University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Tomasz Idziaszek

Algebraic methods in the theory of infinite trees

PhD dissertation

Supervisor

dr hab. Mikołaj Bojańczyk

Institute of Informatics
University of Warsaw

February 2013

Author's declaration:
aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

February 20, 2013	.....
date	*Tomasz Idziaszek*

Supervisor's declaration:
the dissertation is ready to be reviewed

February 20, 2013	.....
date	*dr hab. Mikołaj Bojańczyk*

Abstract

In the thesis we explore an algebraic approach to regular languages of infinite trees. We have decided to take a two-pronged approach: to develop a concept of an algebraic structure for infinite trees and to use it to get an effective characterization for some properties of languages.

In the first part of the thesis we develop three algebras – two for languages of thin trees, and one for languages of arbitrary infinite trees. We show the correspondence between regular languages and languages recognized by our algebras.

An infinite tree is thin if it contains countably many infinite paths. Thin trees can be seen as intermediate structures between infinite words and infinite trees. Since the class of thin trees is simpler than the class of all trees, but at the same time robust and interesting, we focus in the thesis on thin trees. We believe that they are a good stepping stone on the way to understand regular languages of arbitrary infinite trees.

In the second part of the thesis we show some applications of the algebraic theory presented in the first part. We show how to decide whether a given regular language of thin trees is commutative, invariant under bisimulation, open in a certain topology, and definable in the temporal logic EF. For languages of arbitrary infinite trees we show how to decide definability in the logic EF.

Keywords: infinite trees, thin trees, regular languages, algebraic language theory, effective characterizations

ACM Subject Classification: F. Theory of Computation
F.1.1. Models of Computation
F.4.3. Formal Languages

Streszczenie

Celem niniejszej rozprawy jest zbadanie algebraicznego podejścia do języków regularnych drzew nieskończonych. Realizacja tego celu przebiegła dwutorowo: poprzez zaproponowanie algebraicznej struktury dla drzew nieskończonych oraz uzyskanie za jej pomocą efektywnej charakteryzacji dla pewnych własności języków.

W pierwszej części rozprawy przedstawiam trzy algebry – dwie dla języków drzew cienkich i jedną dla języków dowolnych drzew nieskończonych. Pokazuję również zależność pomiędzy językami regularnymi a językami rozpoznawanymi przez te algebry.

Nieskończone drzewo jest cienkie, jeśli zawiera przeliczalnie wiele gałęzi. Cienkie drzewa mogą być traktowane jako pośrednia struktura pomiędzy słowami nieskończonymi a drzewami nieskończonymi. Ponieważ klasa drzew cienkich jest prostsza niż klasa wszystkich drzew, a jednocześnie interesująca i o dobrych własnościach, część wyników rozprawy dotyczy drzew cienkich. Wierzę, że badanie drzew cienkich może nam pomóc lepiej zrozumieć ogólne drzewa nieskończone.

Druga część rozprawy dotyczy zastosowań metod algebraicznych opisanych w części pierwszej. Pokazuję jak rozstrzygać, czy dany język regularny cienkich drzew jest przemienny, zamknięty na bisymulację, otwarty w standardowej topologii oraz definiowalny w logice temporalnej EF. Ponadto dla języków dowolnych drzew nieskończonych pokazuję jak rozstrzygać ich definiowalność w logice EF.

Słowa kluczowe: nieskończone drzewa, cienkie drzewa, języki regularne, algebraiczna teoria języków, efektywne charakteryzacje

Klasyfikacja tematyczna ACM: F. Teoria obliczeń
F.1.1. Modele obliczeń
F.4.3. Języki formalne

Contents

Chapter 1

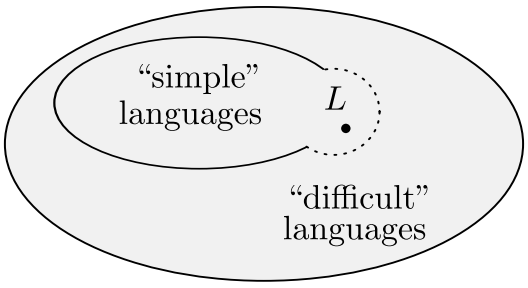
Introduction

The most fundamental concept in the formal language theory is, of course, the *language* itself.

Since it is unfeasible to represent a language explicitly as a set of words (the set could be extremely large or simply infinite), many approaches have been developed to deal with the problem of representation. On the one hand a representation should be succinct and legible: one should be able to easily grasp the idea behind a particular language or its main properties. On the other hand it should be practical: it should be a handy tool in construction of algorithms which decide whether a given language satisfies certain non-obvious properties.

Of course it is difficult to fulfill both of these requirements at the same time, so a human-readable representation could be impractical as an input for a computer algorithm and vice versa.

One way to obtain the first requirement is to divide the realm of formal languages into classes. Not all languages have the same "difficulty" and it is sensible to provide "simpler" forms of representations for "simpler" languages. One of such division is known as the Chomsky hierarchy [17], and on the bottom of it reside *regular languages*.



Of course there is a question whether a particular language L which comes from a class of "difficult" languages is in fact a "simple" language. We will call this the *language membership problem*. Due to Rice's theorem [32] this problem is in general undecidable. Thus it is also interesting whether language membership problem can be effectively tested for a particular choice of classes of "difficult" and "simple" languages.

Regular languages and their representations. In this thesis we assume that the class of "difficult" languages is the class of regular languages. Regular languages of words have a plethora of different representations, such as regular expressions, regular or prefix grammars, and read-only Turing machines. In the *theory of automata* they are represented

as non-deterministic or deterministic finite-state machines (automata). In *logic* they can be defined by formulas of monadic second-order logic (MSO). Finally, in *algebraic* terms they can be recognized by homomorphisms into finite semigroups or monoids. Different representations are useful for different purposes, and although the usefulness of a representation depends on the language itself, we can make some general observations: regular expressions and MSO formulas tend to be human-readable, whereas algebraic structures are most suited for algorithms. Finite-state automata place in the middle, trying to get the "best of both worlds", and serve as a common standard in the field.

We expect that a representation which is meant to be used as an input to algorithms will somehow reveal the inner structure of a language and provide a certain form of compositionality. Automata seem to be a reasonable choice, and in fact a large number of algorithms use them. However, these algorithms are often far from being elegant and they are polluted with technical details due to the nature of automata. That is where algebra comes into play. It turns out that certain properties of languages can be elegantly expressed as identities – simple equations on the elements of the semigroup which recognizes the language – and validity of these identities can be easily checked.

Effective characterizations of logics. Regular languages are precisely those which are definable by formulas of monadic second-order logic [15, 21, 41]. However, many languages can be conveniently described by formulas of simpler logics such as first-order logic or various temporal logics. We may therefore ask whether a certain language can be expressed in one of these simpler logics. We say that a logic has an *effective characterization* if the following decision problem is decidable: "given a regular language, decide if the language can be defined using a formula of the logic". Thus it is an instance of our language membership problem when the class of "difficult" languages is the class of all regular languages and the class of "simple" languages is the class of languages definable in the logic.

In the case of word languages we have many papers devoted to the topic of effective characterizations (see [38, 39, 18, 22, 37, 44, 34]), but arguably best known is the result of Schützenberger [33] and McNaughton and Papert [23], which states that the following four conditions are equivalent for a regular word language L:

(a) L can be defined by a formula of first-order logic (with order and label tests),

(b) L can be defined by a star-free regular expression (a subset of regular expressions, which does not allow the Kleene star, but allows to use complementation),

(c) the minimal deterministic finite-state automaton of L is counter-free (i.e. it does not contain a certain kind of loop),

(d) the syntactic monoid of L (i.e. the minimal monoid which recognizes L) does not contain a non-trivial group.

The above theorem gives a characterization of a certain class of languages in terms of (a) logic, (b) regular expressions, (c) automata, and (d) algebra. Conditions (c) and (d) can be effectively tested, therefore the theorem gives an effective characterization of first-order logic, as well as an effective characterization of star-free expressions.

Languages of infinite trees. Not only finite words can serve as an object to study in the theory of formal languages. The theory was generalized to include such objects as infinite words, finite trees, and (the object in which we will be interested most) infinite trees. We can ask the same questions which we asked in the word case: what representations of a language do we have and what are effective characterizations of wide-known logics?

But before we go further, we should determine what is a counterpart of a regular language? Indeed, this notion for more complicated objects is not yet as standardized as in finite words (see for example [16] and [6]). We will stick to one possible definition, namely that a language is regular if it can be defined by a formula of monadic second-order logic. In fact logic behaves the most robustly in generalization to the worlds of infinite words and (in)finite trees.

The notion of a finite-state automaton over finite trees is quite similar to the previous one. The automata for infinite objects are not so standardized, since dealing with infinite phenomena requires deeper insights for the acceptance condition, and we have different possibilities: Büchi, Muller, or parity automaton to name a few (see [40]). Moreover, in case of finite objects the syntactic algebra of a language is tightly connected with the minimal deterministic finite-state automaton recognizing the language. Unfortunately, deterministic automata for infinite objects are weaker that non-deterministic ones, thus there are regular languages which are not recognized by any deterministic automaton.

The algebraic structures for infinite words are Wilke algebras and ω-semigroups. They are quite natural extensions of semigroups for finite words. However, it is not obvious how to develop an algebra for finite trees, since there is no natural way to succinctly represent numerous ways a tree can grow: if we try to compose two trees horizontally, we get a forest, but it is not clear how to compose them vertically without a certain mark in the "upper" tree. One attempt of such algebra is forest algebra. We look closer at these algebraic structures in the next section.

A satisfying algebraic approach to infinite trees has not been developed yet. There are reasons to suppose that in fact it is a very difficult task. Fighting only one of the two aforementioned obstacles (lack of deterministic automata and natural composition on trees) is quite challenging, and infinite trees require to fight these two at the same time. In this thesis we try to make a step forward in this battle.

The contents of the thesis. The goal of this thesis is to explore an algebraic approach to regular languages of infinite trees. We have decided to take a two-pronged approach, which was reflected in the structure of the thesis:

(1) To develop a concept of an algebraic structure for infinite trees, by extending forest algebra to infinite trees – this is the subject of the first part of the thesis.

(2) To use the algebra to get an effective characterization for some logic – this is the subject of the second part.

A good effective characterization benefits the algebra. Effective characterizations are usually difficult problems, and require insight into the structure of the underlying algebra. We expect that as a byproduct of an effective characterization, we would discover what are the important ingredients of the algebra.

A good algebra benefits effective characterizations. A good algebra makes proofs easier and statements more elegant. We expect that an effective characterization would be a good

test for the quality of an algebraic approach. In the previously studied cases of (infinite and finite) words and finite trees, some of the best work on algebra was devoted to effective characterizations.

Algebras in the thesis

Since we are interested in developing an algebraic framework, we need to know what are the ingredients of an algebra. A new kind of algebra should be given by a set of sorts (we will work with multisort algebras), a certain number of operations on these sorts and a set of axioms that these operations should satisfy. For instance, in the case of finite words, there is only one sort, one operation (concatenation), and one axiom (associativity). Such a structure, of course, is called a semigroup. Given a finite alphabet A, the set of all non-empty words A^+ is simply the free semigroup. Regular languages are those that are recognized by morphisms from the free semigroup into a finite semigroup.

Since we want to develop an algebraic structure for recognizing formal languages of infinite trees, it is worthwhile to examine the counterpart structures for finite trees and infinite words.

The forest algebra. One question we must answer is: what type of trees (binary, ranked, or unranked) we consider. Tree automata are more naturally suited to work with binary and ranked trees, but it turns out that we obtain more robust algebra when we deal with unranked trees (i.e. the number of successors of each node is finite, but not bounded).

The idea used in [14] to deal with finite trees was to have a two-sorted algebra, where one sort described forests (sequences of unranked trees), and the other sort described contexts (forests with a hole, which is a special place to insert another forest or context). The use of forests makes horizontal composition a natural operation. Since we work with unranked objects, this operation is not restricted in any way. The use of contexts allows for a vertical composition (we compose a context with a forest). It is important to note, however, that none of sorts contains only trees, and in fact, technically speaking, forest algebra recognizes languages of forests, not trees.

Forest algebra has been successfully applied in a number of effective characterizations, including fragments of first-order logic [11, 10, 28] and temporal logics [4], see [5] for a survey. An important open problem is to find an effective characterization of first-order logic with the descendant relation (first-order with the successor relation was effectively characterized in [2]).

Algebras for infinite words. When developing an algebraic framework for infinite words we run into the following problem. For an alphabet A with at least two letters, the set A^ω of all infinite words is uncountable. On the other hand, the free object will be countable, as long as the number of operations is countable. There are two solutions to this problem: either have an uncountable number of operations, or have a free object that is different from A^ω. The first approach is called an ω-semigroup [25]. The second approach is called a Wilke algebra [43], [42]. Like in forest algebra, a Wilke algebra is a two-sorted object. The axioms and operations are designed so that the free object will have all finite words on the first sort, and all ultimately periodic words on the second sort. There is a reason why it is possible to ignore words that are not ultimately periodic. It

turns out that any regular language in A^ω is uniquely defined by the ultimately periodic words that it contains. In this sense, a morphism from the free Wilke algebra into a finite Wilke algebra contains all the information about a regular language of infinite words.

Infinite trees. It is clear what a finite tree is, but it may be unclear what trees we will call infinite. In the thesis the "set of infinite trees" contains trees which: are *unranked and finitely branching* (every node has a finite number of successors, but this number is not bounded), *ordered* (there is an order over the successors of every node), and *can have leaves* (and even be finite),

Regular trees. The set of all infinite trees is uncountable even if the alphabet A has only one letter. However, we can still follow the approach of Wilke to deal with this problem. The counterpart of an ultimately periodic word in the realm of infinite trees is called a *regular tree* and it is a tree which has a finite number of non-isomorphic subtrees.

The first contribution of the thesis is to propose an algebraic structure, called *regular-infinite-forest algebra* which is a two-sorted structure of infinite regular forests and infinite regular contexts. Although the set of all regular trees is countable, we were not able to propose an algebra which has a finite number of operations. This poses all sorts of problems.

It is difficult to present an algebra. One cannot, as with a finite number of operations, simply list the multiplication tables of the operations. Our solution is to give an algorithm that inputs the name of the operation and produces its multiplication table. In particular, this algorithm can be used to test validity of identities, since any identity involves a finite number of operations.

It is difficult to prove that something is a regular-infinite-forest algebra, since there are infinitely many axioms to check. Our solution is to define algebras as homomorphic images of the free algebra, which guarantees that the axioms hold. We give an algorithm that computes the syntactic algebra of a regular forest language.

We have proved that every regular language is recognized by a finite regular-infinite-forest algebra. A significant shortcoming is that we have not proved the converse. We do not, however, need the converse for effective characterizations. An effective characterization begins with a regular language, and tests properties of its syntactic algebra (therefore, algebras that recognize non-regular languages, if they exist, are never used).

Thin trees. Keeping the above issues in mind, we also followed another way. Instead of attacking the problem in its full generality, we focused on a certain class of infinite trees which is an intermediate step between finite trees and general infinite trees. The class is called "thin trees" and contains all trees with countably many infinite paths, which is equivalent to saying that they do not contain full binary tree as a minor. It turns out that this class is quite robust.

Indeed, for this class we propose two algebraic structures: *regular-thin-forest algebra* and *unrestricted-thin-forest algebra*. The former recognizes languages of regular thin forests and the latter recognizes languages of thin forests without restriction of regularity. The former has a finite number of operations and countably many axioms (although we can test them by checking a finite number of properties). Moreover, we prove the theorem that a language of thin trees is regular if and only if it is recognized by a morphism into a regular-thin-forest algebra.

It should be noted, that the above algebraic structures can not only be seen as generalizations of forest algebra, but also as generalizations of the structures for infinite words mentioned before. If we identify an infinite word with a thin tree of one infinite path (i.e. a tree which does not branch), then the generalization involves adding the operation of horizontal concatenation of trees. In such view, regular-thin-forest algebra generalizes Wilke algebras and unrestricted-thin-forest algebra generalizes ω-semigroups.

Universal algebra. As we can see, in the thesis we deal with several different algebraic structures. Three of them are new, and we have to present them together with definitions of certain notions. Therefore it is useful to gather common definitions and theorems in one place. To achieve this goal we formalize them in the language of universal algebra.

We present then the general notions behind *multisort algebras*. That includes homomorphisms, free objects, recognizability of a language, syntactic algebras and Myhill-Nerode relations. We provide also a notion of varietes of multisort algebras and prove a multisort version of Eilenberg Theorem [20] which shows a one-to-one correspondence between varietes of algebras and varietes of languages.

Logics characterized in the thesis

There is an ongoing project to effectively characterize certain properties of tree languages. A (slightly outdated) survey of effective characterizations of finite tree logics [5] states that arguably the most important question in case of trees is as follows: does first-order logic on trees have an effective characterization? This question has many variants, and we know an effective characterization to only one of them (trees with successor relation). As we see, even in the case of finite trees the problems are challenging.

Temporal logic EF. Therefore, since we are only beginning to explore the algebra for infinite trees, it is a good idea to start with some logic that is very well understood for finite trees. This is why for our case study we chose the temporal logic EF (which is equivalent to XPath with only descendant operations). For finite trees, this was one of the first nontrivial tree logics to get an effective characterization, for binary trees in [13], and for unranked trees in [14].

We were also curious how some properties of the logic EF would extend from finite trees to infinite trees. For instance, for finite trees, a language can be defined in the logic EF if and only if it is closed under EF-bisimulation (a notion of bisimulation that uses the descendant relation instead of the successor relation). We prove that in the case of infinite trees this condition is not sufficient.

Open languages. Of course, the question whether a given language is definable by a formula of a certain logic is not the only property worth considering. We are also interested in effective algorithms which test other properties of regular languages of infinite forests.

Some of them are connected with a topology imposed upon the set of all infinite forests. In this setting the basic property of a language is to test whether the language is an *open set* with respect to the topology. In the thesis we present an effective characterization of open sets for languages of thin forests.

Commutative languages and languages invariant under bisimulation. Last but not least, we present two simple but quite instructive properties of languages of thin forests. We show an effective characterization of languages which are commutative (i.e. closed under arbitrary rearranging of siblings) and languages which are invariant under bisimulation. It is interesting to note that the identity $h+g = g+h$ which characterizes all commutative languages of finite forests is too weak in case of thin forests. We investigate, however, also this "weaker" notion of commutativity.

Contribution

Part of the thesis is based on the conference paper [7] which is a joint work with my advisor Mikołaj Bojańczyk. In the paper we have developed the regular-infinite-forest algebra and presented an effective characterization of the temporal logic EF. Due to limited space we had to omit the most of the examples and intuitions, as well as formal presentation of some topics such as the model of forest automaton or the translation between forest automata and algebra. We compensate for it in the thesis.

The rest of the contribution was previously unpublished. For the algebraic part this includes the generalization of the theory of varietes to multisort algebras and the theory of algebraic structures for thin forests. For the effective characterizations part this includes the characterizations of the following properties of regular languages of thin forests: commutativity, invariance under bisimulation, being open in a certain topology, and definability in the logic EF.

The sections of this thesis which deals with thin forests are also part of the recently accepted conference paper [8], which is a joint work with Mikołaj Bojańczyk and Michał Skrzypczak.

The author was supported by ERC Starting Grant "SOSNA" No. 239850.

Acknowledgements

I would like to thank my advisor Mikołaj Bojańczyk, who introduced me to the field of algebraic theory of formal languages and then patiently guided me during my research, sharing his invaluable experience with me. I would also like to thank Krzysztof Diks for all his support and encouragement during my PhD studies and work at the University of Warsaw.

Finally, I wish to thank my family and my friends, who were constantly asking me when I planned to finish this thesis. Their inquisitiveness was a real motivational boost.

Chapter 2

Preliminaries

In this chapter we present some fundamental definitions which are used throughout the thesis. In section 2.1 we define trees, forests, contexts, and operations on them, as well as the basic properties of thin trees. In section 2.2 we recall two ways of recognition the languages – automata and logic. Finally, in section 2.3 we present some basic properties of finite semigroups.

2.1 Trees and forests

A *forest* is an ordered sequence of trees. Forests and trees can have both infinite and finite maximal paths. Each node must have finitely many siblings, but the number of siblings is not bounded (we call such forests *unranked*). Moreover, the children of each node are ordered. Forests are *labeled*, i.e. each node of a forest contains a label from a finite alphabet A. On figure 2.1 there is a forest with a single infinite path. The labels of nodes of this path spell out an infinite word $(aab)^\infty = aabaabaab\cdots$

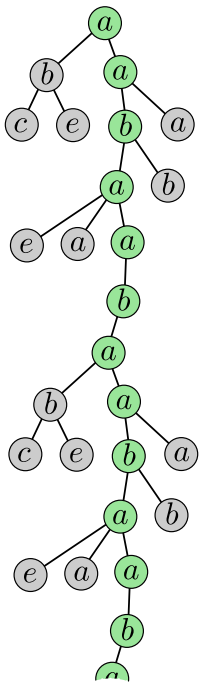


Figure 2.1

Formally, a forest over finite alphabet A is a partial map $t\colon \mathbb{N}^+ \to A$. We denote the domain of the map by $\mathrm{dom}(t)$ and refer to it as the set of nodes of the tree. The domain must be closed under non-empty prefixes, and such that for each $x \in \mathbb{N}^*$ the set $\{i \mid xi \in \mathrm{dom}(t)\}$ is a finite prefix of $\mathbb{N}$. The *empty forest* (the one with the empty domain) is denoted by 0. We use letters s, t to denote forests. The set of all forests over A is denoted by A^{For}.

A *forest language* over A is any subset of A^{For}.

If x, y are two nodes of forest t, we write $x \leq y$ if x is a prefix of y (we write $x < y$ if x is a proper prefix of y). If $x < y$, we call y a *descendant* of x and we call x an *ancestor* of y. If x is a maximal node satisfying $x < y$, then we call y a *successor* (or *child*) of x and we call x the *parent* of y. A *root* of a forest is a node without parent (a forest can have many roots). A *leaf* is a node without successors. Two nodes are *siblings* if they are both roots or if they have the same parent.

A *tree* is a forest with exactly one root. Since we define a tree as a special kind of a forest, the root of the tree is 0, which is different from the usual definition of a tree in which a root is ϵ (an empty word). For instance the full binary tree, in which every node has exactly two

successors, has domain of $0\{0,1\}^*$. If t is a forest and x is a node, we write $t|_x$ for the *subtree* of t rooted in x, defined as $t|_x(0y) = t(xy)$.

Since we are interested in algebraic frameworks for forests, we need a set of operations which will allow us to build forests from basic elements (single letters). The first operation is *forest concatenation* which allows us to compose forests horizontally. We denote this operation by $+$. If s and t are two forests then $s + t$ is a forest that has as many roots as s and t combined. This operation is non-commutative. The empty forest is the neutral element of this operation.

If t is a forest and $a \in A$, we write at for the tree with a root which is labeled with a and has the roots of t as its successors, i.e. $at(0) = a$, $at(0y) = t(y)$. Figure 2.2 shows two forests $s = a(ba0 + e0)$, $t = c0 + d(e0 + b0 + b0)$, their concatenation $s + t$ and the tree at. (Normally, when we write such expressions like $a(ba0 + e0)$, we omit the zeros.)

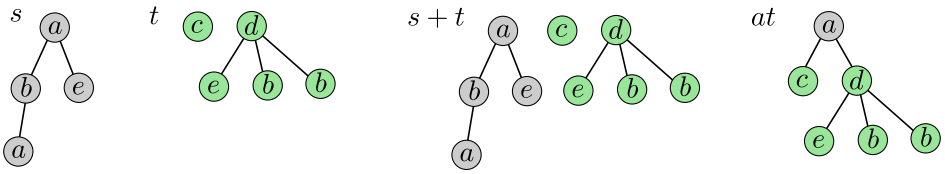


Figure 2.2

We also need a way to compose forests vertically. For this we introduce a new object: a forest with a special marker (called the *hole*) which denotes where we can append another forest. The new object is called a *context*. Formally, a context over an alphabet A is a forest over the alphabet $A \cup \{\Box\}$ where the label $\Box$, called the hole, occurs exactly once and in a leaf. The set of all contexts over A is denoted by A^{Con}. We use letters p, q to denote contexts. The *empty context* is a context with a single node, which is a hole, and is simply denoted by $\Box$.

We can *compose* a context p with a forest t, the result is a forest pt obtained by replacing the hole of p with the forest t (see figure 2.3). At first this operation may look somewhat peculiar, note that the number of successors of the parent of the hole increases by the number of roots in t minus one.

The basic elements which we use to build forests are *single-letter contexts*. For $a \in A$ we denote by $a\Box$ the context with a in the root and a hole below. We can treat the operation at as a composition of the single-letter context $a\Box$ with the forest t.

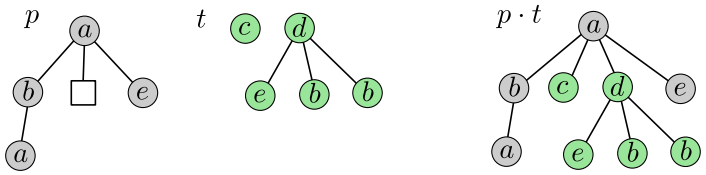


Figure 2.3

We can also compose a context p with another context q, the resulting context pq satisfies $(pq)t = p(qt)$ for all forests t. Finally, we can concatenate a forest t with a context p, which results in a context $t + p$. There is also the symmetric concatenation $p + t$.

In other words:

(a) we can concatenate two objects, as long as the result has at most one hole;

(b) we can compose two objects, as long as the first one has a hole.

A forest or context is *finite* if it has a finite number of nodes. The set of all finite forests over A is denoted by A^{FinFor}. Analogously for finite contexts we write A^{FinCon}. The above operations are sufficient to generate all finite forests. Since we are interested in infinite ones, we need more operations.

We say that the context is *guarded* if its hole is not in a root, e.g. $a(b+\Box)$ is guarded, but $ab+\Box$ is not. We denote by A^{Con_+} the set of all guarded contexts over an alphabet A, and $A^{\mathsf{Con}_\Box} = A^{\mathsf{Con}} - A^{\mathsf{Con}_+}$ is the set of all non-guarded contexts.

We can compose a guarded context p with itself infinitely many times, which results in an infinite forest t, which is the unique forest which satisfies the equality $pt = t$. We denote this operation by p^∞. On figure 2.4 we present the context $p = a(ba+\Box+e)$ and the forest p^∞.

It is easy to see why we disallow such *infinite composition* of a non-guarded context. What should the result for $(a+\Box)^\infty$ be? We could say that it is the forest $a+a+\cdots$ which is infinite to the right. But then, in the similar way, we could generate the forest $\cdots+a+a$ which is infinite to the left. What would happen after concatenating these two forests? We would have to suppose that the order on siblings is an arbitrary linear ordering.

This operation is a special case of a more general operation π which takes an infinite sequence of guarded contexts $p_1, p_2, \ldots$ and builds an infinite forest $\pi(p_1, p_2, p_3, \ldots)$ which is the unique forest which satisfies the equality $\pi(p_1, p_2, p_3, \ldots) = p_1 \cdot \pi(p_2, p_3, \ldots)$.

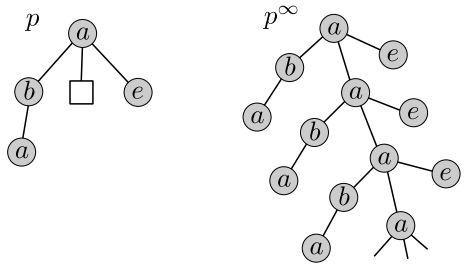


Figure 2.4

We define here two important types of forests, namely thin forests and regular forests. A forest is called *thin* if it has countably many paths. For example, every finite forest is thin, but the full binary tree is not. A context over A is called thin if, treated as a forest over $A \cup \{\Box\}$, it is thin. The set of all thin forests over A is denoted by A^{ThinFor}. Analogously for thin contexts we write $A^{\mathsf{ThinCon}} = A^{\mathsf{ThinCon}_+} \cup A^{\mathsf{ThinCon}_\Box}$.

A forest is called *regular* if it has finitely many distinct subtrees. For example the full binary tree is regular (since its every subtree is isomorphic to the whole tree), but any tree, in which the number of children is not bounded, is not regular. On figure 2.5 there are three trees (we assume that every node of these trees has the same label): t_1 is thin and regular, t_2 is thin, but not regular, since it has as a subtrees binary trees of every height, and t_3 (the full binary tree) is not thin, but regular.

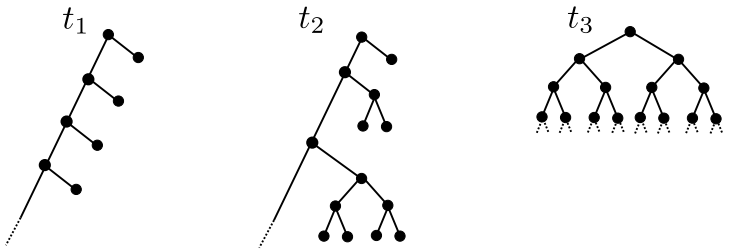


Figure 2.5

Of course, the irregularity of a forest can also be caused by labels. If we assign labels from the set $\{a, b\}$ to the tree t_3 in such a way, that a node on depth n is labeled by a if and only if n is prime, then clearly such tree is not regular.

We denote the set of regular forests over A by A^{regFor} and the set of regular thin forests by $A^{\mathsf{regThinFor}}$. Similarly for contexts.

Working at the same time with trees and forest can be quite tricky. However, we must do this to get the advantages of these two kind of objects. Forests has the advantage of allowing the concatenation, which is important from algebra's point of view. On the other had trees have exactly one root, which is convenient when we want to make a decomposition or perform an induction.

2.1.1 Thin forests

In this section we study thin forests. The following lemma gives a characterization of thin forests. We say that a forest t has a full binary tree as a minor if there is a set of nodes $R \subseteq \mathrm{dom}(t)$ such that for every node $x \in R$ there are at least two different nodes $x_0, x_1 \in R$ which are descendants of x, and x_i is not a descendant of x_{1-i} for $i = 0, 1$. We present the definitions of ordinal-labeling and branch-labeling below. Note that the conditions in the lemma do not depend on a labeling of a forest.

Lemma 1. *For a forest t the following conditions are equivalent:*

(a) t has countably many paths,

(b) t does not have a full binary tree as a minor,

(c) there exists an ordinal-labeling for the forest t,

(d) there exists a branch-labeling for the forest t.

Before we prove Lemma 1, we introduce the definitions that appear in its statement, namely ordinal-labeling and branch-labeling. An *ordinal-labeling* of a tree t is a function $\rho\colon \mathrm{dom}(t) \to \mathsf{Ord}$ which maps every node of t to an ordinal number such that for every node x and its successors $x_1, \ldots, x_n$:

(a) the numbers assigned to the successors of a node are not greater than the number assigned to the node, i.e. $\rho(x_i) \leq \rho(x)$;

(b) at most one successor has a number equal to $\rho(x)$.

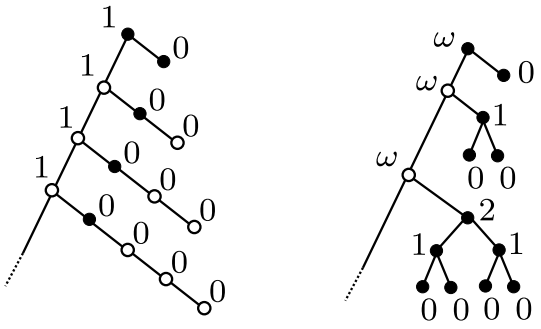


Figure 2.6

On figure 2.6 there are two trees with ordinal-labelings. The white nodes are the ones whose numbers are equal to the numbers assigned to their parents.

A *branch-labeling* of a tree t is a set $R \subseteq \mathrm{dom}(t)$ such that:

(a) for every node at most one of its successors belongs to R;

(b) on every infinite path almost all nodes belong to R.

On figure 2.6 the white nodes belong to the branch-labelings.

Proof of Lemma 1. We begin with the implication "$(a) \Rightarrow (b)$". Let t be a forest. If t has countably many paths, then it obviously cannot have the full binary tree as a minor, since the full binary tree has uncountably many paths.

For the implication "$(b) \Rightarrow (c)$" consider the set $X \subseteq \mathrm{dom}(t)$ of nodes such that for every $x \in X$ there exists an ordinal-labeling of the subtree $t|_x$. We will show that $X = \mathrm{dom}(t)$.

Suppose that there exists a node $x \notin X$ such that all successors $x_1, \ldots, x_n$ of x belong to X. Then for every subtree $t|_{x_i}$ we have an ordinal-labeling ρ_i. We construct an ordinal-labeling for the tree $t' = t|_x$ by labeling nodes from $t'|_i$ according to ρ_i and by labeling the root of t' by ordinal $\max\{\rho_1(0), \ldots, \rho_n(0)\} + 1$, i.e. an ordinal greater than labeling of the root of $t|_{x_i}$ for $i = 1, \ldots, n$. Thus such x cannot exist, therefore every $x \notin X$ has at least one successor $x' \notin X$. That means that from every $x \notin X$ there is an infinite path of nodes which does not belong to X.

Suppose that there exists a node $x \notin X$ and an infinite path π from x which contains all the nodes from $t|_x$ which do not belong to X. Let Y be the set of nodes which are successors of nodes from π, but do not belong to π. For every $y \in Y$ we have an ordinal-labeling ρ_y defined on $t|_y$. Let α be an ordinal greater than the label of the root of $t|_y$ for every $y \in Y$. Again, we construct ordinal-labeling for $t' = t|_x$ by using appropriate labelings ρ_y for nodes which do not belong to the path π, and label α for every node on the path π. Thus such x cannot exist, therefore for every $x \notin X$ there exist two incomparable descendants $x', x'' \notin X$.

Therefore if $\mathrm{dom}(t) - X$ is non-empty, then t contains the full binary tree as a minor. Thus $X = \mathrm{dom}(t)$, in particular the root of t belongs to X, thus there exists an ordinal-labeling of t.

For the implication "$(c) \Rightarrow (d)$" assume that ρ is an ordinal-labeling of the forest t. Let $R \subseteq \mathrm{dom}(t)$ be the set of nodes x which satisfy $\rho(x) = \rho(y)$ where y is the parent of x. We show that R is a branch-labeling of the forest t. From the definition of ρ, for every node x there is at most one successor x' such that $\rho(x) = \rho(x')$, thus from the set of x's

successors only x' can belong to R. Suppose that the second condition defining R is not satisfied, i.e. there is an infinite path $\pi = x_1x_2x_3\ldots$ and an infinite increasing sequence $\{k_i\}_{i\geq 1}$ such that $x_{k_i} \notin R$ for every $i \geq 1$. Therefore $\rho(x_{k_i-1}) > \rho(x_{k_i})$, thus we obtained an infinite decreasing sequence of ordinal numbers

$$\rho(x_{k_1}) > \rho(x_{k_2}) > \rho(x_{k_3}) > \ldots$$

which is impossible.

Finally, for the implication "$(d) \Rightarrow (a)$" let R be a branch-labeling of a forest t. Consider a mapping f which assigns for every maximal infinite path $\pi \subseteq \mathrm{dom}(t)$ the first node $x \in \pi$ such that $x \in R$ and every descendant of x from the path π also belongs to R. From the second condition the mapping f is defined for all maximal infinite paths. We prove that the mapping f is injective. Suppose that there exist two maximal infinite paths π_1,π_2 such that $x_0 = f(\pi_1) = f(\pi_2)$. Of course $x_0 \in \pi_1 \cap \pi_2 \cap R$. Assume that $\pi_1 \neq \pi_2$ and let x_1 be the maximal node which belongs to $\pi_1 \cap \pi_2$. Since $x_0 \leq x_1$, then $x_1 \in R$ and has two successors $x' \in \pi_1$ and $x'' \in \pi_2$. From the definition of x_0 these successors must both belong to R, which is not possible. Thus $\pi_1 = \pi_2$, which proves that f is injective.

From this and the fact that $\mathrm{dom}(t)$ is countable we get that the number of infinite paths in t is countable. Since the number of finite paths is of course countable, then the number of all paths in t is countable. □

The *rank* of a thin tree is the minimal ordinal number assigned to the root of t by ordinal-labelings of t, i.e.

$$\mathrm{rank}(t) = \min\{\rho^t(0) \mid \rho \text{ is a rank-labeling}\}.$$

This is well-defined due to well-foundedness of ordinals. The rank of a node x of a thin tree is the rank of a subtree rooted in x.

Lemma 2. *Let t be a thin tree. The mapping* $\mathrm{rank}\colon \mathrm{dom}(t) \to \mathsf{Ord}$ *is an ordinal-labeling.*

Proof. Consider a node x and its successors $x_1, \ldots, x_n$. Assume that the condition (a) from the definition of ordinal-labeling is violated, i.e. there exists x_i such that $\mathrm{rank}(x) < \mathrm{rank}(x_i)$. Then there exists an ordinal-labeling ρ of $t|_x$ such that it assigns $\mathrm{rank}(x)$ to the root of $t|_x$. Then $\rho(0i) \leq \rho(0) = \mathrm{rank}(x) < \mathrm{rank}(x_i)$ which leads to a contradiction.

Assume then that the condition (b) is violated, i.e. there exists x_i, x_j such that $\mathrm{rank}(x) = \mathrm{rank}(x_i) = \mathrm{rank}(x_j)$. Then there exists an ordinal-labeling ρ of $t|_x$ such that $\rho(0) = \mathrm{rank}(x)$ and without loss of generality $\rho(0) < \mathrm{rank}(x_i)$ which also leads to a contradiction. □

Let t be a thin tree. It is clear that the rank of every leaf of t is equal to 0. Consider a set of nodes of maximum rank in t (which is of course the rank of t). It is easy to see that this set is a (possibly infinite) path from the root. Indeed, from condition (a) in the definition of ordinal-labeling, every parent of a node of maximum rank has the maximum rank, and from the condition (b) there is no node which has two successors of the maximum rank. We call this path the *spine* of a tree. On figure 2.7 there are examples of an infinite and a finite spine.

We will use ranks in inductive proofs. Suppose we want to prove a property for a thin tree t. We will show that from the assumption that the property holds for all trees with ranks less that $\mathrm{rank}(t)$ (thus for all subtrees which do not intersect the spine of t) we get that the property holds for t. Then using transfinite induction we conclude that the property holds for all thin trees.

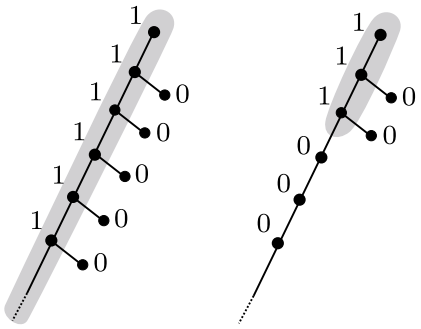


Figure 2.7

Lemma 3. *The rank of a regular thin tree is a natural number.*

Proof. A regular tree t has a finite number of different subtrees, therefore the set of ranks of its nodes

$$S = \{\text{rank}(x) \mid x \in \text{dom}(t)\}$$

is finite. Let $\alpha_0 < \alpha_1 < \ldots < \alpha_n$ be all the ordinals from S. It is easy to see that if we replace α_i by i, we get a valid ordinal-labeling, thus $\text{rank}(t) \leq n$. □

Of course we cannot invert the statement of Lemma 3. For instance on figure 2.6 we have an irregular tree with rank equal to 1.

2.1.2 Components in a forest

Let t be a forest. We say that two nodes x, y of the forest are *in the same component* if the subtree $t|_x$ is a subtree of the subtree $t|_y$ and vice versa.

To a forest we associate a directed graph $G_t = (V_t, E_t)$ (we call it the *component graph* of the forest) in which the set of nodes V_t contains all non-isomorphic subtrees of t and there is an edge $(t_1, t_2) \in E_t$ if the subtree t_2 is an immediate subtree of the subtree t_1 (i.e. $t_2 = t_1|_x$ for some child x of the root of t_1). The graph G_t is finite if and only if the forest t is regular. Every component in t corresponds to a strongly connected component in G_t.

There are two kinds of components: *singleton components*, which correspond to strongly connected components in G_t of exactly one node and no edges, and *connected components*, which correspond to other strongly connected components in G_t. Note that a node x in the forest is in singleton component if and only if $t|_x$ is not a proper subtree of $t|_x$.

On figure 2.8 there is a tree t and the corresponding graph G_t. The tree has five components: two connected (which correspond to strongly connected components c_1, c_2 in G_t) and three singleton (which correspond to s_1, s_2, s_3). Note that the component which corresponds to a strongly connected component c_1 (of one node but with a loop edge) is in fact connected. Note that the graph loses some information, so it is not possible to fully reconstruct the forest t from G_t. However, it is only matter of adding the order and multiplicity to edges of G_t.

In a regular tree the notion of a component is very useful, since the number of components in a tree is a good inductive parameter. Note that each regular forest has finitely many components, but the converse is not necessarily true. Indeed, if we define a family

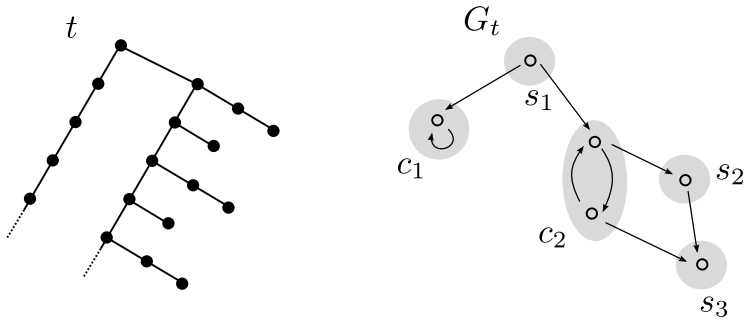


Figure 2.8

of trees t_n for $n \geq 1$ as

$$t_n = \underbrace{a \cdots a}_{n \text{ times}}(t_1 + a\Box)(t_2 + a\Box) \cdots (t_n + a\Box)(t_{n+1} + a\Box) \cdots ,$$

then the forest t_n is not regular, but it has only one component (which contains infinite number of nodes).

We present here two facts about components in a regular thin forest.

Lemma 4. *In a regular thin forest t every connected component corresponds to a strongly connected component in G_t which is a simple cycle, i.e. the graph induced by the nodes of this component is a simple cycle.*

Proof. Let c be the strongly connected component in G_t which corresponds to a connected component in t. Let G' be the graph induced by the nodes of c.

We first show that the out-degree of every node in G' is at most 1. Let assume otherwise – then there is a node u with at least two outgoing edges $u \to v_1$, $u \to v_2$. By adding a path from v_1 and v_2 back to u, we get a full binary tree that is a minor of t, thus from Lemma 1 the forest is not thin.

Similarly, we show that the in-degree of every node in G' is at most 1. Since c does not contain any isolated nodes, the out-degree and in-degree of any node is in fact exactly 1. Since c is connected, it is indeed a simple cycle. □

The rank of a regular thin tree t is a natural number (see Lemma 3). We show that in fact this rank is equal to the maximal height of a finite full binary tree which is a minor of t. Since it is easy to see that all the nodes from the same component of a tree have the same rank, we can refer to it as the *rank of a component*. The following lemma gives us a constructive definition of the rank of a regular thin tree.

Lemma 5. *Let t be a regular thin tree. Let u be a component in t and $v_1, v_2, \ldots, v_k$ be its child-components (i.e. there are edges from u to $v_1, \ldots, v_k$ in G_t) sorted by their ranks, that is* $\mathrm{rank}(v_1) \geq \mathrm{rank}(v_2) \geq \ldots \geq \mathrm{rank}(v_k)$. *Then the following hold:*

(a) If u is a singleton component then

$$\mathrm{rank}(u) = \begin{cases} 0 & \text{for } k = 0, \\ \mathrm{rank}(v_1) & \text{for } k = 1, \\ \max(\mathrm{rank}(v_1), 1 + \mathrm{rank}(v_2)) & \text{for } k \geq 2. \end{cases}$$

(b) If u is a connected component then

$$\operatorname{rank}(u) = \begin{cases} 0 & \text{for } k = 0, \\ 1 + \operatorname{rank}(v_1) & \text{for } k \geq 1. \end{cases}$$

(c) The components with maximum rank form a single path from the root in the component-tree. Moreover, only the lowest component on this path can be connected.

(d) The components which are leaves in the component-tree have the rank of 0.

Proof. For proving (a) and (b) we inductively define the ranks for components, starting from the bottom ones.

For proving (c): we know that $\operatorname{rank}(u) \geq \operatorname{rank}(v_1)$. Thus if a component has maximum rank then all its ancestor-components also have the maximum rank. Let x_1 and x_2 be two components not related in descendant relation, let y be a component which is a least common ancestor of x_1 and x_2 and let x_1' and x_2' be two children of y which lie on the paths from y to x_1 and x_2 respectively. Since ranks of x_1, x_2, x_1' and x_2' are maximal and from (a) and (b) $\operatorname{rank}(y) \geq 1 + \operatorname{rank}(x_1')$, we get that $\operatorname{rank}(y)$ is greater that maximal.

If y is a parent of x and both are on the path, and y is connected, then from (b) $\operatorname{rank}(y) = 1 + \operatorname{rank}(x)$, which is greater that maximal.

In case of (d) such component has no children, so we put $k = 0$ in (a) and (b). □

2.2 Logic and automata over infinite forests

The theory of logic and automata over infinite trees discusses mostly binary (thus ranked) trees (see [19]). On the other hand, we are mainly interested in unranked trees, since they are more suited for the algebraic approach. In this section we introduce MSO logic and a model of automaton over unranked infinite forests, and we show how they are related to their widely known ranked counterparts.

We do this by using a well-known technique of encoding unranked forests by full binary trees. The encoding we use is called *first-child-next-sibling* encoding, and the idea is that a node stores, rather than a list of its children, a pointer to its first child and a pointer to its immediate right sibling.

Let # be a special label outside of the alphabet A and $t_\#$ be a full binary tree labeled only with #. We will encode a forest over A with a full binary tree over $A_\# = A \cup \{\#\}$ in the following manner:

(a) the empty forest is encoded as $\mathsf{fcns}(0) = t_\#$;

(b) a sequence of trees $t_1, \ldots, t_n$ where $t_1 = as$ is encoded as

$$\mathsf{fcns}(t_1 + \ldots + t_n) = a(\mathsf{fcns}(s), \mathsf{fcns}(t_2 + \ldots + t_n)).$$

It is easy to see that this encoding is injective. We extend the definition to languages by putting $\mathsf{fcns}(L) = \{\mathsf{fcns}(t) \mid t \in L\}$.

On the figure 2.9 there is a tree t and its first-child-next-sibling encoding $\mathsf{fcns}(t)$. The presented trees are finite, but the definition also works for infinite trees. In this case on every path from $\mathsf{fcns}(t)$ which from some point turns only right, we eventually see the label #.

We will prove the following theorem (the definitions of an MSO formula and a forest automaton are delegated to the next two sections):

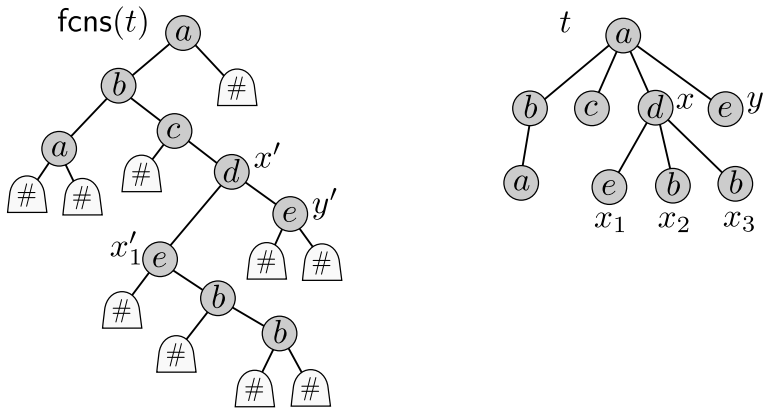


Figure 2.9

Theorem 6. *For every language of forests L the following conditions are equivalent:*

(a) L is recognized by a forest automaton;

(b) L is definable by an MSO formula;

(c) the first-child-next-sibling encoding of L is definable by an MSO formula;

(d) the first-child-next-sibling encoding of L is recognized by a full binary tree automaton.

Languages which satisfies conditions from Theorem 6 are called *regular*. Note that this definition has nothing to do with the definition of a regular tree.

2.2.1 Monadic second-order logic

With a full binary tree t over an alphabet A we associate a relational structure of the form (see [40])

$$\mathsf{struct}(t) = \big(\mathrm{dom}(t) = 0\{0,1\}^*, \mathsf{succ}_0^t, \mathsf{succ}_1^t, \{\mathsf{lab}_a^t\}_{a\in A}\big).$$

Here $\mathsf{succ}_0^t, \mathsf{succ}_1^t$ denote respectively left and right successor relations over $\mathrm{dom}(t)$, i.e. $(x, x0) \in \mathsf{succ}_0^t$ and $(x, x1) \in \mathsf{succ}_1^t$ for every node $x \in \mathrm{dom}(t)$. Additionally, for every $a \in A$ we have a labeling relation $\mathsf{lab}_a^t = \{x \in \mathrm{dom}(t) \mid t(x) = a\}$.

With a forest t over the alphabet A we associate a relational structure

$$\mathsf{struct}(t) = \big(\mathrm{dom}(t), \mathsf{succ}^t, \mathsf{sibl}^t, \{\mathsf{lab}_a^t\}_{a\in A}\big).$$

Here succ^t is the successor relation over $\mathrm{dom}(t)$, i.e. $\mathsf{succ}^t = \{(x, xi) \mid x, xi \in \mathrm{dom}(t), i \geq 0\}$, and sibl^t is the right-sibling relation over $\mathrm{dom}(t)$, i.e. $\mathsf{sibl}^t = \{(xi, xj) \mid x, xi, xj \in \mathrm{dom}(t), i \geq 0, j = i+1\}$. The labeling relation is defined as above.

On figure 2.9 we have for instance $(x, x_i) \in \mathsf{succ}^t$ for $i = 1, 2, 3$, $(x, y) \in \mathsf{sibl}^t$ and $(x', x'_1) \in \mathsf{succ}_0^{\mathsf{fcns}(t)}$, $(x', y') \in \mathsf{succ}_1^{\mathsf{fcns}(t)}$.

Let $V_{\mathrm{sing}}, V_{\mathrm{set}}$ be sets of variables (respectively of singleton variables and set variables). The syntax of *monadic second-order logic* is as follows: if $a \in A$, $x, y \in V_{\mathrm{sing}}$, $X \in V_{\mathrm{set}}$ and φ, ψ are *MSO formulas*, then the following are also MSO formulas:

$$a(x),\ x = y,\ x \neq y,\ X(x),\ \mathsf{succ}_0(x,y),\ \mathsf{succ}_1(x,y),\ \mathsf{succ}(x,y),\ \mathsf{sibl}(x,y),\ x \leq y,\ x < y,$$
$$\neg\varphi,\quad \varphi \vee \psi,\quad \varphi \wedge \psi,\quad \varphi \to \psi,\quad \varphi \leftrightarrow \psi,\quad \exists x\ \varphi,\quad \forall x\ \varphi,\quad \exists X\ \varphi,\quad \forall X\ \varphi$$

Let $\eta_{\text{sing}} \colon V_{\text{sing}} \to \text{dom}(t)$ be a mapping from the set of singleton variables to the positions in the structure and $\eta_{\text{set}} \colon V_{\text{set}} \to 2^{\text{dom}(t)}$ be a mapping from the set of set variables to the sets of positions in the structure. We denote that the formula φ is satisfied in the structure $\mathsf{struct}(t)$ regarding to the valuation $\eta = (\eta_{\text{sing}}, \eta_{\text{set}})$ by writing $\mathsf{struct}(t), \eta \models \varphi$. The semantics of MSO formulas is defined as follows:

$\mathsf{struct}(t), \eta \models a(x)$	if $\eta_{\text{sing}}(x)$ is defined and $\eta_{\text{sing}}(x) \in \mathsf{lab}_a^t$
$\mathsf{struct}(t), \eta \models x = y$	if $\eta_{\text{sing}}(x)$, $\eta_{\text{sing}}(y)$ are defined and $\eta_{\text{sing}}(x) = \eta_{\text{sing}}(y)$
$\mathsf{struct}(t), \eta \models X(x)$	if $\eta_{\text{sing}}(x)$, $\eta_{\text{set}}(X)$ are defined and $\eta_{\text{sing}}(x) \in \eta_{\text{set}}(X)$
$\mathsf{struct}(t), \eta \models \mathsf{succ}(x, y)$	if $\eta_{\text{sing}}(x)$, $\eta_{\text{sing}}(y)$ defined and $(\eta_{\text{sing}}(x), \eta_{\text{sing}}(y)) \in \mathsf{succ}^t$ (similarly for succ_0, succ_1 and sibl)
$\mathsf{struct}(t), \eta \models \neg\varphi$	if $\mathsf{struct}(t), \eta \not\models \varphi$
$\mathsf{struct}(t), \eta \models \varphi \vee \psi$	if $\mathsf{struct}(t), \eta \models \varphi$ or $\mathsf{struct}(t), \eta \models \psi$
$\mathsf{struct}(t), \eta \models \exists x\ \varphi$	if exists $x' \in \text{dom}(t)$ such that $\mathsf{struct}(t), \eta' \models \varphi$ where $\eta'_{\text{sing}}(y) = \begin{cases} \eta_{\text{sing}}(y) & \text{for } y \neq x \\ x' & \text{for } y = x \end{cases}$ and $\eta'_{\text{set}} = \eta_{\text{set}}$
$\mathsf{struct}(t), \eta \models \exists X\ \varphi$	if exists $X' \subseteq \text{dom}(t)$ such that $\mathsf{struct}(t), \eta' \models \varphi$ $\eta'_{\text{sing}} = \eta_{\text{sing}}$ and $\eta'_{\text{set}}(Y) = \begin{cases} \eta_{\text{set}}(Y) & \text{for } Y \neq X \\ X' & \text{for } Y = X \end{cases}$

We also use the following syntactic sugar:

$$\begin{aligned} \varphi \wedge \psi &\equiv \neg(\neg\varphi \vee \neg\psi) & \forall x\ \varphi &\equiv \neg\exists x\ \neg\varphi \\ \varphi \to \psi &\equiv \neg\varphi \vee \psi & \forall X\ \varphi &\equiv \neg\exists X\ \neg\varphi \\ \varphi \leftrightarrow \psi &\equiv (\varphi \to \psi) \wedge (\psi \to \varphi) & x \neq y &\equiv \neg(x = y) \end{aligned}$$

Moreover, we use the descendant relation, that is the transition closure of the successor relation:

$$\begin{aligned} x \leq y &\equiv \forall X\ X(x) \wedge (\forall x' \forall y'\ X(x') \wedge \mathsf{succ}(x', y') \to X(y')) \to X(y) \\ x < y &\equiv x \leq y \wedge x \neq y \end{aligned}$$

Similarly, we denote the transition closure of succ_0 and succ_1 by $\leq_0$ and $\leq_1$, respectively.

We also use the following formulas which check whether a node x is a root of the forest and whether nodes from a set X form a maximal infinite path:

$$\begin{aligned} \text{root}(x) &\equiv \neg\exists y\ \mathsf{succ}(y, x) \\ \text{path}(X) &\equiv (\exists x\ \text{root}(x) \wedge X(x)) \wedge \big(\forall x\ X(x) \to \exists y\ \mathsf{succ}(x, y) \wedge X(y) \\ &\qquad \wedge (\forall z\ \mathsf{succ}(x, z) \wedge X(z) \to z = y)\big) \end{aligned}$$

The language of full binary trees or forests defined by an MSO formula φ is the set of all trees (forests) whose structures satisfy φ with the initially empty valuation:

$$L(\varphi) = \{t \mid \mathsf{struct}(t), \emptyset \models \varphi\}.$$

We say that such a language is *MSO-definable* or *regular*.

Lemma 7. *For every language L of forests defined by an MSO formula, there is an MSO formula which defines* $\mathsf{fcns}(L)$.

Proof. We take an MSO formula φ which defines L and rewrite it with the following rules to obtain a formula φ'. The rewrite rules ensure that φ' quantifies only over nodes in $\mathsf{fcns}(t)$ which have their corresponding nodes in t and that predicates are correctly translated:

$$\begin{aligned} \exists x\ \varphi &\quad\to\quad \exists x\ \neg\#(x) \wedge \varphi \\ \exists X\ \varphi &\quad\to\quad \exists X\ (\forall x\ X(x) \to \neg\#(x)) \wedge \varphi \\ \mathsf{succ}(x,y) &\quad\to\quad \exists z\ \mathsf{succ}_0(x,z) \wedge z \leq_1 y \\ \mathsf{sibl}(x,y) &\quad\to\quad \mathsf{succ}_1(x,y) \end{aligned}$$

We also have to check that the nodes labeled with special letter # form full binary trees. Therefore the formula which defines $\mathsf{fcns}(L)$ is

$$\varphi' \quad\wedge\quad \forall x \forall y\ \#(x) \wedge (\mathsf{succ}_0(x,y) \vee \mathsf{succ}_1(x,y)) \to \#(y). \qquad \square$$

2.2.2 Automata

A (non-deterministic parity) *full binary tree automaton* over an alphabet A is given by a set of states Q, a transition relation $\Delta \subseteq Q \times Q \times A \times Q$, an initial state $q_I \in Q$, and a parity condition $\Omega\colon Q \to \{0, \ldots, k\}$.

A *run* of this automaton over a full binary tree t is a labeling $\rho\colon \mathrm{dom}(t) \to Q$ of tree nodes with states such that for every node x with two children x_0, x_1,

$$(\rho(x_0), \rho(x_1), t(x), \rho(x)) \in \Delta.$$

For every (infinite) path $\pi \subseteq \mathrm{dom}(t)$ we consider the set $\mathrm{Inf}(\pi)$ of states which appear infinitely many often on this path, i.e.

$$\mathrm{Inf}(\pi) = \{q \in Q \mid \text{there are infinite number of nodes } x \in \pi \text{ with } \rho(x) = q\}.$$

A run is *accepting* if for every (infinite) path $\pi \subseteq \mathrm{dom}(t)$, the maximum rank among states from $\mathrm{Inf}(\pi)$, i.e. $\max_{q \in \mathrm{Inf}(\pi)} \Omega(q)$, is even. The *value* of a run over a tree t is a state assigned to the root of the tree. A tree is *accepted* by an automaton if it has an accepting run whose value is the initial state q_I. The set of trees accepted by an automaton is called the language *recognized* by the automaton.

The well-known theorem of Rabin gives a correspondence between languages of full binary trees which are definable by an MSO formula and languages of full binary trees which are recognized by full binary tree automata:

Theorem 8 (Rabin Tree Theorem, [29])**.** *A language of full binary trees is MSO-definable if and only if it is recognizable by a full binary tree automaton.*

A (non-deterministic parity) *forest automaton* over an alphabet A is given by a set of states Q equipped with a monoid structure, a transition relation $\Delta \subseteq Q \times A \times Q$, an initial state $q_I \in Q$ and a parity condition $\Omega\colon Q \to \{0, \ldots, k\}$. We use additive notation $+$ for the monoid operation in Q, and we write 0 for the neutral element.

A *run* of this automaton over a forest t is a labeling $\rho\colon \mathrm{dom}(t) \to Q$ of forest nodes with states such that for every node x with children $x_1, \ldots, x_n$

$$(\rho(x_1) + \rho(x_2) + \cdots + \rho(x_n), t(x), \rho(x)) \in \Delta.$$

Note that if x is a leaf, then the above implies $(0, t(x), \rho(x)) \in \Delta$.

A run is *accepting* if for every (infinite) path $\pi \subseteq \mathrm{dom}(t)$, the maximum rank among states from $\mathrm{Inf}(\pi)$ is even. The *value* of a run over a forest t is obtained by adding, using $+$, all the states assigned to roots of the forest. A forest is *accepted* if it has an accepting run whose value is the initial state q_I of the automaton. The set of forests accepted by an automaton is called the language *recognized* by the automaton.

Example 9. Consider a forest language

$$L = \text{“there is at least one infinite path with all nodes labeled with } a\text{”}.$$

Note that the a-labeled path does not necessarily be maximal (i.e. it may begin in any node). This language is recognized by a forest automaton $\mathcal{A}_1$ with states $Q = \{q_A, q_a, q_b\}$. The idea is that a forest t has an accepting run of value q_a if there is an a-labeled path in t, and it has an accepting run of value q_A if there is an a-labeled path in t which begins in a root. Thus the initial state is q_a.

The state q_b is a monoid identity, and the monoid is given by equations:

$$\begin{aligned} q_A + q = q + q_A = q_A \quad &\text{for all } q \in Q \\ q_a + q_a = q_a + q_b = q_b + q_a = q_a \\ q_b + q_b = q_b \end{aligned}$$

A transition relation is as follows:

$$\Delta = \{(q_A, a, q_A),\ (q_A, a/b, q_a),\ (q_a, a/b, q_a),\ (q_b, a/b, q_b)\},$$

The parity condition is given by $\Omega(q_A) = \Omega(q_b) = 0$ and $\Omega(q_a) = 1$. Note that every forest has a run of value q_b, but a forest t has a run of value q_a only if it belongs to L. In the later case there must be an infinite a-labeled path in t such that every node of the path is assigned state q_A.

Lemma 10. *If L is a forest language such that $\mathsf{fcns}(L)$ is recognized by a full binary tree automaton, then L is recognized by a forest automaton.*

Proof. Let $\mathcal{A} = (A_\#, Q, \Delta, q_I, \Omega)$ be a full binary tree automaton which recognizes $\mathsf{fcns}(L)$. We can assume that we have a special state $q_\# \in Q$ and there is only one transition which involves special label $\#$, namely $(q_\#, q_\#, \#, q_\#) \in \Delta$. First, we construct an auxiliary automaton $\mathcal{A}'$ which also recognizes $\mathsf{fcns}(L)$ but additionally it guesses for every node $x \in \mathrm{dom}(t)$ the maximum rank which appears on the maximal suffix of the path from root to x which descends only to the right.

We define $\mathcal{A}' = (A_\#, Q', \Delta', q'_I, \Omega')$ as follows:

(a) each state remembers an additional rank, thus $Q' = Q \times \{0, \ldots, k\}$,

(b) $q'_I = (q_I, \Omega(q_I))$,

(c) for every $a \in A$, $i \in \{0, \ldots, k\}$ and every tuple $(q_1, q_2, a, q) \in \Delta$ we add a transition which initializes the rank of a left-child node and ensures that the rank of a right-child node takes into account the rank of its parent:

$$\big((q_1, \Omega(q_1)),\ (q_2, \max(i, \Omega(q_2))),\ a,\ (q, i)\big) \in \Delta',$$

we also add transitions for special label $\#$: $\big((q_\#, i), (q_\#, i), \#, (q_\#, i)\big) \in \Delta'$,

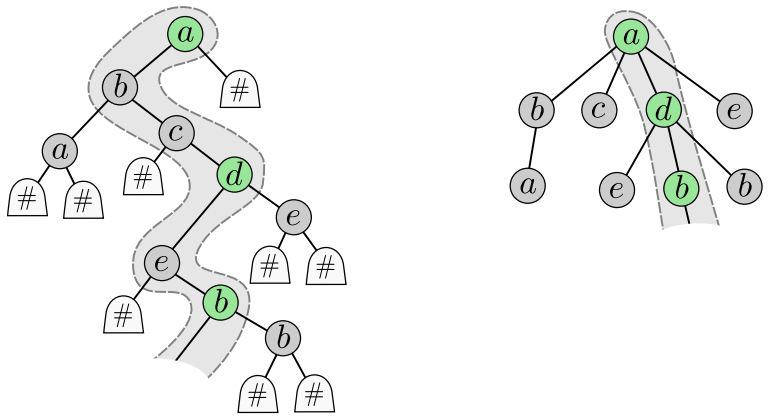


Figure 2.10

(d) parity condition uses the additional rank:

$$\Omega'((q,i)) = i \quad \text{for every } q \in Q',\ i \in \{0,\dots,k\}.$$

We show that the languages recognized by automata $\mathcal{A}$ and $\mathcal{A}'$ are the same. There is a one-to-one correspondence between the runs of these two automata which have accepting state in the root, since the additional ranks are uniquely determined (see also figure 2.10). Consider a tree t, a run ρ of $\mathcal{A}$ over t and the corresponding run ρ' of $\mathcal{A}'$ over t. Consider a path $\pi \subseteq \mathrm{dom}(t)$ and partition it as $\pi = \pi_1\pi_2\pi_3\dots$, where every π_i is a non-empty sequence of nodes such that the first node in π_i is the root or a left-child node and every other node is a right-child node. From the conditions for every $\pi_i = x_0x_1\dots x_n$ we get $\Omega'(\rho'(x_i)) = \max\{\Omega(\rho(x_j)) \mid j \leq i\}$, thus $\max_i \Omega'(\rho'(x_i)) = \max_i \Omega(\rho(x_i))$, therefore the run ρ is accepting if and only if the run ρ' is accepting.

The first coordinate of state of $\mathcal{A}'$ is the exact state of $\mathcal{A}$, therefore if we put $\Omega'((q,i)) = \Omega(q)$ the acceptance would be the same. The value of the second coordinate is the maximum rank which appears on the portion of ancestor-path which goes through right-children edges. Since these paths are always finite, changing $\Omega'((q,i)) = i$ will not affect the acceptance.

Now, using $\mathcal{A}'$, we construct a forest automaton $\mathcal{B}$ which recognizes L. The automaton tries to guess the accepting run of the automaton $\mathcal{A}'$. It labels a node x of a tree t with a pair of states $(q,q') \in Q' \times Q'$ which means: I guess that the state of a node x' in $\mathsf{fcns}(t)$ which corresponds to node x is q, and that the state of the right-child of x' is q'.

We define $\mathcal{B} = (A, Q'', \Delta'', q_I'', \Omega'')$ as follows:

(a) each label is a guess of two states or an error state, thus $Q'' = Q' \times Q' \cup q_\perp$,

(b) $q_I'' = (q_I', q_\#)$,

(c) the monoid structure on states ensures that the guess of the right-child's state is consistent with the right-child's guess of its state:

$$(q_1,q_2) + (q_2,q_3) = (q_1,q_3) \quad \text{for every } q_1,q_2,q_3 \in Q'',$$
$$(\cdot,q_1) + (q_2,\cdot) = q_\perp \quad \text{for every } q_1 \neq q_2,$$

(d) for every $a \in A$ and every tuple $(q_1, q_2, a, q) \in \Delta'$ we add a transition which ensures that the guess in the parent is consistent with guesses in its children:

$$\big((q_1, q_\#),\ a,\ (q, q_2)\big) \in \Delta'',$$

(e) the parity condition uses guessed state:

$$\Omega''((q, q')) = \Omega'(q) \quad \text{for every } (q, q') \in Q''.$$

Again, there is a one-to-one correspondence between the runs of automata $\mathcal{A}'$ and $\mathcal{B}$ which have accepting state in the root. Consider a tree t, a run ρ of $\mathcal{B}$ over t and the corresponding run ρ' of $\mathcal{A}'$ over $\mathsf{fcns}(t)$. Consider an infinite path $\pi \subseteq \mathrm{dom}(t)$ and the corresponding path $\pi' \subseteq \mathrm{dom}(\mathsf{fcns}(t))$. Fix a node $x \in \mathrm{dom}(t)$ and its corresponding node $x' \in \mathrm{dom}(\mathsf{fcns}(t))$. If $\pi = x_1 x_2 x_3 \dots$ then $\pi' = \pi_1 x'_1 \pi_2 x'_2 \pi_3 x'_3 \dots$ where $\pi_i x'_i$ is a maximal suffix of the path from root to x'_i which descends only to the right. But from the construction of $\mathcal{A}'$ and $\mathcal{B}$ we have that

$$\max_{y \in \pi_i x'_i} \Omega'(\rho'(y)) = \Omega'(\rho'(x'_i)) = \Omega''(\rho(x)),$$

therefore the maximum rank which appears infinitely often on the paths π and π' is the same. Therefore ρ satisfies the parity condition of the automaton $\mathcal{B}$ if and only if ρ' satisfies the parity condition of the automaton $\mathcal{A}'$. □

2.2.3 Equivalence of recognizability by automata and definability by MSO formulas

In this section we establish the connection between languages recognizable by tree (forest) automata and languages definable by MSO formulas, namely that they are the same.

Lemma 11. *If L is a forest language recognized by a forest automaton $\mathcal{A}$, then there is an MSO formula φ which defines the language L.*

Proof. We will construct the formula φ explicitly: given a forest t, it will guess the run ρ of the automaton $\mathcal{A}$ over the forest and check the acceptance conditions. The formula guesses a family of sets $X_{q,r}$ for every $q, r \in Q$. A node x belongs to the set $X_{q,r}$ if it is labeled by a state q in the run ρ and if $\rho(x) + \rho(x_1) + \rho(x_2) + \cdots + \rho(x_k) = r$, where $x_1, \dots, x_k$ are the siblings of x which lie to the right of x. In particular if y is the leftmost successor of x and $y \in X_{q',r'}$, then r' is the sum of states assigned to the successors of x, thus $(r', t(x), q) \in \Delta$. The formula φ is as follows:

$$\begin{aligned}
&\exists X_{q,r} \exists X_{q',r'} \cdots \exists X_{q'',r''} \\
&\quad \forall x \ \Big(\textstyle\bigwedge_{(q,r)\neq(q',r')} X_{q,r}(x) \to \neg X_{q',r'}(x) \\
&\qquad \wedge (\mathrm{root}(x) \wedge \neg \exists y\ \mathsf{sibl}(y, x)) \to \textstyle\bigvee_q X_{q,q_I}(x) \\
&\qquad \wedge \forall y\ (\mathsf{succ}(x, y) \wedge \neg \exists z\ \mathsf{sibl}(z, y)) \to \textstyle\bigvee_{q',r} \bigvee_{(r',t(x),q)\in\Delta} X_{q,r}(x) \wedge X_{q',r'}(y) \\
&\qquad \wedge (\neg \exists y\ \mathsf{succ}(x, y)) \to \textstyle\bigvee_r \bigvee_{(0,t(x),q)\in\Delta} X_{q,r}(x) \\
&\qquad \wedge \forall y\ \mathsf{sibl}(x, y) \to \textstyle\bigvee_{q,q',r'} X_{q,q+r'}(x) \wedge X_{q',r'}(y) \\
&\qquad \wedge (\neg \exists y\ \mathsf{sibl}(x, y)) \to \textstyle\bigvee_q X_{q,q}(x) \Big) \\
&\quad \wedge \forall X\ (\mathrm{path}(X) \to \textstyle\bigvee_i \exists Y\ \mathrm{inf}(X, Y, 2i) \wedge \bigwedge_{j>2i} \neg \exists Y\ \mathrm{inf}(X, Y, j))
\end{aligned}$$

The first condition states that the sets $X_{q,r}$ are pairwise disjoint. The second one states that the value of the run is q_I. The third and fourth ensures that the transition relation is satisfied (whenever x has a successor or not). The fifth and sixth ensures that the guesses are consistent among siblings (whenever x has a right sibling or not). The last line encodes the accepting condition, i.e. on every infinite path the maximum rank which appears infinitely often equals $2i$. It uses the following subformula:

$$\inf(X, Y, i) \equiv \forall x \ (Y(x) \to X(x) \wedge (\bigvee_{q,\, \Omega(q)=i} X_q(x)) \wedge \exists y \ Y(y) \wedge x < y). \qquad \square$$

Finally, we conclude with a result which is a counterpart of Rabin Tree Theorem, but for the forest languages.

Proof of Theorem 6: If a forest language L is recognizable by a forest automaton then from Lemma 11 it is definable by an MSO formula. Thus from Lemma 7 the language $\mathsf{fcns}(L)$ is definable by an MSO formula, therefore from Rabin Tree Theorem $\mathsf{fcns}(L)$ is recognizable by a full binary tree automaton. Finally, from Lemma 10 the language L is recognizable by a forest automaton. $\square$

2.2.4 Regular forests and regular languages

Now we are ready to present a connection between regular forests and regular languages. First, we cite a well-known result, which is a part of Rabin Basis Theorem ([30], [40]):

Theorem 12. *Every non-empty regular full binary tree language contains at least one regular tree.*

The following theorem states that a regular forest language is determined by regular forests it contains.

Theorem 13. *If two regular forest languages contain the same set of regular forests, then they are equal.*

Proof. Assume that two regular forest languages L_1, L_2 contain the same set of regular forests, but $L_1 \neq L_2$. Without loss of generality the language $L_1 - L_2$ is not empty, but since the regular languages are closed under Boolean operations, this language is regular. It is easy to see that the first-child-next-sibling encoding preserves regularity of forests, that is t is regular if and only if $\mathsf{fcns}(t)$ is regular, thus from Theorem 12 there is a regular forest $t \in L_1 - L_2$, which contradicts the assumption that L_1 and L_2 contains the same regular forests. $\square$

2.3 Finite semigroups

A *semigroup* is an algebraic structure $(S, \cdot)$ which consists of a set S and a binary operation which is a mapping $S \times S \to S$ denoted by $\cdot$. The operation is associative, i.e. it satisfies the following axiom for all $s, t, r \in S$:

$$(s \cdot t) \cdot r = s \cdot (t \cdot r).$$

In the next chapter we will present the universal theory of algebraic structures in which we will define the notion of the language recognized by an algebraic structure and the free

algebra. Here we only mention that the free semigroup generated by an alphabet A is the set A^+ of all non-empty words over A with word concatenation as the semigroup operation. Thus a language of non-empty finite words is regular if and only if it is recognized by a finite semigroup.

A semigroup which has a neutral element is called a *monoid* and is denoted by $(S, \cdot, 1)$, where 1 is the neutral element. The free monoid generated by an alphabet A is the set A^* of all finite words over A with an empty word being a neutral element.

2.3.1 Idempotents and Ramseyan factorizations

An element s of a semigroup S is called an *idempotent* if $s \cdot s = s$. If $s \in S$ then an *idempotent power* is any idempotent of the form s^n for some $n \geq 1$. It turns out that in finite semigroups every element has an idempotent power.

Lemma 14 ([26])**.** *Let S be a finite semigroup. Then for any $s \in S$ there exists an unique idempotent power of s.*

We denote by s^ω the unique idempotent power of s.

Let $u \in A^\omega$ be an infinite word. A *factorization* of u is a sequence of non-empty finite words $\{u_n\}_{n\geq 0} \in (A^+)^\infty$ such that u is their concatenation, i.e. $u = u_0u_1u_2\cdots$. Let $\varphi\colon A^+ \to E$ be a mapping into a set of colors E. A factorization $u = u'u_0u_1\cdots$ is called *Ramseyan for φ* if there is a color $e \in E$ such that every continuous block of words from the factorization (except the first word u') is of this color, i.e.

$$\varphi(u_iu_{i+1}\cdots u_j) = e \quad \text{for every } 0 \leq i \leq j.$$

Theorem 15. *Let $\varphi\colon A^+ \to E$ be a mapping into a finite set of colors E. Every infinite word $u \in A^\infty$ admits a Ramseyan factorization for φ.*

Proof. We present the proof from [26]. Define a sequence of pairs (k_i, U_i) where $k_i \in \mathbb{N}$ and $U_i \subseteq \mathbb{N}$. First, choose $U_0 = \mathbb{N}$ and $k_0 = 0$. Suppose that U_i has been chosen. Since E is finite, there exists at least one element $e_i \in E$ such that the set

$$T = \{k \in U_i \mid \varphi(u_{k_i}u_{k_i+1}\cdots u_{k-1}) = e_i\}$$

is infinite. Then define $U_{i+1} = T$ and $k_{i+1} = \min U_{i+1}$. By the above construction $\varphi(u_{k_i}u_{k_i+1}\cdots u_{k_{i+j}-1}) = e_i$ for all $i \geq 0$ and $j > 0$. Again from finiteness of E there exists $e \in E$ and an infinite increasing sequence $\{i_n\}_{n\geq 0}$ such that $e_{i_n} = e$ for every $n \geq 0$. The subsequence $k_{i_0}, k_{i_1}, \ldots$ gives the required factorization. □

Lemma 16. *Let $(S, \cdot)$ be a finite semigroup. For every infinite sequence $\{s_n\}_{n\geq 0} \in S^\infty$ there are two elements $s, e \in S_+$ and an increasing sequence $\{k_n\}_{n\geq 1}$ such that $s_0s_1\cdots s_{k_1-1} = s$ and $s_{k_i}s_{k_i+1}\cdots s_{k_{i+1}-1} = e$ for every $k \geq 1$.*

Proof. Let $\varphi\colon S^+ \to S$ be a natural mapping from the finite sequences of S into S. From Theorem 15 there exists a Ramseyan factorization for φ which satisfies the statement of the lemma. □

2.3.2 Wilke algebras and ω-semigroups

Here we mention two algebras which are used to recognize the languages of infinite words. *Wilke algebra* is a two-sorted algebra $(S_+, S_\infty, \cdot, \cdot, ^\infty)$ with operation $\cdot\colon S_+ \times S_+ \to S_+$, operation of mixed product $\cdot\colon S_+ \times S_\infty \to S_\infty$ and operation of infinite loop $^\infty\colon S_+ \to S_\infty$. The algebra satisfies the following axioms:

(W1) $(S_+, \cdot)$ is a semigroup;

(W2) for every $s, t \in S_+$ and $u \in S_\infty$, $s(tu) = (st)u$;

(W3) for every $s, t \in S_+$, $s(ts)^\infty = (st)^\infty$,

(W4) for every $s \in S_+$ and $n \geq 1$, $(s^n)^\infty = s^\infty$;

(W5) every element of S_∞ can be written as st^∞ with $s, t \in S_+$.

The free Wilke algebra generated by an alphabet A (generators are located in S_+) is the set of all infinite ultimately-periodic words over A (the accepting set is located in S_∞). The operations correspond to, respectively, finite words concatenation, concatenation of a finite word to the left of an infinite word and infinite concatenation of a finite word with itself.

An *ω-semigroup* $(S_+, S_\infty, \cdot, \cdot, \pi)$ is essentially Wilke algebra with the infinite loop operation generalized to an infinite product operation of infinite arity $\pi\colon S_+^\infty \to S_\infty$. It satisfies axioms (W1), (W2) and the following ones:

(W3′) for every $s \in S_+$ and for every sequence $\{s_n\}_{n\geq 0} \in S_+^\infty$,

$$s\pi(s_0, s_1, s_2, \ldots) = \pi(s, s_0, s_1, s_2, \ldots),$$

(W4′) for every increasing sequence $\{k_n\}_{n\geq 1}$ and for each sequence $\{s_n\}_{n\geq 0} \in S_+^\infty$,

$$\pi(s_0 s_1 \cdots s_{k_1-1},\ s_{k_1} s_{k_1+1} \cdots s_{k_2-1},\ \ldots) = \pi(s_0, s_1, s_2, \ldots).$$

(W5′) every element of S_∞ can be written as an infinite product of elements of S_+.

The free ω-semigroup generated by an alphabet A is the set of all infinite words over A. The infinite product operation correspond to infinite concatenation of finite words.

The following theorem gives a correspondence between finite Wilke algebras and finite ω-semigroups.

Theorem 17 ([26])**.** *Every finite Wilke algebra $(S_+, S_\infty, \cdot, \cdot, ^\infty)$ can be equipped, in a unique way, with a structure of finite ω-semigroup $(S_+, S_\infty, \cdot, \cdot, \pi)$ such that for every $s \in S$, the product $\pi(s, s, s, \ldots)$ is equal to s^∞.*

Part I

Algebra

Chapter 3

Universal algebra

In this thesis different objects (such as words or trees, finite or infinite) are represented by different algebraic structures. Since these structures are quite numerous, it is therefore useful to gather common definitions and theorems in one place. That force us to take a general approach which can be rather abstract and tedious.

In this chapter we present a general theory of these algebraic structures. The presented notions should be known to the reader, since they are standard in the universal algebra. However, standard structures consist of a carrier set which holds objects of one kind. Since we need to deal with structures which can hold different kinds of objects (e.g. forests and contexts in algebra for finite trees), we need to use the theory of *multisort algebras*.

In section 3.1 we present the definition of multisort algebra and standard algebraic notions of homomorphism and free algebra. Next, we present a definition of recognizability of a language by a multisort algebra and definitions of a syntactic algebra and Myhill-Nerode relation.

In section 3.2 we present a notion of varietes of multisort algebras and varietes of languages. We then prove a general version of the theorem which shows a one-to-one correspondence between these two notions.

We illustrate the definitions with two running examples. The paragraphs which contain the running examples are indicated by a bar on the left side. The first example is the simplest algebra – a semigroup – which recognizes languages of non-empty finite words. The second one is a two-sorted structure which recognizes languages of finite forests, namely forest algebra, developed in [14].

3.1 Multisort algebras

As we said before, we need to work with algebraic structures which can hold objects of different kinds in their carriers. Therefore, a natural choice is to use multisort algebras. The next choice is to decide which sorts of an algebra contain objects which belong to the language recognized by this algebra (only one sort, some of them, or all of them). Since our ultimate goal is to present algebras for infinite trees which extend forest algebra, we think that the first choice (only one sort) is the most suitable.

Multisort algebra is described by the following ingredients:

(1) Sorts. The set T contains all the names of the sorts.

(2) Distinguished *language sort* $\tau_L \in T$.

(3) Operations. Every operation is described by its arity, a list of sorts from which it accepts arguments, and a result sort. An n-ary ($n \geq 0$) operation f is denoted by

$$f\colon \tau_1 \times \cdots \times \tau_n \to \tau$$

if its i-th argument is from the sort $\tau_i \in T$ and the result is from the sort $\tau \in T$.

That description is called the *(multisort) signature* of the multisort algebra and it is denoted by Σ.

The signature of semigroups consists of only one sort, $T = \{\tau_S\}$. Since there is only one sort, it is also the distinguished language sort. The signature contains only one binary operation $\cdot\colon \tau_S \times \tau_S \to \tau_S$.

The signature of forest algebras consists of two sorts, $T = \{\tau_H, \tau_V\}$, where τ_H is the *horizontal sort* and τ_V is the *vertical sort*. The idea is that elements of the horizontal sort correspond to finite forests and elements of the vertical sort correspond to finite contexts. The distinguished sort is τ_H, since the algebra is used to recognize language of forests.

The signature contains the following operations:

(a) horizontal concatenation $+\colon \tau_H \times \tau_H \to \tau_H$ and its identity element $0\colon \emptyset \to \tau_H$, which correspond to forest concatenation and the empty forest;

(b) vertical composition $\cdot\colon \tau_V \times \tau_V \to \tau_V$ and its identity element $\Box\colon \emptyset \to \tau_V$, which correspond to context composition and the empty context;

(c) left action of vertical sort on horizontal sort $\cdot\colon \tau_H \times \tau_V \to \tau_H$, which corresponds to composition of a context with a forests;

(d) two insertion operations $in_l, in_r\colon \tau_H \to \tau_V$, which correspond to creating a context from a forest by placing a hole next to a forest on the right or on the left.

Multisort variants of basic definitions

A *multisort set* is a family of sets $A = \{A_\tau\}_{\tau \in T}$ indexed by the names of sorts. A multisort set B is a *subset* of a multisort set A if for every $\tau \in T$ we have $B_\tau \subseteq A_\tau$. The *cartesian product* of two multisort sets A, B is a family of sets $\{A_\tau \times B_\tau\}_{\tau \in T}$. For two multisort sets A, B we can define a *multisort relation* as a subset of cartesian product $A \times B$. It is an *equivalence multisort relation* if it satisfies the usual three conditions. For two multisort sets A, B we can define a *multisort function* $f\colon A \to B$ as a family of functions $\{f_\tau\colon A_\tau \to B_\tau\}_{\tau \in T}$.

A multisort algebra $\mathbf{A}$ of signature Σ consists of:

(1) a multisort set carrier($\mathbf{A}$);

(2) functions

$$f^{\mathbf{A}}\colon \mathrm{carrier}(\mathbf{A})_{\tau_1} \times \cdots \times \mathrm{carrier}(\mathbf{A})_{\tau_n} \to \mathrm{carrier}(\mathbf{A})_\tau$$

for every operation $f\colon \tau_1 \times \cdots \times \tau_n \to \tau$ from the signature Σ.

Extending in a natural way definitions from universal algebra, we define a *cartesian product* of two multisort algebras, a *multisort subalgebra* of a multisort algebra, a *homomorphism* between two multisort algebras, a *multisort congruence relation*, and a *quotient multisort algebra*.

Terms

Let X be a multisort set which contains all variables which can appear in terms. Let A be a multisort set, which we call a set of generators. A *term* is a tree with elements of $A \cup X$ in leaves and with operations from the signature of the algebra in inner nodes. The tree has to be consistent with the types of the operations: if a node x has operation $f\colon \tau_1 \times \cdots \times \tau_n \to \tau$, then x has n children of types $\tau_1, \ldots, \tau_n$ and x has type τ.

We denote the set of all terms over $A \cup X$ by terms(A). We define the value of a term σ in the multisort algebra $\mathbf{A}$ under the valuation $X \to \text{carrier}(\mathbf{A})$ in the standard way.

We note that we can generalize the definition of a term to signatures with operations of infinite arity. But then a term must be a well-founded tree.

In this chapter we write shortly $\sigma\colon \tau' \to \tau$ for a term of type τ with one variable of type τ'.

For instance we write $\sigma\colon \tau_V \to \tau_H$ for a term with a variable from the vertical sort and result in the horizontal sort. However, in the next chapters we will write it more verbosely as a "forest-valued term with one context-valued variable".

Axioms

The one ingredient of a multisort algebra that we did not mention is an axiom. An *axiom* is a pair of terms σ, σ' of the same type, generated by the empty set. It is denoted by $\sigma = \sigma'$.

A multisort algebra $\mathbf{A}$ satisfies an axiom $\sigma = \sigma'$ if for every valuation both terms have the same value in the algebra.

The semigroup is defined by one axiom of associativity: $s \cdot (t \cdot r) = (s \cdot t) \cdot r$ for $s, t, r \in X_{\tau_S}$.

The forest algebra satisfies the following axioms: axioms of the horizontal monoid $(h + (g + f) = (h + g) + f$ and $h + 0 = 0 + h = h$ for $f, g, h \in X_{\tau_H})$, axioms of the vertical monoid $(v \cdot (w \cdot u) = (v \cdot w) \cdot u$ and $v \cdot \Box = \Box \cdot v = v$ for $v, w, u \in X_{\tau_V})$, action axiom $((v \cdot w) \cdot h = v \cdot (w \cdot h)$ for $v, w \in X_{\tau_V}$, $h \in X_{\tau_H})$ and insertion axioms $(in_l(g) \cdot h = g + h$ and $in_r(g) \cdot h = h + g$ for $g, h \in X_{\tau_H})$.

Let us fix a set Ax of axioms. Two terms σ, σ' of type τ are *immediately axiom-equivalent* if there exists an axiom $\rho = \rho'$ from the set Ax which uses variables $x_1, \ldots, x_n$ and exist terms $\sigma_1, \ldots, \sigma_n$ and term $\sigma_0[x]$ such that

$$\sigma_0[x \leftarrow \rho[x_1 \leftarrow \sigma_1, \ldots, x_n \leftarrow \sigma_n]] = \sigma, \qquad \sigma_0[x \leftarrow \rho'[x_1 \leftarrow \sigma_1, \ldots, x_n \leftarrow \sigma_n]] = \sigma'.$$

Two terms σ, σ' are *axiom-equivalent* if there is a sequence of terms $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n = \sigma'$ such that for $i = 1, \ldots, n-1$ terms σ_i and σ_{i+1} are immediately axiom-equivalent.

It is easy to see that axiom-equivalence of terms is a multisort congruence. We denote it by $\sim_{\mathsf{Ax}}$.

Free objects

Let $\mathbf{A}$ be a multisort algebra from the class $\mathcal{K}$ and let A be its set of generators. The multisort algebra $\mathbf{A}$ is called *free in the class $\mathcal{K}$ over the set A* if for every multisort algebra $\mathbf{B}$ from the class $\mathcal{K}$ and every multisort function $f\colon A \to \text{carrier}(\mathbf{B})$ there is exactly one homomorphism $\alpha\colon \mathbf{A} \to \mathbf{B}$ such that $\alpha(a) = f(a)$ for every $a \in A$.

It is easy to see that two free algebras of the same set of generators are isomorphic, therefore we denote by $\text{free}(A)$ the free algebra for the set of generators A (the class $\mathcal{K}$ is implicit in this denotation).

Let $\mathcal{K}$ be a class of all multisort algebras of signature Σ which satisfy the axioms from the set Ax. It is easy to see that the set $\text{terms}(A)/{\sim_{\mathsf{Ax}}}$ of terms divided by all the axioms is the free algebra.

How can one check if an algebra $\mathbf{A}$ from the class $\mathcal{K}$ is free in this class? The answer is given in the following lemma:

Lemma 18. *Let $\mathbf{A}$ be a multisort algebra from the class $\mathcal{K}$ and A be its set of generators. Let be $\alpha\colon \text{terms}(A) \to \mathbf{A}$ be a unique surjective homomorphism which is an identity on A and satisfies the condition*

$$\text{for all } \sigma, \sigma' \in \text{terms}(A) \quad \alpha(\sigma) = \alpha(\sigma') \Rightarrow \sigma \sim_{\mathsf{Ax}} \sigma'.$$

Then $\mathbf{A}$ is free in the class $\mathcal{K}$ over the set A.

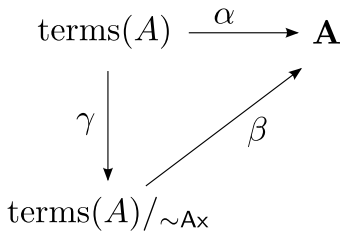


Figure 3.1

Proof. From the first theorem of isomorphism we have $\alpha = \gamma; \beta$ where $\beta\colon \text{terms}(A)/{\sim_{\mathsf{Ax}}} \to \mathbf{A}$ (see figure 3.1). From the surjectiveness of α, β is also surjective. Moreover, for any $\rho, \rho' \in \text{terms}(A)/{\sim_{\mathsf{Ax}}}$ if $\beta(\rho) = \beta(\rho')$, then for any σ, σ' such that $\gamma(\sigma) = \rho, \gamma(\sigma') = \rho'$ we have $\alpha(\sigma) = \alpha(\sigma')$, thus $\sigma \sim_{\mathsf{Ax}} \sigma'$, therefore $\rho = \rho'$. Thus β is a bijection between a free algebra $\text{terms}(A)/{\sim_{\mathsf{Ax}}}$ and $\mathbf{A}$. □

The free semigroup A^+ is the set of all non-empty finite words over A with a binary operation of word concatenation.

Let A be a finite alphabet. By A' we denote a two-sorted set which contains no elements in the horizontal sort and all single-letter contexts $A\Box = \{a\Box \mid a \in A\}$ in the vertical sort. The free forest algebra $A^{\mathsf{Fin}\triangle}$ is a two-sorted algebra over the set of generators A'. The horizontal sort is a set A^{FinFor} of all finite forests over A (which includes an empty forest), and the vertical sort is a set A^{FinCon} of all finite contexts over A (which includes an empty context). The operations are: concatenation of forests, composition of contexts, putting forest in a context (action operation), and putting a hole next to a forest (insertion operation).

We note here that our decision of using single-letter contexts as a set of generators for forests is an arbitrary one.

Recognizability of languages

A (multisort) *alphabet* is any finite multisort set A. A *language* over alphabet A is any subset $L \subseteq \text{carrier}(\text{free}(A))_{\tau_L}$. Thus a language consists of objects from the distinguished sort τ_L. We say that a surjective morphism $\alpha\colon \text{free}(A) \to \mathbf{B}$ *recognizes* L if there exists a set $F \subseteq \text{carrier}(\mathbf{B})_{\tau_L}$ such that $L = \alpha_{\tau_L}^{-1}(F)$. In other words membership $t \in L$ depends only on the value $\alpha_{\tau_L}(t)$. Such a language is said to be *recognizable* by a finite multisort algebra. We note that we could omit the requirement that α is "onto", but then in Lemma 22 we should replace "any morphism" with "any surjective morphism".

The algebra is often developed in such a way that languages recognizable by the finite algebras are precisely regular languages. However, this is not always the case.

A morphism $\alpha\colon A^+ \to S$ from the free semigroup to a semigroup S recognizes a finite word language L if there exists a set $F \subseteq S$ such that $L = \alpha^{-1}(F)$. Since there is only one sort, $\tau_L = \tau_S$. It can be proved [26] that a finite word languages is regular if and only if it is recognized by a finite semigroup.

A morphism $\alpha\colon A^{\mathsf{Fin}\triangle} \to (H, V)$ from the free forest algebra to a forest algebra (H, V) recognizes a finite forest language L if there exists a set $F \subseteq H$ such that $L = \alpha_{\tau_H}^{-1}(F)$. A language consists of elements from the sort $\tau_L = \tau_H$. It can be proved [14] that a finite forest language is regular if and only if it is recognized by a finite forest algebra.

Let L be a language over A. We define the *Myhill-Nerode relation* $\sim_L$ as a relation on $\text{carrier}(\text{free}(A))$ as follows: let τ be a sort and $s, s' \in \text{carrier}(\text{free}(A))_\tau$. Then $s \sim_L s'$ if for every term $\sigma\colon \tau \to \tau_L$ over A we have:

$$\sigma[x \leftarrow s] \in L \quad \text{if and only if} \quad \sigma[x \leftarrow s'] \in L.$$

Lemma 19. *Myhill-Nerode relation is a congruence.*

Proof. Let $f\colon \tau_1 \times \cdots \times \tau_n \to \tau$ be an operation from the signature Σ and for $i = 1, \ldots, n$ we have $s_i, t_i \in \text{carrier}(\text{free}(A))_{\tau_i}$ such that $s_i \sim_L t_i$. Let $\sigma\colon \tau \to \tau_L$ be any term. We build a family of terms $(i = 1, \ldots, n)$ $\sigma_i\colon \tau_i \to \tau_L$ such that $\sigma_i[x] = \sigma[f(t_1, \ldots, t_{i-1}, x, s_{i+1}, \ldots, s_n)]$. Then for $i = 1, \ldots, n$ we have

$$\sigma_i[s_i] \in L \quad \text{if and only if} \quad \sigma_i[t_i] \in L$$

and for $i = 1, \ldots, n-1$ we have $\sigma_i[t_i] = \sigma_{i+1}[s_{i+1}]$ therefore

$$\sigma[f(s_1, \ldots, s_n)] = \sigma_1[s_1] \in L \quad \text{if and only if} \quad \sigma_n[t_n] = \sigma[f(t_1, \ldots, t_n)] \in L,$$

thus $f(s_1, \ldots, s_n) \sim_L f(t_1, \ldots, t_n)$. □

Lemma 20. *Consider a multisort algebra* $\mathbf{A}$*, a term* $\sigma\colon \tau_1 \to \tau_2$ *and a set* $B \subseteq \text{carrier}(\mathbf{A})_{\tau_2}$*. If for* $s, t \in \text{carrier}(\mathbf{A})_{\tau_1}$ *we have* $\sigma[x \leftarrow s] \in B$ *and* $\sigma[x \leftarrow t] \notin B$ *and the term* σ *has multiple occurrences of* x*, then there is another term* $\rho\colon \tau_1 \to \tau_2$ *with exactly one occurrence of* x *and* $\rho[x \leftarrow s] \in B$ *and* $\rho[x \leftarrow t] \notin B$*.*

Proof. Let n be a number of occurrences of x in the term σ. Consider the term

$$\sigma'\colon \underbrace{\tau_1 \times \cdots \times \tau_1}_{n \text{ times}} \to \tau_2$$

which is σ with occurrences of x replaced by different variables $x_1, \ldots, x_n$. Consider for $i = 0, \ldots, k$ a family of terms $\sigma'_i \colon \emptyset \to \tau_2$ such that

$$\sigma'_i = \sigma'[x_1 \leftarrow s, \ldots, x_i \leftarrow s,\ x_{i+1} \leftarrow t, \ldots, x_n \leftarrow t].$$

Since $\sigma'_0 = \sigma[x \leftarrow t] \in B$ and $B \not\ni \sigma[x \leftarrow s] = \sigma'_n$, then there is an index i $(1 \leq i \leq n)$ such that $\sigma'_{i-1} \in B$ and $B \not\ni \sigma'_i$. Therefore we put

$$\rho[x] = \sigma'[x_1 \leftarrow s, \ldots, x_{i-1} \leftarrow s,\ x_i \leftarrow x,\ x_{i+1} \leftarrow t, \ldots, x_n \leftarrow t]. \quad \square$$

Let $s, s' \in S$ be two elements of a semigroup. From Lemma 20 every term $\sigma \colon \tau_S \to \tau_S$ can be written as $v \cdot x \cdot w$ for some $v, w \in S^1$. Therefore the Myhill-Nerode relation can be formulated as follows: $s \sim_L s'$ if for every $v, w \in S^1$ we have $vsw \in L$ if and only if $vs'w \in L$.

For forest algebra we can introduce a following relation $\approx_L$:

(a) $s \approx_L t$ if for every context p we have $ps \in L$ if and only if $pt \in L$.

(b) $p \approx_L q$ if for every forest t, we have $pt \approx_L qt$.

We can show that it is another wording of the Myhill-Nerode relation:

Lemma 21. *The relations $\sim_L$ and $\approx_L$ are the same.*

Proof. If $s \approx_L t$. Consider any term $\sigma \colon \tau_H \to \tau_H$. We can factorize $\sigma = p \cdot x$ for some context p, thus $\sigma[s] = ps$ and $\sigma[t] = pt$. Thus we have $s \sim_L t$. Similarly, if $s \sim_L t$ and any context p we consider a term $\sigma[x] = p \cdot x$. Since $\sigma[s] \in L$ if and only if $\sigma[t] \in L$ then $ps \in L$ if and only if $pt \in L$.

If $p \approx_L q$, then for every forest t, $pt \approx_L qt$, so from the definition for every forest t and context r, $rpt \in L$ if and only if $rqt \in L$. Consider any term $\sigma \colon \tau_V \to \tau_H$, we can factorize it $\sigma = r \cdot x \cdot t$ for some r, t. Thus $\sigma[p] \in L$ if and only if $\sigma[q] \in L$, so $p \sim_L q$. Similarly the other way round. $\square$

Syntactic multisort algebra

The *syntactic multisort algebra* of a language L is the quotient $\mathrm{free}(A)/\sim_L$ and denoted by $\mathrm{synt}(L)$. The *syntactic morphism* is a natural morphism $\alpha \colon \mathrm{free}(A) \to \mathrm{synt}(L)$ defined as $\alpha(w) = [w]_{\sim_L}$.

Lemma 22. *A language L over A is recognized by its syntactic morphism α. Moreover, every surjective morphism $\beta \colon \mathrm{free}(A) \to \mathbf{B}$ that recognizes L can be extended by a morphism $\gamma \colon \mathbf{B} \to \mathrm{synt}(L)$ so that $(\beta; \gamma) = \alpha$.*

Proof. Let $s, s' \in \mathrm{carrier}(\mathrm{free}(A))_{\tau_L}$. If $s \sim_L s'$ then either both s and s' belong to L, or both are outside L, otherwise they could be distinguished by a simple term $\sigma[x] = x$. Hence, the syntactic morphism recognizes L.

For the second part, there is a natural candidate for the mapping γ for $s \in \mathrm{carrier}(\mathbf{B})_\tau$:

$$\gamma_\tau(s) = \alpha_\tau(\beta_\tau^{-1}(s)).$$

We show that this definition is valid, i.e. for every $s, s' \in \mathrm{carrier}(\mathbf{B})_\tau$ if $\beta_\tau(s) = \beta_\tau(s')$, then $\alpha_\tau(s) = \alpha_\tau(s')$, Suppose otherwise that $\alpha_\tau(s) \neq \alpha_\tau(s')$, thus there exists a term $\sigma \colon \tau \to \tau_L$ such that exactly only one of $\sigma[x \leftarrow s]$, $\sigma[x \leftarrow s']$ belongs to L. But $\beta_\tau(s) = \beta_\tau(s')$, thus $\beta_{\tau_L}(\sigma[x \leftarrow s]) = \beta_{\tau_L}(\sigma[x \leftarrow s'])$, which contradicts with the fact that β recognizes L. $\square$

Moore's algorithm

Now we present an algorithm to calculate the syntactic multisort algebra for a language recognized by some finite multisort algebra $\mathbf{A}$ and a subset $F \subseteq \text{carrier}(\mathbf{A})_{\tau_L}$ if we have a finite number of operations in the signature. We mark iteratively all the pairs of elements which are not equivalent. First, one marks all pairs of elements from $\text{carrier}(\mathbf{A})_{\tau_L}$ consisting of an element of F and of an element of $\text{carrier}(\mathbf{A})_{\tau_L} - F$. Then one considers a pair (s, s') of elements of $\text{carrier}(\mathbf{A})_\tau$ and marks it if there exists an operation $f\colon \tau_1 \times \cdots \times \tau_n \to \tau'$ in the signature, index j such that $\tau_j = \tau$, elements $t_i \in \text{carrier}(\mathbf{A})_{\tau_i}$ for $i \neq j$ such that the pair

$$(f^{\mathbf{A}}(t_1, \ldots, t_{j-1},\ s,\ t_{j+1}, \ldots, t_n),\ f^{\mathbf{A}}(t_1, \ldots, t_{j-1},\ s',\ t_{j+1}, \ldots, t_n))$$

is marked. We iterate until we cannot add more pairs. The following lemma shows the correctness of the above algorithm.

Lemma 23. *At the end of the algorithm a pair (s, s') of elements from $\text{carrier}(\mathbf{A})_\tau$ is marked if and only if it does not belong to $\sim_L$.*

Proof. Assume that $s \not\sim_L s'$, that there is a term $\sigma\colon \tau \to \tau_L$ such that exactly one of $\sigma[x \leftarrow s]$ and $\sigma[x \leftarrow s']$ belongs to L. From Lemma 20 the term σ can have only one occurrence of the variable x. We can factorize this term as follows: there is a number m and operations $f_1, \ldots, f_m$ such that for $i = 1, \ldots, m$

$$\sigma_i[x] = f_i^{\mathbf{A}}(s_i^1, \ldots, s_i^{k_i - 1},\ \sigma_{i+1}[x],\ s_i^{k_i+1}, \ldots, s_i^{n_i})$$

where $f_i\colon \tau_i^1 \times \cdots \times \tau_i^{n_i} \to \tau^i$ and $\tau_i^{k_i} = \tau^{i+1}$, and finally $\sigma_{m+1}[x] = x$ and $\tau^{m+1} = \tau$. Since $\sigma_1[x] = \sigma[x]$, then a pair $(\sigma_1[x \leftarrow s], \sigma_1[x \leftarrow s'])$ will be marked in the initial step of the algorithm. It is easy to see that for $i = 1, \ldots, m$ pair $(\sigma_{i+1}[x \leftarrow s], \sigma_{i+1}[x \leftarrow s'])$ will be marked at the end of i-th step. Thus the pair (s, s') will be marked at the end of m-th step.

For the proof for the other way round we just record the operations which lead us to marking (s, s') and from them we construct (in the similar fashion as above) the desired term which justifies that $s \not\sim_L s'$. □

3.2 Varietes of multisort algebras and varietes of languages

We say that two distinct elements $s, s' \in \text{carrier}(\mathbf{A})_\tau$ from the same sort $\tau \in T - \{\tau_L\}$ (which is different from the distinguished language sort τ_L) are *language-equivalent* if they cannot be distinguished by any term of type τ_L, i.e. for every term $\sigma\colon \tau \to \tau_L$ over $\mathbf{A}$ we have $\sigma[x \leftarrow s] = \sigma[x \leftarrow s']$.

From Lemma 20 we get the same definition if we restrict to terms of exactly one occurrence of the variable x.

A multisort algebra $\mathbf{A}$ is called *faithful* if there are no two distinct elements which are language-equivalent. Language-equivalence is a multisort equivalence relation. We define operation $\text{faith}(\mathbf{A})$ which takes a multisort algebra $\mathbf{A}$ and divides it by the language-equivalence, which results in a faithful multisort algebra.

Since the semigroup is an algebra of one sort, it is by definition always faithful.

There is a decidable condition which ensures that a forest algebra is faithful. A forest-term is a term of type τ_H with no variables. A context-term is a term of type τ_V with no variables. From Lemma 20 we know that every term $\sigma\colon \tau_V \to \tau_H$ can be written as $q \cdot x \cdot t$ for some context-term q and forest-term t. Thus, a forest algebra is faithful if there are no two elements $v, w \in V$ such that $q \cdot v \cdot h = q \cdot w \cdot h$ for every $h \in H$, thus it is faithful if there are no two elements $v, w \in V$ such that $v \cdot h = w \cdot h$ for every $h \in H$.

Lemma 24. *Every syntactic multisort algebra is faithful.*

Proof. It follows from the fact that if two elements s, s' from the same sort are language-equivalent, then $s \sim_L s'$. □

A *multisort algebra variety* (see [27]) is a class $\mathbb{V}$ of finite faithful multisort algebras which is closed under:

(1) cartesian products: if $\mathbf{A}, \mathbf{B} \in \mathbb{V}$ then $\mathbf{A} \times \mathbf{B} \in \mathbb{V}$;

(2) faithful quotients of multisort subalgebras: if $\mathbf{A} \in \mathbb{V}$ and $\mathbf{B}$ is a multisort subalgebra of $\mathbf{A}$ then $\text{faith}(\mathbf{B}) \in \mathbb{V}$;

(3) faithful quotients of homomorphic images: if $\mathbf{A} \in \mathbb{V}$ and $\alpha\colon \mathbf{A} \to \mathbf{B}$ is a homomorphism then $\text{faith}(\alpha(\mathbf{A})) \in \mathbb{V}$;

We note that in the literature the term „algebra pseudovariety" is also used – to emphasize that variety consists only of finite algebras.

A *language variety* is an operator $\mathcal{V}$ which assigns to each finite alphabet A a family $\mathcal{V}(A)$ of languages over A and the following conditions hold for every alphabet A and B:

(1) $\mathcal{V}(A)$ is closed under Boolean operations, including complementation: if $L, K \in \mathcal{V}(A)$ then $L \cap K, L \cup K, \text{carrier}(\text{free}(A))_{\tau_L} - L \in \mathcal{V}(A)$;

(2) morphic preimages: if $L \in \mathcal{V}(A)$ and $\alpha\colon \text{free}(B) \to \text{free}(A)$ is a morphism then $\alpha^{-1}_{\tau_L}(L) \in \mathcal{V}(B)$;

(3) quotients: if $L \in \mathcal{V}(A)$ and $\sigma \in \text{carrier}(\text{free}(A \cup \{x\}))_{\tau_L}$ then $\sigma^{-1}L \in \mathcal{V}(A)$ where the quotient is defined as

$$\sigma^{-1}L = \{t \in \text{carrier}(\text{free}(A))_{\tau_L} \mid \sigma[x \leftarrow t] \in L\}.$$

Let $\mathbb{V}$ be a multisort algebra variety. For each finite alphabet A we consider the class of languages that are recognized by some multisort algebra from $\mathbb{V}$ (equivalently that their syntactic multisort algebra belongs to $\mathbb{V}$):

$$\mathcal{V}(A) = \{L \subseteq \text{carrier}(\text{free}(A))_{\tau_L} \mid \text{synt}(L) \in \mathbb{V}\}. \tag{3.1}$$

We call the mapping $\mathcal{V}$ the *language variety associated to* $\mathbb{V}$, and we denote it by $\mathsf{Lang}(\mathbb{V})$. The following lemma shows that the mapping deserves its name.

Lemma 25. *If $\mathbb{V}$ is a multisort algebra variety then $\mathcal{V} = \mathsf{Lang}(\mathbb{V})$ is a language variety.*

Proof. Let L, K be two languages from $\mathcal{V}(A)$, recognized by morphisms f, g into two multisort algebras $\mathbf{A}, \mathbf{B} \in \mathbb{V}$:

$$f\colon \mathrm{free}(A) \to \mathbf{A}, \quad g\colon \mathrm{free}(A) \to \mathbf{B}.$$

(1) The complement of L is also recognized by morphism f, only with complemented accepting set:

$$\mathrm{carrier}(\mathrm{free}(A))_{\tau_L} - L = f^{-1}_{\tau_L}(\mathrm{carrier}(\mathbf{A})_{\tau_L} - f_{\tau_L}(L)).$$

The language $L \cap K$ is recognized by the product morphism $f \times g\colon \mathrm{free}(A) \to \mathbf{A} \times \mathbf{B}$ defined as

$$(f \times g)(s) = (f(s), g(s)),$$

since

$$L \cap K = (f \times g)^{-1}_{\tau_L}(f_{\tau_L}(L) \cap g_{\tau_L}(K)).$$

The result follows from $\mathbf{A} \times \mathbf{B} \in \mathbb{V}$, since $\mathbb{V}$ is closed under cartesian products.

(2) The preimage $\alpha^{-1}_{\tau_L}(L)$ is recognized by composition $(\alpha; f)\colon \mathrm{free}(B) \to \mathbf{A}$, since

$$\alpha^{-1}_{\tau_L}(L) = (\alpha; f)^{-1}_{\tau_L}(f_{\tau_L}(L)).$$

(3) If $\sigma \in \mathrm{carrier}(\mathrm{free}(A \cup \{x\}))_{\tau_L}$ then $\sigma^{-1}L$ is also recognized by morphism f, only with a changed accepting set:

$$\sigma^{-1}L = f^{-1}_{\tau_L}(\{s \in \mathrm{carrier}(\mathbf{A})_{\tau_L} \mid f(\sigma)[x \leftarrow s] \in f_{\tau_L}(L)\}). \qquad \square$$

There is a one-to-one correspondence between multisort algebra varietes and language varietes. In order to prove it, we show that the operation Lang is bijective. First, we show that Lang is monotonic in respect to the inclusion operation, which also show that Lang is injective:

Lemma 26. *Let $\mathbb{V}$, $\mathbb{W}$ be two multisort algebra varietes and $\mathsf{Lang}(\mathbb{V}) = \mathcal{V}$, $\mathsf{Lang}(\mathbb{W}) = \mathcal{W}$. Then $\mathbb{V} \subseteq \mathbb{W}$ if and only if for every alphabet A, $\mathcal{V}(A) \subseteq \mathcal{W}(A)$.*

Proof. The "only if" implication is obvious from the definition. For the "if" implication let $\mathbf{A} \in \mathbb{V}$. Consider a natural morphism $f\colon \mathrm{free}(\mathrm{carrier}(\mathbf{A})) \to \mathbf{A}$. Let $\mathrm{carrier}(\mathbf{A})_{\tau_L} = \{s_1, \ldots, s_n\}$ and consider languages

$$L_i = f^{-1}_{\tau_L}(s_i).$$

Since the morphism f recognizes each language L_i, then from Lemma 22 the syntactic algebra $\mathbf{A}_i$ of L_i divides the algebra $\mathbf{A}$. Thus $\mathbf{A}_i \in \mathbb{V}$, and moreover $L_i \in \mathcal{V}(\mathrm{carrier}(\mathbf{A})) \subseteq \mathcal{W}(\mathrm{carrier}(\mathbf{A}))$. Therefore $\mathbf{A}_i \in \mathbb{W}$. Since $\mathbb{W}$ is closed under cartesian products, thus

$$\mathbf{A}_1 \times \cdots \times \mathbf{A}_n \in \mathbb{W}.$$

Consider the morphism

$$g\colon \mathrm{free}(\mathrm{carrier}(\mathbf{A})) \to \mathbf{A}_1 \times \cdots \times \mathbf{A}_n$$

defined as $g(w) = (g_1(w), \ldots, g_n(w))$.

We show that for every two distinct elements from $\mathrm{carrier}(\mathbf{A})$ their images through g are different. Take $s_i, s_j \in \mathrm{carrier}(\mathbf{A})_{\tau_L}$. Since $f^{-1}_{\tau_L}(s_i) \in L_i$ and $f^{-1}_{\tau_L}(s_j) \notin L_i$, thus $g_i(s_i) \neq g_i(s_j)$, therefore $g(s_i) \neq g(s_j)$.

Now take $t, s \in \text{carrier}(\mathbf{A})_\tau$ where $\tau \in T - \{\tau_L\}$. Since $t \neq s$ then from faithfulness there exists a term $\sigma\colon \tau \to \tau_L$ such that $\sigma[x \leftarrow t] \neq \sigma[x \leftarrow s]$. Thus these are two different elements from $\text{carrier}(\mathbf{A})_{\tau_L}$, therefore $g(\sigma[x \leftarrow t]) \neq g(\sigma[x \leftarrow s])$. It follows that $g(t) \neq g(s)$.

Therefore $\mathbf{A}$ is isomorphic to $g(\mathbf{A})$, thus it is a divisor of $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$, so it belongs to $\mathbb{W}$. □

Let $\mathcal{L}$ be a language variety. By $\mathbb{L} = \mathsf{Alg}(\mathcal{L})$ we define the class of faithful multisort algebras, such that every language over A recognized by these algebras belongs to $\mathcal{L}(A)$. We want to show that in fact $\mathsf{Lang}(\mathbb{L}) = \mathcal{L}$, which shows that Lang is surjective.

Lemma 27. *Let $\mathcal{L}$ be a language variety. Let $\mathbb{L}'$ be a multisort algebra variety generated by syntactic algebras of languages from $\bigcup \mathcal{L}(A)$ where the union ranges over all finite alphabets A. Then $\mathbb{L}' = \mathsf{Alg}(\mathcal{L})$.*

Proof. From the definition of $\mathbb{L} = \mathsf{Alg}(\mathcal{L})$ and $\mathbb{L}'$ it is easy to show that

$$\mathsf{Lang}(\mathbb{L}) \subseteq \mathcal{L} \subseteq \mathsf{Lang}(\mathbb{L}'). \tag{3.2}$$

Thus from Lemma 26 we get that $\mathbb{L} \subseteq \mathbb{L}'$.

To prove the inclusion $\mathbb{L}' \subseteq \mathbb{L}$ we follow the argument from [26]. Let $\mathbf{A} \in \mathbb{L}'$, thus there are multisort algebras $\mathbf{A}_1, \ldots, \mathbf{A}_n$ such that $\mathbf{A}$ divides $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$ and for $i = 1, \ldots, n$ the algebra $\mathbf{A}_i$ is the syntactic multisort algebra of some language $L_i \in \mathcal{L}(A_i)$ over some alphabet A_i with the syntactic morphism $\alpha_i\colon \text{free}(A_i) \to \mathbf{A}_i$.

To prove that $\mathbf{A} \in \mathbb{L}$ we must show that every language $L \in \text{carrier}(\text{free}(A))_{\tau_L}$ recognized by $\mathbf{A}$ is in $\mathcal{L}(A)$. Fix such language L. Observe, that the syntactic multisort algebra $\mathbf{S}$ of L divides $\mathbf{A}$, therefore it divides $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$.

Since $\mathbf{S}$ divides $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$, $\mathbf{S}$ is a quotient of a subalgebra $\mathbf{T}$ of $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n$. Therefore $\mathbf{T}$ recognizes L and there exist a morphism $\gamma\colon \text{free}(A) \to \mathbf{T}$ such that $L = \gamma_{\tau_L}^{-1}(F)$ for some $F \subseteq \text{carrier}(\mathbf{T})_{\tau_L}$.

Denote by $\pi_i\colon \mathbf{A}_1 \times \cdots \times \mathbf{A}_n \to \mathbf{A}_i$ the projection morphism $\pi_i(s_1, \ldots, s_n) = s_i$ and set $\gamma_i = \gamma; \pi_i$.

Since α_i is surjective, for every letter $a \in A$ we can choose an element from $\alpha_i^{-1}(\gamma_i(a))$. By freenes of $\text{free}(A)$ this mapping extends uniquely to a morphism. This morphism $\beta_i\colon \text{free}(A) \to \text{free}(A_i)$ satisfies $\gamma_i = \beta_i; \alpha_i$. On figure 3.2 there is a commuting diagram which illustrates the situation.

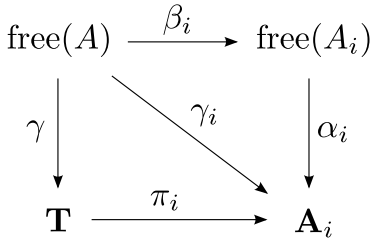


Figure 3.2

Therefore

$$L = \gamma^{-1}(F) = \bigcup_{s \in F} \gamma^{-1}(s) = \bigcup_{s \in F} \bigcap_{1 \leq i \leq n} \gamma_i^{-1}(s_i) = \bigcup_{s \in F} \bigcap_{1 \leq i \leq n} \beta_i^{-1}(\alpha_i^{-1}(s_i)),$$

where $s = (s_1, \ldots, s_n)$. In Lemma 28 we show that $\alpha_i^{-1}(s_i) \in \mathcal{L}(A_i)$. Since language varietes are closed under morphic preimages, $\beta_i^{-1}(\alpha_i^{-1}(s_i)) \in \mathcal{L}(A)$. Since they are closed under Boolean operations, $L \in \mathcal{L}(A)$, which concludes the proof. □

Lemma 28. *Let $\mathcal{L}$ be a language variety and A a finite alphabet. Let $L \in \mathcal{L}(A)$ and $\alpha\colon \mathrm{free}(A) \to \mathrm{synt}(L)$ be the syntactic morphism of L. Then for each $s \in \mathrm{synt}(L)$ we have $\alpha^{-1}(s) \in \mathcal{L}(A)$.*

Proof. Let $s \in \mathrm{carrier}(\mathrm{synt}(L))_\tau$. By the definition of the syntactic morphism, $\alpha_\tau^{-1}(s)$ is an equivalence class of $\sim_L$. Let $w \in \mathrm{carrier}(\mathrm{free}(A))_\tau$, then

$$[w]_{\sim_L} = \bigcap_\sigma \{\sigma^{-1}L \mid \sigma[w] \in L\} - \bigcup_\sigma \{\sigma^{-1}L \mid \sigma[w] \notin L\},$$

where the sums are over terms $\sigma\colon \tau \to \tau_0$. Now every $\sigma^{-1}L$ is in $\mathcal{L}(A)$. Moreover, if $\sigma^{-1}L = \rho^{-1}L$ then $\alpha(\sigma) = \alpha(\rho)$, so there are only finitely many different sets $\sigma^{-1}L$. Therefore $[w]_{\sim_L} \in \mathcal{L}(A)$, since language varietes are closed under quotients. □

It is easy to see that from Lemma 27 and from (3.2) we get that $\mathsf{Lang}(\mathbb{L}) = \mathcal{L}$. Therefore we get the following theorem, which is a multisort version of Eilenberg Theorem [20]:

Theorem 29. *The operation* Lang *is a bijection which establishes a one-to-one correspondence between multisort algebra varietes and language varietes.*

3.2.1 Identities

The reason why we are interested in the variety theory is that we want to effectively characterize classes of forest languages that are definable in various logics, and many such classes are varietes of languages. The algebraic approach often gives us characterizations in terms of identities, i.e. we have a theorem which states that a language L belongs to a certain class of languages if and only if its syntactic algebra $\mathrm{synt}(L)$ satisfies certain identities. An *identity* has the same form as an axiom and, just like an axiom, imposes some restrictions on an algebra.

It is easy to see that the word language $L \subseteq A^+$ is "commutative" (i.e. membership in L does not depend on the order of the letters), if and only if its syntactic semigroup $S = \mathrm{synt}(L)$ satisfies the identity

$$s \cdot t = t \cdot s \qquad \text{for all } s, t \in S,$$

i.e. this semigroup is commutative.

Note that we can combine this identity with the semigroup axiom $(s \cdot t) \cdot r = s \cdot (t \cdot r)$ to get the set of axioms which defines the class of „commutative semigroups". This shows that the difference between identity and axiom depends on context.

Similarly, a finite forest language is "commutative" (i.e. closed under rearranging the siblings) if and only if its syntactic forest algebra (H, V) satisfies the identity

$$h + g = g + h \qquad \text{for all } h, g \in H.$$

It is easy to see that if a class of languages has a characterization in terms of identities, this class is in fact a variety of languages, since the class of algebras satisfying a certain set of identities is a multisort algebra variety. Reiterman ([31], [27]) proved that if the class is defined by a certain kind of identities (slightly more general than defined above), then it is an algebra variety. We believe that this theorem can be generalized to the multisort case.

Chapter 4

The algebra for infinite thin forests

In this chapter we define two algebraic objects which will allow us to deal with regular languages of infinite thin forests. When defining an algebraic object, such as a monoid, semigroup, ω-semigroup, Wilke algebra, or forest algebra, one gives its sorts, operations and axioms. Once these operations and axioms are given, a set of generators (the alphabet) determines the free object (e.g. all words and non-empty words in the case of monoids and semigroups, all finite and infinite words and all ultimately periodic words in the case of ω-semigroups and Wilke algebras, and finite trees and contexts in the case of forest algebras).

The first object we define is a *regular-thin-forest algebra*. Its operations and axioms are constructed in such a manner that the free object of this algebra is the set of all regular thin forests and regular thin contexts. We denote this free algebra by

$$A^{\mathsf{regThin}^{\triangle}} = (A^{\mathsf{regThinFor}}, A^{\mathsf{regThinCon}}).$$

In a sense regular-thin-forest algebra is a generalization of both Wilke algebras and forest algebras.

The second object is an *unrestricted-thin-forest algebra*. The free object of this algebra is the set of all thin forests and thin contexts, which we denote by

$$A^{\mathsf{Thin}^{\triangle}} = (A^{\mathsf{ThinFor}}, A^{\mathsf{ThinCon}}).$$

In a sense unrestricted-thin-forest algebra is a generalization of both ω-semigroups and forest algebras.

Figure 4.1 briefly lists the operations and axioms of three algebras for forests. It can be seen that regular-thin-forest algebra generalizes forest algebra and unrestricted-thin-forest algebra generalizes them both.

In section 4.1 we formally define regular-thin-forest algebras and unrestricted-thin-forest algebras. Next, in section 4.2, we prove that every (regular) thin forest and (regular) thin context can be generated by these algebras, and the free objects are exactly those mentioned before. Finally, in section 4.3, we present a theorem which establishes correspondence between these algebras, namely that every regular-thin-forest algebra can be uniquely extended to an unrestricted-thin-forest algebra. This theorem is a close analogue of similar theorem for Wilke algebras and ω-semigroups.

In section 4.4 we show the correspondence between languages recognized by finite unrestricted-thin-forest algebras and regular languages of thin forests, namely that these

Algebra	Free algebra	Operations	Axioms
forest algebra	$A^{\mathsf{Fin}^\triangle}$ finite trees and finite contexts	$+\colon H \times H \to H$ $\cdot\colon V \times V \to V$ $\cdot\colon V \times H \to H$ $in_l, in_r\colon H \to V$	$(H, +, 0)$ is a monoid $(V, \cdot, \Box)$ is a monoid $v(wh) = (vw)h$ $in_l(h)g = h + g$
regular-thin-forest algebra	$A^{\mathsf{regThin}^\triangle}$ regular thin trees and regular thin contexts	$^\infty\colon V_+ \to H$	$(vw)^\infty = v(wv)^\infty$ $(v^n)^\infty = v^\infty$ for $n \geq 1$
unrestricted-thin-forest algebra	$A^{\mathsf{Thin}^\triangle}$ thin trees and thin contexts	$\pi\colon V_+^\infty \to H$	$\pi(v_0, v_1, v_2, \ldots) =$ $= \pi(v_0 \cdots v_{k_1}, v_{k_1+1} \cdots v_{k_2}, \ldots)$ $v_0\pi(v_1, v_2, \ldots) = \pi(v_0, v_1, v_2, \ldots)$

Figure 4.1

classes of languages are the same. Since regular languages of forests are uniquely determined by regular forests they contain, the above correspondence in a sense applies also to regular-thin-forest algebras.

4.1 Regular-thin-forest algebra and unrestricted-thin-forest algebra

Regular-thin-forest algebra and unrestricted-thin-forest algebra have much in common. First, we formally define the former and then we we describe the differences between these two algebras.

Regular-thin-forest algebra is a three-sorted algebra $(H, V_+, V_\Box, act, in_l, in_r, inf)$. It consists of two monoids H and $V = V_+ \cup V_\Box$ (divided into a subsemigroup V_+ and a submonoid $V_\Box$) along with an operation of left action $act\colon H \times (V_+ \cup V_\Box) \to H$ of $V_+ \cup V_\Box$ on H, two operations $in_l, in_r\colon H \to V_\Box$, and an infinite loop operation $inf\colon V_+ \to H$. Instead of writing $act(h, v)$, we write vh (notice a reversal of arguments). Instead of writing $inf(v)$, we write v^∞.

The above construction is based on forest algebra (see [14]). In fact we take forest algebra and introduce the new operation inf; this operation corresponds to infinite composition of contexts. However, since infinite composition is defined only for guarded contexts, we are forced to make a distinction between guarded and non-guarded objects also in the algebra. There are several ways to do this, e.g. in [7] we forced that no operation produces non-guarded objects, thus instead of operations in_l and in_r we introduced a bigger set of operations (we do the same in chapter 5).

Here we take another approach and we divide the sort V into two parts V_+ and $V_\Box$ which correspond to guarded and non-guarded objects, respectively. Since this distinction is rather technical, we can still think about the algebra as being two-sorted with sorts $V = V_+ \cup V_\Box$ and H, but with every operation on V implicitly working on V_+ and $V_\Box$.

We will call H the *horizontal monoid* and V the *vertical monoid*.

Axioms. A regular-thin-forest algebra must satisfy the following axioms:

(A1) $(H, +, 0)$ is a monoid with an operation $+$ and neutral element 0,

(A2) $(V, \cdot, \Box)$ is a monoid with an operation $\cdot$ and neutral element $\Box$; it contains two disjoint subalgebras: $(V_\Box, \cdot, \Box)$ is a monoid and $(V_+, \cdot)$ is a semigroup,

(A3) (action axiom) $(vw)h = v(wh)$ for every $v, w \in V$, $h \in H$,

(A4) (insertion axiom) $in_l(h)g = h + g$, $in_r(h)g = g + h$ for every $h, g \in H$,

(A5) $(vw)^\infty = v(wv)^\infty$ for $v, w \in V$, excluding the case when $v, w \in V_\Box$,

(A6) $(v^n)^\infty = v^\infty$ for $v \in V_+$ and every $n \geq 1$.

The definition of *unrestricted-thin-forest algebra* is the same as regular-thin-forest algebra, but the infinite loop operation is generalized by an *infinite product*: $\pi \colon V_+^\infty \to H$, which has its own versions of axioms (A5) and (A6):

(A5′) for every $v \in V_+$ and for every sequence $\{v_n\}_{n\geq 0} \in V_+^\infty$,

$$v\pi(v_0, v_1, v_2, \ldots) = \pi(v, v_0, v_1, v_2, \ldots),$$

(A6′) for every increasing sequence $\{k_n\}_{n\geq 1}$ and for each sequence $\{v_n\}_{n\geq 0} \in V_+^\infty$,

$$\pi(v_0 v_1 \cdots v_{k_1-1},\ v_{k_1} v_{k_1+1} \cdots v_{k_2-1},\ \ldots) = \pi(v_0, v_1, v_2, \ldots).$$

It is easy to see that the infinite loop operation can be expressed as $v^\infty = \pi(v, v, v, \ldots)$.

Free objects. Given an alphabet A we define the *free regular-thin-forest algebra* over A, which is denoted by $A^{\mathsf{regThin}^\triangle}$, as follows, using the operations defined in section 2.1:

(a) the horizontal monoid is the set of regular thin forests over A, with the operation of forest concatenation;

(b) the vertical monoid is the set of regular thin contexts over A (respectively guarded and non-guarded), with the operation of context composition;

(c) the action is the operation of composition a context with a forest;

(d) the in_l operation takes a regular thin forest and transforms it into a regular thin context with a hole to the right of all the roots in the forest (similarly for in_r, but the hole is to the left of the roots);

(e) the infinite loop operation takes a regular thin context and transforms it into a regular thin forest by performing infinite composition.

In the same manner we define the *free unrestricted-thin-forest algebra* over A, which is denoted by $A^{\mathsf{Thin}^\triangle}$, by above conditions (a)–(d) without the assumption of regularity and a condition

(e′) the infinite product operation takes an infinite sequence of thin contexts and transforms it into a thin forest by performing infinite composition.

Recall that for a letter $a \in A$ we denote by $a\Box$ a regular thin context which consists of one root with label a and a hole under it. We denote also $A\Box = \{a\Box \mid a \in A\}$.

Theorem 30. *The algebra $A^{\mathsf{regThin}^\triangle}$ is a regular-thin-forest algebra. Moreover, it is the free algebra in the class of regular-thin-forest algebras over the generator set $A\Box$.*

Similarly, the algebra $A^{\mathsf{Thin}^\triangle}$ is an unrestricted-thin-forest algebra and the free algebra in the class of unrestricted-thin-forest algebras over the generator set $A\Box$.

Proof. It is easy to check that $A^{\mathsf{regThin}^\triangle}$ and $A^{\mathsf{Thin}^\triangle}$ satisfy the respective axioms.

Thanks to Lemma 34 and Lemmas 38, 39, the function α satisfies the premises of Lemma 18 which characterizes the free objects. The formulations and proofs of the aforementioned lemmas are delegated to section 4.2. □

Faithfulness. A faithful regular-thin-forest algebra must satisfy an additional condition:

(A7) (faithfulness condition) there are no two elements $v, w \in V$ such that

(a) $vh = wh$ for all $h \in H$ and

(b) $(vu)^\infty = (wu)^\infty$ for all $u \in V$ such that $vu, wu \in V_+$.

Lemma 31. *A regular-thin-forest algebra is faithful if and only if it satisfies the condition (A7).*

Proof. Consider two distinct elements $v, w \in V$. The algebra is faithful if and only if there is a forest-valued term σ with one context-valued variable x over the signature of regular-thin-forest algebra such that $\sigma[x \leftarrow v] \neq \sigma[x \leftarrow w]$ and x appears once in the term (by Lemma 20). Such term can be written as either $p \cdot x \cdot t$ for some context-term p and forest-term t or as $\sigma'[y \leftarrow (q' \cdot x \cdot q'')^\infty]$ for some forest-valued term σ' with one forest-valued variable and some context-terms q', q''. In the former case there are elements $u \in V$ and $h \in H$ (generated by p and t respectively) such that $u \cdot v \cdot h \neq u \cdot w \cdot h$, thus $v \cdot h \neq w \cdot h$ for some $h \in H$. In the latter case there are elements $u', u'' \in V$ (generated by q', q'') such that $(u' \cdot v \cdot u'')^\infty \neq (u' \cdot w \cdot u'')^\infty$. Since $(u' \cdot v \cdot u'')^\infty = u'(v \cdot u''u')^\infty$, thus $(v \cdot u)^\infty \neq (w \cdot u)^\infty$ for $u = u''u'$. □

We will assume in this chapter that all thin-forest algebras are faithful.

Regular-thin-forest algebra can be viewed as a generalization of Wilke algebra where the horizontal monoid H extends the set of infinite words and the vertical monoid V extends the set of finite words. One can observe that the faithfulness condition is very similar to Arnold congruence [1] on finite words in the Wilke algebra. The reason why we have to state this condition along with the congruence on the regular-thin-forest algebra is as follows: in the Wilke algebra objects of both sorts (finite and infinite words) can belong to the language, whereas in the regular-thin-forest algebra objects of only one sort (H) belong to the language. This also means that if we would like to apply the general theory from the chapter 3 to the Wilke algebras, we should allow to distinguish more language sorts.

Syntactic sugar for insertion operations. Since insertion operations are somewhat cumbersome to use, we can introduce two additional operations $+_{HV} \colon H \times V \to V$ and $+_{VH} \colon V \times H \to V$:

$$h +_{HV} v = in_l(h)v, \quad v +_{VH} h = in_r(h)v.$$

The images of the operation $+_{VH}$ on sets $H \times V_+$ and $H \times V_\Box$ are V_+ and $V_\Box$, respectively. Analogously for $+_{HV}$.

Lemma 32. *The operations $+_{HV}$ and $+_{HV}$ are associative with $+$ and preserve zero, i.e.:*

$$\begin{aligned}(h+g)+_{HV} v &= h+_{HV}(g+_{HV} v)\\ v+_{VH}(h+g) &= (v+_{VH} h)+_{VH} g\\ (h+_{HV} v)+_{VH} g &= h+_{HV}(v+_{VH} g)\\ 0+_{HV} v = v \qquad & \qquad v+_{VH} 0 = v\end{aligned}$$

Proof. First, we show that they are associative. We do only the first identity, the remaining two are similar. Thanks to faithfulness we have to show that both sides act the same on any $f \in H$ and behave the same on any $(\,\cdot\, u)^\infty$ for $u \in V$. We do only the former case, the latter is analogous:

$$\begin{aligned}((h+g)+_{HV} v)f = in_l(h+g)vf = in_l(h+g)(vf) = (h+g)+vf = h+(g+vf) =\\ = in_l(h)(g+vf) = in_l(h)(in_l(g)(vf)) = in_l(h)(in_l(g)vf) =\\ = in_l(h)(in_l(g)v)f = (h+_{HV}(in_l(g)v))f = (h+_{HV}(g+_{HV} v))f.\end{aligned}$$

They also preserve zero (again, we show the first case of the first identity):

$$(0+_{HV} v)f = in_l(0)vf = in_l(0)(vf) = 0+vf = vf. \qquad \square$$

Therefore we can omit the subscripts and write $+$ instead of $+_{HV}$ and $+_{VH}$. We can now verify that the operations have a desired meaning:

$$(h+v+g)f = h+vf+g \quad \text{and} \quad \big((h+v+g)u\big)^\infty = (h+vu+g)^\infty.$$

Verifying satisfiability of the axioms. To check if a multisort algebra of the signature of regular-thin-forest algebras is in fact a regular-thin-forest algebra one must verify that the algebra satisfies the axioms. The only difficulty lies in the axiom (A6), which represents an infinite number of axioms. However, checking this axiom can be realized by checking one identity:

Lemma 33. *The axiom (A6) is satisfied if and only if*

$$(v^\omega)^\infty = v^\infty \text{ for } v \in V_+. \tag{4.1}$$

Proof. The "only if" part is obvious. The proof for the "if" part is based on the fact that $(v^n)^\omega = v^{n\cdot\omega} = v^\omega$, thus

$$(v^n)^\infty \overset{(4.1)}{=} \big((v^n)^\omega\big)^\infty = (v^\omega)^\infty \overset{(4.1)}{=} v^\infty. \qquad \square$$

Note that (4.1) involves the idempotent power, so it is not formally an axiom, since the operation $v \mapsto v^\omega$ is not a part of the signature.

4.2 Free objects

This section is devoted to finishing the proof of Theorem 30, i.e. that the free regular-thin-forest algebra over A is the free object (in the sense of universal algebra) among regular-thin-forest algebras over A when the set of generators is $A\Box$, and similarly that the free unrestricted-thin-forest algebra is the free object among unrestricted-thin-forest algebras.

The algebras are generated by the alphabet. Let σ be a term without variables over A and let $\alpha(\sigma)$ be the evaluation of this term in $A^{\mathsf{regThin}^\triangle}$ (respectively in $A^{\mathsf{Thin}^\triangle}$) with the valuation $f(a) = a\Box$.

First, we prove that the free regular-thin-forest algebra and the free unrestricted-thin-forest algebra are generated by elements of $A\Box$, i.e. α is surjective. All trees, forests and contexts in the following proofs are thin.

Lemma 34. *For every thin forest t (thin context p) there is a term $\sigma(t)$ ($\sigma(p)$) without variables such that it evaluates in the unrestricted-thin-forest algebra over A to t (p). For every regular thin forest t (regular thin context p) there is a term $\sigma(t)$ ($\sigma(p)$) without variables such that it evaluates in the regular-thin-forest algebra over A to t (p).*

Proof. We only need prove the lemma for thin trees. Indeed, for every thin forest

$$t = t_1 + t_2 + \cdots + t_n$$

where t_i are thin trees which are generated by terms $\sigma(t_i)$, the forest t is generated by a term

$$\sigma(t) := \sigma(t_1) + \sigma(t_2) + \cdots + \sigma(t_n).$$

Also, every thin context p can be factorized as

$$p = p_0 a_1 p_1 \cdots a_n p_n$$

for some $n \geq 0$, labels on the path to the hole $a_1, \ldots, a_n \in A$ and (possibly empty) non-guarded contexts $p_0, \ldots, p_n$. Thus if $\sigma(p_i)$ generates p_i, then the context p is generated by a term

$$\sigma(p) = \sigma(p_0) a_1 \sigma(p_1) \cdots a_n \sigma(p_n).$$

Finally, every non-guarded context p can be factorized as

$$p = t_1 + \cdots + t_{j-1} + \Box + t_{j+1} + \cdots + t_n$$

where t_i are thin trees generated by terms $\sigma(t_i)$ and thus the context p is generated by a term

$$\sigma(p) = \sigma(t_1) + \cdots + \sigma(t_{j-1}) + \Box + \sigma(t_{j+1}) + \cdots + \sigma(t_n).$$

We prove the lemma by induction over the rank of a thin tree t. For the induction base observe that the empty tree is generated by the term 0. Now let t be any thin tree, and let S be its spine. If S is finite and of size $n+1$, then t can be factorized as

$$a_1 p_1 a_2 p_2 \cdots a_n p_n a_{n+1} s$$

where $a_1, \ldots, a_{n+1} \in A$ are the labels of the path S, $p_1, \ldots, p_n$ are (possibly empty) non-guarded contexts and s is a forest. (See figure 4.2 where a tree ap_1dp_2bs is depicted.) The ranks of every root of p_i and every root of s are less than $\mathrm{rank}(t)$, thus by induction assumption there are terms which generate them; let $\sigma(p_i)$ be a term generating p_i and let $\sigma(s)$ be a term generating s. Then

$$\sigma(t) := a_1 \sigma(p_1) a_2 \sigma(p_2) \cdots a_n \sigma(p_n) a_{n+1} \sigma(s).$$

If S is infinite, then t can be factorized as

$$a_1 p_1 a_2 p_2 \cdots$$

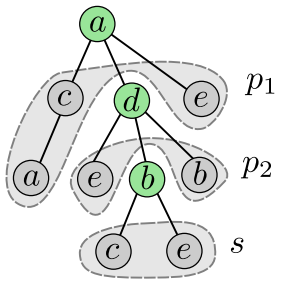


Figure 4.2

where $a_1, a_2, \ldots \in A$ are the labels of the path S and $p_1, p_2, \ldots$ are (possibly empty) non-guarded contexts. The ranks of every root of p_i are less than $\mathrm{rank}(t)$, thus by induction assumption there are terms which generate them. We concatenate them: let $\sigma(p_i)$ be a term generating p_i. Then the thin tree t is generated in $A^{\mathsf{Thin}^\triangle}$ by a term

$$\sigma(t) := \pi(a_1\sigma(p_1), a_2\sigma(p_2), \ldots).$$

If t is regular then it has a finite number of subtrees, therefore there exist two indices i, j such that $i < j$ and the subtree $a_i p_i a_{i+1} p_{i+1} \cdots$ is equal to the subtree $a_j p_j a_{j+1} p_{j+1} \cdots$. Then the regular thin tree t is generated in $A^{\mathsf{regThin}^\triangle}$ by a term

$$\sigma(t) := a_1\sigma(p_1)a_2\sigma(p_2)\cdots a_{i-1}\sigma(p_{i-1})\big(a_i\sigma(p_i)a_{i+1}\sigma(p_{i+1})\cdots a_{j-1}\sigma(p_{j-1})\big)^\infty. \qquad \square$$

Terms which generate the same object are axiom-equivalent. In the realm of finite words the associativity of concatenation operation ensures that it is of no importance in which order we perform the operations, as long as we obtain the same word. We want to prove now that the axioms of forest algebra ensures that the „generalized associativity" holds, i.e. it is not important in which order we perform the operations on forests and contexts, as long as we obtain the same objects. Since we have a fair number of operations, the proof is quite tedious. Therefore we sometimes use the axioms implicitly, especially associativity in horizontal and vertical monoids.

The idea of the proof is to show that every term in regular-thin-forest algebra and unrestricted-thin-forest algebra is axiom-equivalent to some form of a canonical term.

We say that a term is a *forest-term* if it is of sort τ_H and it generates a thin forest; it is a *context-term* if it is of sort τ_V and it generates a thin context. For simplicity of presentation, in the following lemmas we denote by t, s forest-terms rather than forests. Analogously p, q are context-terms, not contexts.

We say that a (possibly empty) non-guarded context-term p is a *brick-context-term* if it is of form $t' + \Box + t''$ for some forest-terms t', t''.

Lemma 35. *For every context-term p there is an axiom-equivalent context-term of form*

$$p' = p_0 a_1 p_1 a_2 p_2 \cdots a_n p_n$$

for some $n \geq 0$, letters $a_1, \ldots, a_n \in A$, and brick-context-terms $p_0, \ldots, p_n$. If p is a tree-context-term then $p_0 = \Box$.

Proof. The proof goes by induction on the structure of the term p. If $p = \Box$, then the term is in the desired form: we just put $n = 0$, $p_0 = \Box$. If $p = a\Box$, we put $n = 1$, $a_1 = a$,

$p_0 = p_1 = \Box$. If $p = in_l(t)$ for some forest-term t, we put $n = 0$, $p_0 = t + \Box$; similarly for $p = in_r(t)$.

Finally, if $p = qr$ for some context-terms q, r, then from the induction assumption we have $q = q_0 a_1 q_1 \cdots a_n q_n$, $r = r_0 b_1 r_1 \cdots b_m r_m$. Then $p = q_0 a_1 q_1 \cdots a_n (q_n r_0) b_1 r_1 \cdots b_m r_m$ is in the desired form thanks to the associativity of the vertical monoid and from the fact that $q_n r_0$ is a brick-context-term. □

Lemma 36. *For every forest-term t there is an axiom-equivalent forest-term of form*

$$t' = t_1 + t_2 + \cdots + t_n$$

for some $n \geq 0$ and non-empty tree-terms $t_1, \ldots, t_n$.

Proof. The proof goes by induction on the structure of the term t. If $t = t_1 + t_2$, then we simply use the induction assumption and the associativity of the horizontal monoid. If $t = ps$ for some context-term p and forest-term s, then from Lemma 35 $p = p_0 a_1 p_1 \cdots a_n p_n$ and from induction assumption $s = s_1 + s_2 + \cdots + s_m$. Thus if $n = 0$ and $p_0 = \Box$, then $t = s_1 + \cdots + s_m$. Otherwise, $p_0 = s' + \Box + s''$ for some forest-terms s', s'' and from induction assumption $s' = s'_1 + \cdots + s'_{m'}$, $s'' = s''_1 + \cdots + s''_{m''}$, thus

$$t = s'_1 + \cdots + s'_{m'} + a_1 p_1 \cdots a_n p_n s + s''_1 + \cdots + s''_{m''}.$$

Finally, if $t = (p)^\infty$ for some guarded context-term p, then from axiom (A5), $t = p(p)^\infty$, and we reduced it to the previous case. Similarly, if $t = \pi(p_1, p_2, \ldots)$ for some guarded context-terms $p_1, p_2, \ldots$, then from axiom (A5′) $t = p_1 \pi(p_2, \ldots)$ and we also reduced it to the previous case. □

It follows from the above lemma that every brick-context-term p is axiom-equivalent to a brick-context-term of form

$$p' = t_1 + \ldots + t_{j-1} + \Box + t_{j+1} + \cdots + t_n$$

for some $n \geq 0$ and non-empty tree-terms $t_1, \ldots, t_n$.

Lemma 37. *Let t be a tree-term and S be the spine of the tree generated by t. If S is finite, then there is a tree-term t' axiom-equivalent to t of form*

$$t' = a_1 p_1 \cdots a_n p_n a_{n+1} s \tag{4.2}$$

for some $n \geq 0$, letters $a_1, \ldots, a_{n+1} \in A$, brick-context-terms $p_1, \ldots, p_n$, and a forest-term s such that the rank of t is the rank of $a_{n+1}s$ and every root of the contexts generated by $p_0, \ldots, p_n$ and every root of the forest generated by s has rank strictly less than the rank of the tree generated by t.

If S is infinite then there is a tree-term t' axiom-equivalent to t of form

$$t' = \pi(a_1 p_1, a_2 p_2, \ldots) \tag{4.3}$$

for letters $a_1, a_2 \in A$ and brick-context-terms $p_1, p_2, \ldots$ such that the rank of the tree is the rank of every subtree generated by $a_i p_i a_{i+1} p_{i+1} \cdots$.

If S is infinite and the tree generated by t is regular, then there is a tree-term t' axiom-equivalent to t of form

$$t' = a_1 p_1 \cdots a_n p_n (b_1 q_1 \cdots b_m q_m)^\infty \tag{4.4}$$

for some $n \geq 0$, $m \geq 1$, letters $a_1, \ldots, a_n, b_1, \ldots, b_m \in A$, and brick-context-terms $p_1, \ldots, p_n$, $q_1, \ldots, q_m$ such that the rank of the tree is the rank of a tree generated by $(b_1 q_1 \cdots b_m q_m)^\infty$ and every root of contexts generated by $p_1, \ldots, p_n, q_1, \ldots, q_m$ has rank strictly less than the rank of the tree generated by t.

Proof. The proof goes by induction on the structure of the term t. Let $t = ps$ for some tree-context-term p and forest-term s. We use Lemmas 35 and 36 to find axiom-equivalent forms of p and s, thus without loss of generality we assume that $p = a_1 p_1 \cdots a_n p_n$ for $n \geq 1$ and brick-context-terms $p_1, \ldots, p_n$; and $s = t_1 + \cdots + t_m$ for $m \geq 0$ and tree-terms $t_1, \ldots, t_m$. We consider following cases:

1. There is an index i such that $\mathrm{rank}(t) = \mathrm{rank}(t_i)$. By the induction assumption tree t_i is axiom-equivalent to t_i' which is of form (4.2), (4.3), or (4.4). Then

$$t' = a_1 p_1 \cdots a_n \big(p_n(t_1 + \cdots + t_{i-1} + \Box + t_{i+1} + \cdots + t_m)\big) t_i'$$

   is also of the same form as t_i'.

2. There are indices i, j, j' and $j > j'$ (the case $j < j'$ is done analogously) such that $p_i = (s_1 + \cdots + s_{j'-1} + \Box + s_{j'+1} + \cdots + s_l)$ and $\mathrm{rank}(t) = \mathrm{rank}(s_j)$. By the induction assumption we have that s_j is axiom-equivalent to s_j' which is of form (4.2), (4.3), or (4.4). Then

$$t' = a_1 p_1 \cdots a_i (s_1 + \cdots + s_{j'-1} + a_{i+1} p_{i+1} \cdots a_n p_n s + \\ + s_{j'+1} + \cdots + s_{j-1} + \Box + s_{j+1} + \cdots + s_l) s_j'.$$

3. Otherwise, if i is the greatest index such that $\mathrm{rank}(t) = \mathrm{rank}(a_i p_i \cdots a_n p_n)$, then the tree is in the desired form (4.2).

Finally, if $t = (p)^\infty$ for some guarded tree-context-term p then from Lemma 35 we have $p = a_1 p_1 \cdots a_n p_n$ and $t = (a_1 p_1 \cdots a_n p_n)^\infty$ is in the desired form (4.4), which obviously can be written also as (4.3). If $t = \pi(p_1, p_2, \ldots)$, then it is easy to write it in form (4.3) using axiom (A6′). □

Let us consider a tree-term of form (4.4) and call it a *loop-term* of size (n, m). On the loop-term t of size (n, m) we can perform two operations. *Shift* of t is a loop-term of size $(n + 1, m)$:

$$a_1 p_1 \cdots a_n p_n b_1 q_1 (b_2 q_2 \ldots b_m q_m b_1 q_1)^\infty,$$

and *k-expansion* of t is a loop-term of size (n, km):

$$a_1 p_1 \cdots a_n p_n \big(\underbrace{b_1 q_1 \cdots b_m q_m \quad \cdots \quad b_1 q_1 \cdots b_m q_m}_{k \text{ times}}\big)^\infty.$$

From the axioms it is easy to see that both shift and k-expansion of t are axiom-equivalent to t.

Lemma 38. *Let t_0, t_1 be two forest-terms which generate the same forest. Then t_0, t_1 are axiom-equivalent.*

Proof. From Lemma 36 we can assume that the terms generate a tree, call it t. The proof is by induction on the rank of t. Let S be the spine of t. From Lemma 37 we get that for $i = 0, 1$, t_i is axiom-equivalent to t'_i of certain form.

If S is finite then

$$t'_i = a_{i,1}p_{i,1} \cdots a_{i,n_i}p_{i,n_i}a_{i,n_i+1}s_i$$

and since letters $a_{i,1}, \ldots, a_{i,n_i+1}$ are the labels of the spine of t, then $n_0 = n_1$ and $a_{0,j} = a_{1,j}$. Moreover, brick-context-terms $p_{0,j}, p_{1,j}$ and forest-terms s_0, s_1 generate the same objects and the trees in the roots of $p_{0,j}, p_{1,j}, s_0, s_1$ are of smaller rank than $\mathrm{rank}(t)$, therefore from the induction assumption we get that they are axiom-equivalent. Thus t'_0 is axiom-equivalent to t'_1 and hence t_0 is axiom-equivalent to t_1.

The same reasoning applies when S is infinite and t'_i are of form (4.3).

If S is infinite and t is regular, then forests-terms t_0 and t_1 are axiom-equivalent to two loop-terms of sizes (n_0, m_0) and (n_1, m_1) respectively. Without loss of generality assume that $n_0 \geq n_1$. We shift the latter term $n_0 - n_1$ times, and we m_{1-i}-expand the i-th term. Therefore we have two loop-terms of sizes (n_0, m_0m_1):

$$t'_i = a_{i,1}p_{i,1} \cdots a_{i,n_0}p_{i,n_0}(b_{i,1}q_{i,1} \cdots b_{i,m_0m_1}q_{i,m_0m_1})^\infty.$$

By the same reasoning as above we get that t_0 is axiom-equivalent to t_1. □

Lemma 39. *Let p_0, p_1 be two context-terms which generate the same context. Then p_0, p_1 are axiom-equivalent.*

Proof. From Lemma 35 we get that for $i = 0, 1$, p_i is axiom-equivalent to

$$p'_i = p_{i,0}a_{i,1}p_{i,1} \cdots a_{i,n_i}p_{i,n_i}$$

and since $a_{i,1}, \ldots, a_{i,n_i}$ are the labels on the path to the hole, then $n_0 = n_1$ and $a_{0,j} = a_{1,j}$. Moreover, brick-context-terms $p_{0,j}$ and $p_{1,j}$ generate the same objects, thus applying Lemma 38 to the trees in the roots of $p_{0,j}$ and $p_{1,j}$ we get that they are axiom-equivalent. Thus p'_0 is axiom-equivalent to p'_1 and hence p_0 is axiom-equivalent to p_1. □

4.3 Correspondence between two algebras

Algebraically, unrestricted-thin-forest algebra is a more general object, since it represents languages of all thin forests. However, one of its drawbacks is the fact that it has an operation of infinite arity and infinitely many axioms.

On the other hand, regular-thin-forest algebra is a finite object, but it represents languages of regular thin forests.

But since regular languages of thin forests are uniquely determined by the regular thin forests they contain, it is not surprising that there is a strong correspondence between regular-thin-forest algebras and unrestricted-thin-forest algebras.

Theorem 40. *Every finite regular-thin-forest algebra can be equipped, in a unique way, with a structure of an unrestricted-thin-forest algebra.*

Proof. Let $(H, V_+, V_\square)$ be a regular-thin-forest algebra. Consider a set $H_\omega \subseteq H$ which consists of all elements of form vw^∞ for $v, w \in V_+$. It is easy to see that (V_+, H_ω) is a Wilke algebra. From Theorem 17 we can, in a unique way, define the operation $\pi\colon V_+^\infty \to H_\omega$ such that (V_+, H_ω) is an ω-subsemigroup. We can naturally extend the definition of

the operation π to $(H, V_+, V_\square)$. Since the axioms of ω-subsemigroup regarding π are the same as axioms of unrestricted-thin-forest algebra, we conclude that $(H, V_+, V_\square)$ with the operation π is an unrestricted-thin-forest algebra.

The uniqueness of this extension follows from the fact that every extension must map elements of V_+ to some element from H_ω (due to the axioms and Theorem 15). Therefore different extensions would differ on (V_+, H_ω) which is impossible, since (V_+, H_ω) is unique. □

4.4 Recognizability by algebra and regularity

In this section we prove that languages which are recognizable by a finite unrestricted-thin-forest algebra are precisely regular languages of thin forests. The proof of the theorem is presented in the two following sections. In the first one we show how to calculate a finite unrestricted-thin-forest algebra which recognizes the language L, given a non-deterministic forest automaton which recognizes L. In the second one we show how to construct an MSO formula which defines L, given a finite unrestricted-thin-forest algebra which recognizes L.

Theorem 41. *A language of thin forests is recognizable by a finite unrestricted-thin-forest algebra if and only if it is regular.*

In particular, an immediate consequence of this theorem is that the language of all thin forests is regular.

4.4.1 From automaton to algebra

In this section we show how to calculate, given a non-deterministic forest automaton $\mathcal{A}$, an unrestricted-thin-forest algebra that recognizes the language recognized by $\mathcal{A}$. This algebra is called the *automaton algebra*.

Let us fix a non-deterministic forest automaton $\mathcal{A}$, with states Q, input alphabet A, and parity ranks $\{0, \ldots, k\}$. We assume that $\mathcal{A}$ recognizes a language of thin forests. Below we describe the automaton algebra (H, V), together with associated morphism $\alpha\colon A^{\mathsf{Thin}^\triangle} \to (H, V)$, which accepts the thin forests contained in the language recognized by $\mathcal{A}$.

Before describing the algebra itself, we define the morphism α. This morphism should explain what are the intended meanings of H and V.

(a) To each thin forest t, the morphism α associates a subset of Q. A state q belongs to $\alpha(t)$ if some accepting run ρ over t has value q.

(b) To each thin context p, the morphism α associates a subset of $Q \times \{0, \ldots, k\} \times Q$. A triple (q_1, i, q_2) belongs to $\alpha(p)$ if there exists a thin forest s and accepting run ρ over ps such that ρ evaluates ps to q_2, evaluates s to q_1, and the highest rank assigned to nodes that are ancestors of the hole is i (this rank is equal to 0 if p is non-guarded).

Therefore, the carriers of the horizontal and vertical monoids are subsets

$$H \subseteq P(Q), \qquad V \subseteq P(Q \times \{0, \ldots, k\} \times Q),$$

which are images of α on thin forests and thin contexts, respectively. These might be proper subsets, for instance not every subset of Q has to be an image of some t. A thin

forest belongs to L if and only if its image under α contains an accepting state, so α recognizes L.

We say that two thin forests s, t are *automaton-equivalent* if the subsets associated to these forests by the morphism α are the same. We denote it by $s \sim_{\mathcal{A}} t$. Similarly we define automaton-equivalence for thin contexts.

Lemma 42. *The relation of automaton-equivalence $\sim_{\mathcal{A}}$ is a congruence with respect to operations in the free unrestricted-thin-forest algebra.*

Proof. We show it for forest concatenation and for infinite loop operation. The proof for other operations follows the same lines.

Let t, t', s be thin forests and $t \sim_{\mathcal{A}} t'$. We must show that $t + s \sim_{\mathcal{A}} t' + s$. Suppose that $q \in \alpha(t + s)$, thus there is an accepting run ρ over $t + s$ in which the sum of states assigned to roots of t is q', the sum of states assigned to roots of s is q'', and $q = q' + q''$. The run ρ is accepting over the forest t, and since $t \sim_{\mathcal{A}} t'$, there is an accepting run ρ' over t' of value q'. Combining the run ρ' over t' with the run ρ over s we get an accepting run over $t' + s$ of value $q = q' + q''$, thus $q \in \alpha(t' + s)$.

Let p, p' be guarded thin contexts and $p \sim_{\mathcal{A}} p'$. We must show that $p^\infty \sim_{\mathcal{A}} p'^\infty$. Suppose that $q \in \alpha(p^\infty)$, thus there is an accepting run ρ over forest p^∞ of value q. For $i \geq 1$ we denote by q_{i-1} the sum of states assigned to the roots of the i-th (counting from the top) instance of the context p (of course $q = q_0$), and by k_i the highest rank assigned to nodes on the path to the i-th hole. Thus for every $i \geq 1$ we have $(q_{i-1}, k_i, q_i) \in \alpha(p)$, and therefore $(q_{i-1}, k_i, q_i) \in \alpha(p')$. That means that for every i there is an accepting run ρ_i of value q_{i-1} over $p's_i$ for some forest s_i which is evaluated to q_i. Combining these runs we get that $q \in \alpha(p'^\infty)$. □

The following lemma is a direct consequence of Lemma 42.

Lemma 43. *The function α preserves all operations of unrestricted-thin-forest algebra. In particular, its image (H, V) is a unrestricted-thin-forest algebra.*

Lemma 44. *The morphism α recognizes the same languages as the automaton $\mathcal{A}$.*

Proof. Let L be the language recognized by the automaton $\mathcal{A}$ and let q_0 be the initial state of the automaton. Let $I = \{h \in H \mid q_0 \in h\}$. From the definition we have that a forest t in in L if some accepting run over t has value q_0. It is equivalent to say that $q_0 \in \alpha(t)$, thus $\alpha(t) \in I$, and $t \in \alpha^{-1}(I)$. Therefore $L = \alpha^{-1}(I)$. □

The operations in the algebra. For completeness we show how to effectively calculate the operations of finite arity in the automaton algebra. We omit, however, the correctness proof of this construction, since it is rather tedious.

Defining the operations is straightforward, keeping in mind the intended meaning of the morphism α. We denote by $TC(v)$ a transitive closure of v in respect to the operation $\cdot$. Formally, $(p, \alpha, q) \in TC(v)$ if there exist a sequence of states $p = q_n, q_{n-1}, \ldots, q_0 = q$ and ranks $\alpha_n, \alpha_{n-1}, \ldots, \alpha_1$ such that $\alpha = \max\{\alpha_n, \ldots, \alpha_1\}$ and $(q_i, \alpha_i, q_{i-1}) \in v$ for every $1 \leq i \leq n$.

The operations are as follows:

$$\begin{array}{rcll} h+g & = & \{p+q \mid p \in h, q \in g\} & \text{for} \quad h,g \in H, \\ vw & = & \{(p,\max(i,j),q) \mid (p,i,r) \in w, (r,j,q) \in v\} & \text{for} \quad v,w \in V. \\ v^\infty & = & \{q \mid (p,i,p),(p,\cdot,q) \in TC(v), i \text{ is even}\} & \text{for} \quad v \in V_+, \\ vh & = & \{q \mid p \in h, (p,\cdot,q) \in v\} & \text{for} \quad v \in V,\ h \in H, \\ in_l(h) & = & \{(q,0,p+q) \mid p \in h\} & \text{for} \quad h \in H, \\ in_r(h) & = & \{(q,0,q+p) \mid p \in h\} & \text{for} \quad h \in H. \end{array}$$

Finally, we also define the morphism:

$$\begin{array}{rcll} \alpha(a\Box) & = & \{(q,\Omega(p),p) \mid (q,a,p) \in \Delta\} & \text{for} \quad a \in A, \\ \alpha(\Box) = \Box & = & \{(p,0,p) \mid p \in Q\}, & \\ \alpha(0) = 0 & = & \{0\} & \end{array}$$

Example 45. Consider the language L from Example 9, i.e. the language of forests with at least one infinite path (not necessarily beginning in a root) with all nodes labeled with a (call such path an a-path). It is recognized by the following unrestricted-thin-forest algebra:

$$H = \{h_a, h_b\}, \quad V = \{v_A, v_a, v_b\},$$

where $\alpha^{-1}(h_a) = L$, $\alpha^{-1}(v_A)$ are the contexts with an a-path, $\alpha^{-1}(v_a)$ are the contexts without an a-path and with a path to the hole labeled with only a's, $\alpha^{-1}(v_b)$ are the contexts without an a-path and with at least one b on the path to the hole.

The operations of finite arity in the algebra are as follows:

$$\begin{array}{ll} h_a + h_a = h_a + h_b = h_b + h_a = h_a & vv_A = v_Av = v_A \quad \text{for every } v \in V \\ h_b + h_b = h_b & v_bv_b = v_av_b = v_bv_a = v_b \\ v_Ah = h_a \quad \text{for every } h \in H & v_av_a = v_a \\ v_ah = v_bh = h \quad \text{for every } h \in H & \\ in_l(h_a) = in_r(h_a) = v_A & v_a^\infty = v_A^\infty = h_a \\ in_l(h_b) = in_r(h_b) = v_a & v_b^\infty = h_b \end{array}$$

The morphism is given by: $\alpha(0) = h_b$, $\alpha(\Box) = v_a$ and $\alpha(a) = v_a$, $\alpha(b) = v_b$.

4.4.2 From algebra to MSO formula

We show here an MSO formula for a language recognized by a given finite unrestricted-thin-forest algebra and a morphism. A similar construction can be applied to obtain a non-deterministic forest automaton, see [8]. The constructed automaton is of index $(1,3)$, i.e. it uses ranks from the set $\{1,2,3\}$.

Let L be a thin-forest language recognized by a finite unrestricted-thin-forest algebra (H,V) and a morphism $\alpha\colon A^{\mathsf{Thin}^\triangle} \to (H,V)$. That is $L = \alpha^{-1}(I)$ for some set $I \subseteq H$. We construct an MSO formula φ which defines the language L. Let $H = \{h_1,\ldots,h_{|H|}\}$ and $V_+ = \{v_1,\ldots,v_{|V_+|}\}$.

For every forest t the formula φ must check whether t belongs to L. First, the formula ensures that t is thin. It does so by constructing a branch-labeling R for t, since from Lemma 1 a forest is thin if and only if it has a branch-labeling. The following formula checks if the set R satisfies the two conditions from the definition of branch-labeling, i.e.

for every two successors of each node at most one belongs to R and on every infinite path there is a node x such that every descendant of x from this path belongs to R:

$$\begin{aligned}\varphi_R \equiv \quad & \exists R \, \big(\forall x_1 \forall x_2 \forall y \; x_1 \neq x_2 \wedge \mathsf{succ}(y, x_1) \wedge \mathsf{succ}(y, x_2) \wedge R(x_1) \to \neg R(x_2) \\ & \wedge \forall X \; (\mathrm{path}(X) \to \exists x \; X(x) \wedge \forall y \; x \leq y \wedge X(y) \to R(y))\big)\end{aligned}$$

Next, the formula guesses for every node x of the tree t the value of the morphism α on the subtree $t|_x$. That is, it guesses a set X_h for every $h \in H$ such that $x \in X_h$ if $\alpha(t|_x) = h$. Additionally, it guesses a set $X'_{h',h''}$ for every pair $h', h'' \in H$ such that $x \in X'_{h',h''}$ if $\alpha(t') = h'$ and $\alpha(t'') = h''$ where t' is a forest which contains all left-siblings of node x as roots, and t'' is a forest which contains all right-siblings of x (see figure 4.3).

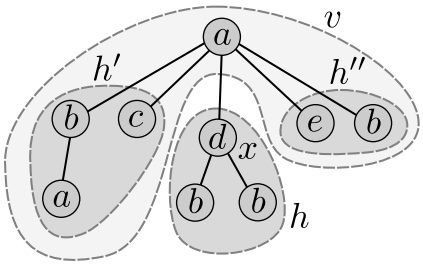


Figure 4.3

The final formula looks as follows:

$$\varphi \equiv \quad \varphi_R \quad \wedge \quad \exists X_{h_1} \cdots \exists X_{h_{|H|}} \; \exists X'_{h_1,h_1} \cdots \exists X'_{h_{|H|},h_{|H|}} \; \varphi_H(\vec{X}),$$

where $\varphi_H(\vec{X})$ is a formula which consists of $|H| + |H|^2$ free variables $\vec{X} = (X_{h_1}, \ldots, X_{h_{|H|}}, X'_{h_1,h_1}, \ldots, X'_{h_{|H|},h_{|H|}})$, which checks validity of the guessed sets. We construct the formula φ_H incrementally:

$$\varphi_H \equiv \quad \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5.$$

The following formula checks that the guess in every root corresponds to an accepting type (of course it is sufficient to check this for one root), the guessed sets are pairwise disjoint, and every node belongs to some set.

$$\begin{aligned}\varphi_2 \equiv \quad & \forall x \, \big(\mathrm{root}(x) \to \textstyle\bigvee_{h',h,h'' \in H;\; h'+h+h'' \in I} X_h(x) \wedge X'_{h',h''}(x) \\ & \wedge \textstyle\bigwedge_{h \neq h_1} X_h(x) \to \neg X_{h_1}(x) \\ & \wedge \textstyle\bigwedge_{(h',h'') \neq (h'_1,h''_1)} X'_{h',h''}(x) \to \neg X'_{h'_1,h''_1}(x) \\ & \wedge \textstyle\bigvee_{h',h,h'' \in H} X_h(x) \wedge X'_{h',h''}(x)\big)\end{aligned}$$

The next formula checks that the guessed sets are consistent with the horizontal operation of the algebra. It considers three cases: a leftmost node, a rightmost node, and a pair of immediate siblings.

$$\begin{aligned}\varphi_3 \equiv \quad & \forall x \, \big(\neg \exists y \; \mathsf{sibl}(y, x) \to \textstyle\bigvee_{h'' \in H} X'_{\alpha(0),h''}(x) \\ & \wedge \neg \exists y \; \mathsf{sibl}(x, y) \to \textstyle\bigvee_{h' \in H} X'_{h',\alpha(0)}(x) \\ & \wedge \exists y \; \mathsf{sibl}(x, y) \to \textstyle\bigvee_{h',h_x,h_y,h'' \in H} X_{h_x}(x) \wedge X'_{h',h_y+h''}(x) \wedge X_{h_y}(y) \wedge X'_{h'+h_x,h''}(y)\big)\end{aligned}$$

The next formula checks that the guessed sets are locally consistent with the vertical operation of the algebra. It considers two cases: a leaf and an inner node.

$$\begin{aligned}\varphi_4 \equiv \quad & \forall x \, \big(\neg\exists y \, \mathsf{succ}(x,y) \to X_{\alpha(t(x))}(x) \\ & \wedge \exists y \, \mathsf{succ}(x,y) \to \textstyle\bigvee_{g,h',h,h'' \in H;\ g=\alpha(t(x))(h'+h+h'')} X_g(x) \wedge X_h(y) \wedge X_{h',h''}(y)\big)\end{aligned}$$

Finally, we check that the guessed sets are consistent with the infinite vertical operation of the algebra. This is the most involved part. The following formula fixes the infinite path $\pi = x_0x_1x_2x_3\ldots$ The set X encodes a maximal infinite path and all nodes from X below x_0 form the path π.

$$\varphi_5 \equiv \quad \forall X \forall x_0 \, (\mathrm{path}(X) \wedge X(x_0)) \to \varphi_6$$

First, for every node $x_i \in \pi$ $(i \geq 0)$ the formula calculates the type v_i of the context which results from replacing in the tree $t|_{x_i}$ the node x_{i+1} by the hole. These types are represented by the sets Y_v for every $v \in V_+$. In other words $x_i \in Y_{v_i}$ if

$$v_i = \alpha(t(x_i))(h' + \Box + h'') \qquad \text{and} \qquad x_{i+1} \in X'_{h',h''}.$$

The last step is to check whether $\pi(v_0, v_1, v_2, \ldots) = \alpha(t|_{x_0})$.

Lemma 46. *For every infinite sequence $\{v_n\}_{n\geq 0} \in V_+^\infty$ there are two elements $s, e \in V_+$ and an increasing sequence $\{k_n\}_{n\geq 1}$ such that $v_0v_1\cdots v_{k_1-1} = s$ and $v_{k_i}v_{k_i+1}\cdots v_{k_{i+1}-1} = e$ for every $i \geq 1$.*

Proof. This is an immediate consequence of Lemma 16. □

From the above lemma and the axiom (A6′) if we find partition of this property, then $\pi(v_0, v_1, v_2, \ldots) = se^\infty$. The formula thus guesses elements s, e and the set S which encodes the partition of the path π in the following manner: $x_i \in S$ if and only if $i = k_j$ for some $j \geq 1$.

Moreover, the formula guesses a set Y'_v for every $v \in V_+$ which stores the type of every part in the partition. Let $x_i \in \pi$ and j is the index such $k_j \leq i < k_{j+1}$. Then $x_i \in Y'_{v'_i}$ if $v'_i = v_iv_{i+1}\cdots v_{k_{j+1}-1}$.

To check whether the guessed sets correspond to a valid partition we do as follows. First, we must ensure the local consistency: for every x_i, x_{i+1} either $i+1 = k_j$ for some j (i.e. $x_{i+1} \in S$) and then $v'_i = v_i$ and $v'_{i+1} = e$, or $i \neq k_j$ (i.e. $x_{i+1} \notin S$) and then $v'_i = v_iv'_{i+1}$. Secondly, we must check that $v'_0 = s$ and $\alpha(t|_{x_0}) = se^\infty$. Finally, we must ensure that the number of parts in the partition is infinite, thus the set S is infinite.

$$\begin{aligned}\varphi_6 \equiv \quad & \exists Y_{v_1} \cdots \exists Y_{v_{|V_+|}} \, \exists Y'_{v_1} \cdots \exists Y'_{v_{|V_+|}} \, \exists S \, \textstyle\bigvee_{s,e\in V_+} \\ & \big(\forall x \forall y \, \mathsf{succ}(x,y) \wedge X(y) \wedge x_0 \leq x \to \\ & \quad (\textstyle\bigvee_{h',h''\in H} X'_{h',h''}(y) \leftrightarrow Y_{\alpha(t(x))(h'+\Box+h'')}(x)) \\ & \quad \wedge (\textstyle\bigvee_{v\in V_+} S(y) \wedge Y_v(x) \wedge Y'_v(x) \wedge Y'_e(y) \\ & \qquad \vee \neg S(y) \wedge \textstyle\bigvee_{w,v\in V_+} Y_v(x) \wedge Y'_{vw}(x) \wedge Y'_w(y)) \\ & \quad \wedge X_{se^\infty}(x_0) \wedge Y'_s(x_0)\big) \\ & \wedge (\forall x \; S(x) \to \exists y \; X(y) \wedge x < y \wedge S(y))\end{aligned}$$

Lemma 47. *Let t be a thin forest. If the formula $\varphi_H(\vec{X})$ is satisfied on $\mathsf{struct}(t)$, then for every node $x \in \mathrm{dom}(t)$ we have that $x \in X_h$ if and only if $\alpha(t|_x) = h$.*

Proof. We do induction over the ranks of the forest t.

Fix a node x. If all its successors $x_1, \ldots, x_n$ have smaller ranks than $\mathrm{rank}(x)$, then from the induction assumption $x_i \in X_{h_i}$ if and only if $\alpha(t|_{x_i}) = h_i$. Thus from φ_3 and φ_4 we have that $x \in X_h$ for $h = t(x)(h_1 + \cdots + h_n)$.

Otherwise, there is a path π from x of nodes which have the same rank as $\mathrm{rank}(x)$. If this path is finite, we reason similarly as above.

Suppose then that the path is infinite $\pi = x_0 x_1 x_2 \ldots$ Every successor y of x_i which does not belong to π has smaller rank than $\mathrm{rank}(x)$, thus from the induction assumption $y \in X_h$ if and only if $\alpha(t|_y) = h$. Let p_i denotes the context which comes after putting hole instead of x_{i+1} in $t|_{x_i}$. It is easy to see that $x_i \in Y_{v_i}$ if and only if $\alpha(p_i) = v_i$. Therefore we must ensure that $\alpha(\pi(p_1, p_2, \ldots)) = \alpha(t|_{x_0})$. But this is the exact thing which is done in φ_5. □

Lemma 48. *The formula φ defines the same thin-forest language as the morphism α.*

Proof. If t is a forest which is not thin, then it is not in the language, and from φ_R is not in $L(\varphi)$. Let t be a thin forest of type $\alpha(t) = h$, thus t is in the language if and only if $h \in I$. From Lemma 47 the guessed types of roots of t sums to a type h, then from φ_2 we have that $t \in L(\varphi)$ if and only if $h \in I$. □

Proof of Theorem 41. It is an immediate consequence of Lemma 44 and Lemma 48. □

4.5 Deciding identities

We present here a general algorithm which shows how to decide if any given identity is true in the syntactic unrestricted-thin-forest algebra of a regular thin-forest language L. (To avoid technical difficulties we assume that identities can only use operations of finite arity.) The algorithm is exponential in the state space of a non-deterministic forest automaton recognizing L.

Firstly, however, we show the correspondence between the syntactic algebras of a regular thin-forest language and regular forests from it:

Theorem 49. *Let L be a regular thin-forest language with the syntactic unrestricted-thin-forest algebra* $\mathrm{synt}(L)$*. Let* $\mathrm{synt}(L_R)$ *be the syntactic regular-thin-forest algebra of the language L_R which contains all regular thin forests from L. Then* $\mathrm{synt}(L)$ *is isomorphic with the extension of* $\mathrm{synt}(L_R)$ *defined in Theorem 40.*

Proof. We prove that $\mathrm{synt}(L)$ is an extension of $\mathrm{synt}(L_R)$. Isomorphism follows from Theorem 40 which states that such an extension is unique.

Therefore we must show that for every two regular thin forests s and t which are equivalent under Myhill-Nerode relation $\sim_L$, they are also equivalent under $\sim_{L_R}$. From the definition $s \sim_L t$ means that for every term σ from the signature of unrestricted-thin-forest algebra we have $\sigma[x \leftarrow s] \in L$ if and only if $\sigma[x \leftarrow t] \in L$. This is equivalent to say that for every thin forest u over the alphabet $A \cup \{x\}$ we have $u[x \leftarrow s] \in L$ if and only if $u[x \leftarrow t] \in L$. Finally, this is equivalent to checking that two inverse images $(x \leftarrow s)^{-1}(L)$ and $(x \leftarrow t)^{-1}(L)$ are equal.

From Lemma 54 these images are regular languages, thus checking they equality is equivalent to testing whether they contain the same regular forests. Since regular thin forests are generated by terms of regular-thin-forest algebra, it is equivalent with stating that for every term σ from the signature of regular-thin-forest algebra $\sigma[x \leftarrow s] \in L$ if and only if $\sigma[x \leftarrow t] \in L$, thus $s \sim_{L_R} t$. □

In the previous section we show how to calculate in EXPTIME, given a non-deterministic forest automaton $\mathcal{A}$, an unrestricted-thin-forest algebra that recognizes the thin-forest language recognized by $\mathcal{A}$. Note that this algebra (treated as a regular-thin-forest algebra) recognizes the language of regular thin forests from language recognized by $\mathcal{A}$. However, it is not immediately clear that the syntactic unrestricted-thin-forest algebra and the syntactic regular-thin-forest algebra, obtained by minimizing the above algebras, will also have the same carriers. Theorem 49 proves this fact. Therefore in order to minimize an unrestricted-thin-forest algebra we can apply Moore's algorithm presented on the page 41, using only operations of finite arity from the signature.

Theorem 50. *The following problem is* EXPTIME*-complete. The input is a non-deterministic forest automaton and an identity. The question is: is the identity true in the syntactic unrestricted-thin-forest algebra of the language recognized by the automaton?*

Proof. It is not difficult to show that the problem is EXPTIME-hard, even for some fixed identities. Indeed, suppose that the identity is $h = g$. When does this identity hold in the syntactic thin-forest algebra of a forest language? There are two possibilities: either the language is empty, or full. The question "is L empty or full?" is EXPTIME-hard. This is because the question "is L full?" is EXPTIME-hard for non-deterministic tree automata, by reduction from emptiness of alternating polynomial space, and the languages used in the reduction are all non-empty.

How do prove the upper bound? The idea is as follows. We calculate in EXPTIME the automaton algebra and minimize it using Moore's algorithm. Given the resulting syntactic algebra we can test the identity by checking in polynomial time all possible valuations of variables. □

Since checking an identity in the syntactic unrestricted-thin-forest algebra of a regular thin-forest language L is equivalent to checking it in the syntactic regular-thin-forest algebra of L limited to regular forests, we will shortly write that we check identities in "the syntactic thin-forest algebra" of L.

Chapter 5

The general algebra for infinite forests

In this chapter we define an algebraic object which will allow us to deal with regular languages of infinite forests. The advantage over the approach from the previous chapter is that we are not limited to thin forests. However, the approach has a drawback: the resulting object has an infinite number of operations and axioms.

The structure we define is a *regular-infinite-forest algebra*. The idea is that the free object of this algebra is the set of all regular forests and all regular guarded contexts.

In section 5.1 we define recursion schemes, which allow us to represent the operations in the algebra. In section 5.2 we formally define regular-infinite-forest algebras.

5.1 Recursion schemes

Recall that a forest is called regular if it has finitely many distinct subtrees. In section 2.1.2 we associated with every regular forest t its component graph G_t. We noted that if we add the ordering and multiplicity to edges of G_t, then the forest t can be reconstructed from G_t. Thus such augmented graph G_t is a finite representation of t. Moreover, every augmented graph G represents some forests – we simply need to "unfold" the graph by replacing every back-edge in the graph by an edge to a new copy of the appropriate part of the graph G.

Let $X = X_H \cup X_V$ be a set of *label variables*. The set X_H represents forest-sorted variables and the set X_V represents context-sorted variables. Given an augmented graph G we can label its nodes by the elements of X, such that every node which has no outgoing edge is labeled by a forest-sorted variable, and the remaining nodes are labeled by context-sorted variables. We call such graph a *recursion scheme*.

Let ϕ be a recursion scheme and let η be a function (called a *valuation*) that maps forest-sorted label variables to regular forests and context-sorted variables to regular guarded contexts. We define $\mathrm{unfold}_\phi[\eta]$ to be a regular forest obtained by replacing the labels x with their values $\eta(x)$ and unfolding the graph.

On figure 5.1 we depicted three recursion schemes ϕ_1, ϕ_2 and ϕ_3. The recursion scheme ϕ_1 has a forest-sorted variable x and two context-sorted variables y' and y''. The tree t results from unfolding ϕ_1 by a valuation which maps a tree c and guarded contexts $a\Box$, $b\Box$ respectively to variables x, y' and y'':

$$t = \mathrm{unfold}_{\phi_1}[x \leftarrow c,\ y' \leftarrow a\Box,\ y'' \leftarrow b\Box].$$

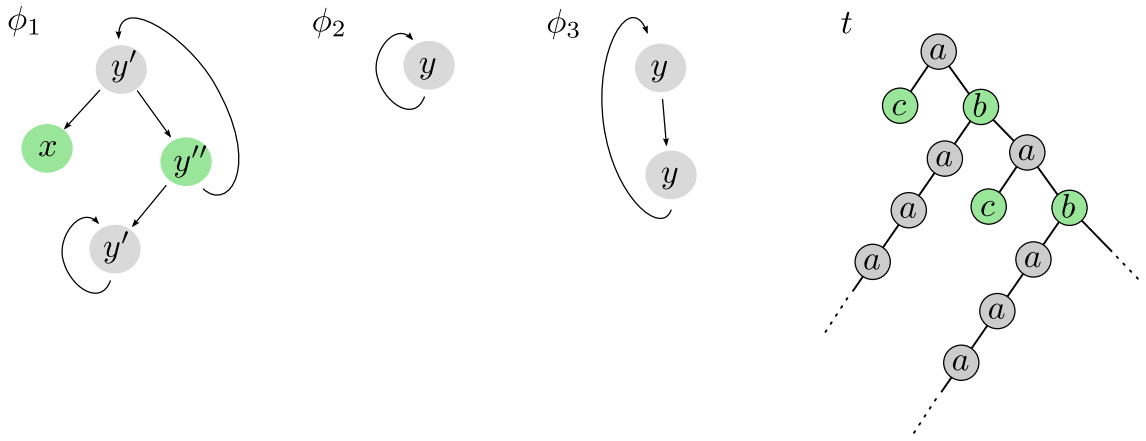


Figure 5.1

For a regular guarded context p, we can treat the operation p^∞ as a syntactic sugar for $\mathrm{unfold}_{\phi_2}[y \leftarrow p]$. Thus the tree t can also be described as

$$t = \mathrm{unfold}_{\phi_2}[y \leftarrow a(c + b(\mathrm{unfold}_{\phi_2}[y \leftarrow a\Box] + \Box))].$$

5.2 Regular-infinite-forest algebra

The operations and axioms of a *regular-infinite-forest algebra* are constructed in such a manner that the free object over an alphabet A of this algebra is the set of all regular forests over A and all regular guarded contexts over A. We denote this free algebra by

$$A^{\mathsf{reg}\triangle} = (A^{\mathsf{regFor}}, A^{\mathsf{regCon}_+}).$$

The algebra is two-sorted with a monoid H and a semigroup V_+.

Operations. There are eight *basic operations* in the algebra, as well as infinitely many *recursion operations*.

There is a constant $0 \in A^{\mathsf{regFor}}$ and seven binary operations

$$\begin{aligned}
s, t \in A^{\mathsf{regFor}} &\mapsto s + t \in A^{\mathsf{regFor}},\\
p, q \in A^{\mathsf{regCon}_+} &\mapsto pq \in A^{\mathsf{regCon}_+},\\
p \in A^{\mathsf{regCon}_+}, s \in A^{\mathsf{regFor}} &\mapsto ps \in A^{\mathsf{regFor}},\\
p \in A^{\mathsf{regCon}_+}, s \in A^{\mathsf{regFor}} &\mapsto p + s \in A^{\mathsf{regCon}_+},\\
p \in A^{\mathsf{regCon}_+}, s \in A^{\mathsf{regFor}} &\mapsto s + p \in A^{\mathsf{regCon}_+},\\
p \in A^{\mathsf{regCon}_+}, s \in A^{\mathsf{regFor}} &\mapsto p(\Box + s) \in A^{\mathsf{regCon}_+},\\
p \in A^{\mathsf{regCon}_+}, s \in A^{\mathsf{regFor}} &\mapsto p(s + \Box) \in A^{\mathsf{regCon}_+}.
\end{aligned}$$

If we had all contexts, instead of only guarded contexts, in the context sort, we could replace the last four operations by two unary operations $s \mapsto s + \Box$ and $s \mapsto \Box + s$. However, having non-guarded contexts would cause problems for the recursion operations.

For each recursion scheme ϕ which uses m forest-sorted variables $x_1, \ldots, x_m$ and n context-sorted variables $y_1, \ldots, y_n$, there is an $(m+n)$-ary recursion operation

$$\begin{aligned}
&s_1, \ldots, s_m \in A^{\mathsf{regFor}}\\
&p_1, \ldots, p_n \in A^{\mathsf{regCon}_+}
\end{aligned} \quad \mapsto \quad \mathrm{unfold}_\phi[x_1 \leftarrow s_1, \ldots, x_m \leftarrow s_m,\ y_1 \leftarrow p_1, \ldots, y_n \leftarrow p_n] \in A^{\mathsf{regFor}}.$$

It is easy to see that every regular forest t over alphabet A can be generated by using as generators single-letter contexts from $A\Box = \{a\Box \mid a \in A\}$. It is sufficient to take the augmented component graph G_t, and label each node with the appropriate variable x_a, which results in a recursion scheme ϕ_t. Then for a valuation η which maps x_a to $a\Box$ for every $a \in A$, we get $t = \mathrm{unfold}_{\phi_t}[\eta]$. It is also easy to see that every guarded regular context is also generated by $A\Box$: it suffices to construct the path to the hole in the context and generate all remaining subtrees. Therefore $A^{\mathsf{reg}\triangle}$ is generated by $A\Box$.

Axioms. We are now ready to define what regular-infinite-forest algebra is. It is a two sorted structure (H, V_+). The operations are the same as in each structure $A^{\mathsf{reg}\triangle}$: eight basic operations and infinitely many recursion operations.

We fix two disjoint countably infinite sets X_H and X_V of variables, which are intended to represent forest-sorted and context-sorted variables, respectively. We write X for the union of $X_H \cup X_V$. Abusing somewhat the notation, we write $X^{\mathsf{reg}\triangle}$ for the regular-infinite-forest algebra where the forest (context) sort contains all regular forests (regular guarded contexts) with labels from X_H in the leaves and labels from X_V in inner nodes.

The axioms of regular-infinite-forest algebra consist of all axioms which hold in $X^{\mathsf{reg}\triangle}$.

Let ϕ_2 and ϕ_3 be recursion schemes which are depicted on figure 5.1. The axioms include, for instance,

$$\mathrm{unfold}_{\phi_2}(x) = \mathrm{unfold}_{\phi_3}(x),$$

for each context-sorted variable x, since both sides evaluate in $X^{\mathsf{reg}\triangle}$ to an infinite tree $xxx\cdots$.

We note that the basic operations of regular-infinite-forest algebra, such as concatenation and composition, are mainly syntactic sugar for recursion schemes:

Lemma 51. *For every forest-valued term σ there is a recursion scheme ϕ such that $\sigma = \mathrm{unfold}_\phi$ is an axiom of regular-infinite-forest algebra.*

Theorem 52. *The algebra $A^{\mathsf{reg}\triangle}$ is a regular-infinite-forest algebra. Moreover, it is the free algebra in the class of regular-infinite-forest algebras over the generator set $A\Box$.*

Proof. It is easy to see that $A^{\mathsf{reg}\triangle}$ satisfies all axioms. We showed that $A^{\mathsf{reg}\triangle}$ is generated from $A\Box$. Moreover, any two terms which generate the same forest are immediately axiom-equivalent. Thus the premises of Lemma 18 are satisfied. □

5.2.1 Recognizing languages with regular-infinite-forest algebra

In this section we prove that regular languages of infinite forests are recognizable by a finite regular-infinite-forest algebra. We note that we do not prove the converse. We do not, however, need it for effective characterizations presented in the thesis, since the effective characterization begins with a regular language and tests properties of its syntactic algebra.

Given a non-deterministic forest automaton $\mathcal{A}$ we construct its automaton algebra (H, V) and morphism α into (H, V) in the same manner as in section 4.4.1. Thus we only present here a definition of the operation which corresponds to unfolding of recursion schemes.

Consider a recursion scheme ϕ, with variables $X_H \cup X_V$. Consider also a valuation

$$\eta\colon (X_H, X_V) \to (H, V).$$

We will show how to define the value $\mathrm{unfold}_\phi[\eta]$, which is a set of states, and also calculate which states it contains. The idea is simple: we evaluate a forest automaton over a regular forest. We take any function

$$\gamma\colon (X_H, X_V) \to A^{\mathsf{reg}\triangle} \tag{5.1}$$

such that $\alpha \circ \gamma = \eta$. Such a function exists by assumption on α being surjective. Inside the free algebra $A^{\mathsf{reg}\triangle}$ we can evaluate $\mathrm{unfold}_\phi[\gamma]$, which is a regular forest. We then define $\mathrm{unfold}_\phi[\eta]$ to be $\mathrm{unfold}_\phi[\alpha \circ \gamma]$. This definition does not depend on the choice of γ, as shown in the following lemma (we use L_q for the set of forests that can be assigned state q by the automaton $\mathcal{A}$).

Lemma 53. *Let t be a forest over $X_H \cup X_V$ and γ a valuation as in* (5.1)*. For $q \in Q$, membership $t[\gamma] \in L_q$ depends only on $\alpha \circ \gamma$, and not on γ.*

We also want to show that the value $\mathrm{unfold}_\phi[\eta]$ can be calculated based on ϕ and η, in time exponential in the size of the automaton. In other words, we want to find the states q such that $\mathrm{unfold}_\phi[\gamma] \in L_q$. Consider first the identity valuation

$$id\colon (X_H, X_V) \to (X_H, X_V).$$

Then $\mathrm{unfold}_\phi[id]$ is a regular tree over the alphabet $X_H \cup X_H$. Now

$$\mathrm{unfold}_\phi[\gamma] \in L_q \qquad \text{if and only if} \qquad \mathrm{unfold}_\phi[id] \in \gamma^{-1} L_q.$$

Therefore, we have reduced the problem to testing membership of a regular forest in a regular language (regularity of $\gamma^{-1}L_q$ is witnessed by Lemma 54). This membership can be tested in time polynomial in the size of ϕ and exponential in the state space of Q (basically, the problem boils down to solving a parity game).

Lemma 54. *For any language L recognized by $\mathcal{A}$, and any γ as in* (5.1)*, the language $\gamma^{-1}L$ is recognized by an automaton polynomial in $\mathcal{A}$.*

Suppose that parity games can be solved in polynomial time (an open problem). In this case, the operations in the automaton algebra can be calculated in polynomial time (in the size of Q).

5.2.2 Deciding identities

Deciding identities in regular-infinite-forest algebra goes in the same manner as described in section 4.5, as long as we can calculate the syntactic algebra. Thus in this section we show how, based on the automaton algebra, we can calculate the syntactic algebra and the syntactic morphism. Once we know this, the proof of the following theorem (including hardness) is the same as the proof of Theorem 50:

Theorem 55. *The following problem is* EXPTIME*-complete. The input is a non-deterministic forest automaton and an identity. The question is: is the identity true in the syntactic regular-infinite-forest algebra of the language recognized by the automaton?*

Thanks to Lemma 22, the syntactic morphism α^L of the language L is obtained from the automaton morphism α as

$$\alpha^L = \gamma \circ \alpha,$$

where γ identifies two elements g, h of the automaton algebra whenever some (equivalently, any) forests $s \in \alpha^{-1}(g)$ and $t \in \alpha^{-1}(h)$ are L-equivalent (likewise for contexts). In this section we show how to decide, in time exponential in Q, which elements of the automaton algebra are identified by γ.

Recall that we want an exponential time algorithm that decides if an identity holds in the syntactic regular-infinite-forest algebra. The algorithm simply tries all out all possible valuations into elements of the automaton algebra, and tests (in at most exponential time) if the equality holds after applying γ.

How to decide which elements of the automaton algebra are identified by γ? We only do the construction for H. For $h, g \in H$, we want to check if there exist forests s, t over A such that

$$\alpha(s) = g, \qquad \alpha(t) = h, \qquad s \sim_L t.$$

By unraveling the definition of L-equivalence, we want to know if there is a forest-valued term τ of regular-infinite-forest algebra over variables $X_H \cup X_V$ and a valuation

$$\eta \colon (X_H, X_V) \to A^{\mathsf{reg}^\triangle},$$

such that for some forest-valued variable $x \in X_H$,

$$\tau[\eta[x \leftarrow s]] \in L \qquad \text{if and only if} \qquad \tau[\eta[x \leftarrow t]] \notin L.$$

This is equivalent to asking if there is a regular forest u over the alphabet $A \cup \{x\}$ such that

$$u[x \leftarrow s] \in L \qquad \text{if and only if} \qquad u[x \leftarrow t] \notin L.$$

The above can be rephrased as asking if the two inverse images

$$(x \leftarrow s)^{-1}(L) \qquad \text{and} \qquad (x \leftarrow t)^{-1}(L),$$

which are regular languages over $A \cup \{x\}$ thanks to Lemma 54, disagree on some regular forest. Since two different regular languages must necessarily disagree on a regular forest, this boils down to checking inequality of two regular forest languages. This inequality can be decided in time exponential in Q. Note also that, thanks to Lemma 53, the automata for the inverse images do not depend on the particular choice of trees s, t, but only on their images $\alpha(s) = g, \alpha(t) = h \in H$.

Part II

Effective characterizations

Chapter 6

Simple applications

In the second part of the thesis we show some applications of the algebraic theory presented in the first part. In this chapter we start with effective characterizations of some basic properties of thin forest languages. We show how to decide whether a given regular thin-forest language is commutative (section 6.1), invariant under bisimulation (section 6.2), and open in a certain topology (section 6.3). Each decision process boils down to testing whether the syntactic forest algebra of the language satisfies a certain condition.

6.1 Commutative languages

In this section we present algebraic characterization of commutative languages. This simple example is quite instructive, since it shows that identities for infinite forests can be easily misunderstood.

The notion of a commutative language of finite forests is quite natural: it is a language closed under rearranging siblings. In the case of finite forests it is easy to show that a language is commutative if and only if its syntactic forest algebra (H, V) satisfies the identity

$$h + g = g + h \qquad \text{for } h, g \in H. \tag{6.1}$$

However, the notion of a commutative language of infinite forests is not so natural. First, let us consider the condition that

$$\text{a language is closed under rearranging siblings finitely many times.} \tag{C1}$$

This is not a very useful definition, especially if we want to find an algebraic characterization of commutativity. If identities like (6.1) are used inside more complicated terms, they could "simultaneously" rearrange infinite number of nodes. For example, the condition (C1) is satisfied by the language

$$L_1 = \text{"finitely many } a\text{-labeled nodes with a } b\text{-labeled left sibling"}.$$

However, the language L_1 does not satisfy (6.1), as witnessed by the term

$$\sigma[x] = (a\Box + x)^\infty,$$

which gives different results depending on whether x is mapped to $a + b$ or $b + a$. Indeed, $\sigma[a + b] \in L_1$, $\sigma[b + a] \notin L_1$, but if (6.1) was to be satisfied, these forests should have the same types.

We can strengthen our argument against condition (C1) – in fact there is no set of identities that captures this condition, since the class of languages satisfying the condition is not a variety of languages. Indeed for the term σ, the quotient $\sigma^{-1}L_1$ does not satisfy the notion, since $a + b \in \sigma^{-1}L_1$ and $b + a \notin \sigma^{-1}L_1$. Thus by Theorem 29 the class of algebras recognizing the languages satisfying (C1) is not a variety.

The second notion could be that

> a language is closed under arbitrary (possibly infinitely many) rearrangings of siblings. (C2)

This condition is more appealing, and it will be a base for a formal definition of commutativity we present later. However, quite surprisingly, it is not captured by the identity (6.1). Consider the language

$$L_2 = \text{“every node has 0 or 2 children and every path goes left only a finite number of times”}. \tag{6.2}$$

The language L_2 does satisfy (6.1), however it does not satisfy (C2), as it is witnessed by two forests

$$a(a + a\Box)^\infty \in L_2, \qquad a(a\Box + a)^\infty \notin L_2.$$

The problem with the above example is that we would like to be able not only to rearrange forests, but also to rearrange forests with contexts. In fact in the case of thin forest this is all we need – a slightly more general identity

$$h + v = v + h \qquad \text{for } h \in H, v \in V, \tag{6.3}$$

which is equivalent to

$$in_l(h) = in_r(h) \qquad \text{for } h \in H.$$

We note, however, that in the case of general infinite forests even this identity is not sufficient. Consider the language of full binary trees labeled with $\{0, 1\}$. We say that a path $\pi = x_1x_2x_3\ldots$ in a full binary tree is *label-consistent* if for every $x_i \in \pi$, the concatenation of labels in nodes $x_1, \ldots, x_i$ is equal to the node x_i, i.e.

$$\text{label}(x_1)\text{label}(x_2)\cdots\text{label}(x_i) = x_i.$$

For example in a full binary tree t_0 (see figure 6.1) in which every left-child is labeled with 0 and every right-child is labeled with 1, every path is label-consistent, however in a tree t_1 in which children are labeled other way round, no path (longer than one node) is label-consistent.

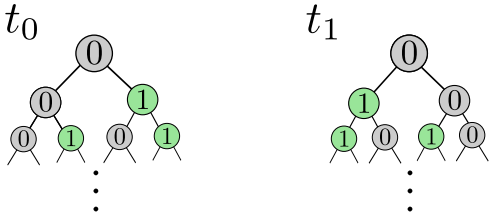


Figure 6.1

The language

$$L_3 = \text{“the number of not label-consistent paths is countable”}.$$

satisfies the identity (6.3). However, it does not satisfy (C2), since $t_0 \in L_3$, $t_1 \notin L_3$, but we can obtain t_1 from t_0 by changing the order of siblings in every node.

Since the identity (6.1), which looks like a natural candidate for defining commutativity, is in fact weaker, one can ask a question: what languages are defined by this identity. We show that this identity is equivalent to the third notion of commutativity:

a language is closed under such rearranging of siblings that
on every path of a forest siblings are rearranged only finitely many times. (C3)

We prove that (6.1) and (C3) are equivalent in the case of thin and general forests.

Definitions

We start by formalizing the definition of a commutative language. We say that two forests t_0, t_1 are *commutatively equivalent* (we denote it by $t_0 \sim_C t_1$) if there exists a bijection $f\colon \mathrm{dom}(t_0) \to \mathrm{dom}(t_1)$ such that for every $x, y \in \mathrm{dom}(t_0)$:

(a) the nodes x and $f(x)$ have the same labels,

(b) the node x is a parent of y if and only if $f(x)$ is a parent of $f(y)$.

Note that the condition (b) implies that the node x is a root if and only if $f(x)$ is a root. Observe that for any node $x \in \mathrm{dom}(t_0)$ trees $t_0|_x$ and $t_1|_{f(x)}$ are commutatively equivalent.

Let $f\colon \mathrm{dom}(t_0) \to \mathrm{dom}(t_1)$ be a bijection which shows that forests t_0 and t_1 are commutatively equivalent. It is easy to see that for every node $x \in \mathrm{dom}(t_0)$ which has n children $x_1, \ldots, x_n$, the node $f(x) \in \mathrm{dom}(t_1)$ has also exactly n children $f(x_1), \ldots, f(x_n)$. If the order $f(x_1), \ldots, f(x_n)$ is different than the order of these nodes in the forest t_1, then we say that the bijection f made a *switch* in the node x.

We say that two forests t_0, t_1 are *weakly commutatively equivalent* (we denote it by $t_0 \sim_{WC} t_1$) if they are commutatively equivalent and the bijection f satisfies additional condition:

(c) on every path $\pi \subseteq \mathrm{dom}(t_0)$ the bijection f makes a switch in a finite number of nodes from π.

A forest language L is called *(weakly) commutative* if for every two forests t_0, t_1 which are (weakly) commutatively equivalent, either both t_0, t_1 belong to L or none of them.

Our goal in this section is to prove the following theorems which effectively characterizes (weakly) commutative languages:

Theorem 56. *A regular thin-forest language L is weakly commutative if and only if its syntactic thin-forest algebra satisfies the identity* (6.1)*, i.e.*

$$h + g = g + h.$$

Theorem 57. *A regular thin-forest language L is commutative if and only if its syntactic thin-forest algebra satisfies the identity* (6.3)*, i.e.*

$$h + v = v + h.$$

The definition of commutativity could be rephrased also in the language of games. We define a game, called the *commutative game*, which is used to test the similarity of two forests in different degrees of commutativity.

Let t_0, t_1 be two forests. The commutative game over t_0 and t_1, denoted by $G(t_0, t_1)$, is played by two players: Spoiler and Duplicator. For convenience we add an auxiliary root node at the top of the forest t_i, which results in a tree t_i'. At the very beginning Duplicator chooses a subset of nodes $S \subseteq \mathrm{dom}(t_0')$, which will be fixed during the whole game. The game has three variants depending on the degree of commutativity we want to test:

(1) the set S must be finite – that corresponds to condition (C1), i.e. finite commutativity,

(2) the set S may be arbitrary – that corresponds to condition (C2).

(3) the set S must be finite on every path of the tree t_0 – that corresponds to condition (C3), i.e. weak commutativity.

The game proceeds in rounds. The state of the game is a pair (x_0, x_1), which means that there is a pebble in a node $x_i \in \mathrm{dom}(t_i')$. Initially both pebbles are in the roots of the trees t_0', t_1'. A round is played as follows. If the number of children of node x_0 is different than the number of children of node x_1, then Spoiler wins the whole game. Otherwise, Duplicator chooses a bijection f which maps the children of x_0 to the children of x_1. Moreover, if the node x_0 was not in S, then the bijection must be monotonic in respect to the children ordering (i.e. Duplicator has no choice).

Then Spoiler moves the pebble x_0 to a child x of x_0 and the pebble x_1 to a child $f(x)$. If the labels of nodes x and $f(x)$ are different – Spoiler wins. Otherwise, the round is finished and a new round is played with the state updated to $(x, f(x))$.

It is easy to see that two forests t_0, t_1 are (weakly) commutatively equivalent if Duplicator can survive for infinitely may rounds in the commutative game $G(t_0, t_1)$ of variant (3) and (2), respectively.

Proofs

Lemma 58. *Let σ be a forest-valued term with one forest-valued variable over the signature of forest algebra and let s, t be forests. If Duplicator wins the commutative game $G(s, t)$, then he also wins the commutative game $G(\sigma[x \leftarrow s], \sigma[x \leftarrow t])$.*

Proof. The strategy of Duplicator is very simple. Let $S \subseteq \mathrm{dom}(s)$ be the set which he uses in his winning strategy in the game $G(s, t)$. In the new game he chooses $\bigcup_{x \in \sigma} xS$, where the union is over all leaves with variables in σ.

As long as the children of nodes with pebbles are in σ, Duplicator choose the identity bijection. Otherwise, he uses the strategy from the game $G(s, t)$. □

Proof of Theorems 56 and 57. The "only if" part is standard. Suppose that we want to show that the identity (6.1) is satisfied. By unraveling the definition of the syntactic algebra we need to show that for every term σ and every forests t, s we have

$$\sigma[x \leftarrow t + s] \in L \quad \text{if and only if} \quad \sigma[x \leftarrow s + t] \in L.$$

It is easy to see that Duplicator wins the commutative game on forests $t+s$ and $s+t$, thus from Lemma 58 he wins the commutative game on forests $\sigma[x \leftarrow t + s]$ and $\sigma[x \leftarrow s + t]$. Therefore we get (6.1) from the fact that the language L is weakly commutative.

To show that (6.3) is satisfied, we use faithfulness of the syntactic thin-forest algebra and we show that the algebra satisfies the identities

$$\begin{aligned} h + vg &= vg + h, && \text{for } v \in V_+,\ h, g \in H, \\ (u(v+h))^\infty &= (u(h+v))^\infty, && \text{for } u, v \in V_+,\ h \in H. \end{aligned}$$

Again, this boils down to show that Duplicator wins the commutative game on forests $t + s$ and $s + t$ for every forests s, t as well as on forests $(p + t)^\infty$ and $(t + p)^\infty$ for every forest t and guarded context p.

The "if" part of the theorems follows directly from Lemmas 59 and 61, respectively. □

Lemma 59. *Suppose that identity* (6.1) *holds. If two forests* t_0, t_1 *are weakly commutatively equivalent, then* $\alpha(t_0) = \alpha(t_1)$.

Proof. We prove the lemma for trees, the generalization for forests is straightforward. Let $f\colon \mathrm{dom}(t_0) \to \mathrm{dom}(t_1)$ be a bijection which witnesses that $t_0 \sim_{WC} t_1$. Let $S \subseteq \mathrm{dom}(t_0)$ be the set of nodes in which f makes a switch.

Let $\rho\colon \mathrm{dom}(t_0) \to \mathsf{Ord}$ be a labeling by ordinal numbers such that for every nodes $x, y \in \mathrm{dom}(t_0)$, where x is a parent of y, we have $\rho(x) \geq \rho(y)$. Moreover, for every $x \in S$, $\rho(x) > \rho(y)$. It is easy to see that such labeling exists, since on every path only finitely many nodes are from S.

We prove the lemma by induction on the labels ρ. Formally we prove: if trees $t_0|_x$ and $t_1|_{f(x)}$ are weakly commutatively equivalent, then $\alpha(t_0|_x) = \alpha(t_1|_{f(x)})$.

The basis of induction is when every node of $t_0|_x$ has the same value of ρ. Then S is empty and trees $t_0|_x$ and $t_1|_{f(x)}$ must be equal, which results in $\alpha(t_0|_x) = \alpha(t_1|_{f(x)})$.

Now we do the induction step. Let r be the maximum value of ρ on the tree $t_0|_x$ and let $x_1, x_2, \ldots$ be nodes from $\mathrm{dom}(t_0)$ such that $\rho(x_i) = r$ and $x_i \in S$. Let σ_0 be the term which comes from the tree t_0 by changing every node x_i by a variable $\overline{x}_i$. Similarly, let σ_1 be the term which comes from the tree t_1 by changing every node $f(x_i)$ by a variable $\overline{x}_i$. We would like to show that

$$\alpha(\sigma_0[\overline{x}_i \leftarrow t_0|_{x_i}]) = \alpha(\sigma_1[\overline{x}_i \leftarrow t_1|_{f(x_i)}]). \tag{6.4}$$

The set S, limited to the nodes from σ_0 which are not variables, is empty, thus $\sigma_0 = \sigma_1$. Let the node x_i has n children $y_1, \ldots, y_n$. Observe that $t_0|_{x_i} \sim_{WC} t_1|_{f(x_i)}$. Then $f(x_i)$ also has n children and there is a bijection between two sets of children such that $t_0|_{y_j} \sim_{WC} t_1|_{f(y_j)}$. Since $x_i \in S$, then $\rho(y_i) < r$, and from the inductive assumption we get $\alpha(t_0|_{y_j}) = \alpha(t_1|_{f(y_j)})$. Together with (6.1) we get that $\alpha(t_0|_{x_i}) = \alpha(t_1|_{f(x_i)})$, and therefore (6.4) is satisfied. That concludes the proof. □

Note that since Lemma 59 does not assume that the forests t_0, t_1 are thin, the Theorem 56 is also true when L is a language of forests.

Lemma 60. *Let* t_0 *and* t_1 *be two thin trees which are commutatively equivalent. Then* $\mathrm{rank}(t_0) = \mathrm{rank}(t_1)$.

Proof. Let $f\colon \mathrm{dom}(t_0) \to \mathrm{dom}(t_1)$ be a bijection which witnesses that $t_0 \sim_{WC} t_1$ and let $\rho\colon \mathrm{dom}(t_0) \to \mathsf{Ord}$ be an ordinal-labeling which witnesses the rank of the tree t_0, i.e. $\rho(0) = \mathrm{rank}(t_0)$. It is easy to see that the labeling $\rho'\colon \mathrm{dom}(t_1) \to \mathsf{Ord}$ defined as

$$\rho'(x) = \rho(f^{-1}(x))$$

is an ordinal-labeling for the tree t_1. Thus $\mathrm{rank}(t_1) \leq \mathrm{rank}(t_0)$. The statement of the lemma follows from the symmetry of the argument. □

Lemma 61. *Suppose that identity* (6.3) *holds. If two thin forests* t_0, t_1 *are commutatively equivalent, then* $\alpha(t_0) = \alpha(t_1)$.

Proof. We prove the lemma for trees, the generalization for forests is straightforward. The proof is by induction on the rank of the trees.

First, observe that from Lemma 60, $\mathrm{rank}(t_0) = \mathrm{rank}(t_1)$. From the same argument, the spines of the trees have the same length. Suppose that they are infinite, the remaining case is similar.

Let $x^i_1, x^i_2, x^i_3, \ldots$ be the nodes on the spine of t_i which give us a decomposition $t_i = p^i_1 p^i_2 p^i_3 \ldots$, where p^i_j is a context with a root in x^i_j and a hole in x^i_{j+1}.

Let $f\colon \mathrm{dom}(t_0) \to \mathrm{dom}(t_1)$ be a bijection which witnesses that $t_0 \sim_{WC} t_1$. Again from Lemma 60, $f(x^0_j) = x^1_j$ for all j.

Let T^i_j be the multiset of trees rooted in the children of x^i_j, but not in x^i_{j+1}. Abusing the notation slightly, we see that mapping f gives a natural bijection between T^0_j and T^1_j, such that for any $s \in T^0_j$, the trees s and $f(s)$ are commutatively equivalent. Since trees from the sets T^i_j have ranks smaller than $\mathrm{rank}(t_0)$, we can use the induction assumption to get that $\alpha(s) = \alpha(f(s))$ for every $s \in T^0_j$. Thus from (6.3) we have $\alpha(p^0_j) = \alpha(p^1_j)$. Since

$$\alpha(t_i) = \alpha(p^i_1 p^i_2 p^i_3 \cdots) = \alpha(\pi(p^i_1, p^i_2, p^i_3, \ldots)) = \pi(\alpha(p^i_1), \alpha(p^i_2), \alpha(p^i_3), \ldots),$$

we get that $\alpha(t_0) = \alpha(t_1)$. □

6.2 Languages invariant under bisimulation

The relation of *bisimilarity* (see [24]) is the greatest relation $\sim_B$ on trees which satisfies the following. For every two trees t_0, t_1 that are bisimilar ($t_0 \sim_B t_1$), the two conditions are satisfied:

(a) they have the same label in the root,

(b) for every $i = 0, 1$ and for every $x_i \in \mathrm{dom}(t_i)$ which is a child of the root, there exists such $x_{1-i} \in \mathrm{dom}(t_{1-i})$ which is a child of the root and the trees $t_0|_{x_0}$ and $t_1|_{x_1}$ are bisimilar.

We can extend this relation to forests. Two forests t_0, t_1 are bisimilar when for every root in one of the trees there is a root in the other such that trees rooted in these nodes are bisimilar.

A regular forest language L is *invariant under bisimulation* if for every forests t_0, t_1 which are bisimilar, either both t_0, t_1 belong to L or none.

The definition of bisimilarity could also be rephrased in terms of games. Let t_0, t_1 be forests. The *bisimulation game* over t_0 and t_1, denoted by $G(t_0, t_1)$, is played by two players: Spoiler and Duplicator. The game proceeds in rounds. For convenience we add an auxiliary root node at the top of the forest t_i, which results in a tree t'_i. At the beginning of each round, the state in the game is a pair of nodes (x_0, x_1), which means that there is a pebble in a node $x_i \in \mathrm{dom}(t'_i)$. Initially both pebbles are in the roots of the trees t'_0, t'_1. A round is played as follows. First Spoiler selects one of the forests t_i ($i = 0, 1$) and moves a pebble x_i to the node x'_i which is the child of x_i. Then Duplicator moves the second pebble from the node x_{1-i} to its child x'_{1-i}. If the labels of nodes x'_0, x'_1 are different, the Spoiler wins the game. Otherwise, a new round is played with the state updated to (x'_0, x'_1).

It is easy to see that two forests t_0, t_1 are bisimilar if and only if Duplicator can survive for infinitely many rounds in the bisimulation game $G(t_0, t_1)$.

In the case of finite forests, a language L is invariant under bisimulation if and only if its syntactic algebra satisfies the identities

$$h + g = g + h \quad \text{and} \quad h + h = h \qquad \text{for } h, g \in H.$$

Since a thin-forest language invariant under bisimulation is necessarily commutative, it is clear (by the observation from the section 6.1) that the former identity should be replaced by a more general one, resulting in the following two identities:

$$h + v = v + h \qquad \text{for } v \in V,\ h \in H, \tag{6.5}$$
$$h + h = h \qquad \text{for } h \in H. \tag{6.6}$$

However, a language

$$L = \text{“finite number of infinite paths”}$$

satisfies these identities, but is not invariant under bisimulation as witnessed by two bisimilar forests

$$a^\infty \in L, \qquad a(a^\infty + a\Box)^\infty \notin L.$$

That is why we need an additional identity

$$(v^\infty + v)^\infty = v^\infty \qquad \text{for } v \in V_+. \tag{6.7}$$

The above counterexample can be generalized. On figure 6.2 there are depicted bisimilar forests

$$t = (ab)^\infty, \qquad s = \big(a((ba)^\infty + \Box)b((ab)^\infty + \Box)\big)^\infty.$$

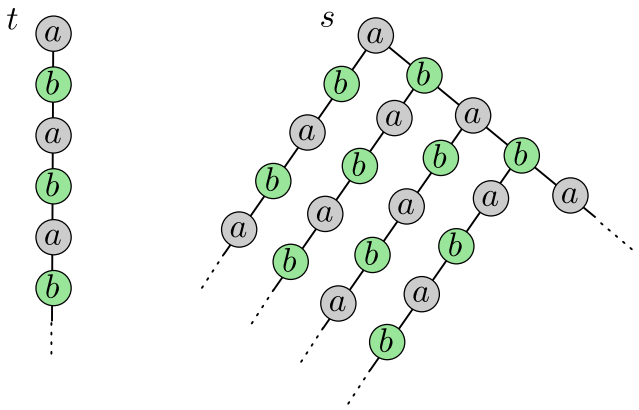


Figure 6.2

Fortunately, this and even more general examples are covered by identity (6.7):

Lemma 62. *If a thin-forest algebra (H, V) satisfies identity (6.7) then it also satisfies identity*

$$(v_1 v_2 \cdots v_n)^\infty = \big(v_1(h_1 + \Box)v_2(h_2 + \Box)\cdots v_n(h_n + \Box)\big)^\infty$$

where $n \geq 1$, $v_i \in V$, $h_i = (v_{i+1} \cdots v_n v_1 \cdots v_i)^\infty$ for $i = 1, \ldots, n$ and $v_1 v_2 \cdots v_n \in V_+$.

Proof. We denote $w_{n+1} = \Box$, $w_i = v_i(h_i + w_{i+1})$ for $i \in \{1, \ldots, n\}$. We will prove by induction that for every i we have

$$(w_{i+1}v_1 \cdots v_i)^\infty = h_i. \tag{6.8}$$

For the base of the induction ($i = n$):

$$(w_{n+1}v_1 \cdots v_n)^\infty = (v_1 \cdots v_n)^\infty = h_n.$$

For the inductive step we assume that (6.8) is true for $i+1 \leq n$ and we prove it for i:

$$\begin{aligned}
(w_{i+1}v_1 \cdots v_i)^\infty &= (v_{i+1}(h_{i+1} + w_{i+2})v_1 \cdots v_i)^\infty = \\
&= v_{i+1}\big((h_{i+1} + w_{i+2})v_1 \cdots v_{i+1}\big)^\infty = \\
&= v_{i+1}(h_{i+1} + w_{i+2}v_1 \cdots v_{i+1})^\infty \overset{(6.8)}{=} \\
&= v_{i+1}\big((w_{i+2}v_1 \cdots v_{i+1})^\infty + w_{i+2}v_1 \cdots v_{i+1}\big)^\infty \overset{(6.7)}{=} \\
&= v_{i+1}(w_{i+2}v_1 \cdots v_{i+1})^\infty \overset{(6.8)}{=} \\
&= v_{i+1}h_{i+1} = \\
&= v_{i+1}(v_{i+2} \cdots v_n v_1 \cdots v_{i+1})^\infty = \\
&= (v_{i+1} \cdots v_n v_1 \cdots v_i)^\infty = \\
&= h_i.
\end{aligned}$$

Finally, putting $i = 0$ in (6.8) we get

$$\big(v_1(h_1 + \Box)v_2(h_2 + \Box) \cdots v_n(h_n + \Box)\big)^\infty = w_1^\infty \overset{(6.8)}{=} h_0 = (v_1v_2 \cdots v_n)^\infty. \qquad \Box$$

We are ready to prove the theorem which characterizes invariance under bisimulation:

Theorem 63. *A regular thin-forest language L is invariant under bisimulation if and only if its syntactic thin-forest algebra satisfies identities* (6.5)–(6.7)*, i.e.*

$$\begin{aligned}
h + v &= v + h, \\
h + h &= h, \\
(v^\infty + v)^\infty &= v^\infty.
\end{aligned}$$

Proof. The "only if" part is standard and follows exactly the same lines as in the proof of Theorems 56 and 57.

The "if" part of the theorem is more involved. Denote by L_R the language which contains all regular thin forests from L and by L_E the language which contains all thin forests which are bisimilar to forests from L. In Lemma 66 we prove that L_R is invariant under bisimulation and in Lemma 64 that L_E is regular. Suppose by contradiction that L is not invariant under bisimulation, thus $L_E - L$ (as a difference of two regular languages) contains a regular thin forest t. Let L_t be the language of all thin forests bisimilar to t (again from Lemma 64 its a regular language). The intersection $L_t \cap L$ contains a regular thin forest s. Therefore we have two bisimilar regular thin forests t, s such that $s \in L_R$ and $t \notin L_R$, which contradicts that L_R is invariant under bisimulation. $\Box$

Lemma 64. *Let L be a regular language of forests. The language which contains of all forests which are bisimilar to some forest from L is regular.*

For a proof of this lemma we need to define another type of automaton recognizing regular languages of forests. A (non-deterministic parity) *hedge automaton* (see also [19]) over an alphabet A is given by a set of states Q, a transition relation $\Delta \subseteq Q^* \times A \times Q$, regular language $L_I \in Q^*$, and a parity condition $\Omega\colon Q \to \{0, \ldots, k\}$.

A run of this automaton over a forest t is a labeling $\rho\colon \mathrm{dom}(t) \to Q$ such that for every node x with children $x_1, \ldots, x_n$ the word $\rho(x_1)\rho(x_2)\cdots\rho(x_n) \in Q^*$ belongs to a regular language L_q such that $(L_q, t(x), \rho(x)) \in \Delta$. A run is accepting if for every (infinite) path $\pi \in \mathrm{dom}(t)$ the maximum rank among states from $\mathrm{Inf}(\pi)$ is even. A forest is accepted if it has an accepting run in which states assigned to roots of the forest form a word from the language L_I. The set of forests accepted by a hedge automaton is called the language recognized by the automaton.

Theorem 65. *Hedge automata recognize regular languages of forests.*

Proof. First we show how, given a forest automaton $\mathcal{A} = (A, Q, \Delta, q_i, \Omega)$ recognizing a forest language L, to construct a hedge automaton $\mathcal{B} = (A, Q, \Delta', L_I, \Omega)$ also recognizing L. For every $a \in A$ and $q \in Q$ we add a transition $(L_{a,q}, a, q)$ to Δ' such that

$$L_{a,q} = \{q_1 \cdots q_n \in Q^* \mid (q_1 + \cdots + q_n, a, q) \in \Delta\}.$$

Similarly, $L_I = \{q_1 \cdots q_n \in Q^* \mid q_1 + \cdots + q_n = q_I\}$. The languages $L_{a,q}$ and L_I are regular and it is easy to see that the language recognized by $\mathcal{B}$ is L.

To show that hedge automata recognize only regular languages of forests, we show that language recognized by a hedge automaton $\mathcal{A} = (A, Q, \Delta, L_I, Q)$ is regular. Let $\Delta = \{(L_{a,q}, a, q) \mid a \in A, q \in Q\}$. Let $\alpha_{a,q}\colon Q \to M_{a,q}$ be a morphism into a monoid $M_{a,q}$ which recognizes the language $L_{a,q}$, i.e. $L_{a,q} = \alpha_{a,q}^{-1}(J_{a,q})$ for some set $J_{a,q} \subseteq M_{a,q}$. Similarly let $\alpha_I\colon Q \to M_I$ be a morphism which recognizes L_I (using $J_I \subseteq M_I$). Let M be a cartesian product of all monoids $M_{a,q}$ and M_I, and $\alpha\colon Q \to M$ be a cartesian product of morphisms.

Let $\mathcal{B}_m = (A, M, \Delta', m, \Omega')$ be a forest automaton. For every $a \in A$, $q \in Q$ and every element $m \in M$ such that m projected on the $M_{a,q}$ coordinate is contained in $J_{a,q}$ we add a transition $(m, a, \alpha(q))$ to Δ'. Finally we put $\Omega'(\alpha(q)) = \Omega(q)$ (a run cannot use other elements of M, so we do not have to define Ω' on them). Let N be the set of such elements of M that projected on the M_I coordinate are contained in J_I. The above construction is developed in such a way that the language recognized by $\mathcal{A}$ is a union of languages recognized by forest automata $\mathcal{B}_m$ for $m \in N$, therefore it is regular. □

Proof of Lemma 64. Let $\mathcal{A} = (A, Q, \Delta, L_I, \Omega)$ be a hedge automaton recognizing L. We construct hedge automaton $\mathcal{B} = (A, Q, \Delta', L'_I, \Omega)$ which recognizes the language of all forests which are bisimilar to some forest from L. For every transition $(L_{a,q}, a, q) \in \Delta$ we have a transition $(L'_{a,q}, a, q) \in \Delta'$ such that

$$L'_{a,q} = \{w \mid \text{exists } v \in L_{a,q} \text{ such that } v \text{ and } w \text{ have the same set of letters}\}.$$

We define L'_I similarly. Since $L'_{a,q}$ and L'_I are regular (as languages of words over Q) thus $\mathcal{B}$ is a hedge automaton. It is easy to see that it recognizes the desired language. □

Lemma 66. *Suppose that identities* (6.5)–(6.7) *hold. If two regular thin forests t_0, t_1 are bisimilar, then $\alpha(t_0) = \alpha(t_1)$.*

The rest of this section is devoted to the proof of Lemma 66.

For a node $x \in \mathrm{dom}(t_i)$ we denote by $B_i(x) = [t_i|_x]_{\sim_B}$ the equivalence class under the bisimilarity relation of the subtree rooted in x. We say that a node $x \in \mathrm{dom}(t_i)$ is

bisimilar to a node $y \in \mathrm{dom}(t_j)$ if the subtrees $t_i|_x$ and $t_j|_y$ are bisimilar, or equivalently $B_i(x) = B_j(y)$. We extend the definition to paths: when $\pi = x_1 x_2 \ldots \in \mathrm{dom}(t_i)^\infty$ is a path, then $B_i(\pi) = B_i(x_1)B_i(x_2)\ldots$

First assume that t_0 and t_1 are trees and their root components are connected. Recall that since the trees are thin and regular, from Lemma 4 their root components correspond to cycles in the component graphs. Thus there is a unique path which starts in the root of t_0 (respectively t_1) and goes only through nodes of the root component.

Lemma 67. *Let t_0, t_1 be two regular thin trees which are bisimilar and their root components are connected. Let π_i be a path which starts in the root of t_i and goes only through nodes of the root component. Then $B_0(\pi_0) = B_1(\pi_1)$.*

Proof. We say that a node $x \in \mathrm{dom}(t_i)$ is *good* if it satisfies the following condition:

$$\text{there are paths } \pi \text{ and } \pi' \text{ from } x \text{ such that } B_i(\pi) = B_0(\pi_0) \text{ and } B_i(\pi') = B_1(\pi_1). \quad (6.9)$$

It is clear that if a node x is good, then every node bisimilar to x is also good. Let ℓ be the least common multiple of the sizes of the root components. We build a sequence (not necessarily a path) $y_0, y_1, \ldots$ of good nodes from t_0 such that y_{i+1} will be in deeper component than y_i.

Let y_0 be the root of t_0. It is easy to see that it is good. Indeed, the path π through the root component is equal to π_0. Moreover, since t_0 is bisimilar to t_1 and from the root of t_1 goes the path π_1, then from y_0 must go a path π' such that $B_0(\pi') = B_1(\pi_1)$.

Suppose that we constructed good nodes $y_0, \ldots, y_j$. Let π and π' be two paths from y_j, which satisfy the condition (6.9). If the component of y_j is connected and both π and π' lie inside this component, then obviously $\pi = \pi'$, and thus $B_0(\pi_0) = B_0(\pi) = B_0(\pi') = B_1(\pi_1)$ and we are done. Otherwise, one of these paths leaves the component. Without loss of generality suppose that it is π and that it leaves the component after k nodes. Decompose it as $\pi = \pi_A \pi_B$ where π_A is of length $\ell \cdot k$. Then the first node of π_B is bisimilar to y_j and thus it is good. We denote this node by y_{j+1}.

Observe that since the number of components in t_0 is finite and y_{j+1} is in the deeper component that y_j, thus at some point this construction will lead to the desired conclusion that $B_0(\pi_0) = B_1(\pi_1)$. □

We say that a tree t is *trimmed* if it does not contain a node $x \in \mathrm{dom}(t)$ from the root component such that it has two bisimilar children x', x'' such that x' is from the root component and x'' lies outside the root component. We denote by $\mathrm{trim}(t)$ a trimmed version of t in which for every such node x'' the subtree $t|_{x''}$ is removed. It is easy to see that t is bisimilar to $\mathrm{trim}(t)$.

Lemma 68. *Suppose that identities (6.5)–(6.7) hold. Let t, s be two regular thin trees which are bisimilar, their root components are connected, s is trimmed and Lemma 66 holds for trees which have smaller number of components than t and s. Then $\alpha(t) = \alpha(s)$.*

Proof. Let ℓ be the least common multiple of the sizes of the root components of t and s. Since $(v)^\infty = (v^k)^\infty$ for every k, we can assume without loss of generality that the sizes of the root components are equal to ℓ.

From Lemma 67 we have that $B_0(\pi_0) = B_1(\pi_1)$. Thus subsequent labels from the root components are the same. Denote

$$t = (a_1 p_1 a_2 p_2 \ldots a_\ell p_\ell)^\infty, \qquad s = (a_1 p'_1 a_2 p'_2 \ldots a_\ell p'_\ell)^\infty$$

for some letters $a_1, \ldots, a_\ell \in A$ and non-guarded contexts $p_1, \ldots, p_\ell$, $p'_1, \ldots, p'_\ell$. We also see that if we denote for $i = 1, \ldots, \ell$

$$t_i = p_i a_{i+1} p_{i+1} \ldots a_\ell p_\ell t, \qquad s_i = p'_i a_{i+1} p'_{i+1} \ldots a_\ell p'_\ell s,$$

then t_i is bisimilar to s_i.

Now fix $i \in \{1, \ldots, \ell\}$. Let T_i be the set of trees which appear in roots of context p_i. Similarly define T'_i as the set of trees which appear in roots of context p'_i. Since $t_i \sim_B s_i$, then for every tree $t \in T_i$ there must be a tree rooted in a root of s_i which is bisimilar to t. Let R_i be as follows:

$$R_i = \{t \in T_i \mid \text{there is a tree } t' \in T'_i \text{ such that } t \sim_B t'\}.$$

In other words, $T_i - R_i$ contains those trees which must be bisimilar to $a_{i+1}s_{i+1}$. Moreover, let

$$g_i = \sum_{t \in R_i} \alpha(t), \qquad h_i = \sum_{t \in T_i - R_i} \alpha(t).$$

Symmetrically, we define R'_i, g'_i and h'_i. Using these notations and identity (6.5), we can express the types of contexts p_i and p'_i as

$$\alpha(p_i) = g_i + h_i + \Box, \qquad \alpha(p'_i) = g'_i + h'_i + \Box.$$

Since all trees from R_i (respectively R'_i) have a smaller number of components than the tree t (respectively s), we can apply the assumption and identities (6.5), (6.6) to get $g_i = g'_i$.

Moreover, since s is trimmed, then $T'_i = R'_i$. Otherwise, $s'' \in T'_i - R'_i$ would be bisimilar to $a_{i+1}t_{i+1}$, thus it would be bisimilar to $a_{i+1}s_{i+1}$ – a contradiction. Hence $h'_i = 0$.

Finally, every tree $t'' \in T_i - R_i$ has a smaller number of components than t and it is bisimilar to $a_{i+1}s_{i+1}$, thus from the assumption and (6.6) we get $h_i = \alpha(a_{i+1}s_{i+1})$.

In conclusion, we can express the types of p_i and p'_i as

$$\alpha(p_i) = g_i + h_i + \Box, \qquad \alpha(p'_i) = g_i + \Box.$$

If we denote for $i = 1, \ldots, \ell$

$$v_i = \alpha(a_i)(g_i + \Box),$$

then the types of trees t and s can be expressed as

$$\alpha(t) = \big(v_1(h_1 + \Box)v_2(h_2 + \Box) \cdots v_\ell(h_\ell + \Box)\big)^\infty, \qquad \alpha(s) = (v_1 v_2 \cdots v_\ell)^\infty$$

with

$$h_i = (v_{i+1} \ldots v_\ell v_1 \ldots v_i)^\infty.$$

Applying Lemma 62 we obtain $\alpha(t) = \alpha(s)$. □

Proof of Lemma 66. (1) First, we show how to reduce the problem to trees: we assume that lemma is true for trees and we show that it is also true for the case when at least one of t_0, t_1 has more than one root.

Let T_i be the set of types in the roots of t_i, i.e. $T_i = \{\alpha(t_i|_x) \mid x \text{ is a root of } t_i\}$. From the definition of bisimilarity for $i = 0, 1$ for every root $x_i \in \mathrm{dom}(t_i)$ exists a root

$x_{1-i} \in \mathrm{dom}(t_{1-i})$ such that $t_0|_{x_0}$ is bisimilar to $t_1|_{x_1}$. From the assumption we get $\alpha(t_0|_{x_0}) = \alpha(t_1|_{x_1})$. Therefore $T_0 = T_1$. Thus from (6.5) and (6.6) we have $\alpha(t_0) = \alpha(t_1)$.

(2) Let n_i be the number of components in the tree t_i for $i = 0, 1$. The proof is by induction over the sum $n_0 + n_1$.

First, assume that the root component of t_0 is a singleton component. If $n_0 = 1$ (i.e. t_0 has only one node), then from bisimilarity also $n_1 = 1$ and the trees are equal (thus they have the same type). This is also the base of the induction.

Thus assume that $n_0, n_1 > 1$. Assume that the root component of t_1 is also a singleton component. From bisimilarity the trees have the same label a in their roots, so the tree t_i can be written as $t_i = as_i$ for a non-empty forest s_i, which has $n_i - 1$ components. Now s_0 and s_1 are bisimilar and every tree rooted in a root of s_i has at most $n_i - 1$ components. Thus repeating reasoning from the case (1) and using induction assumption on these smaller trees gives us that $\alpha(s_0) = \alpha(s_1)$, and therefore $\alpha(t_0) = \alpha(t_1)$.

Now, assume that the root component of t_1 is a connected component. Again we write $t_i = as_i$ for a non-empty forest s_i, but this time forest s_1 can have n_1 components. But since forest s_0 has at most $n_0 - 1$ components (so we can use the induction assumption), we can repeat the above reasoning.

Finally, assume that t_0 and t_1 are trees and their root components are connected. From Lemma 68 we get that $\alpha(t_1) = \alpha(\mathrm{trim}(t_1))$. Since t_1 is bisimilar to $\mathrm{trim}(t_1)$, then from the transitivity of bisimilarity relation we get that t_0 is bisimilar to $\mathrm{trim}(t_1)$ and thus from Lemma 68, $\alpha(t_0) = \alpha(\mathrm{trim}(t_1))$. Therefore $\alpha(t_0) = \alpha(t_1)$. □

We note that Lemma 66 can be also stated in slightly more general setting. Observe that the identities (6.5)–(6.7) can be treated as additional axioms (since they do not use the idempotent power $v \mapsto v^\omega$). Then the lemma can be rephrased as "if two regular forests are bisimilar, then they are axiom-equivalent" when by "axioms" we mean (A1)–(A6) and identities (6.5)–(6.7).

6.3 Open languages

The set of forests can be interpreted as a metric space. We define the distance between two forests to be 2^{-n} if n is the smallest depth where the two forests differ. Since we have a topology, we can ask what are the open sets in this topology. The ball of center in forest t and radius 2^{-n} can be represented by a finite prefix of t, where all the nodes on depth n and above are fixed. An open set is a union of balls.

We say that a forest language L is *open* if it is an open set in this topology.

For example the language

$$L = \text{"there is a node with label } a\text{"}$$

is open, since for any forest $t \in L$ there is a node on some depth n with label a. Changing any nodes on depth greater than n results in a tree which still belongs to L.

Also for any finite forest t the singleton language $L_t = \{t\}$ is open. Indeed, if n is the height of forest t, then there is no other forest in the ball of center in t and radius $2^{-(n+1)}$.

The same topology can be imposed upon the set of thin forests. Note that the set of all forests is a complete metric space, but the set of all thin forests is not, since a Cauchy sequence of a-labeled full finite binary trees of increasing heights has a limit (a

full binary tree) which is not thin. Note however, that both spaces are not compact, since for a sequence of forests $t_n = \underbrace{a + \cdots + a}_{n \text{ times}}$ no subsequence has a limit.

We are interested in checking whether a given regular forest language L is open. This problem is decidable, a solution (which is folklore to the best of our knowledge) is based on the observation that the topological closure of a forest language L is the set

$$\text{closure}(L) = \{t \mid \text{every finite prefix of } t \text{ can be extended to some forest in } L\}.$$

A language L is open if and only its complement $\bar{L}$ satisfies $\bar{L} = \text{closure}(\bar{L})$. Automata for both $\bar{L}$ and closure$(\bar{L})$ can be computed based on the automaton for L, and then one can test two regular languages for equality.

The same argument works for regular languages of thin forests, since the language of all thin forests A^{ThinFor} is regular, the complement of L in the realm of thin forests is just $\bar{L} \cap A^{\mathsf{ThinFor}}$ and the topological closure of L in the realm of thin forests is just closure$(L) \cap A^{\mathsf{ThinFor}}$.

However, we present here an algebraic approach to testing whether a regular thin-forest language is open. The main advantage of our characterization is that it can be phrased as a single identity.

Theorem 69. *A regular thin-forest language L is open if and only if its syntactic morphism $\alpha\colon A^{\mathsf{Thin}^\triangle} \to (H, V)$ satisfies the following condition for every $v \in V_+$, $h \in H$:*

$$\textit{if} \quad v^\infty \in \alpha(L) \quad \textit{then} \quad v^\omega h \in \alpha(L). \tag{6.10}$$

One can extend the theory of ordered algebras (see [26]) to forest algebras. Then the above condition could be simply stated as:

$$v^\infty \geq v^\omega h \qquad \text{for any } v \in V_+, h \in H. \tag{6.11}$$

The notion of open sets is also applicable to the case of infinite words. It is interesting to note that the identity (6.11) also characterizes the open languages of infinite words. (see [26]).

Let X be an infinite set of variable names. A *thin multicontext* over A is a thin forest over $A \cup X$ in which every variable $x \in X$ appears in a leaf. The number of variables appearing in a thin multicontext is not restricted. An *open thin multicontext* over A is a thin context p such that $p0$ is a thin multicontext. For an (open) thin multicontext p we denote by $\text{vars}(p) \subseteq X$ the set of variables appearing in p.

Let p be a (open) thin multicontext and $\zeta\colon \text{vars}(p) \to A^{\mathsf{ThinFor}}$ be a mapping which assigns thin forests to variables appearing in p. We denote by $p[\zeta]$ a forest which results from replacing every variable x in p by the forest $\zeta(x)$. We say then that p is a *prefix* of t.

By pA^{ThinFor} we denote a language of all thin forests such that p is their prefix.

For example on figure 6.3 is depicted an open thin multicontext p with a set of variables $\text{vars}(p) = \{x_1, x_2\}$. For any forest s, ps is a thin multicontext such that

$$psA^{\mathsf{ThinFor}} = \{a(b + t_1 + a(s + t_2)) \mid t_1, t_2 \in A^{\mathsf{ThinFor}}\}.$$

A regular language of thin forests L is open if for every forest $t \in L$ there is a finite prefix of t such that changing nodes outside of the prefix does not affect membership in L. Thus L is open if there exists (possibly infinite) set P of finite thin multicontexts such that

$$L = \bigcup_{p \in P} pA^{\mathsf{ThinFor}}.$$

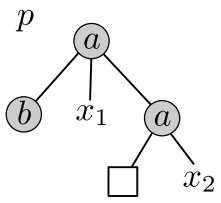


Figure 6.3

Proof of Theorem 69. It is obvious that if L is open, then it must satisfy (6.10). Indeed, let $v \in V_+$ and let $t \in L$ be a thin forest of type v^∞. Thus $t = r^\infty$ for some context $r \in \alpha^{-1}(v)$. Since L is open, then there exists a prefix p (of depth n) of the forest t such that $pA^{\mathsf{ThinFor}} \subseteq L$. Thus $r^k A^{\mathsf{ThinFor}} \subseteq L$ for any $k \geq n$. Taking $k = \omega n$ we get $r^k s = r^{\omega n} s \in L$ for every thin forest s. Let $h = \alpha(s)$, then $v^{\omega n} h = v^\omega h \in \alpha(L)$.

The converse implication follows from Lemma 72, which is formulated at the end of the section. □

Let p, p' be two thin multicontexts. We say that the thin multicontext p could be *immediately reduced* to p' if

$$p = qr^\infty \qquad \text{and} \qquad p' = qr^\omega x_\star$$

for an open thin multicontext q, a thin context r, and a variable $x_\star \notin \mathrm{vars}(q)$. We denote it by $p \to p'$ (see figure 6.4). We say that p could be *reduced* to p' if there is a sequence $p = p_0, p_1, p_2, \ldots, p_{n-1}, p_n = p'$ of thin multicontexts such that p_i could be immediately reduced to p_{i+1}. We denote it by $p \to^* p'$.

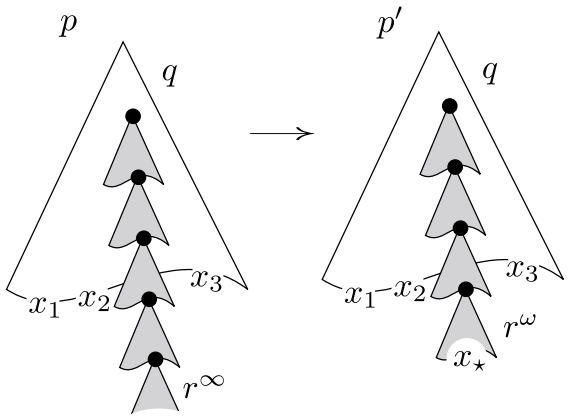


Figure 6.4

Lemma 70. *Let L be a regular thin-forest language which satisfies* (6.10) *and let p, p' be two thin multicontexts. If $pA^{\mathsf{ThinFor}} \subseteq L$ and $p \to p'$, then $p'A^{\mathsf{ThinFor}} \subseteq L$.*

Proof. Let $p = qr^\infty$ and $p' = qr^\omega x_\star$ where q is an open thin multicontext, r is a thin context, and $x_\star$ is a variable not in $\mathrm{vars}(q)$. Observe that all the variables appearing in p are from q. Similarly, all the variables appearing in p' (except for the additional variable $x_\star$) are also in q.

Let t' be any forest from $p'A^{\mathsf{ThinFor}}$ and $\zeta\colon \mathrm{vars}(p') \to A^{\mathsf{ThinFor}}$ satisfies $p'[\zeta] = t'$. Applying ζ to thin multicontext p, we get a forest $t = p[\zeta] \in pA^{\mathsf{ThinFor}}$. Since $pA^{\mathsf{ThinFor}} \subseteq L$ we get that forest $t = q[\zeta]r^\infty$ is in L. From (6.10) the tree $t' = q[\zeta]r^\omega\zeta(x_\star)$ is also in L. Therefore $p'A^{\mathsf{ThinFor}} \subseteq L$. □

Let P' and P'' be two sets of prefixes:

$$P' = \{\text{finite thin multicontext } p \mid pA^{\mathsf{ThinFor}} \subseteq L\},$$
$$P'' = \{\text{finite thin multicontext } p \mid t \to^* p \text{ for some } t \in L\}.$$

Lemma 71. *Let L be a regular thin-forest language. For every regular thin forest $t \in L$ there is a finite thin multicontext $p \in P''$ which is a prefix of t.*

Proof. Let $t \in L$. We prove the lemma by induction over the number of components in the forest t, i.e. we prove the statement: if s is a subforest of t, then there is a finite thin multicontext p such that $s \to^* p$.

We can assume that t is a tree, otherwise we just concatenate prefixes for the trees which are rooted in the roots of forest t.

If the root component of the tree t is a singleton component, then $t = as$ for some $a \in A$ and a forest s. From the inductive assumption there is a finite thin multicontext p such that $s \to^* p$. Clearly, the thin multicontext ap satisfies $t \to^* ap$.

Let the root component of the tree t be connected. Thus $t = (a_1q_1 \cdots a_nq_n)^\infty$ for some labels $a_1, \ldots, a_n \in A$ and non-guarded contexts $q_1, \ldots, q_n$. It is easy to see that for a variable $x_\star$

$$t \to (a_1q_1 \cdots a_nq_n)^\omega x_\star.$$

Let $q_i = t'_i + \Box + t''_i$ for some forests t'_i, t''_i. From the inductive assumption there are finite thin multicontexts p'_i, p''_i such that $t'_i \to^* p'_i$ and $t''_i \to^* p''_i$. Without loss of generality we can assume that these thin multicontexts have different variables appearing in them, i.e. set $\{x_\star\}$ as well as sets $\mathrm{vars}(p'_i), \mathrm{vars}(p''_i)$ for $i = 1, \ldots, n$ are pairwise mutually disjoint. Applying these thin multicontexts ω times we get

$$t \to^* \big(a_1(p'_1 + \Box + p''_1) \cdots a_n(p'_n + \Box + p''_n)\big)^\omega x_\star.$$ □

Lemma 72. *Let L be a regular thin-forest language which satisfies condition* (6.10)*. Then $L = P'A^{\mathsf{ThinFor}}$.*

Proof. Clearly $P'A^{\mathsf{ThinFor}} \subseteq L$. From Lemma 71 we have $L \subseteq P''A^{\mathsf{ThinFor}}$ when restricted to regular forests. Finally, from Lemma 70 we have $P'' \subseteq P'$, since for every $t \in L$ we have $tA^{\mathsf{ThinFor}} = \{t\} \subseteq L$. Therefore

$$L \subseteq P''A^{\mathsf{ThinFor}} \subseteq P'A^{\mathsf{ThinFor}} \subseteq L$$

when restricted to regular forests. Since both L and $P'A^{\mathsf{ThinFor}}$ are regular languages and they contain the same regular forests, they are equal. □

Chapter 7

The temporal logic EF

In this chapter we present another application of our algebraic approach developed in the first part of the thesis. Namely, we present an effective characterization of the infinite forest and tree languages that can be defined in the temporal logic EF.

An important advantage of using algebra is that an effective characterization could be stated in terms of identities that must be satisfied in an algebra. It was shown [14] that in the case of finite forest algebra, the effective characterization of the logic EF has this property. The answer we provide in the case of infinite trees is mixed. In section 7.2 we prove a theorem which states that a language of infinite forests is definable in the logic EF if and only if its syntactic forest algebra satisfies two conditions. One of them is an identity, however, for the other condition, invariance under EF-bisimulation, we were unable to come up with an identity (or a finite set of identities). The situation is different, however, in the case of thin-forest languages. In section 7.3 we prove that in this case when checking whether a thin-forest language is definable in the logic EF, the invariance under EF-bisimulation can be replaced by testing several identities.

7.1 Logic EF

We begin by defining the logic EF and giving some examples. The *logic EF* is a simple temporal logic which uses only one operator EF, whose name stands for "Exists Finally". The formula $\mathsf{EF}\varphi$ means that "φ holds in a proper subtree" (see figure 7.1).

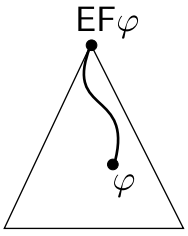


Figure 7.1

The natural objects which are described by the logic EF are trees, that is why we begin with defining the semantics of EF formulas on trees. However, since our goal is to give an algebraic characterization of languages defined by EF formulas, we will also have to deal with EF formulas on forests.

Fix an alphabet A. The following conditions describes the set of valid *tree EF formulas*:

(a) Every letter $a \in A$ is a tree EF formula, which is true in trees with root label a.

(b) Tree EF formulas admit Boolean operations, including negation.

(c) If φ is a tree EF formula, then $\mathsf{EF}\varphi$ is a tree EF formula, which is true in trees that have a proper subtree where φ is true.

A tree t satisfies a tree EF formula φ if φ holds in the root of the tree t. We say that a language of trees is defined by a tree EF formula φ if it contains precisely those trees which satisfy φ.

A number of operators can be introduced as a syntactic sugar:

$$\mathsf{EF}^*\varphi \equiv \varphi \vee \mathsf{EF}\varphi, \qquad \mathsf{AG}\varphi \equiv \neg\mathsf{EF}\neg\varphi, \qquad \mathsf{AG}^*\varphi \equiv \varphi \wedge \mathsf{AG}\varphi.$$

The formula $\mathsf{EF}^*\varphi$ is a weak version which means that "φ holds in a non-necessarily proper subtree" and the formula $\mathsf{AG}\varphi$ means that "φ holds in every proper subtree", that is "Always Globally".

We give here some examples of tree EF formulas:

$$\mathsf{EF}a, \quad \mathsf{EF}^*(a \wedge \neg\mathsf{EF}\top), \quad \mathsf{AG}^*(a \to \mathsf{EF}b \wedge b \to \mathsf{EF}a).$$

The first formula means that there is a non-root node with label a in a tree, the second one states that there is a leaf with label a, and the third one means that every node labeled with a has a b-labeled descendant and vice versa, in particular there is at least one infinite path in a tree.

It is important to dispel possible wrong intuitions about the logic EF, one of them is the intuition that EF talks about paths in trees. Consider the formula

$$\varphi = \mathsf{EF}(b \wedge \mathsf{EF}c \wedge \mathsf{EF}d)$$

which states that there is a non-root node with label b which have two descendants labeled with c and d respectively. On figure 7.2 tree t_1 clearly satisfies formula φ, but tree t_2 does not, still both trees have the same set of paths. On the other hand tree t_3 has more paths than t_1, nevertheless it also satisfies the formula. To say more, there is no tree EF formula which differentiates these trees, i.e. every formula which is satisfied in one of t_1 or t_3 is satisfied in the other (we prove it in the next section).

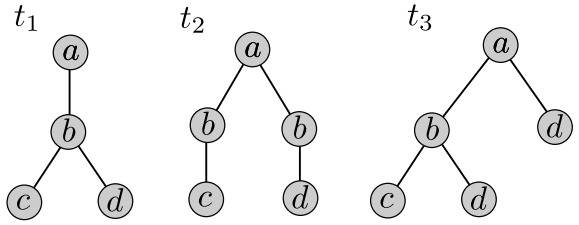


Figure 7.2

Since we deal with forest languages in this thesis, we will also want to define forest languages using the logic. It is clear which forests should satisfy the formula EF^*a (some node in the forest has label a, possibly a root). It is less clear which forests should satisfy $\mathsf{EF}a$ (only non-root nodes are considered?), and even less clear which forests should satisfy a (which root node should have label a?). Therefore we will only use Boolean combinations

of formulas of the first kind to describe forests. That is, a *forest EF formula* is a Boolean combination of formulas of the form $\mathsf{EF}^*\varphi$.

Note that given a forest EF formula φ which defines a forest language L we can treat φ as a tree EF formula. It defines a language of trees which contains precisely those trees which are in the forest language L.

7.1.1 EF game and EF-bisimulation

We define a game, called the *EF game*, which is used to test the similarity of two forests under forest formulas of EF. The name EF comes from the logic, but also, conveniently, is an abbreviation for Ehrenfeucht–Fraïssé (see [35]).

Let t_0, t_1 be forests. The EF game over t_0 and t_1, denoted by $G(t_0, t_1)$, is played by two players: Spoiler and Duplicator. The game proceeds in rounds. At the beginning of each round, the state in the game is a pair of forests, (t_0, t_1). A round is played as follows. First Spoiler selects one of the forests t_i $(i = 0, 1)$ and its subtree s_i, possibly a root subtree. Then Duplicator selects a subtree s_{1-i} in the other tree t_{1-i}. If the root labels a_0, a_1 of s_0, s_1 are different, then Spoiler wins the whole game. Otherwise the round is finished, and a new round is played with the state updated to (r_0, r_1) where the forest r_i is obtained from the tree s_i by removing the root node, i.e $s_i = a_i r_i$.

We can also describe $G(t_0, t_1)$ in terms of a pebble game. For convenience, we add an auxiliary root node at the top of the forest t_i, which results in a tree t'_i. In one of the nodes x_i of the tree t'_i there is a pebble. The state of the game is a pair (x_0, x_1). Initially both pebbles are in the roots of the trees t'_0, t'_1. A round is played as follows. First Spoiler selects one of the pebbles at node x_i and moves it to the node x'_i which is a descendant of x_i, i.e. $x'_i > x_i$. Then Duplicator moves the second pebble from the node x_{1-i} to its descendant x'_{1-i}. If the labels of nodes x'_0, x'_1 are different, the Spoiler wins the game. Otherwise, a new round is played with the state updated to (x'_0, x'_1).

By $d_{\mathsf{EF}}(\varphi)$ we denote the EF-*nesting depth* of EF formula φ. It is defined inductively:

$$\begin{aligned} d_{\mathsf{EF}}(a) &= 0, & d_{\mathsf{EF}}(\varphi \wedge \psi) &= \max(d_{\mathsf{EF}}(\varphi), d_{\mathsf{EF}}(\psi)), \\ d_{\mathsf{EF}}(\neg\varphi) &= d_{\mathsf{EF}}(\varphi), & d_{\mathsf{EF}}(\mathsf{EF}\varphi) &= 1 + d_{\mathsf{EF}}(\varphi). \end{aligned}$$

The EF game is designed to reflect the structure of forest EF formulas, so the following fact, which is proved by induction on m, should not come as a surprise.

Lemma 73. *Spoiler wins the m-round EF game on forests t_0 and t_1 if and only if there is a forest EF formula of EF-nesting depth m that is true in t_0 but not t_1.*

We will also be interested in the case when the game is played for an infinite number of rounds. If Duplicator can survive for infinitely many rounds in the EF game $G(t_0, t_1)$, then we say that the forests t_0 and t_1 are *EF-bisimilar.* A forest language L is called *invariant under EF-bisimulation* if it is impossible to find two forests $t_0 \in L$ and $t_1 \notin L$ that are EF-bisimilar. In other words L is a union of equivalence classes of the EF-bisimulation relation.

Lemma 74. *Two trees t_0, t_1 are EF-bisimilar if and only if for every $i = 0, 1$ and every proper subtree s_i of t_i there is a proper subtree s_{1-i} of t_{1-i} such that s_0 and s_1 are EF-bisimilar.*

It follows from Lemma 73 that if a forest language L is defined by a forest EF formula, then it is invariant under EF-bisimulation. We show that languages which share this property form a variety of languages.

Lemma 75. *The class $\mathcal{L}$ of forest languages which are invariant under EF-bisimulation is a variety of languages.*

Proof. (1) Since a language invariant under EF-bisimulation is a union of equivalence classes of EF-bisimulation relation, it is clear that $\mathcal{L}$ is closed under Boolean operations.

(2) Let $L \in \mathcal{L}(A)$ and α be a morphism between the free forest algebras over alphabets B and A respectively. We will show that $L_B = \alpha^{-1}(L) \in \mathcal{L}(B)$.

Let t_0, t_1 be two forests from L_B which are EF-bisimilar. Let s_0, s_1 be their images under α, i.e. $s_i = \alpha(t_i)$. We will show that s_0 and s_1 are EF-bisimilar, which will conclude the proof, since that means that $s_0 \in L$ if and only if $s_1 \in L$, thus $t_0 \in L_B$ if and only if $t_1 \in L_B$.

To show that s_0, s_1 are EF-bisimilar, we will show a strategy for Duplicator in EF game $G(s_0, s_1)$. For every node $x \in t_i$ we define a set of nodes $\mathrm{Im}_i(x)$, which consist of these nodes from s_i, which are the image of x under α. The strategy of Duplicator will ensure that after every Duplicator's move the pebbles x_0, x_1, y_0, y_1 (placed on t_0, t_1, s_0, s_1 respectively) satisfy:

(a) x_0 and x_1 have the same label, thus $\mathrm{Im}_0(x_0)$ has the same structure as $\mathrm{Im}_1(x_1)$,

(b) $y_i \in \mathrm{Im}_i(x_i)$, for $i = 0, 1$,

(c) the position of y_0 within $\mathrm{Im}_0(x_0)$ is the same as position of y_1 within $\mathrm{Im}_1(x_1)$, in particular the labels of y_0 and y_1 are the same.

Let Spoiler move a pebble from y_i to y'_i. If both positions y_i, y'_i happen to be in the same context $\mathrm{Im}_i(x_i)$, then Duplicator updates the position of y_{1-i} to such node in $\mathrm{Im}_{1-i}(x_{1-i})$ that the condition (c) is satisfied. We make no moves in the game $G(t_0, t_1)$.

Assume otherwise that $y'_i \in \mathrm{Im}_i(x'_i)$ for some node x'_i different from x_i. It is clear that $x'_i > x_i$ and we perform a Spoiler's move in $G(t_0, t_1)$ by moving a pebble from x_i to x'_i. Since t_0 and t_1 are EF-bisimilar, there is a Duplicator's reply which moves a pebble from x_{1-i} to some x'_{1-i} such that the condition (a) is satisfied. The strategy for Duplicator in $G(s_0, s_1)$ is to move from y_{1-i} to a node $y'_{1-i} \in \mathrm{Im}_{1-i}(x'_{1-i})$ such that the condition (c) is satisfied.

(3) Let $L \in \mathcal{L}(A)$ and σ be a forest-valued term with one forest-valued variable over the signature of forest-algebra over A. We will show that $\sigma^{-1}L \in \mathcal{L}(A)$.

Let t_0, t_1 be two forests from $\sigma^{-1}L$ which are EF-bisimilar. Denote s_0, s_1 the trees from L such that $s_i = \sigma[x \leftarrow t_i]$. We will show that s_0 and s_1 are EF-bisimilar, which will conclude the proof.

The strategy for Duplicator in $G(s_0, s_1)$ is as follows. As long as Spoiler restricts his moves to the nodes from $\sigma[x]$, Duplicator simply copies Spoiler's moves. When Spoiler moves a pebble into one of the trees t_i which are substituted for a variable x, Duplicator also moves into the corresponding tree t_{1-i} and now follows the Duplicator's strategy in the game $G(t_0, t_1)$. □

Let L be a forest language and $\alpha_L \colon A^{\mathsf{reg}\triangle} \to (H, V)$ be its syntactic morphism. From Lemma 75 it follows that L is invariant under EF-bisimulation if and only if every language $\alpha_L^{-1}(h)$ for $h \in H$ is invariant under EF-bisimulation.

7.1.2 EF for finite forests

For finite forests, the temporal logic EF was characterized in [14]. The result was:

Theorem 76. *Let L be a regular language of finite forests. There is a forest formula of EF that is equivalent, over finite forests, to L if and only if the syntactic forest algebra (H, V) of L satisfies the identities*

$$vh = vh + h \qquad \text{for } h \in H, v \in V, \tag{7.1}$$

$$h + g = g + h \qquad \text{for } g, h \in H. \tag{7.2}$$

A corollary to the above theorem is that, for finite forests, definability in EF is equivalent to invariance under EF-bisimulation. This is because two finite forests that are EF-bisimilar can be rewritten into each other using the identities (7.1) and (7.2).

Our goal is to prove the usefulness of the algebras from chapters 4 and 5 by extending Theorem 76 to infinite forests. It turns out that this extension cannot be straightforward since, somewhat surprisingly, EF formulas over infinite forests can express nontrivial properties even for unary alphabets, which was not the case with finite forests.

Let us consider the alphabet $A = \{a\}$. Using the EF game it is easy to see that every finite forest is EF-bisimilar to a single path whose length is equal to the length of the longest path of the forest, which reduces all the questions to the finite word case.

In the case of infinite words the case is even simpler: there is only one infinite word over the unary alphabet.

We show now that it is not easy to find such a set T of infinite forests such that every forest is EF-bisimilar to some forest from T, but no two forests from T are EF-bisimilar to each other.

For a forest t we can partition its set of nodes as $\mathrm{dom}(t) = S_\infty \cup S_0 \cup S_1 \cup S_2 \cup \cdots$, where S_n contains nodes x such that $t|_x$ has height n. The formula $\psi_{\geq n} \equiv \mathsf{EF}^n a$ is true in nodes x for which $t|_x$ has height at least n. Similarly $\psi_{=n} \equiv \psi_{\geq n} \wedge \neg\psi_{\geq n+1}$ is true in nodes which are roots of trees of (finite) height equal to n.

Now consider a formula

$$\varphi_n \;\equiv\; \neg\mathsf{EF}^*\psi_{=n} \;\wedge\; \mathsf{EF}^*\psi_{\geq n+1} \;\wedge\; \mathsf{AG}^*\big(\psi_{\geq n+1} \;\rightarrow\; \mathsf{EF}\psi_{\geq n+1} \wedge \mathsf{EF}\psi_{=n-1}\big).$$

It states that in a forest we have $S_n = \emptyset$, thus $\emptyset = S_{n+1} = S_{n+2} = \ldots$ However, a forest satisfying the formula must contain a node which is a root of tree of height greater than n, thus it must be a node from S_∞. Moreover, under every node from S_∞ there is another node from S_∞, and also a node from S_{n-1}.

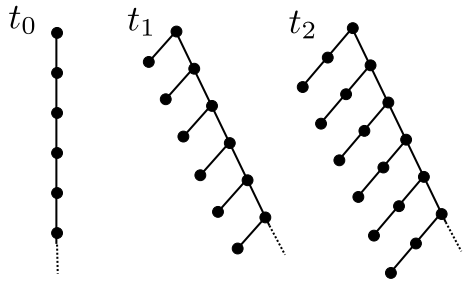


Figure 7.3

On the figure 7.3 there are three trees $t_0 = (a\Box)^\infty$, and $t_n = a(a^n + a\Box)^\infty$ for $n = 1, 2$. They belong to languages defined by φ_0, φ_1 and φ_2, respectively. Note, however, that for $i < j$ the trees t_i and t_j are not EF-bisimilar, since the strategy for Spoiler is as follows:

> Spoiler moves in t_j to a node from S_{j-1}. If Duplicator moves in t_i to a node from S_∞, then Spoiler till the end of the game stays in t_i in the set S_∞; the game will end in j moves. On the other hand, if Duplicator replies in t_i with a move to a node from S_k for $k \leq i - 1$, then Spoiler till the end of the game moves in t_j; the game will end in k moves.

Note that not every infinite forest is EF-bisimilar to t_n for some n. Examples of such forests would be $a(t_1 + t_2)$ or $a(t_1 + a(t_2 + a\Box))^\infty$, or more general ones, such as $a(t_{i_1} + t_{i_2} + \ldots + t_{i_k})$ for natural numbers $i_1, \ldots, i_k$.

As we mentioned before, thanks to Lemma 73, we know that any language defined by a forest EF formula is invariant under EF-bisimulation. Unlike for finite forests, the converse does not hold. Consider, for instance the language "all finite forests". This language is invariant under EF-bisimulation, but it cannot be defined using a forest EF formula.

Indeed, consider two families of trees

$$t_n = a^{n+1}b, \qquad s_n = a(a^n b + a\Box)^\infty,$$

which are depicted on figure 7.4.

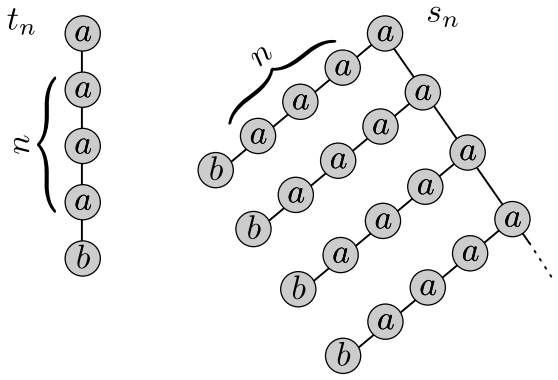


Figure 7.4

Let φ be any forest EF formula of EF-nesting depth n. It is easy to see that Duplicator wins the n-round EF game on trees t_n and s_n. His strategy is as follows: anytime Spoiler moves to a node labeled with b, Duplicator moves to a b-labeled node in another tree. As long as Spoiler makes a move to an a-labeled node which is in distance k from the nearest b-labeled node, Duplicator moves to a node in another tree which is in distance k from the nearest b-labeled node, or if this was not possible, he moves one node down.

Therefore any forest language which is defined by a forest EF formula of EF-nesting depth n contains both trees t_n and s_n or none of them. Since the tree t_n is finite and the tree s_n is infinite, we conclude that the language "all finite forests" is not definable in the logic EF.

In Lemma 92, we will show a weaker form of the converse. Namely, for any fixed regular tree t, the set of trees that are EF-bisimilar to t can be defined in EF.

7.2 Characterization of EF for infinite forests

The fact that trees from figure 7.4 are indistinguishable by EF formulas can be rephrased in the language of algebra (and in a somewhat more general setting): for sufficiently large n the identity

$$v^n h = (v + v^n h)^\infty$$

is satisfied for all $h \in H$, $v \in V_+$ from the syntactic algebra (H, V) of any language definable by a forest EF formula. Clearly we can replace the statement "for sufficiently large n" with the idempotent power.

It turns out that with invariance under EF-bisimulation this identity fully characterizes the logic EF on infinite forests.

Theorem 77. *A regular forest (thin-forest) language L can be defined by a forest formula of EF if and only if*

(a) it is invariant under EF-bisimulation,

(b) the syntactic forest (thin-forest) algebra (H, V) of regular part of L satisfies the identity

$$v^\omega h = (v + v^\omega h)^\infty \qquad \text{for all } h \in H, v \in V_+. \tag{7.3}$$

The above formulation of Theorem 77 is a little bit informal, in fact this theorem comes in four variants, depending on whether L is a language of: (G) forests, (GR) regular forests, (T) thin forests, (TR) regular thin forests. In the first two variants (H, V) is a regular-infinite-forest algebra and in the remaining variants (H, V) is a regular-thin-forest algebra.

First we show that the cases (G) and (T) can be reduced to (GR) and (TR) respectively, and then in the remaining part of this chapter we will assume that the language L contains only regular forests. We show how to reduce (G) to (GR), reduction of (T) to (TR) is analogous.

Take a regular language L of forests, which satisfies the conditions of Theorem 77. Therefore the language L_R which contains regular forests from L also satisfies these conditions, and since the theorem is true in case (GR), L_R is definable by an EF forest formula φ (when restricted to regular forests). The formula φ, when not restricted to regular forests, must define the language L, since should it define another regular language L', there would be a regular forest in $(L' - L) \cup (L - L')$.

One can ask why we use the invariance under EF-bisimulation in the statement of Theorem 77 and not a set of identities like in Theorem 76. The fact is that in the case of infinite forests we do not know how to express invariance under EF-bisimulation in terms of only identities. We managed to do so in the case of thin forests, and we present the result later. At the moment we show why the identities (7.1) and (7.2) are insufficient in the case of infinite forests.

Consider the language $L =$ "there is at least one (not necessarily maximal) infinite path with all nodes labeled with a", which we encountered before, in Example 45. The syntactic algebra of this language satisfies the identities (7.1) and (7.2), since (under the notation from the example) we have:

$$\begin{aligned} v_A h = h_a = h_a + h = v_A h + h, \\ vh = h = h + h = vh + h \qquad \text{for } v = v_a, v_b. \end{aligned}$$

However, the language L is not invariant under EF-bisimulation. Indeed, consider the trees

$$t = (ab)^\infty, \qquad s = \big(a(\Box + (ab)^\infty)\big)^\infty,$$

depicted on figure 7.5.

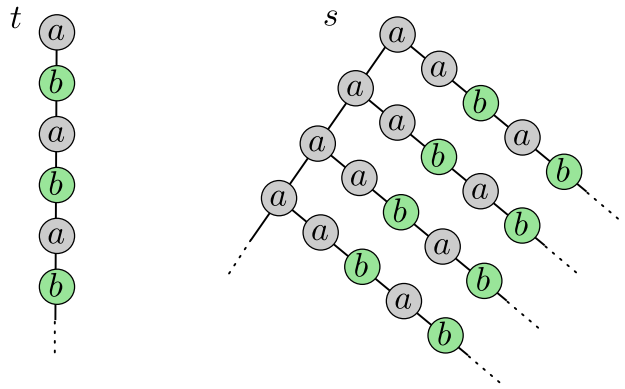


Figure 7.5

These trees are clearly EF-bisimilar, since their every node has both a-labeled descendant and b-labeled one. However, only tree s belongs to the language L, therefore the language is not invariant under EF-bisimulation. We can add a new identity which is necessary for the invariance:

$$(vw)^\infty = \big(v(\Box + (vw)^\infty)\big)^\infty \qquad \text{for } v \in V_+, w \in V.$$

Note that in presence of (7.1) we have $\big(v(\Box + (vw)^\infty)\big)^\infty = (v + (vw)^\infty)^\infty$, thus the above identity can be replaced by a slightly more elegant

$$(vw)^\infty = (v + (vw)^\infty)^\infty. \tag{7.4}$$

Indeed, for $h = \big(v(\Box + (vw)^\infty)\big)^\infty$ we have

$$\begin{aligned} h = v(\Box + (vw)^\infty)h = v(h + \Box)(vw)^\infty \overset{(7.1)}{=} v(h + \Box)(vw)^\infty + (vw)^\infty = \\ = h + (vw)^\infty = (\Box + (vw)^\infty)h = ((\Box + (vw)^\infty)v)^\infty = (v + (vw)^\infty)^\infty. \end{aligned}$$

But again, this identity is not yet sufficient, as witnessed by the following language over $A = \{a, b, c\}$:

$L =$ "every infinite path which ultimately does not branch is labeled with word $A^*(abc)^\infty$".

Clearly L satisfies identities (7.1), (7.2) and (7.4), since they preserve the set of infinite paths which ultimately do not branch. However, it distinguishes two EF-bisimilar trees

$$(abc)^\infty \in L, \qquad (acb)^\infty \notin L.$$

We will prove that, in fact, adding the fourth identity

$$(vwu)^\infty = (vuw)^\infty \qquad \text{for } v \in V_+, u, w \in V \tag{7.5}$$

creates a set of identities which characterizes EF-bisimulation on regular thin trees. We note that these four identities constitute a minimal (in respect to inclusion) set of identities

characterizing EF-bisimulation. Indeed, we already seen it in case of (7.5). Moreover, all identities except (7.4) preserve finiteness of the number of paths in a forest, all identities except (7.2) preserve the left-most root of a forest, and finally, all identities except (7.1) preserve the set of roots of a forest.

We show how to test invariance under EF-bisimulation in the next section. The rest of this section is devoted to proving Theorem 77.

7.2.1 The conditions are necessary

In this section we show that the two conditions in Theorem 77 are necessary for definability in EF. Fix a forest language L that is definable by a forest formula of EF. Let the syntactic morphism of the regular part of L be

$$\alpha_L : A^{\mathsf{reg}\triangle} \to (H, V).$$

Invariance under EF-bisimulation follows immediately from Lemma 73.

We now turn to condition (7.3). Let n be the idempotent power ω of V_+. That is, for every context type $v \in V_+$, the types v^n and $v^n \cdot v^n$ are equal. Such a power exists, since V_+ is a finite monoid. Let $i \cdot n$ be a multiple of n that is larger than the EF-nesting depth of the forest formula defining L. We will show that

$$v^{i\cdot n}h = (v + v^{i\cdot n}h)^\infty \qquad \text{for any } v \in V_+ \text{ and } h \in H.$$

This establishes (7.3), since $v^{i\cdot n} = v^n = v^\omega$. By unraveling the definition of the syntactic regular-infinite-forest algebra, and using Fact 51, we need to show that for every guarded context p, every forest t, every recursion scheme ϕ, every valuation η of the variables in ϕ with elements of $A^{\mathsf{reg}\triangle}$, and every forest-sorted z variable in ϕ, we have

$$\mathrm{unfold}_\phi[\eta[z \leftarrow p^{i\cdot n}t]] \in L \qquad \text{if and only if} \qquad \mathrm{unfold}_\phi[\eta[z \leftarrow (p + p^{i\cdot n}t)^\infty]] \in L. \quad (7.6)$$

We will show a stronger result, namely that Duplicator wins the $(i \cdot n)$-round EF game on the two forests

$$\mathrm{unfold}_\phi[\eta[z \leftarrow p^{i\cdot n}t]] \qquad \text{and} \qquad \mathrm{unfold}_\phi[\eta[z \leftarrow (p + p^{i\cdot n}t)^\infty]].$$

To get rid of the unfolding above, we will use the following fact, which basically says that winning the m-round game behaves like a congruence with respect to unfolding:

Lemma 78. *Let ϕ be a recursion scheme with a single forest-sorted variable, and let s, t be forests. If Duplicator wins the m-round EF game on s and t, then for any ϕ and η he also wins the m-round EF game on*

$$\mathrm{unfold}_\phi[\eta[z \leftarrow s]] \qquad \textit{and} \qquad \mathrm{unfold}_\phi[\eta[z \leftarrow t]].$$

Proof. The proof is similar to the one in Lemma 58. As long as pebbles are in common part of ϕ, Duplicator copies Spoiler's moves. Otherwise, he uses his strategy of game on s and t. □

Thanks to this fact, instead of (7.6), we only have to prove that Duplicator wins the $(i \cdot n)$-round EF game on the two forests

$$p^{i\cdot n}t \qquad \text{and} \qquad (p + p^{i\cdot n}t)^\infty.$$

It is not difficult to generalize the Duplicator's strategy presented on the page 94 to a winning strategy in the above game.

7.2.2 The conditions are sufficient

We now show the more difficult part of Theorem 77. Let $\alpha\colon A^{\mathsf{reg}\triangle} \to (H, V)$ be the syntactic morphism of the regular part of the language L. Suppose that the target algebra satisfies condition (7.3) and that the morphism is invariant under EF-bisimulation. From Lemma 75 languages which are invariant under EF-bisimulation form a variety of languages, thus from Theorem 29 every language recognized by the target algebra is invariant under EF-bisimulation (thus any two EF-bisimilar forests have the same image under the morphism α). For a forest t, we use the name *type of* h for the value $\alpha(h)$. We will show that for any $h \in H$, the language L_h of forests of type h is definable by a forest formula of EF. This shows that the two conditions in Theorem 77 are sufficient for definability in EF.

The proof is by induction on the size of H. The induction base, when h is the only element in H, is trivial. In this case all forests have the same type h, and the appropriate formula is *true*.

We now proceed to the induction step. We say that an element $h \in H$ is *reachable* from $g \in H$ if there is some $v \in V$ with $h = vg$.

Lemma 79. *The reachability relation is transitive and antisymmetric.*

Proof. Note that reachability might not be reflexive, since V is not necessarily a monoid. Transitivity is obvious. For antisymmetry, we first prove that invariance under EF-bisimulation implies property (7.1). Indeed, since α is surjective, there must be some context p with $\alpha(p) = v$ and some forest t with $\alpha(t) = t$. Since the forests $pt + t$ and pt are EF-bisimilar, their types must be equal, and hence (7.1) holds.

Suppose that g is reachable from h, and vice versa. To prove antisymmetry, we need to show that $g = h$. By assumption there are $v, w \in V$ with $g = wh$ and $h = vg$. Then we have

$$g = wh = wvg \overset{(7.1)}{=} wvg + vg = g + vg \overset{(7.1)}{=} vg = h. \qquad \square$$

We say that an element $h \in H$ is *minimal* if it is reachable from all $g \in H$. (The name minimal, instead of maximal, is traditional in algebra. The idea is that the set of elements reachable from h is minimal.) There is at least one minimal element, since for every $v \in V$ an element $v(h_1 + \cdots + h_n)$ is reachable from each h_i. Since reachability is antisymmetric, this minimal element is unique, and we denote it using the symbol $\bot$. An element $h \neq \bot$ is called *subminimal* if the elements reachable from h are a subset of $\{h, \bot\}$.

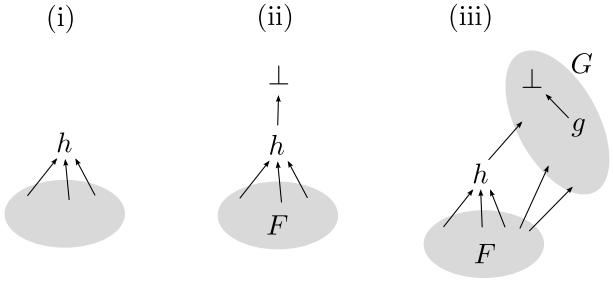


Figure 7.6

Recall that our goal is to give, for any $h \in H$, a formula of EF that defines the set L_h of forests with type h. Fix then some $h \in H$. We partition the set of types as follows:

$H = F \cup \{h\} \cup G$, where F be the set of all elements different from h, from which h is reachable and G be the set of all elements different from h, from which h is not reachable. We consider three cases, shown on figure 7.6, depending on the cardinality of the set G:

(i) $|G| = 0$, which means that h is minimal,

(ii) $|G| = 1$, which means that h is subminimal, and there is exactly one subminimal element,

(iii) $|G| \geq 2$, which means that h is subminimal and there are at least two subminimal elements or h is neither minimal nor subminimal.

Note that the first case follows from the remaining two, since a forest has type $\bot$ if and only if it has none of the other types. Therefore the formula for $h = \bot$ is obtained by negating the disjunction of the formulas for the other types. Now we treat the remaining two cases.

Case (iii).

For this case, we use a standard method. The idea is to squash all elements from G into a single element. Since G has at least two elements, we can use the induction assumption on a smaller algebra. The tricky part is showing that this squashing is a congruence, which requires dealing with the unfoldings.

Let us define the following equivalence relation on H: $g \sim f$ holds if either $g = f$ or both g, f belong to G. In other words $\sim$ identifies all types in G. We extend this relation to V by setting $v \sim w$ if either $v = w$ or both v, w belong to the set

$$W = \{w \in V \mid wH \subseteq G\}.$$

The relation $\sim$ is a two-sorted equivalence relation, i.e. a pair of equivalence relations, one for each of the two sorts H and V of regular-infinite-forest algebra. The following lemma shows that it is actually a congruence, i.e. for every operation of the algebra, the equivalence class of the result is uniquely determined by the equivalence classes of the arguments.

Lemma 80. *The relation $\sim$ is a congruence.*

Proof. We only do the case for unfoldings of recursion schemes. Fix an (m, n)-ary recursion scheme ϕ. We need to show that for any elements

$$g_1 \sim f_1, \quad \ldots, \quad g_m \sim f_m, \qquad v_1 \sim w_1, \quad \ldots, \quad v_n \sim w_n$$

respectively from H and V we have

$$\mathrm{unfold}_\phi[g_1, \ldots, g_m, v_1, \ldots, v_n] \sim \mathrm{unfold}_\phi[f_1, \ldots, f_m, w_1, \ldots, w_n]. \tag{7.7}$$

If all equivalent elements are equal (i.e. $g_i = f_i$, $v_i = w_i$), then (7.7) is obvious. If it is not the case, then we have that for some i either $g_i, f_i \in G$ or $v_i, w_i \in W$. We claim that in this situation both unfolds belong to G. It is true, since every unfold which uses $g \in G$ can be decomposed as $\tau[\eta] \cdot g$ for some term τ and valuation η and from the definition of G we have $Vg \subseteq G$. Similarly, every unfold which uses $v \in W$ can be decomposed as $\tau_1[\eta_1] \cdot v \cdot \tau_2[\eta_2]$ and $VvH \subseteq G$. □

Let β be the function which maps an element of the forest algebra (H, V) to its equivalence class. By the above lemma, β is a homomorphism

$$\beta \colon (H, V) \to (H, V)_{/\sim},$$

where the elements of the regular-infinite-forest algebra on the right hand side are equivalence classes under $\sim$, and the operations are inherited from (H, V). Note also the same forests are mapped to h by α and by $\beta \circ \alpha$. Therefore, we can use the induction assumption, applied to the homomorphism

$$\beta \circ \alpha \colon A^{\mathsf{reg}\triangle} \to (H, V)_{/\sim},$$

to obtain a formula for the set of forests that are mapped to h by α.

Case (ii).

We now consider the case when h is the unique subminimal element.

Recall that F is the set of all elements different from h, from which h is reachable. In other words, F is the set of all elements from H beside h and $\bot$. Thanks to the case (iii), for every $f \in F$ we have a formula φ_f that defines the set L_f of forests with type f. We write φ_F for the disjunction $\bigvee_{f \in F} \varphi_f$.

Recall that the two nodes x, y of a regular forest t are in the same component if the subtree $t|_x$ is a subtree of the subtree $t|_y$ and vice versa. We have two kinds of components: connected and singleton. Since any two nodes in the same component are EF-bisimilar (i.e. their subtrees are EF-bisimilar), we conclude that two nodes in the same component have the same type. Therefore, we can speak of the type of a component. A regular tree is called *prime* if it has exactly one component with a type outside F. Note that the component with a type outside F must necessarily be the root component (the one that contains the root), since no type from F is reachable from types outside F. Depending on the kind of the root component, a prime tree is called a *connected prime* or *singleton prime*.

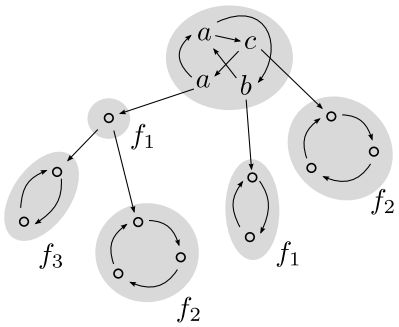


Figure 7.7

The *profile* of a prime tree t is a pair in $P(F) \times (A \cup P(A))$ defined as follows. On the first coordinate, we store the set $G \subseteq F$ of types of components with a type in F. On the second coordinate, we store the labels that appear in the root component. If the tree is connected prime, this is a set $B \subseteq A$ of labels (possibly containing a single label), and if the tree is singleton prime, this is a single label $b \in A$. In the first case, the profile is called a *connected profile*, and in the second case the profile is called a *singleton profile*.

Figure 7.7 shows the component graph of a connected prime tree with connected profile $(\{f_1, f_2, f_3\}, \{a, b, c\})$.

It turns out that the profile of a prime tree determines its type.

Lemma 81. *All prime trees with the same profile have the same type.*

Proof. Let s, t be prime trees. Let G be the first coordinate of the profile, and let $\{s_1, \ldots, s_k\}$ be all the subtrees of s with types in F, likewise for $\{t_1, \ldots, t_n\}$ in t. By property (7.1) we can assume that $\{\alpha(s_1), \ldots, \alpha(s_k)\} = G = \{\alpha(t_1), \ldots, \alpha(t_n)\}$.

We modify the tree t by replacing every subtree t_i with some tree s_j with the same type ($\alpha(t_i) = \alpha(s_j)$). This operation does not change the type of t. The resulting tree t' is EF-bisimilar to tree s, and therefore has the same type, since α is invariant under EF-bisimulation. □

Therefore it is meaningful to define the set $prof_h$ of profiles of prime trees of type h. Since every prime tree has a type from the set $\{h, \bot\}$, the remaining profiles are profiles of prime trees of type $\bot$.

Lemma 82. *Let π be a profile. There is a tree EF formula φ_π such that*

- *Every prime tree with profile π satisfies φ_π.*
- *Every regular tree satisfying φ_π has type h if $\pi \in prof_h$ and $\bot$ otherwise.*

Proof. First consider the case when the profile π is a singleton profile (G, b). The formula φ_π says that the root label is b, all proper subtrees have types in G, and any type in G appears in some proper subtree. Since the types in G all belong to F, we are allowed to use formulas φ_g that define the forests with type g. Note that these are forest EF formulas, but we treat them here as tree EF formulas. Therefore, having profile (G, b) is defined by the following formula $\varphi_{(G,b)}$:

$$b \quad \wedge \quad \mathsf{AG} \bigvee_{g \in G} \varphi_g \quad \wedge \quad \bigwedge_{g \in G} \mathsf{EF} \varphi_g.$$

Note that the formula above satisfies a stronger property than required by the proposition, since it describes exactly the trees with profile (G, b), i.e. we could strengthen the second property in the proposition to "any regular tree satisfying $\varphi_{(G,b)}$ is a tree of profile (G, b)". This will not, however, be the case in the construction below for connected profiles.

We now consider the case when the profile π is a connected profile (G, B). Let us collect some properties of a prime tree t with this profile. Below we use the term "type of a node" to refer to the type of the node's subtree. Recall that for $I \subseteq F$, we write φ_I for the disjunction $\bigvee_{f \in I} \varphi_f$. From the definition of a prime tree, a node in a prime tree is in the root component if and only if it has a type outside F. In particular, the tree t has a type outside F:

$$\neg \varphi_F. \tag{7.8}$$

Since the profile is (G, B), we know that all nodes outside the root component have a type in G, which is stated by the formula:

$$\mathsf{AG}^*(\varphi_F \to \varphi_G). \tag{7.9}$$

In (7.9), we could have used AG instead of AG^*, since the root of the tree is guaranteed to have a type outside F.

Again from the definition of the profile, we know that all nodes in the root component have a label in B:

$$\mathsf{AG}^*\Big(\neg\varphi_F \to \bigvee_{b\in B} b\Big). \tag{7.10}$$

Since the root component is connected, any node in the root component has proper descendant that is still in the root component. In particular, a node in the root component has proper descendants in the root component with all labels from B.

$$\mathsf{AG}^*\Big(\neg\varphi_F \to \bigwedge_{b\in B} \mathsf{EF}(b \wedge \neg\varphi_F)\Big). \tag{7.11}$$

In the same way, any node in the root component has proper descendants with all possible types from G.

$$\mathsf{AG}^*\Big(\neg\varphi_F \to \bigwedge_{g\in G} \mathsf{EF}\varphi_g\Big). \tag{7.12}$$

Let $\varphi_{(G,B)}$ be the conjunction of the formulas (7.8)–(7.12). From the discussion above we obtain the first property required by the proposition: any prime tree with profile (G, B) satisfies $\varphi_{(G,B)}$. We now proceed to show the second property, namely that any regular tree satisfying $\varphi_{(G,B)}$ has type h if $(G, B) \in \mathit{prof}_h$ and $\bot$ otherwise.

Let t be a regular tree that satisfies the formulas (7.8)–(7.12). Let $t_1, \ldots, t_m$ be all the subtrees of t that have types in F, this set is finite since t is regular. Let $b_1, \ldots, b_n$ be all the labels in B. Let us define the following tree (depicted for $n = m = 2$ on figure 7.8).

$$s = \big(b_1 b_2 \cdots b_n(\Box + t_1 + \cdots + t_m)\big)^\infty.$$

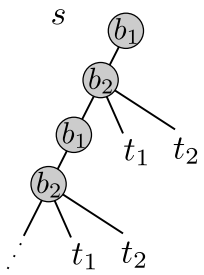


Figure 7.8

The types of the trees $t_1, \ldots, t_m$ are included in G thanks to (7.9). These types are actually exactly G, thanks to (7.12) and (7.8). Therefore, s is a connected prime tree with profile (G, B), and so it has type h if $(G, B) \in \mathit{prof}_h$ and $\bot$ otherwise. We will complete the proof by showing that the trees t and s are EF-bisimilar, even when t is not prime.

Why are s and t EF-bisimilar? Suppose that at the beginning of a round in the EF game, we have a subtree s' of s and a subtree t' of t. Duplicator's strategy is to preserve the following invariant:

> If the type of s' or t' is in F, then the types of s' and t' are equal. Otherwise, the types of both s' and t' are outside F, and the trees have the same root labels.

As a side note observe that the formula $\varphi_{(G,B)}$ is in fact a forest EF formula, thus we could strengthen the second property in the proposition to "any regular forest satisfying $\varphi_{(G,B)}$ has type h if $(G,B) \in \mathit{prof}_h$ and $\bot$ otherwise". We will not use this fact, however. □

We now turn our attention from prime trees to arbitrary regular forests. Let t be a forest. The formula which says when the type is h works differently, depending on whether t has a prime subtree or not. This case distinction can be done in EF, since not having a prime subtree is expressed by the formula $\mathsf{AG}^*\varphi_F$.

There is no prime subtree. We write $\sum\{g_1, \ldots, g_n\}$ for $g_1 + \cdots + g_n$. By invariance under EF-bisimulation, this value does not depend on the order or multiplicity of elements in the set. Suppose $\sum G = \bot$ and t has subtrees with each type in G. Thanks to (7.1), the type of t satisfies $\alpha(t) + \sum G = \alpha(t)$, and hence $\alpha(t) = \bot$. Therefore, a condition necessary for having type h is

$$\neg \bigvee_{G \subseteq F,\ \sum G = \bot} \bigwedge_{g \in G} \mathsf{EF}^* \varphi_g. \tag{7.13}$$

By the same reasoning, a condition necessary for having type h is

$$\bigvee_{G \subseteq F,\ \sum G = h} \bigwedge_{g \in G} \mathsf{EF}^* \varphi_g. \tag{7.14}$$

It is not difficult to show that conditions (7.13) and (7.14) are also sufficient for a forest with no prime subtrees to have type h.

There is a prime subtree. As previously, a forest t with type h cannot satisfy (7.13). What other conditions are necessary? It is forbidden to have a subtree with type $\bot$. Thanks to Lemma 82, t must satisfy:

$$\neg \bigvee_{\pi \notin \mathit{prof}_h} \mathsf{EF}^* \varphi_\pi. \tag{7.15}$$

Since t has a prime subtree, its type is either h or $\bot$. Suppose that t has a subtree with type $f \in F$ such that $f + h = \bot$. By (7.1), we would have $\alpha(t) + f = \alpha(t)$, which implies that the type of t is $\bot$. Therefore, t must satisfy

$$\neg \bigvee_{f \in F,\ f + h = \bot} \mathsf{EF}^* \varphi_f. \tag{7.16}$$

Let us define C to be the labels that preserve h, i.e. the labels $a \in A$ such that $\alpha(a)h = h$. If a forest has type h, then it cannot have a subtree as where $a \notin C$ and s has type h or $\bot$. This is stated by the formula:

$$\mathsf{AG}^* \bigwedge_{c \notin C} (c \to \mathsf{AG}\varphi_F). \tag{7.17}$$

As we have seen, conditions (7.13) and (7.15)–(7.17) are necessary for a forest with a prime subtree t having type h. In the following lemma, we show that the conditions are also sufficient. This completes the analysis of case (ii) of the proof of Theorem 77, since it gives a formula that characterizes the set L_h of forests whose type is the unique subminimal element h.

Lemma 83. *A regular forest with a prime subtree has type h if it satisfies conditions* (7.13) *and* (7.15)–(7.17).

Proof. By induction on the number of components in a forest t with a type outside F, we show that if t satisfies conditions (7.13) and (7.15)–(7.17), its type is not $\bot$. This gives the statement of the lemma, since a forest with a prime subtree cannot have a type in F, so it has type h.

The induction base is when t is prime (so it has exactly one component with a type outside F). The type must be h, since $\bot$ is ruled out by (7.15).

Consider now the induction step. Suppose first that t is a concatenation of trees $t_1 + \cdots + t_n$. Suppose that the statement holds for $t_1, \ldots, t_n$, which have types $G = \{f_1, \ldots, f_n\}$ and $\bot \notin G$. The type of t is $\sum G$. If $G \subseteq F$, then (7.13) implies that $\sum G$ is not $\bot$. If $h \in G$, then (7.16) implies that $\sum G = h$.

It remains to show the lemma when t is a tree. If the root component is a singleton component, a simple argument shows that the type of t is not $\bot$. We are left with the case when t is a tree where the root component is connected. Let $a_1, \ldots, a_n$ be the labels in the root component. Let $t_1, \ldots, t_m$ be all the subtrees of t that have a smaller number of components with a type outside F, and let $f_1, \ldots, f_m$ be their types. Since t is not prime, some f_i is h. Since the root component is connected, each node in the root component has a descendant with type h. Therefore, by (7.17), all the labels $a_1, \ldots, a_n$ belong to C.

It is not difficult to show that t is EF-bisimilar to, and thus has the same type as, the tree

$$s = \big(a_1 \cdots a_n(\Box + t_1 + \cdots + t_m)\big)^\infty.$$

By (7.16), we have $h = h + f_1 + \cdots + f_m$. Therefore, the tree s has the same type as

$$\big(a_1 \cdots a_n(\Box + t_i)\big)^\infty.$$

Let $v = \alpha(a_1 \cdots a_n)$. Our goal is to show that $(v(\Box + h))^\infty = h$. By assumption on $a_1, \ldots, a_n \in C$, we have $vh = h$. In particular, $v^\omega h = h$. Finally

$$(v(\Box + h))^\infty = v(v + h)^\infty = v(v + v^\omega h)^\infty \overset{(7.3)}{=} vv^\omega h = h.$$

The last equality is the only time we use identity (7.3) in this thesis. □

7.3 Checking invariance under EF-bisimulation

In this section we show how to check whether a given regular language of regular (thin) forests is invariant under EF-bisimulation.

Theorem 84. *It is decidable, given a forest automaton $\mathcal{A}$, if there exist two EF-bisimilar regular forests, of which only one is accepted by $\mathcal{A}$.*

We note that, using the reasoning similar to this in proof of Theorem 63, this result could be stated also for languages of (thin) forests, if the following conjecture (similar to Lemma 64) should be true:

Conjecture 85. *Let L be a regular language of forests. The language which contains of all forests which are EF-bisimilar to some forest from L is regular.*

We present a solution in the case of thin forests and in the general case. For thin forests the invariance under EF-bisimulation could be expressed in terms of identities:

Theorem 86. *A regular language of regular thin forests is invariant under EF-bisimulation if and only if its syntactic regular-thin-forest algebra (H, V) satisfies the identities* (7.1) *and* (7.4)*, i.e.*

$$vh + h = vh,$$
$$(v + (vw)^\infty)^\infty = (vw)^\infty,$$

as well as identities

$$h + v = v + h \qquad \text{for } h \in H,\ v \in V, \tag{7.18}$$
$$(vwu)^\infty = (vuw)^\infty \qquad \text{for } v, w, u \in V,\ vwu \in V_+. \tag{7.19}$$

The last identity can be rephrased in a more general way:

Lemma 87. *Let a thin-forest algebra (H, V) satisfy* (7.19)*. Then*

$$(v_1 v_2 \cdots v_n)^\infty = (v_{\pi(1)} v_{\pi(2)} \cdots v_{\pi(n)})^\infty$$

for every permutation π of $\{1, \ldots, n\}$ and every $v_1, \ldots, v_n \in V$ such that $v_1 v_2 \cdots v_n \in V_+$.

Proof. Observe that for any $v, w_1, w_2, u \in V$ such that $vw_1w_2u \in V_+$ we have

$$(v\ w_1\ w_2 u)^\infty \overset{(7.19)}{=} (vw_2\ u\ w_1)^\infty \overset{(7.19)}{=} (vw_2w_1u)^\infty.$$

Now the lemma follows from the fact that every permutation is a product of adjacent transpositions. □

However, in the case of general forests we do not know how to express the condition using only identities. Before we formulate a theorem, we need a definition of a multicontext.

An *n-ary multicontext* over variables $x_1, \ldots, x_n$ is a regular forest over alphabet $A \cup \{x_1, \ldots, x_n\}$ where the variables $x_1, \ldots, x_n$ are allowed only in leaves. We allow multiple (possibly infinitely many) occurrences of each variable. Given forests $s_1, \ldots, s_n$ and an n-ary multicontext p, the forest $p(s_1, \ldots, s_n)$ over A is defined in the natural way. Therefore, an n-ary multicontext is a notation for the unfolding of a recursion scheme ϕ that has free forest-valued variables $x_1, \ldots, x_n$, and some context-valued variables already bound by a valuation:

$$p(s_1, \ldots, s_n) = \mathrm{unfold}_\phi[\eta[x_1 \leftarrow s_1, \ldots, x_n \leftarrow s_n]]. \tag{7.20}$$

An n-ary multicontext is called *prime* if, when treated as a forest over the alphabet $A \cup \{x_1, \ldots, x_n\}$, it has one root component, and also all of the non-variable nodes are in this component. An example of a 4-ary prime multicontext is given on figure 7.9.

We say that two multicontexts are EF-bisimilar if they are EF-bisimilar when treated as forests over the alphabet $A \cup \{x_1, \ldots, x_n\}$.

By using the morphism $\alpha\colon A^{\mathsf{reg}^\triangle} \to (H, V)$, an n-ary multicontext naturally induces a function $H^n \to H$. Under the notation from (7.20), this is the function

$$(\alpha(s_1), \ldots, \alpha(s_n)) \qquad \mapsto \qquad \alpha(\mathrm{unfold}_\phi[\eta[x_1 \leftarrow s_1, \ldots, x_n \leftarrow s_n]]).$$

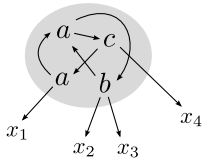


Figure 7.9

Since α is a morphism, the above definition does not depend on the choice of $s_1, \ldots, s_n$.

We say that a morphism satisfies the *cyclic representative condition* if for every letters $a_1, \ldots, a_n \in A$ and multicontext variables $x_1, \ldots, x_m$, every multicontext that is EF-bisimilar to

$$p = (a_1 \cdots a_n(\Box + x_1 + \cdots + x_m))^\infty$$

induces the same transformation $H^m \to H$.

We are now ready to state the necessary and sufficient conditions for invariance under EF-bisimulation.

Theorem 88. *A morphism into a regular-infinite-forest algebra is invariant under EF-bisimulation if and only if its target algebra satisfies identities* (7.1), (7.2), (7.4), *i.e.*

$$\begin{aligned} vh &= vh + h, \\ h + g &= g + h, \\ (vw)^\infty &= (v + (vw)^\infty)^\infty, \end{aligned}$$

and the morphism satisfies the cyclic representative condition.

We prove Theorems 86 and 88 in the next section. Since for thin forests the identities can be decided thanks to Theorem 50, this completes the proof of Theorem 84 in this case.

Then, in section 7.3.2, we show how to decide if the syntactic morphism of the language recognized by $\mathcal{A}$ satisfies the cyclic representative condition. This completes the proof of Theorem 84 in the general case, since identities can be decided thanks to Theorem 55.

7.3.1 Proof of the characterization theorems

In this section we prove Theorems 86 and 88. We do this simultaneously, since most of the arguments are the same in both cases. The only differences in proofs are places when we use either the cyclic representative condition or identities (7.18), (7.19).

The "only if" part of the proofs is quite obvious. The rest of this section is devoted to the "if" part.

We want to show that if regular forests s and t are EF-bisimilar, then they have the same types, i.e. $\alpha(s) = \alpha(t)$. The proof is by induction on the number of components in s plus the number of components in t.

Lemma 89. *Without loss of generality, we can assume that s and t are regular trees.*

Proof. Let $s_1, \ldots, s_n$ be all subtrees in s and $t_1, \ldots, t_m$ be all subtrees in t. By using identities (7.1) and (7.2) we have

$$\alpha(s) \overset{(7.1)}{=} \alpha(s) + \alpha(s_1) + \cdots + \alpha(s_n) \overset{(7.2)}{=} \alpha(s_1) + \cdots + \alpha(s_n).$$

Similarly $\alpha(t) = \alpha(t_1) + \cdots + \alpha(t_m)$. Since s and t are EF-bisimilar, then every s_i is EF-bisimilar to some $\hat{s}_i \in \{t_1, \ldots, t_m\}$ and every t_i is EF-bisimilar to some $\hat{t}_i \in \{s_1, \ldots, s_n\}$. Suppose we proved the proposition for trees. Then $\alpha(s_i) = \alpha(\hat{s}_i)$ and $\alpha(t_i) = \alpha(\hat{t}_i)$, thus $\{\alpha(s_1), \ldots, \alpha(s_n)\} = \{\alpha(t_1), \ldots, \alpha(t_m)\}$. Therefore $\alpha(s) = \alpha(t)$. □

The induction base is when both trees s and t have a single component. If s is finite, then it has a single node a. In this case t also has to be a, since this is the only tree that is EF-bisimilar to a. (Note that we cannot check label in a root of a tree, but we can check whether a tree is of height 1 and check label in a leaf.) Suppose now that s and t are infinite. Let $a_1, \ldots, a_n$ be the labels that appear in s (and therefore also in t). It is easy to see that s and t are EF-bisimilar to a tree $u = (a_1 \cdots a_n \Box)^\infty$. All of trees s, t, u can be treated as prime multicontexts of arity 0.

In the case of general trees, by the cyclic representative condition, both s and t have the same type.

In the case of thin trees, from Lemma 4, $s = (a_{\pi(1)} \cdots a_{\pi(n)} \Box)^\infty$ for some permutation π of $\{1, \ldots, n\}$. Applying Lemma 87 we get that $\alpha(s) = \alpha(u)$. Analogously we get that $\alpha(t) = \alpha(u)$.

We now do the induction step. Let $s_1, \ldots, s_n$ be all the subtrees of s that have fewer components than s. In other words, there is a prime n-ary multicontext p such that

$$s = p(s_1, \ldots, s_n).$$

Likewise, we distinguish all subtrees $t_1, \ldots, t_k$ inside t that have fewer components than t, and find a prime k-ary multicontext q with

$$t = q(t_1, \ldots, t_k).$$

Since the trees s and t are EF-bisimilar, each tree s_i must be EF-bisimilar to some subtree $\hat{s}_i$ of t. By the induction assumption, we know that the trees s_i and $\hat{s}_i$ have the same type (since s_i has fewer components than s and $\hat{s}_i$ has no more components than t). Likewise, each tree t_i has the same type as some subtree $\hat{t}_i$ of s.

By applying (7.1) in the same manner as in Lemma 89, we conclude that if either p or q is finite then s and t have the same type. We are left with the case when both p and q are infinite prime multicontexts. Suppose first that

(1) for some i, the tree $\hat{s}_i$ has the same number of components as t; and

(2) for some j, the tree $\hat{t}_j$ has the same number of components as s.

We use the same notion of reachability on types as was used in Lemma 79. From (1) we conclude that the tree $\hat{s}_i$ is in the root component of t, and therefore the type of both s_i and $\hat{s}_i$ is reachable from the type of t. Since s_i is a subtree of s, we conclude that the type of s is reachable from the type of t. Reasoning in the same way from (2) we conclude that type of t is reachable from the type of s. Therefore, by Lemma 79, the types of s and t are equal (note that Lemma 79 used (7.1)).

Suppose now that one of (1) or (2) does not hold, say (1) does not hold (the other case is symmetric).

Lemma 90. *Without loss of generality, we can assume that* $n \leq k$ *and*

$$s = p(t_1, \ldots, t_n).$$

Proof. Consider the tree $\hat{s} = p(\hat{s}_1, \ldots, \hat{s}_n)$. Since we replaced trees with EF-bisimilar ones, $\hat{s}$ is bisimilar to s. Since we replaced trees with ones of the same type, $\hat{s}$ has the same type as s. So it is enough to prove the result for $\hat{s}$ and t.

Since (1) does not hold, then every $\hat{s}_i$ is equal to some t_j. Rename the subtrees $t_1, \ldots, t_k$ such that $\{\hat{s}_1, \ldots, \hat{s}_n\} = \{t_1, \ldots, t_{n'}\}$ for some $n' \leq \min(n, k)$. After possibly renaming variables in p, the tree $\hat{s}$ has the form $p(t_1, \ldots, t_{n'})$, like in the statement of the lemma. □

What about the trees $t_{n+1}, \ldots, t_k$ that do not appear in s? Each of these is EF-bisimilar to one of $s, t_1, \ldots, t_n$. For those that are EF-bisimilar to some $t_i \in \{t_1, \ldots, t_n\}$, we use the tree instead. Therefore, we can without loss of generality assume that

$$t = q(s, t_1, \ldots, t_n).$$

Lemma 91. *Any label $a \in A$ that appears in q also appears in p.*

Proof. Let $a \in A$ be a label in q and consider the following strategy for Spoiler in the game over the trees s and t: he picks t and in that tree, some occurrence of a in the root component. Duplicator, in his response, cannot pick a node inside any of the trees $t_1, \ldots, t_n$, since none of these is EF-bisimilar to a tree in the root component of t, since (1) does not hold. Therefore, he must pick a node inside p. □

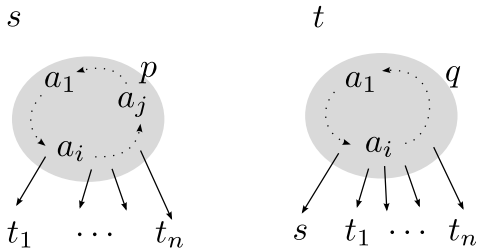


Figure 7.10

Let $a_1, \ldots, a_i$ be the labels that appear in q. Thanks to the above lemma, the labels that appear in p are $a_1, \ldots, a_i$ as well as possibly some other labels, say $a_{i+1}, \ldots, a_j$ for some $j \geq i$. Therefore the trees s, t look like on figure 7.10. Let us define the following two contexts

$$x = a_1 \cdots a_i(\Box + t_1 + \cdots + t_n), \qquad y = a_{i+1} \cdots a_j \Box.$$

In the case of general trees, by using the cyclic representative condition, it is not difficult to show that type of s is the same as the type of $(xy)^\infty$. Again using this condition, one shows that the type of t is the same as the type of $(x(\Box + s))^\infty$, which is the same as the type of $(x(\Box + (xy)^\infty))^\infty$. Therefore, we conclude that the types of s and t are equal by identity (7.4).

In the case of thin trees, observe that from Lemma 4 every tree with a connected component in the root can be written as

$$u = \big(b_1(t'_1 + \Box + t''_1) \cdots b_m(t'_m + \Box + t''_m)\big)^\infty$$

for some $m \geq 1$, letters $b_1, \ldots, b_m \in A$, and thin forests $t'_1, t''_1, \ldots, t'_m, t''_m$. From the identities we conclude that for every $v_1, \ldots, v_m \in V_+$ and $h_1, \ldots, h_m, g_1, \ldots, g_m \in H$ we have

$$\begin{aligned}\big(v_1(h_1 + \Box + g_1) \cdots v_m(h_m + \Box + g_m)\big)^\infty &\overset{(7.19)}{=} \\ &= \big(v_1 \cdots v_m(h_1 + \Box + g_1) \cdots (h_m + \Box + g_m)\big)^\infty = \\ &= \big(v_1 \cdots v_m(h_1 + \cdots + h_m + \Box + g_m + \cdots + g_1)\big)^\infty \overset{(7.18)}{=} \\ &= \big(v_1 \cdots v_m(\Box + h_1 + \cdots + h_m + g_1 + \cdots + g_m)\big)^\infty.\end{aligned}$$

That shows that the type of u is the same as the type of

$$\big(b_1 \cdots b_m(\Box + t'_1 + \cdots + t'_m + t''_1 + \cdots + t''_m)\big)^\infty$$

Using identities (7.19) and (7.18) we can further rearrange letters b_i and trees from forests t'_i, t''_i. From this it is easy to show that $\alpha(s) = \alpha((xy)^\infty)$ and $\alpha(t) = \alpha((x(\Box + s))^\infty)$.

7.3.2 Deciding the cyclic representative condition

Let us fix an automaton $\mathcal{A}$, with states Q and input alphabet A. There are two morphisms that are associated with $\mathcal{A}$.

The first morphism is the automaton morphism $\alpha\colon A^{\mathsf{reg}^\triangle} \to (H, V)$, as defined in chapter 5. Recall that elements of H are subsets of Q.

The second morphism is the syntactic morphism $\alpha_L\colon A^{\mathsf{reg}^\triangle} \to (H_L, V_L)$. We will say type of t for the value $\alpha_L(t)$. Our goal in this section is to decide if α_L satisfies the cyclic representative condition.

A counterexample to the condition is a sequence $a_1, \ldots, a_n \in A$, types $h_1, \ldots, h_m \in H$ and a multicontext q that is EF-bisimilar to

$$p = (a_1 \cdots a_n(\Box + x_1 + \cdots + x_m))^\infty$$

such that the types

$$p(h_1, \ldots, h_m), \qquad q(h_1, \ldots, h_m).$$

are different under the syntactic congruence in (H, V) induced by the language recognized by $\mathcal{A}$. We can assume that $h_1, \ldots, h_m$ are all distinct, since it does not make sense to use the same type under different variables. We also can assume that $a_1, \ldots, a_n$ are all distinct (otherwise, instead of q we could use the multicontext obtained from p by removing duplicates from $a_1 \cdots a_n$).

We try to find a counterexample for each possible choice of $a_1, \ldots, a_n$ and $h_1, \ldots, h_m$. Enumerating all possible sequences $h_1, \ldots, h_m$ is one of the reason why this method is doubly exponential, since the size of H is exponential in Q.

To simplify the proof, we assume that for each type $h \in H$ there is a label a_h such that the tree a_h has type h. Therefore, searching for a counterexample comes down to checking if there is some forest that is EF-bisimilar to

$$t = (a_1 \cdots a_n(\Box + a_{h_1} + \cdots + a_{h_n}))^\infty.$$

but has a different type than t, under the syntactic congruence. Let L_t be the set of forests that are EF-bisimilar to t. We want to know if all forests from L_t are syntactically equivalent to t. This boils down to testing inclusion of two regular languages, since the language L_t is regular (and even definable in EF), by the following lemma.

Lemma 92. *Let t be a regular tree. The set of trees that are EF-bisimilar to t is definable by an EF formula φ_t.*

Proof. The proof is by induction on the number of components in t. Let $t_1, \ldots, t_m$ be the subtrees of t that have fewer components than t. By induction assumption, we already have formulas $\varphi_{t_1}, \ldots, \varphi_{t_m}$, where the formula φ_{t_i} defines a language of trees EF-bisimilar to t_i. The formula is constructed in similar fashion as in Lemma 82.

Suppose first that the root of t is in a singleton component, i.e. all proper subtrees of t are among $t_1, \ldots, t_m$. Let a be the root label of t. The following formula defines L_t:

$$a \quad \wedge \quad \mathsf{AG} \bigvee_{i \leq m} \varphi_{t_i} \quad \wedge \quad \bigwedge_{i \leq m} \mathsf{EF} \varphi_{t_i}.$$

Suppose now that the root of t is in a connected component. Let $a_1, \ldots, a_n$ be the labels that appear in the root component of t. Denote by φ_T the disjunction $\bigvee_{i \leq m} \varphi_{t_i}$. The formula for L_t is

$$\neg \varphi_T \quad \wedge \quad \mathsf{AG}^* \Big(\neg \varphi_T \ \rightarrow \ \Big(\bigvee_{i \leq n} a_i \ \wedge \ \bigwedge_{i \leq n} \mathsf{EF}(a_i \wedge \neg \varphi_T) \ \wedge \ \bigwedge_{i \leq m} \mathsf{EF} \varphi_{t_i} \Big) \Big). \qquad \square$$

Chapter 8

Conclusions

In the thesis we presented some algebraic tools suited for regular languages of infinite forests and we used them to give effective characterizations of several properties of such languages. In this final chapter we shortly discuss some our approaches that did not give us results, some recent work, and some ideas for future work.

Algebras with a finite number of operations

We are only beginning to study algebra for infinite forests. During the work on the paper [7] we concluded that we are not ready to propose the "right" framework (or at least one of the "right" frameworks). This is still true. The main shortcoming is that regular-infinite-forest algebra has an infinite number of operations and, consequently, an infinite number of axioms. This poses all sort of problems, and although we showed that the algebra is useful despite them, the algebra is far from being elegant.

We note that other researchers who study algebraic approach to infinite forests also are fighting these problems (see for example the work of Blumensath [3]).

During the research, we tried another approach and we investigated what would happen if we intentionally restricted the set of operations in regular-infinite-forest algebra to a certain finite subset. That would, obviously, result in restricting the class of forest languages which our algebra works with, but this could at the same time make the problem of designing the framework more tractable. We noted that all the identities we had used so far did not required recursion schemes other than p^∞, thus we looked at the algebra which was a forest algebra equipped with an additional operation p^∞. It turned out that the class of languages recognizable by such algebra was quite robust. This is how the idea of thin-forest algebra came to life.

We also tried other approaches. We thought that the limitation that contexts can have only one hole might be too strict for infinite objects, so we tried to modify thin-forest algebra by allowing richer context sort. For instance, in one of such modifications we allowed to have multiple (but finite) occurrences of hole, but still only one type of hole (thus a composition of a context p with a forest t results in a forest in which every hole of p is replaced with t). This is indeed a richer algebra than thin-forest algebra, since for instance a full binary tree with root and every left-child labeled with a and every right-child labeled with b can be expressed as

$$a(a\Box + b\Box)^\infty.$$

However, we still could not get every forest. For example a full binary tree over $\{a, b, c\}$

such that no child has a label equal to the label of its parent is not expressible in the modified algebra. We also could not find any robust definition of the free algebra.

The question remains whether there is an extension of thin-forest algebra, which results in an algebra with a finite number of operations, which recognizes a robust class of languages.

Other effective characterizations

One of effective characterizations presented in this thesis is testing whether a given regular language of infinite forests is definable by a formula of the temporal logic EF. Basing on this characterization, ten Cate and Facchini [36] presented a topological characterization of the logic EF.

A natural idea for future work is finding effective characterizations for other logics. For instance [12] describes characterizations of logics EX, EF and EF+EX for binary trees. Since a formula of the logic EX examines a forest up to a finite depth, the characterization of this logic in case of infinite forests is exactly the same as in case of finite ones. An interesting open question is to give a characterization for the logic EF+EX.

We tried to characterize the logic AF, in which the operator EF is replaced by the operator "Always Finally": $\mathsf{AF}\varphi$ means that for every path the formula φ is satisfied somewhere on the path. Unfortunately, the similarity between the logics AF and EF seems to be elusive, and we were not able to find an effective characterization for the logic AF, even for finite trees. It may be the case that our algebras which use contexts with only one hole are more suited for checking properties which can be rephrased as a game between two players who move a *single* pebble down a forest. This game does not work for the logic AF.

We also presented effective characterizations for several simple properties of regular languages of thin forests, like testing commutativity and invariance under bisimulation. The open question is how to check these properties on regular languages of infinite forests which are not necessarily thin.

Apart from logics that have been considered for finite trees, there are some interesting logics for infinite trees, which do not have counterparts for finite trees. An important example here is weak monadic second-order logic (WMSO), in which we allow quantification only over finite sets. In the recent work with Bojańczyk and Skrzypczak [8] we tried to tackle this problem, and we gave an effective characterization of regular languages of thin forests which are definable by WMSO formulas when treated as languages of all forests. The paper also presents some topological properties of thin forests, which show that in various meanings they are not as rich as all infinite forests.

In the thesis we also showed an effective characterizations of regular languages of thin forests which are open in the standard topology. This was significantly extended by the work of Bojańczyk and Place [9], who presented an effective characterization of the regular languages of (all) infinite forests, which are Boolean combinations of open languages.

Bibliography

[1] A. Arnold. A syntactic congruence for rational ω-languages. *Theor. Comput. Sci.*, 39:333–335, 1985.

[2] M. Benedikt and L. Segoufin. Regular languages definable in FO. In *Symposium on Theoretical Aspects of Computer Science*, volume 3404 of *Lecture Notes in Computer Science*, pages 327–339, 2005. A revised version, correcting an error from the conference paper, is available at `www.lsv.ens-cachan.fr/ segoufin/Papers/`.

[3] A. Blumensath. Recognisability for algebras of infinite trees. *Theor. Comput. Sci.*, 412(29):3463–3486, 2011.

[4] M. Bojańczyk. Two-way unary temporal logic over trees. In *Logic in Computer Science*, pages 121–130, 2007.

[5] M. Bojańczyk. Effective characterizations of tree logics. In *PODS*, pages 53–66, 2008.

[6] M. Bojańczyk. Beyond ω-regular languages. In J.-Y. Marion and T. Schwentick, editors, *STACS*, volume 5 of *LIPIcs*, pages 11–16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[7] M. Bojańczyk and T. Idziaszek. Algebra for infinite forests with an application to the temporal logic EF. In M. Bravetti and G. Zavattaro, editors, *CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2009.

[8] M. Bojańczyk, T. Idziaszek, and M. Skrzypczak. Regular languages of thin trees. Accepted for *International Symposium on Theoretical Aspects of Computer Science, STACS*, 2013.

[9] M. Bojańczyk and T. Place. Regular languages of infinite trees that are Boolean combinations of open sets. In A. Czumaj, K. Mehlhorn, A.M. Pitts, and R. Wattenhofer, editors, *ICALP (2)*, volume 7392 of *Lecture Notes in Computer Science*, pages 104–115. Springer, 2012.

[10] M. Bojańczyk and L. Segoufin. Tree languages definable with one quantifier alternation. In *International Colloquium on Automata, Languages and Programming*, volume 5126 of *Lecture Notes in Computer Science*, pages 233–245, 2008.

[11] M. Bojańczyk, L. Segoufin, and H. Straubing. Piecewise testable tree languages. In *Logic in Computer Science*, pages 442–451, 2008.

[12] M. Bojańczyk and I. Walukiewicz. Characterizing EF and EX tree logics. In *International Conference on Concurrency Theory, CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 131–145, 2004.

[13] M. Bojańczyk and I. Walukiewicz. Characterizing EF and EX tree logics. *Theoretical Computer Science*, 358(2-3):255–273, 2006.

[14] M. Bojańczyk and I. Walukiewicz. Forest algebras. In *Automata and Logic: History and Perspectives*, pages 107–132. Amsterdam University Press, 2007.

[15] J.R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.

[16] J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. 1960 Int. Congr. for Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

[17] N. Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124, September 1956.

[18] J. Cohen, D. Perrin, and J.-É. Pin. On the expressive power of temporal logic. *Journal of Computer and System Sciences*, 46(3):271–294, 1993.

[19] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

[20] S. Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, New York, 1976.

[21] C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the AMS*, 98:21–52, 1961.

[22] K. Etessami and T. Wilke. An UNTIL hierarchy for temporal logic. In *Logic in Computer Science*, pages 108–117, 1996.

[23] R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.

[24] R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[25] D. Perrin and J.-É. Pin. Semigroups and automata on infinite words. *Semigroups, formal languages and groups (York, 1993)*, pages 49–72, 1995.

[26] D. Perrin and J.-É. Pin. *Infinite Words*. Elsevier, 2004.

[27] J.-É. Pin. Equational descriptions of languages. *Int. J. Found. Comput. Sci.*, 23(6):1227–1240, 2012.

[28] T. Place and L. Segoufin. Deciding definability in $\mathrm{FO}_2(<)$ (or XPath) on trees. In *LICS*, pages 253–262. IEEE Computer Society, 2010.

[29] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–23, 1969.

[30] M.O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Providence, RI, 1972.

[31] J. Reiterman. The Birkhoff theorem for finite algebras. *Algebra Universalis*, 14, 1982.

[32] H.G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 74:358–366, 1953.

[33] M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

[34] I. Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222, 1975.

[35] H. Straubing. *Finite Automata, Formal Languages, and Circuit Complexity.* Birkhäuser, Boston, 1994.

[36] B. ten Cate and A. Facchini. Characterizing EF over infinite trees and modal logic on transitive graphs. In F. Murlak and P. Sankowski, editors, *MFCS*, volume 6907 of *Lecture Notes in Computer Science*, pages 290–302. Springer, 2011.

[37] D. Thérien and T. Wilke. Temporal logic and semidirect products: An effective characterization of the Until hierarchy. In *Foundations of Computer Science*, pages 256–263, 1996.

[38] D. Thérien and T. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *ACM Symposium on the Theory of Computing*, pages 256–263, 1998.

[39] W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25:360–375, 1982.

[40] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.

[41] B.A. Trakthenbrot. Finite automata and the logic of monadic second order predicates (Russian). *Doklady Academii Nauk SSSR*, 140:326–329, 1961.

[42] T. Wilke. An Eilenberg theorem for ∞-languages. In J.L. Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 588–599. Springer, 1991.

[43] T. Wilke. An algebraic theory for languages of finite and infinite words. *Inf. J. Alg. Comput.*, 3:447–489, 1993.

[44] T. Wilke. Classifying discrete temporal properties. In *Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46, 1999.

Index