cdot w_2)^\omega = (u \cdot u' \cdot u' \cdot u)^\omega$ belongs to V^ω but it is not equivalent neither to w_1 nor to w_2, which contradicts the hypothesis that V^ω features only equivalent repeating patterns. Hence we have $w_1 = w_2$, from which $v_1^\omega = v_2^\omega$ follows. This holds for every pair of repeating patterns featured by V^ω. Therefore, if we denote by $\{v_1, v_2, v_3, ...\}$ the set of all and only the repeating patterns featured by V^ω, there is a finite word v of length $gcd(|v_1|, |v_2|, |v_3|, ...)$ such that $v^\omega = v_1^\omega = v_2^\omega = v_3^\omega = ...$ This shows that $V^\omega = \{v\}^\omega$. ■

Now, we can provide a characterization of regular ω-languages consisting of ultimately periodic words only.

Theorem 2.7.10 *Given any regular 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 $U \cdot \{v\}^\omega$, where U is a regular language and v is a non-empty finite word.

Proof. The implication from i) to ii) is easily shown by contraposition with Proposition 2.7.8. Let L be a regular ω-language. We can write L as a finite union of the form $\bigcup_{i \in [n]} U_i \cdot V_i^\omega$. If L featured infinitely many non-equivalent repeating patterns, then there would exist an index $i \in [n]$ such that $U_i \cdot V_i^\omega$ features infinitely many non-equivalent repeating patterns. Then, by Proposition 2.7.8, it would follow that L contains a word which is not ultimately periodic.
As for the implication from ii) to iii), let L be a regular ω-language featuring only finitely many non-equivalent repeating patterns. We write L as $\bigcup_{i \in [n]} U_i \cdot V_i^\omega$. By Proposition 2.7.8, every ω-language $U_i \cdot V_i^\omega$ features only equivalent repeating patterns (otherwise it would feature infinitely many non-equivalent repeating patterns) and

hence, by Proposition 2.7.9, L can be written as $\bigcup_{i\in[n]} U_i \cdot \{v_i\}^\omega$, for suitable non-empty finite words $v_1, ..., v_n$.
The last implication from iii) to i) is trivial. ■

2.7.2 Ultimately periodic automata

From Theorem 2.7.10, it follows that regular ω-languages of ultimately periodic words capture sets of granularities with possibly infinitely many prefixes, but with only a finite number of non-equivalent repeating patterns. Moreover, Theorem 2.7.10 yields a straightforward definition of a restricted class of Büchi automata that captures all and only the regular ω-languages of ultimately periodic words. In this section we formally define such a class of automata.

By viewing a Büchi automaton M as a finite labeled graph, we define the *strongly connected component* of a state s of M as the subgraph of M induced by the maximal set of states that are reachable from s and from which s is reachable as well. Moreover, a state s of M is called *transient* if either it is not reachable from the initial state or it does not belong to any cycle of M.

Definition 2.7.11 *An* ultimately periodic automaton *(UPA for short) is a Büchi automaton $M = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ such that for every final state $s \in \mathcal{F}$, the strongly connected component of s is either a single transient state or a simple loop*[6].

Figure 2.20 depicts two UPA recognizing the ω-languages $\big(\{\square\}^* \cdot \{\blacksquare\blacktriangleleft\}^\omega\big) \cup \big(\{\square\}^* \cdot \{\blacksquare\blacktriangleleft\}^* \cdot \{\square\}^\omega\big)$ and $\{\blacksquare\blacksquare\blacktriangleleft\}^\omega \cup \{\blacksquare\blacktriangleleft\}^\omega$, respectively. The former one represents the (unanchored finite or infinite) granularities that groups days two by two, while the latter one represents the set consisting of two infinite granularities that group days respectively two by two and three by three.

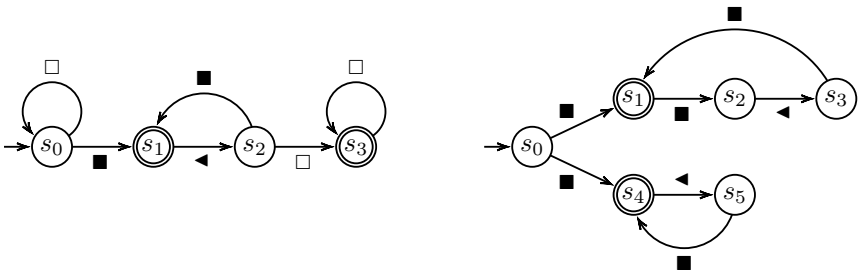

Figure 2.20: Two examples of UPA.

Theorem 2.7.12 *UPA recognize all and only the regular ω-languages of ultimately periodic words.*

[6] A simple loop is a cycle where each vertex has in-degree 1 and out-degree 1.

Proof. Let $M = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ be an UPA, let w be an infinite word recognized by M, and let ρ be a successful run of M on w. We denote by s a final state of M 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 exists only a single infinite run of M starting from s and visiting s infinitely often. Such a run is a suffix of ρ of the form $\rho' = (\rho[i]\rho[i+1]...\rho[j-1])^\omega$, 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. Therefore M accepts only ultimately periodic words.
As for the converse implication, we have to show that, given a regular ω-language L of ultimately periodic words, there is an UPA recognizing L. By exploiting Theorem 2.7.10, we know that $L = \bigcup_{i\in[n]} U_i \cdot \{v_i\}^\omega$ for a suitable n, for suitable regular languages $U_1, ..., U_n$, and for suitable finite non-empty finite words $v_1, ..., v_n$. Such a characterization implicitly defines the three basic operations on UPA: the ω-exponentiation of a non-empty finite word, the concatenation with a regular language, and the finite union. Thus, from closure properties of UPA, we know that there exists an UPA recognizing L. ■

Notice that, by exploiting the standard construction methods for Büchi automata, one can easily show that UPA are effectively closed with respect to union and (non-synchronized) product, namely, given two UPA M and N, one can compute an UPA $M \cup N$ recognizing the ω-language $\mathscr{L}^\omega(M) \cup \mathscr{L}^\omega(N)$ and an UPA $M \times N$ recognizing the ω-language

$$\mathscr{L}^\omega(M) \times \mathscr{L}^\omega(N) = \big\{w : \exists\, u \in \mathscr{L}^\omega(M).\ \exists\, v \in \mathscr{L}^\omega(N).\ \forall\, i > 0.\ w[i] = (u[i], v[i])\big\}.$$

UPA are also closed with respect to intersections with Büchi-recognizable languages, namely, given an UPA M and a Büchi automaton N, one can compute an UPA $M \cap N$ recognizing the ω-language $\mathscr{L}^\omega(M) \cap \mathscr{L}^\omega(N)$ (note that such a language consists of ultimately periodic words only and it features only finitely many non-equivalent repeating patterns). Moreover, 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 ω-language $\{v\}^\omega$. On the other hand, UPA are not closed under (restricted) complementation: this follows from Proposition 2.7.4 and from the fact that UPA capture all and only the regular ω-languages of ultimately periodic words. Finally, it is also clear that *non-determinism* is necessary in order to allow UPA to capture some regular ω-languages of ultimately periodic words (e.g., the ω-language $\{a, b\}^* \cdot \{b\}^\omega$).

Given a loop C of an UPA $M = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$, we say that C is a *final loop* if C is reachable from an initial state of M and $\mathcal{D}om(C) \cap \mathcal{F} \neq \emptyset$ (namely, C contains at least one final state). We then say that a final loop C *encodes* the repeating pattern v if $v \neq \varepsilon$ and there exists a state $s \in \mathcal{D}om(C)$ and a positive integer k such that $s \rightarrow^{v^k} s$, where $\rightarrow^v$ is the relation recursively defined as follows: $s \rightarrow^\varepsilon s$ for every $s \in S$, $s \rightarrow^a s'$ for every $(s, a, s') \in \delta$, and $s \rightarrow^{v\cdot a} s'$ whenever there is $s'' \in S$ such that $s \rightarrow^v s''$ and $s'' \rightarrow^a s'$. It's easy to see that a final loop C encodes only equivalent repeating patterns and, conversely, if v and v' are equivalent repeating patterns, then

C encodes v iff C encodes v'. 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 pairs of repeating patterns which are two-by-two non-equivalent.

Due to the particular 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, notice that, 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, transient final states are not necessary to recognize a given ultimately periodic word. As a last remark, note that we can assume that no transitions exit from final loops (if this were the case, we can simply duplicate the final loop and let one copy of it to be a final loop without exiting transitions and the other copy to be a non-final loop with some exiting transitions). These observations leads to the following definition of normal form of UPA, which forces final states to be all and only the states that belong to final loops and forbids transitions exiting from final loops.

Definition 2.7.13 *An UPA $M = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ is said to be in* normal form *if the following conditions hold:*

- *the set $\mathcal{I}$ of initial states is a singleton $\{s_0\}$, where s_0 has no entering transition,*
- *for every $s \in \mathcal{F}$, there exist a finite word u and a finite non-empty word v such that $s_0 \rightarrow^u s$ and $s \rightarrow^v s$, namely, every final state of M belongs to a final loop,*
- *for every $s \in \mathcal{F}$, if $s \rightarrow^v s'$, then there is a finite word v' such that $s' \rightarrow^{v'} s$, namely, M contains no transitions exiting from final loops,*
- *for every $s \in \mathcal{F}$, if $s \rightarrow^v s'$,then $s' \in \mathcal{F}$, namely, every state in a final loop is final.*

The above conditions allow us to easily distinguish between components recognizing finite prefixes and components recognizing repeating patterns of ultimately periodic words (note that the former components behaves like non-deterministic finite-state automata, while the latter ones behave like single-string automata). The following theorem shows that normal forms do not reduce the expressiveness of UPA. This allows us to considerably simplify algorithmic manipulations by restricting to UPA in normal forms.

Proposition 2.7.14 *For every UPA M, there is an equivalent UPA M' in normal form. Moreover, the number of states and transitions of M' is linear in the number of states and transitions of M.*

Proof. Let $M = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ and let $C_1, ..., C_n$ be all and only the final loops of M (note that, by definition of UPA, these are disjoint subsets of S). We define $M' = (S', \Sigma, \delta', \{s'_0\}, \mathcal{F}')$ as follows:

- $S' = \{s'_0\} \cup S \cup \bigcup_{i \in [n]} \mathcal{D}om(\bar{C}_i)$, where $s'_0 \notin S$ and for every $i \in [n]$, $\bar{C}_i$ is a copy of C_i disjoint from S (for each $s \in \mathcal{D}om(C_i)$, we denote by $\bar{s}$ the state of $\mathcal{D}om(\bar{C}_i)$ corresponding to s);
- for every $(s, a, s') \in \delta$, $(s, a, s') \in \delta'$, moreover, we add to δ' the following transitions

i) (s'_0, a, s'), whenever $(s, a, s') \in \delta$ with $s \in \mathcal{I}$,
ii) $(s, a, \bar{s}')$, whenever $(s, a, s') \in \delta$ with $s' \in C_i$ for some $i \in [n]$,
iii) $(\bar{s}, a, \bar{s}')$, whenever $(s, a, s') \in \delta$ with $s, s' \in C_i$ for some $i \in [n]$;

- $\mathcal{F}' = \bigcup_{\in [n]} \bar{C}_i$.

It is routine to verify that M' is an UPA in normal form equivalent to M. ■

On the grounds of Proposition 2.7.14, we can devise a simple algorithm that receives a generic UPA in input and returns an equivalent UPA in normal form.

UPANormalForm(M)

```
1: let M = (S, Σ, δ, I, F)
2: s'_0 ← a new state not in S
3: for all s ∈ S do
4:    s̄ ← a new copy of the state s
5: end for
6: S' ← {s'_0} ∪ S
7: F' ← ∅
8: for all s ∈ S do
9:    if s belongs to a final loop of M then
10:      S' ← S' ∪ {s̄}
11:      F' ← F' ∪ {s̄}
12:   end if
13: end for
14: δ' ← ∅
15: for all (s, a, s') ∈ δ do
16:   if s ∈ I then
17:      δ' ← δ' ∪ {(s'_0, a, s')}
18:   end if
19:   if s' belongs to a final loop of M then
20:      δ' ← δ' ∪ {(s, a, s̄')}
21:   end if
22:   if s, s' belong to the same final loop of M then
23:      δ' ← δ' ∪ {(s̄, a, s̄')}
24:   end if
25: end for
26: M' ← (S', Σ, δ', {s'_0}, F')
27: return M'
```

UPA can be successfully exploited to efficiently solve a number of fundamental problems involving sets of granularities:

- **Emptiness**. The emptiness problem is the problem of deciding whether a given UPA recognizes the empty language. In the case of a Büchi automaton, the emptiness problem is solved in linear time 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 is reduced to the problem of searching for a path from the initial state to a final state.

- **Membership**. The membership problem consists in deciding whether a given UPA M recognizes a given ultimately periodic word w (represented as a pair u, v) consisting of a prefix and a repeating pattern). Since UPA are closed with respect to intersection, the membership problem is reducible to the emptiness problem for the ω-language $\mathscr{L}^\omega(M) \cap \mathscr{L}^\omega(N)$, where N is an UPA that recognizes the singleton $\{u \cdot v^\omega\}$.
- **Equivalence and inclusion**. The equivalence problem (resp., the inclusion problem) is the problem of deciding whether, given two UPA M and N, $\mathscr{L}^\omega(M) = \mathscr{L}^\omega(N)$ (resp., $\mathscr{L}^\omega(M) \subseteq \mathscr{L}^\omega(N)$). It is known that these problems are inter-reducible, namely, $\mathscr{L}^\omega(M) = \mathscr{L}^\omega(N)$ iff $\mathscr{L}^\omega(M) \subseteq \mathscr{L}^\omega(N) \wedge \mathscr{L}^\omega(N) \subseteq \mathscr{L}^\omega(M)$ and, similarly, $\mathscr{L}^\omega(M) \subseteq \mathscr{L}^\omega(N)$ iff $\mathscr{L}^\omega(M) = \mathscr{L}^\omega(M) \cap \mathscr{L}^\omega(N)$). Moreover, in the general framework of Büchi automata, the equivalence and the inclusion problems can be reduced to the emptiness problem: indeed $\mathscr{L}^\omega(M) \subseteq \mathscr{L}^\omega(N)$ holds iff $\mathscr{L}^\omega(M) \cap \mathscr{L}^\omega(\bar{N})$ is the empty language, where $\bar{N}$ is the Büchi automaton recognizing the complement of $\mathscr{L}^\omega(N)$. Since the complement automaton $\bar{N}$ is exponentially big in the size of N, this leads to an algorithm that tests the equivalence of two given Büchi automata in exponential time and space. In [105] an *implicit* construction of the complement automaton has been provided, which allows one to solve the equivalence problem for Büchi automata in polynomial space. Such a construction is based on the possibility of encoding each state and checking each transition of the complement automaton by using only a polynomial amount of space. Since, in the worst case, the size (i.e., number of states) of the complement automaton $\bar{N}$ is at least $2^{n \log n}$, where n is the size of N, (see [75, 71] for lower-bound results and [99, 98, 88] for constructions that match these bounds), it turns out that that any (either deterministic or non-deterministic) algorithm based on (either explicit or implicit) construction of a complement automaton, must use $\Omega(n \log n)$ space. In the following section, we first give a polynomial lower bound on the space complexity of the equivalence and inclusion problems for UPA. Then, we present two algorithms that solve these problems and require, respectively, *non-deterministic linear space* and *deterministic exponential time* (i.e., $\mathcal{O}(2^n)$) with respect to the size of the input. While the former algorithm has a lower space complexity compared to classical non-deterministic algorithms for the equivalence of Büchi automata, the latter one outperforms classical deterministic algorithms, which are all based on the construction of a complement Büchi automaton.
- **Size-optimization**. The size-optimization problem consists in computing (possibly the most) compact representation of a given set of time granularities. Such a problem is usually connected to the equivalence problem, since, in many cases, size-optimal automata turn out to be unique up to isomorphisms. Like in the case of non-deterministic finite-state automata (see [65, 66, 74]), the size-optimization problem for UPA turns out to be hard (PSPACE-complete) and it may yield different (non-isomorphic) solutions. In Section 2.7.4, we provide an algorithm that solves the size-optimization problem for UPA in normal form. The algorithm can

be further generalized in order to deal with generic UPA; however, in this case, it can only approximate size-optimal UPA starting from size-optimal UPA in normal form.

- **Comparison of granularities**. Granularity comparison problems consist in checking whether a distinguished relation $\sim$ (e.g., partition, group, sub-granularity, aligned refinement, etc.) holds between two granularities $G \in \mathcal{G}$ and $H \in \mathcal{H}$, where $\mathcal{G}$ and $\mathcal{H}$ are given sets of granularities. Standard relations between granularities can be tested by looking at their automaton-based representations. This allows us to reduce granularity comparison problems to the emptiness problem for suitable product automata.

2.7.3 The equivalence problem

We first show that the equivalence problem and the inclusion problem for UPA are PSPACE-hard.

Proposition 2.7.15 *The equivalence problem and the inclusion problem for UPA are PSPACE-hard under LOGSPACE reductions.*

Proof. We provide a LOGSPACE reduction from the equivalence problem for non-deterministic finite-state automata (which is known to be PSPACE-complete, cf. [64]) to the equivalence problem for UPA. Let M and N be two generic finite-state automata recognizing the languages $\mathscr{L}(M)$ and $\mathscr{L}(N)$, respectively. We extend the input alphabet with a fresh symbol $\#$ and we build an UPA M' (resp., an UPA N') by adding to M (resp., N) (i) a new final state f, (ii) a transition from each final state of M (resp., N) to f reading the symbol $\#$, and (iii) a loop from f to f reading the symbol $\#$. Clearly, M' and N' recognize, respectively, the ω-languages $\mathscr{L}^\omega(M') = \mathscr{L}(M) \cdot \#^\omega$ and $\mathscr{L}^\omega(N') = \mathscr{L}(N) \cdot \#^\omega$ and thus we have $\mathscr{L}^\omega(M') = \mathscr{L}^\omega(N')$ iff $\mathscr{L}(M) = \mathscr{L}(N)$. This shows that the equivalence problem (hence the inclusion problem as well) for UPA is PSPACE-hard under LOGSPACE reductions. ■

The non-deterministic solution

We now sketch a *non-deterministic* algorithm that solves the (non-)inclusion problem (hence the equivalence problem) in *linear space* with respect to the size of the input UPA. Given two UPA $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$ and $N = (S', \Sigma, \delta', \{s'_0\}, \mathcal{F}')$ in normal form, the basic idea is to search for an ultimately periodic word w that belongs to $\mathscr{L}^\omega(M)$ but not to $\mathscr{L}^\omega(N)$, thus certifying that $\mathscr{L}^\omega(M) \not\subseteq \mathscr{L}^\omega(N)$. The algorithm non-deterministically guesses such an ultimately periodic word $w = u \cdot v^\omega \in \mathscr{L}^\omega(M)$ and tests whether no successful run of N on w exists. Note that the last condition can be checked while guessing the prefix u and the repeating pattern v of w, by computing, at each step, the set of reachable configurations of M and N. We distinguish between two phases of the algorithm. During the first phase, a prefix u is guessed and the set of reachable states of M and N are computed, until a final state of M has been found. In this phase, configurations are defined as pairs of the form (Q_i, Q'_i), where $Q_i \subseteq S$

and $Q'_i \subseteq S'$. The algorithm starts with the initial configuration (Q_0, Q'_0), where $Q_0 = \{s_0\}$ and $Q'_0 = \{s'_0\}$, and, at each step, it non-deterministically chooses the next symbol $u[i+1]$ and it computes the next configuration (Q_{i+1}, Q'_{i+1}) as follows:

$$Q_{i+1} = \{r \in S : \exists\, s \in Q_i.\ (s, u[i+1], r) \in \delta\}$$
$$Q'_{i+1} = \{r' \in S' : \exists\, s' \in Q'_i.\ (s', u[i+1], r') \in \delta'\}.$$

If, at some step n, $Q_n \cap \mathcal{F} \neq \emptyset$ holds, then M recognizes an ultimately periodic word having $u = u[1]...u[n]$ as a prefix. At this point, the algorithm non-deterministically chooses a final state $s_{loop} \in Q_n \cap \mathcal{F}$ and switches to the second phase. Note that, even though the first phase can be carried on for an arbitrarily long time (this is the case when a loop of non-final states of M is reached), we can assume n to be less than $2^{|S|+|S'|}$. Indeed, if this were not the case, then, from the Pigeonhole Principle, $(Q_n, Q'_n) = (Q_{n'}, Q'_{n'})$ would hold for some $n' < 2^{|S|+|S'|}$ and hence the prefix $u[1]...u[n']$ would be a good candidate as well. During the second phase, the computation proceeds in a *deterministic* way, with configurations being pairs of the form (r_j, R'_j), where $r_0 = s_{loop}$, $R'_0 = Q'_n$, r_{j+1} is the unique state such that $(r_j, v[j+1], r_{j+1}) \in \delta$ for some $v[j+1] \in \Sigma$ (recall that M is in normal form), and $R'_{j+1} = \{r' \in S' : \exists\, s' \in R'_j.\ (s', v[j+1], r') \in \delta'\}$. Clearly, there exist $m < m' \leq |S|2^{|S'|}$ such that $(R_m, R'_m) = (R_{m'}, R'_{m'})$. Moreover, if $R'_{m'} \cap \mathcal{F}' = \emptyset$, then $(u[1]...u[n]v[1]...v[m]) \cdot (v[m+1]...v[m'])^\omega$ is an ultimately periodic word recognized by M but not by N, thus certifying that $\mathscr{L}^\omega(M) \not\subseteq \mathscr{L}^\omega(N)$. Otherwise, the computation is discarded. Below, we report the resulting algorithm (note that the UPA provided in input are supposed to be in normal form and they can be computed from generic UPA using a linear amount of space only).

UPANonInclusion(M, N)

let $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$
let $N = (S', \Sigma, \delta', \{s'_0\}, \mathcal{F}')$
$Q \leftarrow \{s_0\}$
$Q' \leftarrow \{s'_0\}$
$i \leftarrow 0$
for $i = 1...2^{|S|+|S'|}$ **do**
  $a \leftarrow$ guess a symbol from Σ
  $Q \leftarrow \{r \in S : \exists\, s \in Q.\ (s, a, r) \in \delta\}$
  $Q' \leftarrow \{r' \in S' : \exists\, s' \in Q'.\ (s', a, r') \in \delta'\}$
end for
if $Q \cap \mathcal{F} \neq \emptyset$ **then**
  $r \leftarrow$ guess a state from $Q \cap \mathcal{F}$
else
  return false
end if
$R' \leftarrow Q'$
for $j = 1...|S| * 2^{|S'|}$ **do**
  $a \leftarrow$ the unique symbol $a \in \Sigma$ satisfying $\exists\, r' \in S'.\ (r, a, r') \in \delta$

$r \leftarrow$ the unique state $r' \in S'$ satisfying $(r, a, r') \in \delta$
$R' \leftarrow \{r' \in S' : \exists\, s' \in R'.\ (s', a, r') \in \delta'\}$
end for
if $R' \cap \mathcal{F}' = \emptyset$ **then**
  return true
else
  return false
end if

It is easy to see that a linear amount of space is sufficient to carry on the computation (the above algorithm uses $\mathcal{O}(|S| + |S'|)$ space to store the sets of reachable states of M and N and the counters i and j ranging over $\{0, ..., 2^{|S|+|S'|}\}$ and $\{0, ..., |S|2^{|S'|}\}$). This shows that the equivalence problem for UPA is solvable, by exploiting non-determinism, in space linear in the size of the input automata.

The deterministic solution

We now describe our second solution to the equivalence problem for UPA. In order to solve such a problem by means of a deterministic algorithm, we introduce a suitable canonical form for UPA. We show that such a form is unique (up to isomorphisms) for equivalent UPA, thus reducing the equivalence problem to an isomorphism test between canonical UPA. The proposed solution runs in simple exponential time with respect to the size of the input UPA, thus outperforming classical deterministic algorithms for Büchi automata, which are based on the (explicit or implicit) contruction of the complement automaton.

As a preliminary step, we introduce an alternative way of representing UPA in normal form as finite-state automata over an extended alphabet. Recall that any UPA $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$ in normal form contains only finitely many disjoint final loops, say $C_1, ..., C_n$. Thus, we can define the extended alphabet

$$\Sigma_M = \Sigma \cup \bigcup_{i \in [n]} \big(\Sigma \times \{C_i\} \times \mathcal{D}om(C_i)\big),$$

which consists of symbols of Σ plus symbols of the form (a, C_i, s), with $a \in \Sigma$, C_i being a final loop of M, and s being a final state of C_i. Then, we define the *prefix automaton* of M as the finite-state automaton $M_{pre} = ((S \setminus \mathcal{F}) \cup \{f\}, \Sigma_M, \delta', \{s_0\}, \{f\})$, where

- f is a new state, not belonging to S,
- for every $s, s' \in S \setminus \mathcal{F}$ and for every $a \in \Sigma$, $(s, a, s') \in \delta'$ iff $(s, a, s') \in \delta$,
- for every $s \in S \setminus \mathcal{F}$, every $i \in [n]$, every $s' \in \mathcal{D}om(C_i)$, and every $a \in \Sigma$, $(s, (a, C_i, s'), f) \in \delta'$ iff $(s, a, s') \in \delta$.

Intuitively, the prefix automaton M_{pre} is obtained by viewing the set of the final states of M as a single (final) macro-state and by using new input symbols to remember which final loop and which final state are reached by following a transition of M. Notice that M_{pre} recognizes the regular language

$$L = \big\{u \cdot (a, C_i, s) : s \in C_i \ \wedge\ s_0 \rightarrow^{u \cdot a} s\big\},$$

which is called the *prefix language* of M. Moreover, M_{pre} uniquely determines M, precisely, M can be obtained from M_{pre} by (i) adding the final loops $C_1, ..., C_n$ which appear in the extended alphabet Σ_M of M_{pre}, (ii) replacing each transition of the form $(s, (a, C_i, s'), f)$ with the transition (s, a, s'), and (iii) removing the state f. Hence M_{pre} is nothing but an alternative representation of M. Now, we can define the canonical form of an UPA.

Definition 2.7.16 *An UPA $M = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ is in* canonical form *if the following conditions are satisfied:*

- *M is in normal form,*
- *every final loop C of M has the minimum number of states,*
- *M has the minimum number of final loops (among equivalent UPA in normal form),*
- *it never happens that $(s, a, s'), (r', a, s') \in \delta$, with $s, s' \in \mathcal{F}$ and $r' \in S \setminus \mathcal{F} \setminus \{s_0\}$,*
- *the prefix automaton of M is a deterministic finite-state automaton with the minimum number of states (among equivalent finite-state automata).*

In order to prove that the canonical form is unique for all equivalent UPA, we give some preliminary results.

Lemma 2.7.17 *The final loops in a given UPA M in normal form are two-by-two non-isomorphic.*

Proof. This follows trivially from the third condition of Definition 2.7.16. ■

Lemma 2.7.18 *For every pair of equivalent UPA M, M' in canonical form, M and M' have isomorphic final loops.*

Proof. Let C be a final loop of M. Since M has the minimum number of final loops, there exists a word $w \in \mathscr{L}^\omega(M)$ that features all and only the repeating patterns encoded by C (recall that a final loop is reachable from the initial state). Since M' is equivalent to M, $w \in \mathscr{L}^\omega(M')$ and hence there is a final loop C' in M' that encodes all and only the repeating patterns featured by w. Since both C and C' have the minimum number of states, it immediately follows that C and C' are isomorphic. ■

In virtue of Lemma 2.7.17 and Lemma 2.7.18, given two equivalent UPA M, M' in canonical form, we can identify the symbols of the input alphabet Σ_M of the prefix automaton M_{pre} with the symbols of the input alphabet $\Sigma_{M'}$ of the prefix automaton M'_{pre}. Precisely, we say that two symbols $(a, C, s) \in \Sigma_M$ and $(a', C', s') \in \Sigma'_M$ *coincide* iff $a = a'$, C is a final loop of M isomorphic to the final loop C' of M', and s is a state of C that corresponds to s' in C' under the (unique) isomorphism between C and C'. We then extend such a correspondence to languages over Σ_M and $\Sigma_{M'}$.

Lemma 2.7.19 *For every UPA M in canonical form, M_{pre} accepts all and only the finite words of the form $u \cdot (a, C, s)$ for which there is $w \in \mathscr{L}^\omega(M)$ such that $u \cdot a$ is the shortest non-empty prefix of w.*

Proof. Let $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$ and let $u \cdot (a, C, s')$ be a finite word accepted by M_{pre}. By definition, C is a final loop of M, s' is a final state s' belonging to C, and there is a run ρ of M that starts from the initial state s_0, reaches s', and reads the finite word $u \cdot a$. Let v be a repeating pattern encoded by C, starting from the state s', namely, such that $s' \rightarrow^v s'$. Clearly, the ultimately periodic word $w = u \cdot v^\omega$ is accepted by M. We show that $u \cdot a$ is the shortest non-empty prefix of w. Let assume, by contradiction, that $|u| > 0$ and $v[|v|] = a$. Further let r' be the non-final state of M that precedes s' in ρ (such a state exists since $|u| > 0$) and let s be the final state of M that precedes s' in C. Clearly, $r' \neq s_0$ (because s_0 has no entering transitions) and $(r', a, s'), (s, v[|v|], s') \in \delta$, which contradicts the fourth condition of Definition 2.7.16. This shows that $u \cdot a$ is the shortest non-empty prefix of w. As for the converse implication, let w be an ultimately periodic word accepted by M and let ρ be a successful run of M on w. We denote by s' the first final state of M that occurs in ρ and by i its position in ρ. Clearly, $\rho[1, i-1] \cdot f$ is a successful run of M_{pre} on the finite word $u \cdot (a, C, s')$, where $u = w[1, i-2]$, $a = w[i-1]$, C is the (unique) final loop of M that contains s', and f is the (unique) final state of M_{pre}. Since for every final state $s \in \mathcal{F}$, $(s, b, s') \in \delta$ implies $b \neq a$, we know that $u \cdot a$ is the shortest non-empty prefix of w. ■

The following theorem implies that the equivalence problem for UPA is reducible to the equivalence problem for non-deterministic finite-state automata.

Theorem 2.7.20 *Given two UPA M and M' in canonical form, $\mathscr{L}^\omega(M) = \mathscr{L}^\omega(M')$ iff the languages of the corresponding prefix automata M_{pre} and M'_{pre} coincide.*

Proof. This follows trivially from Lemma 2.7.19, since every ultimately periodic word has a unique shortest non-empty prefix and hence $\mathscr{L}^\omega(M)$ is uniquely determined by $\mathscr{L}(M_{pre})$, and vice versa. ■

Corollary 2.7.21 *Given two UPA M and M' in canonical form, M and M' are equivalent iff they are isomorphic.*

Proof. This follows from Theorem 2.7.20 since the two prefix automata M_{pre} and M'_{pre} are deterministic and have the minimum number of states (hence they are isomorphic by results from [64]). ■

In the following, we show how to efficiently compute the canonical form of a given UPA. In particular, we show that the canonical form is the result of the application of following transformations, in the specified order:

i) **Compute the normal form**. Such an operation computes an UPA which is in normal form and equivalent to the input UPA.
ii) **Minimize the size of each final loop**. Such an operation collapses all bisimilar states in each final loop.
iii) **Minimize the number of final loops**. Such an operation collapses all isomorphic (minimal) final loops.
iv) **Remove redundant transitions**. Such an operation removes redundant transitions by adding short-cuts towards final loops.

v) **Determinize and minimize the prefix automaton.** According to this operation UPA are represented by their corresponding prefix automata. The resulting transformation (which now requires exponential time and space in the worst case) computes the minimum deterministic finite-state automaton equivalent to the prefix automaton of the input UPA.

As for the first step, recall that the normal form of any given UPA can be computed by using the procedure *UPANormalForm*, described in the previous section.

As for the minimization of the size and the number of final loops, notice that, given an UPA $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$ in normal form, the subgraph of M induced by $\mathcal{F}$ is a graph satisfying the following properties: (i) there exists at least one loop (provided that $\mathcal{F} \neq \emptyset$), (ii) there are no paths outside loops, (iii) no path can leave a loop. Thus, the two transformations can be viewed as a particular case of a single function coarsest partition problem, which can be solved in linear time by using the algorithm described by Paige, Tarjan and Bonic in [94]. We briefly present the algorithms that implement the two transformations and we refer to [94] for further details and proofs. Given a partition $P = \{S_1, ..., S_n\}$ of S into sets of bisimilar states, we denote by $Collapse(M, P)$ the UPA in normal form $(P, \Sigma, \delta', \{S_0\}, \mathcal{F}')$, where $(S_i, a, S_j) \in \delta'$ iff there are $s \in S_i$ and $s' \in S_j$ such that $(s, a, s') \in \delta$, S_0 is the (unique) set that contains the initial state s_0 of M, and $\mathcal{F}' = \{S_i : i \in [n], S_i \cap \mathcal{F} \neq \emptyset\}$.

The minimization of the size of the final loops of M can be achieved by collapsing bisimilar states in each final loop. Let $C_1, ..., C_l$ be the final loops of M. We represent each final loop C_i with a structure of the form

$$\big(\{s_{i,j} : j \in [n_i]\}, \{(s_{i,j}, a_{i,j}, s_{i,j \bmod n_i+1}) : j \in [n_i]\}\big),$$

where n_i is the number of states of C_i. We then denote by v_i the repeating pattern $a_{i,1}...a_{i,n_i}$ encoded by C_i and we define p_i as the first non-trivial occurrence of v_i in $v_i \cdot v_i$, namely, the least positive integer p_i such that $v_i = (v_i \cdot v_i)[p_i + 1, p_i + n_i]$. Note that such an integer p_i can be computed in time linear in n_i by using a string-matching algorithm [70]. Moreover, it is easy to prove that p_i divides $n_i = |v_i|$ and $v_i[1, p_i]$ is a primitive repeating pattern encoded by C_i. Hence, the states $s_{i,j}$ and $s_{i,j'}$ of C_i are bisimilar iff $((j-1) \bmod p_i)+1 = ((j'-1) \bmod p_i)+1$. This shows that, if we let $S_i = \{s_{i,j+kp_i} : 0 \leq k < \frac{n_i}{p_i}\}$, for every $i \in [l]$, and $P = \big\{\{s\} : s \in S \setminus \mathcal{F}\big\} \cup \big\{S_i : i \in [l]\big\}$, then $M' = Collapse(M, P)$ is an UPA in normal form equivalent to M and such that each final loop has the minimum number of states. We thus obtained an algorithm that minimizes the size of each final loop of M in time linear in the size of the input (i.e., the number of states and transitions of M). In order to simplify the description of such an algorithm, we use an auxiliary data structure P, which represents a partition of the set of states of M and which supports the following operations (see [33] for details):

- $Union(P, S, S')$, which substitutes two given sets S and S' in P (identified by suitable representatives) with their union $S \cup S'$,
- $FindSet(P, s)$, which returns (a representative of) the unique set $S \in P$ that contains the input element s.

UPAMinimizeLoops(M)

let $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$
$\{C_1, ..., C_l\} \leftarrow$ the set of all final loops of M
$P \leftarrow \big\{\{s\} : s \in S\big\}$
for $i = 1...l$ **do**
  let $C_i = \big(\{s_{i,j} : j \in [n_i]\}, \{(s_{i,j}, a_{i,j}, s_{i,j \bmod n_i+1}) : j \in [n_i]\}\big)$
  $v_i \leftarrow a_{i,1}...a_{i,n_i}$
  $p_i \leftarrow$ first non-trivial occurrence of v_i in $v_i \cdot v_i$
  for $j = 1...n_i$ **do**
    Union$(P, \mathit{FindSet}(P, s_{i,j}), \mathit{FindSet}(P, s_{i,((j-1) \bmod p_i)+1}))$
  end for
end for
$M' \leftarrow \mathit{Collapse}(M, P)$
return M'

Proposition 2.7.22 *Given an UPA M in normal form, the automaton M' resulting from UPAMinimizeLoops(M) is an UPA in normal form which is equivalent to M and satisfies the second condition of Definition 2.7.16.*

Proof. Since the algorithm *UPAMinimizeLoops* collapses only bisimilar states of M, M' is an UPA equivalent to M. Finally, from results of [94], the partition P computed by the algorithm is the coarsest partition of final states of M that respects the transitions of M. Hence, each final loop of M' has the minimum number of states and the second condition of Definition 2.7.16 is satisfied. Moreover, it is easy to see that the automaton M' is an UPA in normal form. ■

As for the minimization of the number of final loops, it turns out that, if M is an UPA satisfying the first and the second condition of Definition 2.7.16 and $C_1, ..., C_l$ are all and only the final loops of M, then for every $i, j \in [l]$, the final loops C_i and $C_{i'}$ are bisimilar iff they are isomorphic (this follows from the fact that each final loop of M has the minimum number of states). Moreover, isomorphic final loops can be efficiently found by (i) defining a total ordering on the alphabet Σ, (ii) representing each final loop C_i with the *lexicographically least primitive repeating pattern* v'_i encoded by C_i, and (iii) sorting the loops according to the lexicographic order of their representatives (in this way, isomorphic final loops have the same representative and hence they are contiguous in the ordered list). Notice that for every $i \in [l]$, the lexicographically least primitive repeating pattern v'_i of C_i can be computed in linear time by using the algorithms described in [11, 102]. Below, we report the algorithm that minimizes the number of final loops of M in linear time with respect to the size of the input. It uses the following auxiliary data structures and procedures:

- a list *Loops*, which stores the final loops C_i of M together with their representatives v'_i,
- a procedure *Sort*, which sorts a given list of final loops according to the lexicographic order of their representatives (this can be done in linear time),

- a partition P of the states of M, which supports the operations *Union* and *FindSet*.

UPAMinimizeNumberOfLoops(M)

1: **let** $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$
2: $\{C_1, ..., C_l\} \leftarrow$ the set of all final loops of M
3: $Loops \leftarrow \varepsilon$
4: **for** $i = 1...l$ **do**
5: **let** $C_i = \big(\{s_{i,j} : j \in [n_i]\}, \{(s_{i,j}, a_{i,j}, s_{i,j \bmod n_i+1}) : j \in [n_i]\}\big)$
6: $v_i \leftarrow a_{i,1}...a_{i,n_i}$
7: $v_i' \leftarrow$ lexicographically least rotation of v_i
8: $Loops \leftarrow Loops \cdot (C_i, v_i')$
9: **end for**
10: $Sort(Loops)$
11: $P \leftarrow \big\{\{s\} : s \in S\big\}$
12: **for** $i = 1...|Loops| - 1$ **do**
13: $(C, v) \leftarrow Loops[i]$
14: $(C', v') \leftarrow Loops[i + 1]$
15: **let** $C = \big(\{s_j : j \in [m]\}, \{(s_j, v[j], s_{j \bmod m+1}) : j \in [m]\}\big)$
16: **let** $C' = \big(\{s'_j : j \in [m']\}, \{(s'_j, v'[j], s'_{j \bmod m'+1}) : j \in [m']\}\big)$
17: **if** $v = v'$ **then**
18: **for** $j = 1...m$ **do**
19: $Union(P, FindSet(P, s_j), FindSet(P, s'_j))$
20: **end for**
21: **end if**
22: **end for**
23: $M' \leftarrow Collapse(M, P)$
24: **return** M'

Proposition 2.7.23 *Given an UPA M satisfying the first and the second condition of Definition 2.7.16, the automaton M' resulting from UPAMinimizeNumberOfLoops(M) is an UPA which is equivalent to M and satisfies the first, the second, and the third condition of Definition 2.7.16.*

Proof. It is routine to verify that the automaton M' is an UPA in normal form, equivalent to M, and satisfying the second condition of Definition 2.7.16. Moreover, since M' is obtained from M by collapsing isomorphic final loops and since non-isomorphic *minimal* final loops encode non-equivalent repeating patterns, we have that for every ultimately periodic word $w \in \mathscr{L}^\omega(M')$ and for every repeating pattern v featured by w, there is exactly one final loop of M' that encodes v. This shows that M' contains the minimum number of final loops and hence the third condition of Definition 2.7.16 is satisfied as well. ∎

The fourth operation is implemented in two phases on a given an UPA $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$ in normal form. The first phase consists in selecting a pair of transitions of the form (s, a, s') and (r', a, s'), with $s, s' \in \mathcal{F}$ and $r' \in S \setminus \mathcal{F} \setminus \{s_0\}$,

marking (r', a, s') (in such a way that it cannot be selected again), and adding a transition (r, b, s) for each existing transition (r, b, r'). When no more transitions can be selected, the second phase starts, which consists in removing previously marked transitions. Such a process can be straightforwardly implemented by the following algorithm, which takes polynomial time in the size of the input automaton.

UPAShortcuts(M)

1: **let** $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$
2: $\delta' \leftarrow \delta$
3: **for all** $(s, a, s') \in \delta'$ **do**
4: $\quad mark[(s, a, s')] \leftarrow$ **false**
5: **end for**
6: *newSelections* $\leftarrow$ **true**
7: **while** *newSelections* **do**
8: $\quad$ *newSelections* $\leftarrow$ **false**
9: $\quad$ **for all** $(s, a, s') \in \delta'$ **do**
10: $\quad\quad$ **for all** $(r', a, s') \in \delta'$ **do**
11: $\quad\quad\quad$ **if** $s, s' \in \mathcal{F}$ **and** $r' \in S \setminus \mathcal{F} \setminus \{s_0\}$ **and not** $mark[(r', a, s')]$ **then**
12: $\quad\quad\quad\quad mark[(r', a, s')] \leftarrow$ **true**
13: $\quad\quad\quad\quad$ *newSelections* $\leftarrow$ **true**
14: $\quad\quad\quad\quad$ **for all** $(r, b, r') \in \delta'$ **do**
15: $\quad\quad\quad\quad\quad$ **if** $r \in S \setminus \mathcal{F}$ **then**
16: $\quad\quad\quad\quad\quad\quad \delta' \leftarrow \delta' \cup \{(r, b, s)\}$
17: $\quad\quad\quad\quad\quad$ **end if**
18: $\quad\quad\quad\quad$ **end for**
19: $\quad\quad\quad$ **end if**
20: $\quad\quad$ **end for**
21: $\quad$ **end for**
22: **end while**
23: **for all** $(r, a, s) \in \delta'$ **do**
24: $\quad$ **if** $mark[(r, a, s)]$ **then**
25: $\quad\quad \delta' \leftarrow \delta' \setminus \{(r, a, s)\}$
26: $\quad$ **end if**
27: **end for**
28: $M' \leftarrow (S, \Sigma, \delta', \{s_0\}, \mathcal{F})$
29: **return** M'

Proposition 2.7.24 *Given an UPA M in normal form, the automaton M' resulting from UPAShortcuts(M) is an UPA in normal form, which is equivalent to M and satisfies the fourth condition of Definition 2.7.16. Moreover, if M satisfies the second (resp., the third) condition of Definition 2.7.16, then M' satisfies the second (resp., the third) condition of Definition 2.7.16 as well.*

Proof. Let $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$. The automaton M' is clearly an UPA in normal form. Let assume, by contradiction, that M' contains a pair of transitions of the form

(r', a, s') and (s, a, s'), with $s, s' \in \mathcal{F}$ and $r' \in S \setminus \mathcal{F} \setminus \{s_0\}$. Notice that, since the algorithm only adds/removes transitions departing from a non-final state and reaching a final state, the transition (s, a, s') already belonged to the original automaton M. Hence, two cases arise: either the transition (r', a, s') belonged to M or it was added during some iteration of the 'while' loop at lines 7–22. In the former case, the value of $mark[(r', a, s')]$ would be set to true during the first iteration of the 'while' loop and thus the transition (r', a, s') would have been removed later when line 25 is executed. In the latter case, during the iteration that adds the new transition (r', a, s') to δ', the value of *newSelections* is set to true. Hence, either during that iteration or during the next one (which is taken since *newSelections* is set to true), the transition (r', a, s') must be selected and then marked. This shows that δ' contains no pairs of transitions of the form (r', a, s') and (s, a, s'), with $s, s' \in \mathcal{F}$ and $r' \in S \setminus \mathcal{F} \setminus \{s_0\}$. Therefore, M' satisfies the fourth condition of Definition 2.7.16.
Moreover, the output automaton M' is equivalent to M, namely, $\mathscr{L}^\omega(M) \subseteq \mathscr{L}^\omega(M')$ and $\mathscr{L}^\omega(M') \subseteq \mathscr{L}^\omega(M)$. In order to show the first inclusion, we prove that the following property is an invariant before and after the execution of lines 12–18: for every successful run of M on an infinite word w, there is a run on w that starts from s_0 and uses only unmarked transitions of δ'. We prove this by induction on the number of iterations of lines 12–18. At the beginning, the property holds trivially. Now, let assume that the property holds before a new transition (r', a, s') is going to be marked (line 12) and let ρ be a successful run of M on w and ρ' be the corresponding run on w that starts from s_0 and uses only unmarked transitions from δ'. If ρ' does not use the transition (r', a, s'), then the property clearly holds after the execution of lines 12–18. Otherwise, if ρ' uses the transition (r', a, s'), then it must use (r', a, s') only once (because after the final state s' has been reached, only states from $\mathcal{F}$ can occur). Let (r, b, r') be the transition that immediately precedes (r', a, s') in ρ' and let (s, a, s') be the (unique) transition from δ' such that $s \in \mathcal{F}$. We can substitute the two transitions (r, b, r') and (r', a, s') in ρ' with the new transitions (r, b, s) and (s, a, s'), thus obtaining a new run ρ'' on w that starts from s_0, uses only the transition (r, b, s) and unmarked transitions from δ', and, moreover, it does not use the transition (r', a, s'). Therefore, after the execution of lines 12–18, ρ'' is satisfies the property. This proves that $\mathscr{L}^\omega(M) \subseteq \mathscr{L}^\omega(M')$. The converse inclusion is easily verified since the first part of the algorithm (lines 7–22) adds only redundant transitions.
Finally, since the algorithm modifies only transitions departing from non-final states, the output automaton is guaranteed to satisfy the second (resp., the third) condition of Definition 2.7.16, whenever the input automaton satisfies the second (resp., the third) condition as well. ■

Given an UPA M in normal form, the last transformation computes the minimum deterministic finite-state automaton M'_{pre} equivalent to the prefix automaton M_{pre} of M. It then uses such an automaton M'_{pre} as a representation for an UPA M'. It is worth remarking that, since the problem of determinizing finite-state automata is PSPACE-complete (see [64]), this last transformation is the most demanding one from the computational point of view. From experimental comparisons (see, for example, [17]), it turns out that Brzozowski's algorithm [14] is, in general, the most efficient

solution to the problem of determinizing and minimizing finite-state automata. Such an algorithm works as follows. Given a non-deterministic finite-state automaton N, it first reverses the transitions of N (we denote such an operation by Rev), then it performs a subset construction to build a deterministic finite-state automaton equivalent to the reversed copy of N (we denote such an operation by Det), and finally it iterates such two operations one more time. It can be proved that $Det(Rev(Det(Rev(N))))$ is a deterministic finite-state automaton equivalent to N and with the minimum number of states. Moreover, such a construction requires, in the worst case, $\mathcal{O}(2^n)$ time and space, where n is the size of the input automaton N. It is also clear that, if $N = M_{pre}$ is the prefix automaton of a given UPA M in normal form, then the construction can be performed directly on M (by considering only the non-final states of M), in such a way that the resulting automaton is an UPA M' equivalent to M and satisfying the last condition of Definition 2.7.16.

Proposition 2.7.25 *Given an UPA M in normal form satisfying the second (resp., third, fourth) condition of Definition 2.7.16, the determinization and minimization of the corresponding prefix automaton M_{pre} results in an UPA M' in normal form, which is equivalent to M and satisfies the second (resp., third, fourth) condition of Definition 2.7.16.*

Proof. We prove the most interesting case only, namely, that M' is an UPA satisfying the fourth condition of Definition 2.7.16, provided that M satisfies the fourth condition as well. Let assume, by contradiction, that M' contains two transitions $(\bar{s}, a, \bar{s}')$ and $(\bar{r}', a, \bar{s}')$, with $\bar{s}, \bar{s}' \in \bar{\mathcal{F}}$ and $\bar{r}' \in S \setminus \bar{\mathcal{F}} \setminus \{\bar{s}_0\}$ (here $\bar{s}_0$ denotes the initial state of M' and $\bar{\mathcal{F}}$ denotes the set of final states of M'). Then, it is easy to see the prefix automaton M'_{pre} of M' contains a transition of the form $(\bar{r}', (a, C, \bar{s}'), \bar{f})$, with C being the final loop of M' containing $\bar{s}$ and $\bar{s}'$, $\bar{r}'$ being a non-final state of M'_{pre}, and $\bar{f}$ being the unique final state of M'_{pre}. Since M'_{pre} is equivalent to the prefix automaton M_{pre} of M, M_{pre} contains a transition of the form $(r', (a, C, s'), f)$, where r' is a non-final state of M_{pre} and f is the unique final state of M_{pre}. Since C is also a final loop of M, this implies that M contains two transitions of the form (s, a, s') and (r', a, s'), with $s, s' \in \mathcal{F}$ and $r' \in S \setminus \mathcal{F} \setminus \{s_0\}$ (here s_0 denotes the initial state of M and $\mathcal{F}$ denotes the set of final states of M). This contradicts the hypothesis that M is an UPA satisfying the fourth condition of Definition 2.7.16. ■

For the sake of clearness, we report the complete algorithm that computes the canonical form of a given UPA.

UPACanonicalForm(M)

1: $M_1 \leftarrow UPANormalForm(M)$
2: $M_2 \leftarrow UPAMinimizeLoops(M_1)$
3: $M_3 \leftarrow UPAMinimizeNumberOfLoops(M_2)$
4: $M_4 \leftarrow UPAShortcuts(M_3)$
5: $M_5 \leftarrow Det(Rev(Det(Rev(M_4))))$
6: **return** M_5

2.7.4 The size-optimization problem

In this section we show how to compute compact automaton-based representations of sets of time granularities. First, we describe a size-optimization procedure for UPA in normal form and then we extend the result to generic UPA. In the former case, we are able to compute an UPA which is size-optimal among the class of all UPA in normal form. The latter case is more difficult and we are only able to approximate size-optimal UPA with respect to the class of generic UPA. We start by providing sufficient conditions for an UPA being size-optimal and in normal form.

Proposition 2.7.26 *An UPA in normal form $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$ has the minimum number of states (among equivalent UPA in normal form) if the following conditions are satisfied:*

- *every final loop C of M has the minimum number of states,*
- *M has the minimum number of final loops (among equivalent UPA in normal form),*
- *it never happens that $(s, a, s'), (r', a, s') \in \delta$, with $s, s' \in \mathcal{F}$ and $r' \in S \setminus \mathcal{F} \setminus \{s_0\}$,*
- *the prefix automaton of M is a (non-deterministic) finite-state automaton with the minimum number of states (among equivalent finite-state automata).*

Proof. Let M be an UPA satisfying the above conditions and consider the prefix automaton M_{pre}. Note that Theorem 2.7.20 holds even if the automaton M satisfies only the first four conditions of Definition 2.7.16. Hence, by Theorem 2.7.20, since M_{pre} is a size-optimal finite-state automaton and since M has the minimum number of final states, M is a size-optimal UPA in normal form. ∎

By exploiting Proposition 2.7.26, we can devise an algorithm that computes size-optimal UPA in normal form. Such an algorithm is based on an auxiliary procedure *SizeOptimalNFA*, which minimizes the number of states of a given non-deterministic finite-state automaton (see [66, 74] for implementation details). It is worth mentioning that the size-optimization of non-deterministic finite-state automata is a PSPACE-complete problem (see [65, 66, 74]). Since such a problem is reducible (by using an argument similar to that of Proposition 2.7.15) to the problem of minimizing the number of states of UPA in normal form, the latter problem turns out to be PSPACE-complete as well.

SizeOptimaUPANormalForm(M)
1: $M_1 \leftarrow$ *UPANormalForm*(M)
2: $M_2 \leftarrow$ *UPAMinimizeLoops*(M_1)
3: $M_3 \leftarrow$ *UPAMinimizeNumberOfLoops*(M_2)
4: $M_4 \leftarrow$ *UPAShortcuts*(M_3)
5: $M_5 \leftarrow$ *SizeOptimalNFA*(M_4)
6: **return** M_5

We now show how to further reduce the number of states of a given size-optimal UPA in normal form. Given an UPA $M = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$, a final loop C, and a simple non-final loop C' of M (reachable from some initial state), we say that C is *redundant*

with respect to C' if C has no exiting transitions and there exists a surjective function $f : \mathcal{D}om(C') \rightarrow \mathcal{D}om(C)$ satisfying the following conditions:

- for every $s, s' \in \mathcal{D}om(C')$, $(s, a, s') \in \delta$ iff $(f(s), a, f(s')) \in \delta$,
- for every $s \in S \setminus \mathcal{D}om(C) \setminus \mathcal{D}om(C')$ and for every $s' \in \mathcal{D}om(C)$, $(s, a, s') \in \delta$ iff there is $s'' \in \mathcal{D}om(C')$ such that $s' = f(s'')$ and $(s, a, s'') \in \delta$.

In such a case, we can remove the final loop C and let the states in C' be final states. Below, we prove that the resulting automaton is an UPA equivalent to M. As an example, in Figure 2.21 we depicted two UPA: the right-hand side automaton is obtained by removing from the left-side automaton the loop on state s_4, which is redundant with respect to the non-final loop on states s_1, s_2.

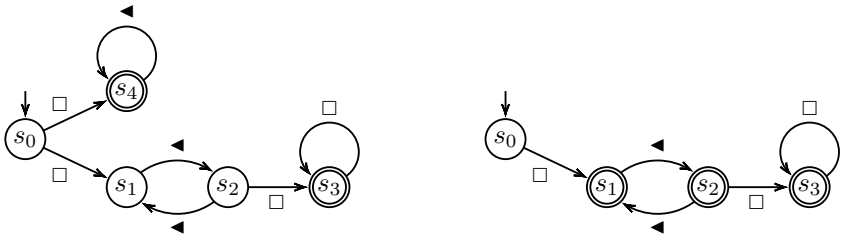

Figure 2.21: An example of redundant final loop.

Proposition 2.7.27 *Given an UPA $M = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$, a final loop C of M, and a simple non-final loop C' of M reachable from some initial state, if C is redundant with respect to C', then the automaton $M' = (S \setminus \mathcal{D}om(C), \Sigma, \delta, \mathcal{I}, (\mathcal{F} \setminus \mathcal{D}om(C)) \cup \mathcal{D}om(C'))$ is an UPA equivalent to M.*

Proof. Clearly M' is an UPA, because for every final state s of M', either $s \in \mathcal{F} \setminus \mathcal{D}om(C)$ (which implies that the strongly connected component of s in M' is a single transient state or a simple final loop) or $s \in \mathcal{D}om(C')$ (which implies that the strongly connected component of s in M' is a simple loop). We now show that M' is equivalent to M. We denote by f a surjective function from $\mathcal{D}om(C')$ to $\mathcal{D}om(C)$ such that

- for every $s, s' \in \mathcal{D}om(C')$, $(s, a, s') \in \delta$ iff $(f(s), a, f(s')) \in \delta$,
- for every $s \in S \setminus \mathcal{D}om(C) \setminus \mathcal{D}om(C')$ and for every $s' \in \mathcal{D}om(C)$, $(s, a, s') \in \delta$ iff there is $s'' \in \mathcal{D}om(C')$ such that $s' = f(s'')$ and $(s, a, s'') \in \delta$.

Let consider an ultimately periodic word $w \in \mathscr{L}^{\omega}(M)$ and let ρ be a successful run of M on w. Since C has no exiting transitions, either ρ contains no state of C or all, but finitely many, states of ρ belong to $\mathcal{D}om(C)$. In the former case, w is clearly accepted by M'. In the latter case, we can obtain a successful run ρ' of M' on w by simply replacing each state $s \in \mathcal{D}om(C)$ with a suitable state s' such that $f(s') = s$ (from the hypothesis on the function f, ρ' respects the transitions of M' on w).
As for the converse inclusion, let w be an ultimately periodic word accepted by M' and let ρ' be a successful run of M' on w. By definition of UPA, the set of states that occur infinitely often in ρ' is either disjoint from $\mathcal{D}om(C')$ or it coincides with

$\mathcal{D}om(C')$. In the former case, ρ' is clearly a successful run of M and w is accepted by M. In the latter case, we can obtain a successful run ρ of M on w by simply replacing each state $s \in \mathcal{D}om(C')$ with the state $f(s)$. ■

By exploiting Proposition 2.7.27, we can devise a polynomial-time algorithm that removes all redundant loops (in any arbitrary order) from a given size-optimal UPA M in normal form, hence obtaining an UPA M' which is in general smaller than M and approximates a size-optimal UPA. It is easy to see that the automaton M' resulting from such an algorithm has always the same size, independently from the removal order of the redundant final loops. However, this does not mean that M' is a size-optimal UPA whenever M is a size-optimal UPA in normal form. As an example, consider the two UPA depicted in Figure 2.22: the left-side automaton has no redundant loops, while in the right-side automaton the loop on s'_5 is redundant with respect to the loop on s'_2. This means that the optimization algorithm produces a smaller UPA when the input is the right-side automaton.

CompactUPA(M)

1: **let** $M = (S, \Sigma, \delta, \{s_0\}, \mathcal{F})$
2: $\{C_1, ..., C_l\} \leftarrow$ the set of all final loops of M
3: $\{C'_1, ..., C'_m\} \leftarrow$ the set of all simple non-final loops of M reachable from $\{s_0\}$
4: $S' \leftarrow S$
5: $\mathcal{F}' \leftarrow \mathcal{F}$
6: **for** $i = 1...l$ **do**
7: $\quad j' \leftarrow \perp$
8: $\quad$ **for** $j = 1...m$ **do**
9: $\quad\quad$ **if** C_i is redundant with respect to C'_j **then**
10: $\quad\quad\quad j' \leftarrow j$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad$ **if** $j' \neq \perp$ **then**
14: $\quad\quad S' \leftarrow S \setminus \mathcal{D}om(C_i)$
15: $\quad\quad \mathcal{F}' \leftarrow (\mathcal{F}' \setminus \mathcal{D}om(C_i)) \cup \mathcal{D}om(C'_{j'})$
16: $\quad$ **end if**
17: **end for**
18: $M' \leftarrow (S', \Sigma, \delta, \{s_0\}, \mathcal{F}')$
19: **return** M'

2.7.5 The granularity comparison problem

In this section we deal with the granularity comparison problem, which consists in checking whether a distinguished relation $\sim$ holds between two granularities $G \in \mathcal{G}$ and $H \in \mathcal{H}$, where $\mathcal{G}$ and $\mathcal{H}$ are given sets of granularities. As an example, we consider a particular relation between granularities, namely, the label-aligned refinement (cf. Section 2.2), and we characterize it in terms of string-based representations. Then, given two UPA representing the sets $\mathcal{G}$ and $\mathcal{H}$ of granularities, we define a suitable

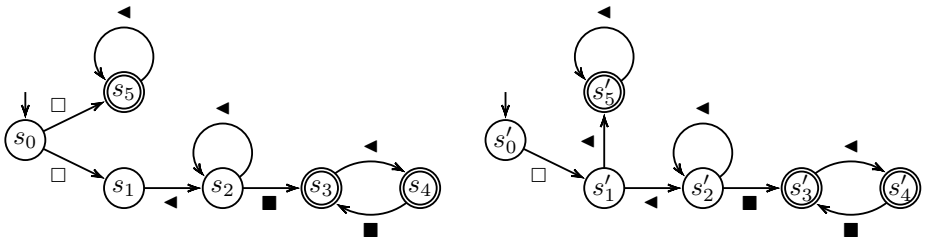

Figure 2.22: Two equivalent size-optimal UPA in normal form.

product automaton that recognizes the set of all pairs of granularities G and H such that G is a label-aligned refinement of H (the input alphabet of such a product automaton is the set $\{\blacksquare, \square, \blacktriangleleft\} \times \{\blacksquare, \square, \blacktriangleleft\}$). Finally, we reduce the granule comparison problem for the label-aligned refinement relation to the emptiness problem for the corresponding product automaton. Note that the method provided in this section can be easily adapted to different relations for time granularities (e.g., partition, group, sub-granularity, etc.).

Recall that a granularity G *is a label-aligned refinement* of a granularity H, if for every $x \in \mathbb{N}$, $G(x) \subseteq H(x)$. The following proposition gives a characterization of the string-based representations of pairs of granularities that satisfy such a relation.

Proposition 2.7.28 *Given two infinite words u and v representing, respectively, G and H in terms of a bottom granularity, G is a label-aligned refinement of H iff for every $i \in \mathbb{N}$, $|u[1,i]|_{\blacktriangleleft} = |v[1,i]|_{\blacktriangleleft}$ and $u[i+1] \in \{\blacksquare, \blacktriangleleft\}$ implies $v[i+1] \in \{\blacksquare, \blacktriangleleft\}$.*

Proof. Let assume that G is a label-aligned refinement of H. For every $x \in \mathbb{N}$, $G(x) \subseteq H(x)$. Since H covers G, for every time point $i \in \mathbb{N}$, $u[i+1] \in \{\blacksquare, \blacktriangleleft\}$ implies that i is covered by G, hence i is covered by H, hence $v[i+1] \in \{\blacksquare, \blacktriangleleft\}$. Moreover, if $i \in G(x)$, then $i \in H(x)$, and hence $|u[1,i]|_{\blacktriangleleft} = x = |v[1,i]|_{\blacktriangleleft}$. The converse implication follows by a symmetric argument. ■

By exploiting Proposition 2.7.28 and closure properties of UPA with respect to (non-synchronized) products and intersections with Büchi-recognizable languages, we can build a suitable product automaton for the relation 'label-aligned refinement'. We first define a Büchi automaton R representing the set of all pairs of time granularities (including non-periodical ones) for which the relation 'label-aligned refinement' holds. Such an automaton $R = (S, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ is defined as follows (see also Figure 2.23):

- $S = \{s_0, s_1, s_2\}$,
- $\Sigma = \{\blacksquare, \square, \blacktriangleleft\} \times \{\blacksquare, \square, \blacktriangleleft\}$,
- $\delta = \{s_0\} \times \{(\begin{smallmatrix}\blacksquare\\\blacksquare\end{smallmatrix}), (\begin{smallmatrix}\square\\\blacksquare\end{smallmatrix}), (\begin{smallmatrix}\blacktriangleleft\\\blacksquare\end{smallmatrix}), (\begin{smallmatrix}\square\\\square\end{smallmatrix}), (\begin{smallmatrix}\blacktriangleleft\\\blacktriangleleft\end{smallmatrix})\} \times \{s_0\} \cup \{s_0\} \times \{(\begin{smallmatrix}\blacktriangleleft\\\blacksquare\end{smallmatrix})\} \times \{s_1\} \cup \{s_1\} \times \{(\begin{smallmatrix}\square\\\blacktriangleleft\end{smallmatrix})\} \times \{s_0\} \cup \{s_0\} \times \{(\begin{smallmatrix}\blacksquare\\\blacktriangleleft\end{smallmatrix})\} \times \{s_2\} \cup \{s_2\} \times \{(\begin{smallmatrix}\square\\\blacksquare\end{smallmatrix}), (\begin{smallmatrix}\square\\\square\end{smallmatrix}), (\begin{smallmatrix}\square\\\blacktriangleleft\end{smallmatrix})\} \times \{s_2\}$,
- $\mathcal{I} = \{s_0\}$,
- $\mathcal{F} = \{s_0, s_1, s_2\}$.

Then, given two UPA M and N, representing two sets of granularities $\mathcal{G}$ and $\mathcal{H}$, we compute the UPA $(M \times N) \cap R$, which clearly represents the set of all pairs of granularities (G, H), where $G \in \mathcal{G}$, $H \in \mathcal{H}$, and G is a label-aligned refinement of H. In this way, the granularity comparison problem for M and N is reduced to checking whether the ω-language recognized by $(M \times N) \cap R$ is non-empty.

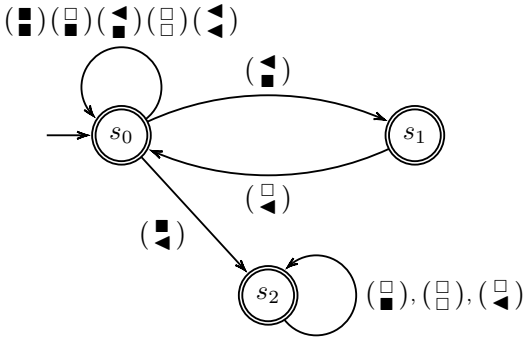

Figure 2.23: An automaton for the 'label-aligned refinement' relation.

We conclude the section by considering a real-world application of UPA-based representations of time granularities and granularity comparison problems. We focus our attention on the medical domain of heart transplant patients. 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 reported in [27]. 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. The ability of dealing with sets of granularities, and not only with single granularities, is thus needed to reason about protocols and patient schedules. As an example, 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). The decidability of this problem gives the possibility of choosing the most compact, or the most suitable, representation for a given protocol. Another meaningful reasoning task is that of checking whether a given therapy/granularity assigned to a patient satisfies the prescribed protocol, that is, whether it belongs to the set of therapies/granularities of the protocol (granularity comparison problem). We

focus on this latter problem. Consider the above given (sub)set of therapies/check-ups of a protocol for hearth transplant patients. Given an UPA M representing the single granularity of the specific therapy (up to a certain date) for some patient and given an UPA N encoding the set of (the granularities for) the prescribed therapies/check-ups, the granularity comparison problem can be decided by checking that the word contained in $\mathscr{L}^{\omega}(M)$ properly relates to some word in $\mathscr{L}^{\omega}(N)$. Thus, the given consistency-checking problem can be seen as a particular case of granularity comparison problem. As an example, consider the instance of the temporal relation `Visits`(`patientId`, `date`, `treatment`) represented in Table 2.3.

`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

Table 2.3: A hypothetical schedule for therapies/check-ups.

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, we can build the granularity of GFR measurements for the patient identified by 1001. We represent such a granularity as a single ultimately periodic word u (starting from 01/01/2003), in which the occurrences of ◀ denote the days of the visits. The UPA M recognizing u is depicted in Figure 2.24, where we use the shorthand $\circ \xrightarrow{a^n} \circ$ to denote a sequence of $n+1$ states and n a-labeled transitions. The UPA N encoding the granularities of the protocol is depicted in Figure 2.25.

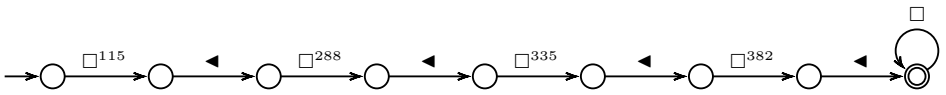

Figure 2.24: An UPA representing GFR measurements for a patient.

In order to check whether the granularity of GFR measurements for patient 1001 is a label-aligned refinement of some granularity in $\mathscr{L}^{\omega}(N)$, we build the 'product' automaton $(M \times N) \cap R$, where R is the Büchi automaton encoding the relation 'label-aligned refinement'. Such an automaton recognizes the ω-language

$$\left\{ \left(\begin{smallmatrix}\square\\ \square\end{smallmatrix}\right)^{100}\left(\begin{smallmatrix}\blacksquare\\ \square\end{smallmatrix}\right)^{15}\left(\begin{smallmatrix}\blacksquare\\ \blacktriangleleft\end{smallmatrix}\right)\left(\begin{smallmatrix}\blacksquare\\ \square\end{smallmatrix}\right)^{13}\left(\begin{smallmatrix}\blacktriangleleft\\ \square\end{smallmatrix}\right)\left(\begin{smallmatrix}\square\\ \square\end{smallmatrix}\right)^{245}\left(\begin{smallmatrix}\blacksquare\\ \square\end{smallmatrix}\right)^{29}\left(\begin{smallmatrix}\blacktriangleleft\\ \blacktriangleleft\end{smallmatrix}\right)\left(\begin{smallmatrix}\square\\ \square\end{smallmatrix}\right)^{335}\left(\begin{smallmatrix}\blacksquare\\ \blacktriangleleft\end{smallmatrix}\right)\left(\begin{smallmatrix}\blacksquare\\ \square\end{smallmatrix}\right)^{28}\left(\begin{smallmatrix}\blacktriangleleft\\ \square\end{smallmatrix}\right)\left(\begin{smallmatrix}\square\\ \square\end{smallmatrix}\right)^{335}\cdot \cdot\left(\begin{smallmatrix}\blacksquare\\ \square\end{smallmatrix}\right)^{18}\left(\begin{smallmatrix}\blacksquare\\ \blacktriangleleft\end{smallmatrix}\right)\left(\begin{smallmatrix}\blacksquare\\ \square\end{smallmatrix}\right)^{10}\left(\begin{smallmatrix}\blacktriangleleft\\ \square\end{smallmatrix}\right)\left(\left(\begin{smallmatrix}\square\\ \square\end{smallmatrix}\right)^{335}\left(\begin{smallmatrix}\blacksquare\\ \square\end{smallmatrix}\right)^{29}\left(\begin{smallmatrix}\blacktriangleleft\\ \square\end{smallmatrix}\right)\right)^{\omega} \right\}.$$

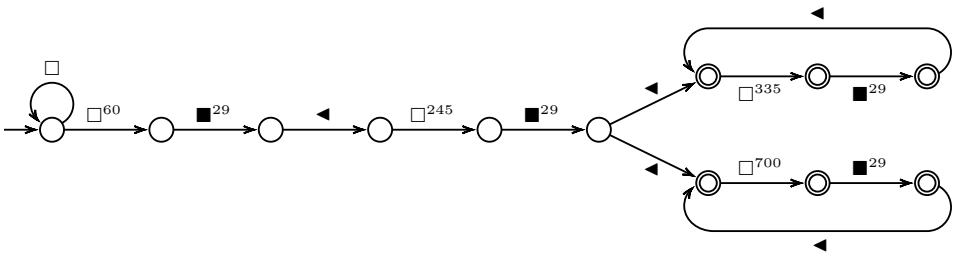

Figure 2.25: An UPA-based specification of the protocol.

Since the resulting language is non-empty, we can conclude that the therapy satisfies the prescribed protocol.

2.8 Discussion

In this chapter we considered the problems of representing and reasoning about time granularities by means of automaton-based formalisms.

In Section 2.2 we gave a formal definition of time granularity, which captures a meaningful class of temporal structures, and we introduced some standard relationships between time granularities.

In Section 2.3 we reviewed the algebraic formalism of Calendar Algebra, which allows one to produce compact and user-friendly representations of a large class of time granularities, including finite and infinite (periodical) ones.

In Section 2.4 we introduced Wijsen's string-based approach and we adapted it to our notion of time granularity (which differs from Wijsen's one in that it introduces labels of granules). We then described an efficient (linear-time) algorithm for solving the equivalence problem of string-based representations of time granularities and we compared the expressiveness of the string-based formalism with that of Calendar Algebra. As for the comparison of such two formalisms, we described some algorithms that map expressions of Calendar Algebra to equivalent string-based representations. Such a mapping, besides showing that the two formalisms are expressively equivalent, provides effective procedures to test the equivalence and to reason on granularity relations in the framework of Calendar Algebra (as a matter of fact, a strong limitation of the formalism of Calendar Algebra is that it focuses on expressiveness issues mainly, and almost completely ignores some basic problems of obvious theoretical and practical importance). In this perspective, the string-based approach (as well as the automaton-based one, which is intimately related to it) can be viewed as a 'low-level' framework into which 'high-level' and 'user-friendly' formalisms like Calendar Algebra can be mapped.

In Sections 2.5, 2.6, and 2.7 the automaton-based approach to time granularity is explored in full detail. We reviewed the notion of single-string (sequential) automaton, which was originally proposed by Dal Lago and Montanari, and we considered extensions of automata with counters, in order to make representations of time gran-

ularities more compact. We showed that single-string automata (possibly extended with counters) are as expressive as the formalism of Calendar Algebra and Wijsen's string-based models and we provided effective solutions to the problems of equivalence, granule conversion, and optimization of time granularity representations.

In Section 2.6 we introduced a new class of single-string automata extended with counters, which we called restricted labeled single-string automata (RLA for short). Suitable restrictions on the transition functions of such automata have been exploited to devise efficient algorithms for the granule conversion and the equivalence problems. Precisely, we gave deterministic polynomial-time algorithms that solve granule conversion problems and a non-deterministic polynomial-time algorithm that solves the equivalence problem. As for the solution to the equivalence problem, recall that it is based on a reduction to the satisfiability problem for linear diophantine equations with bounds on variables. One can easily see that, if $\bigcup_{i\in[m]}\left(k_{i,1}I_{i,1} + ... + k_{i,n_i}I_{i,n_i}\right)$ is a succinct linear progression representing the positions of some labeled state in the labeled run of an RLA, then for every $i \in [m]$ and for every pair of distinct indices $j, j' \in [n_i]$, either $k_{i,j}max(I_{i,j}) \leq k_{i,j'}$ or $k_{i,j'}max(I_{i,j'}) \leq k_{i,j}$ holds. We expect that such a particular structure in a succinct linear progression can be exploited to test, in deterministic polynomial time, the satisfiability of the linear diophantine equations resulting from the reduction of an instance of the equivalence problem. We also considered optimization problems for RLA-based representations and we identified two possible ways of optimizing such representations. According to the first one, optimizing means computing the smallest (i.e., size-optimal) representation of a given time granularity; according to the second one, optimizing means computing the most tractable (i.e., complexity-optimal) representation of a given granularity, that is, the one on which crucial algorithms (e.g., conversion algorithms) run fastest. These two criteria have been showed to be not equivalent and to yield non-unique solutions. We provided polynomial-time algorithms that receive in input a string-based representation of a time granularity and compute an equivalent complexity-/size-optimal automaton-based representation of it. We believe it possible to further improve optimization algorithms by exploiting subtle relationships between the positions of the repeating patterns in a string and the structure of the transitions functions of such automata. As a matter of fact, we conjecture that the loops determined by the secondary transition function of a complexity-optimal RLA can be related to the *maximal repetitions* in the recognized word (a maximal repetition of a word 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 still exploring the possibility of generalizing the size-optimization algorithm in order to produce size-optimal automata with respect to the whole class of RLA, rather than the subclass $\mathcal{R}_\Sigma$ only. We believe that such a task could be accomplished by introducing a suitable operator, in addition to *AppendChar* and *AppendRepeat*, which collapses *non-distinguishable* states of RLA (at the moment, the major stumbling block is the problem of finding an appropriate notion of distinguishability for the states of an RLA).

In Section 2.7 we focused on the problem of representing and reasoning on (possibly infinite) sets of periodical time granularities, rather than single time granularities.

In this respect, we defined a proper subclass of Büchi automata that recognize exactly the regular ω-languages consisting of ultimately periodic words and we gave efficient solutions to several basic problems in automata theory, precisely, the emptiness problem, the membership problem, the equivalence problem, and the size-optimization problem. We also dealt with the problem of comparing sets of granularities with respect to some distinguished relations (e.g., partition, group, sub-granularity, aligned refinement, etc.). Finally, we showed how to apply the proposed framework to a real-world application taken from clinical medicine. As for further research directions, one may consider more generic notions of sequential automata, thus making it possible to capture possibly non-regular ω-languages of ultimately periodic words.

Finally, it is worth mentioning that most constructs for time granularity are typically one-dimensional. A possible extension may concern moving from linear structures to tree-shaped structures, which are more suited to deal with (infinite) hierarchies of differently grained time granularities. In [77, 78, 81] Montanari et al. consider the problem of representing and reasoning on such such a kind of multi-layered structures. They introduce the so-called n-layered temporal structures, downward unbounded ω-layered temporal structures, and upward unbounded ω-layered temporal structures and they prove several decidability results with respect to temporal and classical logics. At the end of Chapter 3 (Section 3.5.3), we shall come back to this subject by briefly reviewing some definitions, expressiveness results, and decidability results for multi-layered temporal structures. Then, we shall introduce a more general notion of ω-layered temporal structure, which subsumes the previous definitions, and we shall apply the decidability results outlined in Chapter 3 to this kind of temporal structures.

3

Tree Automata and MSO Logics

The nature of this chapter is more theoretical in that we deal with decision procedures for the model checking problem of monadic second-order logic interpreted over tree structures. Precisely, we develop a general automaton-based method to establish the decidability of monadic second-order theories of deterministic colored trees. The method takes advantage of (i) the strong correspondence between monadic second-order formulas interpreted over colored trees and automata recognizing tree languages and (ii) a notion of indistinguishability of colored trees with respect to tree automata in order to allow the replacing of a tree by a 'retraction' of it (i.e., a suitable abstraction of the original tree). By exploiting such a method, one can reduce, in a uniform way, instances of the model checking problem to equivalent instances of the acceptance problem for tree automata over regular trees, which can be easily decided. We show that such an automaton-based approach works effectively for a large class of complex (non-regular) trees, which we call reducible trees, including meaningful families of relational structures. Finally, we consider the model checking problem for monadic second-order logic (and its chain fragment) interpreted over the so-called multi-layered temporal structures, which are tree-shaped structures well suited for modeling and reasoning about temporal relations at different 'grain levels'. We introduce a new notion of ω-layered temporal structure, which subsumes previous definitions found in the literature, and we apply the decidability method to solve model checking problems for this kind of structure.

3.1 Background knowledge

Monadic second-order (MSO) logic has been introduced as an extension of first-order logic with set variables, namely, variables that have to be instantiated by sets of elements. Such a logic is powerful enough to express several interesting (and non-trivial) properties of graph structures like reachability, planarity, etc., but its model checking problem is highly undecidable. Nonetheless, several approaches to the model checking problem for MSO logic have been proposed in the literature.

Two fundamental results in this field are Büchi's and Rabin's theorems [5, 96], which show the decidability of the model checking problem for the linear order $(\mathbb{N}, <)$ and for the infinite complete binary tree, respectively. Both results have been achieved

by reducing the model checking problem to the problem of testing the emptiness of the language recognized by a suitable automaton.

Büchi's theorem has also been exploited to deal with extensions of $(\mathbb{N}, <)$ with unary predicates. The acceptance problem for an extended structure $(\mathbb{N}, <, P)$, where $P \subseteq \mathbb{N}$, is the problem of determining, for any given Büchi automaton M, whether M accepts (the infinite word that encodes) $(\mathbb{N}, <, P)$. Elgot and Rabin gave a positive answer to this problem for some relevant predicates, such as the factorial one [50], and later Carton and Thomas generalized such a result to the class of the so-called residually (or profinitely) ultimately periodic words [19, 20].

In [87], Muller and Schupp brought the interest to logical theories of graphs by identifying a large family of structures, the so-called context-free (or end-regular) graphs, which enjoy decidable MSO theories. Several other results have been achieved by following different approaches. As an example, the transformational approach allows one to transfer decidability results to new structures obtained via decidability-preserving transformations starting from a set of well-known decidable structures. Examples of decidability-preserving transformations are MSO-definable interpretations, MSO-definable transductions, inverse rational mappings, unfoldings, tree-graph operations, etc. Moreover, according to [37], one can distinguish between properties evaluated on a given system and properties evaluated on its behavior (i.e., the unfolding of the corresponding transition graph). By exploiting Muchnik's theorem [100, 38, 115, 116], one can transfer a decidability result from the former kind of problem to the latter.

Other approaches define structures with decidable MSO theories by means of rewriting systems [21, 22], transducers [10], or equational systems [35, 36]. As an example, prefix-recognizable graphs [9] can be equivalently described in terms of rational restrictions of inverse rational mappings of the infinite complete binary tree [23, 25], MSO-definable interpretations of infinite regular trees [114], vertex-replacement equational graphs [4], ε-closures of transition graphs of pushdown systems [18, 117].

3.2 Transformations of trees

In this section we introduce and compare several natural tree transformations. The definitions and results reported here will be extensively used in the following sections, in particular in Section 3.4, where we shall introduce a large class of trees with a decidable MSO theory and prove closure properties of this class with respect to a number of tree transformations. In general, a tree transformation maps a Λ-labeled Σ-colored tree to a Λ'-labeled Σ'-colored tree, where Λ may be different from Λ' and Σ may be different from Σ'. By properly choosing a single set Λ'' of labels (e.g., $\Lambda'' = \Lambda \cup \Lambda'$) and a single set Σ'' of colors (e.g., $\Sigma'' = \Sigma \cup \Sigma'$) for both input and output trees, we can restrict our attention to transformations that maps Λ''-labeled Σ''-colored trees to Λ''-labeled Σ''-colored trees. From now on, unless otherwise stated, we assume that tree are labeled over a finite set $\Lambda = \{a_1, ..., a_k\}$ and colored over a finite set $\Sigma = \{c_1, ..., c_m\}$.

Below we define some basic operations on trees.

Definition 3.2.1 *We define the following basic operations on trees:*

- **explicit construction***: given a tuple of trees $T_1, ..., T_k$ and a color $c \in \Sigma$, it returns the tree $c\langle T_1, ..., T_k\rangle$;*
- **subtree selection***: given a tree T and a vertex $v \in \mathcal{D}om(T)$, it returns the tree $T^{\downarrow v}$;*
- **vertex-wise juxtaposition***: given two trees T_1 and T_2, it returns the tree $T_1.T_2$ such that $\mathcal{D}om(T_1.T_2) = \mathcal{D}om(T_1) \cup \mathcal{D}om(T_2)$ and for every $v \in \mathcal{D}om(T_1.T_2)$, $(T_1.T_2)(v) = (T_1(v), T_2(v))$; in case v does not belong to $\mathcal{D}om(T_1)$ (resp. $\mathcal{D}om(T_2)$), a dummy color $\perp$ is used instead of $T_1(v)$ (resp. $T_2(v)$).*

3.2.1 Recolorings

Here, we introduce recolorings of trees. We distinguish between four kinds of recolorings: memoryless recolorings, finite-state recolorings, finite-state recolorings with bounded lookahead, and finite-state recolorings with rational lookahead.

Definition 3.2.2 *Given a tree T and a function $\Omega : \Sigma \to \Sigma$, the* memoryless recoloring *of T via Ω is the tree $\Omega(T)$ such that $\mathcal{D}om\big(\Omega(T)\big) = \mathcal{D}om(T)$ and for every $v \in \mathcal{D}om(T)$, $\Omega(T)(v) = \Omega\big(T(v)\big)$.*

Note that the operation of memoryless recoloring is local, in the sense that it is performed on each vertex v of an input tree T, independently from the colors of the ancestors and the descendants of v. A more powerful notion of recoloring is that of finite-state recoloring, which is specified in terms of deterministic Mealy tree automata (shortly Mealy tree automata) and which allows the recoloring of a vertex v on the grounds of the colors of its ancestors. A Mealy tree automaton can be thought of as a deterministic tree automaton (without acceptance conditions), endowed with an output function which maps the current state and color to a new color.

Definition 3.2.3 *A* Mealy tree automaton *is a tuple $N = (S, \Sigma, \Lambda, \delta, s_0, \Omega)$, where*

- *S is a finite set of states,*
- *Σ is a finite set of colors (for both input and output trees),*
- *Λ is a finite set of labels,*
- *$\delta : S \times \Sigma \times \Lambda \to S$ is a transition function,*
- *s_0 is an initial state,*
- *$\Omega : S \times \Sigma \to \Sigma$ is a recoloring function.*

The unique *run* of N on a tree T is the S-colored tree R such that $\mathcal{D}om(R) = \mathcal{D}om(T)$, $R(\varepsilon) = s_0$, and for every vertex v of R and every a-successor v_a of v, $R(v_a) = \delta\big(R(v), T(v), a\big)$. The *output* of N on T is the tree T' such that $\mathcal{D}om(T') = \mathcal{D}om(T)$ and for every vertex $v \in \mathcal{D}om(T)$, $T'(v) = \Omega\big(R(v), T(v)\big)$. In such a case, we say that T' is a *finite-state recoloring* of T via N.

It is worth mentioning that the vertex-wise juxtaposition of a tree T_1 with a *regular* tree T_2 can be viewed as the finite-state recoloring specified by a Mealy tree automaton

$N = (S, \Sigma \cup \Sigma^2, \Lambda, \delta, v_0, \Omega)$ such that $S = \mathcal{D}om(G)$, $\delta(v, c, a) = v_a$, $v_0 \in \mathcal{D}om(G)$, and $\Omega(v, c) = \big(c, G(v)\big)$ for every $c \in \Sigma$ (the values of Ω on Σ^2 are not relevant), provided that G is a finite colored graph whose unfolding from v_0 yields T_2 and v_a denotes the a-successor of v in G. Note that a converse result does not hold; in particular, there are finite-state recolorings which cannot be characterized in terms of vertex-wise juxtapositions with regular trees. Examples of such cases are given by the operations of path heritage and regular exponentiation, described in the following.

Let assume that $(\Sigma, \cdot)$ is a *finite groupoid* (i.e., a finite set endowed with a binary, possibly non-associative, operation $\cdot$). Given a tree T, the *path heritage* of T returns the tree T' such that $\mathcal{D}om(T') = \mathcal{D}om(T)$ and for every $v \in \mathcal{D}om(T)$, $T'(v) = T(\pi_0) \cdot T(\pi_1) \cdot ... \cdot T(\pi_{|\pi|})$[1], where π denotes the access path of v in T and π_i denotes the $i+1$-th vertex along π. It is easy to see that the tree T' is the finite-state recoloring of T via the Mealy tree automaton $N = (S, \Sigma, \Lambda, \delta, \perp, \Omega)$ such that $S = \Sigma \cup \{\perp\}$, $\delta(\perp, c, a) = c$, $\delta(c', c, a) = c' \cdot c$, $\Omega(\perp, c) = c$, and $\Omega(c', c) = c' \cdot c$.

As for the operation of regular exponentiation, let $(\Sigma, \cdot)$ be a *finite multiplicative monoid* and let f be a function over the set of natural numbers[2]. For every $c \in \Sigma$ and for every $n \in \mathbb{N}$, we denote by $c^{f(n)}$ the $f(n)$-fold (left-associative) product $c \cdot ... \cdot c$ (if $f(n) = 0$, then $c^{f(n)}$ is the identity element of $(\Sigma, \cdot)$). Then, given a tree T, the *exponentiation* of T via f is the tree T' such that $\mathcal{D}om(T') = \mathcal{D}om(T)$ and for every $v \in \mathcal{D}om(T)$, $T'(v) = T(v)^{f(|v|)}$, where $|v|$ denotes the length of the access path of v in T. Now, if the function f is *ultimately periodic with respect to finite monoids* (refer to Section 1.6 for formal definitions and properties, examples of such functions are i^2, 2^i, $2^i - i^2$, i^i, $i!$, and the exponential tower $2^{2^{\cdot^{\cdot^{\cdot^2}}}}$), then, for each color $c \in \Sigma$, one can compute $p_c \geq 0$ and $q_c > 0$ such that for every $i \in \mathbb{N}$, $c^{f(i)} = c^{f([i]_{p_c, q_c})}$ holds, where $[x]_{p_c, q_c}$ denotes either the value x or the value $((x - p_c) \bmod q_c) + p_c$, depending on whether $x < p_c$ or not. Let $p = max\{p_c : c \in \Sigma\}$ and $q = lcm\{q_c : c \in \Sigma\}$. Clearly, $c^{f(i)} = c^{f([i]_{p,q})}$ holds for every color c and for every natural number i. This implies that the tree T' resulting from the exponentiation of T via f is a finite-state recoloring of T via the Mealy tree automaton $N = (S, \Sigma, \Lambda, \delta, 0, \Omega)$ such that $S = \{i : 0 \leq i < p + q\}$, $\delta(i, c, a) = [i + 1]_{p,q}$, and $\Omega(i, c) = c^i$. Thus, if f is an ultimately periodic function with respect to finite monoids, we say that T' is a *regular* exponentiation of T via f. In Section 3.4.3, we shall see that the operation of regular exponentiation preserves decidability of MSO theories of trees. In view of the characterizations given in Section 1.6, such a result generalizes previous results by Montanari and Puppis on *residually ultimately periodic factorizations* [82, 83, 84] and by Carton and Thomas on *residually* (or *profinitely*) *ultimately periodic predicates* [19, 20].

Mealy tree automata works on trees in a top-down fashion, reading the colors of the vertices and storing some bounded amount of information in their control state. In order to define more expressive operations on trees, one can imagine allowing a

[1] Here we assume that $\cdot$ associates to the left.

[2] For the sake of simplicity, here we assume that $(\Sigma, \cdot)$ is a finite monoid; however, if the function f maps naturals numbers to positive values only, then it is possible to relax such an assumption and let $(\Sigma, \cdot)$ be a finite groupoid.

Mealy tree automaton to inspect the subtree issued from the current position before processing it. We call such a facility 'lookahead' and we distinguish between two notions of lookahead, namely, bounded lookahead and rational lookahead. The former allows a Mealy tree automaton to inspect the subtree up to a bounded depth; the latter allows a Mealy tree automaton to inspect the whole (possibly infinite) subtree and classify it according to a given family of rational tree languages. Hereafter, we denote by $\mathcal{T}$ (resp., $\mathcal{T}^{fin}$, $\mathcal{T}^{reg}$) the set of all trees (resp., finite trees, regular trees). Similarly, given a finite prefix-closed set $D \subseteq \Lambda^*$, we denote by $\mathcal{T}^D$ the set of all trees whose domain is a *subset* of D. Note that, for any given finite prefix-closed set $D \subseteq \Lambda^*$, we have $\mathcal{T}^D \subsetneq \mathcal{T}^{fin} \subsetneq \mathcal{T}^{reg} \subsetneq \mathcal{T}$.

Definition 3.2.4 *A* Mealy tree automaton with bounded lookahead *is a tuple* $N = (S, \Sigma, \Lambda, D, \delta, s_0, \Omega)$*, where*

- S *is a finite set of states,*
- Σ *is a finite set of colors,*
- Λ *is a finite set of labels,*
- D *is a finite prefix-closed subset of* Λ^**,*
- $\delta : S \times \mathcal{T}^D \times \Lambda \to S$ *is a transition function,*
- s_0 *is an initial state,*
- $\Omega : S \times \mathcal{T}^D \to \Sigma$ *is a recoloring function.*

The unique *run* of N on a tree T is the S-colored tree R such that $\mathcal{D}om(R) = \mathcal{D}om(T)$, $R(\varepsilon) = s_0$, and, for every vertex v of R and every a-successor v_a of v, $R(v_a) = \delta\big(R(v), T^{\downarrow v}|_D, a\big)$, where $T^{\downarrow v}$ denotes the subtree of T issued from the vertex v and $T^{\downarrow v}|_D$ denotes the tree obtained from $T^{\downarrow v}$ by restricting its domain to the set D (here we assume that the domain of $T^{\downarrow v}$ is a prefix-closed subset of Λ^*). The *output* of N on T is the tree T' such that $\mathcal{D}om(T') = \mathcal{D}om(T)$ and for every vertex $v \in \mathcal{D}om(T)$, $T'(v) = \Omega\big(R(v), T^{\downarrow v}|_D\big)$. In such a case, we say that T' is a *finite-state recoloring with bounded lookahead* of T via N.

Definition 3.2.5 *A* Mealy tree automaton with rational lookahead *is a tuple* $N = (S, \Sigma, \Lambda, \mathcal{L}, \delta, s_0, \Omega)$*, where*

- S *is a finite set of states,*
- Σ *is a finite set of colors,*
- Λ *is a finite set of labels,*
- $\mathcal{L} = \{L_1, ..., L_p\}$ *is a finite collection of rational tree languages (namely, each* L_i *is the language recognized by a tree automaton or, equivalently, the set of all models of an MSO formula interpretable over colored trees),*
- $\delta : S \times \mathscr{P}([p]) \times \Lambda \to S$ *is a transition function,*
- s_0 *is an initial state,*
- $\Omega : S \times \mathscr{P}([p]) \to \Sigma$ *is a recoloring function.*

The unique *run* of N on a tree T is the S-colored tree R such that $\mathcal{D}om(R) = \mathcal{D}om(T)$, $R(\varepsilon) = s_0$, and for every vertex v of R and for every a-successor v_a of v, $R(v_a) = \delta\big(R(v), I, a\big)$, where I is the set of all and only the indices $i \in [p]$ such that $T^{\downarrow v} \in L_i$. The *output* of N on T is the tree T' such that $\mathcal{D}om(T') = \mathcal{D}om(T)$ and,

for every vertex $v \in \mathcal{D}om(T)$, $T'(v) = \Omega\big(R(v), I\big)$, where I is the set of all and only the indices $i \in [p]$ such that $T^{\downarrow v} \in L_i$. In such a case, we say that T' is a *finite-state recoloring with rational lookahead* of T via N.

Clearly, the notions of memoryless recoloring, finite-state recoloring, finite-state recoloring with bounded lookahead, and finite-state recoloring with rational lookahead form a strictly increasing hierarchy of tree operations. Moreover, it is straightforward to extend the above notions to the case of Δ-*augmented* trees, where Δ is any arbitrary finite set disjoint from Σ. Precisely, if T is a (Σ-colored) Δ-augmented tree and $\Omega : \Sigma \cup \Delta \to \Sigma \cup \Delta$ a recoloring function, then we assume that, for every $c \in \Sigma$, $\Omega(c) \in \Sigma$ and, for every $b \in \Delta$, $\Omega(b) = b$ (namely, Ω maps colors from Σ to colors in Σ and it preserves the markers from Δ). In such a way, the memoryless recoloring of T via Ω can be given the status of (Σ-colored) Δ-augmented tree. Similarly, if $N = (S, \Sigma \cup \Delta, \Lambda, \delta, \bot, \Omega)$ is a Mealy tree automaton and T is a (Σ-colored) Δ-augmented tree provided in input to N, then we assume that for every $s \in S$ and every $c \in \Sigma \cup \Delta$, either $\Omega(s, c) \in \Sigma$ or $\Omega(s, c) = c$ holds, depending on whether $c \in \Sigma$ or $c \in \Delta$. As for the notions of finite-state recolorings with bounded (resp., rational) lookahead, we assume that for every $B \in \mathcal{T}^D$ (resp., $B \in \mathcal{T}$), the recoloring function Ω satisfies either $\Omega(s, B) \in \Sigma$ or $\Omega(s, B) = B(\varepsilon)$, depending on whether $B(\varepsilon) \in \Sigma$ or $B(\varepsilon) \in \Delta$.

3.2.2 Tree substitutions

Here we consider particular kinds of transformations on trees, precisely tree morphisms and tree insertions. Both transformations are derived from the notion of second-order tree substitution, which, in its turn, can be defined on the ground of the notion of first-order tree substitution.

Definition 3.2.6 *Given a tree T whose domain is a prefix-closed subset of Λ^*, given a prefix-free set $L \subseteq \mathcal{D}om(T)$, and given a tree B_v for each $v \in L$, we denote by $T[B_v/v]_{v\in L}$ the tree resulting from the (simultaneous)* first-order substitution *in T of each vertex $v \in L$ by B_v, formally, the tree T' defined as follows:*

- $\mathcal{D}om(T') = \big(\mathcal{D}om(T) \setminus (L \cdot \Lambda^*)\big) \cup \big\{v \cdot w : v \in L, w \in \mathcal{D}om(B_v)\big\}$,
- $T'(u) = \begin{cases} T(u) & \text{if } u \in \mathcal{D}om(T) \setminus (L \cdot \Lambda^*), \\ B_v(w) & \text{if } u = v \cdot w, v \in L, w \in \mathcal{D}om(B_v). \end{cases}$

This kind of operation can be though of as replacing subtrees in a tree T. Usually, such an operation is applied to the leaves of a tree (namely, L is usually chosen to be a subset of $\mathcal{F}r(T)$). Thus, by a slight abuse of notation, we shall write $T[B_c/c]_{c\in\Sigma}$ to mean the first order substitution in T of all *c-colored leaves* by B_c, simultaneously for all $c \in \Sigma$.

Unlike first-order substitutions, second-order substitutions can occur simultaneously at vertices along the same path and they can preserve the subtrees rooted at the successors of a substitution occurrence. Even though second-order tree substitutions are usually defined over *ranked* terms (i.e., trees where the out-degree of each vertex is

uniquely determined by the 'arity' of the color assigned to that vertex), here we adapt the standard notion to the case of (deterministic) unranked trees (all results involving second-order tree substitutions can be easily transferred to this framework). In order to specify how the subtrees issued from the 1-st, ..., k-th successor of a substitution occurrence must be attached to a replacing term B, we mark the leaves of B with elements in bijection with the edge labels in Λ; these elements will act as placeholders for the subtrees to be attached. For the sake of simplicity, we can assume that Λ and Σ are disjoint sets and we let the placeholders of a replacing term B be exactly the elements from Λ. We then define a *replacing term* as a Λ-augmented tree.

Providing a direct definition of second-order tree substitution that works fine also in the case of infinite trees, is rather complicated. Instead, by following the presentation of Courcelle in [34], we first define second-order tree substitutions for finite trees and then we extend the definition to the infinite case by exploiting the fact that second-order substitutions are monotone in their arguments with respect to a suitable ω-complete partial order over trees.

Definition 3.2.7 *Given a finite tree T and a (possibly infinite) replacing term B_c for each $c \in \Sigma$, we denote by $T[\![B_c/c]\!]_{c\in\Sigma}$ the tree resulting from the* second-order tree substitution *in T of each c-colored vertex by B_c, simultaneously for all $c \in \Sigma$, that is, the tree T' inductively defined as follows:*

$$T' = \begin{cases} \emptyset & \text{if } T = \emptyset, \\ B_{T(\varepsilon)}\big[T^{(a)}/a\big]_{a\in\Lambda} & \text{otherwise,} \end{cases}$$

where, for each label $a \in \Lambda$, $T^{(a)} = T^{\downarrow a}[\![B_c/c]\!]_{c\in\Sigma}$.

As an example, Figure 3.1 depicts the result of a second-order tree substitution of c by B in T.

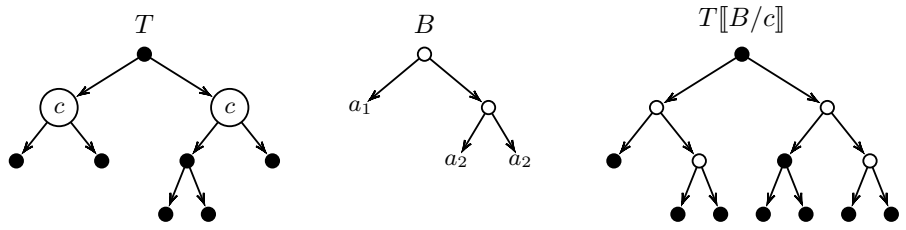

Figure 3.1: An example of second-order tree substitution.

We now extend the notion to the case of infinite trees. We first define the partial order $\sqsubseteq$ over the set $\mathcal{T}$ of all (possibly infinite) trees as follows: for every pair of trees $T_1, T_2 \in \mathcal{T}$, $T_1 \sqsubseteq T_2$ holds iff $\mathcal{D}om(T_1) \subseteq \mathcal{D}om(T_2)$ and for every $v \in \mathcal{D}om(T_1)$, $T_1(v) = T_2(v)$. It is easy to see that the empty tree is the least element of $\mathcal{T}$ with respect to $\sqsubseteq$. Moreover, $(\mathcal{T}, \sqsubseteq)$ is an ω-complete partial order, since every $\sqsubseteq$-directed subset X of $\mathcal{T}$ has a least upper bound $Sup(X)$, defined by

$\mathcal{D}om\big(Sup(X)\big) = \bigcup_{T\in X} \mathcal{D}om(T)$ and, for all $v \in \mathcal{D}om\big(Sup(X)\big)$, $Sup(X)(v) = T(v)$, where T is any tree in X such that $v \in \mathcal{D}om(T)$. We then denote by $\mathcal{T}_\Lambda$ the set of all (Σ-colored) Λ-augmented trees. By construction, any second-order tree substitution, viewed as a function from $\mathcal{T}^{fin} \times \big(\mathcal{T}_\Lambda\big)^{|\Sigma|}$ to $\mathcal{T}$ and mapping $\big(T, (B_c)_{c\in\Sigma}\big)$ to $T[\![B_c/c]\!]_{c\in\Sigma}$, is *monotone*. This implies (cf. Proposition 2.4.2 of [34]) that any second-order tree substitution can be extended *in a unique way* to an ω-continuous function from $\mathcal{T} \times \big(\mathcal{T}_\Lambda\big)^{|\Sigma|}$ to $\mathcal{T}$. As a matter of fact, from the above mentioned results, we also know that, for every tree T and for every tuple of replacing terms $(B_c)_{c\in\Sigma}$,

$$T[\![B_c/c]\!]_{c\in\Sigma} = B_{T(\varepsilon)}\big[T^{(a)}/a\big]_{a\in\Lambda} \tag{3.2.1}$$

where $T^{(a)} = T^{\downarrow a}[\![B_c/c]\!]_{c\in\Sigma}$ for all $a \in \Lambda$.

Like recoloring operations, second-order tree substitutions can be easily extended to the case of Δ-augmented trees, where Δ is any arbitrary finite set disjoint from Σ. If T is a Δ-augmented tree and $T[\![B_c]\!]_{c\in\Sigma\cup\Delta}$ is a second-order tree substitution, then we assume that for every $c \in \Sigma$, the replacing term B_c is a Σ-colored Δ-augmented tree and for every $b \in \Delta$, B_b is the singleton tree b. Under such an assumption, Δ-augmented trees are mapped by second-order tree substitutions to Δ-augmented trees. Thus, by letting $\Delta = \Lambda$, one can combine sequences of second-order tree substitutions and prove that such an operation is associative. In the following, we prove associativeness and other results for tree substitutions, under the assumption that T is a Σ-colored Λ-augmented tree.

Lemma 3.2.8 *Second-order substitutions are homomorphisms with respect to first-order substitutions, namely, given a tree T and some replacing terms $(B_a)_{a\in\Lambda}$ and $(B'_c)_{c\in\Sigma}$, we have*

$$T\big[B_a/a\big]_{a\in\Lambda}[\![B'_c/c]\!]_{c\in\Sigma} = T[\![B'_c/c]\!]_{c\in\Sigma}\big[B_a[\![B'_c/c]\!]_{c\in\Sigma}/a\big]_{a\in\Lambda},$$

Proof. A proof can be found in Section 3.5 of [34]. ∎

Proposition 3.2.9 *Second-order substitutions are associative, namely, given a tree T and some replacing terms $(B_c)_{c\in\Sigma}$ and $(B'_c)_{c\in\Sigma}$, we have*

$$T[\![B_c/c]\!]_{c\in\Sigma}[\![B'_c/c]\!]_{c\in\Sigma} = T[\![B_c[\![B'_{c'}/c']\!]_{c'\in\Sigma}/c]\!]_{c\in\Sigma}.$$

Proof. We fix two m-tuples of replacing terms $(B_c)_{c\in\Sigma}$ and $(B'_d)_{d\in\Sigma}$. By exploiting Lemma 3.2.8 and Equation 3.2.1, we easily see that

$$\begin{aligned}
T[\![B_c/c]\!]_{c\in\Sigma}[\![B'_d/d]\!]_{d\in\Sigma} &= B_{T(\varepsilon)}\big[T^{\downarrow a}[\![B_c/c]\!]_{c\in\Sigma}/a\big]_{a\in\Lambda}[\![B'_d/d]\!]_{d\in\Sigma} \\
&= B_{T(\varepsilon)}[\![B'_d/d]\!]_{d\in\Sigma}\big[T^{\downarrow a}[\![B_c/c]\!]_{c\in\Sigma}[\![B'_d/d]\!]_{d\in\Sigma}/a\big]_{a\in\Lambda} \\
&= B_{T(\varepsilon)}[\![B'_d/d]\!]_{d\in\Sigma}\big[T^{\downarrow a}[\![B_c[\![B'_d/d]\!]_{d\in\Sigma}/c]\!]_{c\in\Sigma}/a\big]_{a\in\Lambda} \\
&= T[\![B_c[\![B'_d/d]\!]_{d\in\Sigma}/c]\!]_{c\in\Sigma}.
\end{aligned}$$

∎

We can view any second-order tree substitution either as a function σ specified by some replacing terms $B_1, ..., B_m$ (recall that $\Sigma = \{c_1, ..., c_m\}$) and mapping a tree T to the tree $T[\![B_1/c_1, ..., B_m/c_m]\!]$, or as a function γ specified by a tree T and by an n-tuple of colors $c_{i_1}, ..., c_{i_n}$, with $1 \leq n \leq m$, and mapping the n-tuple of Λ-augmented trees $(B_1, ..., B_n)$ to $T[\![B_1/c_{i_1}, ..., B_n/c_{i_n}]\!]$. Such two different ways of viewing the operation of second-order tree substitution give rise to the notions of tree morphism and tree insertion, respectively.

Definition 3.2.10 *A* tree morphism *is a function σ, specified by an m-tuple of replacing terms $B_1, ..., B_m$, which maps a tree T to the tree*

$$\sigma(T_i) = T_i[\![B_1/c_1, ..., B_m/c_m]\!].$$

We say that σ is a *regular* tree morphism if $B_1, ..., B_m$ are regular Λ-augmented trees.

Definition 3.2.11 *A* tree insertion *is a function γ, specified by a tree T and by an n-tuple of colors $c_{i_1}, ..., c_{i_n}$ (with $1 \leq n \leq m$), which maps an n-tuple of Λ-augmented trees $B_1, ..., B_n$ to the tree*

$$\gamma(B_1, ..., B_n) = T_i[\![B_1/c_{i_1}, ..., B_n/c_{i_n}]\!].$$

We say that γ is a *regular* tree insertion if T is a regular tree. As a matter of fact, it comes natural to extend the notion of (regular) tree insertion by allowing substitutions *in more than one tree*. Thus, given an n'-tuple $\bar{\gamma}$ of (regular) tree insertions $\gamma_1, ..., \gamma_{n'}$, we can give the function $\bar{\gamma}$ mapping $(B_1, ..., B_n)$ to $(\gamma_1(B_1, ..., B_n), ..., \gamma_{n'}(B_1, ..., B_n))$ the status of (regular) tree insertion.

3.2.3 Tree transducers

We now introduce another family of transformations on trees, which are specified by the so-called (deterministic top-down) tree transducers. For the sake of simplicity, any transformation defined by a tree transducer is called tree transduction. We shall prove that tree transductions subsume both finite-state recolorings and regular tree morphisms. Conversely, it is also possible to prove that any tree transduction can be viewed as a composition of finite-state recolorings and regular tree morphisms. For further explanations and results about tree transducers, we refer to [51, 52, 29, 60, 32]. As usual, we assume that trees are colored over a finite ordered set $\Sigma = \{c_1, ..., c_m\}$ and labeled over a finite ordered set $\Lambda = \{a_1, ..., a_k\}$.

A tree transducer works in a top-down fashion on its input tree by replacing, according to the current state and color, the vertex under its head with a new (regular) tree. Moreover, a tree transducer may spread different states among different copies of the same subtree; in other words, this means that, 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 the leaves of the replacing terms with symbols belonging to a finite set $\Delta = \{b_1, ..., b_l\}$, disjoint from Σ. Then, we identify the subtree to be attached at each leaf of a replacing term by using a function $\lambda : \Delta \rightarrow \Lambda$ that maps each Δ-colored leaf of

the replacing term to the corresponding placeholder for the substitution. Below, we give a formal definition of tree transducer (it is worth mentioning that different but equivalent definitions can be found in the literature). Hereafter, we denote by $\mathcal{T}_{\Delta}^{reg}$ the set of all *regular* (Σ-colored) Δ-augmented trees.

Definition 3.2.12 *A* tree transducer *is a tuple of the form* $M = (S, \Sigma, \Delta, \Lambda, \lambda, \delta, s_0)$*, where*

- S *is a finite set of states,*
- Σ *is a finite set of colors,*
- Δ *is a finite set of markers,*
- Λ *is a finite set of labels,*
- $\lambda : \Delta \to \Lambda$ *is the placeholder function,*
- $\delta : S \times \Sigma \to \mathcal{T}_{\Delta}^{reg} \times S^{\Delta}$ *is a transition function,*
- s_0 *is an initial state.*

In order to define the output of M running on an infinite tree T, we follow the presentation in [29] (this eases definitions for the case of tree transducers with rational lookahead). Recall that $\mathcal{T}$ is the set of all (possibly infinite) trees. For every $n \in \mathbb{N}$, we recursively define the function $M^n : S \times \mathcal{T} \to \mathcal{T}$ as follows:

- $M^n(s, \emptyset) = \emptyset$ for every $n \in \mathbb{N}$,
- $M^0(s, T) = T$ for every tree $T \in \mathcal{T}$,
- $M^{n+1}(s, T) = B\big[T^{(b)}/b\big]_{b \in \Delta}$ for every $n \in \mathbb{N}$ and for every non-empty tree $T \in \mathcal{T}$, where $T^{(b)} = M^n\big(s_b, T^{\downarrow\lambda(b)}\big)$ and $\delta\big(s, T(\varepsilon)\big) = \big(B, (s_b)_{b \in \Delta}\big)$.

We then consider the usual ω-complete partial order $\sqsubseteq$ on $\mathcal{T}$ such that $T_1 \sqsubseteq T_2$ holds iff T_1 can be obtained by pruning some subtrees of T_2. It is easy to verify, by exploiting induction on n, that, given any tree T, the sequence $M^0(s_0, T)$, $M^1(s_0, T)$, $M^2(s_0, T)$, ... is increasing with respect to $\sqsubseteq$ (to see this, simply prune the subtrees issued from the heads of the automaton). This allows us to define the 'limit' $M^{\omega}(s_0, T)$ of the sequence $\big(M^n(s_0, T)\big)_{n \in \mathbb{N}}$ as its least upper bound with respect to $\sqsubseteq$. The resulting tree is called the *output* of M on T. Thus, the *tree tranduction* defined by M is th function that maps a tree T to the tree $M^{\omega}(s_0, T)$.

Like finite-state recolorings, one can imagine extending the notion of tree transducer with the facility of rational lookahead. Below, we define tree transducers with bounded and rational lookahead. It is worth noticing that (deterministic top-down) tree transducers, even if extended with bounded or rational lookahead, are not closed under functional composition, namely, the tree resulting from the application of two tree transducers is not, in general, expressible as the output of a single tree transducer (cf. [51, 52]). As a matter of fact, the closure under composition holds if some restricted notions of tree transducer are adopted, such as total or non-deleting tree transducers (see [97, 110]).

Definition 3.2.13 *A* transducer with bounded *(resp.,* rational*)* lookahead *is a tuple* $M = (S, \Sigma, \Delta, \Lambda, \lambda, D, \delta, s_0)$ *(resp.,* $M = (S, \Sigma, \Delta, \Lambda, \lambda, \mathcal{L}, \delta, s_0)$*), where*

- S *is a finite set of states,*
- Σ *is a finite set of colors,*

- Δ *is a finite set of markers,*
- Λ *is a finite set of labels,*
- $\lambda : \Delta \to \Lambda$ *is the placeholder function,*
- D *is a finite prefix-closed subset of* Λ^* *(resp.,* $\mathcal{L} = \{L_1, ..., L_p\}$ *is a finite collection of rational tree languages),*
- δ *is a transition function from* $S \times \mathcal{T}^D$ *(resp.,* $S \times \mathscr{P}([p])$*) to* $\mathcal{T}_\Delta^{reg} \times S^\Delta$,
- s_0 *is an initial state.*

We now define the output of a tree transducer M with bounded lookahead. For every $n \in \mathbb{N}$, we let $M^n : S \times \mathcal{T} \to \mathcal{T}$ be the function such that

- $M^n(s, \emptyset) = \emptyset$ for every $n \in \mathbb{N}$,
- $M^0(s, T) = T$ for every tree $T \in \mathcal{T}$,
- $M^{n+1}(s, T) = B\big[T^{(b)}/b\big]_{b\in\Delta}$ for every $n \in \mathbb{N}$ and for every non-empty tree $T \in \mathcal{T}$, where $T^{(b)} = M^n\big(s_b, T^{\downarrow\lambda(b)}\big)$ and $\delta\big(s, T|_D\big) = \big(B, (s_b)_{b\in\Delta}\big)$.

For rational lookahead, the function $M^n : S\times\mathcal{T} \to \mathcal{T}$, is defined as follows:

- $M^n(s, \emptyset) = \emptyset$ for every $n \in \mathbb{N}$,
- $M^0(s, T) = T$ for every tree $T \in \mathcal{T}$,
- $M^{n+1}(s, T) = B\big[T^{(b)}/b\big]_{b\in\Delta}$ for every $n \in \mathbb{N}$ and for every non-empty tree $T \in \mathcal{T}$, where $T^{(b)} = M^n\big(s_b, T^{\downarrow\lambda(b)}\big)$ and $\delta(s, I) = \big(B, (s_b)_{b\in\Delta}\big)$, with I being the set of all and only the indices $i \in [p]$ such that $T \in L_i$.

By proceeding as in the case of a simple tree transducer, we introduce the natural partial order $\sqsubseteq$ on trees and we define the output of M on a tree T as the 'limit' $M^\omega(s_0, T)$ of the increasing sequence $M^0(s_0, T)$, $M^1(s_0, T)$, $M^2(s_0, T)$, ... The corresponding function that maps a tree T to the tree $M^\omega(s_0, T)$ is called *tree transduction with bounded/rational lookahead.*

Proposition 3.2.14 *Finite-state recolorings (with bounded, rational lookahead) and regular tree morphisms are special cases of tree transductions (with bounded, rational lookahead). Conversely, given a tree transducer M (with bounded, rational lookahead), the output of M on a tree T can be obtained by applying to T first a regular tree morphism, then a finite state-recoloring (with bounded, rational lookahead), and finally another regular tree morphism.*

Proof. We restrict our attention to finite-state recolorings and tree transductions without lookahead (the cases for bounded and rational lookahead can be handled in a similar way). Let $N = (S, \Sigma, \Lambda, \delta, s_0, \Omega)$ be a Mealy tree automaton generating the tree T' from an input tree T. We have to define a tree transducer $M = (S', \Sigma, \Delta, \Lambda, \lambda, \delta', s'_0)$ that generates T' from T. We set $S' = S$, $s'_0 = s_0$, $\Delta = \Lambda$, $\lambda(a) = a$ for every $a \in \Delta$, and $\delta'(s, c) = \big(\Omega(s, c)\langle a_1, ..., a_k\rangle, (s_a)_{a\in\Delta}\big)$ whenever $\delta(s, c, a) = s_a$ holds for all $a \in \Delta$. It is immediate to see that the output of M on T is exactly T'.

As for regular tree morphisms, let σ be a regular tree morphism specified by an m-tuple $B_{c_1}, ..., B_{c_m}$ of replacing terms. We define the tree transducer $M = (\{s_0\}, \Sigma, \Delta, \Lambda, \lambda, \delta', s_0)$ such that $\Delta = \Lambda$, $\lambda(a) = a$ for every $a \in \Delta$, and $\delta'(s_0, c) = \big(B_c, (s_0)_{a\in\Delta}\big)$. Clearly, the output of M on T is exactly the tree $\sigma(T)$.

It remains to show that any tree transducer can be characterized in terms of finite-state recolorings and regular tree morphisms. Let $M = (S, \Sigma, \Delta, \Lambda, \lambda, \delta, s_0)$ be a tree transducer generating T' on a given input tree T. Without loss of generality, we can assume that $\Delta = \Lambda = \{a_1, ..., a_k\}$ (if this were not the case, simply replace both sets Λ and Δ by $\Lambda \cup \Delta$ and accordingly extend the functions λ and δ of the tree transducer M). Let σ_1 be the regular tree morphism specified by the tuple $(B_c)_{c\in\Sigma}$, where $B_c = c\langle\lambda(a_1), ..., \lambda(a_k)\rangle$. Intuitively, σ_1 transforms T by making $|\lambda^{-1}(a)|$ copies of each a-labeled edge. Clearly, we have that (i) for every $v \in \Lambda^*$, $v \in \mathcal{D}om(\sigma_1(T))$ iff $\lambda(v) \in \mathcal{D}om(T)$ (here λ is naturally extended to sequences of labels by letting $\lambda(v) = \lambda(v[1])\lambda(v[2])...\lambda(v[|v|])$) and (ii) for every $v \in \mathcal{D}om(\sigma_1(T))$, $\sigma_1(T)(v) = T(\lambda(v))$. This means that the tree transducer $M' = (S, \Sigma, \Lambda, \Lambda, id, \delta, s_0)$, where $id(a) = a$ for every $a \in \Lambda$, generates exactly T' given $\sigma_1(T)$ as input. Then, we can define a Mealy tree automaton $N_2 = (S, \Sigma \cup (S \times \Sigma), \Lambda, \delta_2, s_0, \Omega)$ such that, for every $c \in \Sigma$, $\Omega(s, c) = (s, c)$ and $\forall\, a \in \Lambda.\ \delta_2(s, c, a) = s_a$ iff $\delta(s, c) = \big(B, (s_a)_{a\in\Lambda}\big)$ holds for some $B \in \mathcal{T}_\Lambda^{reg}$ (note that the values of Ω and δ for colors $c \in S \times \Sigma$ are not relevant). The role of N_2 is to mark each vertex of $\sigma_1(T)$ with the corresponding state of the run of M'. Finally, by viewing N_2 as a transformation on trees, one can verify that

$$M^\omega(s_0, T) = (M')^\omega\big(s_0, \sigma_1(T)\big) = N_2\big(\sigma_1(T)\big)\big[\![B_{s,c}/(s, c)]\!\big]_{(s,c)\in S\times\Sigma} = \sigma_3\big(N_2(\sigma_1(T))\big),$$

where σ_3 is the regular tree morphism specified by $(B_{s,c})_{(s,c)\in S\times\Sigma}$ and $B_{s,c}$ is the (unique) tree such that $\delta(s, c) = \big(B_{s,c}, (s_a)_{a\in\Lambda}\big)$. This completes the proof of the proposition. ■

As usual, we can allow the input of a tree transducer (with bounded, rational lookahead) $M = (S, \Sigma, \Delta, \Lambda, \lambda, \delta, s_0)$ to be a (Σ-colored) Δ'-augmented tree, where Δ' is any finite set disjoint from Σ. In such a case, we assume that the transition function δ maps a pair $(s, c) \in S \times (\Sigma \cup \Delta)$ (resp., a pair $(s, T') \in S \times \mathcal{T}_{\Delta'}^D$, a pair $(s, T') \in S \times \mathcal{T}_{\Delta'}$) either to an element of $\mathcal{T}_\Delta^{reg} \times S^\Delta$ or to the singleton tree c, depending on whether $c \in \Sigma$ or $c \in \Delta'$ (resp., $T'(\varepsilon) = c \in \Delta'$). From such an assumption it follows that the output of a tree transducer (with bounded, rational lookahead) on a (Σ-colored) Δ'-augmented tree is a (Σ-colored) Δ'-augmented tree as well.

Summing up, in Figure 3.2 we graphically represent the relationships between the tree transformations introduced so far (we do not consider the operations of explicit construction and regular tree insertion, since these are intrinsically incomparable with the others). 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.

3.3 An automaton-based approach to decidability

In this section we develop an automaton-based method to deal with the decidability of MSO theories of deterministic colored trees. Here we focus on main results

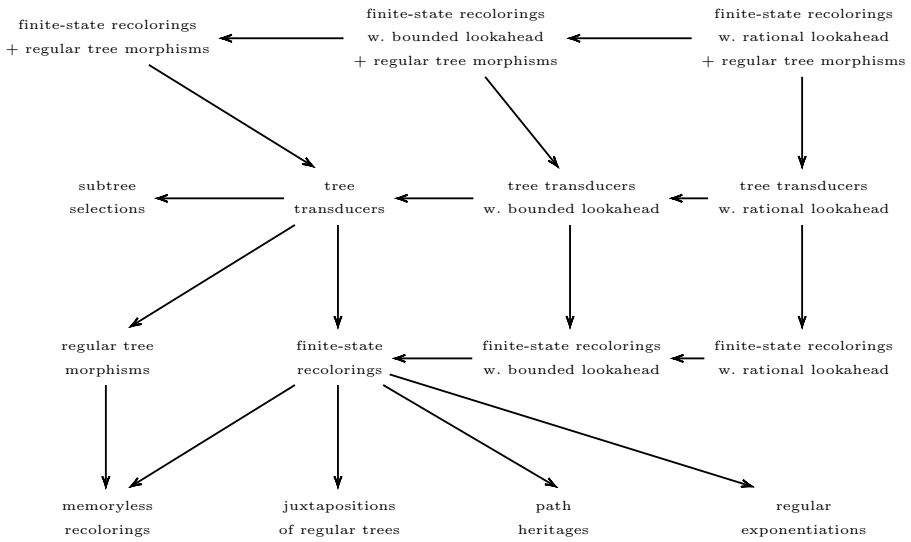

Figure 3.2: Some relationships between tree transformations.

only, temporarily overlooking the problem of finitely representing the instances of the problem. The method we are going to describe can be viewed as a generalization of Carton and Thomas's approach, which exploits noticeable properties of an 'indistinguishability' relation for Büchi automata to decide MSO theories of labeled linear orderings [19, 20]. The method refines and extends previous result of Montanari and Puppis [82, 83, 84].

As a preliminary step, we reduce the model checking problem for MSO logic to the acceptance problem for tree automata. Rabin's Theorem [96] establishes a strong correspondence between MSO formulas satisfied by an expanded tree structure $(T, \bar{V})$ and Rabin/Muller tree automata accepting its canonical representation $T_{\bar{V}}$: for every formula $\varphi(\bar{X})$, one can compute a tree automaton M (and, conversely, for every tree automaton M, one can compute a formula $\varphi(\bar{X})$) such that

$$T \vDash \varphi[\bar{V}] \quad \text{iff} \quad T_{\bar{V}} \in \mathscr{L}(M).$$

Let us call *acceptance problem* of a tree $T_{\bar{V}}$, denoted $Acc(T_{\bar{V}})$, the problem of deciding, for any given tree automaton M, whether M accepts $T_{\bar{V}}$. We have that

$$MTh(T, \bar{V}) \text{ is decidable} \quad \text{iff} \quad Acc(T_{\bar{V}}) \text{ is decidable.}$$

We finitely represent a *regular* tree $T_{\bar{V}}$ as a finite colored graph G together with a designated source vertex v_0 such that the unfolding of G from v_0 is isomorphic to $T_{\bar{V}}$. We call the pair (G, v_0) a *rooted graph* and we give it the status of a *representation* of $T_{\bar{V}}$. The following proposition implies that the problem $Acc(T_{\bar{V}})$ is decidable, given a representation of the regular tree $T_{\bar{V}}$. For the sake of simplicity, hereafter we shall

omit the subscript $\bar{V}$, thus writing T for $T_{\bar{V}}$, and we shall assume that a regular tree T is always represented by a finite rooted graph.

Proposition 3.3.1 *Given (a representation of) a regular tree T and an automaton M running on T, one can compute an input-free tree automaton M' such that $\mathscr{L}(M') \neq \emptyset$ iff $T \in \mathscr{L}(M)$.*

Proof. Let (G, v_0) be a representation of a regular tree T, where G is a graph $(V, (E_a)_{a\in\Lambda}, C)$ and $v_0 \in \mathcal{D}om(G)$. Without loss of generality, we can assume that T is an infinite complete $\Sigma \cup \{\bot\}$-colored tree (if this is not the case, simply append infinite complete $\bot$-colored trees to the leaves of T). Given a tree automaton $M = (S, \Sigma \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$, we define an input-free tree automaton M' such that $\mathscr{L}(M') \neq \emptyset$ iff $T \in \mathscr{L}(M)$. The automaton M' is of the form $(S', \emptyset, \Lambda, \delta', \mathcal{I}', \mathcal{F}')$, where

- $S' = S \times V$,
- $\big((s, v), (s_a, v_a)\big) \in \delta'$ iff $(s, C(v), (s_a)_{a\in\Lambda}) \in \delta$, and, for all $a \in \Lambda$, $(v, v_a) \in E_a$,
- $\mathcal{I}' = \mathcal{I} \times \{v_0\}$,
- $\mathcal{F}'$ consists of all sets $F' \subseteq S'$ such that $\mathcal{P}rj_1(F') \in \mathcal{F}$.

It is easy to verify that the runs of M' are all and only the S'-colored trees R' such that $\mathcal{P}rj_1(R')$ is a run of M on the tree T, where $\mathcal{P}rj_1(R')$ denotes the S-colored tree defined by $\mathcal{P}rj_1(R')(v) = \mathcal{P}rj_1(R'(v))$, for every $v \in \mathcal{D}om(\mathcal{P}rj_1(R'))$. Moreover a run R' of M' is successful iff $\mathcal{P}rj_1(R')$ is successful for M. This proves that $\mathscr{L}(M') \neq \emptyset$ iff $T \in \mathscr{L}(M)$. ■

Corollary 3.3.2 *Given (a representation of) a regular tree T and a tree automaton M running on T, the acceptance problem $T \in \mathscr{L}(M)$ is decidable.*

Proof. This follows trivially from the Proposition 3.3.1 and from the decidability of the emptiness problem for tree automata [96]. ■

3.3.1 Tree indistinguishability

Here we introduce the basic ingredients of the automaton-based approach to the decidability of MSO theories of trees. Like every finite-state machine, a tree automaton M is able to store only a finite (bounded) amount of information during its computation. This allows us to 'distill' some relevant features from trees and then to reason on the equivalence classes that result from such an abstraction. It is worth pointing out that the results provided in this section can be easily tailored to different kinds of automata, such as, for instance, Rabin tree automata and parity tree automata. We start with some preliminary definitions.

Definition 3.3.3 *Given a non-empty full tree T, a* partial run *of a tree automaton $M = (S, \Sigma \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$ on T is any (non-empty full) S-colored tree P such that $\mathcal{D}om(P) = \mathcal{D}om(T)$ and for every internal vertex v of P, $\big(P(v), T(v), (P(v{\cdot}a))_{a\in\Lambda}\big) \in \delta$, where $v \cdot a$ denotes the a-successor of v in P.*

Note that only the colors of the internal nodes of T are involved in the definition of partial run. Moreover, note that if T is an infinite complete tree, then the notion of partial run is equivalent to the standard notion of run of a tree automaton.

We now define features of partial runs. These are data structures collecting the relevant information of a partial run of a tree automaton M on a tree T.

Definition 3.3.4 *Given an automaton $M = (S, \Sigma \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$, a non-empty full tree T, and a partial run P of M on T, the* feature *of (T, P), denoted $[T, P]$, is the triple*

$$\begin{pmatrix} P(\varepsilon) \\ \big\{(T(v), P(v), \mathcal{I}mg(P|\pi_v)) \,:\, v \in \mathcal{F}r(P)\big\} \\ \big\{\mathcal{I}nf(P|\pi) \,:\, \pi \in \mathcal{B}ch(P)\big\} \end{pmatrix}$$

where π_v denotes the access path of a leaf v in P.

Roughly speaking, the above definition accounts for the occurrences of states along finite and infinite branches of the partial run P. In particular, the first component of the feature $[T, P]$ identifies the state at the root of the partial run P, the second component identifies, for each leaf v of T, the color of v, the state at v, and the set of states that occur at least once along the access path of v, and the third component identifies, for each infinite path π in T, the set of states that occur infinitely often along π.

In order to generalize the notions of partial run and feature to the cases of empty, non-full, and/or Δ-augmented trees, we introduce a suitable operation that extends a tree with $\bot$-colored vertices. The Δ-*completion* of a (Δ-augmented) tree T is the tree T_Δ defined as follows:

- if T is the empty tree, then T_Δ is the infinite complete $\bot$-colored tree,
- if T is a non-empty tree, then we set

$$\mathcal{D}om(T_\Delta) = \mathcal{D}om(T) \cup (F \cdot \Lambda^*) \cup (G \cdot \Lambda^*),$$

$$T_\Delta(v) = \begin{cases} T(v) & \text{if } v \in \mathcal{D}om(T), \\ \bot & \text{if } v \in (F \cdot \Lambda^*) \cup (G \cdot \Lambda^*), \end{cases}$$

where

$$F = \big\{v \cdot a \,:\, v \in \mathcal{F}r(T), T(v) \in \Sigma, a \in \Lambda\big\},$$
$$G = \big\{v \cdot a \,:\, v \in \mathcal{D}om(T) \setminus \mathcal{F}r(T), a \in \Lambda, v \cdot a \notin \mathcal{D}om(T)\big\}.$$

Intuitively, T_Δ is obtained from T by appending infinite complete $\bot$-colored trees to every Σ-colored leaf and to every missing successor of an internal vertex. Notice that T_Δ is a non-empty full Δ-augmented tree, whose leaves are all and only the Δ-colored leaves of T (if any). Moreover, the $\emptyset$-completion of a tree T is an infinite complete tree. This allows us to apply the notions of partial run and feature to any given (Δ-augmented) tree via the operation of Δ-completion.

Given a Δ-augmented tree T and a tree automaton M, in order to decide whether $T \in \mathscr{L}(M)$ we define the M, Δ-types of T as collections of features of the form $[T_\Delta, P]$,

where P ranges over a suitable set $\mathcal{P}$ of partial runs of M on T_Δ (different choices of $\mathcal{P}$ may result into different M,Δ-types of T). We allow $\mathcal{P}$ to be a *proper subset* of all partial runs of M on T_Δ, because there can be partial runs which are redundant with respect to others and thus can be 'forgotten'. The notion of *redundant* partial run is defined as follows. Given a Δ-augmented tree T and a tree automaton M, we define a binary relation $\preceq$ over the set of all partial runs of M on T_Δ such that P' is redundant with respect to P iff $P \preceq P'$. For every pair of partial runs P, P' of M on T_Δ, we have $P \preceq P'$ iff

- $P(\varepsilon) = P'(\varepsilon)$,
- for every leaf v of P, there is a leaf v' in P' such that $T(v) = T(v')$, $P(v) = P'(v')$, and $\mathit{Img}(P|\pi_v) = \mathit{Img}(P'|\pi_{v'})$, where π_v (resp., $\pi_{v'}$) denotes the access path of v (resp., of v'),
- for every infinite path τ in P, there is an infinite path τ' in P' such that $\mathit{Inf}(P|\tau) = \mathit{Inf}(P'|\tau')$.

Note that $\preceq$ satisfies reflexivity and transitivity, but it does not satisfy antisymmetry. Hence $\preceq$ can be given the status of *quasi-order*. Given a set $\mathcal{P}$ of partial runs of M on T_Δ, we say that $\mathcal{P}$ is *complete* if for every partial run P' of M on T_Δ, there is $P \in \mathcal{P}$ such that $P \preceq P'$.

Definition 3.3.5 *Given a tree automaton $M = (S, \Sigma \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$ and a Δ-augmented tree T, we call M,Δ-type of T any set consisting of features of the form $[T_\Delta, P]$, where P ranges over a complete set of partial runs of M on T_Δ. The* basic *M,Δ-type of T is the (unique) set of features of the form $[T_\Delta, P]$, where P ranges over all partial runs of M on T_Δ.*

We denote by $\mathscr{T}_{M,\Delta}$ the set of all possible M,Δ-types of (Δ-augmented) trees. Since $\mathscr{T}_{M,\Delta}$ is included in the *finite* set $\mathscr{P}\big(S \times \mathscr{P}(\Delta \times S \times \mathscr{P}(S)) \times \mathscr{P}(\mathscr{P}(S))\big)$, there exist only finitely many M,Δ-types of trees, for any given choice of M and Δ.

It turns out that trees having a common M,Δ-type are, in some sense, 'indistinguishable' by the automaton M. In particular, given an $M,\emptyset$-type of a Σ-colored tree T (which is computable from an M,Δ-type of T, for any set Δ), one can decide whether M accepts T. A converse result also holds, stating that given any tree automaton M with input alphabet $\Sigma \cup \{\bot\}$ and given any set Δ disjoint from Σ, if T is a Δ-augmented tree and $\mathit{Acc}(T)$ is decidable, then one can compute an M,Δ-type (i.e., the basic M,Δ-type) of T.

Lemma 3.3.6 *Given a tree automaton M running on an infinite complete tree T and given a complete set $\mathcal{R}$ of runs of M on T, for every successful run R' of M and for every run $R \in \mathcal{R}$ such that $R \preceq R'$, R is a successful run as well.*

Proof. Let $M = (S, \Sigma \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$, let R' be a successful run of M on T and let $R \in \mathcal{R}$ satisfy $R \preceq R'$. By definition, we know that (i) $R(\varepsilon) = R'(\varepsilon)$ and (ii) for every infinite path τ in R, there is another infinite path τ' in R' such that $\mathit{Inf}(R|\tau) = \mathit{Inf}(R'|\tau')$. Property (i) implies that $R(\varepsilon) \in \mathcal{I}$. Property (ii) implies that, for every infinite path τ in R, $\mathit{Inf}(R|\tau) \in \mathcal{F}$. This shows that R is a successful run of M on T. ∎

Proposition 3.3.7 *Given a tree automaton M running on a Σ-colored tree T and given an $M,\emptyset$-type of T, one can decide whether $T \in \mathscr{L}(M)$.*

Proof. Let $M = (S, \Sigma \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$ be a tree automaton and let t be an $M,\emptyset$-type of the tree T, which we write as follows:

$$\left\{ \begin{pmatrix} P(\varepsilon) \\ \big\{(T(v), P(v), \mathcal{I}mg(P|\pi_v) \,:\, v \in \mathcal{F}r(P)\big\} \\ \big\{\mathcal{I}nf(P|\pi) \,:\, \pi \in \mathcal{B}ch(P)\big\} \end{pmatrix} \,:\, P \in \mathcal{P} \right\}$$

where π_v denotes the access path of v in P and $\mathcal{P}$ is a complete set of partial runs of M on $T_\emptyset$. Recall that the $\emptyset$-completion $T_\emptyset$ of T is an infinite complete Σ-colored tree, hence the second component in each tuple of t is empty. In particular, this means that every partial run $P \in \mathcal{P}$ is also a run of M on $T_\emptyset$. Clearly, P is a successful run provided that $P(\varepsilon) \in \mathcal{I}$ and $\mathcal{I}nf(P|\pi) \in \mathcal{F}$ for every infinite path π in P. Conversely, if R is a successful run of M on $T_\emptyset$, then there is a run $P \in \mathcal{P}$ such that $P \preceq R$ and, by Lemma 3.3.6, P is a successful run as well. This implies that M accepts T iff t contains a triple of the form $\big(s, \{(c_i, q_i, Q_i)\}_{i\in I}, \{W_j\}_{j\in J}\big)$, with $s \in \mathcal{I}$ and, for all $j \in J$, $W_{h,j} \in \mathcal{F}$. Such a condition can be effectively tested given the $M,\emptyset$-type t. ∎

Proposition 3.3.8 *Given a tree automaton M with input alphabet $\Sigma \cup \{\bot\}$ and given any finite set Δ disjoint from Σ, if T is a Δ-augmented tree and $Acc(T)$ is decidable, then one can compute the basic M,Δ-type of T.*

Proof. Let $M = (S, \Sigma \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$, let Δ be a finite set disjoint from Σ, and let T be a Δ-augmented tree. Without loss of generality, we can assume that the set of states of M is $S = \{1, ..., n\}$ and that no vertex in T is colored with $\bot$. We then denote by T_Δ the Δ-completion of T. We compute the basic M,Δ-type of T by evaluating the truth value of suitable MSO sentences (which are defined on the grounds of M) over the tree T_Δ. These sentences are used to 'query' the features of T, that is to compute all and only the tuples $\big(s_h, \{(c_i, q_{h,i}, Q_{h,i})\}_{i\in I}, \{W_{h,j}\}_{j\in J}\big)$ belonging to the basic M,Δ-type of T. For the sake of simplicity, we shall use ε to denote the (MSO-definable) root of T_Δ, E_a to denote the relational symbol for the a-successor relation of T_Δ, R_c to denote the relational symbol for the set of c-colored vertices of T_Δ, with $c \in \Sigma \cup \Delta \cup \{\bot\}$, and some variables $X_1, ..., X_n$ (shortly denoted $\bar{X}$) to encode the state of the automaton M at each vertex. Moreover, we shall use the following basic formulas (here we give an intuitive definition rather than a formal one):

- $\varphi_{prun}(\bar{X})$, which holds in T_Δ iff the interpretation of $\bar{X}$ encodes a partial run of M on T_Δ,
- $\varphi_{finpath}(y, Y)$, which holds in T_Δ iff y is interpreted as a leaf and Y is interpreted as the access path to y in T_Δ,
- $\varphi_{infpath}(Y)$, which holds in T_Δ iff Y is interpreted as an infinite path in T_Δ,
- $\varphi_{suffix}(Y, Z)$, which holds in T_Δ iff, whenever Y is interpreted as an infinite path, Z is interpreted as one of its suffixes.

Now, we choose a generic tuple $t = \big(s, \{(b_i, q_i, Q_i)\}_{i\in I}, \{W_j\}_{j\in J}\big)$ as a candidate element of the basic M, Δ-type of T (note that there are only finitely many of such tuples) and we build the sentence Ψ_t:

$$\begin{aligned}\Psi_t = \exists\, \bar{X}.\, \Big(&\varphi_{prun}(\bar{X}) \quad \wedge \quad \varepsilon \in X_s \quad \wedge \\ &\forall\, y, Y.\, \varphi_{finpath}(y, Y) \;\rightarrow\; \bigvee_{i\in I} \varphi_{finmatch,i}(\bar{X}, y, Y) \quad \wedge \\ &\bigwedge_{i\in I} \exists\, y, Y.\, \varphi_{finpath}(y, Y) \;\wedge\; \varphi_{finmatch,i}(\bar{X}, y, Y) \quad \wedge \\ &\forall\, Y.\, \varphi_{infpath}(Y) \;\rightarrow\; \bigvee_{i\in J} \varphi_{infmatch,j}(\bar{X}, Y) \quad \wedge \\ &\bigwedge_{j\in J} \exists\, Y.\, \varphi_{infpath}(Y) \;\wedge\; \varphi_{infmatch,j}(\bar{X}, Y)\Big)\end{aligned}$$

where $$\varphi_{finmatch,i}(\bar{X}, y, Y) \;=\; y \in R_{c_i} \quad \wedge \quad y \in X_{q_i} \quad \wedge \bigwedge_{r\in Q_i} X_r \cap Y \neq \emptyset \quad \wedge \quad \bigwedge_{r\notin Q_i} X_r \cap Y = \emptyset$$

and $$\varphi_{infmatch,j}(\bar{X}, Y) \;=\; \bigwedge_{r\in W_j} \forall\, Z.\, \varphi_{suffix}(Y, Z) \;\rightarrow\; X_b \cap Z \neq \emptyset \quad \wedge \bigwedge_{r\notin W_j} \exists\, Z.\, \varphi_{suffix}(Y, Z) \;\wedge\; X_b \cap Z = \emptyset.$$

It is routine to check that T_Δ is a model of Ψ_t iff t belongs to the basic M, Δ-type of T. Therefore, by exploiting Rabin's theorem, one can transform Ψ_t into a tree automaton M_t over the alphabet $\Sigma \cup \Delta \cup \{\bot\}$ such that $T_\Delta \in \mathscr{L}(M_t)$ iff $T_\Delta \vDash \Psi_t$. Notice that $T_\Delta \in \mathscr{L}(M_t)$ iff $T \in \mathscr{L}(M_t)$ and hence, being $Acc(T)$ decidable, one can effectively build the basic M, Δ-type of T. ■

Corollary 3.3.9 *Given a tree automaton M, a finite set Δ disjoint from Σ, and a regular Δ-augmented tree T, one can compute the basic M, Δ-type of T.*

Proof. This is an immediate consequence of Corollary 3.3.2 and Proposition 3.3.8. ■

Proposition 3.3.10 *Let Δ be a finite set disjoint from Σ, T a Δ-augmented tree, $\Delta' \subseteq \Delta$, and M a tree automaton with input alphabet $\Sigma \cup (\Delta \setminus \Delta') \cup \{\bot\}$. Then, given an M, Δ-type of T, one can compute an M, Δ'-type of T.*

Proof. Let $M = (S, \Sigma \cup (\Delta \setminus \Delta') \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$. First of all, for every $b \in \Delta \setminus \Delta'$, we denote by $\mathcal{P}_b$ the set consisting of all runs of M on the infinite complete tree T_b, defined by $T_b(\varepsilon) = b$ and $T_b(v) = \bot$ for every $v \in \Lambda^+$. Clearly, the tree T and its Δ-completion T_Δ have the same M, Δ-types. Similarly, the tree T and its Δ'-completion

$T_{\Delta'}$ have the same M, Δ'-types. Moreover, since $\Delta' \subseteq \Delta$, we have that $T_{\Delta'}$ can be obtained from T_Δ by substituting each b-colored leaf by T_b, for $b \in \Delta \setminus \Delta'$ (formally, $T_{\Delta'} = T_\Delta[\![T_b/b]\!]_{b \in \Delta \setminus \Delta'}$). Hence, the domain of T_Δ is included in the domain of $T_{\Delta'}$.

Now, we consider two cases: in the first case, we are given a partial run P' of M on $T_{\Delta'}$ and we have to build a suitable partial run P on T_Δ, in the second case, we are given a partial run P of M on T_Δ and we have to build a suitable partial run P' on $T_{\Delta'}$. As for the first case, let P' be a partial run of M on $T_{\Delta'}$. We define the corresponding partial run P on T_Δ by restricting P' to $\mathcal{D}om(T_\Delta)$, namely, we let $P = P|_{\mathcal{D}om(T_\Delta)}$. Note that, in such a case, for every b-colored leaf v of T_Δ, with $b \in \Delta \setminus \Delta'$, the subtree of P' rooted at v is isomorphic to some run belonging to $\mathcal{P}_b$. As for the second case, let P be a partial run of M on T_Δ. If for every b-colored leaf v of P, with $b \in \Delta \setminus \Delta'$, $\mathcal{P}_b$ contains a run P_v such that $P_v(\varepsilon) = P(v)$, then we can define a partial run P' of M on $T_{\Delta'}$ by letting $\mathcal{D}om(P') = \mathcal{D}om(T_{\Delta'})$, $P'(v) = P(v)$ for every $v \in \mathcal{D}om(P)$, and $P'(v \cdot w) = P_v(w)$ for every $w \in \Lambda^+$ and for every $\Delta \setminus \Delta'$-colored leaf v of P. Note that in such a case, $P = P'|_{\mathcal{D}om(T_\Delta)}$ holds (namely, P is the restriction of P' to $\mathcal{D}om(T_\Delta)$) and for every b-colored leaf v of T_Δ, with $b \in \Delta \setminus \Delta'$, the subtree of P' rooted at v is isomorphic to some run belonging to $\mathcal{P}_b$. Otherwise, if there is a b-colored leaf v in P, with $b \in \Delta \setminus \Delta'$, for which no partial run $P_v \in \mathcal{P}_b$ exists such that $P_v(\varepsilon) = P(v)$, then it is easy to see that no partial run P' on $T_{\Delta'}$ satisfying $P = P'|_{\mathcal{D}om(T_\Delta)}$ exists. By exploiting the above constructions, we can claim that in each of the two cases:

i) $P(\varepsilon) = P'(\varepsilon)$,
ii) for every leaf v in P', $T_{\Delta'}(v) \in \Delta'$ and hence v is also a leaf of T_Δ having the same color; moreover, $P(v) = P'(v)$ and $\mathcal{I}mg(P|\pi) = \mathcal{I}mg(P'|\pi)$, where π denotes the access path of v in P and in P';
iii) for every infinite path π in P', either π is an infinite path in P, hence $\mathcal{I}nf(P|\pi) = \mathcal{I}nf(P'|\pi)$, or there is a b-colored vertex v along π in P' such that $b \in \Delta \setminus \Delta'$ and v is a b-colored leaf of P; in the latter case we know that there is a partial run $P_v \in \mathcal{P}_b$ isomorphic to the subtree of P' rooted at v, hence there is an infinite path τ in P_v (corresponding to the suffix of π starting from v) such that $\mathcal{I}nf(P_v|\tau) = \mathcal{I}nf(P'|\pi)$;
iv) for every leaf v in P, either $T_\Delta(v) \in \Delta'$, which implies that v is a leaf of $T_{\Delta'}$ having the same color and satisfying both equalities $P'(v) = P(v)$ and $\mathcal{I}mg(P|\pi) = \mathcal{I}mg(P'|\pi)$ (where π denotes the access path of v in P and in P'), or $T_\Delta(v) = b \in \Delta \setminus \Delta'$, which implies that there is a partial run $P_v \in \mathcal{P}_b$ isomorphic to the subtree of P' rooted at v and for every infinite path τ in P_v, there is an infinite path π in P' (obtained by concatenating τ to the access path of v) such that $\mathcal{I}nf(P'|\pi) = \mathcal{I}nf(P_v|\tau)$;
v) for every infinite path π in P, π is also an infinite path in P' and hence $\mathcal{I}nf(P'|\pi) = \mathcal{I}nf(P|\pi)$.

This shows that, given an M, Δ'-type t_b of T_b, for each $b \in \Delta \setminus \Delta'$, and given an

M, Δ-type t of T of the form

$$\left\{ \begin{pmatrix} s_h \\ \{(b_i, q_{h,i}, Q_{h,i}) : i \in I\} \\ \{W_{h,j} : j \in J\} \end{pmatrix} : h \in H \right\}$$

there is an M, Δ'-type t' of T consisting of all the triples of the form

$$\big(p, \{(d_{i'}, r_{i'}, R_{i'})\}_{i' \in I'}, \{V_{j'}\}_{j' \in J'}\big)$$

for which there is $h \in H$ such that

- $s_h = p$,
- for all $i' \in I'$, there is $i \in I$ satisfying $d_{i'} = b_i$, $r_{i'} = q_{h,i}$, $R_{i'} = Q_{h,i}$,
- for all $i \in I$, if $b_i \in \Delta'$, then there is $i' \in I'$ satisfying $d_{i'} = b_i$, $r_{i'} = q_{h,i}$, $R_{i'} = Q_{h,i}$,
- for all $i \in I$, if $b_i \in \Delta \setminus \Delta'$, then there is a tuple $\big(x, \emptyset, \{X_{j''}\}_{j'' \in J''}\big)$ in t_{b_i} satisfying $x = q_{h,i}$ and $\forall\, j'' \in J''.\ \exists\, j' \in J'.\ X_{j''} = V_{j'}$,
- for all $j' \in J'$, either there is $j \in J$ satisfying $V_{j'} = W_{h,j}$, or there is $i \in I$, with $b_i \in \Delta \setminus \Delta'$, and there is a tuple $\big(x, \emptyset, \{X_{j''}\}_{j'' \in J''}\big)$ in t_{b_i} such that $x = q_{h,i}$ and $\exists\, j'' \in J''.\ X_{j''} = V_{j'}$;
- for all $j \in J$, there is $j' \in J'$ satisfying $V_{j'} = W_j$.

Moreover, for every $b \in \Delta \setminus \Delta'$, the basic M, Δ'-type t_b is computable in virtue of Corollary 3.3.9 (since T_b is a regular tree) and this completes the proof. ∎

Summing up, in this section we introduced the notion of partial run, feature, and type of a tree with respect to a given tree automaton. Then, we proved that the model checking problem, the acceptance problem, and the problem of computing types are inter-reducible. Formally, if we denote by *Types*(T) the problem of computing, for any given tree automaton M and any given finite set Δ, an M, Δ-type of T, then we have

$$\mathit{MTh}(T) \text{ decidable} \quad \text{iff} \quad \mathit{Acc}(T) \text{ decidable} \quad \text{iff} \quad \mathit{Types}(T) \text{ solvable.}$$

3.3.2 From trees to their retractions

We now show how M, Δ-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 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 tree retractions, which are tree-shaped arrangements of M, Δ-types corresponding to the factors of a tree. Then we prove that the acceptance problem for a tree T can be reduced to the acceptance problem for a retraction of it.

Definition 3.3.11 *Given a tree T, a* factorization *of T with respect to a finite set Δ is a (possibly non-deterministic) Δ-labeled uncolored tree Π such that*

- $\mathcal{D}om(\Pi) \subseteq \mathcal{D}om(T)$,
- $\varepsilon \in \mathcal{D}om(\Pi)$,
- *for every pair of distinct vertices* u, u' *of* Π, (u, u') *is an edge of* Π *iff* u *and* u' *are distinct vertices of* Π *such that* u' *is a descendant of* u *in* T *and no other vertex* $u'' \in \mathcal{D}om(\Pi)$ *exists along the path from* u *to* u' *in* T,
- *the edge labels are arbitrarily chosen from the set* Δ.

We can graphically represent a factorization of a tree by first identifying its vertices (look at those nodes on the left-hand side tree of Figure 3.3 which are surrounded by gray circles) and then drawing the resulting edges, together with the corresponding labels (look at the right-hand side tree of Figure 3.3).

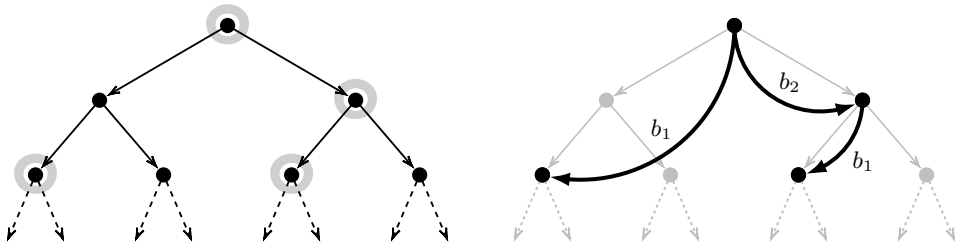

Figure 3.3: An example of factorization.

We now define the (marked) factors of the tree T with respect to the factorization Π. Let $u \in \mathcal{D}om(\Pi)$ and $Succ(u) = \{u' : (u, u') \text{ is an edge of } \Pi\}$. The *unmarked factor* of T in u, denoted $T_\Pi[u]$, is the tree obtained by selecting the vertices of $T_\emptyset$ (i.e. the $\emptyset$-completion of T, note that Π is also a factorization of $T_\emptyset$) which are descendants (in $T_\emptyset$) of u, but not proper descendants (in $T_\emptyset$) of any $u' \in Succ(u)$. Thus, the root of a factor $T_\Pi[u]$ is u and the leaves are the successors of u in Π. For every vertex $u \neq \varepsilon$ in Π, there is a unique label $b \in \Delta$ such that u is the *target* of a b-labeled edge in Π; we call such a label the *marker* of the vertex u and we denote it by $m_\Pi[u]$. We then define the *marked factor* of T in u, denoted $T^+_\Pi[u]$, as the tree obtained from $T_\Pi[u]$ by recoloring each leaf u' with the corresponding marker $m_\Pi[u']$. The notions of unmarked factor and marked factor can be easily generalized to the case of a Δ'-augmented tree. Precisely, if Π is a factorization of a Δ'-augmented tree T with respect to Δ, then we assume that $\Delta' = \Delta$ and we define $T_\Pi[u]$ as the tree obtained by selecting the vertices of T_Δ which are descendants (in T_Δ) of u, but not proper descendants (in $T_\emptyset$) of any $u' \in Succ(u)$. Accordingly, the marked factor $T^+_\Pi[u]$ of T with respect to Π is the tree obtained from $T_\Pi[u]$ by recoloring each leaf u' with the corresponding marker $m_\Pi[u']$. Notice that a marked factor of T is always a non-empty full Δ-augmented tree whose leaves are colored over Δ. This allows us to define the M, Δ-types of the marked factors of T.

Definition 3.3.12 *Given a (Δ-augmented) tree T, a tree automaton M, and a factorization Π of T with respect to Δ, a* retraction *of T with respect to M and Π is any Δ-labeled $\mathscr{T}_{M,\Delta}$-colored tree C such that*

- $\mathcal{D}om(C) = \mathcal{D}om(\Pi)$,
- (u, u') *is a* b*-labeled edge in* C *iff* (u, u') *is a* b*-labeled edge in* Π,
- *each vertex* u *in* C *is colored with an* M, Δ*-type of the corresponding marked factor* $T^+_\Pi[u]$.

In general, a retraction C, as well as a factorization Π, of a tree may be non-deterministic, possibly with vertices of unbounded (or even infinite) out-degree. Since tree automata operate on *deterministic* trees, we restrict ourselves to retractions which are *bisimilar to deterministic trees*. In this perspective, we can think of a retraction C as a deterministic (Δ-labeled) tree by collapsing, for every label $b \in \Delta$, the isomorphic subtrees rooted at all b-successors $u_1, ..., u_h$ of a vertex u in C. Moreover, by definition, retractions depend on automata, but, as a matter of fact, for all the considered tree structures, we shall provide a single factorization from which we will be able to generate a suitable treaction for any tree automaton.

We now show how, given a tree T, a tree automaton M, and a factorization Π of T with respect to a set Δ, one can build a suitable tree automaton M^Δ such that M accepts (the $\emptyset$-completion of) T iff M^Δ accepts (the $\emptyset$-completion of) a retraction C of T with respect to M and Π. The automaton M^Δ mimics the behavior of M at a 'coarser' level and it can be effectively computed from M and Δ. Its input alphabet is the set $\mathscr{T}_{M,\Delta}$ of all M, Δ-types plus the additional symbol $\perp$ (which is read on the missing vertices of C). The states of M^Δ encode the finite information processed by M during its computations up to a certain point and its transitions compute new states which (possibly) extend information provided by the current state with information provided by the input symbol (the M, Δ-type of a marked factor). The automaton M^Δ is formally defined as follows.

Definition 3.3.13 *Given a Muller tree automaton* $M = (S, \Sigma \cup \{\perp\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$ *and a finite set* Δ *disjoint from* Σ*, we define the Muller tree automaton* $M^\Delta = (Z, \mathscr{T}_{M,\Delta} \cup \{\perp\}, \Delta, \delta', \mathcal{I}', \mathcal{F}')$ *as follows:*

- $Z = \Delta \times \big\{0, 1\big\} \times \mathscr{P}\big(S \times \mathscr{P}(S) \times \mathscr{P}(S)\big) \times \mathscr{P}\big(\mathscr{P}(S)\big)$,
- *for every state* $z = (c, x, \mathcal{U}, \mathcal{V})$ *and for every tuple of states* $(z_b)_{b\in\Delta}$*, we have* $\big(z, t, (z_b)_{b\in\Delta}\big) \in \delta'$ *iff for all* $b \in \Delta$*,* $z_b = (c, 1, \mathcal{U}, \mathcal{V})$,
- *for every* M, Δ*-type* $t = \Big\{\big(s_h, \{(b_i, q_{h,i}, Q_{h,i})\}_{i\in I}, \{W_{h,j}\}_{j\in J}\big) : h \in H\Big\} \in \mathscr{T}_{M,\Delta}$*, where* H, I, J *are suitable sets of indices, for every state* $z = (c, 0, \mathcal{U}, \mathcal{V})$*, where* $\mathcal{U} = \{(r_l, U_l, Y_l)\}_{l\in L}$ *and* $\mathcal{V} = \{V_g\}_{g\in G}$*, and for every tuple of states* $(z_b)_{b\in\Delta}$*, we have* $\big(z, t, (z_b)_{b\in\Delta}\big) \in \delta'$ *iff for all* $l \in L$*, there exists* $h_l \in H$ *such that (i)* $r_l = s_{h_l}$ *and (ii) for all* $b \in \Delta$*,* $z_b = (b, 0, \mathcal{U}_b, \mathcal{V}_b)$ *where*

$$\mathcal{U}_b = \bigcup_{l\in L} \big\{(q_{h_l,i}, Q_{h_l,i}, Y_l \cup Q_{h_l,i}) : i \in I, b_i = b\big\}$$
$$\mathcal{V}_b = \mathcal{V} \cup \bigcup_{l\in L} \big\{W_{h_l,j} : j \in J\big\},$$

- $\mathcal{I}'$ *consists of all states of the form* $(c, 0, \{(s, \emptyset, \emptyset)\}, \emptyset)$*, with* $c \in \Delta$ *and* $s \in \mathcal{I}$,

- $\mathcal{F}'$ *consists of all sets of states of the form* $\{(c_1, x, \mathcal{U}_1, \mathcal{V}), ..., (c_n, x, \mathcal{U}_n, \mathcal{V})\} \subseteq Z$ *such that*
  - *i) if* $x = 1$, *then we have* $n = 1$, $\mathcal{U}_1 = \{(r_l, U_l, Y_l)\}_{l \in L}$, *and for all* $l \in L$, *the* $\emptyset$*-completion of the singleton tree* c_1 *is accepted by* $M[\{r_l\}/\mathcal{I}]$, *where* $M[\{r_l\}/\mathcal{I}]$ *is the tree automaton obtained from* M *by substituting* $\{r_l\}$ *for the set of initial states* $\mathcal{I}$,
  - *ii) if* $x = 0$ *and for all* $k \in [n]$, $\mathcal{U}_k = \{(r_{k,l}, U_{k,l}, Y_k)\}_{l \in L_k}$, *where* L_k *is a suitable set of indices, then for all* $\emptyset \subsetneq L'_1 \subseteq L_1$, ..., $\emptyset \subsetneq L'_n \subseteq L_n$, *we have* $\bigcup_{k \in [n]} \bigcup_{l \in L'_k} U_{k,l} \in \mathcal{F}$,
  - *iii)* $\mathcal{V} \subseteq \mathcal{F}$.

The following theorem, which will be proved in the sequel, reduces the acceptance problem for T and M to that for C and M^{Δ}. The upshot of such a result is that, if we were able to compute, for any given automaton M running on a tree T, a retraction of T, and if such a retraction enjoyed a decidable acceptance problem, then the acceptance problem of T would be decidable as well. In succession, we shall report other implications of the theorem.

Theorem 3.3.14 *Given an automaton* M *running on a tree* T, *given a factorization* Π *of* T *with respect to a finite set* Δ, *and given a retraction* C *with respect to* M *and* Π, *we have*

$$T \in \mathscr{L}(M) \quad \textit{iff} \quad C \in \mathscr{L}(M^{\Delta}).$$

Corollary 3.3.15 *Given a tree* T, *if one can compute, for any given automaton* M, *(a representation of) a retraction* C *of* T *with respect to* M *such that* C *enjoys a decidable MSO theory (this happens, for instance, when* C *is a regular retraction), then* T *enjoys a decidable MSO theory as well.*

Proof. This follows trivially from Theorem 3.3.14 and from the correspondence between MSO formulas interpreted over tree structures and Rabin tree automata. ∎

Corollary 3.3.16 *Given an automaton* M *running on two trees* T_1 *and* T_2, *if* T_1 *and* T_2 *enjoy isomorphic retractions w.r.t* M, *then* $T_1 \in \mathscr{L}(M)$ *iff* $T_2 \in \mathscr{L}(M)$.

Proof. This follows from having defined the automaton for a retraction C only on the grounds of the automaton M and the set Δ of the labels of C. Indeed, if Π_1 (resp., Π_2) is a factorization of T_1 (resp., T_2) with respect to Δ_1 (resp., Δ_2), and if C_1 (resp., C_2) is a retraction of T_1 (resp., T_2) with respect to M and Π_1 (resp., Π_2), then we can assume $\Delta_1 = \Delta_2$ (if this is not the case, consider the M, Δ_1-types marking the vertices of C_1 and the M, Δ_2-types marking the vertices of C_2 and note that only elements from $\Delta_1 \cap \Delta_2$ can occur). Hence, we have that $M_{\Delta_1} = M_{\Delta_2}$ and from Theorem 3.3.14 it follows that $T_1 \in \mathscr{L}(M)$ iff $C_1 \in \mathscr{L}(M_{\Delta_1})$, iff $C_2 \in \mathscr{L}(M_{\Delta_2})$, iff $T_2 \in \mathscr{L}(M)$. ∎

The proof of Theorem 3.3.14 is rather difficult and we first need some technical definitions and results. We fix a finite set Δ disjoint from Σ, a Δ-augmented Σ-colored tree T, a tree automaton $M = (S, \Sigma \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$, a factorization Π of

T with respect to Δ, and a retraction C of T w.r.t M and Π. Moreover, we assume that

- T is identified with a function from $\mathcal{D}om(T)$ to $\Sigma \cup \Delta$, where $\mathcal{D}om(T) \subseteq \Lambda^*$ is a language closed under the prefix relation on words over Λ (for instance, given a vertex $v \in \mathcal{D}om(T)$ and a label $a \in \Lambda$, $v \cdot a$ denotes the a-successor of v in T, if exists);
- the domain of the factorization Π is included in the domain of T, but it is not necessarily closed under the prefix relation on Λ^*;
- given a vertex $u \in \mathcal{D}om(\Pi)$, the marked factor of T rooted at u with respect to Π is identified with a function from a subset of Λ^* to $\Sigma \cup \Delta$, whose domain is closed under the prefix relation on words (for instance, the root of a factor $T_\Pi[u]$ is the empty word ε and the leaves are all and only the shortest non-empty words $w \in \Lambda^*$ such that $u \cdot w \in \mathcal{D}om(\Pi)$);
- C is a deterministic Δ-labeled $\mathscr{T}_{M,\Delta}$-colored tree, identified with a function from $\mathcal{D}om(C)$ to $\mathscr{T}_{M,\Delta}$, where $\mathcal{D}om(C) \subseteq \Delta^*$ is a language closed under the prefix relation on words over Δ (for instance, given a vertex $u \in \mathcal{D}om(C)$ and a label $b \in \Delta$, $u \cdot b$ denotes the b-successor of u in C, if exists);
- $M^\Delta = (Z, \mathscr{T}_{M,\Delta} \cup \{\bot\}, \Delta, \delta', \mathcal{I}', \mathcal{F}')$ is the automaton running on C, specified as in Definition 3.3.13.

In order to simplify the notation, we shall not distinguish between the vertices of a tree and their access paths. For instance, given a vertex $u \in \mathcal{D}om(\Pi)$ and its access path π, we may use $T^+_\Pi[\pi]$, instead of $T^+_\Pi[u]$, to denote the marked factor of T rooted at u. Similarly, given a vertex $v \in \mathcal{D}om(T)$, we may use $T|v$ to denote the sequence of colors occurring along the access path of v in T. Furthermore, recall that, given a path π in T, $|\pi|$ denotes the number of edges of π and π_i, with $0 \leq i \leq |\pi|$, denotes the $i+1$-the vertex along π (for instance, π_0 identifies the root of the tree and $\pi_{|\pi|}$ identifies the last vertex along π). We now give define normal decompositions of finite and infinite paths.

Definition 3.3.17 *For every vertex v in T (resp., for every access path τ of a vertex v), the* normal decomposition *of v (resp., of τ) with respect to Π is the unique sequence $\pi, w_0, ..., w_{|\pi|}$ such that*

- *$w_0, ..., w_{|\pi|}$ is a finite sequence of finite words such that $v = w_0 \cdot ... \cdot w_{|\pi|}$;*
- *π is a finite path in Π;*
- *for every $0 \leq i < |\pi|$, w_i is a leaf of the marked factor $T^+_\Pi[\pi_i]$ colored with the label of the $i+1$-the edge of π;*
- *$w_{|\pi|}$ is either an internal vertex or a leaf of the marked factor $T^+_\Pi[\pi]$, depending on whether v is an internal vertex or a leaf of T.*

Note that, given $\pi, w_0, ..., w_{|\pi|}$ satisfying the above conditions, $v = w_0 \cdot ... \cdot w_{|\pi|}$ is a vertex of T such that $T(v) = T^+_\Pi[\pi](w_{|\pi|})$. Moreover, if $\pi, w_0, ..., w_{|\pi|}$ is the normal decomposition of a vertex v in T, then the a-successor $v \cdot a$ of v (if exists) has normal decomposition either of the form π, $w_0, ..., w_{|\pi|} \cdot a$, or of the form π', $w_0, ..., w_{|\pi|} \cdot a, \varepsilon$, where π' is the extension of π by a $T^+_\Pi[\pi](w_{|\pi|} \cdot a)$-labeled edge. Note that in the latter case $v \cdot a$ is exactly the last vertex along the path π' (hence it is also a vertex of the

factorization Π). Finally, the set $\mathcal{I}mg(T|v)$ of all colors that occur along the access path of v in T is given by the union, over all $0 \leq i \leq |\pi|$, of the sets $\mathcal{I}mg(T_\Pi[\pi_i]|w_i)$.

Definition 3.3.18 *For every infinite path τ in T, the* normal decomposition *of τ with respect to Π is the unique sequence $\pi, w_0, w_1, w_2, ...$ such that*

- *$w_0, w_1, w_2, ...$ is a (finite or infinite) sequence of words such that $\tau = w_0 \cdot w_1 \cdot w_2 \cdot ...$;*
- *π is a (finite or infinite) path in Π;*
- *for every $0 \leq i < |\pi|$, w_i is a leaf of the marked factor $T^+_\Pi[\pi_i]$ colored with the label of the $i+1$-th edge in π;*
- *if π is finite, then $w_{|\pi|}$ is an infinite word identifying an infinite path of the marked factor $T^+_\Pi[\pi]$.*

Conversely, given $\pi, w_0, w_1, w_2, ...$ satisfying the above conditions, it is easy to see that $\tau = w_0 \cdot w_1 \cdot w_2 \cdot ...$ is an infinite path of T. Moreover, if π is infinite, then $\mathcal{I}nf(T|\tau)$ coincides with the set of all colors $c \in \Sigma$ for which there are infinitely many $i \geq 0$ such that $c \in \mathcal{I}mg(T_\Pi[\pi_i]|w_i)$. Otherwise, if π is finite, then $\mathcal{I}nf(T|\tau)$ coincides with the set $\mathcal{I}nf(T_\Pi[\pi]|w_{|\pi|})$.

In succession, we give precise conditions that identify a suitable correspondence between a set of partial runs of M on the Δ-completion of T and a set of runs of M^Δ on the $\emptyset$-completion of C. In the sequel, we shall prove that such a correspondence holds if we properly choose complete sets of (partial) runs. In virtue of such a result, given any $M^\Delta, \emptyset$-type of C, one can compute an M, Δ-type of T. Moreover, we shall see that Theorem 3.3.14 follows almost trivially from such a correspondence.

Definition 3.3.19 *Let P be a partial run of M on T_Δ (i.e., the Δ-completion of T) and let R be a run of M^Δ on $C_\emptyset$ (i.e., the $\emptyset$-completion of C). We say that R* corresponds *to P (or, equivalently, P corresponds to R) iff the following properties are satisfied:*

i) $R(\varepsilon) = (c, 0, \mathcal{U}, \mathcal{V})$ with $c \in \Delta$, $\mathcal{U} = \{(P(\varepsilon), \emptyset, \emptyset)\}$, and $\mathcal{V} = \emptyset$;

ii) for every leaf v in P, there is an infinite path π in R, with $\mathcal{I}nf(R|\pi) = \{(b, 1, \mathcal{U}, \mathcal{V})\}$ and $\mathcal{U} = \{(r_l, U_l, Y_l)\}_{l \in L}$, and there is an index $l \in L$ such that $T(v) = b$, $P(v) = r_l$, and $\mathcal{I}mg(P|v) = Y_l$;

iii) for every infinite path τ in P, one of the following conditions holds

1. *there is an infinite path π in R such that $\mathcal{I}nf(R|\pi) = \{(b_k, 0, \mathcal{U}_k, \mathcal{V})\}_{k \in [n]}$ and $\mathcal{U}_k = \{(r_{k,l}, U_{k,l}, Y_k)\}_{l \in L_k}$ for all $k \in [n]$ (where $L_1, ..., L_k$ are finite sets of indices), and there are $\emptyset \subsetneq L'_1 \subseteq L_1$, ..., $\emptyset \subsetneq L'_n \subseteq L_n$ such that $\mathcal{I}nf(P|\tau) = \bigcup_{k \in [n]} \bigcup_{l \in L'_k} U_{k,l}$;*
2. *there is an infinite path π in R such that $\mathcal{I}nf(R|\pi) = \{(b_k, x, \mathcal{U}_k, \mathcal{V})\}_{k \in [n]}$, $x \in \{0, 1\}$, and $\mathcal{V} = \{V_g\}_{g \in G}$, and there is an index $g \in G$ such that $\mathcal{I}nf(P|\tau) = V_g$;*

iv) for every infinite path π in R, one of the following conditions holds

1. *there are $b \in \Delta$, $\mathcal{U} = \{(r_l, U_l, Y_l)\}_{l \in L}$ (where L is a finite set of indices), and $\mathcal{V} \subseteq \mathscr{P}(S)$ such that $\mathcal{I}nf(R|\pi) = \{(b, 1, \mathcal{U}, \mathcal{V})\}$ and for every index $l \in L$, there is a leaf v in P such that $T(v) = b$, $P(v) = r_l$, and $\mathcal{I}mg(P|v) = Y_l$;*

2. *there are* $n > 0$, $b_1, ..., b_n \in \Delta$, $\mathcal{U}_1, ..., \mathcal{U}_n \subseteq S \times \mathscr{P}(S) \times \mathscr{P}(S)$, *and* $\mathcal{V} \subseteq \mathscr{P}(S)$, *such that* $\mathcal{U}_k = \{(r_{k,l}, U_{k,l}, Y_k)\}_{l \in L_k}$ *for all* $k \in [n]$ *(where* $L_1, ..., L_k$ *are finite sets of indices),* $\mathcal{I}nf(R|\pi) = \{(b_k, 0, \mathcal{U}_k, \mathcal{V})\}_{k \in [n]}$, *and, for every* $\emptyset \subsetneq L'_1 \subseteq L_1$, ..., $\emptyset \subsetneq L'_n \subseteq L_n$, *there is an infinite path* τ *in* P *such that* $\mathcal{I}nf(P|\tau) = \bigcup_{k \in [n]} \bigcup_{l \in L'_k} U_{k,l}$;

v) *for every infinite path* π *in* R, *if* $\mathcal{I}nf(R|\pi)$ *is of the form* $\{(b_k, x, \mathcal{U}_k, \mathcal{V})\}_{k \in [n]}$, *with* $\mathcal{V} = \{V_g\}_{g \in G}$, *then, for every* $g \in G$, *there is an infinite path* τ *in* P *such that* $\mathcal{I}nf(P|\tau) = V_g$.

The following lemma is crucial in that it provides a two-way correspondence between the partial runs of M on the Δ-completion of T and the runs of M^Δ on the $\emptyset$-completion of a retraction C of T w.r.t M.

Lemma 3.3.20 *For every run* R *of* M^Δ *on* $C_\emptyset$ *such that* $R(\varepsilon) = (c, 0, \mathcal{U}, \mathcal{V})$, *with* $c \in \Delta$, $\mathcal{U} = \{(r, \emptyset, \emptyset)\}$, $r \in S$, *and* $\mathcal{V} = \emptyset$, *there is a partial run* P *of* M *on* T_Δ *that corresponds to* R. *Conversely, for every partial run* P' *of* M *on* T_Δ, *there is a partial* P *of* M *on* T_Δ *and a run* R *of* M^Δ *on* C_ε *such that* $P \preceq P'$ *and* R *corresponds to* P.

Proof. The proof of the lemma is quite long. Recall that, thinking of C as a deterministic tree, the domain of the factorization Π of T is not necessarily isomorphic to the domain of C. Thus, given a (finite or infinite) path π in Π and a (finite or infinite) path π' in C, we say that π and π' are *corresponding* paths iff the sequences of labels along π and π' coincide. Note that, for each path π in Π, there is a unique path π' in C that corresponds to π; on the other hand, given a path π' in C, there exists at least one, but possibly more than one, path π in Π corresponding to π'. Similarly, given $u \in \mathcal{D}om(\Pi)$ and $u' \in \mathcal{D}om(C)$, we say that u and u' are corresponding vertices iff they have corresponding access paths in Π and in C, respectively. Given a vertex u of Π (resp., a path π in Π), we denote by $\vec{u}$ (resp., $\vec{\pi}$) the unique vertex of C that corresponds to u (resp., the unique path in C that corresponds to π).
We now prove the first implication of the lemma. Let R be a run of M^Δ on $C_\emptyset$ and assume that (i), for each $u \in \Delta^*$, $R(u)$ is of the form $(c_u, x_u, \mathcal{U}_u, \mathcal{V}_u)$, with $c_u \in \Delta$, $x_u \in \{0, 1\}$, $\mathcal{U}_u = \{(r_{u,l}, U_{u,l}, Y_{u,l})\}_{l \in L_u}$, and $\mathcal{V}_u = \{V_{u,g}\}_{g \in G_u}$, and (ii) $x_\varepsilon = 0$, $\mathcal{U}_\varepsilon = \{(r, \emptyset, \emptyset)\}$, and $\mathcal{V}_\varepsilon = \emptyset$. Moreover, for every $u \in \mathcal{D}om(C)$, let

$$C(u) = \{(s_{u,h}, \{(b_{u,i}, q_{u,h,i}, Q_{u,h,i})\}_{i \in I_u}, \{W_{u,h,j}\}_{j \in J_u}) \,:\, h \in H_u\}.$$

In the following, given a vertex $u \in \mathcal{D}om(\Pi)$, we shortly denote by T_u the *marked* factor of T rooted at u with respect to Π and we define an index $l_u \in L_{\vec{u}}$ and a partial run P_u of M on T_u such that, for all $b \in \Delta$,

i) $\mathcal{U}_{\vec{u} \cdot b}$ consists of all tuples $\big(P_{u'}(w), \mathcal{I}mg(P_{u'}|w), Y_{l_{u'}} \cup \mathcal{I}mg(P_{u'}|w)\big)$, where u' ranges over all vertices of Π that correspond to $\vec{u}$ and w ranges over all b-colored leaves of $T_{u'}$,

ii) $\mathcal{V}_{\vec{u} \cdot b}$ is the union of the sets $\mathcal{V}_{\vec{u}}$ and $\{\mathcal{I}nf(P_{u'}|\tau) \,:\, \tau \in \mathcal{B}ch(P_{u'})\}$.

Precisely, we define l_u and P_u by induction on the length n of the access path of u in Π.

- For $n = 0$, the unique vertex u of Π whose access path has length n is the root ε of Π. Recall that $R(\varepsilon) = (c_\varepsilon, x_\varepsilon, \mathcal{U}_\varepsilon, \mathcal{V}_\varepsilon)$, with $\mathcal{U}_\varepsilon$ being a singleton; hence, we can assume that L_ε is a singleton as well and we define l_ε as the unique index of L_ε.
- Let $n \geq 0$ and suppose that $l_{u'}$ (resp., $P_{u'}$) is defined for every vertex u' of Π whose access path has length $n' \leq n$ (resp., $n'' < n$). We first define $P_{u'}$ for every vertex u' of Π whose access path has length n, and then we define $l_{u''}$ for every successor u'' of u' in Π. Let u be any vertex of C such that $|u| = n$ and recall that $C(u)$ consists of all tuples $\big(s_{u,h}, \{(b_{u,i}, q_{u,h,i}, Q_{u,h,i})\}_{i \in I_u}, \{W_{u,h,j}\}_{j \in J_u}\big)$, where h ranges over the finite set H_u. By definition of transition relation of M^Δ, $x_u = 0$ holds and for every $l \in L_u$, there is $h_l \in H_u$ such that $s_{u,h_l} = r_l$ and, for all $b \in \Delta$, $R(u \cdot b) = (c_{u \cdot b}, x_{u \cdot b}, \mathcal{U}_{u \cdot b}, \mathcal{V}_{u \cdot b})$, where

$$\begin{cases} c_{u \cdot b} = b, \\ x_{u \cdot b} = 0, \\ \mathcal{U}_{u \cdot b} = \bigcup_{l \in L_u} \big\{(q_{u,h_l,i}, Q_{u,h_l,i}, Y_l \cup Q_{u,h_l,i}) \,:\, i \in I_u, b_{u,i} = b\big\}, \\ \mathcal{V}_{u \cdot b} = \mathcal{V}_u \cup \bigcup_{l \in L_u} \big\{W_{u,h_l,j} \,:\, j \in J_u\big\}. \end{cases}$$

  Given the above equalities, we can assume without loss of generality that $L_{u \cdot b} = \{(h_l, i) : l \in L_u, i \in I_u, b_{u,i} = b\}$. Therefore, for every $l' = (h_l, i) \in L_{u \cdot b}$, we have $r_{u \cdot b, l'} = q_{u,h_l,i}$, $U_{u \cdot b, l'} = Q_{u,h_l,i}$, and $Y_{u \cdot b, l'} = Y_{u,l} \cup Q_{u,h_l,i}$. Similarly, we can assume that $G_{u \cdot b}$ is the (disjoint) union of the two sets of indices G_u and $\{(h_l, j) : l \in L_u, j \in J_u\}$, and, for all $g' \in G_u$ (resp., for all $g' = (h_l, j) \in H_u \times J_u$), $V_{u \cdot b, g'} = V_{u,g'}$ holds (resp., $V_{u \cdot b, g'} = W_{u,h_l,j}$ holds). By definition of M, Δ-type, for every vertex u' of Π that corresponds to u, there must exist a partial run $P_{u'}$ of M on $T_{u'}$ whose feature is $\big(s_{u,h_{u'}}, \{(b_{u,i}, q_{u,h_{u'},i}, Q_{u,h_{u'},i})\}_{i \in I_u}, \{W_{u,h_{u'},j}\}_{j \in J_u}\big)$, where $h_{u'}$ is a shorthand for $h_{l_{u'}}$ (note that, by inductive hypothesis, $l_{u'}$ is well-defined). Moreover, for every leaf w of $T_{u'}$, there is an index $i_w \in I_u$ satisfying $T_{u'}(w) = b_{u,i_w}$, $P_{u'}(w) = q_{u,h_{u'},i_w}$, and $\mathcal{I}mg(P_{u'}|w) = Q_{u,h_{u'},i_w}$. Since any b-successor u'' of u' in Π can be written as $u' \cdot w$, where w is a suitable b-colored leaf of $T_{u'}$, we can define $l_{u''}$ to be the index $(h_{u'}, i_w)$ belonging to $L_{u \cdot b}$.

Given the above definitions, we can build a partial run P on T_Δ by properly combining the partial runs P_u, for each $u \in \mathcal{D}om(\Pi)$. For each vertex $v \in \mathcal{D}om(T_\Delta)$, if π, $w_0, ..., w_{|\pi|}$ is the normal decomposition of v with respect to Π and u is the last vertex along π, then we set $P(v) = P_u(w_{|\pi|})$. We now show that P is a partial run of M on T_Δ. Given any vertex $v \in \mathcal{D}om(T_\Delta)$, let π, $w_0, ..., w_{|\pi|}$ be the normal decomposition of v with respect to Π. If $v \cdot a \in \mathcal{D}om(T_\Delta) \setminus \mathcal{D}om(\Pi)$, then π, $w_0, ..., w_{|\pi|} \cdot a$ is the normal decomposition of $v \cdot a$ with respect to Π and hence $P(v \cdot a) = P_u(w_{|\pi|} \cdot a)$ follows. Otherwise, if $v \cdot a \in \mathcal{D}om(\Pi)$, then π', $w_0, ..., w_{|\pi|} \cdot a, \varepsilon$ is the normal decomposition of $v \cdot a$ with respect to Π, with π' being the extension of π with a $T_u(w_{|\pi|} \cdot a)$-labeled edge. In such a case, by denoting with u' the last vertex along π' and by exploiting the definition of P_u and $P_{u'}$, we obtain $P(v \cdot a) = P_{u'}(\varepsilon) = P_u(w_{|\pi|} \cdot a)$. Therefore, since P_u is a partial run of M on T_u, we can write $\big(P_u(w_{|\pi|}), T_u(w_{|\pi|}), (P_u(w_{|\pi|} \cdot a))_{a \in \Lambda}\big) \in \delta$, and this implies $\big(P(v), T(v), (P(v \cdot a))_{a \in \Lambda}\big) \in \delta$, for every $v \in \mathcal{D}om(T_\Delta)$. This proves that P is a partial run of M on T_Δ. We now show that P corresponds to R, namely,

P and R satisfy the conditions i)–v) of Definition 3.3.19. We preliminary state two relevant properties satisfied by P and R (these properties follows trivially from the above definitions).

Property 1. For every vertex $u \in \mathcal{D}om(\Pi)$, we have:
- $P_u(\varepsilon) = r_{\vec{u},l_u}$;
- for every leaf w of P_u, $u' = u \cdot w$ is a vertex of Π, $\vec{u}' = \vec{u} \cdot b$, with $b = T_u(w) = c_{\vec{u}'}$, $P_u(w) = r_{\vec{u}',l_{u'}}$, $\mathcal{I}mg(P_u|w) = U_{\vec{u}',l_{u'}}$, and $\mathcal{I}mg(P|u) = Y_{\vec{u},l_u}$;
- for every infinite path τ in P_u and for every successor u' of u in Π, $\mathcal{I}nf(P_u|\tau) \in \mathcal{V}_{\vec{u}'}$.

Property 2. For every vertex $u \in \mathcal{D}om(C)$ and for every index $l \in L_u$, there is a vertex $u'_{u,l}$ of Π that corresponds to u and that satisfies:
- $P_{u'_{u,l}}(\varepsilon) = r_{u,l}$;
- for every $b \in \Delta$ and for every $l' = (l,i) \in L_{u \cdot b}$, with $i \in I_u$, there is a leaf $w_{u,l,i}$ of $P_{u'_{u,l}}$ such that $u'_{u \cdot b,l'} = u'_{u,l} \cdot w_{u,l,i}$, $T_{u'_{u,l}}(w_{u,l,i}) = b$, $P_{u'_{u,l}}(w) = r_{u \cdot b,l'}$, $\mathcal{I}mg(P_{u'_{u,l}}|w) = U_{u \cdot b,l'}$, and $\mathcal{I}mg(P|u'_{u,l}) = Y_{u,l}$;
- for every $b \in \Delta$ and for every $g \in G_{u \cdot b}$, there is an infinite path $\tau_{u,g}$ in P_u such that $\mathcal{I}nf(P_{u'_{u,l}}|\tau_{u,g}) = V_{u \cdot b,g}$.

Now, as regards condition i) of Definition 3.3.19, we have $R(\varepsilon) = (c_\varepsilon, x_\varepsilon, \mathcal{U}_\varepsilon, \mathcal{V}_\varepsilon)$, where $x_\varepsilon = 0$, $\mathcal{U}_\varepsilon = \big\{(P(\varepsilon), \emptyset, \emptyset)\big\}$, and $\mathcal{V}_\varepsilon = \emptyset$.
As for condition ii), let v be a colored leaf of P, $b = T_\Delta(v)$, τ, $w_0, ..., w_{|\tau|}$ the normal decomposition of v with respect to Π, and $\vec{\tau}$ the (unique) path in C that corresponds to τ. Further let u (resp., $\vec{u}$) be the last vertex of Π along τ (resp., the last vertex of C along $\vec{\pi}$). Clearly, $w = w_{|\tau|}$ is a b-colored leaf of T_u, and hence, by letting $u' = u \cdot w$ and $b = c_{\vec{u}'}$ and by applying Property 1, we obtain $T(v) = T_u(w) = b$, $P(v) = P_u(w) = r_{\vec{u}',l_{u'}}$, and $\mathcal{I}mg(P|v) = \mathcal{I}mg(P|u) \cup \mathcal{I}mg(P_u(w)) = Y_{\vec{u},l_u} \cup U_{\vec{u}',l_{u'}} = Y_{\vec{u}',l_{u'}}$. Moreover, by definition of transition relation of M^Δ, for every proper descendant u'' of $\vec{u}'$ in $C_\emptyset$, $R(u'') = (b, 1, \mathcal{U}_{\vec{u}'}, \mathcal{V}_{\vec{u}'})$ and hence, being π any infinite path of $C_\emptyset$ that traverses the vertex $\vec{u}'$, we have $\mathcal{I}nf(R|\pi) = \big\{(b, 1, \mathcal{U}_{\vec{u}'}, \mathcal{V}_{\vec{u}'})\big\}$. This shows that condition ii) of Definition 3.3.19 is satisfied.
Now, let τ be any infinite path in P and let π, $w_0, w_1, w_2, ...$ be the normal decomposition of τ with respect to Π. Two cases arise: either π is an infinite path or π is a finite path. In the former case, by letting $u_i = \pi_i$ (i.e., the $i+1$-the vertex along π), we know that the set $\mathcal{I}nf(P|\tau)$ consists of all and only the states $s \in S$ for which there are infinitely many $i \geq 0$ such that $s \in \mathcal{I}mg(P_{u_i}|w_i)$. Consider a generic state $s \in \mathcal{I}mg(P_{u_i}|w_i)$ and note that, by Property 1, $s \in U_{\vec{u}_i,l_{u_i}}$ follows. Let $\vec{\pi}$ be the (unique) path of C that corresponds to π. From the Pigeonhole Principle, since M^Δ has only a finite number of states, we know that, if there are infinitely many $i \geq 0$ such that $s \in U_{\vec{u}_i,l_{u_i}}$, then there are infinitely many states of M^Δ along $\vec{\pi}$ of the form $(c, 0, \mathcal{U}, \mathcal{V})$, with $\mathcal{U} = \big\{(r_l, U_l, Y_l)\big\}_{l \in L}$ and $s \in U_l$ for some $l \in L$. The converse implication holds trivially. Therefore, we can claim that, if $\mathcal{I}nf(R|\vec{\pi})$ is a set of the form $\big\{(c'_k, 0, \mathcal{U}'_k, \mathcal{V}')\big\}$, with $\mathcal{U}'_k = \big\{(r'_{k,l}, U'_{k,l}, Y'_{k,l})\big\}_{l \in L'_k}$ for each $k \in [n]$, then there are suitable sets of indices $\emptyset \subsetneq L''_1 \subseteq L'_1$, ..., $\emptyset \subsetneq L''_n \subseteq L'_n$ such that $\mathcal{I}nf(P|\tau) = \bigcup_{k \in [n]} \bigcup_{l \in L''_k} U_{k,l}$. This accounts for condition iii), under the proviso

that π is infinite. In case π is a finite path, we denote by u the last vertex of Π along π. By definition of normal decomposition, $\tau' = w_{|\pi|}$ is a suffix of π denoting an infinite path in T_u. Therefore, we have $\mathcal{I}nf(P|\tau) = \mathcal{I}nf(P_u|\tau')$ and, by Property 1, for every $b \in \Delta$, $\mathcal{I}nf(P_u|\tau') \in \mathcal{V}_{\vec{u}\cdot b}$. From the definition of transition relation of M^{Δ}, we know that, for any infinite path π' of $C_\emptyset$ that extends $\vec{\pi}$, $\mathcal{I}nf(R|\pi')$ is a set of the form $\{(c'_k, 0, \mathcal{U}'_k, \mathcal{V}')\}$, with $\mathcal{I}nf(P|\tau) \in \mathcal{V}'$.

As for condition iv), let π be an infinite path in R. We distinguish between two cases: either $\perp \in \mathcal{I}mg(C_\emptyset|\pi)$ (namely, from a certain point on, the vertices of $C_\emptyset$ along π are colored with $\perp$) or $\perp \notin \mathcal{I}mg(C_\varepsilon|\pi)$ (namely, π is a path lying entirely inside C). In the former case, let u be the last vertex along π that belongs to the domain of C, and let $u \cdot b$ the successor of u along π. By definition of transition relation of M^{Δ}, $\mathcal{I}nf(C_\emptyset|\pi)$ is a singleton containing the state $(c_{u\cdot b}, 1, \mathcal{U}_{u\cdot b}, \mathcal{V}_{u\cdot b})$. Let $l' = (l, i) \in L_{u\cdot b}$, with $l \in L_u$ and $i \in I_u$. By Property 2, there is a vertex $u'_{u,l}$ of Π, which corresponds to u, and there is a leaf $w_{u,l,i}$ of $P_{u'_{u,l}}$ such that, being $v = u'_{u,l} \cdot w_{u,l,i}$, $T(v) = T_{u'_{u,l}}(w_{u,l,i}) = b$, $P(v) = P_{u'_{u,l}}(w) = r_{u\cdot b,l'}$, and $\mathcal{I}mg(P|v) = \mathcal{I}mg(P|u'_{u,l}) \cup \mathcal{I}mg(P_{u'_{u,l}}|w_{u,l,i}) = Y_{u,l} \cup U_{u\cdot b,l'} = Y_{u\cdot b,l'}$. This proves that condition iv) is met, under the proviso that π does not lie entirely inside C. Otherwise, suppose that π lies entirely inside C. For every $d \geq 0$, we let $u_d = \pi_d$ (i.e., the $d+1$-th vertex along π). Now, for each $d \geq 0$, we chose a generic $l_d \in L_{u_d}$. Without loss of generality, we can assume that for every $d \geq 0$, there is $i_d \in I_{u_d}$ such that $l_{d+1} = (l_d, i_d)$, $i \in I_{u_d}$, and $u_{d+1} = u_d \cdot b_{u_d,i_d}$. By property 2, we know that, for every $d \geq 0$, there is a vertex $u'_d = u'_{u_d,l_d}$ of Π, which corresponds to u_d, and there is a leaf $w_d = w_{u_d,l_d,i_d}$ of $P_{u'_{d,l}}$ such that $u'_{d+1} = u'_d \cdot w_d$, $T_{u'_d}(w_d) = b_{u_d,i_d}$, $P_{u'_d}(w) = r_{u\cdot b,l_{d+1}}$, $\mathcal{I}mg(P_{u'_d}|w_d) = U_{u_{d+1},l_{d+1}}$, and $\mathcal{I}mg(P|u'_d) = Y_{u_d,l_d}$. The vertices $u'_0, u'_1, ...$ define an infinite path π' in Π (to see this simply notice that, since u_{d+1} is a successor of u_d in C and since each vertex $u_d \in \mathcal{D}om(C)$ corresponds to $u'_d \in \mathcal{D}om(\Pi)$, we have that u'_{d+1} is a successor of u'_d in Π and hence there exists a unique infinite path in Π that traverses the vertices $u'_0, u'_1, ...$ Similarly, the vertices $w_0, w_1, ...$ define an infinite path τ in T_Δ, whose normal decomposition with respect to Π is exactly $\pi', w_0, w_1, ...$ (indeed $u'_{d+1} = u'_d \cdot w_d$). From properties of normal decompositions, we know that $\mathcal{I}nf(P|\tau)$ consists of all and only the states $s \in S$ for which there are infinitely many $d \geq 0$ satisfying $s \in \mathcal{I}mg(P_{u'_d}|w_d) = U_{u_{d+1},l_{d+1}}$. Consider now the set $\mathcal{I}nf(R|\pi) = \{z_k\}_{k\in[n]}$ of states of M^{Δ} occurring infinitely often along π. We write each state z_k as $(c'_k, 0, \mathcal{U}'_k, \mathcal{V}'_k)$, where $\mathcal{U}'_k = \{(r'_{k,l'}, U'_{k,l'}, Y'_{k,l'})\}_{l'\in L'_k}$. From the previous observations and from the Pigeonhole Principle, we know that $s \in \mathcal{I}nf(P|\tau)$ iff there are $k' \in [n]$ and $l'_k \in L'_k$ such that $s \in U'_{k',l'_k}$ and, for infinitely many $d \geq 0$, $l'_k = l_d$, $\mathcal{U}'_k = \mathcal{U}_{u_d}$, and $\mathcal{V}'_k = \mathcal{V}_{u_d}$. This means that $\mathcal{I}nf(P|\tau) = \bigcup_{k\in[n]} \bigcup_{l'\in L''_k} U'_{k,l'}$, where, for each $k \in [n]$, L''_k is the set of all and only the indices $l' \in L'_k$ for which there are infinitely many $d \geq 0$ such that $l' = l_d$, $\mathcal{U}'_k = \mathcal{U}_{u_d}$, and $\mathcal{V}'_k = \mathcal{V}_{u_d}$. Clearly, L''_k is a non-empty set included in L'_k. Conversely, since the indices $l_0 \in L_0$, $l_1 \in L_1$, ..., were chosen arbitrarily, given $\emptyset \subsetneq L''_1 \subseteq L'_1$, ..., $\emptyset \subsetneq L''_1 \subseteq L'_1$, we can properly chose $l_0 \in L_0$, $l_1 \in L_1$, ..., in such a way that the resulting path τ satisfies $\mathcal{I}nf(P|\tau) = \bigcup_{k\in[n]} \bigcup_{l'\in L''_k} U_{k,l'}$. This shows that condition iv) is satisfied if π is a path lying entirely inside C.

As for the last condition, let π be an infinite path in R and let $\mathcal{I}nf(R|\pi)$ be the set $\{(c'_k, x, \mathcal{U}'_k, \mathcal{V}')\}_{k\in[n]}$, with $\mathcal{V}' = \{V'_g\}_{g\in G'}$. Let g be a generic index from G'. By definition of the transition relation of M^Δ, there must exist a vertex u along π such that V'_g belongs to $\mathcal{V}_u$. We can assume without loss of generality that u is the first vertex along π satisfying $V'_g \in \mathcal{V}_u$. Now, let w be the predecessor of u. Clearly, $\mathcal{V}_u = \mathcal{V}_w \cup \bigcup_{l\in L_w} \{W_{w,l,j}\}_{j\in J_w}$ and hence there are $l \in L_w$ and $j \in J_w$ such that $V'_g = W_{w,l,j}$. By applying Property 2 to the vertex w, we know that there is a vertex $u'_{w,l}$ of Π that corresponds to w and there is an infinite path $\tau_{w,g}$ in $P_{u'_{w,l}}$ such that $\mathcal{I}nf(P_{u'_{w,l}}|\tau_{w,g}) = V'_g$. Therefore, there is an infinite path τ in P, obtained by concatenating $u'_{w,l}$ and $\tau_{w,g}$, such that $\mathcal{I}nf(P|\tau) = V'_g$ and this accounts for condition v).

We just proved that, given a suitable run R of M^Δ on $C_\emptyset$, there is a corresponding partial run P of M on T_Δ. Now we prove the second part of the lemma, namely, given any partial run P' of M on T_Δ, there is a partial run P of M on T_Δ and there is a run R of M^Δ on $C_\emptyset$ such that $P \preceq P'$ and R corresponds to P. Let P' be a partial run of M on T_Δ. Since $\mathcal{D}om(P') = \mathcal{D}om(T_\Delta) \supseteq \mathcal{D}om(T)$, Π is also factorization of P'. Hence, for every $u \in \mathcal{D}om(\Pi)$, we can denote by T_u the *marked* factor of T_Δ rooted at u with respect to Π and by P'_u the *unmarked* factor of P' in u with respect to Π. Clearly, for every $u \in \mathcal{D}om(\Pi)$, P'_u is a partial run of M on T_u. Moreover, given a vertex $u \in \mathcal{D}om(C)$, we assume that

$$C(u) = \big\{\big(s_{u,h}, \{(b_{u,i}, q_{u,h,i}, Q_{u,h,i})\}_{i\in I_u}, \{W_{u,h,j}\}_{j\in J_u}\big) \,:\, h \in H_u\big\}.$$

Now, for each $u \in \mathcal{D}om(C_\emptyset)$, let L_u and G_u be the sets of indices defined inductively as follows:

- L_ε is any singleton and $G_\varepsilon = \emptyset$,
- $L_{u\cdot b} = \{(l, i) \,:\, l \in L_u, i \in I_u, b_{u,i} = b\}$ and $G_{u\cdot b} = G_u \cup \{(l, j) \,:\, l \in L_u, j \in J_u\}$.

Below, for any given vertex $u \in \mathcal{D}om(C)$ and for any given index $l \in L_u$, we define

i) a vertex $u'_{u,l}$ of Π that corresponds to u,
ii) an index $h_{u,l} \in H_u$ such that $P'_{u'_{u,l}}(\varepsilon) = s_{u,h_{u,l}}$,
iii) for each $b \in \Delta$ and for each $l' = (l, i) \in L_{u\cdot b}$, a leaf $w_{u,l,i}$ of $P'_{u'_{u,l}}$ such that $T_{u'_{u,l}}(w_{u,l,i}) = b$, $P_{u'_{u,l}}(w_{u,l,i}) = q_{u,h_{u,l},i}$, $\mathcal{I}mg(P'_{u'_{u,l}}|w_{u,l,i}) = Q_{u,h,i}$, and $u'_{u\cdot b,l'} = u'_{u,l} \cdot w_{u,l,i}$,
iv) for each $b \in \Delta$ and for each $g' = (l, j) \in G_{u\cdot b}$, an infinite path $\tau_{u,l,j}$ in $P'_{u'_{u,l}}$ such that $\mathcal{I}nf(P'_{u'_{u,l}}|\tau_{u,l,j}) = W_{u,h_{u,l},j}$.

The definitions are given by induction on the length n of the access path of u in C.

- For $n = 0$, $u = \varepsilon$ is the unique vertex of C whose access path has length $n = 0$ and L_u is a singleton. Hence, being l the unique index from L_u, we set $u'_{\varepsilon,l} = \varepsilon$.
- Let $n \geq 0$ and suppose that $u'_{u,l}$ is defined for every vertex u of C whose access path has length $n' \leq n$ and for every $l \in L_u$. Since $u'_{u,l}$ corresponds to u, $C(u)$ is an M, Δ-type of $T_{u'_{u,l}}$. Hence there is $h_{u,l} \in H_u$ such that $P'_{u'_{u,l}}(\varepsilon) = s_{u,h_{u,l}}$

holds and for every $b \in \Delta$ and $l' = (l,i) \in L_{u\cdot b}$, there is a leaf $w_{u,l,i}$ of $P'_{u'_{u,l}}$ such that $T_{u'_{u,l}}(w_{u,l,i}) = b_{u,i} = b$, $P'_{u'_{u,l}}(w_{u,l,i}) = q_{u,h_{u,l},i}$, and $\mathcal{I}mg(P'_{u'_{u,l}}|w_{u,l,i}) = Q_{u,h_{u,l},i}$. We then define $u'_{u\cdot b,l'} = u'_{u,l} \cdot w_{u,l,i}$. Moreover, we know that, for every $g' = (l,j) \in G_{u\cdot b}$, there must exist an infinite path $\tau_{u,l,j}$ in $P'_{u'_{u,l}}$ such that $\mathcal{I}nf(P'_{u'_{u,l}}|\tau_{u,l,j}) = W_{u,h_{u,l},j}$.

Now, we can build a run R of M^{Δ} on $C_{\emptyset}$. We let $\mathcal{D}om(R) = \Delta^*$ and, for every $u \in \mathcal{D}om(R)$, $R(u) = (c_u, x_u, \mathcal{U}_u, \mathcal{V}_u)$, where c_u, x_u, $\mathcal{U}_u$, and $\mathcal{V}_u$ are inductively defined as follows:

- for $u = \varepsilon$, c_u is any element of Δ, $x_u = 0$, $\mathcal{U}_u = \{(r_{u,l}, U_{u,l}, Y_{u,l})\}_{l \in L_\varepsilon}$ and $\mathcal{V}_u = \emptyset$, where l is the unique index of L_u (recall that L_u is a singleton for $u = \varepsilon$), $r_{u,l} = P'(\varepsilon)$, $U_{u,l} = \emptyset$, and $Y_{u,l} = \emptyset$;
- for every vertex $u \in \mathcal{D}om(C)$ and for every $b \in \Delta$, $c_{u\cdot b} = b$, $x_{u\cdot b} = 0$, $\mathcal{U}_{u\cdot b} = \{(r_{u\cdot b,l'}, U_{u\cdot b,l'}, Y_{u\cdot b,l'})\}_{l' \in L_{u\cdot b}}$, and $\mathcal{V} = \{V_{u\cdot b,g}\}_{g \in G_{u\cdot b}}$, where, for every $l' = (l,i) \in L_{u\cdot b}$, $r_{u\cdot b,l'} = P'_{u'_{u\cdot b,l'}}(\varepsilon)$, $U_{u\cdot b,l'} = \mathcal{I}mg(P'_{u'_{u,l}}|w_{u,l,i})$, $Y_{u\cdot b,l'} = Y_{u,l} \cup U_{u,l}$, for every $g' \in G_u$, $V_{u\cdot b,g'} = V_{u,g'}$, and for every $g' = (l,j) \in L_u \times J_u$, $V_{u\cdot b,g'} = \mathcal{I}nf(P'_{u'_{u,l}}|\tau_{u,l,j})$;
- for every vertex $u \in \mathcal{D}om(C_{\emptyset}) \setminus \mathcal{D}om(C)$ and for every $b \in \Delta$, we set $c_{u\cdot b} = c_u$, $x_{u\cdot b} = 1$, $\mathcal{U}_{u\cdot b} = \mathcal{U}_u$, and $\mathcal{V}_{u\cdot b} = \mathcal{V}_u$.

It is routine to check that R is a partial run of M^{Δ} on C_ε. Moreover, $R(\varepsilon)$ is of the form $\big(c, 0, \{r, \emptyset, \emptyset\}, \emptyset\big)$. Hence, by exploiting the first implication of the lemma, there is a partial run P of M on T_Δ that corresponds to R. We complete the proof of the lemma, by briefly showing that $P \preceq P'$. Clearly, $P(\varepsilon) = r = P'(\varepsilon)$. Let v be a leaf of P. Since P corresponds to R, there is an infinite path π in R, with $\mathcal{I}nf(R|\pi) = \{(b', 1, \mathcal{U}', \mathcal{V}')\}$ and $\mathcal{U}' = \{(r'_{l'}, U'_{l'}, Y'_{l'})\}_{l' \in L'}$, and there is an index $l' \in L'$ such that $T(v) = b'$, $P(v) = r'_{l'}$, and $\mathcal{I}mg(P|v) = Y'_{l'}$. Moreover, there must exist a vertex $u \in \mathcal{D}om(C)$ along π such that $R(u \cdot b') = (b', 0, \mathcal{U}', \mathcal{V}')$. By definition of R, this implies that there is a vertex $v' = u'_{u\cdot b',l'}$ such that $T(v') = b' = T(v)$, $P'(v') = r'_{l'} = P(v)$, and $\mathcal{I}mg(P'|v') = Y'_{l'} = \mathcal{I}mg(P|v)$. Finally, let τ be an infinite path in P. Since P corresponds to R, one of the following conditions holds:

1. there is an infinite path π in R, with $\mathcal{I}nf(R|\pi) = \{(b'_k, 0, \mathcal{U}'_k, \mathcal{V}')\}_{k \in [n]}$ and $\mathcal{U}'_k = \{(r'_{k,l}, U'_{k,l}, Y'_{k,l})\}_{l \in L'_k}$ for all $k \in [n]$, and there are $\emptyset \subsetneq L''_1 \subseteq L'_1$, ..., $\emptyset \subsetneq L''_n \subseteq L'_n$ such that $\mathcal{I}nf(P|\tau) = \bigcup_{k \in [n]} \bigcup_{l \in L''_k} U'_{k,l}$;
2. there is an infinite path π in R, with $\mathcal{I}nf(R|\pi) = \{(b'_k, x, \mathcal{U}'_x, \mathcal{V}')\}_{k \in [n]}$ and $\mathcal{V}' = \{V'_g\}_{g \in G'}$, and there is an index $g \in G'$ such that $\mathcal{I}nf(P|\tau) = V'_g$.

In the former case, for every $d \geq 0$, we let $u_d = \pi_d$ (i.e., the $d+1$-th vertex along π). For each $d \geq 0$, $u_d \in \mathcal{D}om(C)$ holds and hence we can properly choose $l_d \in L_{u_d}$ in such a way that, for every $k \in [n]$, L''_k is the set consisting of all indices l_d for which $b'_k = b_{u_d}$, $\mathcal{U}'_k = \mathcal{U}_{u_d}$, and $L'_k = L_{u_d}$. This implies that for every $d \geq 0$, if we let $u'_d = u'_{u_d,l_d} \in \mathcal{D}om(\Pi)$, i_d such that $l_{d+1} = (l_d, i_d)$, and $w_d = w_{u_d,l_d,i_d}$, then $\mathcal{I}mg(P_{u'_d}|w_d) = U_{u_{d+1},l_{d+1}}$. The vertex u'_d identifies an infinite path π' in Π such that $\pi', w_0, w_1, ...$ is the normal decomposition of an infinite path τ' in P' with respect to Π.

Therefore, we have $\mathcal{I}nf(P'|\tau') = \bigcup_{d\geq 0} \mathcal{I}mg(P_{u'_d}|w_d) = \bigcup_{k\in[n]} \bigcup_{l\in L''_k} U'_{k,l} = \mathcal{I}nf(P|\tau)$. In the latter case, by definition of the transition relation of M^Δ, there must exist a vertex u along π such that V'_g belongs to $\mathcal{V}_u$. We can assume without loss of generality that u is the first vertex along π satisfying $V'_g \in \mathcal{V}_u$. Now, let w be the predecessor of u. Clearly, $\mathcal{V}_u = \mathcal{V}_w \cup \bigcup_{l\in L_w} \{W_{w,l,j}\}_{j\in J_w}$ and hence there are $l \in L_w$ and $j \in J_w$ such that $V'_g = W_{w,l,j}$. This implies that there is a vertex $u'_{w,l}$ of Π that corresponds to w and there is an infinite path $\tau_{w,g}$ in $P'_{u'_{w,l}}$ such that $\mathcal{I}nf(P'_{u'_{w,l}}|\tau_{w,g}) = V'_g$. Consequently, by letting τ' be the infinite path of P' obtained by concatenating $u'_{w,l}$ and $\tau_{w,g}$, we have $\mathcal{I}nf(P'|\tau') = V'_g = \mathcal{I}nf(P|\tau)$. This shows that $P \preceq P'$. ∎

Finally, we can prove Theorem 3.3.14.

Proof of Theorem 3.3.14. Let $M = (S, \Sigma \cup \Delta \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$ be a tree automaton running on a Σ-colored Δ-augmented tree T and let $M^\Delta = (Z, \mathscr{T}_{M,\Delta} \cup \{\bot\}, \Delta, \delta', \mathcal{I}', \mathcal{F}')$ be the tree automaton running on a retraction C of T with respect to M, as specified in Definition 3.3.13. We first prove the left to right implication. Let R be a successful run of M on $T_\emptyset$ and let P' be the restriction of R to T_Δ (note that $\mathcal{D}om(T_\Delta) \subseteq \mathcal{D}om(T_\emptyset)$. By Lemma 3.3.20, there is a partial run P of M on T_Δ and a run R' of M^Δ on the retraction C such that $P \preceq P'$, $R'(\varepsilon) = (c, 0, \mathcal{U}, \mathcal{V})$ with $\mathcal{U} = \{R(\varepsilon), \emptyset, \emptyset\}$ and $\mathcal{V} = \emptyset$, and for every infinite path π in R', $\mathcal{I}nf(R'|\pi)$ is a set of the form $\{(b_k, x, \mathcal{U}'_k, \mathcal{V}')\}_{k\in[n]}$ satisfying:

i) either $n = 1$, $x = 1$, $\mathcal{U}'_1 = \{(r_l, U_l, Y_l)\}_{l\in L}$, and for all $l \in L$, there is $v \in \mathcal{F}r(P)$ such that $T(v) = b_1$ and $P(v) = r_l$ or $n \geq 1$, $x = 0$, $\mathcal{U}'_k = \{(r_{k,l}, U_{k,l}, Y_{k,l})\}_{l\in L_k}$, and, whenever $\emptyset \subsetneq L'_1 \subseteq L_1$, ..., $\emptyset \subsetneq L'_n \subseteq L_n$, there is $\tau \in \mathcal{B}ch(P)$ such that $\mathcal{I}nf(P|\tau) = \bigcup_{k\in[n]} \bigcup_{l\in L'_k} U_{k,l}$;

ii) $\mathcal{V}' = \{V_g\}_{g\in G}$ and for every $g \in G$, there is $\tau \in \mathcal{B}ch(P)$ such that $\mathcal{I}nf(R|\tau) = V_g$.

Clearly, since $P \preceq P'$ and R is a successful run of M on $T_\emptyset$, $P(\varepsilon) = P'(\varepsilon) \in \mathcal{I}$ holds and, for every infinite path τ in P, there exists an infinite path τ' in P' such that $\mathcal{I}nf(P|\tau) = \mathcal{I}nf(P'|\tau) \in \mathcal{F}$ and for every leaf v in P, $T(v) = b \in \Delta$ holds and the singleton tree b is accepted by $M[\{P'(v)\}/\mathcal{I}]$ (i.e., the tree automaton obtained from M by replacing the set of initial states with $\{P(v)\}$). From the above observations we know that $R'(\varepsilon) \in \mathcal{I}'$ and, for every infinite path π in R', $\mathcal{I}nf(R'|\pi) \in \mathcal{F}'$. This implies that R' is a successful run of M^Δ on $C_\emptyset$.

Conversely, let R' be a successful run of M^Δ on C. By Lemma 3.3.20, there is a partial run P of M on T_Δ, such that

i) $R'(\varepsilon)$ is of the form $(c, 0, \mathcal{U}, \mathcal{V})$, with $\mathcal{U} = \{(P(\varepsilon), \emptyset, \emptyset)\}$ and $\mathcal{V} = \emptyset$;

ii) for every leaf v of P, there is an infinite path π in R' such that $\mathcal{I}nf(R'|\pi)$ is of the form $\{(b, 1, \mathcal{U}', \mathcal{V}')\}$, with $\mathcal{U}' = \{(r_l, U_l, Y_l)\}_{l\in L}$, and there is $l \in L$ such that $T(v) = b$ and $P(v) = r_l$;

iii) for every infinite path τ in P, there is an infinite path π in R' such that $\mathcal{I}nf(R'|\pi) = \{(b_k, x, \mathcal{U}'_k, \mathcal{V}')\}_{k\in[n]}$ holds and either $\mathcal{U}_k = \{(r_{k,l}, U_{k,l}, Y_{k,l})\}_{l\in L_k}$ and $\mathcal{I}nf(P|\tau) = \bigcup_{k\in[n]} \bigcup_{l\in L'_k} U_{k,l}$ hold for some $\emptyset \subsetneq L'_1 \subseteq L_1$, ..., $\emptyset \subsetneq L'_n \subseteq L_n$, or $\mathcal{V} = \{V_g\}_{g\in G}$ and $\mathcal{I}nf(P|\tau) = V_g$ hold for some $g \in G$.

Now, for every leaf v in P, let P_v be a successful run of $M\big[\{P(v)\}/\mathcal{I}\big]$ on the singleton tree $T(v)$. We define the run R of M on $T_\emptyset$ by letting $R(v) = P(v)$ for every internal vertex v of P and $R(v \cdot w) = P_v(w)$ for every leaf v of P and for every vertex w of P_v. Since R' was supposed to be a successful run, we have $R(\varepsilon) \in \mathcal{I}$ and, for every infinite path τ in R, $\mathcal{I}nf(R|\tau) \in \mathcal{F}$. This shows that R is a successful run of M on $T_\emptyset$. ■

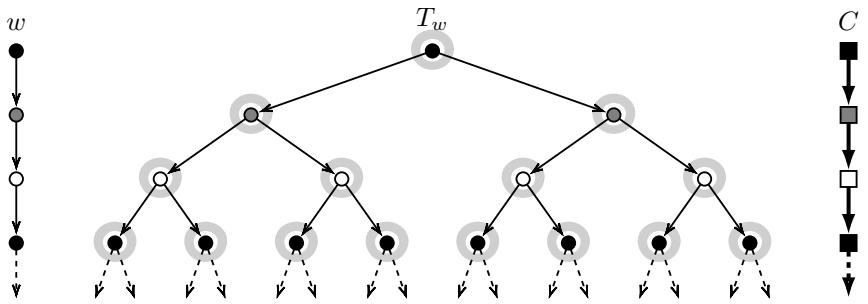

Figure 3.4: An example of application of Corollary 3.3.15.

Example 3.3.21 Here we discuss a simple application of the proposed decision method. Let w be an infinite word over an alphabet Σ. It can be thought of as an expanded linear structure $(\mathbb{N}^+, E, (V_c)_{c\in\Sigma})$, where $(i,j) \in E$ iff $j = i+1$ and $i \in V_c$ iff $w[i] = c$. We then denote by T_w the infinite complete $\{1,2\}$-labeled tree obtained by coloring with $w[i]$ the vertices that belongs to the i-th level of the tree, formally, $T_w(u) = w[|u|+1]$ for every $u \in \{1,2\}^*$ (see Figure 3.4). If the MSO theory of w is decidable, then that of T_w is decidable as well. This can be proved by showing that T_w is nothing but the unfolding of the graph G, labeled over a binary alphabet $\{1,2\}$, that is obtained from w via the MSO-definable interpretation that introduces a new copy of each edge of w (e.g., $\mathcal{I} = \big(\psi(x), \varphi_1(x,y), \varphi_2(x,y), (\varphi_c(x))_{c\in\Sigma}\big)$, where $\psi(x) = \mathit{true}$, $\varphi_1(x,y) = E(x,y)$, $\varphi_2(x,y) = E(x,y)$, and $\varphi_c(x) = V_c(x)$). Thus, by exploiting the MSO-compatibility of unfoldings, one can reduce the model checking problem for T_w to the one for G, which can in its turn be reduced to the one for w. Our method provides an alternative proof of the decidability of the MSO theory of T_w, which is independent from the MSO-compatibility of the unfolding operation. Let $\Delta = \{b\}$ and Π be the factorization of T_w with respect to Δ such that $\mathcal{D}om(\Pi) = \mathcal{D}om(T_w)$ (all edges of Π are labeled with the same symbol b). Given a tree automaton running on T_w, we define a retraction C of T_w with respect to M and Π. There is an obvious computable function Ω that maps each symbol $c \in \Sigma$ to the basic M, Δ-type of the Δ-augmented tree $c\langle b, b\rangle$. Hence, a retraction C can be obtained from an MSO-definable interpretation of w which replaces every color c by the corresponding basic M, Δ-type $\Omega(c)$. The decidability of the MSO theory of R follows from the MSO-compatibility of MSO-definable interpretations. By exploiting Corollary 3.3.15, we can then conclude that the MSO theory of T_w is decidable as well. ■

We conclude the section by giving another fundamental result, which states that one can compute an M,Δ-type t' of T from a given $M^\Delta,\emptyset$-type t of C. Such a result follows from Lemma 3.3.20 and from the following Lemma 3.3.22. It is worth mentioning that, even in the case in which t is the basic $M^\Delta,\emptyset$-type of C, we cannot guarantee the computed t' to be the basic M,Δ-type of T (this explains why we adopted the more general notion of type instead of the notion of basic type).

Lemma 3.3.22 *Let P,P' be two partial runs of M on T_Δ and let R,R' be two runs of M^Δ on C_ε. If P corresponds to R, P' corresponds to R', and $R \preceq R'$, then $P \preceq P'$.*

Proof. Let $R(\varepsilon) = \big(c,0,\{(r,\emptyset,\emptyset)\},\emptyset\big)$. Since $R \preceq R'$, we have $R'(\varepsilon) = R(\varepsilon)$ and hence $P(\varepsilon) = r = P'(\varepsilon)$. Let v be a leaf of P. By Definition 3.3.19, there is an infinite path π in R such that $\mathcal{I}nf(R|\pi)$ is of the form $\big\{(b,1,\mathcal{U},\mathcal{V})\big\}$, with $\mathcal{U} = \big\{(r_l,U_l,Y_l)\big\}_{l\in L}$, and there is $l \in L$ such that $T(v) = b$, $P(v) = r_l$, and $\mathcal{I}mg(P|v) = Y_l$. Since $R \preceq R'$, there exist an infinite path π' in R' such that $\mathcal{I}nf(R'|\pi') = \mathcal{I}nf(R|\pi)$. Again by Definition 3.3.19, since P' corresponds to R', there a leaf v' in P' such that $T(v') = b = T(v)$, $P'(v') = r_l = P(v)$, and $\mathcal{I}mg(P'|v') = Y_l = \mathcal{I}mg(P|v)$. Finally, let τ be an infinite path in P. Since P corresponds to R, one of the following two conditions holds:

1. there is an infinite path π in R, with $\mathcal{I}nf(R|\pi) = \big\{(b_k,0,\mathcal{U}_k,\mathcal{V})\big\}_{k\in[n]}$ and $\mathcal{U}_k = \big\{(r_{k,l},U_{k,l},Y_k)\big\}_{l\in L_k}$ for all $k \in [n]$, and there are $\emptyset \subsetneq L'_1 \subseteq L_1$, ..., $\emptyset \subsetneq L'_n \subseteq L_n$ such that $\mathcal{I}nf(P|\tau) = \bigcup_{k\in[n]} \bigcup_{l\in L'_k} U_{k,l}$;
2. there is an infinite path π in R, with $\mathcal{I}nf(R|\pi) = \big\{(b_k,x,\mathcal{U}_k,\mathcal{V})\big\}_{k\in[n]}$, $x \in \{0,1\}$, and $\mathcal{V} = \big\{V_g\big\}_{g\in G}$, and there is an index $g \in G$ such that $\mathcal{I}nf(P|\tau) = V_g$.

In the former case, since $R \preceq R'$, there is an infinite path π' in R' such that $\mathcal{I}nf(R'|\pi') = \mathcal{I}nf(R|\pi)$. By applying Definition 3.3.19 to P' and R', we know that there is an infinite path τ' in P' such that $\mathcal{I}nf(P'|\tau') = \bigcup_{k\in[n]} \bigcup_{l\in L'_k} U_{k,l} = \mathcal{I}nf(P|\tau)$. The argument for the latter case, is similar. ∎

Theorem 3.3.23 *Given an $M^\Delta,\emptyset$-type of C, one can compute an M,Δ-type of T.*

Proof. Let $\big\{\big(R(\varepsilon),\emptyset,\{\mathcal{I}nf(R|\pi)\}_{\pi\in\mathcal{B}ch(R)}\big) : R \in \mathcal{R}\big\}$ be any $M^\Delta,\emptyset$-type of C, where $\mathcal{R}$ is a complete set of runs of M^Δ on $C_\emptyset$. By Lemma 3.3.20, for every run $R \in \mathcal{R}$ such that $R(\varepsilon) = \big(c,0,\{r,\emptyset,\emptyset\},\emptyset\big)$, there is a corresponding partial run, denoted P_R, of M on T_Δ. Let $\mathcal{P}$ be the set of all partial runs P_R, where R ranges over $\mathcal{R}$. We have to show that $\mathcal{P}$ is a complete set of partial runs of M on T_Δ. Let P' be any partial run of M on T_Δ. By Lemma 3.3.20, there is a partial run P of M on T_Δ and there is a run R' of M^Δ on $C_\emptyset$ such that $P \preceq P'$ and P corresponds to R'. Since $\mathcal{R}$ is a complete set of runs of M^Δ on $C_\emptyset$, there also exists $R \in \mathcal{R}$ such that $R \preceq R'$. Now we know that $P_R \in \mathcal{P}$ corresponds to R and, by Lemma 3.3.22, we obtain $P_R \preceq P \preceq P'$. This shows that $\mathcal{P} = \{P_R : R \in \mathcal{R}\}$ is a complete set of partial runs of M on T_Δ. Therefore, if we denote by $\big\{\big(z_h,\emptyset,\{W_{h,j}\}_{j\in J}\big) : h \in H\big\}$ the $M^\Delta,\emptyset$-type of C, where, for every $h \in H$ and every $j \in J$,

- $z_h = (b_h,x_h,\mathcal{U}_h,\mathcal{V}_h)$,

- $\mathcal{U}_h = \left\{(r_{h,l}, U_{h,l}, Y_{h,l})\right\}_{l \in L_h}$,
- $\mathcal{V}_h = \left\{V_{h,g}\right\}_{g \in G_h}$,
- $W_{h,j} = \left\{(b'_{h,j,k}, x'_{h,j}, \mathcal{U}'_{h,j,k}, \mathcal{V}'_{h,j,k})\right\}_{k \in K_{h,j}}$,
- $\mathcal{U}'_{h,j,k} = \left\{(r'_{h,j,k,l}, U'_{h,j,k,l}, Y'_{h,j,k})\right\}_{l \in L'_{h,j,k}}$,
- $\mathcal{V}'_{h,j} = \left\{V'_{h,j,g}\right\}_{g \in G'_{h,j}}$,

then the set of triples of the form

$$\left(\begin{array}{c} r_{h,l_h}, \\ \left\{\left(b'_{h,j,k_{h,j}}, r'_{h,j,k_{h,j},l}, Y'_{h,j,k_{h,j},l}\right) \;:\; j \in J,\ K_{h,j} = \{k_{h,j}\},\ x'_{h,j} = 1,\ l \in L_h\right\}, \\ \left\{\left(\bigcup_{k \in K_{h,j}} \bigcup_{l \in L''_k} U'_{h,j,k,l}\right) \;:\; j \in J,\ x'_{h,j} = 0,\ \forall\, k \in K_{h,j}.\ \emptyset \subsetneq L''_k \subseteq L'_{h,j,k}\right\} \\ \cup\ \left\{\left(V'_{h,j,g}\right) \;:\; j \in J,\ g \in G'_{h,j}\right\} \end{array}\right)$$

where h ranges over H and satisfies $x_h = 0$, $L_h = \{l_h\}$ (namely, L_h is a set containing a single index l_h), $U_{h,l_h} = \emptyset$, $Y_{h,l_h} = \emptyset$, $G_h = \emptyset$, is an M, Δ-type of T. Finally, notice that such an M, Δ-type can be effectively built on the grounds of the $M^{\Delta}, \emptyset$-type of C. ■

Corollary 3.3.24 *For every Δ-augmented tree T and for every tree automaton M, there is a factorization Π of T with respect to Δ and there is a regular retraction C of T with respect to M and Π. Moreover if T enjoys a decidable acceptance problem, then one can compute a representation of the retraction C.*

Proof. The result follows trivially from Theorem 3.3.23. Given a Δ-augmented tree T and a tree automaton M, we define the factorization Π by letting $\mathcal{D}om(\Pi) = \{\varepsilon\}$. Clearly, any retraction C of T w.r.t M and Π is a singleton tree such that $C(\varepsilon)$ is an M, Δ-type of $T^{+}_{\Pi}[\varepsilon] = T$. By Proposition 3.3.8, one can compute an M, Δ-type of T and hence a retraction C of T with respect to M and Π. ■

3.4 On the effectiveness of the approach

In this section we identify a large class of trees, called reducible trees, which properly include regular trees and enjoy decidable MSO theories. Roughly speaking, the decidability of MSO theories of trees in this class follows from the possibility of recursively reducing their acceptance problem to that for retractions of them. We also prove closure properties regular trees with respect to several natural operations and we then extend such properties to the class of reducible trees. These results, besides showing the robustness of the class of reducible trees, provide a neat framework to reason on retractions of trees and to easily transfer decidability results.

First of all, let us remark the following fact. It might come natural to introduce a class $\mathscr{D}$ consisting of those trees for which one can effectively provide a regular

retraction, for any given tree automaton M. From Corollary 3.3.15, it would follow that each tree in $\mathscr{D}$ enjoys a decidable MSO theory. However, from Corollary 3.3.24, it also follows that $\mathscr{D}$ actually coincides with the class of all trees that enjoy a decidable MSO theory (to see this, consider a tree T with a decidable acceptance problem, define the factorization Π of T such that $\mathcal{D}om(\Pi) = \{\varepsilon\}$ and note that the corresponding retraction is a singleton tree which is computable in virtue of Corollary 3.3.24). Such an attempt to identify a class of decidable trees is not interesting from a combinatorial and algorithmical point of view, since the decision problem has just been postponed to the computation of a regular (singleton) retraction. In order to define a more interesting class of structures, we need to restrict the form of the factorizations in such a way that the types of the factors are always guaranteed to be computable. The natural choice is to enforce the condition that the factors are regular trees. Here, we aim at providing a precise definition of a large class of trees for which the MSO theory can be decided by reducing it to the case of regular trees. From now on, we assume that trees are labeled over $\Lambda = \{a_1, ..., a_k\}$ and colored over $\Sigma = \{c_1, ..., c_m\}$.

Definition 3.4.1 *A* rank 0 tree *is any (Δ-augmented) regular tree. Given a (Δ-augmented) tree and a natural number $n > 0$, we say that T is a* rank n tree *if for every tree automaton M, there are (i) a finite set $\Delta_M \supseteq \Delta$, (ii) a factorization Π_M of T with respect to Δ_M, and (iii) a retraction C_{M,Π_M} of T with respect to M and Π_M such that*

- *for every $u \in \mathcal{D}om(\Pi_M)$, the marked factor $T^+_{\Pi_M}[u]$ is a regular tree,*
- *C_{M,Π_M} is a rank $n-1$ tree.*

A reducible tree *is a* rank n tree *for some $n \geq 0$.*

According to Definition 3.4.1, the decidability of the MSO theories of reducible trees follows from Theorem 3.3.23 and from the decidability of the MSO theories of regular trees, provided that there exists an effective way to compute Δ_M, $(\Pi_M,)$ and C_{M,Π_M} from T for any tree automaton M. Let the *footprint* of a tree T be the minimum amount of information that should be provided to make the reduction from T to its retraction feasible. Such a footprint can be inductively defined as follows.

Definition 3.4.2 *Given a rank 0 tree T, a* footprint *of T is any finite rooted colored graph whose unfolding is isomorphic to T. Given a rank $n > 0$ tree T, a* footprint *of T is any computable function ξ mapping a tree automaton M to a set Δ and a footprint of a rank $n-1$ Δ-labeled tree C which is a retraction of T with respect to M.*

Since the acceptance problem for a rank $n > 0$ tree T and a tree automaton M is reducible to the acceptance problem for a rank $n-1$ retraction C_{M,Π_M} of T and a suitable tree automaton M^Δ, from now on we can focus our attention only on the component of the representation of T that defines a footprint of T. Hereafter, we restrict ourselves to reducible trees which are modeled according to any suitable (extrinsic or intrinsic) representation system (for instance, a system that represents trees as functions from Λ^* to $\Sigma \cup \{\bot\}$) that allows the computation of their footprints. According to such a perspective, we say that the class of rank n trees (resp., reducible

trees) is effectively closed under a family $\mathscr{F}$ of operations whenever the application of any transformation $t \in \mathscr{F}$ results in a tree whose footprint is computable on the grounds of the footprints of the input trees.

Theorem 3.4.3 *Given a (Δ-augmented) reducible tree T and a tree automaton M, one can compute an M, Δ-type of T.*

Proof. Let T be a rank n tree and ξ a footprint of it. We prove the claim by induction n. If $n = 0$, then, by Corollary 3.3.9, one can compute the basic M, Δ-type of T, for any given tree automaton M, on the grounds of the footprint ξ. Let suppose $n > 0$ and let M be a tree automaton, $\Delta_M = \mathcal{P}rj_1(\xi(M))$ and $\xi_M = \mathcal{P}rj_2(\xi(M))$. By definition of footprint, we have that $\Delta \subseteq \Delta_M$ and ξ_M is a footprint of a rank $n-1$ Δ_M-labeled tree C which is a retraction of T with respect to M and Π_M, where Π_M is a suitable factorization with respect to Δ_M. By inductive hypothesis, it immediately follows that one can compute an $M^{\Delta_M}, \emptyset$-type of C and, by Theorem 3.3.23, one can compute an M, Δ_M-type (or, equivalently, an M, Δ-type) of T. ■

Corollary 3.4.4 *Reducible trees enjoy a decidable MSO theory.*

Proof. The result follows trivially from Theorem 3.4.3, Proposition 3.3.10, and Proposition 3.3.7. As a matter of fact, in order to decide the MSO theory of a reducible tree T, it is not necessary to compute an M, Δ-type of T. Indeed, given a footprint ξ of T and a tree automaton M running on T, we can define the sequences $M_0, M_1, ..., M_n$, $\xi_0, \xi_1, ..., \xi_n$, and $\Delta_0, \Delta_1, ..., \Delta_{n-1}$, as follows:

- $M_0 = M$ and $\xi_0 = \xi$,
- $\Delta_i = \mathcal{P}rj_1(\xi_i(M_i))$ for all $0 \leq i < n$,
- $M_{i+1} = M_i^{\Delta_i}$ and $\xi_{i+1} = \mathcal{P}rj_2(\xi_i(M_i))$ for all $0 \leq i < n$.

Notice that, by definition, ξ_n is a footprint of a regular tree, hence a rooted colored graph. By Theorem 3.3.14, we immediately have that T is accepted by M iff the unfolding of ξ_n is accepted by M_n. From Corollary 3.3.2 the latter is a decidable problem and this concludes the proof. ■

It is worth noticing that the inductive definition of rank n tree enforces a hierarchical structure inside the class of reducible trees. It comes natural to wonder whether such a hierarchy is strictly increasing or not. Unlike the case of 'level n residually regular trees' (cf. [83]), we are not able to prove that rank $n+1$ trees strictly include rank n trees, for any $n \geq 0$, nor to find a counterexample to this claim. It remains an open question whether the inductive form of Definition 3.4.1 is really needed to capture all reducible trees.

3.4.1 Closure properties of regular trees

In this section we proof closure properties of regular trees with respect to several natural operations. We start by providing an alternative characterization of regular trees in terms of regular languages of finite words.

Definition 3.4.5 *Given a tree T, we define the* encoding *of T as the language $\{u \cdot c : u \in \mathcal{D}om(T), c = T(u)\}$ of finite words over $\Lambda \cup \Sigma$.*

Let $L \subseteq \Lambda^* \cdot \Sigma$ be a language such that
i) for every word $u \in \Lambda^*$, there is at most one symbol $c \in \Sigma$ such that $u \cdot c \in L$,
ii) for every word $u \in \Lambda^*$, for every color $c \in \Sigma$, and for every prefix u' of u, if $u \cdot c \in L$, then there is a (unique) $c' \in \Sigma$ satisfying $u' \cdot c' \in L$.

It turns out that there is a unique tree T such that L is the encoding of T. This means that there is a bijection between trees and languages of the above form. The following proposition shows that such a correspondence is preserved if we restrict to regular trees and to regular languages.

Proposition 3.4.6 *A tree T is regular iff its encoding is a regular language.*

Proof. We prove the left to right implication. Let fix a generic regular tree T. Further let $G = (S, E_{a\in\Lambda}, C)$ be a finite colored graph and s_0 be vertex of such a graph such that the unfolding of G from s_0 is isomorphic to T. We build a deterministic sequential finite-state automaton M that recognizes exactly the encoding L of T. We define $M = (S \cup \{s_f\}, \Lambda \cup \Sigma, \delta, \{s_0\}, \{s_f\})$ as follows:
- s_f is a fresh state not belonging to S,
- for every $s \in S$ and $a \in \Lambda$, $\delta(s, a) = s'$ iff there is an a-labeled edge from s to s' in G,
- for every $s \in S$ and $c \in \Sigma$, $\delta(s, c) = s'$ iff $c = C(s)$ and $s' = s_f$.

It is routine to check that M recognizes the encoding L of T.
For the converse implication, let L be a regular language that encodes a tree T and let M be a deterministic sequential finite-state automaton recognizing L. Since for every word $u \in L$, $u[|u|]$ is the only symbol from Σ occurring in u, we can assume, without loss of generality, that $M = (S, \Lambda \cup \Sigma, \delta, \{s_0\}, \{s_f\})$ and the unique final state s_f of M is terminal, namely, it is source of no transition of M. We then define the finite colored graph $G = (S \setminus \{s_f\}, E_{a\in\Lambda}, C)$, where
- $(s, s') \in E_a$ iff $\delta(s, a) = s'$,
- $C(s) = c$ iff $\delta(s, c) = s_f$.

Clearly, the unfolding of G from s_0 yields the tree T and this proves that T is regular. ■

Now, we prove some closure properties for the class of regular trees.

Theorem 3.4.7 *The class of regular trees is effectively closed under explicit constructions and tree transductions with bounded lookahead.*

Proof. The closure property for the first operation follows trivially from the definition of regular tree. As for the closure under tree transductions with bounded lookahead, in virtue of Proposition 3.2.14, it is sufficient to prove the closure properties of regular trees with respect to finite-state recolorings with bounded lookahead and regular tree morphisms.
As for finite-state recolorings with bounded lookahead, let T be a regular tree, $N = (S, \Sigma, \Sigma', \Lambda, D, \delta, s_0, \Omega)$ a Mealy tree automaton with bounded lookahead, and T' the

output of N on T. We denote by R the unique run of N on T. From Proposition 3.4.6, we know that there is a deterministic sequential finite-state automaton $M = (S', \Lambda \cup \Sigma, \delta', \{s'_0\}, \{s'_f\})$ that recognizes the encoding of T (note that we assume that M has a single final state, which is source of no transition). By a little abuse of notation, we denote by $\delta'(s', v)$ the (unique) state of M' for which there are $s'_1, ..., s'_n$ such that $\forall\, i \in [n].\ \delta'(s'_{i-1}, v[i]) = s'_{i+1}$. In order to prove that the tree T' is regular, it suffices to provide a deterministic sequential finite-state automaton M' recognizing the encoding of T'. First of all, notice that for every $v \in \mathcal{D}om(T)$, $T^{\downarrow v}|_D$ is uniquely determined by the state $\delta'(s'_0, v)$ of N. Precisely, if, for any given state $s' \in S'$ and for any given finite prefix-closed set $L \subseteq \Lambda^*$, we denote by $T^L_{s'}$ the tree inductively defined by

- $T^{\emptyset}_{s'} = \emptyset$,
- $T^L_{s'} = c\langle T_1, ..., T_k\rangle$, where c is the unique color such that $\delta(s', c) = s'_f$ and for every $i \in [k]$, $T_i = T^{a_i^{-1}\cdot L}_{s'_{a_i}}$, with $s'_{a_i} = \delta(s', a_i)$ and $a_i^{-1} \cdot L = \{w : a_i \cdot w \in L\}$,

then we have $T^{\downarrow v}|_D = T^D_{\delta(s'_0, v)}$. Hence, we can define a finite-state automaton $M' = (S'', \Lambda \cup \Sigma', \delta'', \{s''_0\}, \{s'_f\})$ such that

- $S'' = (S \times S') \cup \{s'_f\}$,
- whenever $\delta\big(s, T^D_{s'}, a\big) = s_a$ and $\delta'(s', a) = s'_a$ hold, then $\delta''\big((s, s'), a\big) = (s_a, s'_a)$ and $\delta''\big((s, s'), \Omega\big(s, T^D_{s'}, a\big) = s_f$ follow,
- $s''_0 = (s_0, s'_0)$.

We now show that M' recognizes the encoding of T'. Given a word $u \in \mathcal{D}om(T)$, let ρ (resp., ρ') be the (unique) run of M (resp., M') on u starting from the initial state. By construction, ρ is exactly the projection of ρ' into the second component, denoted $\mathcal{P}rj_2(\rho')$. Moreover, if π is the access path of u in T, then $R|\pi = \mathcal{P}rj_1(\rho')$. In particular, this implies that $\rho'[|\rho'|] = \big(\rho[|\rho|], R(u)\big)$. Therefore, if we let $s = R(u)$, $s' = \rho[|\rho|]$, and $s'' = \rho'[|\rho'|] = (s, s')$, then we have $u \cdot c \in \mathscr{L}(M)$ iff $\delta'(s, c) = s_f$, iff $\delta''\big(s, \Omega(s', T^D_{s'}\big) = s_f$, iff $u \cdot \Omega(T^D_{s'}, c) \in \mathscr{L}(M')$. This proves that M' recognizes the encoding of T' and hence T' is a regular tree.

As for regular tree morphisms, a proof of the closure of the class of regular trees with respect to regular tree morphisms can be found in Section 4.1 of [34]. ■

3.4.2 Tree substitutions and types

In this section we provide further results about second-order tree substitutions and types of trees. The following proposition relates the types of some trees T, $(B_c)_{c\in\Sigma}$ to the types of the tree $T[\![B_c/c]\!]_{c\in\Sigma}$. By a slight abuse of terminology, given an m-tuple of trees $\bar{T} = (T_1, ..., T_m)$, we say that $\bar{t} = (t_1, ..., t_m)$ is an M, Δ-type of $\bar{T}$ if, for every $i \in [m]$, t_i is an M, Δ-type of T_i.

Proposition 3.4.8 *Let M be a tree automaton and $\bar{t} = (t_c)_{c\in\Sigma}$ a tuple of M, Λ-types. One can compute a tree automaton $M^{\bar{t}}$ and, for any given $M^{\bar{t}}, \emptyset$-type t, an M, Λ-type t' such that, for every tuple of Λ-augmented trees $T, \bar{B} = (B_c)_{c\in\Sigma}$, if t is an $M^{\bar{t}}, \emptyset$-type of T and $\bar{t}$ is an M, Λ-type of $\bar{B}$, then t' is an M, Λ-type of $T[\![B_c]\!]_{c\in\Sigma}$.*

Before proving the proposition, we show some immediate consequences of it and some application examples.

Corollary 3.4.9 *Let M be a tree automaton and σ a regular tree morphism. One can compute a tree automaton M^σ and, for any given $M^\sigma, \emptyset$-type t, an M, Λ-type t' such that, for every Λ-augmented tree T, if t is an $M^\sigma, \emptyset$-type of T, then t' is an M, Λ-type of $\sigma(T)$.*

Proof. The claim follows trivially from Proposition 3.4.8, since any regular tree morphism is specified by some regular trees, whose basic M, Λ-types are computable in virtue of Corollary 3.3.9. ∎

Corollary 3.4.10 *Let M be a tree automaton and $\bar{\gamma}$ a regular tree insertion mapping n-tuples of trees to n'-tuple of trees. For any given M, Λ-type $\bar{t} = (t_1, ..., t_n)$, one can compute an M, Λ-type $\bar{t}' = (t'_1, ..., t'_{n'})$ such that for every n-tuple of Λ-augmented trees $\bar{B} = (B_1, .., B_n)$, if $\bar{t}$ is an M, Λ-type of $\bar{B}$, then $\bar{t}'$ is an M, Λ-type of $\bar{\gamma}(B_1, ..., B_n)$.*

Proof. Again, the claim follows trivially from Proposition 3.4.8, since any tree insertion is specified by a tree T and by a tuple of colors $c_1, ..., c_n$ and since the basic $M^{\bar{t}}, \emptyset$-type of T is computable in virtue of Corollary 3.3.9. ∎

Corollary 3.4.10 allows one to define, for any given regular tree insertion $\bar{\gamma}$ and for any given tree automaton M, a computable function $\bar{\gamma}^M : \mathscr{T}^n_{M,\Lambda} \to \mathscr{T}^{n'}_{M,\Lambda}$ such that, whenever $\bar{t}$ is an M, Λ-type of the tuple of trees $\bar{B}$, then $\bar{\gamma}^M(\bar{t})$ is an M, Λ-type of $\bar{\gamma}(\bar{B})$. In such a case, we say that $\bar{\gamma}^M$ is an *abstraction* of the regular tree insertion $\bar{\gamma}$. In particular, if we denote by $\mathcal{A}_{M,\Lambda,n}$ the set of all abstractions of regular tree insertions and we equip $\mathcal{A}_{M,\Lambda,n}$ with the functional composition, then we can give $\mathcal{A}_{M,\Lambda,n}$ the status of *finite monoid*. Indeed, we know that

i) $\mathcal{A}_{M,\Lambda,n}$ is a finite set, since there are only finitely many M, Λ-types in $\mathscr{T}_{M,\Lambda}$,
ii) from Proposition 3.2.9, tree insertions are closed under functional composition (namely, $\bar{\gamma}' \circ \bar{\gamma}$ is a tree insertion, provided that both $\bar{\gamma}$ and $\bar{\gamma}'$ are tree insertions),
iii) from Theorem 3.4.7, regular tree insertions map regular trees to regular trees, hence regular tree insertions, as well as their abstractions, are closed under functional composition,
iv) there exists a regular tree insertion id mapping an n-tuple of trees to itself and hence there exists an abstraction id^M playing the role of the identity in $\mathcal{A}_{M,\Lambda,n}$.

We now discuss some examples of application of Theorem 3.4.7 and Corollary 3.4.10.

Example 3.4.11 Let $L = (\mathbb{N}, E_a, E_{\bar{a}}, E_\#)$ be the semi-infinite line with a-labeled forward edges, $\bar{a}$-labeled backward edges and #-labeled loops (see the top part of Figure 3.5). Let T_L be the unfolding of L from the leftmost vertex. Since T_L is the unfolding of a relational structure with a decidable MSO theory, T_L has a decidable MSO theory as well. We give an alternative proof of the decidability of the MSO theory of T_L by exploiting Corollary 3.4.10 in order to show that T_L is a rank 1 tree. The bottom part of Figure 3.5 depicts the tree T_L, where, for each $i \in \mathbb{N}$, F_i denotes the unfolding from the rightmost vertex of the subgraph L_i obtained by

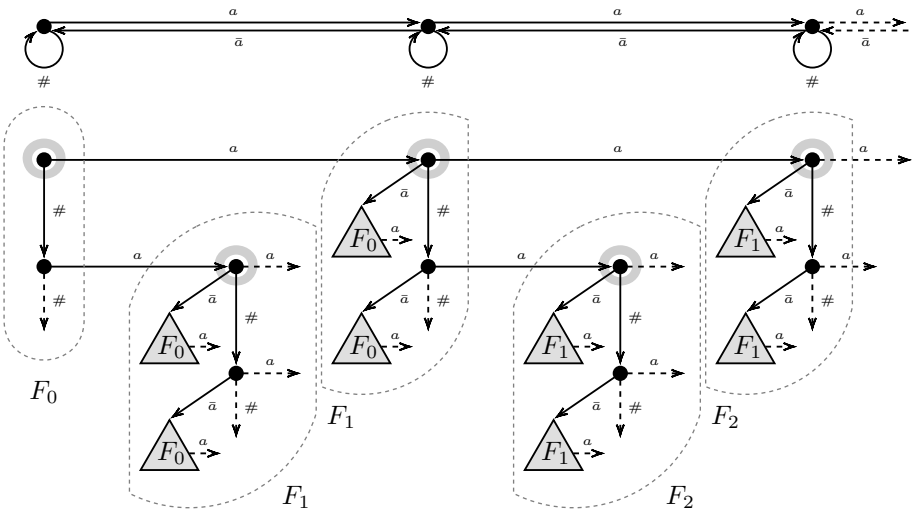

Figure 3.5: The unfolding T_L of the semi-infinite line L.

restricting L to set of vertices $\{0, ..., i-1\}$. The idea is to break up T_L into the basic components $F_0, F_1, F_2, ...$ and then show that each of these components can be obtained by 'inserting' the preceding component into a suitable regular tree. Notice that every vertex v of T_L corresponds to a *unique* path π in L. We denote by $\overrightarrow{\pi}$ the last vertex of L along the path π and we define the factorization Π of T with respect to Λ by letting $\mathcal{D}om(\Pi)$ be the set of all vertices v of T_L such that there is no ancestor $v' \neq v$ of v for which $\overrightarrow{\pi}_v = \overrightarrow{\pi}_{v'}$, where π_v (resp., $\pi_{v'}$) denotes the unique path in L that corresponds to v (resp., v') (the set $\mathcal{D}om(\Pi)$ is represented in Figure 3.5 by circled nodes). We then label the resulting edges of Π with a single symbol a. Even though Π has unbounded degree, by identifying access paths with the same length, we can provide a deterministic Λ-labeled retraction C_L of T_L with respect to Π and any given tree automaton M. Indeed, it is easy to see that for every vertex u of Π at distance i from the root, the marked factor of T_L rooted at u with respect to Π is isomorphic to the tree F_i, where F_0 is the tree depicted in the top left part of Figure 3.6 and $F_{i+1} = \gamma(F_i)$, with γ being the regular tree insertion specified by the tree in the top right part of Figure 3.6 (x denotes the color to be replaced with the input of the tree insertion). Thus, the tree C_L defined by

- $\mathcal{D}om(C_L) = \{a\}^*$,
- $C_L(\varepsilon)$ is the basic M, Λ-type of F_0,
- $C_L(a^{i+1}) = \big(\gamma^M\big)(C_L(a^i))$ for all $i \in \mathbb{N}$

(see the bottom part of Figure 3.6) is a retraction of T_L with respect to M and Π. Moreover, from Corollary 3.4.10, C_L can be thought of as a (memoryless) recoloring of the $\mathcal{A}_{M,\Lambda,1}$-colored tree C defined by

- $\mathcal{D}om(C) = \{a\}^*$,
- $C(\varepsilon)$ is the identity of the monoid $\mathcal{A}_{M,\Lambda,1}$,

- $C(a^{i+1}) = \gamma^M \circ C(a^i)$ for all $i \in \mathbb{N}$

(the recoloring function maps an element $e \in \mathcal{A}_{M,\Lambda,1}$ to the M, Λ-type $e(t)$, where $t = C_L(\varepsilon)$). In its turn, the tree C is regular (this follows from the Pigeonhole Principle since $\mathcal{A}_{M,\Lambda,1}$ is a finite set). From Theorem 3.4.7, it immediately follows that C_L is a regular retraction of T_L and hence T_L is a rank 1 tree, whose footprint can be effectively computed. Finally, from Corollary 3.4.4, we know that T_L enjoys a decidable MSO theory. ■

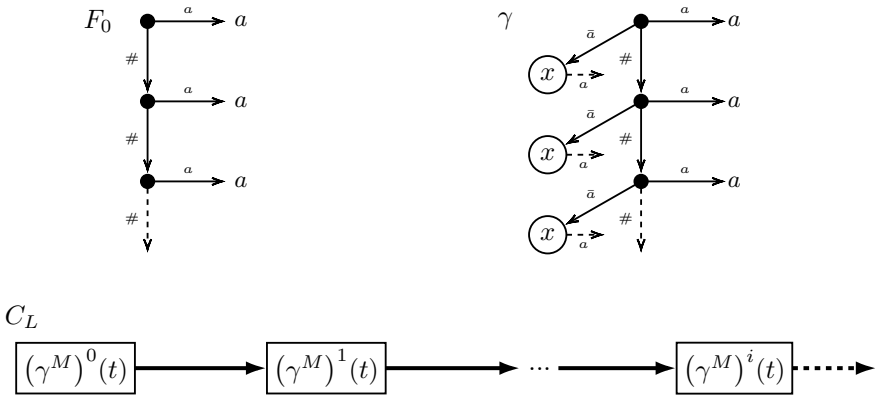

Figure 3.6: Construction of a retraction C_L of T_L.

Example 3.4.12 Let *tow* be the 'tower-of-exponentials' function, recursively defined as follows: $tow(0) = 1$ and $tow(i+1) = 2^{tow(i)}$ for every $i \in \mathbb{N}$. Such a function can be showed to be ultimately periodic with respect to finite monoids (see Section 1.6). We denote by T_{tow} the Λ-labeled Σ-colored tree, where $\Lambda = \{a_1, a_2\}$ and $\Sigma = \{1\}$, whose domain consists of all and only the finite words of the form $v = a_2^i \cdot a_1^j$, with $i \in \mathbb{N}$ and $j \leq tow(i)$ (see the top part of Figure 3.7). In [18], Carayol and Wöhrle show that such a tree enjoys a decidable MSO theory, even thought it does not belong to the Caucal hierarchy. We give an alternative proof of the decidability of the MSO theory of T_{tow} by exploiting the closure of regular trees under the operation of regular exponentiation, which is a particular case of finite-state recoloring (cf. results of Section 3.2.1). We define the factorization Π of T_{tow} with respect to Λ as follows:

- $\mathcal{D}om(\Pi)$ is the set of all vertices u of the form a_2^i, with $i \in \mathbb{N}$ (these vertices are represented by circled nodes in Figure 3.7),
- (u, u') is an a_2-labeled edge of Π iff $u = a_2^i$ and $u' = a_2^{i+1}$, with $i \in \mathbb{N}$.

Now, let $F = 1\langle a_1, a_2\rangle$ and $G = x\langle a_1, a_2\rangle$ be the two Λ-augmented trees depicted in the right part of Figure 3.7 and let γ be the regular tree insertion specified by the tree G (the vertices of G that must be substituted with the input of γ are identified by the color x). Then, we set $F_i = \gamma^{tow(i)}(F)$, for every $i \in \mathbb{N}$. It is easy to see that, for every vertex $u = a_2^i$ of Π, with $i \in \mathbb{N}$, the marked factor of T_{tow} rooted at u

with respect to Π is isomorphic to F_i. From the previous results and from Corollary 3.4.10, we know that, if γ^M is the abstraction of the regular tree insertion γ with respect to M and if t is the basic M, Λ-type of F, then for every $i \in \mathbb{N}$, $\left(\gamma^M\right)^{tow(i)}(t)$ is an M, Λ-type of F_i. This implies that the Λ-labeled $\mathscr{T}_{M,\Lambda}$-colored tree C_{tow} such that

- $\mathcal{D}om(C_{tow}) = \{a_2\}^*$,
- $C_{tow}(a_2^i) = \left(\gamma^M\right)^{tow(i)}(t)$, for all $i \in \mathbb{N}$,

is a retraction of T_{tow} with respect to M and Π. Moreover, C_{tow} can be viewed as a memoryless recoloring of the $\mathcal{A}_{M,\Lambda,1}$-colored tree C such that $\mathcal{D}om(C) = \{a_2\}^*$ and $C(a_2^i) = \left(\gamma^M\right)^{tow(i)}$, for all $i \in \mathbb{N}$ (the recoloring function maps an element $e \in \mathcal{A}_{M,\Lambda,1}$ to the M, Λ-type $e(t)$). In its turn, C is obtained by applying a regular exponentiation to the regular $\mathcal{A}_{M,\Lambda,1}$-colored tree C' such that $\mathcal{D}om(C') = \{a_2\}^*$ and $C'(u) = \gamma^M$ for all $u \in \mathcal{D}om(C')$. From Theorem 3.4.7, it immediately follows that C_{tow} is a regular retraction of T_{tow} and hence T is a rank 1 tree, whose footprint can be effectively computed. This shows that T_{tow} enjoys a decidable MSO theory. ■

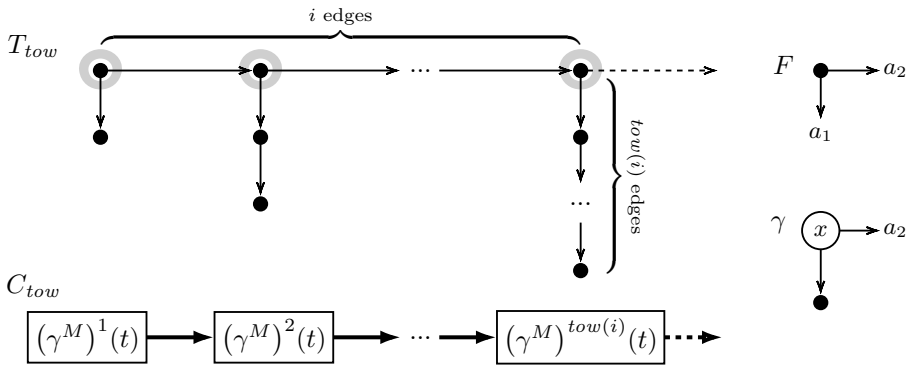

Figure 3.7: An example of a reducible tree outside the Caucal hierarchy.

Example 3.4.13 Let f be a *strictly monotone* function over the natural numbers, namely, such that $i < j$ implies $f(i) < f(j)$ for every $i, j \in \mathbb{N}$, and let g be the function defined by $g(0) = f(0)$ and $g(i+1) = f(i+1) - f(i)$, for all $i \in \mathbb{N}$. We set $\Sigma = \{0, 1\}$ and $\Lambda = \{a_1, a_2\}$ and we denote by T_f the Σ-colored Λ-labeled tree obtained from the infinite complete $\{0\}$-colored Λ-labeled tree by recoloring with 1 every vertex at distance $f(i)$, with $i \in \mathbb{N}$, from the root (see the left part of Figure 3.8, where we represented 0-colored vertices with white nodes and 1-colored vertices with black nodes). We now define a factorization Π of T_f with respect to Λ by letting $\mathcal{D}om(\Pi)$ be the set consisting of the root and all successors of 1-colored vertices and by labeling the resulting edges of Π with a_1. Notice that Π has unbounded degree. However, for every pair of vertices u, u' in Π, if the access paths of u and u' in Π have the same length, then the marked factor of T_f rooted at u and the marked factor

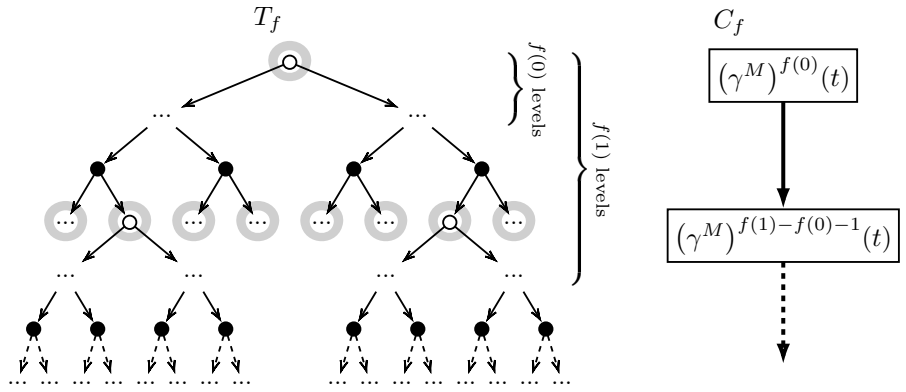

Figure 3.8: Another example of reducible tree.

of T_f rooted at u' are isomorphic. This means that we can identify access paths in Π having the same length, thus showing that, for any given tree automaton M, there is a deterministic retraction of T_f with respect to Π and M. Such a retraction can be obtained as follows. Let $F = 1\langle a_1, a_1\rangle$ and $G = 0\langle x\langle a_1, a_1\rangle, x\langle a_1, a_1\rangle\rangle$ and let γ be the regular tree insertion specified by the tree G (the vertices of G that must be substituted with the input of γ are identified by the color x). Then, we set $F_i = \gamma^{g(i)}(F)$, for every $i \in \mathbb{N}$. It is easy to see that, for every a vertex u of Π at distance $i \in \mathbb{N}$ from the root, the marked factor of T_f rooted at u with respect to Π is isomorphic to F_i. Hence, by following a reasoning similar to the previous example, we know that if t is the basic M, Λ-type of F, then for every tree automaton M, the tree C_f such that

- $\mathcal{D}om(C_f) = \{a_1\}^*$,
- $C_f(a_1^i) = \big(\gamma^M\big)^{g(i)}(t)$ for all $i \in \mathbb{N}$

is a retraction of T_f with respect to M and Π. Moreover, if the function g turns out to be a *ultimately periodic with respect to finite monoids* (as an example, this holds if we choose f to be the tower of exponentials *tow*), then C_f is a regular tree and hence T_f is a rank 1 tree. This would imply that T_f enjoys a decidable MSO theory. ■

The rest of this section is devoted to prove Proposition 3.4.8. We first distinguish between two kinds of substitutions, namely, erasing and non-erasing ones. A second-order tree substitution $T[\![B_c/c]\!]_{c\in\Sigma}$ is said to be *erasing* if there is $c \in \Sigma$ such that B_c is either the empty tree or a tree consisting of a single vertex marked with an element from Λ. In such a case, c (resp., B_c) is said to be an *erased color* (resp., an *erasing term*) of the substitution. Given an erasing second-order tree substitution $T[\![B_c/c]\!]_{c\in\Sigma}$, we then denote by Σ_- the set of the erased colors of Σ and by Σ_+ the set of all other colors. Moreover, if $B_c = c\langle a_1, ..., a_k\rangle$ holds for every $c \in \Sigma_+$ (namely, if the substitution preserves non-erased colors), then we say that the substitution is *shrinking*. It is easy to see that, given a generic second-order tree substitution

$T[\![B_c/c]\!]_{c\in\Sigma}$, we have

$$T[\![B_c/c]\!]_{c\in\Sigma} = T[\![B_c/c]\!]_{c\in\Sigma_-}[\![B_c/c]\!]_{c\in\Sigma_+},$$

where the first substitution in the right-hand side of the equation is shrinking and the second one is non-erasing. Such a property allows us to prove results separately for shrinking substitutions and for non-erasing substitutions and then to extend these results to the general case. In particular, we can split the proof of Proposition 3.4.8 in two parts; the first part (Lemma 3.4.14) deals with shrinking substitutions and the second parts (Lemma 3.4.15) deals with non-erasing substitutions.

Let us consider the case of a shrinking substitution $T[\![B_c/c]\!]_{c\in\Sigma}$. Clearly, for every $c \in \Sigma$, one of the following conditions holds: $B_c = c\langle a_1, ..., a_k\rangle$, $B_c = a_i$ for some $i \in [k]$, or $B_c = \emptyset$. Then, it is easy to see that

i) in the first case ($B_c = c\langle a_1, ..., a_k\rangle$), any M, Λ-type of B_c is a set consisting of triples of the form $\big(s_h, \{(a_i, q_{h,i}, Q_{h,i})\}_{i\in[k]}, \emptyset\big)$, where h ranges over a suitable finite set of indices,
ii) in the second case ($B_c = a_i$), any M, Λ-type of B_c is a set consisting of triples of the form $\big(s_h, \{(a_i, q_h, Q_h)\}, \emptyset\big)$, where h ranges over a suitable finite set of indices,
iii) in the third case ($B_c = \emptyset$), any M, Λ-type of B_c is a set consisting of triples of the form $\big(s_h, \emptyset, \{V_j\}_{j\in J}\big)$, where h ranges over a suitable finite set of indices.

This means that for every tree automaton M and for every pair of shrinking substitutions $T[\![B_c/c]\!]_{c\in\Sigma}$ and $T'[\![B'_c/c]\!]_{c\in\Sigma}$, if for every $c \in \Sigma$, B_c and B'_c have a common M, Λ-type, then for every $c \in \Sigma$, $B_c = B'_c$ holds and hence the two substitutions are indeed the same. Therefore, in the case of a shrinking substitution, Proposition 3.4.8 can be reformulated as follows.

Lemma 3.4.14 *Let M be a generic tree automaton and let $(B_c)_{c\in\Sigma}$ be the replacing terms of a shrinking substitution. One can compute a tree automaton $M^{\bar{B}}$ and for every $M^{\bar{B}}, \emptyset$-type t, an M, Λ-type t' such that for every tree T, if t is an M, Λ-type of T, then t' is an M, Λ-type of $T[\![B_c]\!]_{c\in\Sigma}$.*

Proof. Let $M = (S, \Sigma \cup \{\bot\}, \Lambda, \delta, \mathcal{I}, \mathcal{F})$, $B_\bot = \bot\langle a_1, ..., a_k\rangle$, and $B_a = a\langle a_1, ..., a_k\rangle$ for every $a \in \Lambda$ (equivalently, we can let $B_a = a$ since the marker a can occur only at the leaves of a Λ-augmented tree). We define the tree automaton $M^{\bar{B}} = (Z, \Sigma \cup \Lambda \cup \{\bot\}, \Lambda, \delta', \mathcal{I}', \mathcal{F}')$ as follows:

- $Z = (\Sigma \cup \Lambda\{\bot\}) \times \{0, 1\} \times S \times \mathscr{P}(S)$,
- for every tuple of states (c, x, s, U), $(c_{a_1}, x_{a_1}, s_{a_1}, U_{a_1})$, ..., $(c_{a_k}, x_{a_k}, s_{a_k}, U_{a_k})$ in Z, we have $\big((c, x, s, U), c', ((c_a, x_a, s_a, U_a))_{a\in\Lambda}\big) \in \delta'$ iff one of the following conditions holds
  1. $c' \in \Sigma$, $B_{c'} = c'\langle a_1, ..., a_k\rangle$, $x = 0$, $c_{a_1} = ... = c_{a_k} = c'$, $x_{a_1} = ... = x_{a_k} = 0$, $\big(s, c', (s_a)_{a\in\Lambda}\big) \in \delta$, and $U_a = U \cup \{s_a\}$ for all $a \in \Lambda$,
  2. $c' = \bot$, $x = 0$, $c_{a_1} = ... = c_{a_k} = \bot$, $x_{a_1} = ... = x_{a_k} = 0$, $\big(s, \bot, (s_a)_{a\in\Lambda}\big) \in \delta$, and $U_a = U \cup \{s_a\}$ for all $a \in \Lambda$,
  3. $c' \in \Lambda$, $x = 0$, $c_{a_1} = ... = c_{a_k} = c'$, $x_{a_1} = ... = x_{a_k} = 1$, $s_{a_1} = ... = s_{a_k} = s$, and $U_{a_1} = ... = U_{a_k} = U$,

4. $c' \in \Sigma$, $B_{c'} = a_i$ for some $i \in [k]$, $x = 0$, $c_{a_1} = ... = c_{a_k} = c$, $x_{a_i} = 0$, $x_{a_j} = 1$ for all $j \in [k] \setminus \{i\}$, $s_{a_1} = ... = s_{a_k} = s$, and $U_{a_1} = ... = U_{a_k} = U$,
5. $c' \in \Sigma$, $B_{c'} = \emptyset$, $x = 0$, $c_{a_1} = ... = c_{a_k} = c$, $x_{a_1} = ... = x_{a_k} = 1$, $s_{a_1} = ... = s_{a_k} = s$, and $U_{a_1} = ...U_{a_k} = U$,
6. $c' \in \Sigma \cup \Lambda \cup \{\bot\}$, $x = 1$, $c_{a_1} = ... = c_{a_k} = c$, $x_{a_1} = ... = x_{a_k} = 1$, $s_{a_1} = ... = s_{a_k} = s$, and $U_{a_1} = ... = U_{a_k} = U$

- $\mathcal{I}'$ is any subset of Z (since the initial states are not relevant here),
- $\mathcal{F}'$ is any subset of $\mathscr{P}(Z)$ (since the final states are not relevant here).

Let T be a Λ-augmented tree and let $T' = T[\![B_c/c]\!]_{c\in\Sigma}$. Intuitively, the tree automaton $M^{\bar{B}}$ works on T by simulating the transitions of M on T'. Thus, given a state (c, x, s, U) of $M^{\bar{B}}$, the first component c is used to store the last symbol read by the automaton M, the second component x is used to disable the computation of $M^{\bar{B}}$ on an erased subtree of T, the third component s is used to store the current state of M, and the last component U is used to store the set of states that occur along the access path to the current position in the run of M. As a preliminary remark, notice that the tree T' is empty iff either T is empty or, for every run R of $M^{\bar{B}}$ on T satisfying $R(\varepsilon) = (\bot, 0, s, \emptyset)$ and for every infinite path π in R, $\mathcal{I}nf(R|\pi) = \{(\bot, 1, s, \varepsilon)\}$ holds. Therefore, given an $M^{\bar{B}}, \emptyset$-type of T, we can easily detect whether the tree T' is empty and, in such a case, compute an M, Λ-type of it. From now on, we assume that both T and T' are non-empty full trees (if T or T' are not full trees, then we simply replace T and T' with the corresponding Λ-completions). Now, given a vertex $v \in \mathcal{D}om(T)$, we denote by $\mathit{first}(v)$ the first vertex (if exists) in the subtree of T issued from v which is not erased by the substitution $T[\![B_c/c]\!]_{c\in\Sigma}$. Such a vertex can formally defined as follows:

- $\mathit{first}(v) = v$, if $B_{T(v)} = T(v)\langle a_1, ..., a_k\rangle$,
- $\mathit{first}(v) = \mathit{first}(v \cdot a)$, if $B_{T(v)} = a$,
- $\mathit{first}(v)$ is undefined, if $B_{T(v)} = \emptyset$.

For each vertex $v \in \mathcal{D}om(T')$, we then define the vertex $\overleftarrow{v} \in \mathcal{D}om(T)$ recursively as follows:

- given $v = \varepsilon \in \mathcal{D}om(T')$, we set $\overleftarrow{v} = \mathit{first}(\varepsilon)$,
- given $v \in \mathcal{D}om(T')$ and $\overleftarrow{v} \in \mathcal{D}om(T)$, for each a-successor v' of v in T', we set $\overleftarrow{v}' = \mathit{first}(\overleftarrow{v} \cdot a)$.

It is easy to verify, that for every $v \in \mathcal{D}om(T')$, $T'(v) = T(\overleftarrow{v})$ (simply use induction on the length of the access path of v in T'). We now show that there is a suitable correspondence between the runs of $M^{\bar{B}}$ on T and the partial runs of M on T'.

Let R be a run of $M^{\bar{B}}$ on T such that $P'(\varepsilon) \in \{\bot\} \times \{0\} \times S \times \{\emptyset\}$. By construction of $M^{\bar{B}}$, given any vertex v of R, if $R(v)$ is of the form $(c, 0, s, U)$, then

i) $B_{T(v)} = T(v)\langle a_1, ..., a_k\rangle$ implies, for all $a \in \Lambda$, $R(v \cdot a) = (T(v), 0, s_a, U \cup \{s_a\})$, where $\big(s, T(v), (s_a)_{a\in\Lambda}\big) \in \delta$,
ii) $B_{T(v)} = \emptyset$ implies $R(v \cdot a) = (c, 1, s, U)$ for every $a \in \Lambda$,
iii) $B_{T(v)} = a_i$ implies $R(v \cdot a_i) = (c, 0, s, U)$ and $R(v \cdot a_j) = (c, 1, s, U)$ for all $j \in [k] \setminus \{i\}$,

otherwise, if $R(v)$ is of the form $(c, 1, s, U)$ then for every $a \in \Lambda$, $R(v \cdot a) = (c, 1, s, U)$. We then define the S-colored tree P such that $\mathcal{D}om(P) = \mathcal{D}om(T')$ and for every $u \in$

$\mathcal{D}om(T')$, $P(T') = \mathcal{P}rj_2(R(\overleftarrow{u}))$. Notice that for every $u \in \mathcal{D}om(T')$, $\mathcal{P}rj_1(R(\overleftarrow{u})) = 0$ holds. Thus, it is routine to check that P is a partial run of M on T' satisfying the following properties:

i) $P(\varepsilon) = \mathcal{P}rj_3(R(\overleftarrow{\varepsilon})) = \mathcal{P}rj_3(R(\varepsilon))$,
ii) for every leaf u of P, there is an infinite path π in R such that $\mathcal{I}nf(R|\pi)$ is a singleton of the form $\{(c, 1, s, U)\}$, with $c = T(u)$ and $s = P(u)$, and, $U = \mathcal{I}mg(P|u)$,
iii) for every infinite path τ in P, there is an infinite path π in R such that $\mathcal{I}nf(R|\pi)$ is of the form $\{(c_g, 0, s_g, U)\}_{g\in G}$, with $\{s_g\}_{g\in G} = \mathcal{I}nf(P|\tau)$,
iv) for every infinite path π in R, one of the following conditions holds
   1. $\mathcal{I}nf(R|\pi)$ is a singleton of the form $\{(c, 1, s, U)\}$ and there is a leaf u of P such that $c = T(u)$, $s = P(u)$, and, $U = \mathcal{I}mg(P|u)$,
   2. $\mathcal{I}nf(R|\pi)$ is of the form $\{(c_g, 0, s_g, U)\}_{g\in G}$ and there is an infinite path τ in P such that $\{s_g\}_{g\in G} = \mathcal{I}nf(P|\tau)$.

This shows that the feature of the partial run P of M on T' can be computed on the grounds of the feature of the run R of $M^{\bar{B}}$ on T.

Conversely, let P be a partial run of M on T'. We have to define a corresponding run R of $M^{\bar{B}}$ on T. We recursively define R as follows:

- $R(\varepsilon) = (\perp, 0, P(\varepsilon), \emptyset)$,
- given $v \in \mathcal{D}om(R)$ and $R(v) = (c, x, s, U)$,
   1. if $c' \in \Sigma \cup \{\perp\}$, $B_{c'} = c'\langle a_1, ..., a_k\rangle$, and $x = 0$, then, for all $a \in \Lambda$, we set $R(v \cdot a) = (c', 0, s_a, U \cup \{s_a\})$, where $s_a = P(\overleftarrow{u} \cdot a)$,
   2. if $c' \in \Lambda$ and $x = 0$, then we set $R(v \cdot a) = (c', 1, s, U)$ for all $a \in \Lambda$,
   3. if $c' \in \Sigma$, $B_{c'} = a_i$ for some $i \in [k]$, and $x = 0$, then we set $R(v \cdot a_i) = (c, 0, s, U)$ and $R(v \cdot a_j) = (c, 1, s, U)$ for all $j \in [k] \setminus \{i\}$,
   4. if $c' \in \Sigma$, $B_{c'} = \emptyset$, and $x = 0$, then we set $R(v \cdot a) = (c, 1, s, U)$ for all $a \in \Lambda$,
   5. if $x = 1$, then we set $R(v \cdot a) = R(v)$ for all $a \in \Lambda$.

Clearly, R is a run of $M^{\bar{B}}$ on T. Moreover, it is easy to see that R satisfies the above properties i)–iv). This shows that the feature of R can be computed on the grounds of the feature of P.

The above results show that one can compute an M, Λ-type of T' from a given $M^{\bar{B}}, \emptyset$-type of T. In particular, if we denote by $\{(z_h, \emptyset, \{W_{h,j}\}_{j\in J}) : h \in H\}$ the $M^{\Delta}, \emptyset$-type of T, where for every $h \in H$ and for every $j \in J$,

- $z_h = (c_h, x_h, s_h, U_h)$,
- $W_{h,j} = \left\{(c'_{h,j,l}, x'_{h,j}, s'_{h,j,l}, U'_{h,j})\right\}_{l\in L_{h,j}}$,

then the set of triples of the form

$$\left(\begin{array}{c} s_h, \\ \left\{(c'_{h,j,l_{h,j}}, s'_{h,j,l_{h,j}}, U'_{h,j}) \ : \ j \in J,\ L_{h,j} = \{l_{h,j}\},\ x'_{h,j} = 1\right\}, \\ \left\{\left\{s'_{h,j,l}\right\}_{l\in L_{h,j}} \ : \ j \in J,\ x'_{h,j} = 0\right\} \end{array}\right)$$

where h ranges over H and satisfies $c_h = \perp$, $x_h = 0$, and $U_h = \emptyset$, is an M, Λ-type of T, which can be effectively built on the grounds of the $M^{\bar{B}}, \emptyset$-type of C. ■

Now, we consider the case of non-erasing second-order tree substitutions. We first give some preliminary definitions and results. Then, we prove a result analogous to Proposition 3.4.8, for the case of non-erasing substitutions. Let T' be the tree obtained from a non-erasing substitution $T[\![B_c/c]\!]_{c\in\Sigma}$. For each vertex v of T, we recursively define a set V_v of vertices of T' that correspond to v:

- given $v = \varepsilon$, we set $V_w = \{\varepsilon\}$,
- given $v \in \mathcal{D}om(T')$ and $V_v \subseteq \mathcal{D}om(T')$, for every a-successor v' of v in T', we set $V_{v'} = V_v \cdot \{w \in \mathcal{F}r(B_{T(v)}) : B_{T(v)}(w) = a\}$.

We then define the factorization Π of T' with respect to the set Λ as follows:

- $\mathcal{D}om(\Pi) = \bigcup_{v\in\mathcal{D}om(T)} V_v$,
- (u, u') is an a-labeled edge in Π iff there exist $v, v' \in \mathcal{D}om(T)$ such that $u \in V_v$, $u' \in V_{v'}$, and (v, v') is an a-labeled edge in T.

We say that Π is the factorization *induced* by the substitution $T[\![B_c/c]\!]_{c\in\Sigma}$. Moreover, if u is a vertex of Π, then we denote by $\overleftarrow{u}$ the (unique) vertex of T such that $u \in V_{\overleftarrow{u}}$. From the above definitions it is clear that for every vertex u of Π, the marked factor of T' rooted at u with respect to u is exactly the replacing term $B_{T(\overleftarrow{u})}$.

Lemma 3.4.15 *Let M be a tree automaton and $\bar{t} = (t_c)_{c\in\Sigma}$ a tuple of M, Λ-types. One can compute a tree automaton $M^{\bar{t}}$ and, for any given $M^{\bar{t}}, \emptyset$-type t, an M, Λ-type t' such that, for every tuple of Λ-augmented trees $T, \bar{B} = (B_c)_{c\in\Sigma}$, where each B_c contains at least one internal node, if t is an $M^{\bar{t}}, \emptyset$-type of T and $\bar{t}$ is an M, Λ-type of $\bar{B}$, then t' is an M, Λ-type of $T[\![B_c]\!]_{c\in\Sigma}$.*

Proof. Given the above definitions and results, the proof is rather trivial. Let T be a Λ-augmented tree, let $(B_c)_{c\in\Sigma}$ be a tuple of non-empty non-singleton Λ-augmented trees, and let $T' = T[\![B_c]\!]_{c\in\Sigma}$. We denote by Π the factorization of T' induced by the substitution $T[\![B_c]\!]_{c\in\Sigma}$, by C a retraction of T' with respect to M and Π such that $C(u) = t_{T(\overleftarrow{u})}$ for all $u \in \mathcal{D}om(\Pi)$, and by M^Λ the retraction automaton specified as in Definition 3.3.13. Further assume that, for each $a \in \Lambda$, t_a is an M, Λ-type of the singleton tree a and $t_\perp$ is an M, Λ-type of the empty tree. Then, we define the tree automaton $M^{\bar{t}}$ as follows: the states of $M^{\bar{t}}$ are exactly those of M^Λ, the input alphabet is $\Sigma \cup \Lambda \cup \{\perp\}$, and $\big(z, c, (z_a)_{a\in\Lambda}\big)$ is a transition of $M^{\bar{t}}$ iff $\big(z, t_c, (z_a)_{a\in\Lambda}\big)$ is a transition of M^Λ. Now, since for every vertex u of Π, $B_{T(\overleftarrow{u})}$ coincides with the marked factor of T' rooted at u with respect to Π and since $t_{T(\overleftarrow{u})}$ is an M, Λ-type of such a factor, we have that C can be thought of as the image of T under the recoloring function that maps $c \in \Sigma \cup \Lambda \cup \{\perp\}$ to the M, Λ-type t_c. Therefore, R is a run of M^Λ on C iff R is a run of $M^{\bar{t}}$ on T. This implies that each $M^\Lambda, \emptyset$-type of C can be viewed as an $M^{\bar{t}}, \emptyset$-type of T, and vice versa. Hence, by Theorem 3.3.23, for any given $M^{\bar{t}}, \emptyset$-type t of T, one can compute an M, Λ-type t' of T' and this concludes the proof. ∎

Finally, we can prove Proposition 3.4.8

Proof of Proposition 3.4.8. Let M be a tree automaton, $\bar{t} = (t_c)_{c\in\Sigma}$ a tuple of M, Λ-types, T a Λ-augmented tree, $\bar{B} = (B_c)_{c\in\Sigma}$ a tuple of replacing terms having M, Λ-type $\bar{t}$, and $T'' = T[\![B_c/c]\!]_{c\in\Sigma}$. We denote by Σ_- the set of all colors erased

by the substitution $T[\![B_c/c]\!]_{c\in\Sigma}$ and by Σ_+ the set of all other colors. We can write $T'' = T'[\![B_c/c]\!]_{c\in\Sigma_+}$, where $T' = T[\![B_c/c]\!]_{c\in\Sigma_-}$. Now, by applying Lemma 3.4.15 to the automaton M and to the M,Λ-types $(t_c)_{c\in\Sigma_+}$, we can compute a tree automaton $M' = M^{\bar{t}}$ and, for any given $M',\emptyset$-type t'', an M,Λ-type t''' such that, whenever t'' is an $M',\emptyset$-type of T', then t''' is an M,Λ-type of T''. Similarly, by applying Lemma 3.4.14 to the tree automaton M' (which depends on M and $\bar{t}$ only) and to the erasing terms $(B_c)_{c\in\Lambda_-}$, we can compute a tree automaton $M'' = (M')^{\bar{B}}$ and, for any given $M'',\emptyset$-type t, we can compute an M',Λ-type t' such that, whenever t is an $M'',\emptyset$-type of T, then t' is an M',Λ-type of T'. Finally, by Proposition 3.3.10, given the M',Λ-type t' of T' (which depends on the given $M'',\emptyset$-type t of T), one can compute an $M',\emptyset$-type t'' of T' and hence, from previous observations, an M,Λ-type t''' of T''. We just proved that, given any automaton M and any tuple of M,Λ-types $\bar{t}$, one can compute an automaton M'' and, for any given $M'',\emptyset$-type t, an M,Λ-type t''' such that, whenever $\bar{t}$ is an $M'',\emptyset$-type of T, then t''' is an M,Λ-type of T''. ■

3.4.3 Closure properties of reducible trees

In this section we extend the closure properties provided in Section 3.4.1 to the class of rank n trees, for any $n \in \mathbb{N}$.

Theorem 3.4.16 *For every natural number n, the class of rank n trees is closed under explicit constructions and tree transductions with bounded lookahead.*

In order to prove the theorem for the case of finite-state recolorings with rational lookahead, we need some preliminary results. Precisely, we need relate the types of the factors of a tree T to the types of the factors of the tree T' resulting from T via a finite-state recoloring with bounded lookahead. Since the application of a finite-state recoloring with bounded lookahead requires the inspection of some descendants of the vertices in a tree, we cannot consider the types of the factors of the tree resulting from such an operation independently from the possible contexts where the corresponding factors of the input tree can be embedded. As an example, given a Mealy tree automaton N with bounded lookahead and given some Δ-augmented trees T, $(B_b)_{b\in\Delta}$, the recoloring of the tree $T[B_b/b]_{b\in\Delta}$ via N yields a tree T' where the color of each vertex $v \in \mathcal{D}om(T)$ may depend on the substituting terms $(B_b)_{b\in\Delta}$. Such a situation has to be taken into account when dealing with retractions of trees and, in particular, when relating the types of the factors of a tree T to the types of the corresponding factor of the recoloring of T via N. In the following, given a tuple of (Σ-colored) Δ-augmented trees T, $(B_b)_{b\in\Delta}$ and given a Mealy tree automaton with bounded lookahead $N = (S,\Sigma,\Lambda,D,\delta,s_0,\Omega)$, we denote by $N\big(T[B_b/b]_{b\in\Delta}\big)$ the output of N on $T[B_b/b]_{b\in\Delta}$ and by $N\big(T,(B_b)_{b\in\Delta}\big)$ the (Σ-colored) $\Delta \cup (S\times\Delta)$-augmented tree T'' defined as follows:

- $\mathcal{D}om(T'') = \mathcal{D}om(T)$,
- for every Σ-colored vertex v of T, $T''(v) = N\big(T[B_b/b]_{b\in\Delta}\big)(v)$,
- for every b-colored leaf v of T, with $b \in \Delta$, if B_b is the singleton tree b, then $T''(v) = b$, otherwise $T''(v) = \big(R(v),b\big)$, where R is the (unique) run of N on $T[B_b/b]_{b\in\Delta}$.

Intuitively, the tree $N\big(T,(B_b)_{b\in\Delta}\big)$ is obtained from $N\big(T[B_b/b]_{b\in\Delta}\big)$ by first restricting the domain to the set $\mathcal{D}om(T)$ and then recoloring each leaf v such that $T(v) = b \in \Delta$ and $B_b \neq b$ with a marker of the form (s,b), where s is the state that N reaches at the vertex v of the tree $T[B_b/b]_{b\in\Delta}$.

Lemma 3.4.17 *Let M be a tree automaton, $N = (S,\Sigma,\Lambda,D,\delta,s_0,\Omega)$ a Mealy tree automaton with bounded lookahead, Δ a finite set of markers, and $\Delta' = \Delta \cup (S\times\Delta)$. One can compute a tree automaton M^N (which does not depend on the initial state of N) and, for any given M^N,Δ-type t and for any given tuple of Δ-augmented trees $(B_b)_{b\in\Delta}$, one can compute a Δ-augmented tree B and an M,Δ'-type t' such that for every Δ-augmented tree T, if t is an M^N,Δ-type of T, then $T|_D = B$ and t' is an M,Δ'-type of the tree $N\big(T,(B_b)_{b\in\Delta}\big)$.*

Proof. We first give an intuitive idea of how the automaton M^N behaves. The states of M^N are quadruples consisting of a state s of N, a state s' of M, and two Δ-augmented trees $F,G \in \mathcal{T}^D_\Delta$. The first two components are used to mimic the behavior of N and M, respectively, on the input tree. The third (resp., fourth) component is a guess for the subtree of T (resp., for the subtree of $T[B_b/b]_{b\in\Delta}$) issued from the vertex at the current position and restricted to the domain D (note that we need such two components because M^N must be defined independently from the substituting terms B_b). More formally, given $N = (S,\Sigma,\Lambda,D,\delta,s_0,\Omega)$ and $M = (S',\Sigma,\Lambda,\delta',\mathcal{I},\mathcal{F})$, we define $M^N = (S'',\Sigma,\Lambda,\delta'',\mathcal{I}',\mathcal{F}')$ as follows:

- $S'' = S\times S'\times\mathcal{T}^D_\Delta\times\mathcal{T}^D_\Delta$,
- for every tuple of states $z,(z_a)_{a\in\Lambda}$ of M^N and for every $c\in\Sigma$, $\big(z,c,(z_a)_{a\in\Lambda}\big)\in\delta''$ iff z is of the form (s,s',F,G), with $F(\varepsilon) = G(\varepsilon) = c$, and there are some states $(s'_a)_{a\in\Lambda}$ of M and some Δ-augmented trees $(F_a)_{a\in\Lambda}$ and $(G_a)_{a\in\Lambda}$ such that, for all $a\in\Lambda$, $z_a = \big(\delta(s,G,a),s'_a,F_a,G_a\big)$, $\big(s',\Omega(s,G),(s'_a)_{a\in\Lambda}\big)\in\delta'$, and, for all a and w such that $a\cdot w\in D$, $F_a(w) = F(a\cdot w)$ and $G_a(w) = G(a\cdot w)$ hold

(the initial states and the sets of final states are not relevant here). Now, let T be a Δ-augmented tree and let $(B_b)_{b\in\Delta}$ be a tuple of substituting terms. For the sake of simplicity, we can assume that T is a non-empty full tree whose leaves are colored over Δ (if this is not the case, simply substitute T with its Δ-completion and accordingly extend the transitions of N, M, and M^N over $\bot$-colored vertices). We then denote

- by R the (unique) run of N on $T[B_b/b]_{b\in\Delta}$,
- by R' the restriction of R to $\mathcal{D}om(T)$,
- by T' the Δ'-augmented tree $N\big(T,(B_b)_{b\in\Delta}\big)$, where $\Delta' = S\times\Delta$,
- by F_v the tree $T^{\downarrow v}|_D$, for each $v\in\mathcal{D}om(T)$,
- by G_v the tree $\big(T[B_b/b]_{b\in\Delta}\big)^{\downarrow v}|_D$, for each $v\in\mathcal{D}om(T)$,
- by $\mathbf{F}$ the tree such that $\mathcal{D}om(\mathbf{F}) = \mathcal{D}om(T)$ and for every $v\in\mathcal{D}om(T)$, $\mathbf{F}(v) = F_v$ holds,
- by $\mathbf{G}$ the tree such that $\mathcal{D}om(\mathbf{G}) = \mathcal{D}om(T)$ and for every $v\in\mathcal{D}om(T)$, $\mathbf{G}(v) = G_v$ holds.

Clearly, for every Σ-colored vertex v of T, we have $T'(v) = \Omega\big(R(v),G(v)\big)$ and for every Δ-colored leaf v of T, $T'(v) = \big(R(v),T(v)\big)$. Moreover, it is easy to see that,

if P is a partial run of M on T', then $R'.P.\mathbf{F}.\mathbf{G}$ (i.e., the vertex-wise juxtaposition of R', P, $\mathbf{F}$, and $\mathbf{G}$) is a partial run of M^N on T (simply apply the definition of transition relation of M^N). Conversely, given any partial run P' of M^N on T such that

i) $\mathcal{P}rj_1(P')(\varepsilon) = s_0$,
ii) for every leaf v of P', $\mathcal{P}rj_3(P'(v)) = T^{\downarrow v}$,
iii) for every v of P', $\mathcal{P}rj_4(P'(v)) = B_{T(v)}$,

we have $\mathcal{P}rj_1(P') = R'$, $\mathcal{P}rj_3(P') = \mathbf{F}$, $\mathcal{P}rj_4(P') = \mathbf{G}$, $\mathcal{P}rj_3(P'(\varepsilon)) = T|_D$, and $\mathcal{P}rj_2(P')$ is a partial run of M on T'. This shows that, given any M^N, Δ-type t of T, one can compute the tree $T|_D$ and an M, Δ'-type t' of T' by simply selecting those partial runs P' of M^N on T for which the subtrees in the third and fourth components at the leaves of P' have been correctly guessed. Precisely, let t be a set of the form

$$\left\{ \begin{pmatrix} z_h \\ \{(b_i, r_{h,i}, U_{h,i}) \,:\, i \in I\} \\ \{V_{h,j} \,:\, j \in J\} \end{pmatrix} \,:\, h \in H \right\}$$

and let $b'_{h,i}$ be either the marker b_i or the marker $(\mathcal{P}rj_1(r_{h,i}), b_i)$, depending on whether $\mathcal{P}rj_4(r_{h,i})$ is the singleton tree b_i or not. Then the set

$$\left\{ \begin{pmatrix} \mathcal{P}rj_2(z_h) \\ \{(b'_{h,i}, \mathcal{P}rj_2(r_{h,i}), \mathcal{P}rj_2(U_{h,i})) \,:\, i \in I\} \\ \{\mathcal{P}rj_2(V_{h,j}) \,:\, j \in J\} \end{pmatrix} \,:\, h \in H' \right\}$$

where H' is the set of all indices $h \in H$ such that

i) $\mathcal{P}rj_1(z_h) = s_0$,
ii) for every $i \in I$, $\mathcal{P}rj_3(r_{h,i}) = b_i$,
iii) for every $i \in I$, $\mathcal{P}rj_4(r_{h,i}) = B_{b_i}$,

is an M, Δ'-type of T'. Moreover, we clearly have $T|_D = \mathcal{P}rj_3(z_h)$ and this concludes the proof. ∎

Now, we can prove Theorem 3.4.16.

Proof of Theorem 3.4.16. Theorem 3.4.7 shows closure properties for rank 0 trees. Thus it suffices to prove closure properties for rank n trees, where $n > 0$, with respect to explicit constructions, finite-state recolorings with bounded lookahead, and regular tree morphisms, under the assumption that these properties hold for rank $n-1$ trees. As for the closure under explicit construction, let c be a color from Σ and for all $i \in [k]$, let T_i be a rank $n > 0$ tree with footprint ξ_i. Let M be a generic tree automaton running on T. For every $i \in [k]$, we define $\Delta_i = \mathcal{P}rj_1(\xi_i(M))$ and $\bar{\xi}_i = \mathcal{P}rj_2(\xi_i(M))$. By definition of rank n tree, we know that there exist a factorization Π_i of T_i with respect to Δ_i and a rank $n-1$ tree C_i with footprint $\bar{\xi}_i$ which is a retraction of T_i with respect to M and Π_i. We then define $\Delta = \Lambda \cup \bigcup_{i\in[k]} \Delta_i$ and the factorization Π of $T = c\langle T_1, ..., T_k\rangle$ with respect to Δ as follows:

- $\mathcal{D}om(\Pi) = \{\varepsilon\} \cup \bigcup_{i\in[k]} \{a_i\} \cdot \mathcal{D}om(\Pi_i)$,
- (ε, a_i) is an a_i-labeled edge in Π, for every $i \in [k]$,

- $(a_i \cdot u, a_i \cdot u')$ is a b-labeled edge in Π iff (u, u') is a b-labeled edge in Π_i, for every $i \in [k]$.

Clearly, the tree $C = t\langle C_1, ..., C_k\rangle$, where t is the basic M, Δ-type of $c\langle a_1, ..., a_k\rangle$, is a retraction of T with respect to M and Π. By inductive hypothesis, C is a rank $n-1$ tree and one can compute a footprint $\bar{\xi}$ of C on the grounds of the footprints $\bar{\xi}_1, ..., \bar{\xi}_k$ of $C_1, ..., C_k$. Hence, the function ξ that maps a tree automaton M to the pair $(\Delta, \bar{\xi})$ is a footprint of T.

As for the closure under finite-state recoloring with bounded lookahead, let $N = (S, \Sigma, \Lambda, D, \delta, s_0, \Omega)$ be a Mealy tree automaton with bounded lookahead, T be a rank n tree with footprint ξ, R be the run of N on T, and T' the output of N on T. Further let $M = (S', \Sigma, \Lambda, \delta', \mathcal{I}, \mathcal{F})$ be a generic tree automaton running on T'. We can assume, without loss of generality, that $D = \bigcup_{0 \leq i \leq l} D^i$, where $D_i = \Lambda^i$ and l is a suitable natural number (if this is not the case, simply extend the lookahead trees appearing in δ and Ω). We denote by M^N be the tree automaton specified in Lemma 3.4.17 (notice that such an automaton is computable on the grounds of M and N). We then define $\Delta = \mathcal{P}rj_1(\xi(M^N))$ and $\bar{\xi} = \mathcal{P}rj_2(\xi(M^N))$. By definition of rank n tree, we know that there exist a factorization Π of T with respect to Δ and a rank $n-1$ tree C with footprint $\bar{\xi}$ which is a retraction of T with respect to M^N and Π. We can build a factorization Π' and a restraction C' of T' on the grounds of Π and C. Let $\Delta' = \Delta \cup (S \times \Delta)$ and let Π' be the factorization of T' with respect to Δ' defined as follows:

- $\mathcal{D}om(\Pi') = \mathcal{D}om(\Pi)$,
- (u, u') is a $\big(R(u'), b\big)$-labeled edge of Π' iff (u, u') is a b-labeled edge of Π.

We now compare the marked factors of T (with respect to Π) with those of T' (with respect to Π'). For every $u \in \mathcal{D}om(\Pi)$, we define the following shorthands:

- F_u is the marked factor of T rooted at u with respect to Π,
- F'_u is the marked factor of T' rooted at u with respect to Π',
- A_u is the tree $F_u|_D$,
- $B_u^{(i)}$ is the tree $T^{\downarrow u}|_{D_i}$, for every $0 \leq i \leq n$,
- $N_{R(u)}$ is the Mealy tree automaton obtained from N by replacing the initial state s_0 with $R(u)$.

First of all, note that for every $u \in \mathcal{D}om(\Pi)$,

$$T^{\downarrow u} = F_u\big[T^{\downarrow u \cdot w}/w\big]_{w \in \mathcal{F}r(F_u)}.$$

Since every leaf of F_u is a proper descendant of the root of F_u, the tree $B_u^{(n)}$ can be recursively built as follows:

i) $B_u^{(1)} = A_u|_{D_1}$,

ii) $B_u^{(i+1)} = \Big(A_u\big[B_{u \cdot w}^{(i)}/w\big]_{w \in \mathcal{F}r(A_u)}\Big)\Big|_{D^{(i+1)}}$,

(here we assume that $B_{u \cdot w}^{(i)}$ is the singleton tree b whenever $u \cdot w \notin \mathcal{D}om(\Pi)$, $A_u(w) = b$, and $b \in \Delta$; notice that, in such a case, $u \cdot w$ is a Δ-colored leaf of T). In virtue of Lemma 3.4.17, for each vertex u of Π, the tree A_u can be computed on the grounds of the M^N, Δ-type $C(\vec{u})$, where $\vec{u}$ is the (unique) vertex of C corresponding to u. This also implies that the tree $B_u^{(n)} = T^{\downarrow u}|_D$ can be effectively built on the grounds of the

tree $C^{\downarrow\vec{u}}|_{D'}$, where $D' = \bigcup_{0\leq i\leq l}\Delta^i$. Therefore, we can denote by g a *computable* function such that, for every b-labeled edge (u,u') of Π, $g\big(b, C^{\downarrow\vec{u}}|_{D'}\big) = T^{\downarrow u'}|_D$ and, for every leaf u of Π, $g\big(b, C^{\downarrow\vec{u}}|_{D'}\big) = b$ (note that such a function is well defined, since the tree $T^{\downarrow u'}$ depends only on b and on $C^{\downarrow\vec{u}}|_{D'}$). Now, it routine to check that for every $u \in \mathcal{D}om(\Pi)$,

$$F'_u = N_{R(u)}\Big(F_u, \big(g(b,U)\big)_{b\in\Delta'}\Big),$$

where $U = C^{\downarrow\vec{u}}|_{D'}$. By exploiting Lemma 3.4.17, one can compute an M,Δ'-type of F'_u given the state $R(u)$ and the tree $C^{\downarrow\vec{u}}|_{D'}$. We denote such an M,Δ'-type by $C'(\pi)$, where π is the access path of u in Π'. Since $R(u) = \mathcal{P}rj_2(\pi[|\pi|])$ holds, $C'(\pi)$ depends only on the label of the last edge of π and on the subtree of C issued from $\vec{u}$. This means that C' can be thought of as a *deterministic* Δ'-labeled tree. Moreover, C' is clearly a retraction of T' with respect to M and Δ'. We now show that C' can be obtained as the output of a suitable tree transducer with bounded lookahead running on C. The transducer is of the form $M' = (S'', \Sigma', \Delta', \Lambda', \lambda, D'', \delta'', s'_0)$, where

- $S'' = S$,
- $\Sigma' = \Sigma_{in} \cup \Sigma_{out}$, where Σ_{in} is the set of all M^N,Δ-types and Σ_{out} is the set of all M,Δ'-types,
- $\Lambda' = \Lambda_{in} \cup \Lambda_{out}$, where $\Lambda_{in} = \Delta$ and $\Lambda_{out} = \Delta'$,
- λ maps an element $b' = (s,b) \in \Delta'$, with $s \in S$ and $b \in \Delta$, to the marker $\mathcal{P}rj_2(b') = b \in \Delta$ (the values of λ for the elements of Δ are not relevant),
- $D' = \bigcup_{0\leq i\leq l+1}\Delta^i$,
- δ'' maps a pair of the form (s,U), where $s \in S$ and U is a Λ_{in}-labeled Σ_{in}-colored tree whose domain is a subset of D', to the tuple $\big(Y, (\mathcal{P}rj_2(b'))_{b'\in\Delta'}\big)$ such that, if $U(\varepsilon)$ is an M^N,Δ-type of the form

$$\left\{\begin{pmatrix} z_h \\ \big\{\big(b_i, r_{h,i}, U_{h,i}\big) \,:\, i\in I\big\} \\ \big\{V_{h,j} \,:\, j\in J\big\}\end{pmatrix} \,:\, h\in H\right\},$$

  then Y is the Λ_{out}-labeled Σ_{out}-colored Δ'-augmented tree such that
  - i) $\{\varepsilon\} \subseteq \mathcal{D}om(Y) \subseteq \{\varepsilon\} \cup \Delta'$,
  - ii) $Y(\varepsilon)$ is the M,Δ'-type t' calculated as in Lemma 3.4.17, where we let $t = U(\varepsilon)$ and $B_b = g(b,U)$ for every $b \in \Delta'$,
  - iii) given $b' \in \Delta'$, if there is $i \in I$ such that $b_i = \mathcal{P}rj_1(b')$, then $b' \in \mathcal{D}om(Y)$ and $Y(b') = b'$ follow, otherwise $b' \notin \mathcal{D}om(Y)$,
- $s'_0 = s_0$.

We let the reader check that M' generates exactly C' when running on C (to this end, it is sufficient to show, by exploiting induction on $n \in \mathbb{N}$, that for every $i \in \mathbb{N}$, the trees $(M')^i(s_0, C)$ and C' coincide on those vertices which are at distance less than i from the roots). Since C belongs to the class of rank $n-1$ tree, which was supposed to be closed under tree transductions with bounded lookahead, we immediately have that C' is a rank $n-1$ tree and one can compute a footprint $\bar{\xi}'$ of C' on the grounds of the footprint $\bar{\xi}$ of C. Hence, the function ξ' that maps a tree automaton M to the pair $(\Delta', \bar{\xi}')$ is a footprint of T'.

As for the closure of rank n trees under regular tree morphisms, let σ be a regular tree morphism specified by the tuple $(B_c)_{c\in\Sigma}$, M a tree automaton over the alphabet Σ, T a rank n tree with footprint ξ, and $T' = \sigma(T)$. We have to show that T' is a rank n tree and build a footprint of T' on the grounds of σ and the footprint of T. To this end, we assume that rank $n-1$ trees are closed under regular tree morphisms. We denote by M^σ the (computable) tree automaton as defined in Corollary 3.4.9 with respect to M and σ. Given any finite set Δ disjoint from Σ, we identify $2+|\Delta|$ categories of Δ-augmented trees:

1. trees whose image under σ is the empty tree,
2. trees whose image under σ is the singleton tree b, for every choice of b in Δ,
3. trees whose image under σ is a non-empty non-singleton tree.

Without loss of generality, we can assume that for any arbitrary finite set Δ disjoint from Σ, the tree automaton M^σ is capable of 'distinguishing' between the above categories of trees, namely, if F and F' are two Δ-augmented trees having a common M^σ, Δ-type t, then either $F = F'$ holds or both F and F' are non-empty non-singleton trees. Notice that, if this were not the case, we can reason on a suitable tree automaton $\bar{M}$ whose states are all and only the pairs (s, c), with s being a state of M and $c \in \Sigma \cup \{\perp\}$, and whose transitions are all and only the tuples $\big((s,c), c', \big((s_a, c_a)\big)_{a\in\Lambda}\big)$ such that $c = c'$ and $\big(s, c, (s_a)_{a\in\Lambda}\big)$ is a transition of M. Clearly, such an automaton $\bar{M}$ is capable of distinguishing between the empty tree, the singleton tree b, where b ranges over Δ, and a non-empty non-singleton tree. As a consequence, the automaton $\bar{M}^\sigma$ as well would be able to distinguish between the above categories of trees. Moreover, given an $\bar{M}, \Delta$-type t of a Δ-augmented tree F, one can compute an M, Δ-type t' of F by simply projecting each state appearing in t into its first component. This shows that there exist two computable functions f and g such that, whenever t is an M^σ, Δ-type of a Δ-augmented tree F, then $f(t)$ is an M, Δ-type of $\sigma(F)$ and $g(t) = 0$ if $\sigma(F) = \emptyset$, $g(t) = b$ if $\sigma(F) = b$, $g(t) = 1$ if $\sigma(F)$ is a non-empty non-singleton tree. Now, we define $\Delta = \mathcal{P}rj_1(\xi(M^\sigma))$ and $\bar{\xi} = \mathcal{P}rj_2(\xi(M^\sigma))$. By definition of rank n tree, we know that there exist a factorization Π of T with respect to Δ and a rank $n-1$ tree C with footprint $\bar{\xi}$ which is a retraction of T with respect to M^σ and Π. As usual, given $u \in \mathcal{D}om(\Pi)$, we denote by F_u the marked factor of T (with respect to Π) rooted at u. We have to define a suitable factorization Π' of T' in such a way that each marked factor F' of T' can be viewed as the image of a corresponding marked factor F of T under the regular tree morphism σ. Given a vertex v of T, we define the set D_v consisting of all vertices of T' that correspond to v under σ:

- given $v = \varepsilon$, we set $D_v = \{\varepsilon\}$,
- given $v \in \mathcal{D}om(T)$ and the set D_v, we define, for each a-successor v' of v, $D_{v'} = D_v \cdot \{w : w \in \mathcal{F}r(B_{T(v)}), B_{T(v)}(w) = a\}$.

We then define the factorization Π' of T' with respect to Δ as follows:

- $\mathcal{D}om(\Pi') = \bigcup_{v\in\mathcal{D}om(\Pi)} D_v$,
- for every $u, u' \in \mathcal{D}om(\Pi')$, (u, u') is a b-labeled edge in Π' iff u' is a descendant of u in T' and there is a b-labeled edge (v, v') in Π such that $u \in D_v$ and $u' \in D_{v'}$.

Note that, given a vertex $u \in \mathcal{D}om(\Pi')$, there exists a vertex $v \in \mathcal{D}om(\Pi)$ such that

$u \in D_v$. Moreover, for every pair of vertices $v, v' \in \mathcal{D}om(\Pi)$, either $D_v \cap D_{v'} = \emptyset$ or $D_v = D_{v'}$ (in the latter case v is an ancestor or a descendant of v' and $B_{T(v)}$ or $B_{T(v')}$ is a singleton tree). Thus, for every vertex u of Π', we can define a corresponding vertex $\bar{u}$ in Π as the *least* vertex (according to the ordering given by the successor relation of Π) among all vertices v of Π such that $u \in D_v$ (note that all such vertices lie on the same path). We let the reader check that for every vertex u of Π', the marked factor F'_u of T' rooted at u with respect to Π' is isomorphic to the tree $\sigma(F_{\bar{u}})$. Now, let assume $\Delta = \{b_1, ..., b_h\}$. For any given M^σ, Δ-type t, we define the Δ-augmented tree B'_t as follows:

- if $g(t) = 0$, then B'_t is the empty tree,
- if $g(t) = b$, with $b \in \Delta$, then B'_t is the singleton tree b,
- if $g(t) = 1$, then B'_t is the tree $t'\langle b_1, ..., b_h\rangle$, where $t' = f(t)$.

The tree B'_t can be effectively built on the grounds of the M^σ, Δ-type t. We now denote by σ' the regular tree morphism that replaces each M^σ, Δ-type t with the Δ-augmented tree B'_t and we define the tree $C' = \sigma'(C)$. We prove that C' is a retraction of T' with respect to M and Π'. Notice that σ' is somehow similar to a shrinking substitution, since it can only erase vertices or substitute colors. Thus, by using the same argument of the proof of Lemma 3.4.14, given $u \in \mathcal{D}om(C)$, we can denote by $\mathit{first}(u)$ the first vertex (if any) in the subtree of C issued from u which is not erased by the substitution $\sigma'(C)$. Formally:

- $\mathit{first}(u) = v$, if $g(C(u)) = 1$,
- $\mathit{first}(u) = \mathit{first}(u \cdot b)$, if $g(C(u)) = b$, with $b \in \Delta$,
- $\mathit{first}(u)$ is undefined, if $g(C(u)) = 0$.

For each vertex u of C', we then define a corresponding vertex $\overleftarrow{u}$ of C as follows:

- given $u = \varepsilon$, we set $\overleftarrow{u} = \mathit{first}(\varepsilon)$,
- given $u \in \mathcal{D}om(C')$ and $\overleftarrow{u} \in \mathcal{D}om(C)$, for each b-successor u' of u in C', we set $\overleftarrow{u}' = \mathit{first}(\overleftarrow{u} \cdot b)$.

By exploiting induction on $|u|$ and the definition of σ', it is easy to see that $C'(u) = f(C(\overleftarrow{u}))$ holds, for all $u \in \mathcal{D}om(C')$. We now show that there is a suitable correspondence between the vertices of C' and the vertices of Π'. Let v' be a vertex of Π' and u' be the sequence of labels along the access path of v' in Π'. We show that u' identifies a vertex in C' such that $C'(u')$ is an M, Δ-type of $F'_{v'}$. Let $v = \bar{v}'$ (recall that $F'_{v'} = \sigma(F'_{\bar{v}})$) and let u be the vertex of C that corresponds to v, namely, the (unique) vertex u such that the sequence of labels along the access path of u in C and the sequence of labels along the access path of v in Π coincide (this implies that $f(C(u))$ is an M, Δ-type of $F'_{v'}$). It is easy to verify, by exploiting induction on $|u'|$ and the definition of σ', that $\overleftarrow{u}' = u$, from which it follows that $C'(u')$ is an M, Δ-type of $F'_{u'}$. Conversely, for every vertex $u' \in \mathcal{D}om(C')$, $u = \overleftarrow{u}'$ is a vertex of C such that $C'(u') = f(C(u))$. Since C is a retraction of T with respect to Π, there is at least one vertex v in Π that corresponds to u and such that $C(u)$ is an M^σ, Δ-type of F_u. Given such a vertex v, we let $v' = \bar{v}$ and we notice that $F'_{v'} = \sigma(F_v)$ holds. This implies that $C'(u')$ is an M, Δ-type of $F'_{v'}$. In this way, we proved that C' is a retraction of T' with respect to M and Π'. Since rank $n-1$ trees are closed under regular tree morphisms, one can compute a footprint $\bar{\xi}'$ of C' on the grounds of the

footprint $\bar{\xi}$ of C. Hence the function ξ' that maps a tree automaton M to the pair $(\Delta', \bar{\xi}')$ is a footprint of T' and this concludes the proof. ■

We strongly believe that the proof of closure of rank n trees under finite-state recolorings with bounded lookahead could be improved to cope with the more general case of finite-state recolorings with *rational* lookahead. As a matter of fact, if such a conjecture were proved, then, from the expressiveness results given in Section 3.2, it would follow that for every natural number n, the class of rank n trees is closed under tree transductions with rational lookahead. In Section 3.5.2, we shall mention other relevant implications of such a conjecture.

3.5 Application examples

In this section we compare the expressiveness of our approach with that of other frameworks proposed in the literature. In particular, we show that the class of reducible trees includes several interesting decidable structures like the unfoldings of the deterministic context-free graphs introduced by Muller and Schupp in [87] (hence the algebraic trees of Courcelle [35]), the so-called 'tree generators' for the levels of the Caucal hierarchy, and trees embedding the downward, upward, and totally unbounded ω-layered structures [77, 78, 81].

3.5.1 Unfoldings of context-free graphs

Here we prove that the unfolding of a deterministic context-free graph is a reducible (rank 1) tree. In [87], context-free graphs have been defined on the grounds of the notion of end-isomorphism. We briefly recall such a notion and we refer to [87] for further details. We say that a graph is *weakly connected* (*connected* for short), if for every pair of vertices v, v', there is a sequence $v_1, v_2, ..., v_n$ such that $v_1 = v$, $v_n = v'$, and, for all $1 \leq i < n$, v_i and v_{i+1} are adjacent vertices (note that, in general, this does not imply that there is a directed path from v to v' or from v' to v). Hereafter, we shall restrict our attention to (weakly) connected graphs having uniformly bounded out-degree.

Given a graph $G = (S, (E_a)_{a \in \Lambda})$ with a source vertex v_0, we inductively define $G^{(n)}$ as follows:

- $G^{(0)}$ is the empty graph,
- $G^{(1)}$ is the subgraph of G induced by $\{v_0\}$ (the subgraph of G induced by V is simply the graph consisting of all vertices from V and all edges of G connecting two vertices in V),
- $G^{(n+1)}$ is the subgraph of G induced by the set consisting of all vertices from $\mathcal{D}om(G^{(n)})$ plus the vertices in G that are adjacent to some vertices in $G^{(n)}$.

Intuitively, $G^{(n)}$ is the subgraph of G induced by the set of all vertices reachable from v_0 by a 'path' of length at most $n-1$, which can traverse edges in both directions. Note that $G \setminus G^{(n)}$ (i.e., the subgraph induced by $\mathcal{D}om(G) \setminus \mathcal{D}om(G^{(n)})$ may be an unconnected graph. We say that C is a *level n component of G* if it is a connected

component (i.e., a maximally connected subgraph) of $G \setminus G^{(n)}$. Given a vertex $v \in \mathcal{D}om(G)$, we say that v is a *frontier point* of a level n component C if $v \in \mathcal{D}om(C)$ and $v \notin G \setminus G^{(n+1)}$; otherwise, we say that v is an *internal point*. For instance, there is a unique level 0 component (i.e., G itself) which contains a unique frontier point (i.e., v_0). For any $v \in \mathcal{D}om(G)$, if n is the maximum integer such that there is a level n component of G containing v, then we call n the *level* of v and we denote by $G^{\downarrow v}$ the (unique) component of $G \setminus G^{(n)}$ containing v. We further define $F_G^{\downarrow v}$ to be the set of all the frontier points of $G^{\downarrow v}$. Since G was supposed to have bounded degree, $F_G^{\downarrow v}$ is a finite set. Now, we give the notion of end-isomorphism. We say that Ψ is an *end-isomorphism* between $G^{\downarrow v}$ and $G^{\downarrow v'}$ if Ψ is a bijection from $\mathcal{D}om\big(G^{\downarrow v}\big)$ to $\mathcal{D}om\big(G^{\downarrow v'}\big)$ such that (i) (w, w') is an a-labeled edge in $G^{\downarrow v}$ iff $\big(\Psi(w), \Psi(w')\big)$ is an a-labeled edge in $G^{\downarrow v'}$ and (ii) Ψ maps $F_G^{\downarrow v}$ onto $F_G^{\downarrow v'}$ (that is $\Psi\big(F_G^{\downarrow v}\big) = \Psi\big(F_G^{\downarrow v'}\big)$. Two components $G^{\downarrow v}$ and $G^{\downarrow v'}$ are said to be *end-isomorphic* if there is an end-isomorphism between them; we denote this by writing $G^{\downarrow v} \equiv G^{\downarrow v'}$. Now, we can give the definition of context-free graph.

Definition 3.5.1 *A graph G is said to be* context-free *(or* end-regular*, or* pushdown transition graph*) if there are only finitely many non-end-isomorphic components $G^{\downarrow v}$, for any $v \in \mathcal{D}om(G)$.*

In [87], Muller and Schupp provide an equivalent characterization of context-free graphs in terms of configuration graphs of pushdown systems (i.e., pushdown automata devoid of acceptance conditions), where the vertices are given by the configurations (i.e., the pairs consisting of a control state and a sequence of stack symbols) that are reachable from the initial configuration and the edge labels are given by the input symbols $a_1, ..., a_k$, the special symbol ε (denoting a transition that consumes no input symbol), and the corresponding reversed copies $a_1^{-1}, ..., a_k^{-1}, \varepsilon^{-1}$. As an example, Figure 3.9 depicts a (deterministic) pushdown system over the input alphabet $\Lambda = \{1, 2, 3\}$ and its configuration graph.

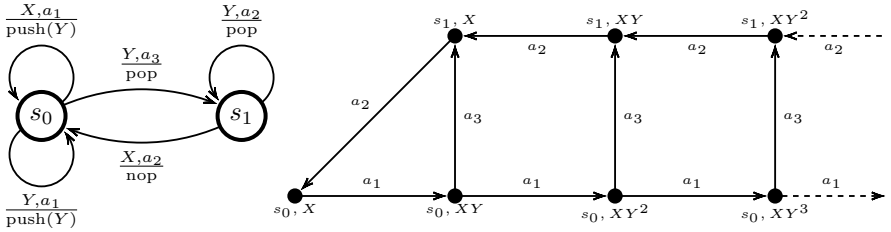

Figure 3.9: A pushdown system and its configuration graph.

Theorem 3.5.2 (Muller and Schupp [87]) *A graph G is context-free iff it is the configuration graph of a push-down system.*

In the sequel we disclose other relevant properties of context-free graphs. In order to simplify the presentation, from now on, we let G be a fixed deterministic context-free graph and T be the unfolding of G from a distinguished source vertex v_0. We

then denote by $\left[G^{\downarrow v}\right]_{\equiv}$ the equivalence class consisting of all and only the graphs which are end-isomorphic to $G^{\downarrow v}$. By definition of context-free graph, the set of all such equivalence classes is finite and hence there exist a positive natural number m and a finite set of *representatives* $C_1, ..., C_m$ such that for every $v \in \mathcal{D}om(G)$, there is $i \in [m]$ (and vice versa, for every $i \in [m]$, there is $v \in \mathcal{D}om(G)$) satisfying $G^{\downarrow v} \equiv C_i$. We further denote by F_i the set $\{f_{i,1}, ..., f_{i,l_i}\}$ of the frontier points of C_i.

From the definition of component of a graph G, it immediately follows that, for every edge (v, v') in G, one of the following conditions holds:

1. v and v' are frontier points of the same component $G^{\downarrow v} = G^{\downarrow v'}$; in this case, we say that (v, v') is an edge of *type* 1;
2. v' is an internal point of $G^{\downarrow v}$, hence $G^{\downarrow v'}$ is the connected component of the subgraph of $G^{\downarrow v}$ induced by $\mathcal{D}om(G^{\downarrow v}) \setminus F_G^{\downarrow v}$ (i.e., the subgraph obtained from $G^{\downarrow v}$ by removing the frontier points) that contains v'; in such a case, we say that (v, v') is an edge of *type* 2;
3. v is an internal point of $G^{\downarrow v'}$, hence $G^{\downarrow v}$ is the connected component of the subgraph of $G^{\downarrow v'}$ induced by $\mathcal{D}om(G^{\downarrow v'}) \setminus F_G^{\downarrow v'}$ that contains v; in such a case, we say that (v, v') is an edge of *type* 3.

The same classification is preserved for each representative C_i. As an example, Figure 3.10 represents a possible situation for a component of a context-free graph (black-colored circles denote frontier points, white-colored circles denote internal points, and numbers denote the types of the edges).

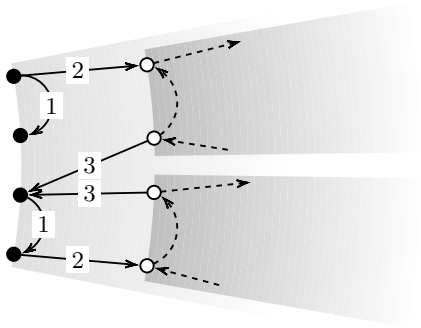

Figure 3.10: Types of edges in a component of a context-free graph.

It is clear that for every internal point w of C_i that is adjacent to a frontier point in F_i, $C_i^{\downarrow w}$ is the (unique) connected component of the subgraph of C_i induced by $\mathcal{D}om(C_i) \setminus F_i$ that contains w. In [87], a component of the form $C_i^{\downarrow w}$, with w being an internal point adjacent to some frontier point in F_i, is called *second level subgraph* of C_i. Clearly, there can be only finitely many distinct second level subgraphs of C_i and each of them is end-isomorphic to some representative $C_{i'}$. Now, for every internal point w that is adjacent to a frontier point in the same representative C_i, we fix (i) an index $sub(i, w)$ such that $C_{sub(i,w)}$ is a representative of $C_i^{\downarrow w}$ (namely, $C_{sub(i,w)}$ and $C_i^{\downarrow w}$ are end-isomorphic) and (ii) an end-isomorphism $\Psi_{i,w}$ between $C_i^{\downarrow w}$ and

$C_{sub(i,w)}$. Without loss of generality, we can assume that for every $i \in [m]$, distinct second level subgraphs of C_i have distinct representatives; precisely, for every pair of internal points w, w' which are adjacent to the same frontier point of C_i, we assume that either $w = w'$ or $sub(i,w) \neq sub(i,w')$ holds. Note that if such an assumption were not satisfied, one can take distinct copies of each representative in such a way that it would be possible to assign a *unique* index to each second level subgraph of C_i (since there are only finitely many distinct second level subgraphs of C_i, we need only finitely many copies of the representatives). Therefore, building up over Muller and Schupp's ideas, for every label $a \in \Lambda$ and every tuple of indices $i, i' \in [m]$, $j \in [l_i]$, $j' \in [l_{i'}]$, we can define the partial functions $\widetilde{\delta}_a$, $\overrightarrow{\delta}_a$, $\overleftarrow{\delta}_a$ as follows[3]:

- $\widetilde{\delta}_a(i,j) = j'$ holds iff $(f_{i,j}, f_{i,j'})$ is an a-labeled edge of type 1 in C_i;
- $\overrightarrow{\delta}_a(i,j) = (i',j')$ holds iff C_i contains an internal point w and an a-labeled edge $(f_{i,j}, w)$ of type 2, with $i' = sub(i,w)$ (this means that $C_i^{\downarrow w}$ is end-isomorphic to $C_{i'}$) and $f_{i',j'}$ being the frontier point in $C_{i'}$ that corresponds to w (under the fixed end-isomorphism $\Psi_{i,w}$ between $C_i^{\downarrow w}$ and $C_{i'}$);
- $\overleftarrow{\delta}_a(i',i,j) = j'$ holds iff $C_{i'}$ contains an internal point w and an a-labeled edge $(w, f_{i',j'})$ of type 2, with $i = sub(i',w)$ (this means that $C_{i'}^{\downarrow w}$ is end-isomorphic to C_i) and $f_{i,j}$ being the frontier point in C_i that corresponds to w (under the fixed end-isomorphism $\Psi_{i,w}$ between $C_{i'}^{\downarrow w}$ and C_i).

We assume that the values of $\widetilde{\delta}_a$, $\overrightarrow{\delta}_a$, and $\overleftarrow{\delta}_a$ are undefined whenever not specified. It is worth noticing that the edges of G (hence G itself) are uniquely determined once we know (i) the number m of the representatives of the components of G, (ii) for each representative C_i, the set of frontier points $F = \{f_{i,1}, ..., f_{i,l_i}\}$, (iii) for each $a \in \Lambda$, the partial functions $\widetilde{\delta}_a$, $\overrightarrow{\delta}_a$, and $\overleftarrow{\delta}_a$, (iv) an index $i_* \in [m]$ such that C_{i_*} is end-isomorphic to G, and (v) the index $j_* \in [l_{i_*}]$ such that f_{i_*,j_*} is the frontier point of C_{i_*} corresponding to the source vertex v_0 of G under the fixed end-isomorphism between C_{i_*} and G. To see why this holds, recall that

i) (v,v') is an a-labeled edge of type 1 in G iff (v,v') is an a-labeled edge of type 1 in $G^{\downarrow v}$, iff $\widetilde{\delta}_a(i,j) = j'$, where i is the index of the representative C_i of $G^{\downarrow v}$ and $f_{i,j}$ (resp., $f_{i,j'}$) is the frontier point of C_i corresponding to v (resp., to v') under the end-isomorphism between C_i and $G^{\downarrow v}$;

ii) (v,v') is an a-labeled edge of type 2 in G iff (v,v') is an a-labeled edge of type 2 in $G^{\downarrow v}$, iff $\overrightarrow{\delta}_a(i,j) = (i',j')$, where i is the index of the representative C_i of $G^{\downarrow v}$, $f_{i,j}$ is the frontier point of C_i corresponding to v under the end-isomorphism between C_i and $G^{\downarrow v}$, w is the internal point of C_i corresponding to v' under the end-isomorphism between C_i and $G^{\downarrow v}$, and $f_{i',j'}$ is the frontier point of $C_{sub(i,w)}$ corresponding to w under the end-isomorphism between C_i and $C_{i'} = C_{sub(i,w)}$;

iii) (v,v') is an a-labeled edge of type 3 in G iff (v,v') is an a-labeled edge of type 3 in $G^{\downarrow v'}$, iff $\overleftarrow{\delta}_a(i',i,j) = j'$, where i' is the index of the representative $C_{i'}$ of $G^{\downarrow v'}$, $f_{i',j'}$ is the frontier point of $C_{i'}$ corresponding to v' under the end-isomorphism between $C_{i'}$ and $G^{\downarrow v'}$, w is the internal point of $C_{i'}$ corresponding to v under the

[3]Note that from the previous assumptions, these are well-defined partial functions.

end-isomorphism between $C_{i'}$ and $G^{\downarrow v'}$, and $f_{i,j}$ is the frontier point of $C_{sub(i',w)}$ corresponding to w under the end-isomorphism between $C_{i'}$ and $C_i = C_{sub(i',w)}$.

This means that the tuple $\big(m, (l_i)_{i\in[m]}, (\widetilde{\delta}_a)_{a\in\Lambda}, (\overrightarrow{\delta}_a)_{a\in\Lambda}, (\overleftarrow{\delta}_a)_{a\in\Lambda}, i_*, j_*\big)$ can be given the status of representation of the context-free graph G. Moreover, we can exploit the results given in [21, 22, 26] (which provide an effective characterization of the context-free graphs in terms of graphs generated by graph grammars) in order to compute the above defined representation on the grounds of a given pushdown system generating G.

We now give some technical definitions, which are needed to prove the main result of this section, namely, that T (i.e., the unfolding of the context-free graph G) is a rank 1 tree. We encode each path π in G with a sequence of the form $(h_1, i_1, j_1), ..., (h_n, i_n, j_n)$, where, for every $0 \leq k \leq n$, if v is the $k+1$-th vertex along π, then h_k gives the level of v, i_k gives the index of the representative C_{i_k} of the component $G^{\downarrow v}$, and j_k gives the index of the frontier point in C_{i_k} corresponding to the vertex v in $G^{\downarrow v}$. Formally, we say that a sequence $(h_1, i_1, j_1), ..., (h_n, i_n, j_n)$ is a *δ-path with labels* $a_1, ..., a_{n-1}$ iff f_{i_1,j_1} is a frontier point in C_{i_1} and for every $1 \leq k < n$, one of the following conditions holds:

1. $h_{k+1} = h_k$, $i_{k+1} = i_k$, and $\widetilde{\delta}_{a_k}(i_k, j_k) = j_{k+1}$,
2. $h_{k+1} = h_k + 1$ and $\overrightarrow{\delta}_{a_k}(i_k, j_k) = (i_{k+1}, j_{k+1})$,
3. $h_k > 0$, $l_{k+1} = h_k - 1$, and $\overleftarrow{\delta}_{a_k}(i_{k+1}, i_k, j_k) = j_{k+1}$.

We can further define the *concatenation* of two given δ-paths α and α', under the proviso that one of the above conditions holds by instanciating (h_k, i_k, j_k) (resp., $(h_{k+1}, i_{k+1}, j_{k+1})$) with the last (resp., the first) triple in α (resp., in α'), and a_k with a suitable label in Λ. The sequence resulting from such a concatenation is a δ-path with labels $a_1, ..., a_n, a_k, a'_1, ..., a'_{n'}$, provided that α has labels $a_1, ..., a_n$ and α' has labels $a'_1, ..., a'_{n'}$. Moreover, it turns out that, given a vertex $v \in \mathcal{D}om(G)$, the component $G^{\downarrow v}$ of G contains a path π from v with labels $a_1, ..., a_n$ iff there is a δ-path $\alpha_\pi = (h_0, i_0, j_0), (h_1, i_1, j_1), ..., (h_n, i_n, j_n)$ with labels $a_1, ..., a_n$ such that h_0 is the level of the vertex v in G, C_{i_0} is the representative of $G^{\downarrow v}$, and f_{i_0,j_0} is the frontier point of C_{i_0} that corresponds to the vertex v in $G^{\downarrow v}$. In such a case, we say that $\alpha_p i$ is a δ-path corresponding to π. Clearly, since G is deterministic, for every path π from v in $G^{\downarrow v}$, there exists a *unique* corresponding δ-path α_π.

Now, we can provide a suitable factorization Π of T. On the grounds of such a factorization, we will be able to effectively build, for any given tree automaton M, a regular retraction of T with respect to M and Π. We let $\Delta = \bigcup_{i\in[m]} \{b_{i,j} : j \in [l_i]\}$ be a finite set in bijection with, but disjoint from, $\bigcup_{i\in[m]} F_i$. We then define the factorization Π of T with respect to Δ in such a way that

- $\mathcal{D}om(\Pi)$ consists of all and only the paths from v_0 in G that encompass only edges of type 2 (recall that T is the unfolding of G, hence each vertex of T identifies a path from v_0 in G),
- (π, π') is a $b_{i,j}$-labeled edge in Π iff $\pi \in \mathcal{D}om(\Pi)$, π' is the extension of π with an edge of type 2, and the last triple of the δ-path $\alpha_{\pi'}$ corresponding to π' is of the form $(|\pi'|, i, j)$.

Note that, by construction, Π is a deterministic Δ-labeled tree. In order to show that the corresponding retraction is a regular tree, we need to consider the form of the marked factors of T. The basic idea is to define, for each vertex u of Π, a suitable tuple of trees, which can be obtained from the tuple at the predecessor vertex via a regular tree insertion. Then, the marked factor rooted at a vertex u of Π turns out to be a projection of the corresponding tuple into a suitable component. We shall use the elements from a set $X = \{x_{1,1}, ..., x_{1,l_1}, ..., x_{m,1}, ..., x_{m,l_m}\}$ in bijection with Δ to denote the substitution variables for the regular tree insertions (note that the ordering in which we list the elements of X is relevant in order to correctly perform substituttions). We then specify the regular tree insertions by suitable (tuples of) $\Lambda \cup \Delta$-labeled $\{1\} \cup X$-colored Δ-augmented trees (here we assume that Λ, Δ, $\{1\}$, and X are all disjoint sets). For every $i \in [m]$, we define the colored graph B_i (see Figure 3.11) such that

- $\mathcal{D}om(B_i) = \big(\bigcup_{i' \in [m]} F_{i'}\big) \cup \Delta \cup X$ (here we assume that the sets $F_{i'}$ are disjoint),
- B_i contains an a-labeled edge from $f_{i',j'} \in F_{i'}$ to $f_{i'',j''} \in F_{i''}$ iff $i'' = i'$ and $\widetilde{\delta}_a(i', j') = j''$,
- B_i contains an a-labeled edge from $f_{i',j'} \in F_{i'}$ to $b_{i'',j''} \in \Delta$ iff $\overrightarrow{\delta}_a(i', j') = (i'', j'')$,
- B_i contains an a-labeled edge from $f_{i',j'} \in F_{i'}$ to $x_{i'',j''} \in X$ iff $i'' = i$ and $\overleftarrow{\delta}_a(i'', i', j') = j''$,
- B_i contains a $b_{i',j'}$-labeled edge from $x_{i'',j''} \in X$ to $f_{i',j'} \in F_{i'}$ iff $i'' = i$,
- no other edges exist,
- the vertices from $\bigcup_{i' \in [m]} F_{i'}$ are colored by 1,
- each vertex $x_{i',j'} \in X$ is marked with the substitution variable $x_{i',j'}$,
- each vertex $b_{i',j'} \in \Delta$ is marked with the label $b_{i',j'}$.

The graph B_i is called a *slice* of the context-free graph G. Note that the graphs $B_1, ..., B_m$ differ only in the incoming and in the outcoming edges of the X-colored vertices. We then define the tree $S_{i,i',j'}$ as the unfolding of the slice B_i from the vertex $f_{i',j'}$. Note that, since B_i is a *deterministic finite* graph (this follows by construction from having assumed G to be a deterministic context-free graph), $S_{i,i',j'}$ is a *deterministic regular* tree. For every $i, i' \in [m]$ and $j' \in [l_{i'}]$, we let $\gamma_{i,i',j'}$ be the regular tree insertion specified by $S_{i,i',j'}$ and we define $\bar{\gamma}_i$ as the regular tree insertion $(\gamma_{i,1,1}, ..., \gamma_{i,1,l_1}, ..., \gamma_{i,m,1}, ..., \gamma_{i,m,l_m})$. Note that, being $l = \sum_{i \in [m]} l_i$, $\bar{\gamma}_i$ can be given the status of function transforming l-tuples of (deterministic regular) trees to l-tuples of (deterministic regular) trees.

The following lemma is the key ingredient to prove that the unfolding of a deterministic context-free graphs is a reducible tree.

Lemma 3.5.3 *For any path π in Π of length n, if the δ-path α_π corresponding to π is of the form $(0, i_0, j_0)...(n, i_n, j_n)$, with $i_0 = i_*$ and $j_0 = j_*$, then the marked factor $T^+_\Pi[u]$, where u is the last vertex along π, is isomorphic to the tree*[4]

$$\mathcal{P}rj_{i_n,j_n}\big((\bar{\gamma}_{i_{n-1}} \circ ... \circ \bar{\gamma}_{i_0} \circ \bar{\gamma}_i)(\bar{\emptyset})\big)$$

[4]By a slight abuse of notation, we write $\mathcal{P}rj_{i,j}(\bar{T})$ to denote the projection of an l-tuple of the form $\bar{T} = (T_{1,1}, ..., T_{1,l_1}, ..., T_{m,1}, ..., T_{m,l_m})$ to the component $T_{i,j}$ of $\bar{T}$.

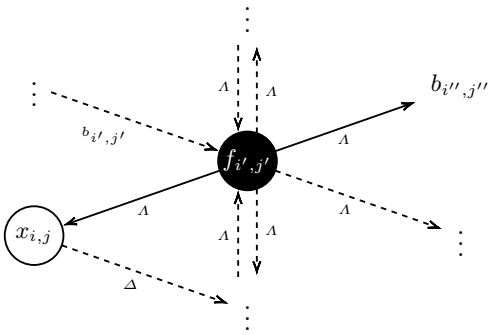

Figure 3.11: A slice B_i of a context-free graph.

where $\bar{\emptyset}$ denotes the l-tuple of empty trees and i is an arbitrary index from $[m]$.

Proof. We shall prove a result stronger than the one claimed by the lemma. Let $n \geq 0$ and $i_0, ..., i_n$ be a sequence of indices from $[m]$ such that, for all $0 \leq h < n$, $C_{i_{h+1}}$ is the representative of a second level subgraph of C_{i_h}. Further let $j_n \in [l_{i_n}]$, and, for all $0 \leq h \leq n$, inductively define $v^{(h)}$ as follows: $v^{(1)} = f_{i_n,j_n}$ and $v^{(h+1)}$ is the vertex of $C_{i_{n-h-1}}$ corresponding to $v^{(h)}$ under the end-isomorphism between the second level subgraph $C_{i_{n-h-1}}[v^{(h)}]$ and its representative $C_{i_{n-h}}$. Finally, let $v = v^{(n)}$. Intuitively, v is the vertex of C_{i_0} corresponding to the frontier point f_{i_n,j_n} of C_i under the chain of end-isomorphisms between the second level subgraphs and their representatives. Let G_n the colored graph obtained by (i) restricting C_{i_0} to those vertices whose level does not exceed n, together with those vertices of level $n+1$ which are targets of edges of type 2, and (ii) assigning color 1 to any vertex of level n and assigning color $b_{i',j'}$ to any vertex w of level $n+1$, where $f_{i',j'}$ is the frontier point of the representative of the component $G[w]$. Then, we denote by $T_{i_0 \ldots i_n,j}$ the unfolding of G_n from the vertex v. We prove, by induction on n, that the trees $T_{i_0 \ldots i_n,j}$ and $\mathcal{P}rj_{i_n,j}\big((\bar{\gamma}_{i_{n-1}} \circ \ldots \circ \bar{\gamma}_{i_0} \circ \bar{\gamma}_i)(\bar{\emptyset})\big)$, where i is an arbitrary element from $[m]$, are isomorphic. Note that for every path π in Π, if $\alpha_\pi = (0, i_0, j_0)...(n, i_n, j_n)$ is the δ-path corresponding to π and u is the last vertex along π, then the marked factor $T^+_\Pi[u]$ is isomorphic to the tree $T_{i_0 \ldots i_n,j_n}$ and hence the claim of the lemma would immediately follow. Hereafter, we say that a path π in a deterministic graph corresponds to a path π' in another deterministic graph iff the sequences of labels associated with π and π' coincide. We can use a similar terminology for the vertices of some deterministic trees, since each vertex is uniquely identified by its access path. For the base case $n = 0$, suppose that v is a 1-colored vertex of $T_{i_0,j}$. By definition of unfolding, v corresponds to a unique path π in G_0, which encompasses edges of type 1 only. Let i be an arbitrary element from $[m]$. By construction of B_i, there is a unique corresponding path π' in B_i, starting from $f_{i_0,j}$ and touching only vertices from F_{i_0}. Again, by definition of unfolding, such a path π' corresponds to a unique vertex w in $\gamma_{i,i_0,j}(\bar{\emptyset})$, which is colored with 1. The converse implication is symmetrical. Consider

now the case of v being a $b_{i',j'}$-colored vertex in $T_{i_0,j}$. By definition of unfolding, v corresponds to a unique path π in G_0, which encompasses only edges of type 1 with the exception of the last edge, which is of type 2. By construction, there is a unique corresponding path π' in B_i, for any $i \in [m]$, starting from $f_{i_0,j}$, ending in $b_{i',j'}$, and such that all the intermediate vertices belong to the set F_{i_0}. Again, by definition of unfolding, such a path π' corresponds to a unique $b_{i',j'}$-colored vertex w in $\gamma_{i,i_0,j}(\bar{\emptyset})$. The converse implication follows by symmetry. This proves that $T_{i_0,j}$ and $\gamma_{i,i_0,j}(\bar{\emptyset})$ are isomorphic trees.

For the induction step, let $n > 0$ and $i_0, ..., i_n$ be a sequence of indices such that, for all $0 \leq h < n$, $C_{i_{h+1}}$ is the representative of a second level subgraph of C_{i_h}. Further let $j \in [l_{i_n}]$. We consider the case of v being a $b_{i',j'}$-colored vertex in $T_{i_0 \ldots i_n,j}$ (the case of v being a 1-colored vertex can be handled in a similar way). By definition of unfolding, v corresponds to a unique path π in G_n, which touches only vertices whose level does not exceed n, with the exception of the last $b_{i',j'}$-colored vertex, which is of level $n+1$. Now, in the case where every vertex along π has level exactly n, the proof proceeds like for the case $n = 0$. Otherwise, if π contains some edges of type 2 or 3, then π can be broken up into a sequence of paths $\iota_1, \tau_1, \iota_2, \tau_2, ...$ such that for every $t \geq 1$,

i) ι_t starts from a vertex v_t of level n and encompasses only edges of type 1 with the exception of the last edge, which is of type 3 and reaches a vertex w_t of level $n-1$,
ii) τ_t starts from the vertex w_t, lies entirely inside the graph G_{n-1}, and ends with a vertex v_{t+1} having level n.

Now, let i be an arbitrary element from $[m]$. As for the path ι_t, we know that there is a unique corresponding path ι'_t in B_i, starting from the frontier point f_{i_t,j_t} of C_{i_t} (which corresponds to the vertex v_t in G_n) and such that every vertex along ι'_t belongs to F_{i_t} with the exception of the last one, which is of the form $x_{i_{n-1},j'_t} \in X$, where f_{i_{n-1},j'_t} is the frontier point of $C_{i_{n-1}}$ corresponding to the vertex w_t in G_{n-1}. Again, by definition of unfolding, the path ι'_t corresponds to a unique path ι''_t in S_{i_{n-1},i_t,j_t}, which starts from the root f_{i_t,j_t} and ends in the vertex x_{i_{n-1},j'_t}. Further note that the latter vertex x_{i_{n-1},j'_t} of S_{i_{n-1},i_t,j_t} is substituted by the tree $\mathcal{P}rj_{i_{n-1},j'_t}\big((\bar{\gamma}_{i_{n-2}} \circ ... \circ \bar{\gamma}_{i_0} \circ \bar{\gamma}_i)(\bar{\emptyset})\big)$ when we apply the tree insertion $\bar{\gamma}_{i_{n-1}}$ to the tree $(\bar{\gamma}_{i_{n-2}} \circ ... \circ \bar{\gamma}_{i_0} \circ \bar{\gamma}_i)(\bar{\emptyset})$. As for the path τ_t, we know that there is a unique path τ'_t in $T_{i_0 \ldots i_{n-1},j'_t}$ corresponding to τ_t. From the inductive hypothesis, it follows that there is a unique corresponding path τ''_t in the tree $\mathcal{P}rj_{i_{n-1},j'_t}\big((\bar{\gamma}_{i_{n-2}} \circ ... \circ \bar{\gamma}_{i_0} \circ \bar{\gamma}_i)(\bar{\emptyset})\big)$. Hence, by concatenating the paths $\iota''_1, \tau''_1, \iota''_2, \tau''_2, ...$, we obtain a path in the tree $\mathcal{P}rj_{i_n,j}\big((\bar{\gamma}_{i_{n-1}} \circ ... \circ \bar{\gamma}_{i_0} \circ \bar{\gamma}_i)(\bar{\emptyset})\big)$, which corresponds exactly to the path π. The converse implications are proved symmetrically. Therefore, we can state that $T_{i_0 \ldots i_n,j}$ and $\mathcal{P}rj_{i_n,j}\big((\bar{\gamma}_{i_{n-1}} \circ ... \circ \bar{\gamma}_{i_0} \circ \bar{\gamma}_i)(\bar{\emptyset})\big)$ are isomorphic trees, for any given $i \in [m]$. ■

Theorem 3.5.4 *The unfolding of a deterministic context-free graph is a rank* 1 *(hence reducible) tree.*

Proof. Given the representation $\big(m, (l_i)_{i\in[m]}, (\widetilde{\delta}_a)_{a\in\Lambda}, (\overrightarrow{\delta}_a)_{a\in\Lambda}, (\overleftarrow{\delta}_a)_{a\in\Lambda}, i_*, j_*\big)$ of a context-free graph G, let Π be the factorization of T with respect to Δ and $\bar{\gamma}_1, ..., \bar{\gamma}_m$

the regular tree insertions as defined before. By Corollary 3.4.10, we can compute, for any given tree automaton M, the abstractions $\bar{\gamma}_1^M, ..., \bar{\gamma}_m^M$, corresponding to the regular tree insertions $\bar{\gamma}_1, ..., \bar{\gamma}_m$ and mapping l-tuples of M, Δ-types to l-tuples of M, Δ-types. By Lemma 3.5.3, we know that for every path π in Π of length n, if the δ-path α_π corresponding to π is of the form $(0, i_0, j_0)...(n, i_n, j_n)$, with $i_0 = i_*$ and $j_0 = j_*$, then $\mathcal{P}rj_{i_n,j_n}\big((\bar{\gamma}_{i_{n-1}}^M \circ ... \circ \bar{\gamma}_{i_0}^M \circ \bar{\gamma}_i^M)(\bar{t}_\emptyset)\big)$ is an M, Δ-type of the marked factor $T_\Pi^+[u]$, where u is the last vertex along π and i is an arbitrary element from $[m]$. We denote such an M, Δ-type by $C(\vec{u})$, where $\vec{u}$ denotes the sequence of labels along the access path of u in Π. Clearly, C is a retraction of T with respect to M and Π. Moreover, if we let $I = \bigcup_{i\in[m]} \{(i, j) : j \in [l_i]\}$ and we denote by F the Δ-labeled I-colored tree F such that

- $\varepsilon \in \mathcal{D}om(F)$ and $F(\varepsilon) = (i_*, j_*)$,
- $\vec{u} \cdot b_{i,j} \in \mathcal{D}om(F)$ iff $\vec{u} \in \mathcal{D}om(F)$ and there is $a \in \Lambda$ satisfying $\vec{\delta}_a\big(F(\vec{u})\big) = (i, j)$ (in such a case $F(\vec{u} \cdot b_{i,j}) = (i, j)$),

then C can be viewed as the output of the Mealy tree automaton $N' = (S, I \cup \mathscr{T}_{M,\Lambda}, \Delta, \delta, s_0, \Omega)$ on input F, where

- $S = \mathscr{T}_{M,\Lambda}^I$,
- $\delta\big(\bar{t}, (i, j), b_{i',j'}\big) = \bar{\gamma}_i^M(\bar{t})$,
- $s_0 = \gamma_i^M(\bar{t}_\emptyset)$, where i is an arbitrary element from $[m]$ and $\bar{t}_\emptyset$ is the tuple consisting of l copies of the basic M, Δ-type of the empty tree,
- $\Omega\big(\bar{t}, (i, j)\big) = \mathcal{P}rj_{i,j}(\bar{t})$.

Finally, notice that F is a regular tree and, since the class of regular trees is closed under finite-state recolorings, C is a regular tree as well. This shows that T is a rank 1 tree. ■

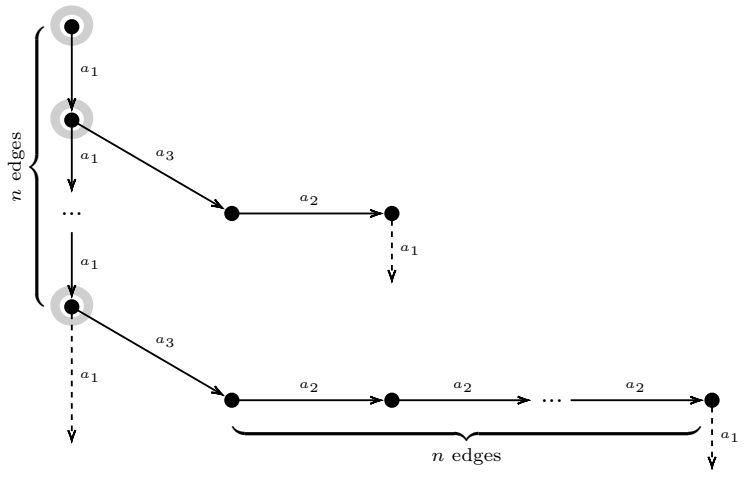

Figure 3.12: The unfolding of a contex-free graph.

In the following example, we run through the crucial steps of the proof of Theorem

3.5.4 by considering a particular context-free graph.

Example 3.5.5 Let consider the graph G of Figure 3.9 and its unfolding T from $v_0 = (s_0, X)$ (see Figure 3.12). Note that (i) (s_0, X) is the only vertex of level 0 in G, (ii) the vertices of level 1 are (s_0, XY) and (s_1, X), (iii) the vertices of level 2 are (s_0, XYY) and (s_1, XY), and so on. Hence, we have only two representatives, say C_1 and C_2, for the components $G[v]$ of the graph G. Let $C_1 = G$ be the representative of the component $G[(s_0, X)]$ and $C_1 = G[(s_0, XY)]$ be the representative of the other components $G[(s_1, X)]$, $G[(s_0, XYY)]$, etc. The representative C_1 has a single frontier point $f_{1,1} = (s_0, XY)$, while the representative C_2 has two frontier points $f_{2,1} = (s_0, XY)$ and $f_{2,2} = (s_1, X)$. Moreover, each representative has a single second level subgraph end-isomorphic to C_2. Hence, we can set $\widetilde{\delta}_{a_3}(2,1) = 2$, $\overrightarrow{\delta}_{a_1}(1,1) = \overrightarrow{\delta}_{a_1}(2,1) = (2,1)$, $\overleftarrow{\delta}_{a_2}(1,2,2) = 1$, $\overleftarrow{\delta}_{a_2}(2,2,2) = 2$, and let $\widetilde{\delta}_a$, $\overrightarrow{\delta}_a$, $\overleftarrow{\delta}_a$ be undefined elsewhere. All paths from v_0 in G consisting only of edges of type 2 are of the form $\big((s_0, X), (s_0, XY)\big), ..., \big((s_0, XY^{n-1}), (s_0, XY^n)\big)$. Thus, the factorization Π of the unfolding T of G is obtained by simply selecting the vertices that lie in the leftmost branch of T. Then, by letting $\Delta = \{b_{1,1}, b_{2,1}, b_{2,2}\}$, $X = \{x_{1,1}, x_{2,1}, x_{2,2}\}$, and $\Sigma = \{1\} \cup \Delta \cup X$, we can define the slices B_1 and B_2 of G (see Figure 3.13) as follows:

- $\mathcal{D}om(B_1) = \mathcal{D}om(B_2) = \{f_{1,1}, f_{2,1}, f_{2,2}, b_{1,1}, b_{2,1}, b_{2,2}, x_{1,1}, x_{2,1}, x_{2,2}\}$,
- both B_1 and B_2 have two a_1-labeled edges $(f_{1,1}, b_{2,1})$ and $(f_{2,1}, b_{2,1})$),
- B_1 (resp., B_2) has a single a_2-labeled edge $(f_{2,2}, x_{1,1})$ (resp., $(f_{2,2}, x_{2,2})$),
- both B_1 and B_2 have a single a_3-labeled edge $(f_{2,1}, f_{2,2})$,
- both B_1 and B_2 have two $b_{2,1}$-labeled edges $(x_{1,1}, f_{2,1})$ and $(x_{2,1}, f_{2,1})$,
- no other edges exist,
- the vertices $f_{1,1}, f_{2,1}, f_{2,2}$ are colored by 1 (these are represented by black-colored nodes in Figure 3.13),
- the vertices $b_{1,1}, b_{2,1}, b_{2,2}$ are colored by $b_{1,1}, b_{2,1}, b_{2,2}$, respectively,
- the vertices $x_{1,1}, x_{2,1}, x_{2,2}$ are colored by $x_{1,1}, x_{2,1}, x_{2,2}$.

Now, let $\gamma_{1,1,1}$, $\gamma_{1,2,1}$, and $\gamma_{1,2,2}$ (resp., $\gamma_{2,1,1}$, $\gamma_{2,2,1}$, and $\gamma_{2,2,2}$) be the regular tree insertions specified by the unfoldings of B_1 (resp., B_2) from the vertices $f_{1,1}$, $f_{2,1}$, and $f_{2,2}$ and let $\bar{\gamma}_1 = (\gamma_{1,1,1}, \gamma_{1,2,1}, \gamma_{1,2,2})$ and $\bar{\gamma}_2 = (\gamma_{2,1,1}, \gamma_{2,2,1}, \gamma_{2,2,2})$. By Lemma 3.5.3, the marked factor $T^+_\Pi[\varepsilon]$ is isomorphic to $\mathcal{P}rj_{1,1}\big(\bar{\gamma}_i(\bar{\emptyset})\big)$, for any choice of $i \in [m]$, for instance $i = 1$. Similarly, for every vertex u of Π at distance $n > 0$ from the root, the marked factor $T^+_\Pi[u]$ is isomorphic to $\mathcal{P}rj_{2,1}\big(\bar{\gamma}_2^{n-1} \circ \bar{\gamma}_1^2(\bar{\emptyset})\big)$. In Figure 3.14 we depicted the triples of trees $\bar{\gamma}_1(\bar{\emptyset})$, $\bar{\gamma}_1 \circ \bar{\gamma}_1(\bar{\emptyset})$, $\bar{\gamma}_2 \circ \bar{\gamma}_1 \circ \bar{\gamma}_1(\bar{\emptyset})$ and we highlighted the corresponding marked factor inside each triple. ■

We now make a few remarks on the noticeable class of (deterministic) context-free graphs. In [108], Stirling showed that every *prefix-recognizable graph* (i.e., a graph belonging to the second level of Caucal hierarchy, generated by applying a MSO-definable interpretation to the infinite complete binary tree, see [23, 25]) can be obtained from the configuration graph of a pushdown system by 'collapsing' the ends (i.e., source and target vertices) of each ε-labeled edge. More precisely, given

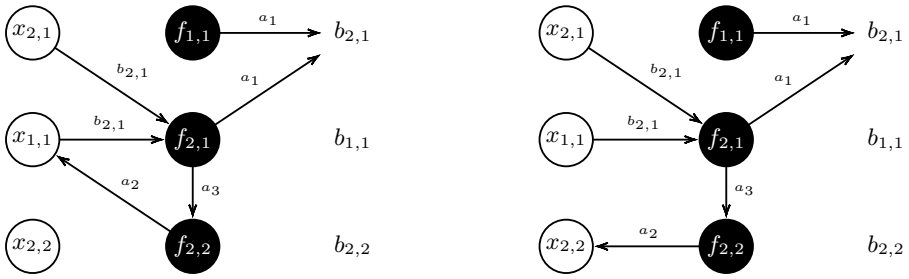

Figure 3.13: Two slices B_1 and B_2 for a context-free graph.

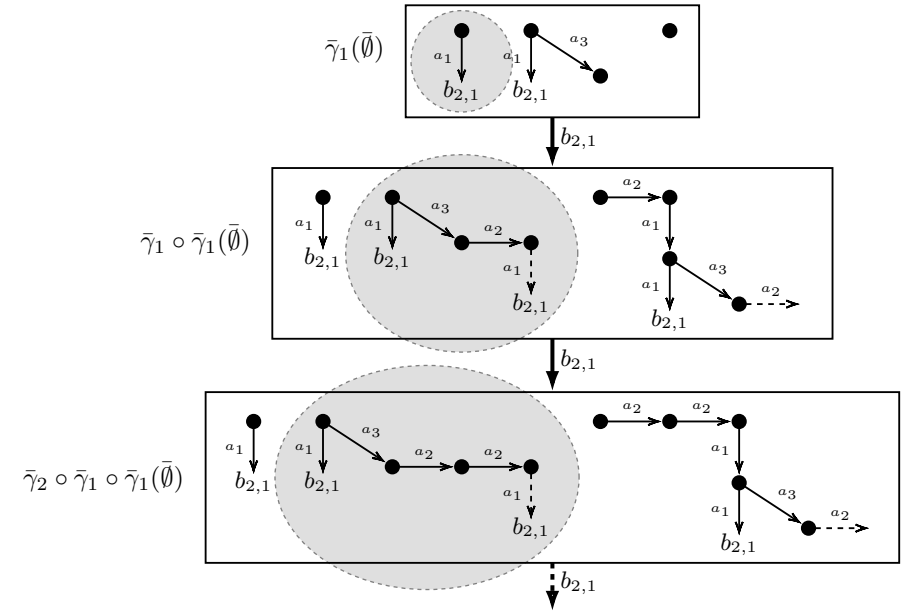

Figure 3.14: The trees generating the unfolding of a contex-free graph.

a pushdown system M, we can assume, without loss of generality, that there is no configuration from which M is able to perform both ε-moves and non-ε-moves. Under such an assumption, one can define the ε-closure of the configuration graph G of M as the graph obtained from G by (i) adding an a-labeled edge (v, v') whenever G contains a path from v to v' labeled by $a, \varepsilon, ...\varepsilon$ and (ii) removing all vertices with outgoing ε-labeled edges. Stirling showed that every prefix-recognizable graph is the ε-closure of a context-free graph. Moreover, in [23, 25] Caucal proved that the context-free graphs are exactly the prefix recognizable graphs of *bounded degree*, which can be equivalently characterized as the *HR-equational graphs* of bounded-degree (see [35] for the definition and [4, 54, 55] for the characterization). Finally, the unfoldings of

the HR-equational graphs are usually called *algebraic trees*. As a consequence of these results, it turns out that the *deterministic* prefix-recognizable graphs are exactly the deterministic algebraic trees.

3.5.2 Generators for the levels of the Caucal hierarchy

In [18], Carayol and Wöhrle show that, for each level of the Caucal hierarchy, there exists a representative graph, called *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 $n+1$-th level of the Caucal hierarchy is defined as the n-fold application of the treegraph operation (see [116, 115] for a formal definition) to the infinite binary complete tree. These generators are closely related to another family of trees, which was introduced by Cachat in order to simulate games on higher order pushdown systems [15]. These trees are obtained from the infinite complete binary tree via n-fold iterations of the unfolding with backward edges and loops. In particular, it can be proved that each level n generator G_n can be defined inside the level n tree T_n of Cachat via an MSO-definable interpretation that first restricts the domain to those vertices that correspond to words not containing occurrences of $a \cdot \bar{a}$ or $\bar{a} \cdot a$ (where a denotes a forward edge and $\bar{a}$ the corresponding backward edge introduced by the treegraph operation) and then reverses the $\bar{a}$-labeled edges. Therefore the trees introduced by Cachat are also generators of the graphs of the Caucal hierarchy via MSO-definable interpretations. Here, we define a more general notion of tree generator, which turns out to be powerful enough to capture all *deterministic* trees in the Caucal hierarchy via inverse rational mappings (i.e., special cases of MSO-definable interpretations). We then show that these tree generators are reducible trees.

Before all, we introduce some basic terminology. For the sake of simplicity, we shall not distinguish between isomorphic graphs/trees, thus writing $G = G'$ whenever G is isomorphic to G'. Now, let Λ, Δ be two finite sets of edge labels and let $\bar{\Lambda} = \{\bar{a} : a \in \Lambda\}$ be a set disjoint from but in bijection with Λ. We use Λ_+ as a shorthand for $\Lambda \cup \bar{\Lambda}$. Given any Λ-labeled graph $G = \big(V, (E_a)_{a\in\Lambda}\big)$, we extend G with backward edges $E_{\bar{a}} = \{(v, v') : (v', v) \in E_a\}$ and we associate to each path in the extended graph G a sequence of labels from Λ_+. A *rational mapping* is a function $h : \Delta \to \mathscr{P}(\Lambda_+^*)$ that maps a symbol from Δ to a regular language of finite words over Λ_+. Such a mapping can be applied by the inverse to any Λ-labeled graph $G = \big(V, (E_a)_{a\in\Lambda}\big)$, thus obtaining the Δ-labeled graph $h^{-}1(G) = \big(V', (E'_b)_{b\in\Delta}\big)$ defined as follows:

- for all $b \in \Delta$, E'_b is the set of all pairs $(v, v') \in V \times V$ such that there is a path from v to v' in the extended graph G labeled with a word in $h(b)$,
- V' is the subset of V that is implicitly given by the edge relations $(E'_b)_{b\in\Delta}$.

The notion of inverse rational mapping can be extended to deal with colored graphs, by simply allowing $h(b)$, for any $b \in \Delta$, to be a language consisting of finite sequences of edge labels interleaved by vertex colors. Here, for simplicity, we restrict our attention to uncolored graphs and rational mappings as defined above. Since the edge relations of the output graph are MSO-definable in the input graph, the operation of

inverse rational mapping is a special case of MSO-definable interpretation. On the contrary, there clearly exist MSO-definable interpretations which are not expressible as inverse rational mappings. Even though inverse rational mappings are less expressive than MSO-definable interpretations, we can equivalently define the Caucal hierarchy in terms of unfoldings and inverse rational mappings, starting from finite graphs (cf. [24]). Formally, the level 0 graphs of the Caucal hierarchy are all and only the finite rooted graphs and for every $n \in \mathbb{N}$, the level $n+1$-th trees (resp., the level $n+1$ graphs) of the Caucal hierarchy are obtained by applying unfoldings to the level n graphs (resp., by applying inverse rational mapping to the level n-trees). This implies that the trees in the first level of the Caucal hierarchy are the regular trees and for every $n > 0$, the trees in the $n+1$-th level of the Caucal hierarchy are obtained from the trees in the n-th level by applying an inverse rational mapping followed by an unfolding.

We now investigate on particular forms of inverse rational mappings. Given a finite set Λ of edge labels, we denote by h_Λ the mapping from $\Delta = \Lambda \cup \bar{\Lambda} \cup \{\#\}$ to $\mathscr{P}(\Lambda_+^*)$ such that, for all $a \in \Lambda$, $h_\Lambda(a) = \{a\}$, $h_\Lambda(\bar{a}) = \{\bar{a}\}$, and $h_\# = \{\varepsilon\}$. Notice that the application of h_Λ by the inverse to a Λ-labeled graph $G = \big(V, (E_a)_{a\in\Lambda}\big)$ results in a Δ-labeled graph $h_\Lambda^{-1}(G)$ of the form $\big(V, (E'_b)_{b\in\Delta}\big)$, where, for all $a \in \Lambda$, $E'_a = E_a$, $E'_{\bar{a}} = \{(v,v') : (v',v) \in E_a\}$, and $E'_\# = \{(v,v) : v \in V\}$. Roughly speaking, the inverse rational mapping h_Λ^{-1} extends the input graph G with backward edges and loops. We shortly denote by $\mathcal{B}ack\mathcal{U}nf$ the *unfolding with backward edges and loops*, formally defined as the transformation that maps a Λ-labeled rooted graph G to the $\Lambda \cup \bar{\Lambda} \cup \{\#\}$-labeled tree $\mathcal{U}nf\big(h_\Lambda^{-1}(G)\big)$.

We then define another form of rational mapping. A *rational Λ-forward mapping* is a rational mapping of the form $h : \Delta \to \mathscr{P}(\Lambda^+)$ (here Λ^+ denotes the set $\bigcup_{n>0} \Lambda^n$, not the shorthand Λ_+ for $\Lambda \cup \bar{\Lambda}$), where Δ is any arbitrary finite set. Intuitively, a rational forward mapping h maps each symbol $b \in \Delta$ to a regular language consisting only of non-empty words with no occurrences of symbols from $\bar{\Lambda}$. Notice that, if h is a rational Λ-forward mapping and T is a Λ-labeled tree, then $h^{-1}(T)$ is a (possibly non-deterministic) Δ-labeled tree. Below, we provide other properties of inversee rational mappings.

Lemma 3.5.6 *For every rational mapping $h : \Delta \to \mathscr{P}(\Lambda_+^*)$, there is a rational Δ'-forward mapping $\overrightarrow{h} : \Delta \to \mathscr{P}\big((\Delta')^+\big)$, with $\Delta' = \Lambda \cup \bar{\Lambda} \cup \{\#\}$, such that $h^{-1} = \overrightarrow{h}^{-1} \circ h_\Lambda$.*

Proof. The proof is almost trivial. For any given word $w \in \Lambda_+^*$, we denote by $\vec{w}$ the non-empty word over Δ' defined by:

- if $w = \varepsilon$, then $\vec{w} = \#$,
- if $w = w' \cdot a$, with $a \in \Lambda_+$, then $\vec{w} = \vec{w}' \cdot a \cdot \#$

(intuitively, $\vec{w}$ is obtained by interleaving each symbol of w with $\#$). Then, we define $\overrightarrow{h}$ by setting $\overrightarrow{h}(b) = \{\vec{w} : w \in h(b)\}$, for every $b \in \Delta$. Clearly, $\overrightarrow{h}$ is a rational Δ'-

forward mapping. Moreover, we have

$$\begin{aligned}\big(h_\Lambda \circ \overrightarrow{h}\big)(b) &= h_\Lambda\big(\overrightarrow{h}(b)\big)\\ &= h_\Lambda\big(\{\overrightarrow{w} \,:\, w \in h(b)\}\big)\\ &= \big\{h_\Lambda(\overrightarrow{w}) \,:\, w \in h(b)\big\} = h(b)\end{aligned}$$

which implies that $h^{-1} = \overrightarrow{h}^{-1} \circ h_\Lambda$. ■

Lemma 3.5.7 *For every rational Λ-forward mapping $h : \Delta \to \mathscr{P}(\Lambda^+)$ and for every Λ-labeled rooted graph G, $\mathcal{U}nf\big(h^{-1}(G)\big)$ and $h^{-1}\big(\mathcal{U}nf(G)\big)$ are bisimilar trees.*

Proof. We denote by T the tree $\mathcal{U}nf\big(h^{-1}(G)\big)$ and by T' the tree $h^{-1}\big(\mathcal{U}nf(G)\big)$. Every vertex v in T identifies a (unique) path from the source vertex to a vertex x in $h^{-1}(G)$; x is also a vertex of G and hence we can denote it with $\overrightarrow{v}$. Similarly, every vertex v' in T' is also a vertex of $\mathcal{U}nf(G)$ and such a vertex identifies a (unique) path from the source vertex to a vertex x' in G, which we denote by $\overrightarrow{v}'$. Now, we define the relation $\sim \subseteq \mathcal{D}om(T) \times \mathcal{D}om(T')$ as follows: for every vertex v of T and for every vertex v' of T', $v \sim v'$ iff $\overrightarrow{v} = \overrightarrow{v}'$ and the sequences of labels along the access path of v in T and along the access path of v' in T' are the same. Now, let $u \in \mathcal{D}om(T)$ and $u' \in \mathcal{D}om(T')$ such that $u \sim u'$ and let v be a b-successor of u in T, with $b \in \Delta$. Since T is the unfolding of the graph $h^{-1}(G)$, u denotes a path τ in $h^{-1}(G)$ starting from the source vertex and reaching a vertex x. Similarly, v denotes a path in $h^{-1}(G)$ which is the extension of τ by a b-labeled edge of the form (x, y). Since $h(b)$ does not contain the empty word or words with occurrences of symbols in $\bar{\Lambda}$, G contains a path π from $\overrightarrow{u}$ to $\overrightarrow{v}$ labeled with a word $w \in h(b)$. Now, consider the vertex u' of T'. Since $T' = h^{-1}\big(\mathcal{U}nf(G)\big)$, u' can be thought of as a vertex of $\mathcal{U}nf(G)$ that identifies a (unique) path τ' in G starting from the source vertex and reaching the vertex $\overrightarrow{u}' = \overrightarrow{u}$. This means that the path τ' can be prolonged with the sequence π of edges in such a way that the resulting path $\tau' \cdot \pi$ goes from the source vertex to the vertex $\overrightarrow{v}$ of G. By definition of unfolding, π (resp., $\tau' \cdot \pi$) identifies a path π' (resp., $\tau'' \cdot \pi'$) in $\mathcal{U}nf(G)$ starting from u' (resp., from the root) and reaching a vertex v' such that $\overrightarrow{v}' = \overrightarrow{v}$. Moreover, since $u \sim u'$ holds, the path τ' is labeled with the same sequence of symbols associated to τ. This shows that there is a b-successor v' of u' such that $v \sim v'$. The converse implication can be in a similar way. ■

Proposition 3.5.8 *For any given rational mapping $h : \Delta \to \mathscr{P}(\Lambda_+^*)$, there is a rational Δ'-forward mapping $\overrightarrow{h} : \Delta \to \mathscr{P}\big((\Delta')^+\big)$, with $\Delta' = \Lambda \cup \bar{\Lambda} \cup \{\#\}$, such that, for every Λ-labeled rooted graph G, $\mathcal{U}nf\big(h^{-1}(G)\big)$ and $\overrightarrow{h}^{-1}\big(\mathcal{B}ack\mathcal{U}nf(G)\big)$ are bisimilar trees.*

Proof. The claim follows trivially from the previous two lemmas. Given the rational mapping h, by Lemma 3.5.6, there is a forward rational mapping $\overrightarrow{h}$ such that $h^{-1} = \overrightarrow{h}^{-1} \circ h_\Lambda^{-1}$. By Lemma 3.5.7, the tree $\mathcal{U}nf\big(\overrightarrow{h}^{-1}(h_\Lambda^{-1}(G))\big)$ is bisimilar to the tree $\overrightarrow{h}^{-1}\big(\mathcal{U}nf(h_\Lambda^{-1}(G))\big) = \overrightarrow{h}^{-1}\big(\mathcal{B}ack\mathcal{U}nf(G)\big)$ and this proves the claim. ■

Now, we introduce tree generators and their properties. As a matter of fact, recall that the unfolding of the semi-infinite line with backward edges and loops, described in Example 3.4.11, is isomorphic to the tree $\mathcal{B}ack\mathcal{U}nf\big((\mathbb{N}, succ)\big)$, where $(\mathbb{N}, succ)$ is the structure of the natural numbers equipped with the successor function. Tree generators are nothing but a generalization based on that example. Intuitively, a level n tree generator is obtained from a regular tree via an n-fold application of the unfolding with backward edges and loops.

Definition 3.5.9 *A* level 0 tree generator *is a regular tree. For every $n \in \mathbb{N}$, a* level $n+1$ tree generator *is a tree of the form $T' = \mathcal{B}ack\mathcal{U}nf(T)$, where T is a level n tree generator.*

From the previous results, it follows that tree generators together with inverse rational forward mappings suffice to generate all deterministic trees in the Caucal hierarchy.

Theorem 3.5.10 *For every tree T in the $n+1$-th level of the Caucal hierarchy, there exist a level n-tree tree generator T' and a rational forward mapping $\overrightarrow{h}$ such that T is bisimilar to $\overrightarrow{h}^{-1}(T')$.*

Proof. We prove the theorem by induction on n. The case $n = 0$ is trivial, since the trees in the first level of the Caucal hierarchy are exactly the regular trees, namely, the level 0 tree generators. For the inductive step, we assume that the claim holds for $n \geq 0$ and we prove it for $n + 1$. Let T be a tree in the $n + 2$-th level of the Caucal hierarchy. By construction, there are a tree T'' in the $n + 1$-th level of the Caucal hierarchy and a rational mapping h_1 such that $T = \mathcal{U}nf\big(h_1^{-1}(T'')\big)$. From the inductive hypothesis, we know that there exist a level n tree generator T' and a rational forward mapping h_2 such that T'' is bisimilar to $h_2^{-1}(T')$. Thus, by applying Proposition 3.5.8 to the rational mapping $h_2 \circ h_1$, we know that there is a rational forward mapping $\overrightarrow{h}$ such that T is bisimilar to $\overrightarrow{h}^{-1}\big(\mathcal{B}ack\mathcal{U}nf(T')\big)$ (recall that the operation of unfolding preserves bisimilarity of graphs). ■

Corollary 3.5.11 *For every deterministic tree T in the $n + 1$-th level of the Caucal hierarchy, there exist a level n-tree tree generator T' and rational forward mapping h such that $T = h^{-1}(T')$.*

Proof. The claim follows trivially from Theorem 3.5.10, since bisimilar deterministic trees are isomorphic. ■

We now show that tree generators are reducible trees. This comes as a consequence of the fact that the whole class of reducible trees is closed under the operation $\mathcal{B}ack\mathcal{U}nf$. The idea underlying the proof, is based on a generalization of Example 3.4.11, which exploits the closure properties of rank n trees presented in Section 3.4.3.

Theorem 3.5.12 *The class of reducible trees is effectively closed under $\mathcal{B}ack\mathcal{U}nf$, precisely, given a rank n tree T, $\mathcal{B}ack\mathcal{U}nf(T)$ is a rank $n + 1$ tree.*

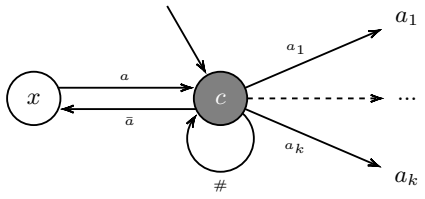

Figure 3.15: The 'slice' $G_{c,a}$ of a tree generator.

Proof. Let T be a Λ-labeled Σ-colored rank n tree. We set $\Lambda' = \Lambda \cup \bar{\Lambda} \cup \{\#\}$ and we denote by G the extension of T with backward edges and loops, namely $G = h_\Lambda^{-1}(T)$, and by T' the Λ'-labeled Σ-colored tree resulting from the unfolding of G, namely, $T' = \mathcal{U}nf(G) = \mathcal{B}ack\mathcal{U}nf(T)$. Note that for every vertex v' in T', there is a (unique) corresponding path $\pi_{v'}$ in G starting from the source vertex. Hence, we can denote by $\overleftarrow{v}'$ the last vertex of G (or, equivalently, the last vertex of T) along the path $\pi_{v'}$. Now, let Π be the factorization of T' with respect to Λ such that (i) $\mathcal{D}om(\Pi)$ is the set of all vertices v' of T' for which there is no ancestor $v'' \neq v'$ of v' in T' such that $\overleftarrow{v}' = \overleftarrow{v}''$ and (ii) (v', v'') is an a-labeled edge in Π iff $(\overleftarrow{v}', \overleftarrow{v}'')$ is an a-labeled edge in T. Notice that Π can have unbounded degree. However, for every pair of vertices u, u' in Π, if the sequences of labels along the access paths of u and u' in Π are the same, then, because the corresponding paths in G lead to the same vertex, we have that $(T')^{\downarrow u}$ and $(T')^{\downarrow u'}$ are isomorphic trees, and, in particular, $(T')^+_\Pi[u] = (T')^+_\Pi[u']$. Therefore, by identifying access paths having the same sequence of labels, we can provide a (deterministic) retraction of T' with respect to Π and any given tree automaton M. Moreover, we can prove that such a retraction is a rank n tree, from which it follows that T' is a rank $n+1$ tree.

Let assume that Σ and Λ are disjoint sets. We define $\Sigma' = \Sigma \cup \Lambda \cup \{x\}$, where x is a fresh color not belonging to $\Sigma \cup \Lambda$ (such a color x is be used as an insertion variable for second-order tree substitutions). Then, for each color $c \in \Sigma$ and for each marker $a \in \Lambda$, let $\gamma_{c,a}$ be the regular tree insertion specified by the unfolding from c of the finite Λ-labeled Σ'-colored graph $G_{c,a} = (V, (E_{a'})_{a' \in \Lambda}, C)$ (see Figure 3.15), where

- $V = \{x, c\} \cup \Lambda$,
- $E_\# = \{(c, c)\}$, $E_a = \{(c, a), (x, c)\}$, $E_{\bar{a}} = \{(c, x)\}$, and for every $a' \neq a$, $E_{a'} = \{(c, a)\}$ and $E_{\bar{a}'} = \emptyset$,
- the coloring of $G_{c,a}$ is given by the identity function, namely, the vertex x is colored by x, the vertex c is colored by c, and each vertex $a \in \Lambda$ is colored by a.

For the sake of brevity, given a vertex u of Π (resp., a finite path π in Π), we shall denote by T'_u (resp., by T'_π) the marked factor of T' rootead u (resp., at the last vertex along π). Note that, if $c = T(\varepsilon)$, then, for any $a \in \Lambda$, $T'_\varepsilon = \gamma_{c,a}(\emptyset)$, where $\emptyset$ denotes the empty tree. Moreover, it is easy to verify that, if $u \in \mathcal{D}om(\Pi)$, u' is an a-successor of u in Π, and $c = T(\overleftarrow{u}')$, then $T'_{u'} = \gamma_{c,a}(T'_u)$. Thus, the marked factor T'_u depends only on the labels along the access path of u in Π. Now, by Corollary 3.4.10, given any regular tree insertion γ and any tree automaton M, we can denote

by γ^M a computable function over $\mathscr{T}_{M,\Lambda}$ such that for every Λ-augmented tree T'', $\gamma^M\big([T'']_M\big) = \big[\gamma(T'')\big]_M$ holds. In particular, this implies that (i) $\gamma^M_{c,a}(t_\emptyset)$ is an M,Λ-type of T'_ε, provided that $t_\emptyset$ is the basic M,Λ-type of the empty tree, and (ii) for every a-labeled edge (u,u') in Π, $\gamma_{c,a}(t_u)$ is an M,Λ-type of $T'_{u'}$, provided that $T(\vec{u}') = c$ and t_u is an M,Λ-type of T'_u. Since for every $u \in \mathcal{D}om(\Pi)$ the above defined M,Λ-type of T'_u depends only on the sequences of labels and colors along the access path of $\vec{u}$ in T, we can denote it by $C(\vec{u})$. In particular, we can give the function $C : \mathcal{D}om(T) \to \mathscr{T}_{M,\Lambda}$ the status of a deterministic Λ-labeled $\mathscr{T}_{M,\Lambda}$-colored tree. Moreover, such a tree is clearly a retraction of T' with respect to M and Π and it can be obtained as a finite-state recoloring of T via a Mealy tree automaton $N = (S, \Sigma \cup \mathscr{T}_{M,\Lambda}, \Lambda, \delta, s_0, \Omega)$ such that

- $S = \mathscr{T}_{M,\Lambda} \times \Lambda$,
- for every $s = (t,a) \in S$, $c \in \Sigma$, and $a' \in \Lambda$, $\delta(s,c,a') = (t', a')$, where $t' = \gamma^M_{c,a}(t)$,
- s_0 is any element of the form $(t_\emptyset, a)$, with $t_\emptyset$ being the basic M,Λ-type of the empty tree and $a \in \Lambda$,
- for every $s = (t,a) \in S$, $\Omega(s,c) = \gamma^M_{c,a}(t)$.

By Theorem 3.4.16, C is a rank n tree whose footprint ξ_M (recall that C depends on the automaton M) can be computed on the grounds of the footprint of T. This implies that the computabel function ξ mapping a tree automaton M to ξ_M is a footprint of T' and hence T' is a rank $n+1$ tree. ∎

Corollary 3.5.13 *For every $n \in \mathbb{N}$, the level n tree generators are rank n trees.*

Proof. This is an immediate consequence of Theorem 3.5.12 and of the definition of tree generator. ∎

We conclude the section with the following remark. Since inverse rational forward mappings are special cases of MSO-definable transductions preserving bisimilarity of graphs, by a result of Colcombet and Löding [29], it follows that any inverse rational forward mapping h^{-1} can be implemented by a tree transducer with rational lookahead T_h. For the sake of completeness, we directly provide such a transducer T_h, given any rational Λ-forward mapping $h : \Lambda' \to \mathscr{P}(\Lambda^*)$. We fix an arbitrary order $<$ on the set Λ and we denote by $M_b = (S_b, \Lambda, \delta_b, s_{b,0}, F_b)$, for every $b \in \Lambda'$, a deterministic finite-state automaton recognizing the language $h(b)$. We then define:

- $Z = \bigcup_{b \in \Lambda'} \{(b,s) : s \in S_b\}$ (here we assume that the sets S_b, where b ranges over Λ', are disjoint);
- $\Delta = Z \times \Lambda$;
- $I = \Sigma \cup (Z \times \Lambda)$ (here we assume that Σ and $Z \times \Lambda$ are disjoint sets);
- for every index $i \in I$, if $i = c \in \Sigma$, then L_i is the rational language consisting of all and only the trees that have a c-colored root; otherwise, if $i = ((b,s),a) \in Z \times \Lambda$, then L_i is the rational language consisting of all and only the trees that contain a vertex of the form $a \cdot w$, where $a \cdot w$ is accepted by M_b starting from the state s;
- $\lambda : \Delta \to \Lambda$ such that $\lambda((b,s),a) = a$;
- $\beta : \Delta \to S$ such that $\beta((b,s),a) = (b,s)$,

- for every $(b, s) \in Z$ and for every subset J of I containing a unique index $c \in \Sigma$, $R_{s,b,J}$ is the Λ'-labeled Δ-augmented tree defined as follows
  1. if $s \notin F_b$ (namely, if s is not a final state of M_b), then $\mathcal{D}om(R_{s,b,J}) = \{\varepsilon\}$ and $R_{s,b,J}(\varepsilon) = \big(b, \delta_b(s, a), a\big)$, where a is the least element of Λ (according to the order $<$) such that $((b, s), a) \in J$;
  2. if $s \in F_b$, then $\mathcal{D}om(R_{b,s,J}) \subseteq \{\varepsilon\} \cup \Lambda$, $R_{b,s,J}(\varepsilon) = c$, and for every $b' \in \Lambda'$, $b' \in \mathcal{D}om(R_{b,s,J})$ and $R_{b,s,J}(b') = \big(b', \delta_{b'}(s_{b',0}, a), a\big)$ hold iff a is the least element of Λ (according to the order $<$) such that $((b', s_{b',0}), a) \in J$ (if such an element a does not exist, then $b' \notin \mathcal{D}om(R_{b,s,J})$).

Now, we define the tree transducer $M_h = (Z, \Sigma, \Delta, \Lambda \cup \Lambda', \lambda, \mathcal{L}, \delta, z_0)$:

- $\mathcal{L} = \{L_i : i \in I\}$,
- $\delta(z, J) = (R_{s,b,J}, \beta)$ for every $z = (s, b) \in Z$ and for every subset J of I containing a unique index from Σ (note that the cases where $|J \cap \Sigma| \neq 1$ are not relevant),
- $z_0 = (b, s)$, with b being an arbitrary element of Λ' and s being an arbitrary final state of M_b.

We let the reader check that M_h generates exactly $h^{-1}(T)$ on input T, provided that both T and $h^{-1}(T)$ are deterministic trees. Intuitively, the transducer M_h generates the tree $h^{-1}(T)$ in a top-down fashion by exploiting the facility of rational lookahead to find, for each b-labeled edge in $h^{-1}(T)$, a path in a suitable subtree of T of the form $a \cdot w \in h(b)$ (such a path can be assumed to be the least path according to the lexicographic order induced by $<$).

Given the above results and the fact that tree transductions with rational lookahead are subsumed by regular tree morphisms and finite-state recolorings with rational lookahead, we know that, if rank n trees were closed under finite-state recolorings with rational lookahead, then the class of reducible trees would capture *all* deterministic trees in the Caucal hierarchy. From this perspective, our conjecture stating that rank n trees are closed under finite-state recolorings with rational lookahead gains a crucial role in relating our approach with that of Caucal. Furthermore, we already know that rank n trees are closed under finite-state recolorings with bounded lookahead. This implies that reducible trees capture all deterministic trees obtained by iterating unfoldings and inverse *finite* mappings (i.e., functions of the form $h : \Lambda' \to \mathscr{P}(\Lambda_+^*)$ such that $h(b)$ is a finite language for all $b \in \Lambda'$), starting from regular trees. However, such a class of trees is properly included in the Caucal hierarchy (for instance, the tree whose branches are all and only the words of the form $w \cdot w$, with $w \in \Lambda^*$, belongs to the third level of the Caucal hierarchy and it cannot be obtained via inverse finite mappings and unfoldings starting from regular trees). Finally, it is worth mentioning that the proposed notion of reducible tree captures also several deterministic trees *outside* the Caucal hierarchy, like, for instance, the tree T_{tow} defined in Example 3.4.12. By applying unfoldings and inverse finite mappings to these trees, one obtains an infinite (strictly increasing) hierarchy of deterministic trees enjoying a decidable MSO theory.

3.5.3 Layered temporal structures

Here we consider (ω-)layered structures, which have been originally introduced by Montanari et al. in [77, 78, 81] in order to model finite and infinite hierarchies of time granularities. The focus is on three kinds of layered structures: the k-refinable n-layered structure, abbreviated n-LS, 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 downward unbounded k-refinable ω-layered structure, abbreviated DULS, which consists of an infinite number of arbitrarily fine layers, and the upward unbounded k-refinable ω-layered structure, abbreviated UULS, which consists of an infinite number of arbitrarily coarse layers. In their original formulation, layered 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 seen 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 structures have been shown to be expressive enough to capture meaningful temporal properties of reactive systems (such as 'P holds at all time points k^i, with $i \geq 0$' or 'P holds 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 showed 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 to the MSO theory of the infinite complete k-ary tree, and that of k-refinable UULS to the MSO theory of the k-ary systolic tree [80, 84, 79, 81].

In this section we introduce a new kind of structure, called k-refinable totally unbounded ω-layered structure, abbreviated TULS, which can be viewed as the composition of the k-refinable DULS and the k-refinable UULS. We show that such a kind of structure (extended with a suitable coloring predicate) embeds MSO logics of both DULS and UULS. Then, by exploiting the method previously presented in this chapter, we establish the decidability of the model checking problem for MSO logic interpreted over TULS, thus generalizing previous results in the literature. Finally, we give new decidability results for the chain fragment of MSO logic interpreted over TULS extended with either the equi-level or the equi-column predicate.

In the following, we denote by Λ_k the set $\{a_1, ..., a_k\}$ consisting of exactly k edge labels.

Definition 3.5.14 *A* k-refinable layered structure *is a graph* $G = \big(S, <, (F_a)_{a \in \Lambda_k}\big)$, *where* $S = \bigcup_{i \in I} L_i$, $I \subseteq \mathbb{Z}$, $L_i = \{(i, n) : n \in \mathbb{N}\}$, $<$ *is a suitable total order over* S, *and, for all* $a \in \Lambda_k$, F_a *is a function that maps the element* (i, n) *to the element* $(i + 1, kn + j - 1)$ *(if such an element belongs to* S*).*

For every $i \in I$, L_i is called a *layer* of the structure and, for every $j \in [k]$, F_{a_j} is called the *j-th projection function* of the structure, since it maps elements of a given

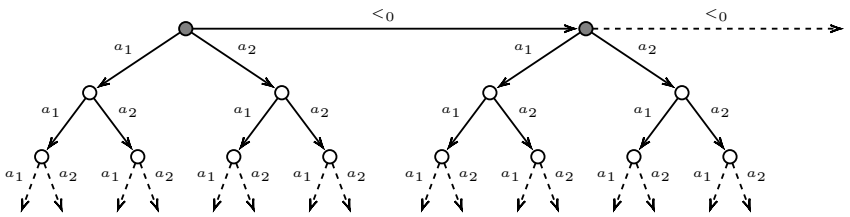

Figure 3.16: The 2-refinable DULS.

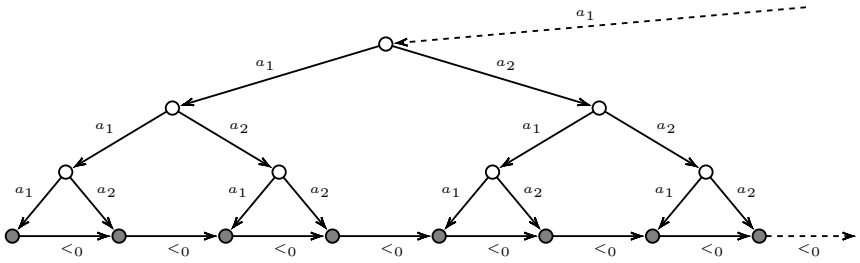

Figure 3.17: The 2-refinable UULS.

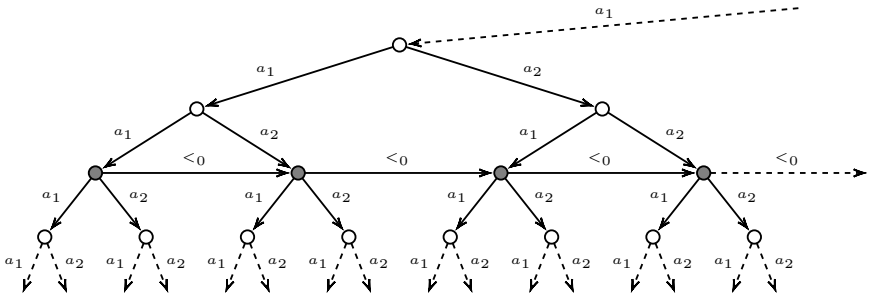

Figure 3.18: The 2-refinable TULS.

layer to elements of the immediately finer layer (if any). For the sake of simplicity, here we restrict the attention to 2-refinable layered structures only (the general case of k-refinable layered 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 structure $G = \big(S, <, (F_a)_{a\in\Lambda_k}\big)$, a *subtree of* G is a subgraph induced by the set of all vertices which are reachable from a designated source vertex $(i, n) \in S$ via the relations $F_{a_1}, ..., F_{a_k}$. Formally, the (2-refinable) DULS is obtained from Definition 3.5.14 by letting $I = \mathbb{N}$ and by defining $<$ as the total order induced by $\prec_0$, for elements belonging to a common subtree, and by the

linear order $<_0$ on the top layer L_0 (i.e., $(0,0) <_0 (0,1) <_0 (0,2) <_0 ...$), for elements belonging to distinct subtrees. The (2-refinable) UULS is obtained by letting $I = -\mathbb{N}$ and by defining $<$ as the total order induced by $\prec_1$. Finally, the (2-refinable) TULS is obtained by letting $I = \mathbb{Z}$ and by defining $<$ as the order induced by $\prec_1$. In Figures 3.16, 3.17, and 3.18 we depicted (parts of) the 2-refinable DULS, UULS, and TULS, with the projection functions and (the transitive reduction of) the ordering relation $<_0$ of the layer L_0, which is identified by gray colored vertices. If we denote by $\text{DULS}^{\not<}$ (resp., $\text{UULS}^{\not<}$, $\text{TULS}^{\not<}$) the downward (resp., upward, totally) unbounded layered structure devoid of the ordering relation $<$, then we easily see that $<$ is MSO-definable in $\text{UULS}^{\not<}$ and in the structure $\text{TULS}^{\not<}$, but not in $\text{DULS}^{\not<}$. On the other hand, the relation $<$ is MSO-definable in $\text{DULS}^{\not<}$ if we equip the structure with the partial order $<_0$ of the top layer. In particular, this means that the MSO logics of $\text{DULS}^{\not<}$, $\text{UULS}^{\not<}$, and $\text{TULS}^{\not<}$ are pairwise expressively non-comparable, since each of such structure allows one to capture properties that cannot be expressed in the other structures. Moreover, both downward unbounded and upward unbounded ω-layered structures, endowed with the usual ordering relation $<$, are MSO-definable in the structure $\text{TULS}^{\not<,L_0}$, which denotes the totally unbounded layered structure devoid of the ordering relation $<$ and extended 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). Given these results, we can focus our attention on the structure $\text{TULS}^{\not<,L_0}$ only (note that decidability results for $\text{TULS}^{\not<,L_0}$ can be easily transferred to the other structures by exploiting the MSO-compatibility of MSO-definable interpretations).

As a preliminary step, we show that the structure $\text{TULS}^{\not<,L_0}$ can be obtained via an MSO-definable interpretation (indeed an inverse rational mapping) from a suitable ternary colored tree. Such an embedding allows us to move from the setting of layered structures to the more standard framework of (deterministic colored) trees and hence to exploit the results given in Section 3.3. Let $\Lambda_2 = \{a_1, a_2\}$, $\Lambda_3 = \{a_1, a_2, a_3\}$, $G = \big(S, (F_a)_{a\in\Lambda_2}\big), L_0\big)$ be the structure $\text{TULS}^{\not<,L_0}$, and T be the Λ_3-labeled $\{0,1\}$-colored tree (see Figure 3.19) such that

- $\mathcal{D}om(T) = \Lambda_2^* \cup \big(\{a_3\}^+ \cdot \{a_2\} \cdot \Lambda_2^*\big)$ (here $\{a_3\}^+$ denotes the regular language consisting of all non-empty sequences of the form $a_3 \cdot ... \cdot a_3$),
- for every $v \in \mathcal{D}om(T)$, if v is the empty sequence or a sequence belonging to the language $\{a_3\}^n \cdot \{a_2\} \cdot \Lambda_2^n$, then $T(v) = 1$, otherwise $T(v) = 0$.

We denote by $E_{a_1}, E_{a_2}, E_{a_3}$ the successor relations of the tree T and by R_1 the unary predicate consisting of all and only the 1-colored vertices in T. Now, it is easy to see that G is the result of an MSO-definable interpretation of T that reverses the a_3-labeled edges and renames them by a_1, formally: $G = \mathcal{I}(T)$, where

- $\mathcal{I} = \big(\psi(x), \varphi_{F_{a_1}}(x,y), \varphi_{F_{a_2}}(x,y), \varphi_{L_0}(x)\big)$,
- $\psi(x) = \texttt{true}$,
- $\varphi_{F_{a_1}}(x,y) = E_{a_1}(x,y) \vee E_{a_3}(y,x)$,
- $\varphi_{F_{a_2}}(x,y) = E_{a_2}(x,y)$,
- $\varphi_{L_0}(x) = R_1(x)$.

Since MSO-definable interpretations are compatible with respect to MSO logic, from

the decidability of the MSO theory of T it would follow that $G = \mathrm{TULS}^{\not<,L_0}$ as well has a decidable MSO theory. In fact, it can be proved that T is a rank 1 tree and hence it enjoys a decidable MSO theory.

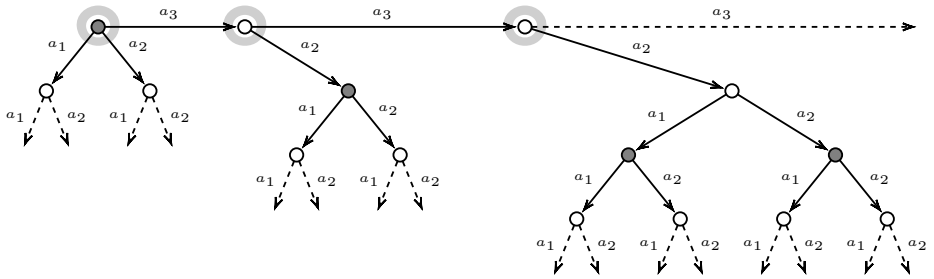

Figure 3.19: The ternary colored tree embedding $\mathrm{TULS}^{\not<,L_0}$.

Proposition 3.5.15 *The tree T embedding the structure $TULS^{\not<,L_0}$ is a rank 1 tree.*

Proof. We prove that T is a rank 1 tree by providing a suitable factorization of it and by showing that, for any given tree automaton M, the corresponding retraction is regular. The factorization Π of T is defined by letting $\mathcal{D}om(\Pi) = \{a_3\}^*$ and by labeling the induced edges with a single symbol b (see the circled vertices in Figure 3.19). We denote by T'_1 the infinite $\{a_1, a_2\}$-labeled $\{0,1\}$-colored tree such that $\mathcal{D}om(T'_0) = \Lambda_2^*$, $T'_1(\varepsilon) = 1$, and $T'_1(v) = 0$ for every $v \in \Lambda_2^+$. Then, for every $n > 1$, we recursively define the tree T'_n such that $T'_n = 0\langle x, x\rangle[\![T'_{n-1}/x]\!]$. It is easy to see that for every vertex $u = a_3^n \in \mathcal{D}om(\Pi)$, with $n > 0$, the marked factor $T^+_\Pi[u]$ is isomorphic to $0\langle\emptyset, T'_n, b\rangle$ (i.e., the tree obtained from the second-order tree substitution in $0\langle\emptyset, x, b\rangle$ of x by T'_n). Now, from properties of second-order tree substitutions and types (cf. Corollary 3.4.10), we know that, for any given tree automaton M, there is a computable function γ^M that maps an $M, \{b\}$-type of T'_n, with $n \geq 1$, to an $M, \{b\}$-type of T'_{n+1}. Therefore, there is an ultimately periodic sequence $t_0, t_1, t_2, \ldots$ such that for every $n \in \mathbb{N}$, t_n is an $M, \{b\}$-type of T'_n. This implies that there is a retraction C of T with respect to M and Π which is a memoryless recoloring of the unary tree C' defined by $\mathcal{D}om(C') = \{b\}^*$ and $C'(b^n) = t_n$ and hence T is a rank 1 tree. ■

Corollary 3.5.16 *The structure $TULS^{\not<,L_0}$ enjoys a decidable MSO theory.*

Proof. The result follows from Proposition 3.5.15 and Corollary 3.4.4, since MSO-definable interpretations preserve the decidability of the MSO theories of graphs. ■

As we previously mentioned, Corollary 3.5.16 subsumes previous results on the decidability of the MSO theories of DULS and UULS. In [59], model checking problems for (fragments of) MSO logic over n-LS, DULS, and UULS extended 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 $S = \{(i,n) : i \in I, n \in \mathbb{N}\}$ is the domain of the layered structure, with I being a subset of $\mathbb{Z}$, then $L = \big\{\big((i,n),(i,n')\big) : i \in I, n, n' \in \mathbb{N}\big\}$ and $C = \big\{\big((i,n),(i',n)\big) : i, i' \in I, n \in \mathbb{N}\big\}$. In [59] it has been showed 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. In that paper, the authors also show 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 monadic chain logic[5] interpreted over $\mathrm{TULS}^{\not<,L_0}$ extended 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 $\mathrm{TULS}^{\not<,L_0}$ holds even if we restrict to interpretations with chain quantifiers only, our result extends the above mentioned results and answers positively to the open problem for DULS^C. The following proof is partly based on a method introduced by Thomas in [111], which allows one to reduce the monadic chain logic interpreted over an expanded tree structure to the full MSO logic interpreted over a linear structure. As usual, we consider, for simplicity, 2-refinable layered structures.

Proposition 3.5.17 *The model checking for monadic chain logic interpreted over $\mathrm{TULS}^{\not<,L_0}$ extended with either the equi-level or the equi-column predicate is decidable.*

Proof. The idea is to translate a given monadic chain sentence φ over $\mathrm{TULS}^{\not<,L_0,L}$ (resp., $\mathrm{TULS}^{\not<,L_0,C}$) to an equi-satisfiable MSO sentence $\vec{\varphi}$ over $(\mathbb{Z},<)$ (resp., $(\mathbb{Z} \cup \{\infty\}, <, neg)$, where ∞ denotes a special element not belonging to $\mathbb{Z}$ and neg denotes the relation $\{(z,-z) : z \in \mathbb{Z}\}$). The translation is achieved by first encoding each set variable X with a suitable tuple of vertex/set variables and then by properly rewriting atomic formulas. Let first consider the translation for the structure $\mathrm{TULS}^{\not<,L_0,L}$. Without loss of generality, we can assume that quantifications are restricted to non-empty chains and that any formula contains only set variables and atoms of the following forms:

- $X \subseteq Y$, meaning that 'the chain X is included in the chain Y',
- $E_{a_j}(X,Y)$, meaning that 'X, Y are singletons of the form $\{x\}, \{y\}$, where y is the j-th successor of x',
- $L_0(X)$, meaning that 'X is a singleton of the form $\{x\}$, where $x \in L_0$',
- $L(X,Y)$, meaning that 'X, Y are singletons of the form $\{x\}, \{y\}$, where x and y belong to the same layer'.

Let $S = \bigcup_{i\in\mathbb{Z}} L_i$ be the domain of the structure $\mathrm{TULS}^{\not<,L_0,L}$. As a preliminary remark, note that for every non-empty chain C and for every $i \in \mathbb{Z}$, there is at most

[5] Here a chain is any subset of a path in the k-refinable layered structure consisting of Λ_k-labeled edges only. The chain fragment of MSO logic is obtained by instanciating set variables with chains only.

one $n \in \mathbb{N}$ such that $(i,n) \in C$. Now, we explain how one can encode a generic non-empty chain C with two subsets Z_C and W_C of $\mathbb{Z}$. We say that $P \subseteq S$ is a *cover* of C if P is a maximal path including C, namely, if $C \subseteq P$ and for every $i \in \mathbb{Z}$, there is exactly one $n \in \mathbb{N}$ such that $(i,n) \in P$. We denote by P_C the leftmost cover of C, that is, the (unique) cover P_C such that, whenever i is the greatest integer for which there is n satisfying $(i,n) \in C$, then every descendant of (i,n) along P_C is of the form $\big(i', 2^{i'-i}n\big)$, with $i' \geq i$. Then, we define Z_C and W_C in such a way that for every $i \in \mathbb{Z}$,

i) $i \in Z_C$ iff there is a (unique) *odd* index $n \in \mathbb{N}$ such that $(i,n) \in P_C$ (namely, (i,n) is the target of an a_2-labeled edge along the path P_C),
ii) $i \in W_C$ iff there is a (unique) index $n \in \mathbb{N}$ such that $(i,n) \in C$ (namely, C intersects the layer L_i).

Note that the encoding (Z_C, W_C) determines in an unambiguous way the non-empty chain C. Moreover, we can map the above construction in the logic. Precisely, for each chain variable X, we introduce two set variables Z_X and W_X (to be instanciated by sets of integers) and we rewrite a given chain formula φ to $\vec{\varphi}$ inductively as follows:

- if φ is of the form $X \subseteq Y$, then we set $\vec{\varphi}$ to be $W_X \subseteq W_Y \ \wedge \ \big(Z_X = Z_Y \ \vee \ \exists\, w \in W_X.\ (\forall\, w' \in W_X.\ w \geq w' \ \wedge \ \forall\, z \leq w.\ z \in Z_X \leftrightarrow z \in Z_Y)\big)$ (read 'W_X is included in W_Y and either $Z_X = Z_Y$ holds or W_X has a maximum element w and $Z_X \cap \{z \leq w\} = Z_Y \cap \{z \leq w\}$');
- if φ is of the form $E_{a_1}(X,Y)$, then we set $\vec{\varphi}$ to 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 set $\vec{\varphi}$ to 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 set $\vec{\varphi}$ to be $W_X = \{0\}$;
- if φ is of the form $L(X,Y)$, then we set $\vec{\varphi}$ to be $\exists\, w.\ W_X = W_Y = \{w\}$;
- if φ is of the form $\varphi_1 \vee \varphi_2$, then we set $\vec{\varphi}$ to be $\vec{\varphi}_1 \vee \vec{\varphi}_2$;
- if φ is of the form $\neg\varphi'$, then we set $\vec{\varphi}$ to be $\neg\vec{\varphi}'$;
- if φ is of the form $\exists\, X.\ \varphi'$, then we set $\vec{\varphi}$ to be $\exists\, Z_X.\ \exists\, W_X.\ \vec{\varphi}' \ \wedge \ W_X \neq \emptyset \ \wedge \ \forall\, w \in W_X.\ (\forall\, w' \in W_X.\ w \geq w') \ \rightarrow \ (\forall\, z \in Z_X.\ w \geq z)$ (read 'there are Z_X and $W_X \neq \emptyset$ satisfying $\vec{\varphi}'$ and, whenever W_X has a maximum element w, then w is greater than or equal to every element of Z_X', meaning that the cover corresponding to the encoding of the chain X is the leftmost one).

It is routine to check that for every monadic chain sentence φ, φ holds in the structure $\mathrm{TULS}^{\not<,L_0,L}$ iff $\vec{\varphi}$ holds in $(\mathbb{Z},<)$. Therefore, from the decidability of the MSO theory of $(\mathbb{Z},<)$ it follows that the model checking problem for the monadic chain logic interpreted over $\mathrm{TULS}^{\not<,L_0,L}$ is decidable.

As for the structure $\mathrm{TULS}^{\not<,L_0,C}$, in order to make it possible to check whether two vertices lies on the same diagonal, we need to encode chains in a slightly different way. As in the previous case, we restrict to an equivalent language where formulas are build up starting from atoms of the form $X \subseteq Y$, $E_{a_j}(X,Y)$, $L_0(X)$, $C(X,Y)$ and existential quantifiers over non-empty chains. Let $S = \bigcup_{i\in\mathbb{Z}} L_i$ be the domain of the structure $\mathrm{TULS}^{\not<,L_0,C}$. We encode a generic non-empty chain C with an integer

s_C and three subsets Z_C, W_C, Q_C of $\mathbb{N}$. We denote by P_C the rightmost cover of C, formally, the superset of C that contains exactly one element (i,n) for each $i \in \mathbb{Z}$ and such that, whenever i is the greatest integer for which there is n satisfying $(i,n) \in C$, then every descendant of (i,n) along P_C is of the form $\big(i', 2^{i'-i}(n+1)-1\big)$, with $i' \geq i$. Then, we distinguish between two cases: either P_C coincides with the leftmost branch of the structure $\mathrm{TULS}^{\not<, L_0, C}$ (this happens when C is a downward infinite chain lying entirely on the leftmost branch), or there is a maximum index $i \in \mathbb{Z}$ such that $(i,0) \in P_C$. In the former case, we set $s_C = \infty$, $Z_C = \emptyset$, $W_C = \{i \in \mathbb{N} : (i,0) \in C\}$, and $Q_C = \{i \in \mathbb{N}^+ : (-i,0) \in C\}$. In the latter case, we define s_C as the maximum $i \in \mathbb{Z}$ such that $(i,0) \in P_C$ and we define $Z_C, W_C, Q_C \subseteq \mathbb{N}$ as follows:

- $i \in Z_C$ iff there is a (unique) *odd* index $n \in \mathbb{N}$ such that $(s_C + i, n) \in P_C$ (namely, $(s_C + i, n)$ is the target of an a_2-labeled edge along the path P_C),
- $i \in W_C$ iff there is a (unique) index $n \in \mathbb{N}$ such that $(s_C + i, n) \in C$ (namely, C intersects the layer L_{s_C+i}),
- $i \in Q_C$ iff $i > 0$ and $(s_C - i, 0) \in C$ (namely, C intersects the layer L_{s_C-i}).

Notice that, in both cases, the encoding (s_C, Z_C, W_C, Q_C) uniquely determines the non-empty chain C. 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 rewrite a chain formula φ to $\vec{\varphi}$ inductively as follows:

- if φ is of the form $X \subseteq Y$, then we set $\vec{\varphi}$ to be $W_X \subseteq W_Y \ \wedge\ Q_X \subseteq Q_Y \ \wedge\ \big((s_X = s_Y = \infty) \ \vee\ (s_X = s_Y \neq \infty \ \wedge\ Z_X = Z_Y) \ \vee\ (s_X = s_Y \neq \infty \ \wedge\ \exists\, w \in W_X.\, (\forall\, w' \in W_X.\, w \geq w' \ \wedge\ \forall\, z \leq w.\, z \in Z_X \leftrightarrow z \in Z_Y))\big)$;
- if φ is of the form $E_{a_1}(X,Y)$, then we set $\vec{\varphi}$ to 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 set $\vec{\varphi}$ to 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 set $\vec{\varphi}$ to be $W_X = \{\mathit{neg}(s_X)\} \ \wedge\ Q_X = \emptyset$;
- if φ is of the form $C(X,Y)$, then we set $\vec{\varphi}$ to 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 set $\vec{\varphi}$ to be $\vec{\varphi}_1 \vee \vec{\varphi}_2$;
- if φ is of the form $\neg\varphi'$, then we set $\vec{\varphi}$ to be $\neg\vec{\varphi}'$;
- if φ is of the form $\exists\, X.\, \varphi'$, then we set $\vec{\varphi}$ to be $\exists\, s_X.\, \exists\, Z_X, W_X, Q_X \subseteq \mathbb{N}.\, \vec{\varphi}' \ \wedge\ W_X \cup Q_X \neq \emptyset \wedge \big(s_X = \infty \wedge Z_C = \emptyset\big) \vee \big(s_X \neq \infty \wedge \forall\, w \in W_X.\, (\forall\, w' \in W_X.\, w \geq w') \ \rightarrow\ (\forall\, z > w.\, z \in Z_X)\big)$ (read 'there are 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 $Z_C = \emptyset$, or $s_X \neq \infty$ and, whenever W_X has a maximum element w, then Z_X contains all elements greater than w, meaning that either the chain X is bounded from below, and in that case $Z_C = \emptyset$, or the cover corresponding to the encoding of X is the rightmost one).

Again, it is routine to check that for every monadic chain sentence φ, φ holds in the 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, from which it would follow that the structure $\mathrm{TULS}^{\not<, L_0, L}$ has a decidable model checking problem for monadic chain logic. In fact, the relational structure $(\mathbb{Z} \cup \{\infty\}, <, \mathit{neg})$ can be obtained from

$(\mathbb{N}, <)$ via an MSO-definable interpretation. Formally, we denote by *even* and by *odd* the (MSO-definable) unary predicates $\{2n : n \in \mathbb{N}\}$ and $\{2n+1 : n \in \mathbb{N}\}$, respectively, and we define the MSO-definable interpretation $\mathcal{I} = \big(\psi(x), \varphi_<(x,y), \varphi_{neg}\big)$, where

- $\psi(x) = \texttt{true}$,
- $\varphi_<(x,y) = (x = 1 \wedge y = 2) \vee (odd(x) \wedge odd(y) \wedge x < y) \vee (x \neq 0 \wedge even(x) \wedge even(y) \wedge y < x)$,
- $\varphi_{neg}(x,y) = (x = y = 1) \vee (odd(x) \wedge x = y{+}1) \vee (x \neq 0 \wedge even(x) \wedge y = x{+}1)$.

Clearly, $(\mathbb{Z} \cup \{\infty\}, <, neg)$ is isomorphic to $\mathcal{I}\big((\mathbb{N}, <)\big)$ (the elements $2, 4, 6, ...$ of $(\mathbb{N}, <)$ are mapped to $0, 1, 2, ...$, the elements $1, 3, 5, ...$ of $(\mathbb{N}, <)$ are mapped to $-1, -2, -3, ...$, and the element 0 of $(\mathbb{N}, <)$ is mapped to ∞). Since MSO-definable interpretations preserve the decidability of MSO theories, we have that $(\mathbb{Z} \cup \{\infty\}, <, neg)$ has a decidable MSO theory. ■

3.6 Discussion

In this chapter we considered the problem of deciding MSO theories of colored tree structures.

In Section 3.2 we preliminary reviewed some commonly-used operations on colored trees, such as finite-state recolorings, tree substitutions, tree transductions, etc. We then compared these operations and gave characterizations for some of them (for instance, we showed that finite-state recolorings together with regular tree morphisms are as powerful as tree transductions).

In Section 3.3 we developed an automaton-based method to establish the decidability of MSO theories of colored tree structures. First, by taking advantage of well-known results from automata theory, we reduced the model checking problem for MSO logic to the acceptance problem for Muller tree automata. We showed that the latter problem is decidable for the class of regular colored trees. Then, we extended such a result to non-regular trees by exploiting a suitable notion of indistinguishability of trees with respect to tree automata. Such a notion allows the replacing of a tree by a retraction of it in such a way that one can reduce, in a uniform way, instances of the model checking problem to equivalent instances of the acceptance problem for tree automata over regular trees. It is worth mentioning that the proposed method generalizes the one developed by Carton and Thomas to decide the theory of the linear order $(\mathbb{N}, <)$ extended with suitable unary relations, that is, those relations which are encoded by residually ultimately periodic words [19, 20].

In Section 3.4 we showed that the automaton-based approach for the decidability of MSO logics works effectively for a large class of complex (possibly non-regular) trees, which we called reducible trees. As a matter of fact, by exploiting our decision method one can capture meaningful examples of relational structures enjoying decidable MSO theories. For instance, the linear order $(\mathbb{N}, <)$ extended with the 'flip' function [80, 114] can be defined by MSO formulas in a suitable reducible tree (i.e., the tree that embeds the so-called 2-refinable upward unbounded ω-layered temporal structure []). We then proved closure properties for the class of regular trees with re-

spect to several natural transformations of trees (for instance, finite-state recolorings with bounded lookahead, regular tree morphisms) and we extended such properties to the class of reducible trees. These results, besides showing the robustness of the class of reducible trees, provided a neat framework to reason on retractions of trees and to easily transfer decidability results. As for closure properties, we left open the problem of establishing the closure of reducible trees with respect to finite-state recolorings with rational lookahead. We believe that such a closure property could be proved by using an argument similar to the one we used for finite-state recolorings with bounded lookahead. If such a conjecture were proved, then, from the expressiveness results given in Section 3.2, reducible trees would turn out to be closed under tree transductoins with rational lookahead and, by the results given in Section 3.5.2, they would capture all deterministic trees in the Caucal hierarchy. As a matter of fact, we already know that there exist several deterministic trees (e.g. the tree T_{tow} defined in Example 3.4.12) that do not belong to the Caucal hierarchy, but can be handled by our method.

In Section 3.5 we compared the expressiveness of our approach with that of other frameworks proposed in the literature. Precisely, we showed that the class of reducible trees includes several interesting decidable structures, like the unfoldings of the deterministic context-free graphs, the algebraic trees, and the tree generators for the levels of the Caucal hierarchy. Moreover, we considered the model checking problem for MSO logics interpreted over the so-called multi-layered temporal structures. In order to generalize previous results in the literature, we introduced a new notion of multi-layered structure, called k-refinable totally unbounded ω-layered structure, abbreviated TULS. We showed that such a kind of structure (extended with a suitable coloring predicate) embeds MSO logics of both downward and upward ω-layered structures. Then, by exploiting the automaton-based approach described in Section 3.3, we established the decidability of MSO theories of TULS. Finally, we gave new decidability results for the chain fragments of MSO logics interpreted over TULS extended with either the equi-level or the equi-column predicate.

As for possible research directions, we are currently trying to extend the method to make it possible to deal with non-deterministic colored trees and, possibly, with colored graphs. To this end, one can look for either a more general notion of tree automaton (see, for instance, [115, 116]) or a more general notion of type, which, for instance, captures indistinguishability of relational structures with respect to MSO formulas rather than Muller tree automata. In this respect, the composition method of Shelah [101, 112] seems to be intimately related to our approach and hence a promising line of research. Another interesting problem is to extend the notion of reducible tree in order to capture the trees generated by the so-called higher-order recursive program schemes [44, 37, 68, 2, 69, 67], which are still an active area of research. In this respect, it is worth mentioning that the MSO theories of trees generated by (unsafe) higher-order recursive program schemes have been proved to be decidable only recently by Ong in [93].

Conclusions

The goal of this thesis was to exploit different classes of automata for modeling and reasoning on infinite complex systems. Its main contributions can be summarized as follows:

- We provided an efficient algorithm for the equivalence problem of Wijsen's string-based specifications of time granularities and we described suitable procedures that map Calendar Algebra expressions to equivalent string-based (and hence automaton-based) representations of time granularities.
- 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 single time granularities, and 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 representing and reasoning on (possibly infinite) sets of periodical time granularities. We characterized the subclass of Büchi automata that recognize exactly the regular ω-languages consisting of ultimately periodic words only (these languages represent possibly infinite sets of periodical time granularities) and we provided efficient solutions to several basic problems (the emptiness problem, the membership problem, the equivalence problem, and the size-optimization problem), as well as to the granularity comparison problem.
- We addressed the model checking problem for MSO logics over tree structures. We developed a general automaton-based method for establishing the decidability of MSO theories of deterministic vertex-colored trees. To this end, we introduced the class of reducible trees, which naturally extends the class of regular trees and enjoys a decidable model checking problem with respect to MSO logics.
- We identified several natural operations on trees (e.g., finite-state recolorings, tree transductions) and we proved closure properties of reducible trees with respect to such operations. Then, by exploiting our automaton-based approach and closure properties of reducible trees, we established the decidability of the model checking problem for MSO logics interpreted over a number of meaningful relational structures (e.g., unfolding of context-free graphs, algebraic trees, tree generators for the Caucal hierarchy).
- We defined k-refinable totally unbounded ω-layered structures, whose theories subsume the theories of previously known multi-layered structures (i.e., n-layered structures, downward unbounded and upward unbounded ω-layered structures). We proved the decidability of the MSO theories of these structures and the decidability of the chain fragments of these theories for k-refinable totally unbounded ω-layered structures extended with either the equi-level or the equi-column pred-

icate. Such results subsume and extend previous results in the literature.
Finally, some open problems deserve further investigations:

- to find a deterministic polynomial-time algorithm that solves the equivalence problem for RLA or to prove that such a problem is Co-NP-complete (the current algorithm is based on a reduction to the satisfiability problem for linear diophantine equations with bounds on variables and it works in non-deterministic polynomial time);
- to improve the algorithms for the complexity- and the size-optimization problems (currently, both algorithms take polynomial time, but the latter one can only compute size-optimal automata with respect to a proper subclass of RLA);
- to establish whether the hierarchy of rank n trees (whose union gives the class of reducible trees) is strictly increasing or not;
- to prove the closure of reducible trees with respect to finite-state recolorings with rational look-ahead (in virtue of the results given in Section 3.2 and in Section 3.5.2, a solution to such a problem would imply that the class of reducible trees captures all, but not only, the deterministic trees in the Caucal hierarchy);
- to extend the notion of reducible tree in order to capture trees generated by higher-order recursive program schemes;
- to generalize the automaton-based approach in order to make it possible to deal with model checking problems for non-deterministic colored graphs, rather than deterministic colored trees.

Bibliography

[1] K. Aardal, C. A.J. Hurkens, and A.K. Lenstra. Solving a system of linear diophantine equations with lower and upper bounds on the variables. *Mathematics of Operations Research*, 25(3):427–442, 2000.

[2] K. Aehlig, J.G. de Miranda, and C.-H. L. Ong. The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In *Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications (TLCA)*, volume 3461 of *Lecture Notes in Computer Science*, pages 39–54. Springer, 2005.

[3] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] K. Barthelmann. On equational simple graphs. Technical Report 9, Universität Mainz, Institut für Informatik, 1997.

[5] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress for Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[6] J.R. Büchi. State-strategies for games in $f_{\sigma\delta} \cap g_{\delta\sigma}$. *Journal of Symbolic Logic*, 48:1171–1198, 1983.

[7] C. Bettini and R. De Sibi. Symbolic representation of user-defined time granularities. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):53–92, 2000.

[8] C. Bettini, S. Jajodia, and X.S. Wang. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer-Verlag, July 2000.

[9] A. Blumensath. Prefix-recognizable graphs and monadic second-order logic. Technical Report AIB-06-2001, RWTH Aachen, 2001.

[10] A. Blumensath and E. Grädel. Automatic structures. In *Logic in Computer Science*, pages 51–62, 2000.

[11] K.S. Booth. Lexicographically least circular substrings. *Information Processing Letters*, 10(4/5):240–242, 1980.

[12] D. Bresolin, A. Montanari, and G. Puppis. Time granularities and ultimately periodic automata. Research Report UDMI/2003/24, Dipartimento di Matematica e Informatica, Università di Udine, Italy, 2003.

[13] D. Bresolin, A. Montanari, and G. Puppis. Time granularities and ultimately periodic automata. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA)*, volume 3229 of *LNCS*, pages 513–525. Springer-Verlag, 2004.

[14] J.A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. *Mathematical Theory of Automata*, 12:529–561, 1962.

[15] T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming*, volume 2719 of *LNCS*, pages 556–569, 2003.

[16] H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational ω-languages. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, volume 802 of *LNCS*, pages 554–566. Springer-Verlag, 1994.

[17] C. Campeanu, K. Culik II, K. Salomaa, and S. Yu. State complexity of basic operations on finite languages. In *Proceedings of the 4th International Workshop on Implementing Automata, WIA*, pages 60–70, 1999.

[18] A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *LNCS*, pages 112–123. Springer-Verlag, 2003.

[19] O. Carton and W. Thomas. The monadic theory of morphic infinite words and generalizations. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, LNCS, pages 275–284. Springer-Verlag, 2000.

[20] O. Carton and W. Thomas. The monadic theory of morphic infinite words and generalizations. *Information and Computation*, 176(1):51–65, 2002.

[21] D. Caucal. On the regular structure of prefix rewriting. In *Proceedings of the 15th International Colloquium on Trees in Algebra and Programming (CAAP)*, volume 431 of *LNCS*, pages 87–102. Springer-Verlag, 1990.

[22] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[23] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Proceedings of the 23nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1099 of *LNCS*, pages 194–205. Springer-Verlag, 1996.

[24] D. Caucal. On infinite terms having a decidable monadic theory. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, volume 2420 of *LNCS*, pages 165–176. Springer-Verlag, 2002.

[25] D. Caucal. On infinite transition graphs having a decidable monadic theory. *Theoretical Computer Science*, 290:79–115, 2003.

[26] D. Caucal and T. Knapik. An internal presentation of regular graphs by prefix-recognizable graphs. *Theoretical Computer Science*, 34(4):299–336, 2001.

[27] Loma Linda University Medical Center. Pediatric heart transplantation protocol, 2002.

[28] J. Clifford and A. Rao. A simple general structure for temporal domains. In C. Rolland and M. Leonard, editors, *Temporal Aspects of Information Systems*, pages 17–28. Elsevier Science Publishers, North-Holland, 1988.

[29] T. Colcombet and C. Löding. On the expressiveness of deterministic transducers over infinite trees. In *Proceedings of the 21st Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *LNCS*, pages 428–439. Springer-Verlag, 2004.

[30] C. Combi, M. Franceschet, and A. Peron. Representing and reasoning about temporal granularities. *Journal of Logic and Computation*, 14:51–77, 2004.

[31] H. Comon and V. Cortier. Flatness is not a weakness. In *Proceedings of the 14th International Workshop on Computer Science Logic*, volume 1862 of *LNCS*, pages 262–276. Springer-Verlag, 2000.

[32] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 1997.

[33] T.H. Cormen, C. Stein, R.L. Rivest, and C.E. Leiserson. *Introduction to Algorithms*. McGraw-Hill, 2001.

[34] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.

[35] B. Courcelle. The monadic second-order logic of graphs II: Infinite graphs of bounded tree width. *Mathematical Systems Theory*, 21:187–221, 1989.

[36] B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 193–242. Elsevier, 1990.

[37] B. Courcelle. The monadic second-order theory of graphs ix: Machines and their behaviors. *Theoretical Computer Science*, 151:125–162, 1995.

[38] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings, and unfoldings of transition systems. *Annals of Pure and Applied Logic*, 92:35–62, 1998.

[39] U. Dal Lago and A. Montanari. Calendars, time granularities, and automata. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD)*, volume 2121 of *LNCS*, pages 279–298. Springer-Verlag, 2001.

[40] U. Dal Lago, A. Montanari, and G. Puppis. Time granularities, calendar algebra, and automata. Research Report UDMI/2003/04, Dipartimento di Matematica e Informatica, Università di Udine, Italy, 2003.

[41] U. Dal Lago, A. Montanari, and G. Puppis. Towards compact and tractable automaton-based representations of time granularity. In *Proceedings of the 8th Italian Conference on Theoretical Computer Science (ICTCS)*, volume 2841 of *LNCS*, pages 72–85. Springer-Verlag, 2003.

[42] U. Dal Lago, A. Montanari, and G. Puppis. Towards compact and tractable automaton-based representations of time granularity. Submitted to Journal of Theoretical Computer Science, 2005.

[43] W. Damm. An algebraic extension of the Chomsky-hierarchy. In *Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 74 of *LNCS*, pages 266–276. Springer-Verlag, 1979.

[44] W. Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

[45] W. Damm and A. Goerdt. An automata-theoretic characterization of the OI-hierarchy. In *Proceedings of the 9th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 140 of *LNCS*, pages 141–153. Springer-Verlag, 1982.

[46] W. Damm and A. Goerdt. An automata-theoretical characterization of the OI-hierarchy. *Information and Control*, 71(1):1–32, 1986.

[47] S. Demri. LTL over integer periodicity constraints. Research Report LSV-03-13, Laboratoire Spécification et Vérification, ENS Cachan, France, 2003.

[48] S. Demri. LTL over integer periodicity constraints. In I. Walukiewicz, editor, *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 2987 of *LNCS*, pages 121–135. Springer-Verlag, 2004.

[49] C.E. Dyreson, W.S. Evans, H. Lin, and R.T. Snodgrass. Efficiently supporting temporal granularities. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):568–587, 2000.

[50] C.C. Elgot and M.O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *Journal of Symbolic Logic*, 31(2):169–181, 1966.

[51] J. Engelfriet. Bottom-up and top-down tree transformations – a comparison. *Mathematical Systems Theory*, 9(3):198–231, 1975.

[52] J. Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10:289–303, 1977.

[53] J. Engelfriet. Iterated stack automata and complexity classes. *Information and Computation*, 95(1):21–75, 1991.

[54] J. Engelfriet and L.M. Heyker. Hypergraph languages of bounded degree. *Journal of Computer and System Sciences*, 48(1):58–89, 1994.

[55] J. Engelfriet, L.M. Heyker, and G. Leih. Context-free graphs languages of bounded degree are generated by apex graph grammars. *Acta Informatica*, 31(4):341–368, 1994.

[56] J. Euzenat and A. Montanari. Time granularity. In M. Fisher, D. Gabbay, and L. Vila, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, pages 59–118. Elsevier, 2005.

[57] N.J. Fine and H.S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16:109–114, 1965.

[58] M. Franceschet and A. Montanari. Time granularities in databases, data mining, and temporal reasoning, by Claudio Bettini, Sushil Jajodia, and Sean X. Wang (book review). *The Computer Journal*, 45(6):683–685, 2002.

[59] M. Franceschet, A. Montanari, A. Peron, and G. Sciavicco. Definability and decidability of binary predicates for time granularity. In *Proceedings of the 10th International Symposium on Temporal Representation and Reasoning (TIME) and 4th International Conference on Temporal Logic (ICTL)*, pages 192–202. IEEE Computer Society Press, 2003.

[60] F. Gécseg and M. Steinby. Tree automata. *Akadémiai Kiodó*, 1984.

[61] S. Ginsburg and E. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

[62] T.A. Henzinger and R. Majumdar. A classification of symbolic transition systems. Research Report UCB/CSD-99-1086, EECS Department, University of California, Berkeley, 1999.

[63] T.A. Henzinger and R. Majumdar. A classification of symbolic transition systems. In *Proceedings of the 17th International Conference on Theoretical Aspects of Computer Science (STACS)*, volume 1770 of *LNCS*, pages 13–34. Springer-Verlag, 2000.

[64] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 2001.

[65] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[66] T. Kameda and P. Weiner. On the state minimization of nondeterministic finite automata. *IEEE Transactions on Computers*, C-19(7), 1970.

[67] Zurab Khasidashvili. On higher order recursive program schemes. In *Colloquium on Trees in Algebra and Programming*, pages 172–186, 1994.

[68] T. Knapik, D. Niwiński, and P. Urzyczyn. Deciding monadic theories of hyperalgebraic trees. *Lecture Notes in Computer Science*, 2044:253–267, 2001.

[69] T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1450–1461. Springer, 2005.

[70] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350, 1977.

[71] C. Löding. Optimal bounds for the transformation of omega-automata. In *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 1738 of *LNCS*, pages 97–109. Springer-Verlag, 1999.

[72] B. Leban, D. McDonald, and D. Foster. A representation for collections of temporal intervals. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, volume 1, pages 367–371. AAAI Press, 1986.

[73] H. Lin. Efficient conversion between temporal granularities. Technical Report 19, TimeCenter, 1997.

[74] O. Matz and A. Potthoff. Computing small nondeterministic automata. In *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, BRICS Notes Series, pages 74–88, 1995.

[75] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, Unpublished, 1988.

[76] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[77] A. Montanari. *Metric and Layered Temporal Logic for Time Granularity*. ILLC Dissertation Series 1996-02. Institute for Logic, Language and Computation, University of Amsterdam, 1996.

[78] A. Montanari, A. Peron, and A. Policriti. Theories of ω-layered metric temporal structures: Expressiveness and decidability. *Logic Journal of IGPL*, 7(1):79–102, 1999.

[79] A. Montanari, A. Peron, and A. Policriti. The taming (timing) of the states. *Logic Journal of IGPL*, 8(5), 2000.

[80] A. Montanari, A. Peron, and A. Policriti. Extending Kamp's theorem to model time granularity. *Journal of Logic and Computation*, 12(4):641–678, 2002.

[81] A. Montanari and A. Policriti. Decidability results for metric and layered temporal logics. *Notre Dame Journal of Formal Logic*, 37(2):260–282, 1996.

[82] A. Montanari and G. Puppis. Decidability of MSO theories of tree structures. In *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 3328 of *LNCS*, pages 430–442. Springer-Verlag, 2004.

[83] A. Montanari and G. Puppis. Decidability of MSO theories of tree structures. Research Report UDMI/2004/01, Dipartimento di Matematica e Informatica, Università di Udine, Italy, 2004.

[84] A. Montanari and G. Puppis. Decidability of the theory of the totally unbounded ω-layered structure. In *Proceedings of the 11th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 156–160, 2004.

[85] A. Montanari and G. Puppis. Deciding MSO theories of tree structures by tree contractions. Research Report UDMI/2005/10, Dipartimento di Matematica e Informatica, Università di Udine, Italy, 2005.

[86] A.W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Proceedings of the 5th Symposium on Computation Theory*, volume 208 of *LNCS*, pages 157–168. Springer-Verlag, 1984.

[87] D. Muller and P. Schupp. The theory of ends, pushdown automata, and second-order logics. *Theoretical Computer Science*, 37:51–75, 1985.

[88] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: new results and new proofs of the theorems of rabin, mcnaughton and safra. *Theoretical Computer Science*, 141(1-2):69–107, 1995.

[89] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 161–168, Baltimore, MD, 1992. ACM Press.

[90] P. Ning, S. Jajodia, and X.S. Wang. An algebraic representation of calendars. *Annals of Mathematics and Artificial Intelligence*, 36:5–38, 2002.

[91] H.J. Ohlbach. Calendar logic. In D.M. Gabbay, M. Finger, and M. Reynolds, editors, *Temporal Logic: Mathematical Foundations and Computational Aspects*, volume 2. Oxford University Press, 2000.

[92] H.J. Ohlbach and D.M. Gabbay. Calendar logic. *Journal of Applied Non-Classical Logics*, 8(4):291–324, 1999.

[93] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. Unpublished, 2006.

[94] R. Paige, R.E. Tarjan, and R. Bonic. A linear time solution to the single function coarsest partition problem. *Theoretical Computer Science*, 40:67–84, 1985.

[95] D. Perrin and P.E. Schupp. Automata on integers, recurrence distinguishability, and the equivalence and decidability of monadic theories. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, pages 301–304. IEEE Computer Society, 1986.

[96] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[97] W.C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4:257–287, 1970.

[98] S. Safra. On the complexity of ω-automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 319–327. IEEE, 1988.

[99] S. Safra. *Complexity of Automata on Infinite Objects*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1989.

[100] A.L. Semenov. Decidability of monadic theories. In *Proceedings of the Mathematical Foundations of Computer Science (MFCS)*, volume 176 of *LNCS*, pages 162–175. Springer-Verlag, 1984.

[101] S. Shelah. The monadic theory of order. *Annals of Mathematics*, 102:379–419, 1975.

[102] Y. Shiloach. Fast canonization of circular strings. *Journal of Algorithms*, 2(2):107–121, 1981.

[103] D. Siefkes. Decidable extensions of monadic second order successor arithmetic. In J. Dörr and G. Hotz, editors, *Automatentheorie und formale Sprachen*, pages 441–472. B.I. Hochschultaschenbücher, Mannheim, 1970.

[104] A. Sistla and E. Clarke. The complexity of propositional linear temporal logic. *Journal of the Association for Computing Machinery*, 32:733–749, 1985.

[105] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[106] R.T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.

[107] R.T. Snodgrass, I. Ahn, G. Ariav, D.S. Batory, J. Clifford, C.E. Dyreson, R. Elmasri, F. Grandi, C.S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T.Y.C. Leung, N. Lorentzos, J.F. Roddick, A. Segev, M.D. Soo, and S.M. Sripada. TSQL2 language specification. *SIGMOD Record*, 23(1):65–86, 1994.

[108] C. Stirling. Decidability of bisimulation equivalence for pushdown processes. Research Report EDI-INF-RR-0005, Division of Informatics, University of Edinburgh, 2000.

[109] L. Stockmeyer. *The Complexity of Decision Problems in Automata and Logic*. PhD thesis, M.I.T., 1974.

[110] J.W. Thatcher. Generalized sequential machine maps. *Journal of Computer and System Sciences*, 4:339–367, 1970.

[111] W. Thomas. Infinite trees and automaton definable relations over ω-words. In *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 415 of *LNCS*, pages 263–277. Springer-Verlag, 1990.

[112] W. Thomas. Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In *Structures in Logic and Computer Science*, pages 118–143, 1997.

[113] W. Thomas. Languages, automata, and logic. In G. Rozemberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer-Verlag, 1997.

[114] W. Thomas. Constructing infinite graphs with a decidable MSO-theory. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, LNCS, pages 113–124. Springer-Verlag, 2003.

[115] I. Walukiewicz. Monadic second-order logic on tree-like structures. In *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1046 of *LNCS*, pages 401–413. Springer-Verlag, 1996.

[116] I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275:311–346, 2002.

[117] S. Wöhrle. *Decision Problems over Infinite Graphs: Higher-order pushdown systems and synchronized products*. PhD thesis, RWTH Aachen, 2005.

[118] J. Wijsen. A string-based model for infinite granularities. In C. Bettini and A. Montanari, editors, *Proceedings of the AAAI Workshop on Spatial and Temporal Granularities*, pages 9–16. AAAI Press, 2000.

[119] G. Zhang. Periodic functions for finite semigroups. Unpublished, 1998.

[120] G. Zhang. Automata, boolean matrices, and ultimate periodicity. *Information and Computation*, 152(1):138–154, 1999.