Decidable Characterizations for Tree Logics

Author:
Thomas Place

Defended in front of:

Denis Lugiez	*Examiner*
Jean-Éric Pin	*Reviewer*
Luc Segoufin	*Advisor*
Howard Straubing	*Reviewer*
Igor Walukiewicz	*Examiner*
Marc Zeitoun	*Examiner*

Thank you!

Before doing some science, I want to thank all the people who made this thesis a reality. The first person that comes to mind is my supervisor Luc Segoufin, he has been the best advisor I could have dreamed of for these three years. I also really need to thank all the people of the LSV as a whole, this lab was a wonderful place to work in. Especially, I thank all my fellow PhD students friends. Many thanks, to Camille, Diego, Étienne, Hilal, Jean-Loup, Pierre and all the others!

I am also very grateful to Jean-Éric Pin and Howard Straubing who accepted to review my thesis as well as to the other members of my Jury: Denis Lugiez, Igor Walukiewicz and Marc Zeitoun.

Finally I want to thank all my non-scientists friends as well as my family!

Contents

Introduction

In this thesis we investigate the expressive power of several logics over finite trees. In particular we want to understand precisely the expressive power of first-order logic over finite trees. Because we study many logics, we proceed by comparison to a logic that subsumes them all and serves as a yardstick: monadic second-order logic. Each logic we consider is a fragment of monadic second-order logic (MSO). MSO is linked to the theory of formal languages. To each logical formula corresponds a tree language, which is the language of trees satisfying this formula. Furthermore, given a logic we can associate a class of tree languages: the class of languages definable by a formula of this logic. In the setting of finite trees MSO corresponds exactly to the class of regular tree languages ([TW68]). Given a logic, we actually look for a decidable characterization of the class of languages defined in this logic. By decidable characterization, we mean an algorithm for solving the following problem: given as input a finite tree automaton, decide if the recognized language belongs to the class in question. We will actually obtain our decidable characterizations by exhibiting for each class a set of closure properties such that a language is in the class under investigation if and only if it satisfies these closure properties. Each such closure property is then shown to be decidable. Stating and proving such closure properties usually yields a solid understanding of the expressive power of the corresponding logic. The main open problem in this research area is to obtain a decidable characterization for the class of tree languages that are definable in first-order logic (FO).

Before we study the general case of trees, we present a natural subcase: the setting of words. A word structure is a natural restriction of a tree structure. The problem of obtaining decidable characterizations of classes of regular word languages has been extensively studied and is now well understood. Using algebraic techniques, decidable characterizations were provided for many classes of regular word languages. The closure properties involved in each of these characterizations are expressed using identities on the syntactic semigroup or the syntactic monoid of the regular language. The syntactic monoid (resp. syntactic semigroup) of a regular language is the transition monoid (resp. semigroup) of its minimal deter-

ministic automaton. Since in the case of regular languages these objects are finite and computable from the automaton recognizing the language, the identities are easily verified. In particular, a decidable characterization was provided for the class of languages definable in FO($<$). By FO($<$) we mean first-order logic over words using a binary relation representing the left to right order of positions on the words. The decidable characterization of FO($<$), which is perhaps the most celebrated one in this area, states that a word language is definable in FO($<$) iff its syntactic monoid is aperiodic [Sch65]. Aperiodicity corresponds to the identity $x^\omega = x^{\omega+1}$. This means that a monoid is aperiodic if and only if all elements x of the monoid verify $x^\omega = x^{\omega+1}$ where ω is the size of the monoid. This identity is easily verified on the syntactic monoid. Hence this yields a decision algorithm for testing definability in FO($<$) of a word language. Many other classes have been studied in the setting of words. We present some of them here.

The first one is the class of languages definable in first-order logic using only two variables (FO^2). There are actually two interesting logics corresponding to FO^2. The first one is $\mathrm{FO}^2(<)$ using the order predicate and the second one is $\mathrm{FO}^2(<, Succ)$ which moreover uses the successor predicate. Both logics were given a decidable characterization in [TW98] using identities on the syntactic monoid for $\mathrm{FO}^2(<)$ and the syntactic semigroup for $\mathrm{FO}^2(<, Succ)$. Another related logic is Δ_2. A language is definable in $\Delta_2(<)$ if and only if it is definable by a formula of FO($<$) with a quantifier prefix $\exists^*\forall^*$ and simultaneously by a first-order formula with a quantifier prefix $\forall^*\exists^*$. Decidable characterizations were provided for $\Delta_2(<)$ and $\Delta_2(<, Succ)$ in [PW97]. Since these characterizations are identical to the ones of [TW98] for $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$ this proves that the two logics are equivalent in terms of expressive power. Note that this was apriori not obvious at all and that there is yet no known direct way to translate FO^2 into Δ_2 and vice versa.

Another interesting setting is the class of piecewise testable languages or equivalently the class of languages definable by a boolean combination of existential first-order formulas (BC-$\Sigma_1(<)$). In [Sim75], a decidable characterization for this class of languages is provided using again identities on the syntactic monoid.

The last class we consider in this thesis is LT. LT denotes the class of regular languages called *locally testable*. A language L is in LT if membership in L depends only on the presence or absence of neighborhoods of a certain fixed size in the word. LT is also closely related to the *locally threshold testable languages* (LTT) which are exactly the languages definable in FO($Succ$), first-order logic using only the successor relation. Membership in such languages is obtained by counting the number of neighborhoods of a certain size up to some threshold. The class LT is the special case where no counting is done. In [BS73, McN74, BP89, TW85], both classes are given decidable characterizations using identities on the syntactic semigroup.

All these results demonstrate how successful monoids and semigroups were for providing decidable characterizations in the word setting. This usefulness is actually reflected in two ways. First, monoids and semigroups were useful in order to state characterizations as identities. Monoids and semigroups are also crucial in the proofs of these characterizations: the algebraic techniques and tools that were developed for monoids and semigroups provided induction mechanisms that are crucial in these proofs.

Over trees, the situation is more complicated. A first problem is that there are many possible definitions of trees. In this thesis we consider three kinds of trees: *unranked trees, forests and trees of bounded rank*. Unranked trees are trees for which there exists no bound on the number of children of each node. A forest is an ordered sequence of trees. And finally a tree of rank k for some fixed integer k is a tree for which all nodes have at most k children. For expressive logics, like FO, that can express the fact that a forest is actually a tree of rank k for some fixed k, this distinction is unimportant. However, for weaker logics, like FO^2, that are not expressive enough to express such properties this leads to different characterizations.

A second problem that makes the situation more complicated in the tree setting is that we have many more natural predicates to consider. In the word setting, the relations that were natural to consider were the successor relation $Succ$ and its transitive closure $<$. In this thesis we consider two principal orders on trees, the first one, denoted $<_{\mathbf{v}}$, corresponds to the ancestor relation and can be viewed as the generalization of $<$ on words. We also consider a second order, denoted $<_{\mathbf{h}}$, which correspond to the order on siblings. But these are not the only natural orders that could be considered, one example is the lexicographic order, denoted $<_{lex}$, that was considered in [BS08]. For this reason, we get a much bigger zoology of logics to consider. For example, first-order logic can refer to at least two different logics, the first one is $\mathrm{FO}(<_{\mathbf{v}})$ which uses only the vertical order $<_{\mathbf{v}}$, and the second one is $\mathrm{FO}(<_{\mathbf{h}}, <_{\mathbf{v}})$ which uses both the vertical and horizontal orders.

The third problem is finding a formalism for stating and proving the characterizations. Unlike for words there is no commonly accepted formalism. Several such formalisms were introduced in the setting of trees but none has been successful in expressing all the characterizations obtained in the literature. Among these formalisms, perhaps the most successful so far are *forest algebras*. Forest algebras were introduced in [BW07] and can be used for characterizing classes of languages of forests. However it is specific to the setting of forests. Another example of such formalism would be the preclones as introduced in [ÉW05] which were successful for proving characterizations of some logics but unsuccessful so far for getting decidable characterizations. Finally a specific formalism is intro-

duced in [BS09, PS09] for giving the characterizations of local classes such as LT and LTT. So far, all these algebraic formalisms remain unsuccessful for providing decidable characterizations for the classes of languages definable in $\mathrm{FO}(<_{\mathbf{v}})$ and $\mathrm{FO}(<_{\mathbf{h}}, <_{\mathbf{v}})$. For now, the classes for which we have decidable characterizations correspond to logics that are all fragments of FO obtained by restricting either the quantifications or the number of variables that can be used.

Our Results. In this thesis we provide decidable characterizations for several fragments of FO. First we provide three decidable characterizations for classes of regular languages of trees of bounded rank. The three classes are characterized using an algebraic formalism that we call k-algebra (where k stands for the bound that exists on the rank of the trees). This formalism can be seen as an adaptation of forests algebra in the setting of trees of rank k. The first class we consider is the class of languages definable in the temporal logic $EF + F^{-1}$. $EF + F^{-1}$ essentially navigates the trees using two modalities, EF which is used to go to a descendant node and F^{-1} which is used to go to an ancestor node. The second class we consider is the class of trees of bounded rank definable using one quantifier alternation, $\Delta_2(<_{\mathbf{v}})$. Note, that in the setting of words the logics $F + F^{-1}$ and $\Delta_2(<)$ have the same expressibility as $\mathrm{FO}^2(<)$. We will see that this is no longer true over trees. The last class, is the class of languages definable using a boolean combination of existential first order formulas (BC-$\Sigma_1(<_{\mathbf{v}})$).

In the setting of forests, we investigate the class of languages definable in first-order logic using only two variables and the predicates $<_{\mathbf{h}}$ and $<_{\mathbf{v}}$: $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. This logic has an equivalent counterpart in temporal logic ([Mar05]). More precisely, it corresponds to the temporal logic $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ that navigates in the tree using two "vertical" modalities, one for going to some ancestor node (F^{-1}) and one for going to some descendant node (EF), and two "horizontal" modalities for going to some following sibling ($\mathrm{F}_{\mathbf{h}}$) or some preceding sibling ($\mathrm{F}_{\mathbf{h}}^{-1}$). We provide a characterization for this logic. Note that $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ can express that a forest is a tree and that for any fix number k, the fact that a tree has rank k, hence this result also apply for languages of unranked trees and languages of trees of bounded rank.

The last class for which we provide a decidable characterization is the class of locally testable language (LT). A language L is in LT if membership in L depends only on the presence or absence of neighborhoods of a certain fixed size in the tree. We define notions of LT for both unranked trees and trees of bounded rank by adapting the definition of neighborhood to each setting. Then we provide a decidable characterization for both notions of LT.

For each of our results, we use a specific technique in order to state and prove our characterization. For trees of bounded rank, the three logics we investigate, $EF+F^{-1}$, $\Delta_2(<_{\mathbf{v}})$ and BC-$\Sigma_1(<_{\mathbf{v}})$, were already given decidable characterizations for unranked trees and forests using forest algebras in [Boj07b, BSS08, BS08]. However, since these three logics cannot express that a tree is of bounded rank, new characterizations are required for the bounded rank setting. Our characterization will naturally share similarities with their unranked counterpart. However we will see that we need to introduce new identities compared to the unranked setting in order to capture the expressive power of these logics in the bounded rank setting. Furthermore, all proofs of these characterizations involve inductions that construct new trees. Because of the restriction on the number of children of nodes, these constructions become more complicated in the ranked case than in the unranked case. A simple example of this would be the languages definable in $EF + F^{-1}$ which are closed under bisimulation. The characterization of [Boj07b] reflects this property with an identity stating that the duplication of subtrees does not affect membership in the language. For trees of bounded rank, this property obviously does not make sense since such an operation would affect the number of children of a node. Therefore this operation becomes unavailable in the proof.

Our characterization of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ over forests is expressed in term of closure properties corresponding partly to identities in the syntactic forest algebra of the language as defined in [BW07], and partly via closure under a saturation mechanism designed specifically for this logic. Even though $EF+F^{-1}$ and $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ do not have the same expressive power, our proof has many similarities with the one of [Boj07b] for his characterization of $EF + F^{-1}$ and we reuse several ideas developed in this paper. However it departs from it in many essential ways. First of all the closure under bisimulation of $EF + F^{-1}$ was used in an essential way in order to compute a subalgebra and perform inductions on the size of the algebra. Moreover, because $EF + F^{-1}$ does not have horizontal navigation, it was possible to isolate certain labels and then perform an induction on the size of the alphabet. It is the combination of the induction on the size of the alphabet and on the size of the algebra that gave an elegant proof of the correctness of the identities for $EF + F^{-1}$ given in [Boj07b]. The logic $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ is no longer closed under bisimulation and we were not able to perform an induction on the algebra. Moreover because our logic has horizontal navigation, it is no longer possible to isolate the label of a node from the labels of its siblings, hence it is no longer possible to perform an induction on the alphabet. In order to overcome these problems our proof replace the inductions used in [Boj07b] by an induction on the set of forbidden patterns. This make the two proofs fairly different.

Finally, we were not able to obtain a reasonable set of identities for LT either by using forest algebra or the formalism used for characterizing locally threshold testable languages (LTT) in [BS09]. Our approach is slightly different. There

is another technique that was used on words for deciding the class LT. It is based on the "delay theorem" [Str85, Til87] for computing the expected size of the neighborhoods: Given an automaton recognizing the language L, a number k can be computed from that automaton such that if L is in LT then it is in LT by investigating the neighborhoods of size k. Once this k is available, deciding whether L is indeed in LT or not is a simple exercise. On words, a decision algorithm for LT (and also for LTT) has been obtained successfully using this approach [Boj07a]. Unfortunately all efforts to prove a similar delay theorem on trees have failed so far. We obtain a decidable characterization of LT by combining the two approaches mentioned above. We first exhibit a set of necessary conditions for a regular tree language to be in LT. Those conditions are expressed using the formalism introduced for characterizing LTT. We then show that for languages satisfying such conditions one can compute the expected size of the neighborhoods. Using this technique we obtain a characterization of LT for ranked trees and for unranked unordered trees.

Relations with Other Known Results. Several other decidable characterizations were proved both for classes of regular word and tree languages. In the setting of words we already mentioned FO($<$) [Sch65],FO^2($<$) [TW98], LT [BS73, McN74] and LTT [BP89, TW85]. Another important example is the quantifier alternation hierarchy. We already mentioned logics in this hierarchy with BC-$\Sigma_1(<_{\mathbf{v}})$ [Sim75] and $\Delta_2(<)$ [PW97]. A $\Sigma_n(<)$ formula is a FO($<$) formula that has $n-1$ quantifier alternations starting with $\exists$. Symmetrically a $\Pi_n(<)$ formula is a FO($<$) that has $n-1$ quantifier alternations starting with $\forall$. Finally a language is definable in $\Delta_n(<)$ iff it is definable by both a $\Sigma_n(<)$ and a $\Pi_n(<)$ formula. As we mentioned earlier decidable characterizations were introduced for logics that are low in this hierarchy: BC-$\Sigma_1(<_{\mathbf{v}})$ and $\Delta_2(<)$. Decidable characterizations were also obtained for $\Sigma_2(<)$ and $\Pi_2(<)$ [PW97]. However obtaining decidable characterization remains an open problem for logics that are higher in this hierarchy. For example we do not possess yet decidable characterization for the class of languages definable with a boolean combination of $\Sigma_2(<)$ formula or the class of languages definable in $\Delta_3(<)$. More details about all these results can be found in [Pin96, DG08, DGK08].

In the setting of forests and unranked trees we already mentioned, $EF + F^{-1}$ [Boj07b], $\Delta_2(<_{\mathbf{v}})$ [BS08] and BC-$\Sigma_1(<_{\mathbf{v}})$ [BSS08]. These three characterizations use forest algebras. Recall that the three logics cannot express that a tree is of rank for some integer k. Therefore, as we already said, new characterizations are necessary for trees of bounded rank. Note that $\Delta_2(<_{\mathbf{v}})$ is highest logic in the quantification hierarchy for which we have a decidable characterization in the tree setting. Getting a decidable characterization for $\Sigma_2(<_{\mathbf{v}})$ remains an open problem

over trees. Decidable characterizations were also provided for the temporal logics EF and $EF + EX$ [BW06]. The modality EF is used to navigate to some descendant and the modality EX to navigate to some child. Several characterizations for temporal logics were presented in [Ési06, ÉI08a, ÉI08b]. In particular, an alternative decidable characterization for EF was presented in [ÉI08a]. However the characterizations provided for more expressive logics are not decidable. We also mention the locally threshold testable languages (LTT) which were given a decidable characterization in [BS09]. This characterization use a specific formalism that is partially used in our decidable characterization of LT. Another example of known decidable characterization would be the class of frontier testable languages which is a class of trees of bounded rank ([Wil96]). Note that the class of frontier testable languages is incomparable with all the classes we investigate in this thesis. In [Str10] a formalization of the concept of a "delay theorem" on trees is presented for some logics. In particular it uses this concept to give a new presentation of the characterization of LT over unranked trees that we will present in this thesis.

Organization. We divide the thesis in two parts. The first part is devoted to words and the second part to trees. The word part serves as an introduction to the difficult proofs we will present in the tree part. Note that some of the theorems we prove in the word part are used as black boxes in the tree part. Also note that while some proofs are just new presentations of ideas that were developed for the original proof of the characterization (for example the proof of the characterization of $\mathrm{FO}^2(<)$), some other proofs use new ideas (for example the proof of the characterization of $\mathrm{FO}^2(<, Succ)$).

Both parts follow the same outline. In the first chapters we give the definitions and notations relative to the setting under investigation. Then we have a chapter dedicated to the definition of all the formalisms we consider. Finally we devote a chapter for each characterization we prove.

Part I

Words

Chapter 1

Notations and Algebra for Words

In this chapter we introduce the various notions related to the word setting that we use in this first part. In particular, we define the notion of word as well as the notion of regular word language. In Section 1.1 we introduce the notion of word and give a few definitions that we use in the following chapters. The rest of the chapter is devoted to the class of regular word languages (REG). In Section 1.2, we give a first definition of REG using automata. In Section 1.3 we give the algebraic definition of REG using monoids and semigroups. Note that we postpone the logical definition of REG to the next chapter which we devote to logic.

1.1 Words

In this section we fix some notations that we will use in this whole part for words. An *alphabet* A is just a finite set. We call *labels* or *letters* its elements. Fix some alphabet A, a word over A is a finite sequence of letters of A. More formally, there exists a special word that we call the empty word, ϵ, and if w is a word, for any $a \in A$, wa is a word. Given some word w, its *length* is its number of letters, we write it $|w|$.

We use standard terminology for the notion of *position* in a word. We view a word $a_1 \cdots a_n$ of length n over A as a sequence of n positions that we write $1, \ldots, n$. These positions are labeled over A (i has label a_i). The notions of *next position*, *previous position*, *following position* and *preceding position* are also defined using standard terminology (See Figure 1.1). Given a word w of length n the next position of i_k is i_{k+1} and its preceding position i_{k-1}. For any $k' > k$, i'_k

is a following position of i_k and for any $k'' < k$, $i_{k''}$ is a preceding position of i_k. If w is a word, x is the position i_k in some word w and k' some integer, we write $x + k'$ the position $i_{k+k'}$.

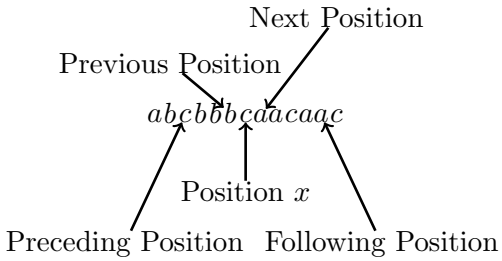


Figure 1.1: Definitions on Words

Factors of words. Given a word w, w' is a *factor* of w iff there exists two words w_1, w_2 such that $w = w_1 w' w_2$. When w' is a factor of w, we say that w' is a *prefix* iff w_1 is the empty word and that it is a *suffix* iff w_2 is the empty word. Given a word w and two positions x, y in w such that y is a following position of x, the factor of w between x and y is the word w' such that $w = w_1 w' w_2$ and x is the leftmost position of w' and y the leftmost position of w_2 (in particular, this means that y is not in w'). We write this factor $w[x, y]$. Given some word w we call *prefix at* x the prefix of w whose rightmost position is x and *suffix at* x the suffix of w whose leftmost position is x. All these definitions are illustrated in Figure 1.2.

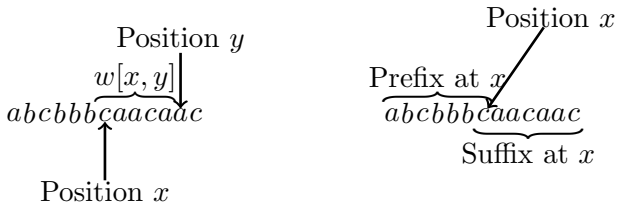


Figure 1.2: Illustration of the notions of factor, prefix and suffix over words

1.2 Regular Languages

Languages of words. Fix some alphabet A, a language over A is a set of words over A. For all alphabet A, we write A^* the language of all words over A including the empty word and A^+ the language of all words over A excluding the empty word.

Regular Languages. All classes of word languages we investigate are fragments of the class of regular word languages (REG). A word language is regular iff it is definable using a finite word automaton. We briefly recall the definition of finite automata. An automaton $\mathcal{A}$ is a tuple (A, Q, δ, q_i, Q_f) with:

- A an alphabet,
- Q a set of states,
- $q_i \in Q$ an initial state,
- $Q_f \subseteq Q$ a set of final states,
- $\delta : Q \times A \to Q$ a transition function.

Given a word $w = a_1 \dots a_n$, we say that w is accepted by $\mathcal{A}$ iff there exists a sequence of $n+1$ states $q_0 \dots q_n$ such that $q_0 = q_i$ and for all j, $\delta(q_{j-1}, a_j) = q_j$. The language L accepted by $\mathcal{A}$ is the set of words accepted by $\mathcal{A}$. We give an example of regular language below:

Example 1.1. *$L_e = \{w \in A^* \mid |w| = 0 \mod 2\}$, the language of words of even length. L_e is accepted by the automaton $\mathcal{A} = (A, Q, \delta, q_i, Q_f)$ with:*

- $Q = \{q_e, q_o\}$,
- $q_i = q_e$,
- $Q_f = \{q_i\}$,
- $\forall a \in A \left[\begin{array}{lcl} \delta(q_i, a) & = & q_e \\ \delta(q_e, a) & = & q_i \end{array} \right.$.

Recall that REG can equivalently be defined using MSO logic or finite monoids. We come back to monoids in the next section and postpone the definition of MSO to Chapter 2. From now on, all the word languages we consider are regular.

1.3 Algebra

We introduce here the minimal algebraic notions necessary for this thesis. More details can be found in [Pin96].

A *semigroup* is a pair $(S, \cdot)$ where S is a set and $\cdot$ an associative operation over S. A *monoid* is a semigroup $(M, \cdot)$ such that M contains some neutral element 1_M for the operation $\cdot$. When the operation is evident from the context we will just write S or M for a semigroup or a monoid.

Given some semigroup S, we write S^1 the following monoid: if S is a monoid, $S^1 = S$, otherwise $S^1 = S \cup \{1_S\}$ where 1_S acts as a neutral element for the operation $\cdot$.

Recognition of a language. Take some word alphabet A. Notice that the pair $(A^+, \cdot)$, where $\cdot$ is the concatenation operation, is a semigroup and that $(A^*, \cdot)$ is a monoid with the empty word as the neutral element. Given some semigroup S, we say that a language $L \subseteq A^+$ is recognized by S iff there exists a morphism $\alpha : A^+ \to S$ and a subset $F \subseteq S$ such that $L = \alpha^{-1}(F)$. Similarly, given some monoid M, we say that a language $L \subseteq A^*$ is recognized by M iff there exists a morphism $\alpha : A^* \to M$ and a subset $F \subseteq M$ such that $L = \alpha^{-1}(F)$.

A very well known result is that regular languages corresponds exactly to recognition by finite semigroups and monoids.

Proposition 1.1. *Fix $L \subseteq A^*$ a language over some alphabet A, the following properties are equivalent:*

1. *L is regular.*
2. *$L \cap A^+$ is recognized by a finite semigroup.*
3. *L is recognized by a finite monoid.*

Syntactic semigroup, syntactic monoid. We define the notion of syntactic semigroup of a given language L. This notion corresponds to the notion of minimal deterministic automaton is the automata view of regular languages. Fix some language L over an alphabet A. Given two words w, w' in A^+ we say that $w \sim_L w'$ iff for all words x, y in A^*, $xwy \in L$ iff $xw'y \in L$. The relation $\sim_L$ is a congruence, meaning that it is compatible with the concatenation operation $\cdot$. We call S_L the quotient of A^+ by this equivalence. The resulting semigroup recognizes L and is finite iff L is regular, it is called the syntactic semigroup of L. Similarly we can define a notion of syntactic monoid of L, M_L, using the same equivalence over the words of A^*.

Example 1.2. *Consider L_e the language of words of even length. We have $w \sim_{L_e} w'$ iff $|w| = |w'| \mod 2$. It follows that there are two elements in the syntactic monoid, M_e, of L_e, one corresponding to the words of even length and one to the words of odd length. We have $M_{L_e} = \{u_e, u_i\}$ with u_e as neutral element and $u_i \cdot u_i = u_e$.*

Idempotents. Given some semigroup $(S, \cdot)$, we say that $e \in S$ is idempotent iff $e^2 = e$. It is folklore that for any finite semigroup S, there exists a number $\omega(S)$ (denoted by ω when S is understood from the context) such that for each element s of S, s^ω is an idempotent.

Chapter 2

Fragments of First Order Logic over Words

All the logics we consider are actually fragments of first-order logic in terms of expressive power. We regroup their definitions in this chapter. An overview of these fragments that depicts the relations between them can be found in Figure 2.1 (all classes that are not linked in the figure are incomparable). Each logic that we investigate is given its own section. In Section 2.1 we introduce the definitions and notations of the logical point of view of regular languages. In particular we define first-order logic (FO) and monadic second-order logic (MSO) which characterizes regular languages. We also quote the decidable characterization of FO. The second section, Section 2.2, is devoted to the fragment of first-order logic using only two variables. We give the definitions of FO^2 as well as the definitions of the other fragments of FO that are equivalent to FO^2 in terms of expressive power. We do not state the decidable characterization of FO^2 in this section. We will study this characterization in depth, therefore we give it its own chapter, Chapter 3. In Section 2.3, we introduce the class of languages definable by a boolean combination of existential first-order formulas ($BC - \Sigma_1(<)$) and quote the associated decidable characterization. Finally in Section 2.4 we introduce the class of locally testable languages (LT). Again, we will study the decidable characterization of LT extensively, therefore we give it its own chapter, Chapter 4.

2.1 Monadic Second-order Logic and First-Order Logic

We view each word as a relational structure whose domain is its set of positions. The signature contains several predicates. For every label a in the alphabet A, we have a unary predicate P_a. We have two binary relations: $<$ for the following position relation and $Succ$ for the next position relation.

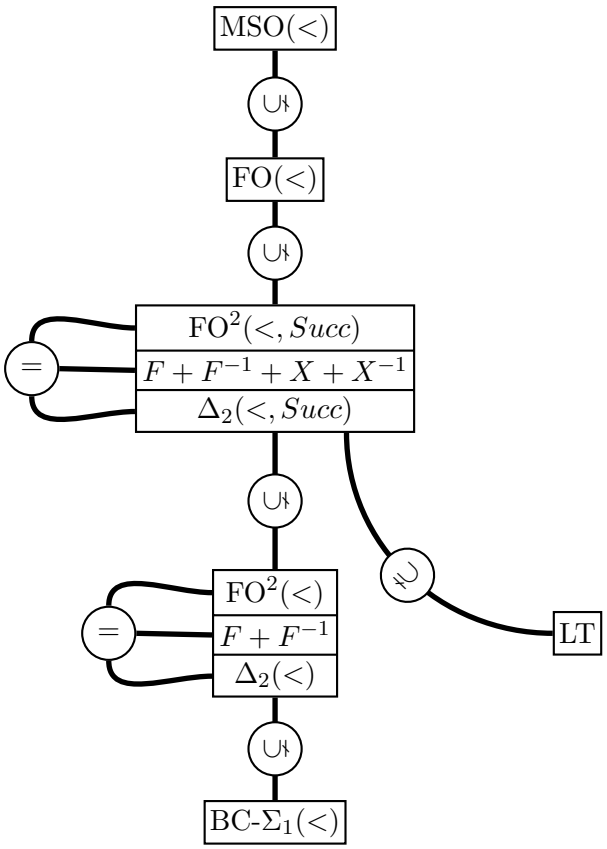


Figure 2.1: Overview of the Classes Under Investigation in the Word Part

FO. First-order logic, denoted FO(<), uses first order variables that we will write $x, y, z, \ldots$. A first order-formula is defined by the following grammar (x, y are first-order variables):

$$\varphi = \begin{array}{l} P_a(x) \text{ for } a \in A \mid x < y \mid Succ(x, y) \mid x = y \mid \exists x\ \varphi \mid \forall x\ \varphi \mid \\ \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \Rightarrow \varphi \end{array}$$

Given a first-order formula φ we call the set of its *free variables* the set of variables that are not under the scope of some quantifier $\forall$ or $\exists$. If φ has no free variables we say that it is *closed*. Notice that variables may be reused in the same formulas. For example the following formula is a valid first-order formula:

$$\forall x(P_a(x) \vee \exists y(y < x \wedge P_b(y) \wedge \exists x(x < y \wedge P_a(x))))$$

Fix a first-order formula φ with n free variables $x_1, \ldots, x_n$. We say that a word

w together with n positions $i_1, \ldots, i_n$ in w satisfy φ and write $w, i_1, \ldots, i_n \models \varphi$ iff:

- $\varphi = P_a(x_k)$ and i_k is labeled with the letter a.
- $\varphi = x_k = x_l$ and $i_l = i_k$.
- $\varphi = x_k < x_l$ and i_l is a following position of i_k.
- $\varphi = Succ(x_k, x_l)$ and i_l is the next position of i_k.
- $\varphi = \varphi_1 \wedge \varphi_2$ and $w, i_1, \ldots, i_n \models \varphi_1$ and $w, i_1, \ldots, i_n \models \varphi_2$.
- $\varphi = \varphi_1 \vee \varphi_2$ and $w, i_1, \ldots, i_n \models \varphi_1$ or $w, i_1, \ldots, i_n \models \varphi_2$.
- $\varphi = \neg\varphi'$ and $w, i_1, \ldots, i_n \models \varphi'$ is false.
- $\varphi = \varphi_1 \Rightarrow \varphi_2$ and $w, i_1, \ldots, i_n \models \neg\varphi_1 \vee \varphi_2$.
- $\varphi = \exists x_{n+1}\ \varphi'$ and there exists a position i_{n+1} in w such that: $w, i_1, \ldots, i_n, i_{n+1} \models \varphi'$.
- $\varphi = \forall x_{n+1}\ \varphi'$ and for all positions i_{n+1} in w we have: $w, i_1, \ldots, i_n, i_{n+1} \models \varphi'$.

We say a closed first-order formula φ defines a language L iff $L = \{w \mid w \models \varphi\}$. Notice that the successor relation is expressed by the following formula:

$$Succ(x, y) = x < y \wedge \neg\exists z(x < z \wedge z < y)$$

We quote below the decidable characterization of the class of word languages definable in FO(<).

Theorem 2.1. ([Sch65]) *Fix L some regular word language and M its syntactic monoid. L is definable in* FO(<) *iff* $\forall u \in M$*:*

$$u^\omega = u^{\omega+1} \tag{2.1}$$

Since the syntactic monoid of a regular language is a finite object and ω a computable number it is simple to check if the syntactic monoid of a regular language verifies (2.1). Therefore we get the following corollary:

Corollary 2.2. *It is decidable whether a regular word language is definable in* FO(<).

Example 2.1. *The language L_e of words of even length is not definable in* FO(<). *Indeed the syntactic monoid of L_e, $M_{L_e} = \{u_e, u_i\}$ is defined with the following operation:*

$\cdot$	u_e	u_i
u_e	u_e	u_i
u_i	u_i	u_e

It is simple to see that for all integer k, $u_i^{\omega} = u_i^{2\omega} = u_e$ and $u_i^{\omega+1} = u_i^{2\omega+1} = u_i$. It follows that M_{L_e} does not verify (2.1).

MSO. We view monadic second-order as an extension of first-order logic. We add a set of second-order variables that we write $X, Y, Z, \ldots$. These variables represent sets of positions. The grammar of FO is now extended by the following rules (with X a second-order variable and x a first-order variable):

$$\varphi = \exists X\ \varphi \mid \forall X\ \varphi \mid x \in X$$

Fix a monadic second-order formula φ with n free first-order variables $x_1, \ldots, x_n$ and m free second-order variables $X_1, \ldots, X_m$. We extend the semantic of FO in the following way: given n positions $i_1, \ldots, i_n$ in w and m sets of positions $I_1, \ldots, I_n$ we say that they satisfy φ and we write $w, i_1, \ldots, i_n, I_1, \ldots, I_n \models \varphi$ iff:

- $\varphi = \exists\ X_{n+1}\varphi'$ and there exists a set of positions I_{n+1} of w such that: $w, i_1, \ldots, i_n, I_1, \ldots, I_n, I_{n+1} \models \varphi'$.
- $\varphi = \forall\ X_{n+1}\varphi'$ and for all sets of positions I_{n+1} of w we have: $w, i_1, \ldots, i_n, I_1, \ldots, I_n, I_{n+1} \models \varphi'$.
- $\varphi = x_k \in X_l\varphi'$ and $i_k \in I_l$.

We say a closed MSO formula φ defines a language L iff $L = \{w \mid w \models \varphi\}$. A very well known result is that regular languages are exactly the languages that are definable in MSO(<):

Theorem 2.3. ([Bü60]) *A language L is definable in* MSO(<) *iff it is regular.*

Example 2.2. *The language L_e of words of even length is defined by the following* MSO(<) *formula (with* 1 *and n respectively the leftmost and rightmost positions in the word, both are easily defined in* MSO(<)*):*

$$\begin{aligned} \exists X\ 1 \in X \wedge n \notin X \wedge \forall x \quad & (x \in X \wedge (\forall y(\neg Succ(x,y) \vee y \notin X)) \\ \vee \quad & (x \notin X \wedge (\forall y(\neg Succ(x,y) \vee y \in X)) \end{aligned}$$

2.2 First-Order Logic Using Only Two Variables

This section is devoted to first-order logic using only two variables. We begin with the definition of the logics $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$ which are the two fragments we investigate in this section. Then we consider other fragments of FO that have the same expressive power as these two logics. The first one is unary temporal logic (UTL) and the second one is the restriction of FO to only one quantifier alternation (Δ_2).

$\mathbf{FO}^2(<)$. $\mathrm{FO}^2(<)$ is the two variables restriction of $\mathrm{FO}(<)$ using only the $<$ and $=$ predicates. This means that the formulas can only use two distinct variables. However these variables may be reused. For example the following formula is a formula of $\mathrm{FO}^2(<)$:

$$\forall x\ (P_a(x) \vee \exists y(y < x \wedge P_b(y) \wedge \exists x(x < y \wedge P_a(x))))$$

Example 2.3. *The language of words such that the leftmost position labeled with a is to the right of the leftmost position labeled with b is defined by the following* $\mathrm{FO}^2(<)$ *formula:*

$$\forall x\ (P_a(x) \wedge \neg(\exists y\ (P_a(y) \wedge y < x)) \Rightarrow \exists y\ (y < x \wedge P_b(y))$$

$\mathbf{FO}^2(<, Succ)$. Recall that it is possible to express the successor relation, $Succ$, using only the relation $<$ and three variables. However three variables are needed. Therefore, this is not possible in $\mathrm{FO}^2(<)$. We call $\mathrm{FO}^2(<, Succ)$ the extension of $\mathrm{FO}^2(<)$ with the predicate $Succ$.

Example 2.4. *The language L of words such that the leftmost factor ab is to the right of the leftmost factor ac is definable in* $\mathrm{FO}^2(<, Succ)$*. Consider the two formulas below:*

$$\begin{aligned} \varphi_{ab}(x) &= P_a(x) \wedge \exists y\ Succ(x, y) \wedge P_b(y) \\ \varphi_{ac}(x) &= P_a(x) \wedge \exists y\ Succ(x, y) \wedge P_c(y) \end{aligned}$$

L is defined by the following formula:

$$\forall x\ (\varphi_{ab}(x) \wedge \neg(\exists y\ (\varphi_{ab}(y) \wedge y < x)) \Rightarrow \exists y\ (y < x \wedge \varphi_{ac}(y))$$

Decidable characterizations for both $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$ were proved in [TW98]. We will investigate these two characterizations in depth in Chapter 3.

We turn to the definitions of other fragments of $\mathrm{FO}(<)$ that have the same expressive power as $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$. We begin with unary temporal logic (UTL).

2.2.1 Unary Temporal Logic

We define two temporal logics, $F + F^{-1}$ and $F + F^{-1} + X + X^{-1}$, corresponding respectively to $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$ in terms of expressive power. Note that these two logics are usually introduced with the names $LTL(F, F^{-1})$ and $LTL(F, F^{-1}, X, X^{-1})$. We choose to use different names in order to preserve symmetry with the notations we use in the next part on trees.

An $F + F^{-1} + X + X^{-1}$ formula φ over an an alphabet A is defined by the following grammar:

$$\varphi = a \in A \mid F\varphi \mid F^{-1}\varphi \mid X\varphi \mid X^{-1}\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \Rightarrow \varphi$$

An $F + F^{-1}$ formula is an $F + F^{-1} + X + X^{-1}$ formula that does not use X and X^{-1}. Given a word w, a position x in w and some $F + F^{-1} + X + X^{-1}$ formula φ, we say that w satisfies φ at position x and we write $w, x \models \varphi$ iff:

- $\varphi = a \in A$ and the position x in w is labeled by a.
- $\varphi = F\varphi'$ and there exists a position $x' > x$ such that $w, x' \models \varphi'$.
- $\varphi = F^{-1}\varphi'$ and there exists a position $x' < x$ such that $w, x' \models \varphi'$.
- $\varphi = X\varphi'$ and if x' is the position next to x, $w, x' \models \varphi'$.
- $\varphi = X^{-1}\varphi'$ and if x is the position next to x', $w, x' \models \varphi'$.
- $\varphi = \varphi_1 \wedge \varphi_2$ and $w, x \models \varphi_1$ and $w, x \models \varphi_2$
- $\varphi = \varphi_1 \vee \varphi_2$ and $w, x \models \varphi_1$ or $w, x \models \varphi_2$
- $\varphi = \neg\varphi_1$ and w, x do not satisfy φ_1.
- $\varphi = \varphi_1 \Rightarrow \varphi_2$ and $w, x \models \neg\varphi_1 \vee \varphi_2$.

If x is the leftmost position of the word w, we just say that w satisfies φ and we write $w \models \varphi$. We say that a language L over A is defined by a formula φ of $F + F^{-1}$ or $F + F^{-1} + X + X^{-1}$ iff $w \in L \Leftrightarrow w \models \varphi$.

Nesting Depth. Given a formula of UTL we call its nesting depth the maximal number of nested modalities F, F^{-1}, X, X^{-1} in the formula.

It turns out that the expressive power of $F + F^{-1}$ and $F + F^{-1} + X + X^{-1}$ corresponds exactly to the expressive power of respectively $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$. This result is taken from [EVW02]. We state it below and devote the rest of this section to its proof:

Theorem 2.4. ([EVW02]) *Fix L a regular language. We have the following two properties:*

- *L is definable in $\mathrm{FO}^2(<)$ iff L is definable in $F + F^{-1}$.*
- *L is definable in $\mathrm{FO}^2(<, Succ)$ iff L is definable in $F + F^{-1} + X + X^{-1}$.*

As this result is important for this thesis, we provide its proof below:

Proof. We concentrate on the direction from first order logic to temporal logic as the other direction is straightforward.

We show that for every $\mathrm{FO}^2(<)$ or $\mathrm{FO}^2(<, Succ)$ formula φ with one free variable x there exists an UTL formula $[\varphi]$ such that for every word w and every position i in w $w, i \models \varphi$ iff $w, i \models [\varphi]$. We proceed by induction on the size of the formula φ.

The base case is when $\varphi = P_a(x)$ for some $a \in A$. We naturally set $[\varphi] = a$. Boolean operators $\wedge$, $\vee$ and $\neg$ are treated in a straightforward manner. The only interesting case corresponds to quantification over a second variable y. Since $\forall$ as the same semantic as $\neg\exists\neg$ we can suppose that this quantification is existential. This means φ is of the form:

$$\varphi(x) = \exists y \ \Psi(x, y)$$

We first show that there exists a FO^2 formula that is equivalent to φ and is in a normal form. Intuitively, we want to delete all atomic formulas in Ψ that involve the variable x. This way we obtain a new formula with only one free variable y that can be translated into UTL by induction. It turns out that φ is equivalent to a disjunction of formulas of the following form:

$$\exists y \ \Psi_1(x, y) \wedge P_a(x) \wedge \Psi_2(y)$$

With Ψ_1 a conjunction of two atomic formulas that fixes the truth values of $x = y$ and $x < y$ (in the case of $\mathrm{FO}^2(<, Succ)$ there is a third atomic formula fixing the truth value of $Succ(x, y)$). Ψ_2 is the formula obtained from Ψ by replacing the atomic formulas involving x by their truth value fixed as by Ψ_1. therefore Ψ_2 has only one free variable and is of the same size as Ψ. Ψ is equivalent to the disjunction of all such formulas for all possible Ψ_1 and all possible labels a. Let $[\Psi_2]$ be the translation of Ψ_2 obtained by induction we show how to translate the

whole formula into UTL depending on Ψ_1 (we do this for all possible formulas Ψ_1 that are coherent). We first do this for $\mathrm{FO}^2(<)$:

$\Psi_1(x,y)$	$F + F^{-1}$ formula
$x = y$	$a \wedge [\Psi_2]$
$x < y$	$a \wedge F[\Psi_2]$
$y < x$	$a \wedge F^{-1}[\Psi_2]$

And for $\mathrm{FO}^2(<, Succ)$:

$\Psi_1(x,y)$	$F + F^{-1} + X + X^{-1}$ formula
$x = y$	$a \wedge [\Psi_2]$
$Succ(x,y)$	$a \wedge X[\Psi_2]$
$Succ(y,x)$	$a \wedge X^{-1}[\Psi_2]$
$x < y \wedge \neg Succ(x,y)$	$a \wedge XF[\Psi_2]$
$y < x \wedge \neg Succ(y,x)$	$a \wedge X^{-1}F^{-1}[\Psi_2]$

□

2.2.2 Quantifier Alternation

We define two logics, $\Delta_2(<)$ and $\Delta_2(<, Succ)$. They are fragments of $\mathrm{FO}(<)$ and they have the same expressive power as $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$. This is a difficult result that relies on the decidable characterizations of $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$ that we will present in Chapter 3.

$\Delta_2(<)$. We say that a language is definable in $\Delta_2(<)$ iff it is definable by two fragments of $\mathrm{FO}(<)$. A $\Sigma_2(<)$ formula is a $\mathrm{FO}(<)$ formula of the form:

$$\exists x_1 ... \exists x_n \ \forall y_1 ... \forall y_m \ \varphi(x_1, ..., x_n, y_1, ..., y_m)$$

Where φ is a $\mathrm{FO}(<)$ quantifier free formula using only $=$ and $<$. A $\Pi_2(<)$ formula is the negation of a $\Sigma_2(<)$ formula, meaning a formula of the form:

$$\forall x_1 ... \forall x_n \ \exists y_1 ... \exists y_m \ \varphi(x_1, ..., x_n, y_1, ..., y_m)$$

Where φ is a $\mathrm{FO}(<)$ quantifier free formula using only $=$ and $<$. We say a language is definable in $\Delta_2(<)$ if it is definable by both a $\Sigma_2(<)$ formula and a $\Pi_2(<)$ formula.

$\Delta_2(<, Succ)$**.** Similarly we define $\Pi_2(<, Succ)$ and $\Sigma_2(<, Succ)$ by adding the successor predicate, $Succ$ to $\Pi_2(<)$ and $\Sigma_2(<)$. A language is then definable in $\Delta_2(<, Succ)$ iff its is definable by both a $\Sigma_2(<, Succ)$ formula and a $\Pi_2(<, Succ)$ formula.

Example 2.5. *The language of words such that the leftmost position labeled with a is to the right of the leftmost position labeled with b is defined by the following $\Sigma_2(<)$ and $\Pi_2(<)$ formulas:*

$$\exists x \forall y (P_b(x) \wedge (P_a(y) \Rightarrow x < y))$$
$$\forall x \exists y (P_a(x) \Rightarrow (x < y \wedge P_b(x)))$$

It turns out that these formalisms are equivalent to $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$ in terms of expressive power:

Theorem 2.5. ([PW97, TW98]) *Fix L a regular language.*

- *L is definable in $\mathrm{FO}^2(<)$ iff L is definable in $\Delta_2(<)$.*
- *L is definable in $\mathrm{FO}^2(<, Succ)$ iff L is definable in $\Delta_2(<, Succ)$.*

This result is proved using the decidable characterizations of $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$ that we will present in Chapter 3 and the decidable characterizations of $\Delta_2(<)$ and $\Delta_2(<, Succ)$ presented in [PW97]. It turns out that the characterizations for $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$ are the same as the characterizations for $\Delta_2(<)$ and $\Delta_2(<, Succ)$. This implies the equality of their expressive power. We quote the characterizations of $\Delta_2(<)$ and $\Delta_2(<, Succ)$ below:

Theorem 2.6. ([PW97, Alm96]) *Fix L a regular language. Then L is definable in $\Delta_2(<)$ iff its syntactic monoid M verifies for all $u, v \in M$:*

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \tag{2.2}$$

L is definable in $\Delta_2(<, Succ)$ iff its syntactic semigroup S verifies for all $u, v, e \in S$ with e idempotent:

$$(eueve)^\omega = (eueve)^\omega v (eueve)^\omega \tag{2.3}$$

A simple corollary of Theorem 2.6 is that definability in $\Delta_2(<)$ and $\Delta_2(<, Succ)$ are decidable properties:

Corollary 2.7. *It is decidable whether a regular word language is definable in $\Delta_2(<)$. It is decidable whether a regular word language is definable in $\Delta_2(<, Succ)$.*

2.3 Boolean Combinations of Existential First-Order Formulas

Boolean combinations of existential first order formulas (BC-$\Sigma_1(<)$) is another restriction of FO($<$). A $\Sigma_1(<)$ is a first order formula of the form:

$$\exists x_1 \dots \exists x_n \varphi(x_1, ..., x_n)$$

With φ a quantifier free formula FO($<$) formula using only $<$ and $=$. A language is definable in BC-$\Sigma_1(<)$ iff it is definable by a boolean combination of such formulas.

Example 2.6. *Consider the language L of words that contain a label a to the right of a label b but no c to the right of a label a. L is defined by the following BC-$\Sigma_1(<)$ formula:*

$$(\exists x \exists y \ x < y \wedge P_b(x) \wedge P_a(y)) \wedge \neg(\exists x \exists y \ x < y \wedge P_a(x) \wedge P_c(y))$$

We quote the decidable characterization of BC-$\Sigma_1(<)$ below. This is a result from [Sim75].

Theorem 2.8. ([Sim75]) *Fix L a regular language. Then L is definable in BC-$\Sigma_1(<)$ iff its syntactic monoid M verifies for all $u, v \in M$:*

$$(uv)^\omega = (uv)^\omega u = v(uv)^\omega \tag{2.4}$$

A simple corollary of Theorem 2.8 is that definability in BC-$\Sigma_1(<)$ is a decidable property:

Corollary 2.9. *It is decidable whether a regular word language is definable in BC-$\Sigma_1(<)$.*

2.4 Locally Testable Languages

We define the class of *locally testable languages* (LT). Locally testable languages are languages whose membership for a word depends only on the presence or absence of some fixed set of neighborhoods.

Types. We begin by making the notion of neighborhoods more precise. Fix some integer k. Let w be a word and x be a position in w, let w_s be the suffix at x of w, the *k-type* of x is the prefix of length k of w_s if w_s is of length at least k and w_s otherwise. When k will be clear from the context we will simply say *type*. A k-type τ *occur* in a word w if there exists a position of w of type τ. If w' is a

factor $t[x, y]$ of some word w and some positions x, y of w, then the k-type of a position of w' is the k-type of the corresponding position in w. Notice that the k-type of a position of w' may depend on w.

Given two words w and w' we denote by $w \preccurlyeq_k w'$ the fact that all k-types that occur in w also occur in w'. We denote by $w \simeq_k w'$ the property that the leftmost positions of w and w' have the same k-type and w and w' agree on their k-types: $w \preccurlyeq_k w'$ and $w' \preccurlyeq_k w$. Note that when k is fixed the number of k-types is finite and hence the equivalence relation $\simeq_k$ has a finite number of equivalence classes.

A language L is said to be κ-locally testable if L is a union of equivalence classes of $\simeq_\kappa$. A language is said to be *locally testable* (is in LT) if there is a κ such that it is κ-locally testable. In words this says that in order to test whether a word w belongs to L it is enough to check for the presence or absence of κ-types in w, for some big enough κ. An simple example of language that is in LT would be the language of words that contain the factor *aba* iff they contain the factor *ccb*.

There exists a decidable characterization of the class LT given in [BS73] and [McN74]. However, we will study this characterization in details and give it its own chapter. Therefore, as we did for FO^2 we postpone its statement to Chapter 4.

Logical characterization There exists a logical characterization of the class LT. It corresponds to the languages definable by a temporal logic defined on the following grammar:

$$\varphi = a \in A \mid X\varphi \mid G\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi$$

Intuitively, **G** stands for "everywhere in the word" while **X** stands for "next position." Formally this gives the following semantic: Given a word w and a position x in w we say that w satisfies a formula φ and we write $w, x \models \varphi$ iff:

- $\varphi = a \in A$ and the position x in w is labeled by a.
- $\varphi = G\varphi'$ and for all positions x' in w, $w, x' \models \varphi'$.
- $\varphi = X\varphi'$ and if x' is the position next to x, $w, x' \models \varphi'$.
- $\varphi = \varphi_1 \wedge \varphi_2$ and $w, x \models \varphi_1$ and $w, x \models \varphi_2$
- $\varphi = \varphi_1 \vee \varphi_2$ and $w, x \models \varphi_1$ or $w, x \models \varphi_2$
- $\varphi = \neg\varphi_1$ and w, x do not satisfy φ_1.

Chapter 3

First-Order Logic With Only Two Variables over Words

In this chapter we investigate the logics $\mathrm{FO}^2(<)$ and $\mathrm{FO}^2(<, Succ)$. Recall that these two logics are restriction of $\mathrm{FO}(<)$ to only two variables that may be reused. More details can be found in Chapter 2. This chapter is devoted to achieving the following goals:

- Obtain a decidable characterization for $\mathrm{FO}^2(<)$.
- Obtain a decidable characterization for $\mathrm{FO}^2(<, Succ)$.

These two goals were originally achieved in [TW98] using algebraic characterizations. In this chapter, we present the characterizations of [TW98] and provide new proofs for them. We do this for two main reasons. First, we use both characterizations as blackboxes in the second part on trees. Second, the techniques involved in these proofs can be seen as a restriction to the setting of words of the techniques we will use in the setting of trees.

Note that the proof we provide for the characterization of $\mathrm{FO}^2(<)$ is simply a new presentation of the ideas used in [TW98]. However, the proof of the characterization of $\mathrm{FO}^2(<, Succ)$ is somewhat different. The proof of [TW98] relied on a difficult result of [Alm96], while our proof is direct.

Note that there are many more characterizations of $\mathrm{FO}^2(<)$ that are a priori not decidable. For example the languages definable in $\mathrm{FO}^2(<)$ are exactly the languages accepted by a partially ordered two-way finite automaton. A presentation of all these characterizations can be found in [TT02].

The chapter is organized as follows. In Section 3.1 we define tools that will be crucial in our proofs. Section 3.2 is devoted to $\mathrm{FO}^2(<)$ and Section 3.3 is devoted to $\mathrm{FO}^2(<, Succ)$.

3.1 Definitions

In this section we define two notions that are pivotal to the proofs of the characterizations. First we define Ehrenfeucht-Fraïssé games. Ehrenfeucht-Fraïssé games characterize the expressive power of a logic. Then we define two orders on semigroups and monoids that will play a key role in the inductions used in the proofs.

Ehrenfeucht-Fraïssé games. Based on Theorem 2.4, we use an $F + F^{-1}$,$F + F^{-1} + X + X^{-1}$ point of view and define two games corresponding intuitively to these two logics. We first define the Ehrenfeucht-Fraïssé game for $F + F^{-1}$. There are two players, Duplicator and Spoiler, the board consists in two words and the players agree on the number of moves in advance. At any time there is one pebble placed on a position of each of the two words and the corresponding positions have the same label in A. At the beginning of the game the two pebbles are placed on the leftmost position of each word. At each step Spoiler moves one of the pebbles to another positions in the word. Duplicator must respond by moving the other pebble in the same direction to a node of the same label. For example if Spoiler moved the pebble on a node to the right and labeled with a, then Duplicator must also move the pebble to the right on a node labeled with a. If Duplicator cannot move then Spoiler wins. For $F + F^{-1} + X + X^{-1}$, there is an added condition for Duplicator's answer. If Spoiler plays on a next or previous position, then Duplicator has to answer on a next or previous position We speak of the $F + F^{-1}$ Ehrenfeucht-Fraïssé game and the $F + F^{-1} + X + X^{-1}$ Ehrenfeucht-Fraïssé game, when the logic is clear from the context we just write Ehrenfeucht-Fraïssé game. We say that Duplicator has a wining strategy for the k rounds Ehrenfeucht-Fraïssé game if there exists a function that tells her how to play depending on Spoiler's moves such that she wins the game if she plays according to this function. Given two words w, w', we write $w \simeq_k w'$ the fact that Duplicator has a winning strategy for the k-rounds $F + F^{-1}$ game on w and w'. We write $w \simeq_k^+ w'$ the fact that Duplicator has a winning strategy for the k-rounds $F + F^{-1} + X + X^{-1}$ game on w and w'. The following Lemma can be proved using classical techniques (recall that the nesting depth of a UTL formula is the maximal number of nested modalities F, F^{-1}, X, X^{-1} in the formula):

Lemma 3.1. *Fix two words w, w' and some integer k, the two following propositions hold:*

- *$w \simeq_k w'$ iff w and w' satisfy the same $F + F^{-1}$ formulas of nesting depth k.*
- *$w \simeq_k^+ w'$ iff w and w' satisfy the same $F + F^{-1} + X + X^{-1}$ formulas of nesting depth k.*

Reachability. We define two relations on semigroups (therefore these relations are also defined on monoids). These relations will later play a key role in our inductions. Note that these relations are usually introduced as the *Green relations*. In order to stay close to the proofs we will present on trees in the second part, we give a slightly different definition. Therefore, in order to avoid confusion, we give them a different name. Fix some semigroup S and $u, v \in S$. We say that u is r-reachable from v iff there exists $v' \in S$ such that $u = vv'$. Symmetrically we say that u is l-reachable from v iff there exists $v' \in S$ such that $u = v'v$.

3.2 FO$^2(<)$

In this section we give a decidable characterization of FO$^2(<)$ using monoids. This characterization was first presented in [TW98]. While the proof we provide uses the same ideas as the one of [TW98], the presentation is slightly different in order to be extended later to the tree setting.

Theorem 3.2. ([TW98]) *A regular language L over an alphabet A is definable in* FO$^2(<)$ *iff its syntactic monoid M verifies, for all $u, v \in M$:*

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \tag{2.2}$$

Notice that this characterization is the same as the one of Theorem 2.6 for $\Delta_2(<)$. Therefore Theorem 3.2 yields the FO$^2(<)$ part of Theorem 2.5: $\Delta_2(<)$ and FO$^2(<)$ have the same expressive power. Given Theorem 3.9, deciding definability in FO$^2(<)$ becomes a simple matter. It is sufficient to compute the syntactic monoid of the input regular language and then we check if it satisfies Equation (2.2). Therefore we get the following corollary:

Corollary 3.3. *It is decidable whether a regular word language is definable in* FO$^2(<)$.

We devote the rest of this section to the proof of Theorem 3.2. We have to prove both directions of the Theorem. We begin with the easier "only if" direction.

Proposition 3.4. *The syntactic monoid of a language L definable in* FO$^2(<)$ *verifies* (2.2).

Proof. Rather than using the FO$^2(<)$ point of view, in regards to Theorem 2.4 we use the $F + F^{-1}$ point of view and show than the syntactic monoid of a language definable in $F + F^{-1}$ verifies (2.2).

Consider some language L that is defined by an $F + F^{-1}$ formula φ. Let k be the nesting depth of modalities in φ and M be the syntactic monoid of L with α

the associated morphism. We show that for all $u, v \in M$, $(uv)^{2k} = (uv)^k v (uv)^k$, which concludes the proof. The proof is an Ehrenfeucht-Fraïssé argument.

Consider w_1, w_2 two words and p, q two words such that $\alpha(q) = u$ and $\alpha(q) = v$. We show that Duplicator wins the k-move game between $w_1(pq)^k(pq)^k w_2$ and $w_1(pq)^k q(pq)^k w_2$. Since φ is of nesting depth k, it then follows from Lemma 3.1 that $w_1(pq)^k(pq)^k w_2 \in L$ iff $w_1(pq)^k q(pq)^k w_2 \in L$. By definition of the syntactic monoid, the result then follows.

We finish the proof by giving a winning strategy for Duplicator. To simplify the arguments we suppose that p and q are just letters and that w_1 and w_2 are empty words. The extension of the strategy in the general case is straightforward. Therefore the two words composing the board are of the form $(ab)^k b (ab)^k$ and $(ab)^k (ab)^k$ with a and b letters. In order to give the strategy, we need to give a few extra definitions. Consider some position x on the board. We call the *right number* (resp. *left number*) of x the number of full copies of the factor ab to the right (resp. left) of x. Given a position x in $(ab)^k b (ab)^k$ and a position y in $(ab)^k (ab)^k$ we that they are *identical* iff they are either both on the $(ab)^k$ prefix of the words or both in the $(ab)^k$ suffix of the words at the same position (see Figure 3.1).

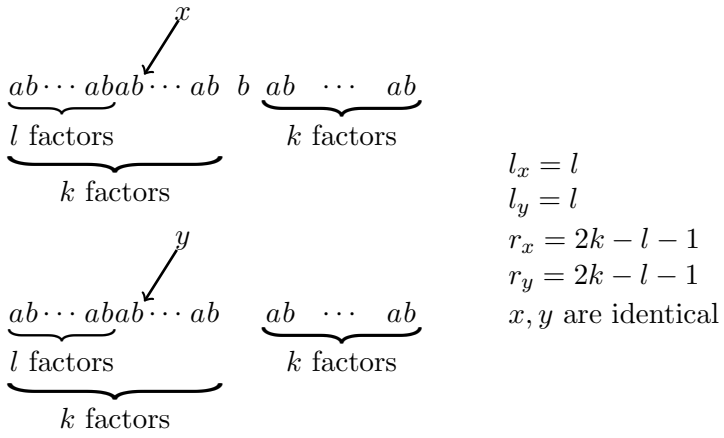


Figure 3.1: Definitions for the strategy in Proposition 3.4

We define a property $\mathcal{P}(i)$ depending on some integer i. We then show that Duplicator can preserve this property while playing, where i is the number of turns left to play. Suppose that the pebbles are in positions x and y on the board, we write r_x, r_y the right numbers of x, y and l_x, l_y their left numbers. $\mathcal{P}(i)$ holds iff one the following properties holds:

1. x, y are identical.

or

2. $l_x, l_y, r_x, r_y \geq i$.

At the beginning of the game, x and y are identical, therefore $\mathcal{P}(k)$ holds. Now suppose that there are i turns left to play, that the pebbles are in positions x and y and that $\mathcal{P}(i)$ holds. Suppose that Spoiler moves the pebble at position x to a new position x', depending on the item holding for $\mathcal{P}(i)$ and Spoiler's move, we Duplicator's strategy in order to make $\mathcal{P}(i-1)$ true.

x, y were identical. There are two cases. If Spoiler did not move on the central b in $(ab)^k b(ab)^k$, Duplicator just answers on a position y' that is identical to x' and $\mathcal{P}(i-1)$ holds for the first item. If Spoiler moves to the central b in $(ab)^k b(ab)^k$, then we have $r_{x'} = l_{x'} = k \geq i > i-1$. If $x < x'$, Duplicator answers on the leftmost b of the suffix $(ab)^k$. If $x' < x$, Duplicator answers on the rightmost b of the prefix $(ab)^k$. We then have $r_{y'}, l_{y'} \geq k-1 \geq i-1$. $\mathcal{P}(i-1)$ holds for the second item.

We had $l_x, l_y, r_x, r_y \geq i$. There are two cases. If $l_{x'}, r_{x'} \geq i$. Since we had $l_y, r_y \geq i-1$, Duplicator can find a position y' such that $l_{y'}, r_{y'} \geq i-1$ in both directions. Depending on the direction of Spoiler's move she chooses one of them as her answer and $\mathcal{P}(i-1)$ holds for the second item. Otherwise, suppose that $l_{x'} < i-1$ (the case where $r_{x'} < i-1$ is symmetric). Because $l_x \geq i$, $x' < x$, also since $l_y \geq i$ there exists $y' < y$ that is identical to x', this is Duplicator's answer. $\mathcal{P}(i-1)$ is then verified for the first item.

□

We turn to the other direction of Theorem 3.2. Fix a finite monoid M satisfying (2.2) and a morphism $\alpha : A^* \to M$. Theorem 3.2 is a consequence of the following Proposition:

Proposition 3.5. *For all $u_r, u_l, v \in M$, $L^v_{u_l,u_r} = \{w \mid u_l\alpha(w)u_r = v\}$ is definable in* FO$^2(<)$.

Indeed, if we fix $u_r, u_l = 1_M$, we get that for all $v \in M$, $\alpha^{-1}(v)$ is definable in FO$^2(<)$. Therefore any language recognized by M is definable in FO$^2(<)$, which concludes the proof of Theorem 3.2. We finish with the proof of Proposition 3.5. We construct a formula that defines $L^v_{u_l,u_r}$ using an induction mechanism. Therefore we need a way to concatenate two languages definable in FO$^2(<)$ into a single language that is also definable in FO$^2(<)$. While this not possible in the general case, we use the following weaker Lemma:

Lemma 3.6 (Concatenation Principle)**.** *Let L and K be two languages that are definable in* FO$^2(<)$*. Let $\varphi(x)$ be a* FO$^2(<)$ *formula with one free variable x with the following property: for all words w there exists at most one position i such that w, i satisfies $\varphi(x)$. (Notice that this property is semantic and might not be apparent just by looking at the syntax of the formula). The following language is definable in* FO$^2(<)$*:*

$$\{w = a_1 \dots a_{i-1} a_i a_{i+1} \dots a_n \mid a_1 \dots a_{i-1} \in K, a_{i+1} \dots a_n \in L \text{ and } w, i \models \varphi(x)\}$$

Proof. This is done by relativizing the quantifiers inside the formula recognizing L and the formula recognizing K respectively to the position before and after the position selected by φ. □

We are now ready for the proof of Proposition 3.5:

Proof. We proceed by induction on the three following parameters:

1. The size of the alphabet A.
2. The number of elements of M that are r-reachable from u_l.
3. The number of elements of M that are l-reachable from u_r.

Note that since M is finite all this parameters are well defined. We consider three cases depending on the elements of M that are l-reachable from u_r or r-reachable from u_l. In the first case we suppose, that there exists elements of M that are r-reachable from u_l but not the other way around. We conclude by induction on parameters 1) and 2). In the second case, we suppose that there exists elements of M that are l-reachable from u_r but not the other way around. We conclude by induction on parameters 1) and 3). Finally in the last case, we conclude by showing that if neither of the two previous cases hold, then $L^v_{u_l,u_r}$ is either empty or universal. Since both are easily defined in FO$^2(<)$ this ends the proof.

Case 1: There exists u such that u_l is not r-reachable from $u_l u$. We proceed as follows: first we show that given any word w, we can detect in FO$^2(<)$ the leftmost position x in w such that the prefix at x, w' is such that u_l is not r-reachable from $u_l \alpha(w')$ (if such a position exists). We show that it corresponds to the first occurrence of some letter a. We then finish the proof by induction on the alphabet size on the left of x and by induction on the second parameter on the right of x using Lemma 3.6 to merge the two sides. This structure is shown in Figure 3.2.

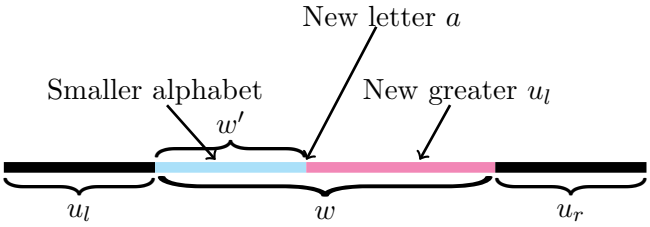


Figure 3.2: Arguments in the first case of Proposition 3.5

Lemma 3.7. *There exists some set of labels B such that:*

- *For any letter $b \in B$ u_l is not r-reachable from $u_l\alpha(b)$.*
- *For any word w such that u_l is not r-reachable from $u_l\alpha(w)$, the smallest prefix w' of w such that u_l is not r-reachable from $u_l\alpha(w')$ has only its rightmost letter in B.*

Proof. We choose B as the set of labels such that for all $b \in B$ u_l is not r-reachable from $u_l\alpha(b)$. By definition, B satisfies the first item.

Now fix some word w such that u_l is not r-reachable from $u_l\alpha(w)$, and w_1 the smallest prefix of w such that u_l is not r-reachable from $u_l\alpha(w_1)$. Let $b \in A$ and $w_2 \in A^*$ such that $w_1 = w_2 b$. We show that $b \in B$ and that no letter of w_2 is in B which ends the proof.

We show a slightly more general result, for all $u \in M$, u_l is not r-reachable from $u_l u\alpha(b)$. Indeed, fix some u and suppose that u_l is r-reachable from $u_l u\alpha(b)$, there exists u' such that $u_l = u_l u\alpha(b)u'$. By hypothesis on w_2 there exists v such that $u_l\alpha(w_2)v = u_l$

$$\begin{array}{lcl}
u_l\alpha(w_2) & = & u_l\alpha(w_2) \cdot vu\alpha(b)u'\alpha(w_2) \\
u_l\alpha(w_2) & = & u_l\alpha(w_2) \cdot (vu\alpha(b)u'\alpha(w_2))^\omega \\
u_l\alpha(w_2) & = & u_l\alpha(w_2) \cdot e \quad \text{With fix } e = (vu\alpha(b)u'\alpha(w_2))^\omega \\
u_l\alpha(w_2) & = & u_l\alpha(w_2) \cdot e \cdot \alpha(b)u'\alpha(w_2) \cdot e \quad \text{using (2.2)} \\
u_l\alpha(w_2) & = & u_l\alpha(w_2) \cdot \alpha(b)u'\alpha(w_2) \cdot e \\
u_l & = & u_l\alpha(w_1) \cdot u'\alpha(w_2) \cdot e \cdot v
\end{array}$$

Which means that u_l is r-reachable from $u_l\alpha(w_1)$ which is a contradiction. It follows that $b \in B$ (consider $u = \epsilon$), and that w_2 contains no b (u_l is r-reachable from $u_l\alpha(w_2)$). □

It follows from Lemma 3.7 and the hypothesis of this case that B is not empty. Therefore, by induction on the size of the alphabet the restriction of $L^v_{u_l,u_r}$ to $A-B$

is definable in FO$^2(<)$. Therefore we can suppose without loss of generality that all words of $L^v_{u_l,u_r}$ contain letters of B.

For all $u \in M$ such that u_l is r-reachable from $u_l u$, consider the languages $K^u_{u_l,1_M}$. By definition and Lemma 3.7 they contain no letter in B, therefore by induction on the size of the alphabet they are definable in FO$^2(<)$. For all $u' \in M$ such that u_l is not r-reachable from $u_l u'$, consider the languages $L^v_{u_l u',u_r}$. By definition there are less elements of M that are r-reachable from $u_l u'$ than from u_l. Therefore, by induction on the second parameter the languages $L^v_{u_l u',u_r}$ are definable in FO$^2(<)$. By Lemma 3.7, the language $L^v_{u_l,u_r}$ is a union of languages of the form:

$$K^u_{u_l,1_M} b L^v_{u_l u',u_r} \text{ for } u\alpha(b) = u'$$

If we fix $K = K^u_{u_l,1_M}$, $L = L^v_{u_l u',u_r}$ and φ the formula selecting the leftmost position labeled with b, it is a simple consequence of Lemma 3.6 that this language is definable in FO$^2(<)$. This ends the proof of this case.

Case 2: There exists u such that u_r is not l-reachable from uu_r. This case is symmetric to the previous one. We proceed as follows, first we show that given any word w, we can detect in FO$^2(<)$ the rightmost position x in w such that the suffix at x, w' is such that u_r is not l-reachable from $\alpha(w')u_r$ (if such a position exists). We show that it corresponds to the first occurrence of some letter a. We then finish the proof by induction on the alphabet size on the right of x and by induction on the third parameter on the left of x using Lemma 3.6 to merge the two sides. This structure is shown in Figure 3.3.

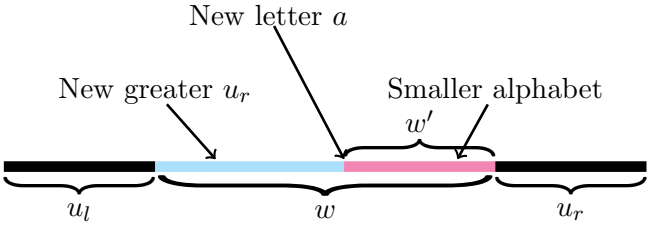


Figure 3.3: Arguments in the second case of Proposition 3.5

Lemma 3.8. *There exists some set of labels B such that:*

- *For any letter $b \in B$ u_r is not l-reachable from $\alpha(b)u_r$.*
- *For any word w such that u_r is not l-reachable from $\alpha(w)u_r$, the smallest suffix w' of w such that u_r is not l-reachable from $\alpha(w')u_r$ has only its leftmost letter in B.*

Proof. We choose B as the set of labels such that for all $b \in B$ u_r is not l-reachable from $\alpha(b)u_l$. By definition, B satisfies the first item.

Now fix some word w such that u_r is not l-reachable from $\alpha(w)u_r$, and w_1 the smallest suffix of w such that u_r is not l-reachable from $\alpha(w_1)u_r$. Let $b \in A$ and $w_2 \in A^*$ such that $w_1 = bw_2$. We show that $b \in B$ and that no letter of w_2 is in B which ends the proof.

We show a slightly more general result, for all $u \in M$, u_r is not l-reachable from $\alpha(b)uu_r$. Indeed, fix some u and suppose that u_r is l-reachable from $\alpha(b)uu_r$, there exists u' such that $u_r = u'\alpha(b)uu_r$. By hypothesis on w_2 there exists v such that $v\alpha(w_2)u_l = u_l$

$$\begin{array}{lcl}
\alpha(w_2)u_r & = & \alpha(w_2)u'\alpha(b)uv \cdot \alpha(w_2)u_r \\
\alpha(w_2)u_r & = & (\alpha(w_2)u'\alpha(b)uv)^{\omega+1} \cdot \alpha(w_2)u_r \\
\alpha(w_2)u_r & = & \alpha(w_2)u'\alpha(b) \cdot (uv\alpha(w_2)u'\alpha(b))^{\omega} \cdot uv \cdot \alpha(w_2)u_r \\
\alpha(w_2)u_r & = & \alpha(w_2)u'\alpha(b) \cdot e \cdot uv \cdot \alpha(w_2)u_r \quad \text{with } e = (uv\alpha(w_2)u'\alpha(b))^{\omega} \\
\alpha(w_2)u_r & = & \alpha(w_2)u'\alpha(b) \cdot e \cdot \alpha(w_2)u'\alpha(b) \cdot e \cdot uv \cdot \alpha(w_2)u_r \quad \text{using (2.2)} \\
\alpha(w_2)u_r & = & (\alpha(w_2)u'\alpha(b)uv)^{\omega} \cdot \alpha(w_2)u'\alpha(b) \cdot (\alpha(w_2)u'\alpha(b)uv)^{\omega+1} \cdot \alpha(w_2)u_r \\
\alpha(w_2)u_r & = & (\alpha(w_2)u'\alpha(b)uv)^{\omega} \cdot \alpha(w_2)u'\alpha(b) \cdot \alpha(w_2)u_r \\
u_r & = & v \cdot (\alpha(w_2)u'\alpha(b)uv)^{\omega} \cdot \alpha(w_2)u' \cdot \alpha(w_1)u_r
\end{array}$$

Which means that u_r is l-reachable from $\alpha(w_1)u_r$ which is a contradiction. It follows that $b \in B$ (consider $u = \epsilon$), and that w_2 contains no b (u_r is l-reachable from $\alpha(w_2)u_r$). □

It follows from Lemma 3.8 and the hypothesis of this case that B is not empty. Therefore, by induction on the size of the alphabet the restriction of $L^v_{u_l,u_r}$ to $A-B$ is definable in FO$^2(<)$. Therefore we can suppose without loss of generality that all words of $L^v_{u_l,u_r}$ contain letters of B.

For all $u \in M$ such that u_r is l-reachable from uu_r, consider the languages $K^u_{1_M,u_r}$. By definition and Lemma 3.8 they contain no letter in B, therefore by induction on the size of the alphabet they are definable in FO$^2(<)$.

For all $u' \in M$ such that u_r is not l-reachable from $u'u_r$, consider the languages $L^v_{u_l,u'u_r}$. By definition there are less elements of M that are l-reachable from $u'u_r$ than from u_r. Therefore, by induction on the third parameter the languages $L^v_{u_l,u'u_r}$ are definable in FO$^2(<)$.

By Lemma 3.8, the language $L^v_{u_l,u_r}$ is a union of languages of the form:

$$L^v_{u_l,u'u_r} b K^u_{1_M,u_r} \text{ for } \alpha(b)u = u'$$

If we fix $L = K^u_{1_M,u_r}$, $K = L^v_{u_l,u'u_r}$ and φ the formula selecting the rightmost position labeled with b, it is a simple consequence of Lemma 3.6 that this language is definable in FO$^2(<)$. This ends the proof of this case.

Case 3: Induction Base We are not in any of the two previous cases, therefore, for all u, u_l is r-reachable from $u_l u$ and u_r is l-reachable from uu_r. We show that $L^v_{u_l,u_r}$ is either A^* or the empty set. Since both are easily defined in FO$^2(<)$, this concludes the proof. we show that for all $u, u' \in M$ $u_l u u_r = u_l u' u_r$.

Indeed, by hypothesis there exists $u_1 \in M$ such that $u_l u u_r u_1 = u_l u' u_r$ and there exists $u_2 \in M$ such that $u_l u u_r = u_2 u_l u' u_r$. We get:

$$\begin{array}{lcll} u_l u u_r & = & u_2 u_l u u_r u_1 & \\ u_l u u_r & = & (u_2)^\omega u_l u u_r (u_1)^\omega & \\ u_l u u_r u_1 & = & (u_2)^\omega u_l u u_r (u_1)^{\omega+1} & \\ u_l u u_r u_1 & = & (u_2)^\omega u_l u u_r (u_1)^\omega & \text{using (2.1)} \\ u_l u' u_r & = & u_l u u_r & \end{array}$$

This ends the proof. Notice that we actually only use Equation (2.1) which is a simple consequence of Equation (2.2). □

3.3 FO$^2(<, Succ)$

In this section we give a decidable characterization for the class of languages definable in FO$^2(<, Succ)$ using semigroups. This characterization was first presented in [TW98]. However the proof was using difficult results from [Alm96]. The proof we present here is direct.

Theorem 3.9. *A regular word language L over an alphabet A is definable in* FO$^2(<, Succ)$ *iff its syntactic semigroup S verifies, for all $u, v, e \in S$ with e idempotent:*

$$(eueve)^\omega = (eueve)^\omega v (eueve)^\omega \tag{2.3}$$

Notice that this characterization is identical to the one of Theorem 2.6 for $\Delta_2(<, Succ)$. Therefore Theorem 3.9 yields the FO$^2(<, Succ)$ part of Theorem 2.5: FO$^2(<, Succ)$ and $\Delta_2(<, Succ)$ have the same expressive power over words. Again, a simple consequence of Theorem 3.9 is that in order to check definability of a regular language in FO$^2(<, Succ)$ it is sufficient to compute the syntactic semigroup and verify Equation (2.3). Therefore we get the following corollary:

Corollary 3.10. *It is decidable whether a regular word language is definable in* FO$^2(<, Succ)$.

The rest of this section is devoted to the proof of Theorem 3.9. Both direction are proved using Ehrenfeucht-Fraïssé game techniques. Therefore, we use Theorem 2.4 in order to adopt an $F + F^{-1} + X + X^{-1}$ point of view. There are two directions in Theorem 3.9. We begin with the easier 'only if' direction.

Proposition 3.11. *If a language L is definable in* FO$^2(<, Succ)$*, its syntactic semigroup verifies* (2.3)*.*

Proof. This is done using Ehrenfeucht-Fraïssé techniques using a strategy for Duplicator similar to the one used for the proof of Proposition 3.4. □

The other direction is proved using the following proposition. Recall that given two words w, w' we write:

- $w \simeq_k w'$ iff Duplicator has a winning strategy in the k rounds $F + F^{-1}$ game on w and w'.
- $w \simeq_k^+ w'$ iff Duplicator has a winning strategy in the k rounds $F + F^{-1} + X + X^{-1}$ game on w and w'.

Proposition 3.12. *Let S be a semigroup verifying Equation* (2.3) *and $\alpha : A^+ \to S$ some morphism. Then the following property holds:*

$$\exists k \; w \simeq_k^+ w' \Longrightarrow \alpha(w) = \alpha(w')$$

It follows from Proposition 3.12 that any language recognized by a semigroup verifying Equation (2.3) is a union of classes of $\simeq_k^+$ for some k. It is also a consequence from Lemma 3.1 that any union of classes of $\simeq_k^+$ for some k is definable in $F + F^{-1} + X + X^{-1}$. It follows that any language recognized by a semigroup verifying Equation (2.3) is definable in $F + F^{-1} + X + X^{-1}$. This finishes the proof of Theorem 3.9. We turn to the proof of Proposition 3.12:

Proof. We begin with the definition of a construction on words that we use throughout the proof. We define a new alphabet B and to each word over A we associate a word over B in a canonical manner. Note that this construction was first presented in [Str85].

We fix an arbitrary order on $E(S)$ the set of idempotents of S. We define a new alphabet $B = \{(e, w, f), (e, w), (w, f) \mid e, f \in E(S), w \in A^* \; |w| \leq |S|\}$, and to each word $w = a_0 a_1 \cdots a_n$ of length greater than $|S|$ of A^+ we associate a word $\lfloor w \rfloor = b_0 \cdots b_m$ of B^+ such that:

- $b_0 = (w_0, f_0)$, $b_m = (e_m, w_m)$, and $b_i = (e_i, w_i, f_i)$ for $1 \leq i < m$.
- $e_{i+1} = f_i$ for $0 \leq i < m$.
- $w_0 \cdots w_m = w$.
- We have $\alpha(w_i) f_i = \alpha(w_i)$.

Because of the third item, to each position x in w corresponds a position $\hat{x}$ which is the position i such that x is in the factor w_i. We also want this construction to be *locally canonical*. By this we mean that for each position x of w it is possible to infer the label $b \in B$ of $\hat{x}$ in $\lfloor w \rfloor$ only by inspecting the neighborhood of x of a bounded size.

The construction of $\lfloor w \rfloor$ is based on the following observation:
If $w = a_0 \cdots a_n$ is such that $n > |S|$, we can find i, j, $i < j$ such that: $\alpha(a_0 \cdots a_i) = \alpha(a_0 \cdots a_j)$. We then have $\alpha(a_0 \cdots a_i) = \alpha(a_0 a_1 \cdots a_i)(\alpha(a_{i+1} \cdots a_j))^\omega$. This implies that there is a idempotent e such that $\alpha(a_0 \cdots a_i) = \alpha(a_0 a_1 \cdots a_i)e$.

In order to make this canonical we always consider the smallest such i and, once i is determined, the smallest e in the order chosen for $E(s)$, such that the property above holds.

The construction of $\lfloor w \rfloor$ is now obtained from w by considering all sequences of $|S| + 1$ letters of w successively from left to right and inserting the idempotent determined as above at the right place.

It is easy to verify that all desired properties are verified. In particular the construction is locally canonical for a neighborhood of size $2|S| + 1$.

We now show:

Lemma 3.13. $\forall m\ \exists n \qquad w \simeq_n^+ w' \Longrightarrow \lfloor w \rfloor \simeq_m \lfloor w' \rfloor$

Lemma 3.14. $\exists m\ \lfloor w \rfloor \simeq_m \lfloor w' \rfloor \Longrightarrow \alpha(w) = \alpha(w')$

Using the Lemmas we have for all w, w' such that $|w| > |S|$, $|w'| > |S|$:

$$\exists k\ w \simeq_k^+ w' \Longrightarrow \alpha(w) = \alpha(w')$$

Then for all w, w':

$$w \simeq_{k+|S|+1}^+ w' \Longrightarrow \alpha(w) = \alpha(w')$$

Which ends the proof of Proposition 3.12. We prove Lemmas 3.13 and 3.14:

Proof. **(of Lemma 3.13)** This proof relies on the local canonicity of the construction. Let $k = |S|$ and take $n = m + k$. Assume that $w \simeq_n^+ w'$. We describe a winning strategy for Duplicator on the m rounds $F + F^{-1}$ Ehrenfeucht-Fraïssé-game on $\lfloor w \rfloor$ and $\lfloor w' \rfloor$, that we call $\mathcal{G}$ in our proof. The strategy uses Duplicator's strategy on the n round $F + F^{-1} + X + X^{-1}$ Ehrenfeucht-Fraïssé-game on w and w'. During the play of the game $\mathcal{G}$, Duplicator will play alone a shadow $F + F^{-1} + X + X^{-1}$ Ehrenfeucht-Fraïssé-game on w and w' that we denote by $\mathcal{G}_+$.

- Spoiler plays on a position y' labeled with a letter $b = (e, p, f)$ of $\lfloor w' \rfloor$ (the case in which he plays on $\lfloor w \rfloor$ is symmetric).

- Duplicator simulate a play of Spoiler in $\mathcal{G}_+$ by putting the pebble on the position x' in w' corresponding to the first letter of w_i. (notice that $\hat{x'} = y'$)
- Her strategy on $\mathcal{G}_+$ gives her an answer on a position x of w. Since there are at least k more moves in $\mathcal{G}_+$, x and x' agree on their label and on the label of their k predecessors and k successors. It follows that x and x' have the same neighborhood of size $2k+1$, by local canonicity $\hat{x}$ and $\hat{x'}$ have the same label b. Therefore, the position $\hat{x}$ in $\lfloor w \rfloor$ is Duplicator's answer to Spoiler's move.

As Duplicator can survive n rounds, this concludes the proof. □

Proof. **(of Lemma 3.14)** Given $b = (e, p, f) \in B$, we write $\alpha(b)$ for $e\alpha(p)f$. We say that a word w of B^+ is well formed iff $w = b_1 \dots b_m$ with $b_1 = (w_1, f_1)$, $b_n = (e_n, w_n)$ and $b_i = (e_i, w_i, f_i)$ and for all i, $f_i = e_{i+1}$. In particular, by definition, if w is a word of A^+, $\lfloor w \rfloor$ is well-formed. We show that given two well formed words w and w' of B^+, we have:

$$w \simeq_k w' \Longrightarrow \alpha(w) = \alpha(w')$$

The proof is done by induction on $|B|$ and we show that the property holds for $k > |S^1||B|$:

- If $|B| = 0$ then the result is just $\alpha(\epsilon) = \alpha(\epsilon)$.
- Otherwise we prove two properties. For $u \in S^1$, we write $dp_r(u)$ (resp. $dp_l(u)$) the number of elements v of S^1 such that u is r-reachable (resp. l-reachable) from v.

$$\begin{array}{ll} \forall u_l \in S^1 & w \simeq_{k-dp_r(u_l)} w' \Rightarrow u_l\alpha(w) \text{ and } u_l\alpha(w') \text{ are mutually } r\text{-reachable} \\ \forall u_r \in S^1 & w \simeq_{k-dp_l(u_r)} w' \Rightarrow \alpha(w)u_r \text{ and } \alpha(w')u_r \text{ are mutually } l\text{-reachable} \end{array}$$

Before proving the properties we show how to conclude using them. Consider the properties in the case where $u_l = u_r = \epsilon$. We then have $w \simeq_k w' \Rightarrow \alpha(w)$ and $\alpha(w')$ are mutually r-reachable and l-reachable. Therefore we have $\alpha(w) = u\alpha(w')$ and $\alpha(w') = \alpha(w)v$. A little algebra then yields:

$$\begin{array}{rcl} \alpha(w) & = & u\alpha(w)v \\ \alpha(w) & = & u^\omega\alpha(w)v^\omega \\ \alpha(w)v & = & u^\omega\alpha(w)v^{\omega+1} \\ \alpha(w') & = & u^\omega\alpha(w)v^\omega \text{ using (2.3)} \\ \alpha(w') & = & \alpha(w) \end{array}$$

We now prove the first property, the second one being obtained by symmetry. We proceed by induction on the number of elements that r-reachable from u_l. We consider three cases:

1. u_l is r-reachable from both $u_l\alpha(w)$ and $u_l\alpha(w')$.
2. u_l is not r-reachable from $u_l\alpha(w)$.
3. u_l is not r-reachable from $u_l\alpha(w')$.

In the first case there is nothing to be done, since the mutual r-reachability between $u_l\alpha(w)$ and $u_l\alpha(w')$ is an immediate consequence. We do the proof for the second case, the third case is handled by symmetry.

Suppose u_l is not r-reachable from $u_l\alpha(w)$. Then there exists $b \in B$ such that $w = w_1 b w_2$ and u_l is r-reachable from $u_l\alpha(w_1)$ while u_l is not r-reachable from $u_l\alpha(w_1 b)$

Claim 3.15. *There is no position in w_1 with label b.*

Proof. Suppose that $b \in w_1$, and that $b = (e, p, f)$ and $v = \alpha(p)$. Because w is well formed, b is compatible with w_1, therefore, the last letter of w_1 ends with the idempotent e. Therefore, we can decompose $u_l\alpha(w_1)$ as:

$$u_l\alpha(w_1) = v_1 evf v_2 e$$

Because u_l is r-reachable from $u_l\alpha(w_1)$, v_1 is r-reachable from $v_1 evf v_1 e$ and there exists h such that:

$$v_1 \quad = \quad v_1 evf v_2 eh$$

It follows that:

$$\begin{aligned}
v_1 evf v_2 e &= v_1 evf v_2 eh\ evf v_2 e \\
v_1 evf v_2 e &= v_1 evf v_2 e\ eh\ evf v_2 e \\
v_1 evf v_2 e &= v_1 evf v_2 e\ (eh\ evf v_2 e)^\omega \\
v_1 evf v_2 e &= v_1 evf v_2 e\ (eh\ evf v_2 e)^\omega\ evf v_2 e\ (eh\ evf v_2 e)^\omega && \text{Using (2.3)} \\
v_1 evf v_2 e &= v_1 evf v_2 e\ evf\ v_2 e\ (eh\ evf v_2 e)^\omega
\end{aligned}$$

Therefore $v_1 evf v_2 e$ is r-reachable from $v_1 evf v_2 e\ evf$. This is exactly says that $u_l\alpha(w_1)$ is r-reachable from $u_l\alpha(w_1 b)$, which is a contradiction by hypothesis. □

We decompose w' as $w' = w'_1 b w'_2$ such that $b \notin w'_1$. Note that such a decomposition exists because $w \cong_{k-dp_r(x)} w'$ and $k - dp_r(x) \geq 1$ and therefore b is a letter of w' (if Spoiler plays on the b in w, there is a b in w' which Duplicator can respond on).

Claim 3.16. *$w_1 \cong_{k-dp_r(x)-1} w'_1$ and $w_2 \cong_{k-dp_r(x)-1} w'_2$*

Proof. In both cases this is a simple consequence of Claim 3.15. In the game on w, w', if Spoiler place a pebble on w_1 then Duplicator's response must be in w'_1. Otherwise Spoiler would play in w' his next move on the b separating w'_1 from w'_2 and by Claim 3.15 Duplicator would not be able to find such a b to the left of the current pebble which is in w_1. Similarly if Spoiler plays in w_2 Duplicator must answer in w'_2, as soon as one extra move is available. □

We conclude the proof with this last claim. First considering w_1 and w'_1 we have:

- $b \notin w_1, w'_1$.
- Since, u_l is not reachable from $u_l\alpha(w_1 b)$, $dp_r(u_l) \geq |S^1|$, therefore: $k - dp_r(u_l) - 1 > |S^1||B| - (|S^1| - 1) - 1 > |S^1|(|B - 1|)$.

Therefore, using the outer induction hypothesis on the size of $|B|$, $u_l\alpha(w_1) = u_l\alpha(w'_1)$. Now considering w_2 and w'_2 we have:

- $k - dp_r(u_l) - 1 > |S^1||B| - dp_r(u_l\alpha(w_1 b))$.
- Thanks to the inner induction hypothesis, with $v = u_l\alpha(w_1 b)$ we have $v\alpha(w_2)$ and $v\alpha(w'_2)$ mutually r-reachable.

But then $u_l\alpha(w') = u_l\alpha(w'_1)\alpha(bw'_2) = v\alpha(w'_2)$ and $u_l\alpha(w) = v\alpha(w'_2)$ are mutually r-reachable which ends the proof. □

□

Chapter 4

Locally Testable Languages over Words

In this Chapter we provide and prove a decidable characterization for LT. Decidable characterizations for LT were obtained independently by [BS73] and [McN74]. These characterizations are based on the syntactic semigroup of the language. We present a new proof for this characterization. It is based on a non algebraic characterization of LT that we call tame languages. This proof also implies the algebraic characterization of [BS73] and [McN74].

Note that there exists another method to decide membership in LT for a regular word language. This algorithm, presented in [Boj07a], relies on the delay theorem. In the special case of LT, the delay theorem says that if a finite state automaton A recognizes a language in LT then this language must be κ-locally testable for a κ computable from A. This theorem was proved over words in [Str85] and can be used in order to decide whether a regular language is in LT as explained in [Boj07a].

In Section 4.1, we define the notion of a tame language. Then in Section 4.2, we state and prove the characterization.

4.1 Tame Languages

Guarded Operations Fix some integer k, we define two closure properties on languages that we call *k-guarded swap* and *k-guarded stutter*.

Let w be a word and x, y, z be three positions in w such that $x < y < z$. The *swap* of w at x, y, z is the word w' constructed from w by switching $w[x, y]$ with $w[y, z]$, see Figure 4.1. A swap is k-guarded if x, y, z have the same k-type.

Let w be a word and x, y, z be three positions in w such that $x < y < z$ and such that $w[x, y] = w[y, z]$. The *stutter* of w at x, y, z is the word w' constructed

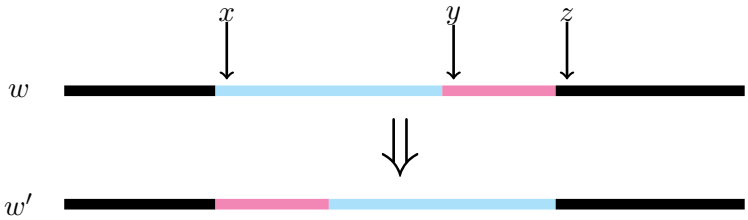


Figure 4.1: Swap

from w by deleting $w[y, z]$, see Figure 4.2. A stutter is k-guarded if x, y, z have the same k-type.

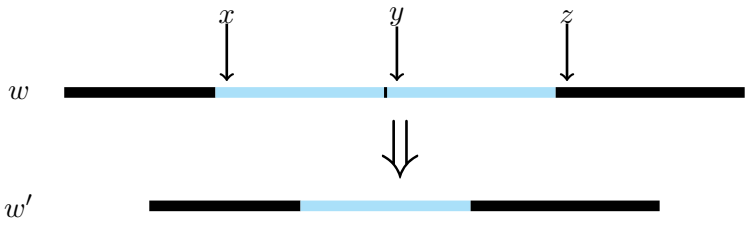


Figure 4.2: Stutter

Let L be a word language and k be a number. We say that L is *closed under k-guarded swap* (resp. k-guarded stutter) if for every word w and every word w' constructed from w using k-guarded swap (resp. k-guarded stutter) then $w \in L$ iff $w' \in L$. An important observation is that k-guarded operations do not affect the set of $(k+1)$-types occurring in a word.

If L is closed under the two k-guarded operations described above, we say that L is *k-tame.* A language is said to be *tame* if it is k-tame for some k.

4.2 Decidable Characterization

Theorem 4.1. *Consider L, a regular language over some alphabet A, the following properties are equivalent:*

1. *L is in* LT.
2. *L is tame.*
3. *The syntactic semigroup S of L verifies, for all $u, v, e \in S$ such that e is an idempotent:*

$$eueue = eue \tag{4.1}$$
$$eueve = eveue \tag{4.2}$$

The algebraic part of Theorem 4.1 is exactly the algebraic characterization of [BS73] and [McN74]. It follows from Theorem 4.1 that membership in LT is a decidable property of regular languages.

Corollary 4.2. *It is decidable whether a regular language is in* LT.

We devote this section to the proof of Theorem 4.1. We proceed in two steps. First we prove that being in LT is equivalent to being tame. In a second step we prove that tameness is equivalent to the algebraic characterization.

4.2.1 LT $\Leftrightarrow$ *tame*

We prove that tameness is equivalent to being in LT. There are two directions that we separate. We begin with the easier "only if" direction and show that any LT language is tame.

Proposition 4.3. *If L is* LT *then L is tame.*

Proof. By definition L is k-locally testable for some integer k. Since the k-guarded operations do not affect the k-types in the words, it follows that L is k-tame. Therefore L is tame which ends the proof. □

We turn to the harder "if" direction, if a language is tame then it is in LT. It follows from the following proposition:

Proposition 4.4. *Let L be a k-tame language. Consider w_1, w_2 two words:*

$$w_1 \simeq_{k+1} w_2 \Rightarrow w_1 \in L \text{ iff } w_2 \in L$$

Indeed a simple consequence of Proposition 4.4 is that if L is k-tame, then it is a union of classes of $\simeq_{k+1}$ which is exactly the definition of being $(k+1)$-locally testable. Therefore if L is tame, it is in LT, and this finishes this direction of the proof. We now prove Proposition 4.4.

Proof. Before proving Proposition 4.4 we need some extra terminology. Let w be some word and $x < y$ be two positions in w, we say the factor $w[x, y]$ is a *loop of k-type τ* if x and y have the same k-type and that this k-type is τ. A factor w' of some word w is a k-loop if there is some k-type τ such that w is a loop of k-type τ. Notice that the notion of k-loop depends on the surrounding word w.

Consider L that is k-tame and fix w_1, w_2 two words such that $w_1 \simeq_{k+1} w_2$. We show that $w_1 \in L$ iff $w_2 \in L$. We proceed in two steps. First we show that using a sequence of k-guarded stutters, we can transform w_1 into a new word w_3. Intuitively w_3 corresponds to the word w_2 plus some added factors that are k-loops. By hypothesis of tameness, since w_3 is built using k-guarded stutters, we get that $w_1 \in L$ iff $w_3 \in L$. Then in a second step we show that we can add these k-loops within w_2 using k-guarded stutters and swaps. Again by hypothesis of tameness, this does not affect membership in L. It follows that $w_2 \in L$ iff $w_3 \in L$ and subsequently that $w_1 \in L$ iff $w_2 \in L$, which ends the proof. This proof structure is depicted in Figure 4.3.

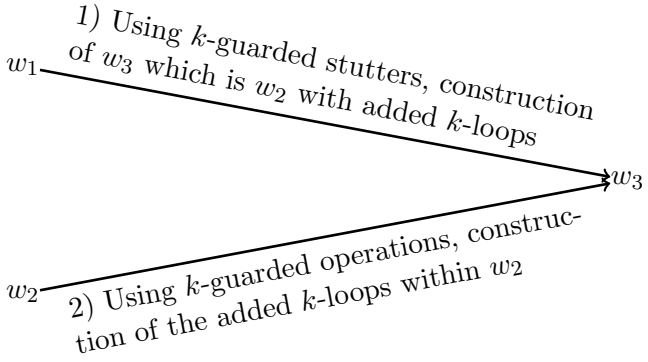


Figure 4.3: Proof structure in Proposition 4.4

Construction of w_3 from w_1. Before starting the proof we give a formal definition for w_3. Fix two words w, w' and $x_1, \ldots, x_n$ the sequence of positions of w and $x'_1, \ldots, x'_m$ the sequence of positions of w'. We say that w' *contains* w iff there exists a injective map β from the positions of w to the positions of w' that verifies the following conditions:

- For $x_i < x_j$, $\beta(x_i) < \beta(x_j)$.
- For $y_i < y_j$ such that for all l that verifies $i \leq l < j$, $y_l \notin \beta(w)$ and $y_{i-1}, y_j \in \beta(w)$, $w'[y_{i-1}, y_j]$ is a k-loop.

We illustrate this notion in Figure 4.4. We want to construct w_3 from w_1 such that it contains w_2. We prove a slightly more general lemma by induction:

Lemma 4.5. *For every position x_2 in w_2, there exists a word w_3 and a position x_3 in w_3 such that:*

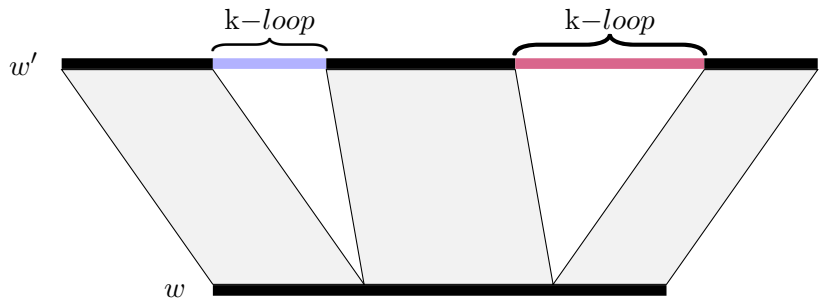


Figure 4.4: Illustration of w' that contains w

- *w_3 can be constructed from w_1 using k-guarded stutters.*
- *x_2 and x_3 have the same $(k+1)$-type.*
- *The prefix at x_2 of w_2 is contained into the prefix at x_3 of w_3.*

Proof. This is by induction on the length of the prefix at x_2, if x_2 is the leftmost position of w_2 then we just choose w_1 for w_3 and x_3 for the leftmost position in w_3. Indeed the empty word is contained in itself and by definition, since $w_1 \simeq_{k+1} w_2$, x_2 and x_3 have the same $(k+1)$-type.

Now suppose that the property is verified for a prefix of length n and fix x_2 such that the prefix at x_2 is of length $n+1$. Let $y_2 = x_2 - 1$ by induction hypothesis there exists a word w_4 and a position y_4 in w_4 such that:

- w_4 can be constructed from w_1 using k-guarded stutters.
- y_2 and y_4 have the same $(k+1)$-type.
- The prefix at y_2 of w_2, is contained into the prefix at y_4 of w_4.

We show that we can construct the desired word w_3 from w_4 using at most one stutter. Let τ be the $(k+1)$-type of x_2. Since $w_1 \simeq_{k+1} w_2$ and because k-guarded stutter does not affect $(k+1)$-types, by construction we have $w_4 \simeq_{k+1} w_2$. Therefore τ occur in w_4 at some position z_4. We distinguish two cases depending on the relation between z_4 and y_4:

We have $y_4 < z_4$ In this case there is no need to use stutter and we just choose $w_3 = w_4$ and $x_3 = z_4$. By choice of x_3 it has the same $(k+1)$-type as x_2. We just need to show that the prefix at x_2 in w_2 is contained into the prefix at x_3 in w_3. By choice of w_3 we already have a mapping from the prefix at y_4 into the prefix at y_2. We complete it by mapping x_2 to x_3. To finish the proof we just

need to show that if the factor $w_3[y_4+1, x_3]$ is not empty then it is a k-loop. This is true because y_2 and y_4 have the same $(k+1)$-type therefore their successors x_2 and y_4+1 have the same k-type. Since x_3 has been chosen to have the same $(k+1)$-type as x_2, it follows that x_3 and y_4+1 have the same k-type. We illustrate the construction in Figure 4.5.

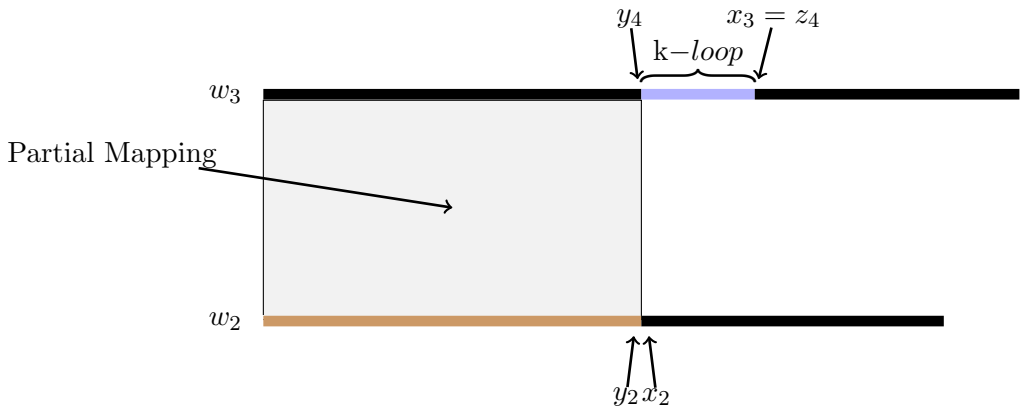


Figure 4.5: Case $y_4 < z_4$ in Lemma 4.5

We have $y_4 \geq z_4$ We show that we can reduce this case to the previous one using one k-guarded stutter on w_4. By hypothesis z_4 and y_4+1 have the same k-type. Therefore we apply k-guarded stutter between z_4 and y_4+1 in w_4. This yields a word w_3 with a position z_4' with $(k+1)$-type τ to the right of y_4 and we are in the previous case (see Figure 4.6).

□

Now that we have Lemma 4.5, to get w_3 from w_1 such that it contains w_2 we fix x_2 as the rightmost position of w_2. Since x_2 and x_3 have the same $(k+1)$-type, it follows that x_3 is also the rightmost position of w_3. Therefore w_2 is contained in w_3.

Construction of w_3 from w_2. We want to insert the added loops of w_3 in w_2 using k-guarded operations. We prove the following lemma:

Lemma 4.6. *Assume L is k-tame. Let w be a tree and x a node of w of k-type τ. Let w' be another word such that $w \simeq_{k+1} w'$ and w'' be a k-loop of type τ in w'. Consider the word W constructed from w by inserting a copy of w'' at x. Then $w \in L$ iff $W \in L$.*

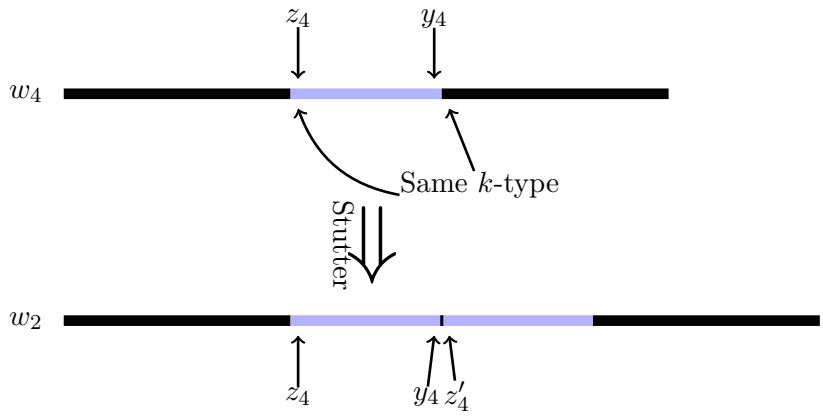


Figure 4.6: Case $y_4 \geq z_4$ in Lemma 4.5

Proof. Consider the factor w'' occurring in w'. Let $y_0, \cdots, y_n$ be the positions of w'' in w' ($w'' = w'[x_0, x_n]$) and $\tau_0, \cdots, \tau_n$ be their respective $(k+1)$-types.

From w we construct using k-guarded swaps and k-guarded stutters a word w_1 such that there is a sequence of nodes $x_0, \cdots, x_n$ in w_1 with for all $0 \leq i < n$, x_i is of type τ_i and $x_i < x_{i+1}$. The word w_1 is constructed by induction on n. If $n = 0$ then this is a consequence of $w \simeq_{k+1} w'$ that one can find in w a position of type τ_0. Consider now the case $n > 0$. By induction we have constructed from w a word w'_1 such that $x_0, \cdots, x_{n-1}$ is an appropriate sequence in w'_1. Because the two k-guarded operations preserve $(k+1)$-types, we have $w \simeq_{k+1} w'_1$ and hence there is a position x of w'_1 of type τ_n. If $x_{n-1} < x$ then we are done. Otherwise consider $x' = Succ(x_{n-1})$ and notice that because $y_n = Succ(y_{n-1})$ and x_{n-1} has the same $(k+1)$-type than y_{n-1} then x', y_n and x have the same k-type.

By hypothesis, $x < x'$ and the factor $w'_1[x, x']$ is a k-loop, therefore we can use k-guarded stutter to duplicate it. This places a node having the same $(k+1)$-type as x as the successor of x_{n-1} and we are done.

This concludes the construction of w_1. From w_1 we construct using k-guarded swaps and k-guarded stutters a word w_2 such that there is a sequence $x_0, \cdots, x_n$ in w_2 with for all $0 \leq i < n$, x_i is of type τ_i and $x_i = Succ(x_{i-1})$.

Consider the sequence $x_0, \cdots, x_n$ obtained in w_1 from the previous step. Recall that the k-type of x_0 is the same as the k-type of x_n. Hence using k-guarded stutter we can duplicate in w_1 the factor $w_1[x_0, x_n]$. Let w'_1 the resulting word. We thus have two copies of the sequence $x_0, \cdots, x_n$ that we denote by the *right copy* and the *left copy*. Assume $x_i \neq Succ(x_{i+1})$. Notice then that $w'_1[x_{i-1}, x_i]$ is a k-loop. Using k-guarded swap we can move the left copy of this context next to its right copy. Using k-guarded stutter this extra copy can be removed. We are

left with an instance of the initial sequence in the right copy, while in the left one $x_i = Succ(x_{i-1})$.

Repeating this argument yields the desired word w_2. Since w_2 was constructed from w using a sequence of k-guarded swaps and k-guarded stutter and since L is k-tame $w \in L$ iff $w_2 \in L$.

We end the proof by showing that we can construct w_2 from w using the same sequence of k-guarded operations. Indeed W is just w with an added k-loop, w'' therefore, we can easily identify the nodes of w with the nodes of W outside of this loop. Consider the same sequence of k-guarded operations applied to W. Observe that this yields a tree W_2 corresponding to W_2 with possibly several extra copies of w''. With appropriates k-guarded swaps, all the extra copies can be brought together and using k-guarded stutter only one copy remains resulting in w_2. Hence $W \in L$ iff $w_2 \in L$ and finally $w \in L$ iff $W \in L$ and this finishes the proof. See figure 4.7.

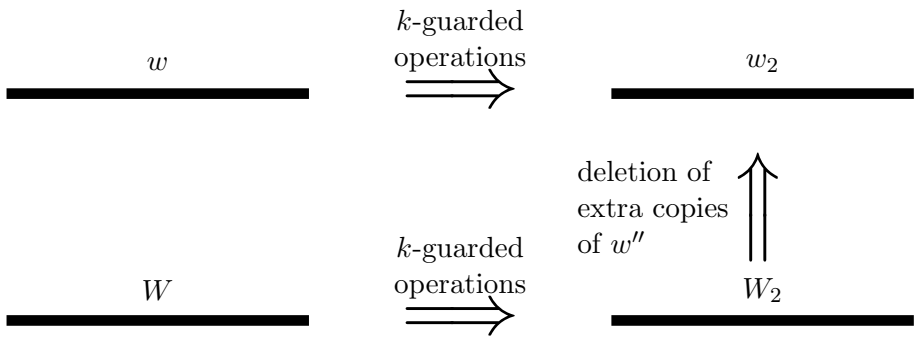


Figure 4.7: Relation with w_2

□

By construction of w_3 we have $k+1 \simeq_{w_3} w_1$ and therefore $k+1 \simeq_{w_3} w_2$. Hence we can construct w_3 from w_2 via a repeated use of Lemma 4.6. It follows that $w_3 \in L$ iff $w_2 \in L$, since we already had that $w_3 \in L$ iff $w_1 \in L$, we conclude that $w_1 \in L$ iff $w_2 \in L$ which concludes the proof.

□

4.2.2 Algebraic characterization

We now prove the equivalence between the algebraic characterization given by Equations (4.1) and (4.2) and tameness. We prove the following Proposition:

Proposition 4.7. *Let L be a regular language. L is tame iff the syntactic semigroup of L, S, verifies for all $u, v, e \in S$ such that e is idempotent:*

$$eueue = eue \tag{4.1}$$

$$eueve = eveue \tag{4.2}$$

Proof. There are two directions. First suppose that L is k-tame, we show that S verifies (4.1) (the proof is similar for (4.2)). Let w, w' be two words such that $\alpha(w) = u, \alpha(w') = e$. Now consider the words $(w')^k w (w')^k w (w')^k$ and $(w')^k w (w')^k$. By closure under k-guarded stutter we get that for all words w_1, w_2, $w_1 (w')^k w (w')^k w (w')^k w_2 \in L$ iff $w_1 (w')^k w (w')^k w_2 \in L$. By definition of the syntactic semigroup it follows that $eueue = eue$. Therefore S verifies Equation (4.1).

We turn to the other direction and suppose that S verifies Equations (4.1) and (4.2). We begin to show that we can extract an idempotent of any word of length greater than $|S|$. Indeed if $w = a_0 \cdots a_k$ is such that $k > |S|$, we can find i, j, $i < j$ such that: $\alpha(a_0 \cdots a_i) = \alpha(a_0 \cdots a_j)$. We then have $\alpha(a_0 \cdots a_i) = \alpha(a_0 a_1 \cdots a_i)(\alpha(a_{i+1} \cdots a_j))^\omega$. This implies that there is a idempotent e such that $\alpha(a_0 \cdots a_i) = \alpha(a_0 a_1 \cdots a_i)e$.

Fix $k = |S|$, we show that L is k-tame. We show that L is closed under k-guarded stutter (the closure under k-guarded swap is proved similarly using Equation (4.2)).

Let w be a word and let x, y, z be three positions as in the definition of k-guarded stutter and with the same k-type. The word w can be written $w = w_1 w' w_2 w' w_2 w' w_3$ with w' of length greater than k and with x, y, z being the leftmost positions of respectively the three copies of w' from left to right. By choice of k we have $\alpha(w') = uev$ with e idempotent. Therefore:

$$\begin{array}{lcl} \alpha(w_1 w' w_2 w' w_2 w' w_3) & = & \alpha(w_1) uev \alpha(w_2) uev \alpha(w_2) uev \alpha(w_3) \\ \alpha(w_1 w' w_2 w' w_2 w' w_3) & = & \alpha(w_1) uev \alpha(w_2) uev \alpha(w_3) \text{ using (4.1)} \\ \alpha(w_1 w' w_2 w' w_2 w' w_3) & = & \alpha(w_1 w' w_2 w' w_3) \end{array}$$

It follows that $w \in L$ iff the k-guarded stutter of w at x, y, z is in L. Therefore L is closed under k-guarded stutter which finishes the proof.. □

4.3 Discussion

An interesting corollary of the characterization is that we can bound the expected k such that a language is k-LT. It is linear in the size of the syntactic semigroup of L.

Corollary 4.8. *Let L be some language and S be the syntactic semigroup. Then L is in* LT *iff L is $(|S|+1)$-locally testable iff L is $|S|$-tame.*

Proof. L is LT iff its syntactic semigroup $|S|$ verifies Equations (4.1) and (4.2). By the proof of Proposition 4.7 we get that $L \in \text{LT} \Rightarrow L$ $|S|$-tame. Finally by Proposition 4.4 we get that L $|S|$-tame $\Rightarrow L$ $(|S|+1)$-locally testable. □

Notice that this provides an alternate decision procedure for deciding membership in LT. Indeed for any fixed k there exists a finite amount of k-locally testable languages. Therefore, to decide if L is LT, one only need to compare it to all $(|S|+1)$-LT languages for S the syntactic semigroup of L. This decision procedure will be particularly useful for the tree setting in the second part.

Part II

Trees

Chapter 5

Notations and Algebra for Trees

In this chapter we give definitions and fix notations for trees that we use in this second part. In particular, we define the notions of unranked trees, forests and trees of bounded rank. We give actually give these definitions by regrouping them into only two categories. In Section 5.1 we regroup the definitions of forests and unranked trees in Section 5.1. In Section 5.2 we give the definitions for trees of bounded rank. Note that the trees of bounded rank we investigate are actually terms. This means that the number of children of a node is determined by its label. Sections 5.3 and 5.4 are devoted to regular tree languages. In Section 5.3, we define regular languages of trees and forests and in Section 5.4, we define regular languages of trees of bounded rank. In Section 5.5 we give the definition of forest algebras for forests. Note that all the definitions and results of this section are taken from [BW07]. Finally, in Section 5.6 we define k-algebras for trees of rank k.

5.1 Unranked Trees and Forests

Trees and Forests. We consider *finite unranked ordered trees and forests* whose nodes are labeled over an alphabet. Recall that an alphabet A is a finite set. Formally if A is an alphabet then our unranked trees and forests over A are generated by the following rules: For all $a \in A$, a is a tree, if $k \geq 1$ and $t_1, \cdots, t_k$ are trees then $t_1 + \cdots + t_k$ is a forest, furthermore if $a \in A$ and s is a forest then $a(s)$ is a tree. We also consider an empty forest that we write $\Box$. See Figure 5.1. Note that in Chapter 11 we will also consider *unordered unranked trees*. Intuitively this means that there is no order between siblings. Formally, this means that for any permutation σ, the forests $t_1 + \cdots + t_k$ and $t_{\sigma(1)} + \cdots + t_{\sigma(k)}$ are viewed as

the same forest. We will come back to this notion in Chapter 11.

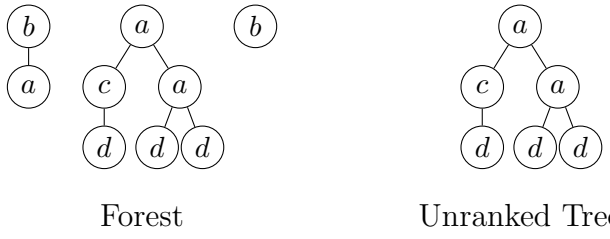
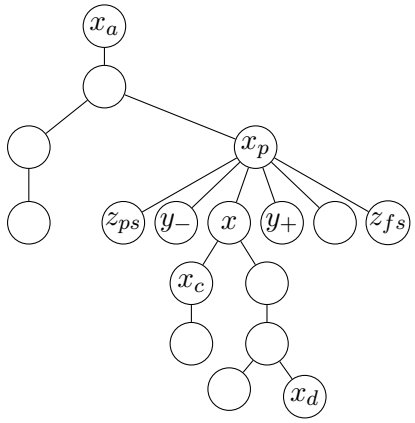


Figure 5.1: Illustration of the Notions of Unranked Trees and Forests

Relations. We use standard terminology for forests and unranked trees in order to define the notion of *node* or *position* in a tree or a forest. We also use standard definitions for the notions of *ancestor*, *parent*, *descendant*, *child*, *following sibling*, *next sibling*, *preceding sibling* and *previous sibling* of a node. We also call *root* a node that has no parent and *leaf* a node that has no children. Notice that a tree has only one root while a forest may have several. We depict these relations in Figure 5.2.



x_a is an ancestor of x
x_p is the parent of x
x_d is a descendant of x
x_c is a child of x
z_{fs} is a following sibling of x
y_+ is the next sibling of x
z_{ps} is a preceding sibling of x
y_- is the previous sibling of x

Figure 5.2: Illustration of the Relations in a Forest

Contexts and Strict Contexts. We define the notion of *context* and the notion of *strict context* which is a refinement of the notion of context. Contexts are defined in the usual way. A context is a special kind of forest (or unranked tree) over the alphabet $A \cup \{\Box\}$ with only one node with label $\Box$ that must be a leaf.

We call this particular node the *the port* of the context. Notice that the port may be a root. We have an empty context, $\Box$. A strict context is a context whose port has no siblings (see Figure 5.3). Moreover, a context such that the port is a root is not strict. The notion of strict context will be crucial in Chapter 10.

Given contexts c and c', their concatenation $c \cdot c'$ is the context formed by identifying the root of c' with the port of c. A forest $c \cdot t$ can be obtained similarly by combining a context c and a forest t.

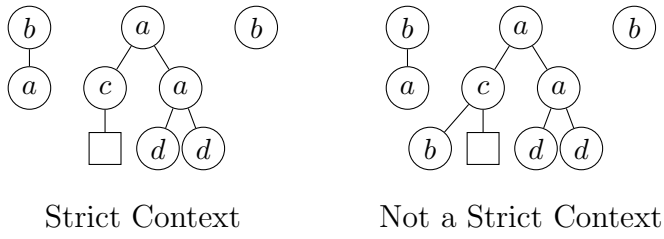


Figure 5.3: Illustration of the Notion of Strict Context

If x is a node of a forest then the *subtree at* x is the tree rooted at x. The *subforest of* x is the forest consisting of all the subtrees that are rooted at siblings of x (including x), see Figure 5.4. Notice that by definition of strict contexts and subforests we have that s is a subforest of a forest t iff there exists a strict context c such that $t = c \cdot s$. Therefore if we consider the forest $a \cdot (a + b + d)$, $a + b + d$ is a subforest but not $b + d$.

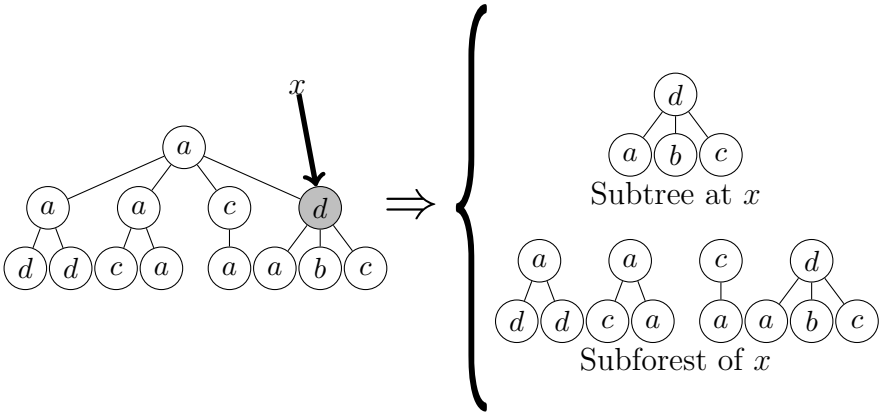


Figure 5.4: Illustration of the Notion of Subtrees and Subforests

5.2 Trees of Bounded Rank

Trees. An alphabet of rank k for some integer k is a tuple $A = (A_0, A_1, \ldots, A_k)$, with A_i is a finite set of symbols of rank i. Intuitively this means that any node labeled with a label of a_i will have i children. Given an alphabet of rank k, $A = (A_0, A_1, \ldots, A_k)$, the trees of rank k over A are generated by the following rules: for all $a \in A_0$, a is a tree, for $1 \leq i \leq k$, $a \in A_i$ and i trees $t_1, ..., t_i$, $a(t_1, ..., t_i)$ is a tree. Notice that there is no empty tree in this setting.

Relations. Notice that a tree of rank k is in particular an unranked tree. Therefore the notions of *nodes*, *ancestor*, *parent*, *descendant*, *child*, *following sibling*, *next sibling*, *preceding sibling* and *previous sibling* are defined as in Section 5.1 for unranked trees. (see Figure 5.2)

Contexts and i-contexts. The notion of context also remains identical, a context is a tree with a designated leaf that has a special label and which is called *the port* of the context. We do not consider a notion of strict context for trees of rank k since we do not need it. However we consider a new notion that we call i-contexts. For any integer $1 \leq i \leq k$ an i-context is a tree that has exactly i ports that are all siblings (see Figure 5.5). In particular a context is a 1-context. A context c can be composed with another context c' or with a tree t in the obvious way. An i-context can be attached below a context to get a new i-context and we can attach a tree below a i-context to get a $(i-1)$-context

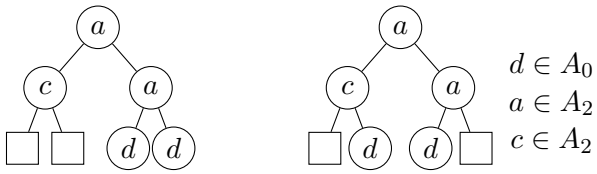


Figure 5.5: Illustration of the Notion of Strict Context

Given a tree t and a node x of t the *subtree of t rooted at x*, consisting of all the nodes of t that are descendant of x, is denoted by $t|_x$. Given a tree t and two nodes x, y of t such that y is a descendant (not necessarily strict) of x, *the context of t between x and y*, denoted by $t[x, y]$, is defined by keeping all the nodes of t that are descendant of x but not descendant of y and by placing the port at y. We say that a context C *occur* in t if C is the context of t between x and y for some nodes x and y of t.

5.3 Regular Languages of Unranked Trees and Forests

Languages of trees and languages of forests. Fix some alphabet A, an unranked tree language (resp. a forest language) over A is a set of unranked trees (resp. of forests) over A. We write A^{Δ} the pair (H, V) where H is the set of all forests over A (including the empty forest) and V the set of all contexts over A. We write $A^{\bar{\Delta}}$ the pair (H, V) where H is the set of all forests over A (excluding the empty forest) and V the set of all strict contexts over A.

Regular Languages. All the classes we investigate are fragments of the class of regular unranked trees languages (resp. regular forest languages). Regular languages are the languages that are definable using a finite automaton. Note that regular languages can also be equivalently defined using algebraic and logical notions. We provide the algebraic definition in the Section 5.5 and the logical definition in Chapter 6. We briefly recall the definition of finite automata. An unranked tree automaton is a tuple $(A, Q, \delta, Q_f))$. A denotes an alphabet, Q a set of states, $Q_f \in Q$ a set of final states and δ a finite partial transition function that associates a new state of Q to a label in A and a regular word language over Q. We say that an unranked tree t is accepted by a tree automaton iff there exists a mapping α from the nodes of t into Q such that:

- If r is the root of t, $\alpha(r) \in Q_f$.
- If x is some node of t labeled with $a \in A$ and $x_1, \ldots, x_n$ is the sequence of children of x, there exists some regular language $L \subseteq Q^*$ such that: $\delta(L, a) = \alpha(x)$ and $\alpha(x_1) \ldots \alpha(x_n) \in L$.

The notion of automaton for forests is defined similarly. Instead of having a set of final states, we have a final regular language $L \subseteq Q^*$.

5.4 Regular Languages of Trees of Bounded Rank

Languages of terms. Fix an alphabet A of rank k for some k. A tree language over A is a set of trees of rank k over A. We write A^{Δ} the tuple $(H, W_1, W_2, \ldots, W_k)$ where H is the set of all trees over A and for all i, W_i is the set of all i-contexts over A. Note that W_1 includes the empty context.

Regular Languages. All the classes we investigate are fragments of the class of regular tree languages. Regular languages are the languages that are definable using a finite automata. We briefly recall the definition of finite automata. An unranked tree automaton is a tuple $(A, Q, \delta, Q_f))$. A denotes an alphabet of rank

k for some k, Q a set of states, $Q_f \in Q$ a set of final states and δ a transition function. For all i, δ associates a new state to each label of rank i along with a sequence of i states. We say that an unranked tree t is accepted by a tree automaton iff there exists a mapping α from the nodes of t into Q such that:

- If r is the root of t, $\alpha(r) \in Q_f$.
- If x is some node of t labeled with $a \in A$ of arity i and $x_1, \ldots, x_i$ is the sequence of children of x, then $\delta(a, \alpha(x_1), \ldots, \alpha(x_i)) = \alpha(x)$.

5.5 Forest Algebras

Forest algebras were introduced by Bojańczyk and Walukiewicz as an algebraic formalism for studying regular forest languages [BW07]. They correspond to semigroups and monoids over words. In the same spirit as we have monoids and semigroups over words, we can define several notions of forest algebra. We consider two such definitions here. We call them forest algebras using monoids and forest algebras using semigroups. Intuitively forest algebras using semigroups correspond to $A^{\bar{\Delta}}$ and forest algebras using monoids to A^{Δ} in the same way that semigroups correspond to A^+ and monoids to A^*. We give a brief summary of the definition of forest algebras and of their important properties. More details can be found in [BW07]. We begin with the definition of forest algebra using semigroups.

A forest algebra using semigroups consists of a pair (H, V) of semigroups, subject to some additional requirements, which we describe below. We write the operation in V multiplicatively and the operation in H additively, although H is not assumed to be commutative.

We require that V acts on the left of H. That is, there is a map $(h, v) \mapsto vh \in H$ such that $w(vh) = (wv)h$ for all $h \in H$ and $v, w \in V$. We require this action to be faithful. This means that for all $v, v' \in V$ if for all $h \in H$, $vh = v'h$ then $v = v'$. We further require that for every $g \in H$ and $v \in V$, V contains elements $(v + g)$ and $(g + v)$ such that $(v + g)h = vh + g, (g + v)h = g + vh$ for all $h \in H$.

A forest algebra using monoids is a forest algebra using semigroups such that the pair (H, V) is a pair of monoids. Notice that $A^{\bar{\Delta}}$ together with the natural actions is a forest algebra using semigroups and A^{Δ} a forest algebra using monoids.

Recognition of a language. A morphism $\alpha : (H_1, V_1) \to (H_2, V_2)$ of forest algebras using semigroups is actually a pair (γ, δ) of semigroup morphisms $\gamma : H_1 \to H_2$, $\delta : V_1 \to V_2$ such that $\gamma(vh) = \delta(v)\gamma(h)$ for all $h \in H$, $v \in V$. However, we will abuse notation slightly and denote both component maps by α.

We say that a forest algebra using semigroups (H, V) *recognizes* a forest language L if there is a morphism $\alpha : A^{\bar{\Delta}} \to (H, V)$ and a subset X of H such that

$L = \alpha^{-1}(X)$. We also say that the morphism α recognizes L. Similarly we can define recognition of a language by forest algebras using monoids. The following Proposition is proved in [BW07].

Proposition 5.1. ([BW07]) *Fix L a forest language over some alphabet A, the following properties are equivalent:*

1. *L is regular.*
2. *$L \cap A^{\bar{\Delta}}$ is recognized by a finite semigroup forest algebra.*
3. *L is recognized by a finite monoid forest algebra.*

Syntactic forest algebra. Consider some forest language L over an alphabet A. We define an equivalence $\sim_L$ over contexts and forests of $A^{\bar{\Delta}}$. Given two forests t_1, t_2 of $A^{\bar{\Delta}}$, we say that $t_1 \sim_L t_2$ iff for any two forests s, s' of A^{Δ} and any context c of $A^{\bar{\Delta}}$, $c \cdot (s + t_1 + s') \in L$ iff $c \cdot (s + t_2 + s') \in L$. Given two contexts c_1, c_2 of $A^{\bar{\Delta}}$ we say that $c_1 \sim_L c_2$ iff for any forest s of $A^{\bar{\Delta}}$, $c_1 \cdot s \sim_L c_2 \cdot s$. This equivalence is of finite index for both forests ans contexts iff L is regular. This yields a forest algebra recognizing L which we call the *syntactic forest algebra using semigroup* of L. Similarly we can define a notion of syntactic forest algebra using monoids of L using the same equivalence over A^{Δ}. Given a tree automaton, both *syntactic forest algebras*, recognizing the same language can be computed. See [BW07] for more details.

Idempotents. As we said in Chapter 1, given any finite semigroup S, there is a number $\omega(S)$ (denoted by ω when S is understood from the context) such that for each element x of S, x^{ω} is an idempotent. Given a forest algebra (H, V) we will denote by $\omega(H, V)$ the product of $\omega(H)$ and $\omega(V)$ and for any element $u \in V$ and $g \in H$ we will write u^{ω} and ωg for the corresponding idempotents.

5.6 Algebra for Trees of Bounded Rank

In order to express our characterizations over classes of languages if trees of bounded rank,, we define our own algebraic formalism. This formalism is intentionally close to the definition of forest algebras [BW07] as we want to be able to compare the ranked and unranked cases.

Notice that this notion of algebra is related to the notion of preclones defined in [ÉW05]. The difference is the preclones work on multi-contexts, which are contexts that may have an arbitrary number of ports with no restriction relative to their position in the tree. Here we restrict our multi-contexts to be i-contexts. Meaning that all the ports are sibling.

In order to simplify the definition we suppose that the trees we consider are unordered. This means that we make no difference between the tree $a(t_1, t_2)$ and the tree $a(t_2, t_1)$. We choose to make this restriction because this simplifies the definitions and all the classes for which we will use k-algebra only contain languages that are closed under commutation of subtrees. However, note that it is possible to extended the definition of k-algebras to ordered trees.

We defined three types of objects for trees of bounded rank, trees, contexts and k-contexts. Our algebra reflects this and also contains three types of objects. The most important object, which corresponds to contexts is the transition monoid of the automaton. The second objects corresponds to trees and corresponds to the set of states of the automaton. Finally the third object corresponds to k-contexts and is very close to the transition function of the automaton. However, we choose this algebraic viewpoint in order to remain close to the notions used for unranked trees.

We first give the formal definition of our algebra, then move on with the definitions of morphisms, recognition of a language and syntactic Algebra of a language.

k-algebras. A k-Algebra is a tuple $(H, V, W_2, ..., W_k)$ where $H, W_2..., W_k$ are sets and V is a monoid, we write $\cdot$ its operation. Intuitively, H corresponds to trees, V to contexts and W_i to i-contexts. We will often write W_1 for V, we separate V from the sets W_i in order to keep notations close to the notations of forest algebras. Several operations are defined on this tuple, each one reflecting the corresponding operation on trees, contexts and i-contexts. The operations of contexts over trees and i-contexts are reflected by actions of V on H and each set W_i. An action of a monoid V on H is function $f : V \times H \to H$ such that $f(v \cdot v', h) = f(v, f(v', h))$. We abusively write all actions $\cdot$ ($f(v, h) = v \cdot h$), we also ask that those actions are faithful, meaning that for $v \neq v'$ the actions of v and v' are different. Finally, for each $2 \leq i \leq k$ we have an operation $\diamond$, from $W_i \times H$ onto W_{i-1}, such that $(w \diamond h) \diamond h' = (w \diamond h') \diamond h$ (since our trees are unordered the order in which we attach trees under k-contexts is not important). Because of this last axiom, if $w \in W_i$, we write $w \diamond (h_1, ..., h_j)$ for $w \diamond h_1 \diamond ... \diamond h_j$ if $j < i$ and $w \cdot (h_1, ..., h_i)$ for $(w \diamond h_1 \diamond ... \diamond h_{i-1}) \cdot h_i$.

Morphisms of k-algebras. We use the usual notion of morphism: A morphism of k algebras $\alpha : (H^1, W_1^1, .., W_k^1) \to (H^2, W_1^2, ..., W_k^2)$ is composed of $k + 1$ applications $\alpha_0 : H^1 \to H^2$ and $\alpha_i : W_i^1 \to W_i^2$ for $1 \leq i \leq k$. We ask α_1 to be a morphism, for all $h \in H^1$ and all $v \in W_1^1$ $\alpha_1(v)\alpha_0(h) = \alpha_0(vh)$ and for all $h \in H^1$, all i such that $2 \leq i \leq k$ and all $w \in W_i^1$, $\alpha_i(w) \diamond \alpha_0(h) = \alpha_{i-1}(v \diamond h)$.

Recognition by a k-algebra. Notice that if A is an alphabet of rank k, A^Δ together with the natural operations is a k-algebra. We say a tree language L is recognized by a k-Algebra $(H, W_1, ..., W_k)$ iff there exists a morphism $\alpha : A^\Delta \to (H, W_1, ..., W_k)$ and a subset G of H such that $L = \alpha^{-1}(G)$. Classical techniques show the following proposition:

Proposition 5.2. *Fix L an unordered language of trees of rank k, the following properties are equivalent:*

1. *L is regular.*
2. *L is recognized by a finite k-algebra.*

Syntactic k-algebra Given a tree language L we define an equivalence relation on A^Δ. Given two trees t, t', $t \sim_L t'$ iff for all contexts p, $pt \in L$ iff $pt' \in L$. Given two contexts q, q', $q \sim_L q'$ iff for all trees t and all contexts p, $pqt \in L$ iff $p'qt \in L$. By induction on i we extend this definition on i-contexts, given q, q' two i-contexts $q \sim_L q'$ iff for all trees t, $q \diamond t \sim_L q' \diamond t'$. Those relations define a congruence over A^Δ and we call the quotient, the syntactic k-algebra of L. The syntactic k-algebra of L recognizes L, if L is regular, it is finite and computable from the automata recognizing L.

Chapter 6

Fragments of First Order Logic over Trees

In this chapter, we define all the fragments of first-order logic that we will study. The situation is more complicated than it was in the setting of words. First some fragments that had the same expressive power in the setting of words are no longer equal on trees. For example, $\Delta_2(<_{\mathbf{v}})$ and $\mathrm{FO}^2(<_{\mathbf{v}})$ are incomparable in terms of expressive power. We consider several predicates on trees. All these predicates are based on two orders. The first one, $<_{\mathbf{v}}$ is the ancestor relation and the second one, $<_{\mathbf{h}}$, is the following sibling relation. The chapter is organized as follows: In Section 6.1 we define first order logic and monadic second order logic over forests. Notice that the definition is actually independent on whether we consider forests, unranked trees or trees of bounded rank. Since trees of bounded rank and unranked trees are in particular forests we choose to give the definitions for forests. Section 6.2 is devoted to first-order logic using only two variables. In Section 6.3 we study unary temporal logic over forests. We will see that contrary to words, depending on the modalities we consider, it is not always equal to FO^2 in terms of expressive power. In Section 6.4 we define Δ_2 on forests, Δ_2 is the last fragment of FO that was related to FO^2 in the word setting. We will see that it no longer has the same expressive power as FO^2. In Section 6.5 we define boolean combinations of existential first-order formulas. Finally, Section 6.6 is devoted to the definition of locally testable tree languages. Note that the primary definition of this class is not logical. Therefore we define notions of LT separately for trees of bounded rank and unranked trees. An overview of the logics we consider as well as a comparison of their expressive power can be found in Figure 6.1.

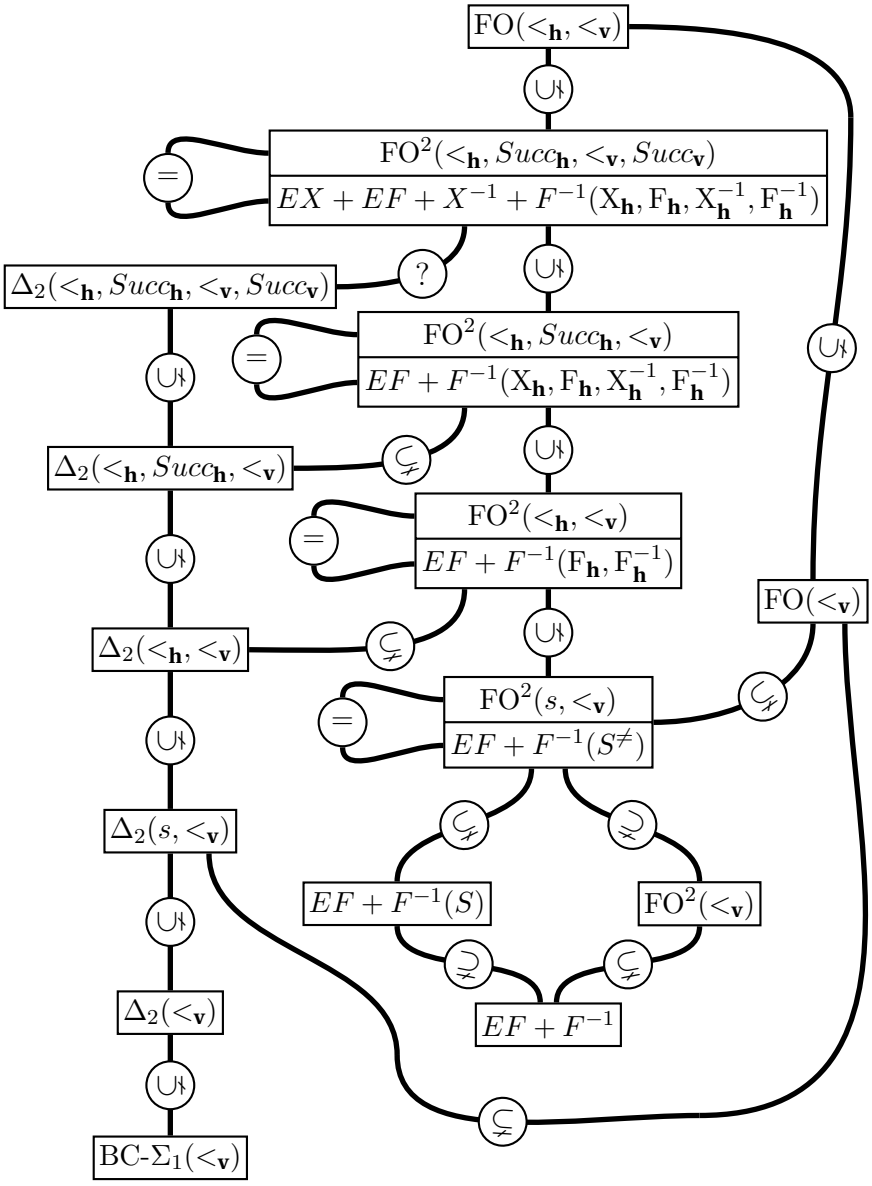


Figure 6.1: Overview of the Logics Considered in Part II

6.1 Monadic Second-order Logic and First-Order Logic

The definition of FO and MSO for forests is similar to the one we gave for words. We view each forest as a relational structure whose domain is its set of nodes. The signature contains several predicates. For every label a in the alphabet A, we have a unary predicate P_a. We have two "vertical" binary relations: $<_{\mathbf{v}}$ for the ancestor relation and $Succ_{\mathbf{v}}$ for the child relation. Finally we have three "horizontal" binary relations: $<_{\mathbf{h}}$ for following sibling, s for the sibling relation and $Succ_{\mathbf{h}}$ for the next sibling relation.

FO. First-order logic, denoted FO($<_{\mathbf{h}}, <_{\mathbf{v}}$), uses first order variables that we will write $x, y, z, \ldots$. A first order-formula is defined by the following grammar (x, y are first-order variables):

$$\begin{aligned}\varphi \quad = \quad & P_a(x) \quad \text{for } a \in A \mid x <_{\mathbf{v}} y \mid Succ_{\mathbf{v}}(x,y) \mid x <_{\mathbf{h}} y \mid s(x,y) \mid Succ_{\mathbf{h}}(x,y) \mid \\ & \mid x = y \mid \exists x\ \varphi \mid \forall x\ \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \Rightarrow \varphi\end{aligned}$$

Given a first-order formula φ we call the set of its *free variables* the set of variables that are not under the scope of some quantifier $\forall$ or $\exists$. If φ has no free variables we say that it is *closed*. Notice that variables may be reused in the same formulas. For example the following formula is a valid first-order formula:

$$\forall x(P_a(x) \vee \exists y(y <_{\mathbf{v}} x \wedge P_b(y) \wedge \exists x(x <_{\mathbf{h}} y \wedge P_a(x))))$$

Fix a first-order formula φ with n free variables $x_1, \ldots, x_n$. We say that a word w together with n positions $i_1, \ldots, i_n$ in w satisfy φ and we write $w, i_1, \ldots, i_n \models \varphi$ iff:

- $\varphi = P_a(x_k)$ and i_k is a labeled with an a.
- $\varphi = x_k = x_l$ and $i_k = i_l$.
- $\varphi = x_k <_{\mathbf{v}} x_l$ and i_l is an ancestor of i_k.
- $\varphi = Succ_{\mathbf{v}}(x_k, x_l)$ and i_l is a child of i_k.
- $\varphi = x_k <_{\mathbf{h}} x_l$ and i_l is a following sibling of i_k.
- $\varphi = s(x_k, x_l)$ and i_k, i_l are siblings.
- $\varphi = Succ_{\mathbf{h}}(x_k, x_l)$ and i_l is the next sibling of i_k.
- $\varphi = \varphi_1 \wedge \varphi_2$ and $w, i_1, \ldots, i_n \models \varphi_1$ and $w, i_1, \ldots, i_n \models \varphi_2$.

- $\varphi = \varphi_1 \vee \varphi_2$ and $w, i_1, \ldots, i_n \models \varphi_1$ or $w, i_1, \ldots, i_n \models \varphi_2$.
- $\varphi = \neg\varphi'$ and $w, i_1, \ldots, i_n \models \varphi'$ is false.
- $\varphi = \varphi_1 \Rightarrow \varphi_2$ and $w, i_1, \ldots, i_n \models \neg\varphi_1 \vee \varphi_2$.
- $\varphi = \exists x_{n+1}\ \varphi'$ and there exists a node i_{n+1} in w such that: $w, i_1, \ldots, i_n, i_{n+1} \models \varphi'$.
- $\varphi = \forall x_{n+1}\ \varphi'$ and for all nodes i_{n+1} in w we have: $w, i_1, \ldots, i_n, i_{n+1} \models \varphi'$.

We say a closed first-order formula φ defines a language L iff $L = \{w \mid w \models \varphi\}$. We write $\mathrm{FO}(<_{\mathbf{v}})$ the restriction of $\mathrm{FO}(<_{\mathbf{h}}, <_{\mathbf{v}})$ to the "vertical" predicates $<_{\mathbf{v}}$ and $Succ_{\mathbf{v}}$. Notice that the relations $Succ_{\mathbf{v}}$ and s can be expressed using $<_{\mathbf{v}}$ and that $Succ_{\mathbf{h}}$ can be expressed using $<_{\mathbf{h}}$.

There is no known decidable characterization of either $\mathrm{FO}(<_{\mathbf{h}}, <_{\mathbf{v}})$ or $\mathrm{FO}(<_{\mathbf{v}})$ whether its on forests, unranked trees or trees of bounded rank. In particular the problem of deciding if a regular tree or forest language is definable in $\mathrm{FO}(<_{\mathbf{v}})$ or in $\mathrm{FO}(<_{\mathbf{h}}, <_{\mathbf{v}})$ remains open.

MSO. We view monadic second-order as an extension of first-order logic. We add a set of second-order variables that we write $X, Y, Z, \ldots$. This variables represent sets of positions. The grammar of FO is now extended by the following rules (with X a second variable and x a first-order variable):

$$\varphi = \exists X\ \varphi \mid \forall X\ \varphi \mid x \in X$$

Fix a monadic second-order formula φ, n free first-order variables $x_1, \ldots, x_n$ and m free second-order variables $X_1, \ldots, X_m$. We extend the semantic of FO in the following way: given n positions $i_1, \ldots, i_n$ in w and m sets of positions $I_1, \ldots, I_n$ we say that they satisfy φ and we write $w, i_1, \ldots, i_n, I_1, \ldots, I_n \models \varphi$ iff:

- $\varphi = \exists X_{n+1}\ \varphi'$ and there exists a set of nodes I_{n+1} of w such that: $w, i_1, \ldots, i_n, I_1, \ldots, I_n, I_{n+1} \models \varphi'$.
- $\varphi = \forall X_{n+1}\ \varphi'$ and for all sets of nodes I_{n+1} of w we have: $w, i_1, \ldots, i_n, I_1, \ldots, I_n, I_{n+1} \models \varphi'$.
- $\varphi = x_k \in X_l\ \varphi'$ and $i_k \in I_l$.

We say a closed MSO formula φ defines a language L iff $L = \{w \mid w \models \varphi\}$. We consider the two logics $\mathrm{MSO}(<_{\mathbf{v}})$ and $\mathrm{MSO}(<_{\mathbf{h}}, <_{\mathbf{v}})$. A very well known result is that regular languages are exactly the languages that are definable in $\mathrm{MSO}(<_{\mathbf{h}}, <_{\mathbf{v}})$:

Theorem 6.1. ([TW68]) *We have the following properties:*

- *A forest language L is definable in* $\mathrm{MSO}(<_{\mathbf{h}}, <_{\mathbf{v}})$ *iff it is regular.*
- *An unranked tree language L is definable in* $\mathrm{MSO}(<_{\mathbf{h}}, <_{\mathbf{v}})$ *iff it is regular.*
- *A language of trees of bounded rank L is definable in* $\mathrm{MSO}(<_{\mathbf{h}}, <_{\mathbf{v}})$ *iff it is regular.*

6.2 First Order Logic Using Only Two Variables

This section is devoted to first-order logic using only two variables. As in the setting of words, FO^2 is the two variable restriction of FO. Because of this restriction it is no longer possible to express s and $Succ_{\mathbf{v}}$ using $<_{\mathbf{v}}$ and to express $Succ_{\mathbf{h}}$ using $<_{\mathbf{h}}$. For this reason we have several more fragments to consider depending on the predicates we include. Sorted by increasing expressive power we consider the following logics: $\mathrm{FO}^2(<_{\mathbf{v}})$, $\mathrm{FO}^2(s, <_{\mathbf{v}})$, $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$, $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ and $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$.

Recall that in the setting of words, in terms of expressive power, FO^2 was equivalent to unary temporal logic and to first-order logic restricted to only one quantifier alternation. More precisely, $\mathrm{FO}^2(<)$ had the same expressive power as $F + F^{-1}$ and as $\Delta_2(<)$, and $\mathrm{FO}^2(<, Succ)$ had the same expressive power as $F + F^{-1} + X + X^{-1}$ and as $\Delta_2(<, Succ)$. In Sections 6.3 and 6.4, devoted respectively to unary temporal and one quantifier alternation on trees, we will investigate these relationships in the tree and forest setting. In particular, we will see that the situation is much more complicated than it was in the word setting.

We will see that fragments using horizontal predicates such as $\mathrm{FO}^2(s, <_{\mathbf{v}})$, $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$, $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ and $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$ have the same expressive power as their natural unary temporal logic counterpart. However, we will see that the natural unary temporal logic fragment for $\mathrm{FO}^2(<_{\mathbf{v}})$ is actually less expressive. In Chapter 6.4, we will see that none of the Δ_2 fragments are equivalent to their FO^2 counterpart in the tree and forest setting.

In Chapter 10 we present decidable characterizations for the logics $\mathrm{FO}^2(s, <_{\mathbf{v}})$, $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$. These characterizations are presented in the forest setting using forest algebras. However using the sibling predicate s, it is simple to express that a forest is a tree (the root has no sibling). Therefore, the result extends to unranked trees for the three logics. Moreover, it is possible to bound the number of children the nodes of a tree may have with the following sibling relation ($<_{\mathbf{h}}$). Therefore, the characterizations of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ extend to trees of bounded rank.

6.3 Unary Temporal Logic

We begin with the definition of unary temporal logic (UTL) over trees and forests. For some alphabet A every UTL formula is defined by the following grammar:

$$\begin{aligned}\varphi \quad = \quad & a \in A \mid EF\ \varphi \mid F^{-1}\ \varphi \mid EX\ \varphi \mid X^{-1}\ \varphi \mid F_{\mathbf{h}}\ \varphi \mid F_{\mathbf{h}}^{-1}\ \varphi \mid X_{\mathbf{h}}\ \varphi \mid X_{\mathbf{h}}^{-1}\ \varphi \mid \\ & \mid S\ \varphi \mid S^{\neq}\ \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \Rightarrow \varphi\end{aligned}$$

We say a tree or a forest t together with a node x in t satisfy a UTL formula φ and we write $t, x \models \varphi$ iff:

- $\varphi = a \in A$ and the node x in t is labeled by a.
- $\varphi = EF\ \varphi'$ and there exists a descendant x' of x such that $t, x' \models \varphi'$.
- $\varphi = F^{-1}\ \varphi'$ and there exists an ancestor x' of x such that $t, x' \models \varphi'$.
- $\varphi = EX\ \varphi'$ and there exists a child x' of x such that $t, x' \models \varphi'$.
- $\varphi = X^{-1}\ \varphi'$ and the parent x' of x verifies $t, x' \models \varphi'$.
- $\varphi = F_{\mathbf{h}}\ \varphi'$ and there exists a following sibling x' of x such that $t, x' \models \varphi'$.
- $\varphi = F_{\mathbf{h}}^{-1}\ \varphi'$ and there exists a preceding sibling x' of x such that $t, x' \models \varphi'$.
- $\varphi = X_{\mathbf{h}}\ \varphi'$ and the next sibling x' of x verifies $t, x' \models \varphi'$.
- $\varphi = X_{\mathbf{h}}^{-1}\ \varphi'$ and the previous sibling x' of x verifies $t, x' \models \varphi'$.
- $\varphi = S\ \varphi'$ and some sibling x' of x verifies $t, x' \models \varphi'$ (this is non strict, x' might be x).
- $\varphi = S^{\neq}\ \varphi'$ and some strict sibling x' of x verifies $t, x' \models \varphi'$.
- $\varphi = \varphi_1 \wedge \varphi_2$ and $w, x \models \varphi_1$ and $w, x \models \varphi_2$.
- $\varphi = \varphi_1 \vee \varphi_2$ and $w, x \models \varphi_1$ or $w, x \models \varphi_2$.
- $\varphi = \neg\varphi_1$ and w, x do not satisfy φ_1.
- $\varphi = \varphi_1 \Rightarrow \varphi_2$ and $w, x \models \neg\varphi_1 \vee \varphi_2$.

Using these modalities, we consider several variants: $EF+F^{-1}$, $EF+F^{-1}(S)$, $EF+F^{-1}(S^{\neq})$, $EF+F^{-1}(\mathrm{F}_{\mathbf{h}},\mathrm{F}_{\mathbf{h}}^{-1})$, $EF+F^{-1}(\mathrm{X}_{\mathbf{h}},\mathrm{F}_{\mathbf{h}},\mathrm{X}_{\mathbf{h}}^{-1},\mathrm{F}_{\mathbf{h}}^{-1})$ and $EX+EF+X^{-1}+F^{-1}(\mathrm{X}_{\mathbf{h}},\mathrm{F}_{\mathbf{h}},\mathrm{X}_{\mathbf{h}}^{-1},\mathrm{F}_{\mathbf{h}}^{-1})$. Given some forest t we say that t satisfies an UTL formula φ iff there exists some root r in t such that $t,r \models \varphi$. Note that a more natural definition would be to fix r as the root of the leftmost tree in the forest. This does not work for $EF+F^{-1}$, $EF+F^{-1}(S)$ and $EF+F^{-1}(S^{\neq})$. These logics do not have enough expressive power to detect that a tree in the forest is the leftmost one. Therefore choosing the leftmost root to evaluate the formula would distinguish this root. For more expressive logics, $EF+F^{-1}(\mathrm{F}_{\mathbf{h}},\mathrm{F}_{\mathbf{h}}^{-1})$, $EF+F^{-1}(\mathrm{X}_{\mathbf{h}},\mathrm{F}_{\mathbf{h}},\mathrm{X}_{\mathbf{h}}^{-1},\mathrm{F}_{\mathbf{h}}^{-1})$ and $EX+EF+X^{-1}+F^{-1}(\mathrm{X}_{\mathbf{h}},\mathrm{F}_{\mathbf{h}},\mathrm{X}_{\mathbf{h}}^{-1},\mathrm{F}_{\mathbf{h}}^{-1})$ the two definitions are equivalent in terms of expressive power. We say that a language L is recognized by an UTL formula φ iff $L=\{t \mid t \models \varphi\}$.

6.3.1 Relations with First-Order Logic Using Only Two Variables

The logics $EF+F^{-1}$ and $EF+F^{-1}(S)$ have no FO^2 equivalent in terms of expressive power. The reason is that these two logics are closed under bisimulation. This means that if a language is definable in $EF+F^{-1}$ or $EF+F^{-1}(S)$ it is closed under the action of duplicating subtrees. In particular, if L is definable in $EF+F^{-1}$ or $EF+F^{-1}(S)$, then the tree a is in L iff the tree $a+a$ is in L. This is not the case in general for languages definable in FO^2, for example the following language contains all forests containing two nodes labeled with a, which is not closed under bisimulation:

$$\exists x \exists y\ a(x) \wedge a(y) \wedge x \neq y$$

However for expressive logics, FO^2 and UTL remain equivalent in terms of expressive power:

Theorem 6.2. ([Mar05]) *Fix L a regular language (L may be a language of unranked trees, trees of bounded rank or forests). We have the following properties:*

- *L definable in $\mathrm{FO}^2(s,<_{\mathbf{v}})$ iff*
  L definable in $EF+F^{-1}(S^{\neq})$.
- *L definable in $\mathrm{FO}^2(<_{\mathbf{h}},<_{\mathbf{v}})$ iff*
  L definable in $EF+F^{-1}(\mathrm{F}_{\mathbf{h}},\mathrm{F}_{\mathbf{h}}^{-1})$.
- *L definable in $\mathrm{FO}^2(<_{\mathbf{h}},Succ_{\mathbf{h}},<_{\mathbf{v}})$ iff*
  L definable in $EF+F^{-1}(\mathrm{X}_{\mathbf{h}},\mathrm{F}_{\mathbf{h}},\mathrm{X}_{\mathbf{h}}^{-1},\mathrm{F}_{\mathbf{h}}^{-1})$.
- *L definable in $\mathrm{FO}^2(<_{\mathbf{h}},Succ_{\mathbf{h}},<_{\mathbf{v}},Succ_{\mathbf{v}})$ iff*
  L definable in $EX+EF+X^{-1}+F^{-1}(\mathrm{X}_{\mathbf{h}},\mathrm{F}_{\mathbf{h}},\mathrm{X}_{\mathbf{h}}^{-1},\mathrm{F}_{\mathbf{h}}^{-1})$.

Proof. This is proved using the same techniques as the for the corresponding theorem in the word setting: Theorem 2.4. We just need to show that the following FO^2 formula is definable in UTL:

$$\exists y \Psi_1(x,y) \wedge P_a(x) \wedge \Psi_2(y)$$

When Ψ_1 is a conjunction of atomic formulas and we have a UTL formula $[\Psi_2]$ such that $w, i \models \Psi_2(y)$ iff $w, i \models [\Psi_2]$. We give the translation tables for all logics and all maximal coherent conjunctions Ψ_1. We begin with $EF + F^{-1}(S^{\neq})$:

Ψ_1	UTL formula
$x = y$	$a \wedge [\Psi_2]$
$s(x,y) \wedge x \neq y$	$a \wedge S^{\neq}[\Psi_2]$
$<_{\mathbf{v}} (x,y)$	$a \wedge EF[\Psi_2]$
$<_{\mathbf{v}} (y,x)$	$a \wedge F^{-1}[\Psi_2]$
$x \neq y \wedge \neg x <_{\mathbf{v}} y \wedge \neg y <_{\mathbf{v}} x \wedge \neg s(x,y)$	$a \wedge F^{-1}\ S^{\neq}\ EF[\Psi_2]$

For $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$:

Ψ_1	UTL formula
$x = y$	$a \wedge [\Psi_2]$
$<_{\mathbf{h}} (x,y)$	$a \wedge F_{\mathbf{h}}[\Psi_2]$
$<_{\mathbf{h}} (y,x)$	$a \wedge F_{\mathbf{h}}^{-1}[\Psi_2]$
$x <_{\mathbf{v}} y$	$a \wedge EF[\Psi_2]$
$y <_{\mathbf{v}} x$	$a \wedge F^{-1}[\Psi_2]$
$x \neq y \wedge \neg x <_{\mathbf{v}} y \wedge \neg y <_{\mathbf{v}} x \wedge \neg <_{\mathbf{h}} (x,y) \wedge \neg <_{\mathbf{h}} (y,x)$	$a \wedge F^{-1}\ S^{\neq}\ EF[\Psi_2]$

For $EF + F^{-1}(\mathrm{X}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}, \mathrm{X}_{\mathbf{h}}^{-1}, \mathrm{F}_{\mathbf{h}}^{-1})$:

Ψ_1	UTL formula
$x = y$	$a \wedge [\Psi_2]$
$<_{\mathbf{h}} (x,y) \wedge \neg Succ_{\mathbf{h}}(x,y)$	$a \wedge X_{\mathbf{h}}\ F_{\mathbf{h}}[\Psi_2]$
$Succ_{\mathbf{h}}(y,x)$	$a \wedge X_{\mathbf{h}}[\Psi_2]$
$<_{\mathbf{h}} (y,x) \wedge \neg Succ_{\mathbf{h}}(y,x)$	$a \wedge X_{\mathbf{h}}^{-1}\ F_{\mathbf{h}}^{-1}[\Psi_2]$
$Succ_{\mathbf{h}}(y,x)$	$a \wedge X_{\mathbf{h}}^{-1}[\Psi_2]$
$<_{\mathbf{v}} (x,y)$	$a \wedge EF[\Psi_2]$
$<_{\mathbf{v}} (y,x)$	$a \wedge F^{-1}[\Psi_2]$
$x \neq y \wedge \neg x <_{\mathbf{v}} y \wedge \neg y <_{\mathbf{v}} x \wedge \neg <_{\mathbf{h}} (x,y) \wedge \neg <_{\mathbf{h}} (y,x)$	$a \wedge F^{-1}\ S^{\neq}\ EF[\Psi_2]$

For $EX + EF + X^{-1} + F^{-1}(\mathrm{X}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}, \mathrm{X}_{\mathbf{h}}^{-1}, \mathrm{F}_{\mathbf{h}}^{-1})$:

Ψ_1	UTL formula
$x = y$	$a \wedge [\Psi_2]$
$<_{\mathbf{h}} (x, y) \wedge \neg Succ_{\mathbf{h}}(x, y)$	$a \wedge X_{\mathbf{h}}\ F_{\mathbf{h}}[\Psi_2]$
$Succ_{\mathbf{h}}(y, x)$	$a \wedge X_{\mathbf{h}}[\Psi_2]$
$<_{\mathbf{h}} (y, x) \wedge \neg Succ_{\mathbf{h}}(y, x)$	$a \wedge X_{\mathbf{h}}^{-1}\ F_{\mathbf{h}}^{-1}[\Psi_2]$
$Succ_{\mathbf{h}}(y, x)$	$a \wedge X_{\mathbf{h}}^{-1}[\Psi_2]$
$<_{\mathbf{v}} (x, y) \wedge \neg Succ_{\mathbf{v}}(x, y)$	$a \wedge EX\ EF[\Psi_2]$
$Succ_{\mathbf{v}}(x, y)$	$a \wedge EX[\Psi_2]$
$<_{\mathbf{v}} (y, x) \wedge \neg Succ_{\mathbf{v}}(y, x)$	$a \wedge X^{-1}\ F^{-1}[\Psi_2]$
$Succ_{\mathbf{v}}(y, x)$	$a \wedge X^{-1}[\Psi_2]$
$x \neq y \wedge \neg x <_{\mathbf{v}} y \wedge \neg y <_{\mathbf{v}} x \wedge \neg <_{\mathbf{h}} (x, y) \wedge \neg <_{\mathbf{h}} (y, x)$	$a \wedge F^{-1}\ S^{\neq}\ EF[\Psi_2]$

□

6.3.2 Decidable Characterizations

In Chapter 10, we present decidable characterizations for $\mathrm{FO}^2(s, <_{\mathbf{v}})$, $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ in Chapter 10. It follows from Theorem 6.2 that these will also be decidable characterizations for $EF+F^{-1}(S^{\neq})$, $EF+F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ and $EF + F^{-1}(\mathrm{X}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}, \mathrm{X}_{\mathbf{h}}^{-1}, \mathrm{F}_{\mathbf{h}}^{-1})$. The Chapter will also include a decidable characterization for $EF+F^{-1}(S)$. The case of $EX+EF+X^{-1}+F^{-1}(\mathrm{X}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}, \mathrm{X}_{\mathbf{h}}^{-1}, \mathrm{F}_{\mathbf{h}}^{-1})$ remains open.

A decidable characterization for forest languages definable in $EF + F^{-1}$ was presented in [Boj07b]. We quote this characterization below. It uses forest algebras and a specific relation that we define below. Intuitively, this relation compares contexts. Considering a context, a smaller context can be built by deleting the subtrees hanging on the path leading from the root to the port. More formally, given a forest algebra using monoids (H, V) and $u, v \in V$ we write $u \dashv v$ iff:

- $u = u_0 \cdots u_n$
- $v = (u_0 + h_0) \cdots (u_n + h_n)$
- $u_o, \ldots, u_n \in V$, $h_0, \ldots h_n \in H$

Theorem 6.3. ([Boj07b]) *Fix L a regular forest language. L is definable in $EF + F^{-1}$ iff its syntactic forest algebra using monoids verifies:*

$$h + g = g + h \quad \forall h, g \in H \tag{6.1}$$

$$h + h = h \quad \forall h \in H \tag{6.2}$$

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \forall u, v \in V \tag{2.2}$$

$$\begin{gathered}\forall u_1, u_2, v_1, v_2 \in V \quad \textit{such that } u_1 \dashv u_2 \textit{ and } v_1 \dashv v_2 \\ (u_1 v_1)^\omega (u_2 v_2)^\omega = (u_1 v_1)^\omega u_1 v_2 (u_2 v_2)^\omega\end{gathered} \tag{6.3}$$

Notice that the characterization involves Equation (2.2) which characterizes $F + F^{-1}$ in the word setting. It is shown in [Boj07b] that the $\dashv$ relation is computable. Therefore since the syntactic forest algebra of a regular forest language is computable, definability in $EF + F^{-1}$ is decidable property of regular forest languages. Also it is shown using technical arguments in [Boj07b] that the decidability can be adapted for unranked trees. Therefore we have the following corollary:

Corollary 6.4. ([Boj07b]) *It is decidable whether a regular forest language is definable in $EF + F^{-1}$. It is decidable whether a regular unranked tree language is definable in $EF + F^{-1}$.*

However this corollary does not extend to the setting of trees of bounded rank. In particular $EF + F^{-1}$ is not expressive enough to express that a tree is of rank k for some k. Also Equation (6.2) and the $\dashv$ relation do not make sense in the bounded rank setting since they affect the number of children of nodes. Therefore, obtaining a decidable characterization for $EF + F^{-1}$ in the bounded rank involves additional work. We present such a characterization in Chapter 8.

6.4 First-Order Logic Using Only One Quantifier Alternation

First-order logic using only one quantifier alternation is defined as in the setting of words. We consider several formalisms depending on the predicates we consider: $\Delta_2(<_{\mathbf{v}})$ $\Delta_2(s, <_{\mathbf{v}})$, $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$, $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$. We give the definition for $\Delta_2(<_{\mathbf{v}})$. The other logics are defined similarly.

We say that a language is definable in $\Delta_2(<_{\mathbf{v}})$ iff it is definable by two fragments of FO$(<_{\mathbf{v}})$. A $\Sigma_2(<_{\mathbf{v}})$ formula is a FO$(<)$ formula of the form:

$$\exists x_1 ... \exists x_n \forall y_1 ... \forall y_m \ \varphi(x_1, ..., x_n, y_1, ..., y_m)$$

Where φ is a FO$(<_{\mathbf{v}})$ quantifier free formula. A $\Pi_2(<_{\mathbf{v}})$ formula is the negation of a $\Sigma_2(<_{\mathbf{v}})$ formula, meaning a formula of the form:

$$\forall x_1 ... \forall x_n \exists y_1 ... \exists y_m \ \varphi(x_1, ..., x_n, y_1, ..., y_m)$$

Where φ is a FO($<_{\mathbf{v}}$) quantifier free formula. We say a language is definable in $\Delta_2(<_{\mathbf{v}})$ if it is definable by both a $\Sigma_2(<_{\mathbf{v}})$ formula and a $\Pi_2(<_{\mathbf{v}})$ formula. The logics $\Delta_2(<_{\mathbf{h}}, <_{\mathbf{v}})$, $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ and $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$ are defined similarly.

6.4.1 Relations With First-Order Logic Using Only Two Variables

In the setting of words Δ_2 had the same expressive power as FO^2. In particular, $\Delta_2(<)$ had the same expressive power as $\mathrm{FO}^2(<)$ and $\Delta_2(<, Succ)$ the same expressive power as $\mathrm{FO}^2(<, Succ)$. This is no longer true for any of the fragments in the tree and forest setting:

We begin with the logic $\Delta_2(<_{\mathbf{v}})$ which is incomparable with $\mathrm{FO}^2(<_{\mathbf{v}})$ in terms of expressive power. We first give an example of a property definable in $\Delta_2(<_{\mathbf{v}})$ but not in $\mathrm{FO}^2(<_{\mathbf{v}})$. It is simple to see that $\mathrm{FO}^2(<_{\mathbf{v}})$ is closed under the equation $3h = 2h$ over its syntactic forest algebra. However the language of forests that contain three nodes labeled by a is not closed under that equation. We show that we can define it in $\Delta_2(<_{\mathbf{v}})$. We actually exhibit a formula that is both a $\Sigma_2(<_{\mathbf{v}})$ and a $\Pi_2(<_{\mathbf{v}})$ formula:

$$\exists x \exists y \exists z \; x \neq y \wedge y \neq z \wedge x \neq z \wedge P_a(x) \wedge P_a(y) \wedge P_a(z)$$

We now present a property that can be expressed in $\mathrm{FO}^2(<_{\mathbf{v}})$ but not in $\Delta_2(<_{\mathbf{v}})$. Consider the language K of trees such that any node labeled by a has an ancestor labeled by a b. K is definable by the following $\mathrm{FO}^2(<_{\mathbf{v}})$ formula:

$$\forall x \; (\neg P_a(x) \vee \exists y \; (y < x \wedge P_b(y)))$$

However K is not definable in $\Delta_2(<_{\mathbf{v}})$. Indeed consider the following forest with k some integer: $t = (p + c \cdot b \cdot a)^k$. It is simple to prove that for any language definable in $\Delta_2(<_{\mathbf{v}})$, for k big enough, if it contains t then it contains $(p + c \cdot b \cdot a)^k \cdot (p + c \cdot a) \cdot (p + c \cdot b \cdot a)^k$. This last tree contains a a which has no ancestor labeled with a b.

Using the same examples, we can show that $\Delta_2(s, <_{\mathbf{v}})$ and $\mathrm{FO}^2(s, <_{\mathbf{v}})$ are also incomparable.

For the logics $\Delta_2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ we can prove that they are strictly less expressive than respectively $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$. The inclusion is a difficult result. It is a corollary of the decidable characterizations of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ we present in Chapter 10. Once these characterizations are proved it is simple to verify that the characterizations of respectively $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ are implied by definability in

respectively $\Delta_2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$. That the inclusions are strict is proved using the same example as for $\Delta_2(<_{\mathbf{v}})$. The language K of trees such that any node labeled by a has an ancestor labeled by a b cannot be expressed in $\Delta_2(<_{\mathbf{h}}, <_{\mathbf{v}})$ or $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$. But is is expressed by the following FO^2 formula:

$$\forall x \ (\neg P_a(x) \vee \exists y \ (y < x \wedge P_b(y)))$$

The case of $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$ remains open. Using the language K we can still show that the expressive power of $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$ is not included in the expressive power of $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$. However, the inclusion of $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$ in $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$ remains open. Since we have no decidable characterization for $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$ we cannot use the same argument as for the other logics.

6.4.2 Decidable Characterizations

Obtaining decidable characterizations for $\Delta_2(s, <_{\mathbf{v}})$, $\Delta_2(<_{\mathbf{h}}, <_{\mathbf{v}})$, $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ and $\Delta_2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$ remains an open problem for the settings of forest unranked trees and trees of bounded rank.

A decidable characterization for $\Delta_2(<_{\mathbf{v}})$ was introduced in [BS08]. This characterization uses forest algebra and a relation called the piece relation. Intuitively, given a tree, we get a piece by suppressing some nodes and attaching the remaining nodes while preserving the ancestor relation. We give a formal definition below.

Piece Relation. We define a relation $\preceq$ forest algebras using monoids. Given an alphabet A and two forests s, t over A, we say that t is a *piece* of s iff there exists an injective morphism of the nodes of t to the nodes of s that preserves labels and the descendant relation, we write $s \preceq t$ (see Figure 6.2).

Since contexts are forests with a special leaf we can extend this definition to contexts. Given two contexts p, q, we say that q is a piece of p iff seen as trees $q \preceq p$.

Given a forest algebra using monoids (H, V) and a morphism $\alpha : A^{\Delta} \to (H, V)$ we extend the notion of piece to elements of this forest Given $h, g \in H$ we say that h is a *piece* of g and write $h \preceq g$ iff there exists two trees t, s such that $\alpha(t) = h$, $\alpha(s) = g$ and $t \preceq s$. Similarly, given $u, v \in V$ we say that u is a *piece* of v and write $u \preceq v$ if and only if there exists two contexts p, q such that $\alpha(p) = u$, $\alpha(q) = v$ and $p \preceq q$.

Lemma 6.5. ([BSS08, BS08]) *Given a forest algebra and a morphism $\alpha : A^{\Delta} \to (H, V)$. The piece relation, $\preceq$, is computable.*

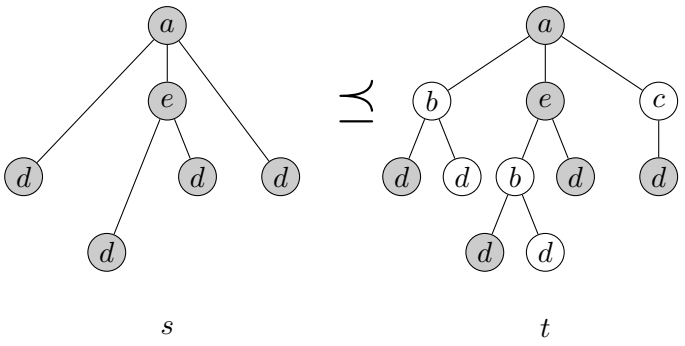


Figure 6.2: Illustration of the Piece Relation on trees.

Theorem 6.6. ([BS08]) *Fix L a regular forest language. L is definable in $\Delta_2(<_{\mathbf{v}})$ iff its syntactic forest algebra using monoids, (H, V), verifies:*

$$h + g = g + h \quad \text{for } h, g \in H \tag{6.4}$$

$$u^\omega = u^\omega v u^\omega \quad \text{for } u, v \in V,\ v \preceq u \tag{6.5}$$

Notice that for $u, v \in V$, $v \preceq uv$, therefore the identity characterizing $\Delta_2(<)$ in the word setting, Equation 2.2 is a consequence of Equation (6.5):

$$(uv)^\omega = (uv)^\omega v (uv)^\omega$$

It follows from Lemma 6.5 that the piece relation is computable, therefore a simple consequence of Theorem 6.6 is that membership in $\Delta_2(<_{\mathbf{v}})$ is a decidable property of forest languages. Also, notice that it is possible to express in $\Delta_2(<_{\mathbf{v}})$ that a forest is a tree with the following $\Pi_2(<_{\mathbf{v}})$ and $\Sigma_2(<_{\mathbf{v}})$ formulas:

$$\begin{array}{ll} \forall x\ \forall y\ \exists z & (z = x \vee z <_{\mathbf{v}} x) \wedge (z = y \vee z <_{\mathbf{v}} x) \\ \exists z\ \forall x & x = z \vee x <_{\mathbf{v}} z \end{array}$$

Therefore, the decidability result also extends to unranked tree lamguages. Altogether, we get the following corollary:

Corollary 6.7. *It is decidable whether a regular forest language is definable in $\Delta_2(<_{\mathbf{v}})$. It is decidable whether a regular unranked tree language is definable in $\Delta_2(<_{\mathbf{v}})$.*

However this corollary does not extend to the setting of trees of bounded rank. In particular $\Delta_2(<_{\mathbf{v}})$ is not expressive enough to express that a tree is of rank k for some k. However, we will see in Chapter 7 that there exists a decidable characterization of $\Delta_2(<_{\mathbf{v}})$ for trees of bounded rank that uses Equation (6.5) with an adapted notion of piece.

6.5 Boolean Combination of Existential First Order Formulas

Boolean combinations of existential first order formulas (BC-$\Sigma_1(<_{\mathbf{v}})$) have the same definition as in the word setting. A $\Sigma_1(<_{\mathbf{v}})$ is a FO($<_{\mathbf{v}}$) formula of the form $\exists x_1 ... \exists x_n \varphi(x_1, ..., x_n)$ with φ a quantifier free formula. A language is definable in BC-$\Sigma_1(<_{\mathbf{v}})$ iff it is definable by a boolean combination of such formulas. We quote the decidable characterization of BC-$\Sigma_1(<_{\mathbf{v}})$ below. This is a result from [BSS08]. It uses forest algebras and the piece relation, $\preceq$, we defined in the previous section for the characterization of $\Delta_2(<_{\mathbf{v}})$.

Theorem 6.8. ([BSS08]) *Fix L a regular forest language. Then L is definable in BC-$\Sigma_1(<_{\mathbf{v}})$ iff its syntactic forest algebra using monoids (H, V) verifies for all $u, v \in V$ such that $v \preceq u$:*

$$h + g = g + h \quad \text{for } h, g \in H \tag{6.6}$$

$$u^\omega = u^\omega v = v u^\omega \tag{6.7}$$

Notice that Equation (2.4) is a consequence of Equation (6.7) which characterizes BC-$\Sigma_1(<)$ on words. Again it is a consequence of Theorem 6.8 that definability in BC-$\Sigma_1(<_{\mathbf{v}})$ is a decidable property of forest languages. Using technical arguments, it is shown in [BSS08] that the decidability result can be extended to unranked trees. Therefore we obtain the following corollary:

Corollary 6.9. *It is decidable whether a regular forest language is definable in BC-$\Sigma_1(<_{\mathbf{v}})$. It is decidable whether a regular unranked tree language is definable in BC-$\Sigma_1(<_{\mathbf{v}})$.*

However this corollary does not extend to the setting of trees of bounded rank. In particular BC-$\Sigma_1(<_{\mathbf{v}})$ is not expressive enough to express that a tree is of rank k for some k. We present a characterization for BC-$\Sigma_1(<_{\mathbf{v}})$ in the setting of trees of bounded rank in Chapter 9. This characterization use k-algebra and an adapted notion of piece. In particular we will see that we need to add new equations in the bounded rank setting.

6.6 Locally Testable Languages

In this section we define the notion of locally testable languages for unranked trees and trees of bounded rank. Since this class is not defined as a fragment of FO the definitions are specific to each setting. Notice that we do not talk about forests for this class. However the definitions for unranked trees could be adapted in a straightforward way to obtain a notion of LT for forests.

6.6.1 Trees of Bounded Rank

For this class we actually work we actually work with binary trees. These are trees for which all nodes have either 2 or 0 children. We make this restriction in order to simplify the notations. However all the definitions an results we will present for LT on binary trees extend to trees of rank k for any fixed k in a straightforward way. We extend to binary trees the notion of types we defined on words in Chapter 2.

Types. Let t be a tree and x be a node of t and k be a positive integer, the *k-type* of x is the (isomorphism type of the) restriction of $t|_x$ to the set of nodes of t at distance at most k from x. When k will be clear from the context we will simply say *type*. A k-type τ *occur* in a tree t if there exists a node of t of type τ. If C is the context $t[x, y]$ for some tree t and some nodes x, y of t, then the k-type of a node of C is the k-type of the corresponding node in t. Notice that the k-type of a node of C depends on the surrounding tree t, in particular the port of C has a k-type, the one of y in t.

Given two trees t and t' we denote by $t \preccurlyeq_k t'$ the fact that all k-types that occur in t also occur in t'. Similarly we can speak of $t \preccurlyeq_k C$ when t is a tree and C is $t'[x, y]$ for some tree t' and some nodes x, y of t'. We denote by $t \simeq_k t'$ the property that the root of t and the root of t' have the same k-type and t and t' agree on their k-types: $t \preccurlyeq_k t'$ and $t' \preccurlyeq_k t$. Note that when k is fixed the number of k-types is finite and hence the equivalence relation $\simeq_k$ has a finite number of equivalence classes. This property is no longer true for unranked trees and this is why we will have to use a different technique for this case.

A language L is said to be κ-locally testable if L is a union of equivalence classes of $\simeq_\kappa$. A language is said to be *locally testable* (is in LT) if there is a κ such that it is κ-locally testable. In words this says that in order to test whether a tree t belongs to L it is enough to check for the presence or absence of κ-types in t, for some big enough κ.

Decidable Characterization. We present a decidable characterization of LT for binary trees in Chapter 11. This characterization is presented in an unusual way. We were not able to obtain a reasonably simple set of identities for characterizing LT similar to the ones we provide in Theorem 4.1 for the word setting. Nevertheless we can show that membership in LT is a decidable property of regular binary tree languages.

6.6.2 Unranked Trees

In the unranked tree case, there are several natural definitions of LT. We will consider two such notions. Both these notions are unable to express an order on

siblings. Therefore in order to simplify the notations we will suppose that our unranked trees are unordered. This means that no order is assumed on children. In particular even if a node has only two children we can not necessarily distinguish the left child from the right child.

Our goal is to adapt the notion of LT we defined for binary trees to unranked unordered trees. Recall the definition of k-type: the k-type of a node x is the isomorphism type of the subtree induced by the descendant of x at distance at most k from x. With unranked trees this definition generates infinitely many k-types. We therefore introduce a more flexible notion of type, (k, l)-type, based on one extra parameter l restricting the horizontal information. It is defined by induction on k. Consider an unordered tree t and a node x of t. For $k = 0$, the (k, l)-type of x is just the label of x. For $k > 0$ the (k, l)-type of x is the label of x together with, for each $(k-1, l)$-type, the number, up to threshold l, of children of x of this type. The reader can verify that over binary trees, the $(k, 2)$-type and the k-type of x always coincide. As in the previous section we say that two trees are (k, l)-equivalent, and denote this using $\simeq_{(k,l)}$, if they have the same occurrences of (k, l)-types and their roots have the same (k, l)-type. We also use $t \preccurlyeq_{(k,l)} t'$ to denote the fact that all (k, l)-types of t also occur in t'.

Based on this new notion of type, we define two notions of locally testable languages. The most expressive one, denoted ALT (A for *Aperiodic*), is defined as follows. A language L is in (κ, λ)-ALT if it is a union of (κ, λ)-equivalence classes. A language L is in ALT if there is a k and a λ such that L is in (κ, λ)-ALT.

The second one, denoted ILT in the sequel (I for *Idempotent*), assumes $\lambda = 1$: A language L is in ILT if there is a κ such that L is a union of $(\kappa, 1)$-equivalence classes.

Decidable Characterization. We present decidable characterizations for both ILT and ALT for unranked unordered trees in Chapter 11. The characterization of ILT is presented using a set of identities expressed with the formalism used in [BS09] for the decidable characterization of LTT on trees. The characterization of ALT is presented in the same way as the characterization for the trees of bounded rank setting. We were not able to obtain a reasonably simple set of identities for characterizing ALT.

Logical characterization There is a logical characterization of languages that are locally testable. It corresponds to the languages definable by a temporal logic defined by the following grammar:

$$\varphi = a \in A \mid EX\varphi \mid G\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi$$

Intuitively, **G** stands for "everywhere in the tree" while **EX** stands for "child".

In the binary tree case, we also requires two predicates distinguishing the left child from the right child. In the unranked unordered setting the logic above is closed under bisimulation and therefore correspond to ILT. Since this corresponds exactly to the logical characterization of LT in the word setting (see Chapter 2), this shows that in a sense ILT is the natural extension of LT to the unranked setting.

Chapter 7

One Quantifier Alternation over Trees of Bounded Rank

In this Chapter we provide a decidable characterization for the class of languages of trees of bounded rank that are definable in $\Delta_2(<_{\mathbf{v}})$. We briefly recall the definition of $\Delta_2(<_{\mathbf{v}})$, see Chapter 6 for more details. We say that that a language is definable in $\Delta_2(<_{\mathbf{v}})$ iff it is definable by both a $\Sigma_2(<_{\mathbf{v}})$ formula and a $\Pi_2(<_{\mathbf{v}})$ formula. A $\Sigma_2(<_{\mathbf{v}})$ formula is a first-order formula using only one quantifier alternation starting with an existential quantification. A $\Pi_2(<_{\mathbf{v}})$ formula is the negation of a $\Sigma_2(<_{\mathbf{v}})$ formula.

We already quoted decidable characterizations for $\Delta_2(<_{\mathbf{v}})$ in the settings of words and forests. In Chapter 2 we presented the following result using monoids:

Theorem 2.6. ([PW97]) *Fix L a regular word language. Then L is definable in $\Delta_2(<)$ iff its syntactic monoid M verifies for all $u, v \in M$:*

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \tag{2.2}$$

In the forest setting, we quoted a decidable characterization using forest algebras together with a relation called the piece relation that we defined in Chapter 6.

Theorem 6.6. ([BS08]) *Fix L a regular forest language. L is definable in $\Delta_2(<_{\mathbf{v}})$ iff its syntactic forest algebra using monoids, (H, V), verifies:*

$$h + g = g + h \quad \text{for } h, g \in H \tag{6.4}$$

$$u^\omega = u^\omega v u^\omega \quad \text{for } u, v \in V,\ v \preceq u \tag{6.5}$$

Notice that Equation (2.2) is a consequence of Equation (6.5). Therefore Theorem 6.6 generalizes Theorem 2.6. However, Theorem 6.6 does not apply to

trees of bounded rank. In particular, $\Delta_2(<_{\mathbf{v}})$ is not expressive enough to express that a tree is of rank k for some fixed integer k.

We present a decidable characterization for the languages of trees of rank k for some fixed k that are definable in $\Delta_2(<_{\mathbf{v}})$. Our characterization uses k-algebras (see Chapter 5) together with an equation that is similar to Equation (6.5). This equation uses a notion of piece relation that is adapted to trees of bounded rank. This new piece relation can be viewed as a restriction of the corresponding relation on forests.

In Section 7.1 we present our decidable characterization and discuss its relation with the characterization of [BS08] for the forest setting. Then in Section 7.2 we provide a proof for the difficult "if" direction of this characterization. Throughout this chapter we compare both the statements and the proofs we use to the ones used in [BS08] for the forest setting.

7.1 Characterization of $\Delta_2(<_{\mathbf{v}})$

We begin with the statement of the characterization. Note that like the characterization of [BS08] this characterization uses a *piece relation*. However, we need to adapt this notion to trees of bounded rank. Our notion of piece has to be more restrictive than the one of [BS08]. Intuitively, given a tree, we get a piece by suppressing some nodes and attaching the remaining nodes while preserving the ancestor relation. With such a definition every piece of a forest remains a forest. However with this definition a piece of a tree of bounded rank tree need not be a valid tree of bounded rank. To solve this problem we consider only pieces that are valid trees.

Pieces. Given an alphabet A and two trees s, t over A, we say that t is a *piece* of s iff there exists an injective morphism of the nodes of t to the nodes of s that preserves labels and the descendant relation, we write $s \preceq t$ (see Figure 7.1).

Since i-contexts are trees with special leaves we can extend this definition to i-contexts. Given two i-contexts p, q, we say that q is a piece of p iff seen as trees $q \preceq p$ (see Figure 7.2).

Given a k-algebra $(H, W_1, ..., W_k)$ and a morphism $\alpha : A^\Delta \to (H, W_1, ..., W_k)$ we extend the notion of piece to elements of this k-algebra. Given $h, g \in H$ we say that h is a *piece* of g and write $h \preceq g$ iff there exists two trees t, s such that $\alpha(t) = h$, $\alpha(s) = g$ and $t \preceq s$. Similarly, given $u, v \in W_i$ we say that u is a *piece* of v and write $u \preceq v$ if and only if there exists two i-contexts p, q such that $\alpha(p) = u$, $\alpha(q) = v$ and $p \preceq q$.

Since we want decidability, we need to be able to compute the piece relation given a k-algebra and an associated morphism. This can be proved using the same

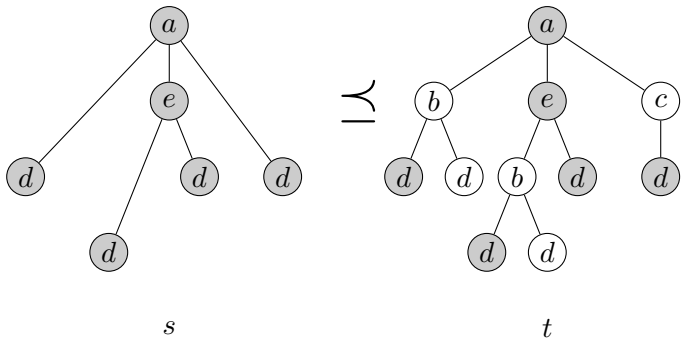


Figure 7.1: Illustration of the Piece Relation on trees.

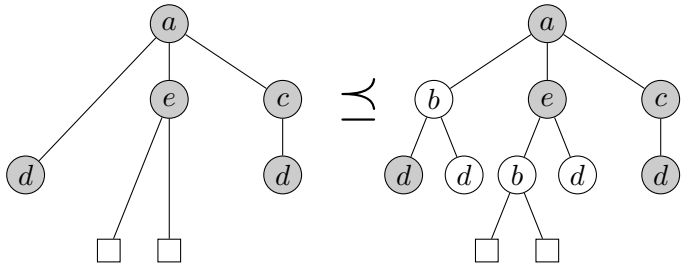


Figure 7.2: Illustration of the Piece Relation on 2-contexts.

arguments as in [BS08] for the unranked setting:

Lemma 7.1. ([BS08]) *Given a k-algebra $(H, W_1, \ldots, W_k)$ and a morphism $\alpha : A^\Delta \to (H, W_1, \ldots, W_k)$ the piece relation, $\preceq$ can be computed.*

We are now ready to state our characterization. It uses k-algebras which are defined in Chapter 5.

Theorem 7.2. *Fix some integer k and L a regular tree language of bounded rank k. L is definable in $\Delta_2(<_{\mathbf{v}})$ iff its syntactic k-algebra verifies the following equation:*

$$u^\omega = u^\omega v u^\omega \quad \forall u, v \in V \text{ such that } v \preceq u \tag{7.1}$$

Our identity is identical to the provided in [BS08], the only difference is the restriction we made while defining the piece relation. Notice that there is no equivalent to Equation (6.4). This equation implied that the languages of forest

definable in $\Delta_2(<_{\mathbf{v}})$ are unordered. However, in our case this property is already assumed in the definition of k-algebras (see Chapter 5).

Notice that Equation (2.2), which characterizes $\Delta_2(<)$ on words in [PW97] (see Theorem 2.6) is a simple consequence of Equation (7.1) (this is because $v \preceq uv$):

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \quad \forall u, v \in V \tag{2.2}$$

In the proof, we will distinguish the cases where we only need Equation (2.2) from the cases where we need the full power of Equation (7.1).

Also notice that since the piece relation is computable (see Lemma 7.1) we get the following corollary from Theorem 7.2:

Corollary 7.3. *It is decidable whether a regular language of trees of bounded rank is definable in* $\Delta_2(<_{\mathbf{v}})$.

We now turn to the proof of Theorem 7.2. Of the three characterizations we present for logics on trees of bounded rank, this one has the most similar proof to its unranked counterpart. We begin we the easier "only if" direction, a $\Delta_2(<_{\mathbf{v}})$ language verifies (7.1). We show that we can actually derive this result from the unranked characterization of [BS08].

Proposition 7.4. *Let L be a tree language of bounded rank k definable in $\Delta_2(<_{\mathbf{v}})$, then the syntactic k-algebra of L verifies* (7.1).

Proof. We use the following Lemma taken from [BS08]:

Lemma 7.5. ([BS08]) *Let φ be a formula of $\Sigma_2(<_{\mathbf{v}})$ and let $q \preceq p$ be two contexts. For $n \in \mathbb{N}$ sufficiently large, trees satisfying φ are closed under replacing $p^n p^n$ with $p^n q p^n$.*

This Lemma was proved for forests, but any tree of bounded rank k can be seen as a forest and the piece relation on trees of bounded rank is a restriction of the piece relation for forests. Therefore the Lemma also holds for trees of bounded rank.

Given this Lemma since, L is definable by both a $\Sigma_2(<_{\mathbf{v}})$ and a $\Pi_2(<_{\mathbf{v}})$ formula, it follows that for $n \in \mathbb{N}$ sufficiently large, L is closed under replacing $p^n p^n$ with $p^n q p^n$.

It follows that the syntactic k-algebra of L verifies (7.1). □

We turn to the other direction of Theorem 7.2, given L a regular tree language, if the syntactic k-algebra of L verifies (7.1), then L is definable in $\Delta_2(<_{\mathbf{v}})$. We actually prove a slightly more general proposition:

Proposition 7.6. *Given $\alpha : A^\Delta \to (H, W_1, ..., W_k)$, with $(H, W_1, ..., W_k)$ satisfying (7.1), for every $h \in H$, the following language is definable in $\Sigma_2(<_{\mathbf{v}})$.*

$$L = \{t \mid \alpha(t) = h\}$$

Before proving Proposition 7.6, we show that it concludes the proof of Theorem 7.2. Indeed it means that $\forall h \in H$ the language $\alpha^{-1}(h)$ is definable in $\Sigma_2(<_{\mathbf{v}})$. We show that it is definable in $\Pi_2(<_{\mathbf{v}})$. Consider $K = \cup_{g \neq h} \alpha^{-1}(g)$, since H is finite K is finite a disjunction of language definable with a $\Sigma_2(<_{\mathbf{v}})$ formula and is therefore itself definable with a $\Sigma_2(<_{\mathbf{v}})$ formula. By definition we have $\alpha^{-1}(h) = \{t \mid t \notin K\}$, because the negation of a $\Sigma_2(<_{\mathbf{v}})$ formula is a $\Pi_2(<_{\mathbf{v}})$ formula, it follows that $\alpha^{-1}(h)$ is definable with a $\Pi_2(<_{\mathbf{v}})$ formula. We conclude that $\alpha^{-1}(h)$ is in $\Delta_2(<_{\mathbf{v}})$.

We devote Section 7.2 to the proof of Proposition 7.6. The proof is similar to the one of [BS08], in particular it uses the same induction structure. The main differences occur in the proofs of the decomposition lemmas that decompose L as a combination of several simpler languages. Technical differences also arise due to the fact that we need to be careful about the arity of the trees we build.

7.2 Proof of Proposition 7.6

For $g \in H$ we write $L_g = \alpha^{-1}(g)$. We also work with languages of contexts, for $u \in V$ we write $K_u = \alpha^{-1}(u)$. We state here a composition lemma that we will use in the proof:

Lemma 7.7. *Let K a context language definable in $\Sigma_2(<_{\mathbf{v}})$, $a \in A_i$ for some $i \leq k$ and $L_1, ..., L_i$ tree languages definable in $\Sigma_2(<_{\mathbf{v}})$. The language, $Ka(L_1, ..., L_i)$ is definable in $\Sigma_2(<_{\mathbf{v}})$.*

Proof. The formula quantifies existentially over the position labeled with a and relativizes the formulas for $K, L_1, ..., L_i$ to this position. □

Given this lemma we proceed with the proof. It goes by induction on two parameters, the size of the k-algebra and the the position of h relatively to a pre-order we define below.

Reachability We define the pre-order on H. This pre-order is the adaptation on trees of the reachability order we defined on words back in Chapter 3. Given $h, g \in H$, we say that h is *reachable* from g iff there exists $v \in V$ such that $h = vg$. If h and g are mutually reachable, we write $h \sim g$ and one can verify that $\sim$ is an equivalence relation. There exists a maximal class of $\sim$ regarding reachability, indeed if we consider some $w \in W_2$, and $H = \{h_1, ..., h_n\}$, the type

$(w \diamond h_1) \cdot ... \cdot (w \diamond h_{n-1}) \cdot h_n$ is reachable from any type in H. We call this maximal class H_m.

We prove Proposition 7.6 by induction on the size of the algebra $(H, W_1, ..., W_k)$ then the number of types that are reachable from h. We distinguish two cases depending on whether $h \in H_m$ or not.

7.2.1 $h \notin H_m$

As in [BS08], we use the notion of *stabilizer* of h. We then use Equation (7.1) to show this stabilizer depends only on the $\sim$-class of h. However our algebra contains one object for every possible arity, this leads us to the definition of a stabilizer for every arity. Take $i \in \{0, k\}$ and $h \in H$ we define the *i-stabilizer* of h as follows:

$$\begin{aligned} stab_0(h) &= \{g \in H : \exists w \in W_2 \ w(h, g) \sim h\} \\ stab_1(h) &= \{v \in V : v \cdot h \sim h\} \\ stab_i(h) &= \{w \in W_i : \exists g \in H \ w \diamond g \in stab_{i-1}(h)\} \ i \in \{2, k\} \end{aligned}$$

For k-algebras satisfying (7.1), the stabilizers have two crucial properties. The first property which is only a consequence of (2.2) is that the stabilizers only depends on the $\sim$-class of h. The second property which is a consequence of the full equation (7.1) is that the stabilizers are closed under pieces.

Lemma 7.8. *For $i \in \{0, k\}$ and $h, h' \in H$ such that $h \sim h'$ we have $stab_i(h) = stab_i(h')$.*

Proof. We give the proof for the 0-stabilizer and the 1-stabilizer, the result for $i > 1$ then follows by a simple induction on the definition.

Let $h \sim h'$ and $g \in stab_0(h)$, we show that $g \in stab_0(h')$. By definition, there exists $w \in W_2$ such that $(w \diamond g)h \sim h \sim h'$, by definition of $\sim$ this means that there exists $u, v \in V$ such that $h = uh'$ and $h' = v(w \diamond g)h$. We get that:

$$\begin{aligned} h' &= v(w \diamond g)uh' \\ h' &= (v(w \diamond g)u)^\omega h' \\ h' &= (v(w \diamond g)u)^\omega (w \diamond g)(v(w \diamond g)u)^\omega h' \text{ using (2.2)} \\ h' &= (v(w \diamond g)u)^\omega (w \diamond g)h' \end{aligned}$$

Therefore, $g \in stab_0(h')$.

Now consider $v \in stab_1(h)$, we show that $v \in stab_1(h')$. There exists $x, y \in V$ such that $h = xh'$ and $h' = yvh$. We get that:

$$\begin{aligned} h' &= yvxh' \\ h' &= (yvx)^\omega h' \\ h' &= (yvx)^\omega v(yvx)^\omega h' \text{ using (2.2)} \\ h' &= (yvx)^\omega v h' \end{aligned}$$

Therefore $v \in stab_1(h')$. □

Lemma 7.9. *For $i \in \{0, k\}$, let $w \in stab_i(h)$ and $w' \preceq w$, then $w' \in stab_i(h)$.*

Proof. We begin with $i = 0$, take $g \in stab_0(h)$ and $g' \preceq g$, there exists $w \in W_2$ and $x \in V$ such that $x(w \diamond g)h = h$, it follows that $(x(w \diamond g))^\omega h = h$. By definition $w \diamond g' \preceq w \diamond g$, therefore using (7.1) we get:

$$h = (x(w \diamond g))^\omega (w \diamond g')(x(w \diamond g))^\omega h = (x(w \diamond g))^\omega (w \diamond g')h$$

It follows that $g' \in stab_0(h)$.

Now suppose that $i = 1$, take $v \in stab_1(h)$ and $v' \preceq v$, there exists x such that $xvh = h$, it follows that $(xv)^\omega h = h$. Using (7.1), $h = (xv)^\omega v'(xv)^\omega h = (xv)^\omega uh$, $v' \in stab_1(h)$.

For $i \geq 2$, the result follows by a simple induction on the definition. □

Like in [BS08], we consider two cases depending on whether $h \in stab_0(h)$. If $h \in stab_0(h)$ we show that L is recognized by a smaller algebra and conclude by induction on the size of the algebra. If $h \notin stab_0(h)$ we conclude by decomposing L as several languages that we show to be definable in $\Sigma_2(<_{\mathbf{v}})$ using the induction hypothesis on the reachability order.

h is a 0-stabilizer of h In this case we show that L_h is recognized by a smaller k-algebra and conclude using the induction hypothesis. We show that $(stab_0(h), stab_1(h), ..., stab_k(h))$ is a k-algebra and that it recognizes L_h. Because $h \notin H_m$ we have by definition of H_m, $H_m \cap stab_0(h) = \emptyset$, it follows that $stab_0(h) \subsetneq H$ and that $(stab_0(h), stab_1(h), ..., stab_k(h))$ is of smaller size than $(H, W_1, ..., W_k)$.

Lemma 7.10. *$(stab_0(h), stab_1(h), ..., stab_k(h))$ is a k-algebra.*

Proof. We show that $(stab_0(h), stab_1(h), ..., stab_k(h))$ is preserved under all the operations of k-algebras.

$$\begin{aligned} stab_1(h)stab_1(h) &\subseteq stab_1(h) \\ stab_1(h)stab_i(h) &\subseteq stab_i(h) \text{ for } i \geq 2 \\ stab_i(h) \diamond stab_0(h) &\subseteq stab_{i-1}(h) \text{ for } i \geq 2 \\ stab_1(h)stab_0(h) &\subseteq stab_0(h) \end{aligned}$$

The first item is a consequence of Lemma 7.8. Let $u, v \in stab_1(h)$ we have $vh \sim h$, therefore by Lemma 7.8 $uvh \sim h$, $uv \in stab_1(h)$. The second item follows by induction on i.

Using Lemma 7.9 we show that the third and fourth items hold.

We prove the third item, take $i > 2$, $g \in stab_0(h)$ and $w \in stab_i(h)$. By definition, there exists $w' \in W_2$ and $g_1, ..., g_{i-1}$ such that $(w \diamond (g_1, ..., g_{i-1})) \in stab_1(h)$ and $w' \diamond g \in stab_1(h)$. Therefore using the first item we obtain that $(w \diamond (g_1, ..., g_{i-1}))(w' \diamond g) \in stab_1(h)$, since $(w \diamond (g, g_2, ..., g_{i-1})) \preceq (w \diamond (g_1, ..., g_{i-1}))(w' \diamond g)$, it follows from Lemma 7.9 that $(w \diamond (g, g_2, ..., g_{i-1})) \in stab_1(h)$ and $w \diamond g \in stab_{(i-1)}(h)$.

We move to the fourth item, let $v \in stab_1(h)$ and $g \in stab_0(h)$, we have $w \in W_2$ such that $w \diamond g \in stab_1(h)$. There exists u and u' in V such that $h = uvh$ and $h = u'(w \diamond g)h$, combining both we get $h = uvu'(w \diamond g)h$. Since $vg \preceq uvu'w(h, g) = h$ and by hypothesis $h \in stab_0(h)$, it follows from Lemma 7.9 that $vg \in stab_0(h)$. □

h is not a 0-stabilizer of h In this case we describe L_h as a union of several languages and show that each of them is definable in $\Sigma_2(<_{\mathbf{v}})$. Let G be the set of types g such that h is reachable from g but g is not reachable from h. By induction hypothesis, for all $g \in G$, L_g is definable in $\Sigma_2(<_{\mathbf{v}})$. Notice that $stab_0(h) \subseteq G$, indeed for all $g \in stab_0(h)$, h is reachable from g, but by Lemma 7.9, $stab_0(h)$ is closed under pieces and therefore if g was reachable from h we would have $h \in stab_0(h)$ which is false by hypothesis.

Lemma 7.11. *A tree t belongs to L_h iff it belongs to one of the following type of languages:*

$K_u a_0$	$u \in stab_1(h)$		$u\alpha(a_0) = h$	$a_0 \in A_0$
$K_u a_i(L_{g_1}, ..., L_{g_i})$	$u \in stab_1(h)$	$g_1, .., g_i \in G$	$u\alpha(a_i)(g_1, ..., g_i) = h$	$a_i \in A_i$

Proof. By definition, if t belongs to any such language, $\alpha(t) = h$, therefore $t \in L_h$. Now assume that $t \in L_h$, let t' be a subtree of t such that $\alpha(t') \sim h$ and no subtree of t' is mutually reachable with h and let p such that $t = pt'$. It follows that $u = \alpha(p) \in stab_1(h)$, then depending on whether t is a leaf or not, t is in a language of the first kind or of the second kind. □

We show that the languages described in Lemma 7.11 are definable in $\Sigma_2(<_{\mathbf{v}})$. It will then follow from Lemma 7.7 that L_h is definable in $\Sigma_2(<_{\mathbf{v}})$, concluding this case. By induction we know that for every $g \in G$ L_g is definable in $\Sigma_2(<_{\mathbf{v}})$. It follows that it is sufficient to prove that for any $u \in stab_1(h)$, K_u is definable in $\Sigma_2(<_{\mathbf{v}})$.

Lemma 7.12. *For any $u \in stab_1(h)$ K_u is definable in $\Sigma_2(<_{\mathbf{v}})$.*

Proof. This is done using the same techniques as in [BS08]. We see K_u as a word language over an infinite alphabet and conclude using the following Proposition taken from [BS08].

A *stratified monoid* is a monoid along with a pre-order $\preceq$ that satisfies the following property:

$$m^\omega n m^\omega = m^\omega \quad \text{for } n \preceq m$$

Proposition 7.13. *([BS08]) Let B be an alphabet (possibly infinite), and let $\beta : B^* \to M$ be a morphism into a stratified monoid $(M, \preceq)$ that satisfies the following identity:*

$$(mn)^\omega m (mn)^\omega = (mn)^\omega \tag{7.2}$$

For any $m \in M$, the language $\beta^{-1}(m)$ is defined by a finite union of expressions

$$B_0^* C_1 B_1^* ... C_i B_i^*$$

where each C_j is of the form $B \cap \beta^{-1}(n)$ for some $n \in M$, and each B_j is of the form $B \cap \beta^{-1}(N)$ for dome $N \subseteq M$ closed under $\preceq$.

Now we define an infinite alphabet B, B contains two types of letters which are all contexts of over A:

- $a \in A_1$.
- $a \diamond (t_1, ..., t_{i-1})$ for $i \geq 2$ and $a \in A_i$.

Consider the morphism $\beta : B \to V$ which is alpha restricted to contexts of B. By construction we have $K_u = \beta^{-1}(u)$. Now observe that together with the piece relation $(V, \preceq)$ is a stratified monoid, therefore Proposition 7.2.1 applies we get that K_u is of the form:

$$(B \cap \beta^{-1}(N_0))^* (B \cap \beta^{-1}(n_1)) ... (B \cap \beta^{-1}(n_k))^* (B \cap \beta^{-1}(N_k))^*$$

where $n_1, ..., n_k$ are elements of V, and $N_1, ..., N_k$ are subsets of V that are closed under the piece relation.

We just need to show that the expressions used are definable in $\Sigma_2(<_{\mathbf{v}})$. For expressions of the form $(B \cap \beta^{-1}(N))^*$ where N is closed under pieces, it follows that $(B \cap \beta^{-1}(N))^*$ is closed under pieces and is therefore definable in $\Sigma_2(<_{\mathbf{v}})$. Languages of the form $(B \cap \beta^{-1}(n_1))$ are just union of singletons $\{b\}$ for $b \in B$. We show that we can define such singletons in $\Sigma_2(<_{\mathbf{v}})$, it is clear $a \in A_0$ now consider $b = a \diamond (t_1, ..., t_{i-1})$ for $a \in A_i$, we show that we can define the language $\{a \diamond (t'_1, ..., t'_{i-1}) \mid \alpha(t_j) = \alpha(t'_j),\ 1 \leq j \leq i-1\}$ which concludes the proof. This is because $v \in stab_1(h)$ and therefore it follows from Lemma 7.8 that for all $1 \leq j \leq i-1$, $\alpha(t_i) \in stab_0(h) \subseteq G$ and therefore $\alpha^{-1}(\alpha(t_j))$ is definable in $\Sigma_2(<_{\mathbf{v}})$.

□

7.2.2 $h \in H_m$

We let $G = H - H_m$, it follows from the previous case that for all $g \in G$, L_g is definable in $\Sigma_2(<_{\mathbf{v}})$. Notice that in this case we have $h \in stab_0(h)$, but we cannot conclude using the induction hypothesis since $stab_0(h) = H$. We treat this case using a new induction parameter over an added induction variable $v \in V$. We show that for all $v \in V$ the following language is definable in $\Sigma_2(<_{\mathbf{v}})$:

$$M_{v,h} = \{t : v\alpha t = h\}$$

The result follows since $L_h = M_{1_v,h}$ for 1_v the neutral element of V. Similarly to what we did for H we introduce a pre-order over V and proceed by induction on the position of v regarding this order. Given $u, v \in V$, we say that v is *reachable* from u iff there exists $x \in V$ such that $ux = v$. If $u, v \in V$ are mutually reachablewe write $u \sim v$. We proceed by induction on the number of context types reachable from v. First we prove some properties on reachability over V, they are similar to the ones we used for reachability over H.

We define a notion of *stabilizer* for v. Take $i \in \{0, k\}$ and $h \in H$ we define the *i-stabilizer* of h as follows:

$$\begin{aligned} stab_0(v) &= \{g \in H : \exists w \in W_2\ v(w \diamond g) \sim v\} \\ stab_1(v) &= \{u \in V : vu \sim v\} \\ stab_i(v) &= \{w \in W_i : \exists g \in H\ w \diamond g \in stab_{i-1}(v)\} \text{ for } i \in \{2, k\} \end{aligned}$$

Lemma 7.14. *For $u, v \in V$ such that $u \sim v$ and $i \in \{0, k\}$ then $stab_i(v) = stab_i(u)$.*

Proof. The proof is similar to the proof of Lemma 7.8. We give the proof for the 0-stabilizer and the 1-stabilizer, the result for $i > 1$ then follows by a simple induction on the definition.

Take $v \sim v'$ and $g \in stab_0(v)$, we show that $g \in stab_0(v')$. There exists $w \in W_2$ such that $v(w \diamond g) \sim v \sim v'$, y definition of $\sim$ this means that there exists $x, y \in V$ such that $v = v'x$ and $v' = v(w \diamond g)y$. We get that:

$$\begin{aligned} v' &= v'x(w \diamond g)y \\ v' &= v'(x(w \diamond g)y)^\omega \\ v' &= v'(x(w \diamond g)y)^\omega(w \diamond g)(x(w \diamond g)y)^\omega \text{ using (2.2)} \\ v' &= v'(w \diamond g)(x(w \diamond g)y)^\omega \end{aligned}$$

Therefore, $g \in stab_0(v')$.

Now take $x \in stab_1(v)$, we show that $x \in stab_1(v')$. There exists $y, y' \in V$ such that $v = v'y$ and $v' = vxy'$. We get that:

$$\begin{aligned} v' &= v'yxy' \\ v' &= v'(yxy')^\omega \\ v' &= v'(yxy')^\omega x(yxy')^\omega \text{ using (2.2)} \\ v' &= v'x(yxy')^\omega \end{aligned}$$

Therefore $x \in stab_1(v')$.

□

Lemma 7.15. *The following are equivalent:*

1. *v is maximal with respect to reachability.*
2. *$stab_0(v) \cap H_m \neq \emptyset$.*
3. *For all $g, g' \in H$, $vg = vg'$.*

Proof. First if v is maximal regarding reachability it means that for all $g \in H$ and all $w \in W_2$, v is reachable from $v(w \diamond g)$, $stab_0(v) = H$. Therefore 1) $\Rightarrow$ 2).

Now suppose that there exists $h_m \in H_m$ such that $h_m \in stab_0(v)$. We first show that $H = stab_0 v$, take $h' \in H_m$, by definition $h' \preceq h_m$. There exists $w \in W_2$ such that $v(w \diamond h_m) \sim v$. It follows that there exists $x \in V$ such that $v = v(w \diamond h_m)x = v((w \diamond h_m)x)^\omega$, since $(w \diamond h') \preceq (w \diamond h_m)$ it follows from (7.1) that $v = v(w \diamond h')((w \diamond h_m)x)^\omega$, $h \in stab_0(v)$. We write $H = \{h_1, ..., h_n\}$, and take $w \in W_2$, $e = ((w \diamond h_1)...(w \diamond h_n))^\omega$. From our hypothesis on v there exists $x \in V$ such that:

$$ve(w \diamond eh_1) \cdot x = v$$

Using (2.2) we get that $\forall g \in H \quad e(w \diamond eh_1)ug = eh_1$, hence $\forall g \in H \quad vg = veh_1$. Therefore 2) $\Rightarrow$ 3).

Finally suppose that for all $g, g' \in H$, $vg = vg'$ and take $u \in V$, we show that $vu = v$, we have for all g, $vug = vg$, it follows that $vu = v$. Therefore 3) $\Rightarrow$ 1).

□

With these definitions we prove that $M_{v,h}$ is definable in $\Sigma_2(<_{\mathbf{v}})$. We distinguish two cases. First, we treat the base case of the induction and suppose that v is maximal regarding reachability. In the second case we suppose that v is not maximal and conclude by induction.

v is maximal From Lemma 7.15 we get that v is constant, which makes $M_{v,h}$ either universal or empty. Both are easily defined in $\Sigma_2(<_{\mathbf{v}})$.

v is not maximal We conclude as in the previous case by proving that $M_{v,h}$ is a union of specific languages that we prove to be definable in $\Sigma_2(<_{\mathbf{v}})$. Before stating these languages, we give extra definitions. Let t_1, t_2 be two trees, we write:

$$t_1 \equiv t_2 \text{ when } \forall u \notin stab_1(v)\ vu\alpha(t_1) = vu\alpha(t_2')$$

Lemma 7.16. *$\equiv$ is an equivalence relation and its classes are definable in Σ_2.*

Proof. Take L a class of $\equiv$, we consider two cases:

- If L contains $h_m \in H_m$ then we have:

$$L = \bigcap_{u \notin stab_1(v)} M_{vu,h_m}$$

  For all $u \notin stab_1(v)$, v is not reachable from vu. Therefore by induction hypothesis on the position of v in the reachability order, M_{vu,h_m} is definable in $\Sigma_2(<_{\mathbf{v}})$. It follows that L is definable in $\Sigma_2(<_{\mathbf{v}})$.

- Otherwise L is definable in $\Sigma_2(<_{\mathbf{v}})$ since it is a union of L_g for $g \notin H_m$ which we know to be definable in $\Sigma_2(<_{\mathbf{v}})$ by induction hypothesis.

□

For $g \in H$, consider s such that $\alpha(s) = g$, we write $\Gamma_g = \{t \mid t \equiv s\}$. By definition of $\equiv$, Γ_g does not depend on the choice of s. By Lemma 7.16, Γ_g is definable in $\Sigma_2(<_{\mathbf{v}})$.

We are now ready to state the main result of this case:

Lemma 7.17. *A tree t is in $M_{v,h}$ iff there exists $u \in stab_V(v)$ such that it belongs to one of the five types of languages below:*

$K_u a$	$a \in A_0$		$vu\alpha(a) = h$
$K_u a \Gamma_{h_1}$	$a \in A_1$	$\alpha(a) \notin stab_1(v)$	$vu\alpha(a)h_1 = h$
$K_u a(L_g, \Gamma_{h_1}, ..., \Gamma_{h_{i-1}})$	$a \in A_i$	$g \notin (stab_0(v) \cup H_m)$	$vu\alpha(a)(g, h_1, ..., h_{i-1}) = h$
$K_u a(\Gamma_{h_1}, ..., \Gamma_{h_i})$	$a \in A_i$	$\alpha(a) \notin stab_i(v)$	$vu\alpha(a)(h_1, ..., h_i) = h$
$K_u a(\Gamma_{h_1}, ..., \Gamma_{h_i})$	$a \in A_i$	$h_1, h_2 \in H_m$	$vu\alpha(a)(h_1, ..., h_i) = h$

Proof. We first show that if t belongs to one of those languages, we have $t \in M_{v,h}$. We prove that $v\alpha(t) = h$. This is clear for the first type of language. For the second type of language, there exists $s \in \Gamma_{h_1}$ such that $v\alpha(t) = vu\alpha(a)\alpha(s)$. Since $\alpha(a) \notin stab_1(v)$, by Lemma 7.14 we get that $u\alpha(a) \notin stab_1(v)$. By definition of Γ_{h_1} we get $v\alpha(t) = vu\alpha(a)h_1 = h$.

The other types are treated by repeating the previous argument. For the third type we know by Lemma 7.14 that $(\alpha(a) \diamond g) \notin stab_{i-1}(v)$, therefore by definition of $\Gamma_{h_1}, ..., \Gamma_{h_{i-1}}$ we have $v\alpha(t) = vu\alpha(a)(g, h_1, ..., h_{i-1}) = h$. The fourth case is similar since by Lemma 7.14 we have $u\alpha(a) \notin stab_i(v)$.

For the fifth type, there exists $g_1, ..., g_i \in \Gamma_{h_1}, ..., \Gamma_{h_i}$ such that $v\alpha(t) = vu\alpha(a)(g_1, ..., g_i)$. Since $g_1, g_2 \in H_m$, it follows from 7.15 that $u(\alpha(a) \diamond g_1) \notin stab_{i-1}(v)$ and $u(\alpha(a) \diamond g_2) \notin stab_{i-1}(v)$. By definition of $\Gamma_{h_1}, ..., \Gamma_{h_i}$ we get $v\alpha(t) = vu\alpha(a)(h_1, ..., h_i) = h$.

Assume now that $t \in M_{v,h}$, we have $v\alpha(t) = h$. There exists a node x in t such that the context p formed by putting a port at x is such that $\alpha(p) \in stab_1(v)$, and this property does not hold for any descendant of x. We distinguish several cases and show that each one corresponds to a type of language defined in the lemma.

- If x is labeled by $a \in A_0$, t belongs to a language of the first type.
- If x is labeled by $a \in A_1$, by definition of x and Lemma 7.14, we have $\alpha(a) \notin stab_1(v)$, t belongs to a language of the second type.
- If x is labeled by $a \in A_i$ and has a child with type $g \notin stab_0(v) \cup H_m$, then by definition t belongs to a language of the third type.
- If x is labeled by $a \in A_i$ such that $\alpha(a) \notin stab_i(v)$ then by definition t belongs to a language of the fourth type.
- Otherwise x is labeled with some $a \in A_i$ such that $\alpha(a) \in stab_i(v)$. We show that x has two children that are in H_m. Let y by a child of x, by hypothesis we know that the context formed by putting the port at y is not in $stab_1(v)$. Because $\alpha(a) \in stab_i(v)$, it follows that at least one sibling of y, z, is such

that the subtree s at z is not in $stab_0(v)$. Since we are not in Case (3) by hypothesis, we get that $\alpha(s) \in H_m$. By repeating the previous argument for the z, we get a sibling of z, z' such that the tree s' at z' is such that $\alpha(s') \in H_m$. t belongs to a language of the fifth type.

□

We now prove that each type of language defined in Lemma 7.17 is definable using a $\Sigma_2(<_{\mathbf{v}})$ formula, which concludes this case. Given Lemma 7.16, we only need to show that for $u \in stab_1(v)$, K_u is definable in $\Sigma_2(<_{\mathbf{v}})$.

Lemma 7.18. *For $u \in stab_1(v)$, K_u is definable in $\Sigma_2(<_{\mathbf{v}})$.*

Proof. This is done using similar arguments as for Lemma 7.12. We use Proposition 7.2.1. We work with the same infinite alphabet B, B contains two types of letters which are all contexts of over A:

- $a \in A_1$.
- $a \diamond (t_1, ..., t_{i-1})$ for $i \geq 2$ and $a \in A_i$.

Consider the morphism $\beta : B \to V$ which is alpha restricted to contexts of B. By construction we have $K_u = \beta^{-1}(u)$. Now observe that together with the piece relation $(V, \preceq)$ is a stratified monoid, therefore Proposition 7.2.1 applies we get that K_u is of the form:

$$(B \cap \beta^{-1}(N_0))^*(B \cap \beta^{-1}(n_1))...(B \cap \beta^{-1}(n_k))^*(B \cap \beta^{-1}(N_k))^*$$

where $n_1, ..., n_k$ are elements of V, and $N_1, ..., N_k$ are subsets of V that are closed under the piece relation.

We just need to show that the expressions used are definable in $\Sigma_2(<_{\mathbf{v}})$. For expressions of the form $(B \cap \beta^{-1}(N))^*$ where N is closed under pieces, it follows that $(B \cap \beta^{-1}(N))^*$ is closed under pieces and is therefore definable in $\Sigma_2(<_{\mathbf{v}})$. Languages of the form $(B \cap \beta^{-1}(n_1))$ are just union of singletons $\{b\}$ for $b \in B$. We show that we can define such singletons in $\Sigma_2(<_{\mathbf{v}})$, it is clear $a \in A_0$ now consider $b = a \diamond (t_1, ..., t_{i-1})$ for $a \in A_i$, we show that we can define the language $\{a \diamond (t'_1, ..., t'_{i-1}) \mid \alpha(t_j) = \alpha(t'_j),\ 1 \leq j \leq i-1\}$ which concludes the proof. This is because $u \in stab_1(v)$ and therefore it follows from Lemma 7.15 that for all $1 \leq j \leq i-1$, $\alpha(t_i) \in stab_0(h) \subseteq G$ and therefore $alpha^{-1}(\alpha(t_j))$ is definable in $\Sigma_2(<_{\mathbf{v}})$. □

Chapter 8

Unary Temporal Logic over Trees of Bounded Rank

In this chapter, we study the temporal logic $EF + F^{-1}$ in the setting of trees of bounded rank. Our main goal in to obtain a decidable characterization for the class of trees of bounded rank definable in $EF + F^{-1}$.

We briefly recall the definition of $EF + F^{-1}$, more details can be found in Chapter 6. $EF + F^{-1}$ is a temporal logic using a modality EF used to navigate to some descendant node and a modality F^{-1} used to navigate to some ancestor node.

Recall that we presented decidable characterizations for $EF + F^{-1}$ in the settings of words and forests. In Chapter 3 we presented a characterization for $F + F^{-1}$ that uses monoids:

Theorem 3.2. ([TW98]) *A regular word language L over an alphabet A is definable in $F + F^{-1}$ iff its syntactic monoid M verifies, for all $u, v \in M$:*

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \tag{2.2}$$

In Chapter 6 we quoted a decidable characterization for $EF + F^{-1}$ on forests. It uses forest algebras and a specific relation $\dashv$ that tells that considering a context, a smaller context can be built by deleting the subtrees hanging on the path leading from the root to the port.

Theorem 6.3. ([Boj07b]) *Fix L a regular forest language. L is definable in $EF + F^{-1}$ iff its syntactic forest algebra using monoids verifies:*

$$h + g = g + h \quad \forall h, g \in H \tag{6.1}$$

$$h + h = h \quad \forall h \in H \tag{6.2}$$

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \quad \forall u, v \in V \tag{2.2}$$

$$\begin{gathered}\forall u_1, u_2, v_1, v_2 \in V \quad \textit{such that } u_1 \dashv u_2 \textit{ and } v_1 \dashv v_2 \\ (u_1 v_1)^\omega (u_2 v_2)^\omega = (u_1 v_1)^\omega u_1 v_2 (u_2 v_2)^\omega\end{gathered} \tag{6.3}$$

Theorem 6.3 does not extend to trees of bounded rank in a straightforward manner. In particular, it is impossible to express using $EF + F^{-1}$ that a tree is of rank k for some fixed integer k. A simple way to see this is that forest languages definable in $EF + F^{-1}$ are closed under bisimulation.

In this chapter we provide a decidable characterization for the class of tree languages of bounded rank that are definable in $EF + F^{-1}$. Our characterization uses k-algebras which we defined in Chapter 5. We will also have to define a new relation $\dashv$ that is adapted to trees of bounded rank. The relation used in the forest setting makes no sense in the bounded rank setting since it affects the number of children of the nodes. Notice that Equation (6.2) makes no sense over trees of bounded rank. Therefore we will need to replace it with another equation. Finally, our characterization will contain no equivalent to Equation (6.1). This is because k-algebras already assume that the languages are closed under commutation of sibling subtrees.

In Section 8.1 we present our decidable characterization and discuss its relation with the characterization of [Boj07b] in the forest setting. In Section 8.2 we prove the "only if" part of the characterization. Finally, in Section 8.3 we provide a proof for the difficult "if" direction of this characterization. Throughout this chapter we compare both the statements and the proofs we use to the ones used in [Boj07b] for the unranked setting.

8.1 Characterization of $EF + F^{-1}$

In order to give our characterization we need to define a relation over k-algebras. In [Boj07b], a relation, $\dashv$, over contexts is used in order to state the characterization. Considering a context, a smaller context can be built by deleting the subtrees hanging on the path leading from the root to the port. In our setting this obviously does not preserve the arity of nodes, therefore we cannot just delete subtrees as it would not preserve the tree structure. We define a variant of this relation over trees of bounded rank, rather than deleting subtrees, we ask that all the subtrees hanging on the path from the root to the port of the smaller context appear on the same path in the bigger context. Given $u, v \in V$, we write $u \dashv v$ iff we have:

- $u = (u_0 \diamond (h, ..., h))...(u_n \diamond (h, ..., h))$

- $v = (u_0 \diamond (g_0^0, ..., g_{\delta_0}^0))...(u_n \diamond (g_0^n, ..., g_{\delta_n}^n))$
- $\forall i\ u_i \in W_{\delta_i+1}$
- $h \in \{g_0^0, ..., g_{\delta_0}^0, ..., g_0^n, ..., g_{\delta_n}^n\}$

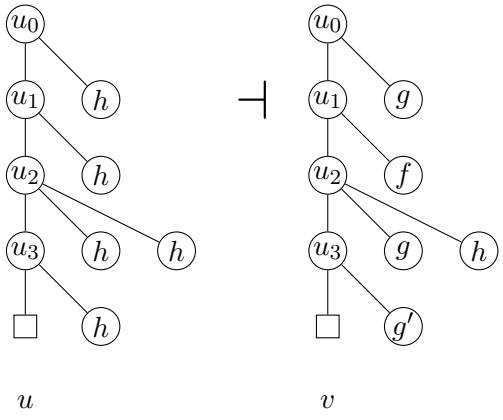


Figure 8.1: Illustration of the $\dashv$ Relation.

Since we use this relation in the characterization, for the characterization to be decidable, we need $\dashv$ to be computable given a finite k-algebra. This is a consequence of the following lemma:

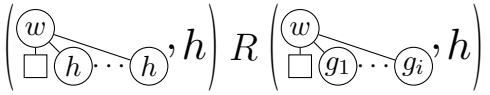
Lemma 8.1. *The relation $\dashv$ is computable.*

Proof. To compute $\dashv$, we define a relation $\preccurlyeq \subseteq (V \times H) \times (V \times H)$. We prove that $\preccurlyeq$ can be computed and characterize $\dashv$ as the restriction of $\preccurlyeq$ on V which ends the proof. Given $u, v \in V$ and $h \in H$, we write $(u, h) \preccurlyeq (v, g)$ iff we have:

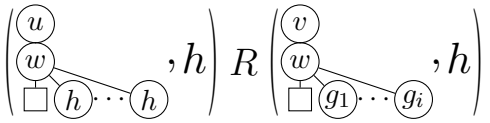
- $h = g$
- $u = (u_0 \diamond (h, ..., h))...(u_n \diamond (h, ..., h))$
- $v = (u_0 \diamond (g_1^0, ..., g_{\delta_0}^0))...(u_n \diamond (g_1^n, ..., g_{\delta_n}^n))$
- $\forall i\ u_i \in W_{\delta_i-1}$
- $h \in \{g_1^0, ..., g_{\delta_0}^0, ..., g_1^n, ..., g_{\delta_n}^n\}$

By definition $\dashv$ is the restriction of $\preccurlyeq$ to V. We show that $\preccurlyeq$ is computable. The relation $\preccurlyeq$ is the least R such that:

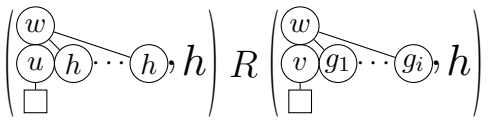
- $\forall i \leq k, \forall w \in W_{i+1}, \forall g_1, ...g_i \in H$ and $\forall h \in \{g_1, ..., g_i\}$:



- $\forall i \leq k, \forall w \in W_{i+1}, \forall g_1, ...g_i \in H$, if $(u, h)R(v, h)$ then:



- $\forall i \leq k, \forall w \in W_{i+1}, \forall g_1, ...g_i \in H$, if $(u, h)R(v, h)$ then:



Therefore, we can then compute $\preccurlyeq$ using a least fixpoint algorithm. □

With the relation $\dashv$ defined, we are now ready to state our characterization of $EF + F^{-1}$:

Theorem 8.2. *Fix k some integer and L a regular tree language of bounded rank k. L is definable in $EF + F^{-1}$ iff its syntactic k-algebra verifies the equations:*

$$\begin{gathered}\forall w \in W_3 \ \forall h, g \in H \\ w \cdot (h, h, g) = w \cdot (h, g, g)\end{gathered} \tag{8.1}$$

$$\begin{gathered}\forall w \in W_2 \ \forall h, g \in H \\ (w \diamond h)^\omega g = (w \diamond h)^\omega w((w \diamond h)^\omega g, (w \diamond h)^\omega g)\end{gathered} \tag{8.2}$$

$$\begin{gathered}\forall w, w' \in W_2 \ \forall h, h' \in H \quad \textit{and for } e = ((w \diamond h)(w' \diamond h'))^\omega \\ e = e(w \diamond h')(w' \diamond h)e\end{gathered} \tag{8.3}$$

$$\begin{gathered}\forall u_1, u_2, v_1, v_2 \in V \quad \textit{such that } u_1 \dashv u_2 \textit{ and } v_1 \dashv v_2 \\ (u_1 v_1)^\omega (u_2 v_2)^\omega = (u_1 v_1)^\omega v_2 (u_2 v_2)^\omega\end{gathered} \tag{8.4}$$

The characterization proposed in [Boj07b] for unranked tree languages shares some similarities with our definition. This characterization can be seen as divided in two parts, an horizontal one and a vertical one.

The horizontal part states the closure under bisimulation. This closure does not make sense in our case since it affects the rank of the nodes of the trees. The essence of this closure is replaced with (8.1) and (8.2). Notice that there is no equivalent to Equation (6.1). This is because k-algebras work on unordered trees.

Vertically, the characterization of [Boj07b] used two identities. The first one is Equation (2.2) which characterizes $EF + F^{-1}$ over words (see Theorem 3.2). The second one uses the $\dashv$ relation. In our characterization both those equations are replaced by (8.4). Recall however, that we had to redefine the relation $\dashv$ used in (8.4), this leads to different uses of this equation. We also need to add a new identity, equation (8.3). In particular, notice that Equation (8.4) implies Equation (2.2).

Before proving Theorem 8.2, we show that the characterization is decidable. Indeed, the syntactic k-algebra of a regular language is computable, finite and the relation $\dashv$ is computable (see Lemma 8.1). Therefore, in order to decide definability in $EF + F^{-1}$, it is sufficient to compute the syntactic k-algebra and to check that it verifies the equations.

Corollary 8.3. *It is decidable whether a language of trees of bounded rank is definable in $EF + F^{-1}$.*

The rest of this chapter is devoted to the proof of Theorem 8.2. In Section 8.2 we prove that the equations are necessary. Then, in Section 8.3, we prove that they are sufficient.

8.2 Necessity of the Equations

We use an Ehrenfeucht-Fraïssé game argument. We begin with the definition of Ehrenfeucht-Fraïssé game for $EF + F^{-1}$. A k rounds Ehrenfeucht-Fraïssé game is played on two trees t and t', there are two players called Spoiler and Duplicator and they both possess one pebble. At the start of the game each player has his pebble on the root of one of the two trees. A round is played as follows, assume there is a pebble at position x_t on t and at position $x_{t'}$ on t':

- Spoiler chooses a tree, say t, and moves the pebble at x_t on a node y_t that is either a proper descendant or a proper ancestor of x_t.
- Duplicator has to answer by moving the pebble at $x_{t'}$ on a node $y_{t'}$ of t' with the same label as y_t and which is a proper descendant of $x_{t'}$ if y_t was a proper descendant of x_t and a proper ancestor of $x_{t'}$ if y_t was a proper ancestor of x_t. If Duplicator cannot play, Spoiler wins.
- If Duplicator did not loose, the game continues to the next round with $y_t, y_{t'}$ playing the role of $x_t, x_{t'}$.

If the game lasts k rounds Duplicator wins. If Duplicator has a winning strategy for the Ehrenfeucht-Fraïssé game on t and t', we write $t \cong_k t'$. The

Ehrenfeucht-Fraïssé is closely linked to the notion of definability in $EF + F^{-1}$ as stated in the following Lemma. We call rank of an $EF + F^{-1}$ formula the nesting depth of modalities.

Lemma 8.4. *$t \cong_k t'$ iff t and t' satisfy the same $EF + F^{-1}$ formulas of rank k.*

Lemma 8.4 is proved using classical Ehrenfeucht-Fraïssé techniques. We use it in order to prove that our equations are necessary. Suppose that L is definable in $EF + F^{-1}$ and let l be the rank of the corresponding $EF + F^{-1}$ formula φ.

The correctness of all equations is proved by extending the strategies we described for proving the correctness of Equation (2.2) in Chapter 3 (see Proposition 3.4). In this section we only provide the strategy for proving the correctness of (8.4). We make this choice because this strategy is the most technical one since it uses the relation $\dashv$. Let $(H, V, W_2, ..., W_k)$ be the syntactic k-algebra of L, we show that it verifies (8.4). In order to make the notations simpler we suppose that $k = 2$. The case $k > 2$ is treated using the similar arguments.

Let u_1, u_2, v_1, v_2 as in the statement of (8.4). We exhibit two contexts p and q such that $\alpha(p) = (u_1v_1)^\omega(u_2v_2)^\omega$, $\alpha(q) = (u_1v_1)^\omega v_2(u_2v_2)^\omega$ and for all trees t and all contexts r, $rpt \cong_k rqt$. It follows from Lemma 8.4 that for all trees t and all contexts r, $rpt \in L$ iff $rqt \in L$, and by definition of the syntactic k-algebra that $(u_1v_1)^\omega(u_2v_2)^\omega = (u_1v_1)^\omega v_2(u_2v_2)^\omega$.

We have $u_1 \dashv u_2$ and $v_1 \dashv v_2$. By definition of $\dashv$ it follows that there exists $w_1^u, ..., w_n^u \in W_2$, $w_1^v, ..., w_m^v \in W_2$ and $h, h_1, ..., h_n, g, g_1, ..., g_m \in H$ such that:

- $h \in \{h_1, ..., h_n\}$
- $g \in \{g_1, ..., g_m\}$
- $u_1 = (w_1^u \diamond h)...(w_n^u \diamond h)$
- $v_1 = (w_1^v \diamond g)...(w_m^v \diamond g)$
- $u_2 = (w_1^u \diamond h_1)...(w_n^u \diamond h_n)$
- $v_2 = (w_1^v \diamond g_1)...(w_m^v \diamond g_m)$

To simplify the notations further without loss of generality we suppose that the types $w_1^u, ..., w_n^u, w_1^v, ..., w_m^v$, h, $h_1, ..., h_n$, g, $g_1, ..., g_m$ are all reached by single node labels. We get labels $c_1^u, ..., c_n^u, c_1^v, ..., c_m^v \in A_2$ and $a, a_1, ..., a_n, b, b_1, ..., b_m \in A_1$ such that $\forall i, j$ $\alpha(c_i^j) = w_i^j$, $\alpha(a) = h$, $\alpha(a_i) = h_i$, $\alpha(b) = g$ and $\alpha(b_i) = g_i$. We define:

- $d_1 = (c_1^u \diamond a)...(c_n^u \diamond a)$

- $e_1 = (c_1^v \diamond b)...(c_m^v \diamond b)$
- $d_2 = (c_1^u \diamond a_1)...(c_n^u \diamond a_m)$
- $e_2 = (c_1^v \diamond b_1)...(c_m^v \diamond b_m)$
- $p = (d_1 e_1)^{l\omega}(d_2 e_2)^{l\omega}$
- $q = (d_1 e_1)^{l\omega} e_2 (d_2 e_2)^{l\omega}$

By construction we have $\alpha(p) = (u_1 v_1)^\omega (u_2 v_2)^\omega$ and $\alpha(q) = (u_1 v_1)^\omega v_2 (u_2 v_2)^\omega$. Now consider t some tree and r some context we show that Duplicator has a winning strategy for the l rounds Ehrenfeucht-Fraïssé game on $s_1 = rpt$ and $s_2 = rqt$. In order to describe Duplicator's strategy we give a few definitions.

We call the *skeleton* of s_1 and s_2 the sequence of nodes that goes from the root of r to the root of t. Given a node x in rpt or rqt we call its *key ancestor* the first ancestor of x that is on the skeleton. We call the *downward level* of x the sum of the number of copies of the context e_2 and the numbers of copies of the context e_1 that are below its key ancestor and its *upward level* the sum of the number e_1 and the number of copies of e_2 that are above its key ancestor. Given a node x in p and a node y in q we say that x and y are *locally similar* iff they are both at the same position in a context d_1, d_2, e_1 or e_2 or one is in a context e_1 (resp. d_1) and the other in a context e_2 (resp. d_2) and they are at the same position on the skeleton.

We are now ready to give Duplicator's strategy. We show that she can play maintaining the following invariant. We call x and y the positions of the pebbles in s_1 and s_2. We say $\mathcal{P}(l')$ is verified iff the play is in one of the following situations:

1. x, y are in the context r of their respective trees and are at the same position.
2. x, y are in the subtree t of their respective trees and are at the same position.
3. x, y are in p, q. They are locally similar is they are on inner nodes. Either the upward levels of x, y are equal and their downward levels are both greater than l' or the upward levels of x, y are both greater than l' and their downward levels are equal.

We now show that Duplicator can play while maintaining $\mathcal{P}(l')$ with l' the number of turns left to play. At the start of the game $\mathcal{P}(l)$ is verified for the first item by definition. Now suppose that there are l' turns left to play and that $\mathcal{P}(l')$ is verified. We explain how Duplicator can answer to Spoiler's move in order to make $\mathcal{P}(l')$ true at the next turn. Note that Duplicator always answer in a locally similar way.

If the first or the second item of $\mathcal{P}(l')$ holds, as long as Spoiler does not move on the extra copy of e_2 in q, Duplicator answers to any move by copying Spoiler's move and $\mathcal{P}(l'-1)$ is verified. If Spoiler plays on the extra copy of e_2 in q then Duplicator answers on the topmost copy of e_2 in p, the upward levels of x and y are now equal and their downward levels are both greater than $l-1 \geq l'-1$, therefore $\mathcal{P}(l'-1)$ is verified for the third item.

Now suppose that the third item of $\mathcal{P}(l')$ holds. If Spoiler moves in r or t Duplicator's answer is clear. We suppose that Spoiler moves on some node $x' \in s_1$ (the case of a move in s_2 is similar). We consider two cases depending on the relationship between x and x':

- x' is an ancestor of x. If x, y had the same upward level then Duplicator's answer is straightforward as he can find a locally similar y' with the same upward level as x. If x, y had upward level bigger then l' then either x' has an upward level smaller than $l'-1$ and Duplicator can find an y' with the same upward level or Duplicator can find an y' with upward level bigger than $l'-1$.

- x' is a descendant of x and is an inner node. This case is symmetrical to the previous one.

- x' is a leaf descendant of x. This leaf is in a context d_1, d_2, e_1 or e_2, if there exists a copy of the same context below y then this case is treated as the previous one. We describe how we treat the case when the leaf is in a context d_1 (or e_1) and there is no copy of d_1 (or e_1) below y. In this case, we know by construction that there exists a leaf with the same label in the contexts d_2, Duplicator just plays on that node in the topmost copy of d_2 below y.

8.3 Sufficiency of the Equations

We turn to the hardest direction of Theorem 8.2, if the syntactic k-algebra of L verifies the equations, then L is definable in $EF + F^{-1}$. Consider L a language whose syntactic k-algebra verifies (8.1), (8.2), (8.3) and (8.4), we call α the associated morphism. We show that L is definable in $EF + F^{-1}$. We begin with a few definitions.

Antichain Composition Principle. The proof makes use of the following composition lemma, taken from [Boj07b]. We reuse notations from [Boj07b]. It extends the Concatenation Principle we stated for words in Lemma 3.6 of Chapter 3.

A formula of $EF + F^{-1}$ called *antichain* if in every tree, the set of nodes where it holds forms an antichain, i.e. a set (not necessarily maximal) of nodes pairwise

incomparable with respect to the descendant relation. This is a semantic property, and may not be apparent just by looking at the syntax of the formula.

We fix (i) an antichain formula φ, (ii) disjoint tree languages $L_1, \cdots, L_n$ and (iii) leaves of label $a_1, \cdots, a_n$. Given a tree s. We define the tree $s[(L_1, \varphi) \to a_1, \cdots, (L_n, \varphi) \to a_n]$ as follows. For each node x of s where the antichain formula φ holds, we determine the unique i such the tree language L_i contains the subtree of x. If such an i exists, we remove the subtree of x (including x), and replace x by a leaf labeled with a_i. Since φ is antichain, this can be done simultaneously for all x. Note that the formula φ may depend also on ancestors of x, while the languages L_i only talk about the subtree of x.

Lemma 8.5. *[Antichain Composition Lemma][Boj07b] Let φ, $L_1, \cdots, L_n$ and $a_1, \cdots, a_n$ be as above. If $L_1, \cdots, L_n$ and K are languages definable in $EF + F^{-1}$, then so is $\{t \mid t[(L_1, \varphi) \to a_1, \cdots, (L_n, \varphi) \to a_n] \in K\}$.*

X-trimmed trees In the following X will always be a subset of H. The following notion is adapted from [Boj07b]. A tree s is said to be *X-trimmed* if the only nodes of s that are of type in X are leaves. We say that a tree language L is *definable modulo X* if there is a definable tree language L' that agrees with L over X-trimmed trees.

P-reachability We reuse the notion of reachability we defined for the $\Delta_2(<_{\mathbf{v}})$ characterization and for words in Chapter 3. Since most of the properties we proved for this pre-order depended only on (2.2), it follows that this remains a natural pre-order to use for $EF + F^{-1}$. However we need to modify its definition in this new setting. We parametrize the reachability relation with a subset P of forbidden labels of A, intuitively a type will be P-reachable from an other type if it is reachable using a context that does not use the forbidden labels of P.

We call a twig a tree which has exactly one inner node which is its root. Given $P \subseteq A$, we say that a tree t is P-valid, iff the only nodes of P that are labeled with elements of P are in twigs. We extend this definition to contexts and k-contexts by adding the condition that if the port is in a twig, this twig must not contain labels in P. The notion of P-validity is extended to types of $(H, V, W_2, ..., W_k)$, $h \in H$ is P-valid iff there exists a P-valid tree t such that $\alpha(t) = h$, $w \in W_i$ is P-valid iff there exists some P-valid k-context p such that $\alpha(p) = w$.

Given this definition, we define the notion of *P-reachability* for types of H and V. Given $h, g \in H$ we say that h is *P-reachable* from g if there exists a P-valid $x \in V$ such that $h = xg$. Similarly, given $u, v \in V$ we say u is *P-reachable* from v iff there exists a P-valid $x \in V$ such that $u = vx$. A first observation regarding P-reachability is that as long as P does not forbid all labels of arity greater or equal to 2 there exists a maximal class of P-reachable types in H.

Lemma 8.6. *If there exists $i \geq 2$ and $a \in A_i$ such that $a \notin P$ then there exists a unique subset $H_m^P \subseteq H$ such that every type in H_m^P is reachable from all P-valid types.*

Proof. This is because given $h, g \in H$ that are P-valid we can build a type g' that is P-reachable from both h and g using a, $g' = \alpha(a) \cdot (h, g, g, ...)$. □

We have now all the definitions we need in order to start the proof. For all $h \in H$, all $v \in V$ and all $P \subseteq A$ consider the following language:

$$L_{v,h}^P = \{t : v \cdot \alpha(t) = h \text{ and } t \text{ is } P\text{-valid}\}$$

We show that:

Proposition 8.7. *$\forall h \in H, v \in V, P \subseteq A$ and $X \subseteq H$, $L_{v,h}^P$ is definable in $EF + F^{-1}$ modulo X.*

This Proposition concludes the proof of Theorem 8.2 as L is a union of $L_{v,h}^P$ for $h \in \alpha(L)$, P empty and $v = 1_V$. Hence by applying Proposition 8.7 with $X = \emptyset$, each of these $L_{v,h}^P$ are definable in $EF + F^{-1}$, and therefore L is definable in $EF + F^{-1}$.

Before going on with the proof we make a simple observation that we will use several times in the proof of Proposition 8.7:

Lemma 8.8. *For all $h \in H$, the language $T_h = \{t : \alpha(t) = h$ and t is a twig$\}$ is definable in $EF + F^{-1}$.*

Proof. This is a simple consequence of (8.1). It follows from (8.1) that if two twigs have the same root label and the same set of leaves they have the same type. It is then simple to describe the root and the set of leaves of a twig in $EF + F^{-1}$. □

We proceed by induction on the following parameters listed by order of importance:

1. The size of X
2. The number of inner node labels in P
3. The number of leaf labels in P
4. The number of context types P-reachable from v

We consider three cases. The first case is basically the case of words, we suppose that P forbids all letters of arity greater than 2 and conclude using known results over word languages definable in $EF + F^{-1}$. In the second and third cases we

suppose that there exists at least a letter of arity greater than 2 that is not forbidden by P, therefore by Lemma 8.6 there exists a class of H_m^P of types that are P-reachable from all P-valid types. Our second case assumes that there exists a type which is neither in X nor in H_m^P. In this case we conclude by induction on the size of X and the size of P. In the remaining case, when $H_m^P \cup X = H$ we conclude by induction on the size of P and the number of contexts P-reachable from v or by showing that $L_{v,h}^P$ is either empty of universal.

Case 1: P forbids all letters of arity greater than 2

In this case, all P-valid trees are of the form $p \cdot t$ where t is a twig and $w \in A_1^*$.

For every $u \in V$, we consider the word language $L_u = \{p \in (A_1 - P)^* \mid \alpha(p) = u\}$. It follows from Equation (8.4) that V satisfies Equation (2.2), therefore, using Theorem 3.2 and Theorem 2.4, we get that for all $u \in V$ there exists an $F + F^{-1}$ formula φ_u that defines L_u.

Now notice that:

$$L_{v,h}^P = \cup_{\{t \text{ twig}, u \in V : vu\alpha(t) = h\}} L_u t$$

Because of Lemma 8.8 all of these languages is definable in $EF + F^{-1}$ and the result follows.

Case 2: $H_m^P \cup X \neq H$

Fix G as a class of mutually P-reachable P-valid types such that $G \neq H_m$, $G \not\subseteq X$, and if G is P-reachable from a P-valid type h, $h \in G$ or $h \in X$. The existence of such a class G is ensured by our hypothesis. Intuitively, G is just above X regarding P-reachability and is different form H_m^P. We first show that membership in G can be detected in $EF + F^{-1}$ modulo X.

Lemma 8.9. *There is a formula $\varphi \in EF + F^{-1}$ such that for any X-trimmed tree t the set of nodes x such that the subtree at x has type in G is exactly the set of nodes at which φ holds.*

Proof. This is proved using properties of the reachability relation similar to the ones we used in Lemma 7.8. Most of them depended only Equation (2.2). Therefore many arguments are similar. However some properties depend on equations (8.2) and (8.3) that are specific to this case.

We show that a X-trimmed subtree has type in G iff it does not contain certain labels. Fix some $g \in G$, for $0 \leq i \leq k$, we define the set of labels B_i as follows:

$$\begin{aligned} B_0 &= \{a_0 \in A_0 : \forall w \in W_2 \ (w \diamond \alpha(a_0))g \notin G\} \\ B_1 &= \{a_1 \in A_1 : \ \alpha(a_1)g \notin G\} \\ B_i &= \{a_i \in A_i : \forall h_1, ..., h_{i-1} \ (\alpha(a_i) \diamond (h_1, ..., h_{i-1}))g \notin G\} \text{ for } i \geq 2 \end{aligned}$$

We write $P' = B_0 \cup ... \cup B_k$, the result is a consequence of the following Lemma.

Lemma 8.10. *Given a P-valid tree, t a subtree s of t has type outside G iff one the following properties holds:*

1. *s is a leaf of type outside G.*

2. *s has a twig subtree which has type outside G.*

3. *s is not P'-valid.*

Before proving this Lemma we conclude the proof of Lemma 8.9. We only need to show that we can express the conditions stated in Lemma 8.10 in $EF + F^{-1}$. The first condition is simply expressed, it follows from Lemma 8.8 that the second one can be expressed. We just need to show that we can test P'-validity. To test P-validity we need to test that all nodes that are not in twigs are not labeled with labels of P'. This is simple to do if we can test test if a node is in a twig. This is simple for inner nodes, we test if all the descendants of the node are leaves, for leaves the formula tests that the node has an ancestor which is in a twig. We turn to the proof of Lemma 8.10.

Proof. Before getting to the proof we make two observations that we will use several times. First if for some P-valid $v \in V$ and $g', g'' \in G$, if $vg' \in G$, then $vg'' \in G$. This property is a consequence of Equation (8.4) and is proved using the techniques we used in the study of $\Delta_2(<_{\mathbf{v}})$. We know there exists $x, y, z \in V$ P-valid and such that $g'' = xvg'$, $g' = yg''$ and $g = zg''$, therefore we have $g'' = xvyg'' = (xvy)^\omega g'' = (xvy)^\omega vg''$ using (2.2). It follows that $g = z(xvy)^\omega vg''$, since $z(xvy)^\omega v$ is P-valid, g is P-reachable from vg''.

The second observation is that as soon as there exists a subtree s' of s of type outside G that is not a leaf, s has type outside G. Indeed suppose there is such a subtree s' and let h' be its type, since s' is not a leaf and s is X-trimmed, $h' \notin X$ and by definition $\alpha(s)$ is P-reachable from $h' \notin G$, by choice of G it follows that s has type outside G.

The conditions are sufficient The first condition is obviously sufficient and the second one follows from our second observation. We turn to the third condition. First suppose that there exists an inner node in s labeled with $a \in P$, let s' be the subtree at this node, because of our second observation we can suppose that all subtrees of s' that are not leaves are of type in G. This means that s' is of the form $(a \diamond (t_1, ...t_{i-1}))s'')$ with $a \in P \cap A_i$ for some i and $\alpha(s'') \in G$. By definition of a, $\alpha(a \diamond (t_1, ...t_{i-1}))g \notin G$, but then using our first observation we have $\alpha(s') = \alpha((a \diamond (t_1, ...t_{i-1}))s'') \notin G$. It follows from our second observation that $\alpha(s) \notin G$.

Now suppose that there exists a leaf in s that is not in a twig and that is labeled with $a \in P'$, let s' be the subtree at the father of this node, like before we can suppose that all subtrees of s' that are not leaves have type in G. Therefore by hypothesis s' is of the form $(q \diamond (a_o))s''$ with q a 2-context $a_0 \in P'$ and s'' of type in G. Take $g' = \alpha(s'')$ and $w = \alpha(q) \in W_2$. By definition of a_0, we have $(w \diamond g')g \notin G$, like in the previous case it follows from our first observation that $\alpha(s') \notin G$, and our second allows us to conclude that $\alpha(s) \notin G$.

The conditions are necessary Suppose that s has type outside G, we show that is satisfies one of our four conditions. If s is a leaf the first condition is verified. Otherwise, let s' be a subtree of s such that $\alpha(s') \notin G$, s' is not a leaf and all the subtrees of s' that are not leaves have type in G. If s' is a twig the second condition is verified. Suppose that s' is not a twig, if the root of s' is labeled by $a \in P'$ then the third property is verified. Otherwise, if the root of s' is labeled with $a \notin P'$ we show that the root of s' has a child that is a leaf and is labeled with $a \in P'$. We do this in two steps, first we show using Equation (8.2) that the root of s' has at least one child that is a leaf then we show using Equation (8.3) that at least one of this leaf child has a label in P'. In order to simplify the notation we suppose that the root of s' is of arity 2, however the proof extends in a straightforward way to arbitrary rank by repeating the arguments.

Suppose that the root of s' has no child that is a leaf, it means that $s' = (a \diamond (t_1)) \cdot t_2$, with $\alpha(t_1), \alpha(t_2) \in G$. We have $x \in V$ P-valid such that $g = x\alpha(t_2)$. Moreover, since $a \notin P'$, the exists $g' \in H$ such that $(\alpha(a) \diamond g')g \in G$, therefore there exists $y \in V$ P-valid such that $y(\alpha(a) \diamond g')x\alpha(t_2) = \alpha(t_2)$. Using a little algebra we get:

$$\begin{aligned}
\alpha(t_2) &= (y(\alpha(a) \diamond g')x)^\omega \alpha(t_2) \\
\alpha(t_2) &= (y(\alpha(a) \diamond g')x)^\omega y(\alpha(a) \diamond \alpha(t_2))x(y(\alpha(a) \diamond g')x)^\omega \alpha(t_2) \text{ using (8.2)} \\
\alpha(t_2) &= (y(\alpha(a) \diamond g')x)^\omega y(\alpha(a) \diamond \alpha(t_2))g
\end{aligned}$$

It follows that $(\alpha(a) \diamond (\alpha(t_2)))g \in G$, by our first observation we get that

$s' = (\alpha(a) \diamond (\alpha(t_2))) \cdot \alpha(t_1) \in G$ which is false. Therefore the root of s' has a child that is a leaf, suppose that this leaf has a label a_0 outside P'. Since s' is not a twig, it is of the form $(a \diamond a_0)s''$ with s'' of type in G and $a_0 \notin P'$. Therefore there exist $w \in W_2$ and $x \in V$ P-valid such that $x(w \diamond \alpha(a_0))g = g$, also since $a \notin P'$ there exists $h' \in H$ and $y \in V$ P-valid such that $y(\alpha(a) \diamond h')g = g$. Combining the two equalities, we get:

$$\begin{aligned} g &= y(\alpha(a) \diamond h')x(w \diamond \alpha(a_0))g \\ g &= (y(\alpha(a) \diamond h')x(w \diamond \alpha(a_0)))^\omega g \\ g &= (y(\alpha(a) \diamond h')x(w \diamond \alpha(a_0)))^\omega y(\alpha(a) \diamond \alpha(a_0))x(w \diamond h')g \text{ using (8.3)} \end{aligned}$$

From our first observation it follows that $\alpha(s') \in G$, which is false by hypothesis. Therefore $a_0 \in P'$ and the third condition is verified. □

This ends the proof of Lemma 8.9. □

We now aim at applying Lemma 10.4, the antichain formula being essentially the one given by Lemma 8.9. The next two lemmas show that the appropriate languages are definable in $EF + F^{-1}$.

Lemma 8.11. *$L^P_{v,h}$ is definable in $EF + F^{-1}$ modulo $X \cup G$.*

Proof. This is by induction on X in Proposition 8.7. □

Lemma 8.12. *For any $g \in G$, $L^P_{1_V,g}$ is definable in $EF + F^{-1}$ modulo X.*

Proof. There are two cases, first if all the P-valid trees of type in G are leaves or twigs $L^P_{1_V,g}$ is definable from Lemma 8.8. Otherwise it follows from Lemma 8.10 and the fact that since $G \neq H^P_m$, P' is not empty. Also by Lemma 8.10, $L^P_{1_V,g} = L^{P' \cup P}_{1_V,g}$. Therefore by induction on the numbers of labels in P, we get the result. □

We are now ready to give the final argument which is depicted in Figure 8.2. Let φ be the formula which holds at a node x of a tree t iff the subtree at x is of type in G and there is no node between the root of t and x with a subtree of type in G. From Lemma 8.9, φ is definable in $EF + F^{-1}$ and by definition it is an antichain formula. Let K be the restriction of $L^P_{v,h}$ to $(X \cup G)$-trimmed trees. By Lemma 8.11, K is definable in $EF + F^{-1}$. Assume $G = \{g_1, \cdots, g_l\}$. For any $i \leq l$, let L_i be the restriction of $L^P_{1_V,g_k}$ to X-trimmed trees and let a_k be a leaf node such that $\alpha(a_k) = g_i$. By Lemma 8.12 each L_i is definable in $EF + F^{-1}$. Hence from the Antichain Composition Lemma, Lemma 10.4, we have that $\{t \mid t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k] \in K\}$ is also definable in $EF + F^{-1}$. This precisely says that $L^P_{v,h}$ is definable in $EF + F^{-1}$ modulo X. This concludes the proof of Proposition 8.7 in this case.

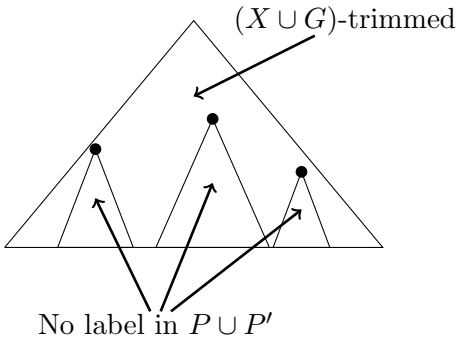


Figure 8.2: Illustration of the Antichain Composition Lemma for Case 2. The marked nodes are the topmost nodes in G.

Case 3: $H_m^P \cup X = H$

We distinguish two cases depending on whether there exists some inner node symbol $a \in A - P'$ of some arity i such that for all P-valid $h_1, ..., h_{i-1} \in H$, v is not P-reachable from $v(a \diamond (h_1, ..., h_{i-1}))$. If there exist such a label we call it P-bad and we proceed by induction, otherwise we conclude using (8.4).

Case 1: There exists a P-bad label Let a be the P-bad label and let i be its arity. Given two elements h and h' of H, we say that h is v^+-equivalent to h' if for all u P-reachable from v such that v is not P-reachable from u (hence the P-depth of u is strictly higher that the P-depth of v) we have $uh = uh'$.

Take two P-valid trees s, s' such that their root is labeled with a we say that they are equivalent iff for each child of the root of s there exists a child of the root of s' that is v^+ equivalent and for each child of the root of s' there exists a child of the root of s that is v^+ equivalent. Let $L_1, \cdots, L_n$ be all the equivalence classes of this relation.

Lemma 8.13. *$L_1, \cdots, L_n$ are definable in $EF + F^{-1}$ modulo X.*

Proof. Notice first that the equivalence classes of the v^+ equivalence restricted to P-valid trees are definable in $EF + F^{-1}$ modulo X. This is immediate by induction on the P-depth of v in Proposition 8.7.

To determine if a tree is in $L_1, \cdots, L_{n-1}$ or L_n we just need to check that its root is a and the set of classes of the v^+ equivalence that appear in the children of the root. Since we can test if a node is a child to the root (its only ancestor is the root) and check the v^+ equivalence class of tree, it follows that the $L_1, \cdots, L_n$ are definable in $EF + F^{-1}$ modulo X □

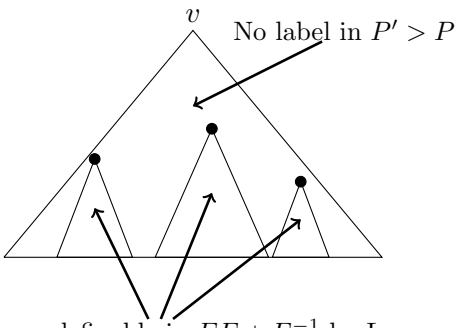


Figure 8.3: Illustration of the Antichain Composition Lemma. The marked nodes are the topmost nodes of label a.

For any j let h_j be an arbitrarily chosen type in L_j and let a_j be a leaf label whose type is h_j.

Let $P' = (P - A_0) \cup \{a\}$ we have:

Lemma 8.14. *The set $L^{P'}_{v,h}$ is definable in $EF + F^{-1}$ modulo X.*

Proof. This is by induction on the number of inner node labels in P in Proposition 8.7. □

Based on the above lemmas, we conclude this case of Proposition 8.7 as follows. Consider the property that holds at a node y of a tree t if the y is labeled with a and has no ancestor labeled with a, this property is easily expressed in $EF + F^{-1}$ by a formula φ. By definition φ is antichain.

Let $K = L^{P'}_{v,h}$, by Lemma 8.14, K is definable in $EF + F^{-1}$ modulo X. Hence we can apply the Antichain Composition Lemma (see Figure 8.3) and have that $\{t \mid t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k] \in K\}$ is definable in $EF + F^{-1}$ modulo X.

We conclude by showing that $L^{P}_{v,h} = \{t \mid t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k] \in K\}$. This is a simple consequence of the following two lemmas.

Lemma 8.15. *For any P-valid X-trimmed tree t, $t[(L_1, \varphi) \to a_1, \cdots, (L_n, \varphi) \to a_n]$ is P'-valid.*

Proof. This follows from the construction of P' and the definition of φ. □

Lemma 8.16. *For any X-trimmed tree t, $v\alpha(t) = v\alpha(t[(L_1, \varphi) \to a_1, \cdots, (L_n, \varphi) \to a_n])$.*

Proof. The proof uses (8.1) and (2.2) through Equation (8.4). We first prove a preliminary result. Take u that is P-reachable from v and $j \in \{0, n\}$, then for

every t, t' in L_j, we have $u\alpha(t) = u\alpha(t')$. In order to make the notations simpler we suppose that a is of arity 3, since all technical difficulties arise for arity 3. One can solve this case with greater arity via a repeated use of (8.1). We have $t = a(t_1, t_2, t_3)$ and $t' = a(t'_1, t'_2, t'_3)$, and since $t, t' \in L_j$, for each t_i there exists a t'_j that is v^+-equivalent and for each t'_j there exists a t_i that is v^+-equivalent. Since our trees are unordered we only have two different cases, in the first one t_i is v^+-equivalent to t'_i for $i = 1, 2, 3$ and in the second one t_1, t'_1, t'_2 are v^+-equivalent and t'_3, t_2, t_3 are v^+-equivalent. We first prove the result in the first case and then reduce the second case to the first case using (8.1).

By definition $u\alpha(a \diamond (t_2, t_3)))$ is P-reachable from v, we show that v is not P-reachable from $u\alpha(a \diamond (t_2, t_3)))$. Indeed suppose that there exists $x \in V$ P-valid such that $v = u\alpha(a \diamond (t_2, t_3)))x$, by definition of u there exists y P-valid such that $u = vy$ therefore:

$$\begin{aligned} v &= vy\alpha(a \diamond (t_2, t_3)))x \\ v &= v(y\alpha(a \diamond (t_2, t_3)))x)^\omega \\ v &= v\alpha(a \diamond (t_2, t_3)))(y\alpha(a \diamond (t_2, t_3)))x)^\omega \text{ using (2.2)} \end{aligned}$$

This means that v is P-reachable from $v\alpha(a \diamond (t_2, t_3)))$, which implies that a is not P-bad. By choice of a this is impossible. We conclude that v is not P-reachable from $u\alpha(a \diamond (t_2, t_3)))$, therefore since t_1 and t'_1 are v^+-equivalent we have $u\alpha(a(t_1, t_2, t_3)) = u\alpha(a(t'_1, t_2, t_3))$. After repeating this argument we get $u\alpha(a(t_1, t_2, t_3)) = u\alpha(a(t'_1, t'_2, t'_3))$.

Now we turn to our second case, consider the tree $s = a(t'_1, t'_3, t'_3)$, t and s verify the conditions of our first case, therefore we have $u\alpha(t) = u\alpha(s)$. Now consider the tree $s' = a(t'_1, t'_1, t'_3)$ by Equation (8.1) we have $\alpha(s) = \alpha(s')$, also s' and t' are in the conditions of our first case, therefore we have $u\alpha(t') = u\alpha(s')$. Combining the three equalities we get $u\alpha(t) = u\alpha(t)$.

We can now prove Lemma 8.16. We proceed by induction on the number of labels a in t. If t contains no a then the result is clear as φ holds nowhere in t. Now suppose that t contains an a and consider a node x of t where φ holds, x is labeled with a. Let p be the context obtained by putting the port at x in t and s such that $t = ps$. Take $j \in \{0, n\}$ such that $s \in L_j$, by definition $v\alpha(p)$ is P-reachable from v. Therefore by our preliminary result we have $v\alpha(p)\alpha(s) = v\alpha(p)\alpha(a_j)$. Take $t' = pa_j$, by construction t' contains less labels a than t, therefore by induction we have $v\alpha(t') = v\alpha(t'[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k])$. By construction we have $(t'[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k] = (t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k]$. We get that $v\alpha(t) = v\alpha(t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k])$ which ends the proof.

□

Case 2: There is no P-bad label. In this case we show that over trees that are neither leaves nor twigs $L^P_{v,h}$ is either empty or universal. This concludes this case because thanks to Lemma 8.8 languages of leaves and languages of twigs are easily expressed in $EF + F^{-1}$.

Take two X-trimmed P-valid trees t, t' that are not leaves nor twigs, we show that $v\alpha(t) = v\alpha(t')$. Take $g = \alpha(t)$ and $f = \alpha(t')$, since t, t' are not leaves and are X-trimmed by hypothesis on X for this case we have $g, f \in H^P_m$. We first show the following Lemma:

Lemma 8.17. *There exists P-valid $u_g, u_f \in V$ such that for all P-valid $h \in H$, $u_g h = g$ and $u_f h = f$.*

Proof. This is a consequence of g, f being in H^P_m, therefore we only do the proof for g. Let $\{h_1, ..., h_n\}$ be the set of P-valid types and take $w \in W_2$ that is P-Valid. Consider the type $((w \diamond h_1)...(w \diamond h_n))^\omega h_n$. It is P-valid by construction and in H_m because it is P-reachable from all P-valid types. Therefore there exists a P-valid $x \in V$ such that $x((w \diamond h_1)...(w \diamond h_n))^\omega h_n = g$. Now consider the context type $u_g = x((w \diamond h_1)...(w \diamond h_n))^\omega w \diamond ((w \diamond h_1)...(w \diamond h_n))^\omega h_n)$, consider some P-valid $h \in H$, we have:

$$\begin{aligned} u_g h &= x((w \diamond h_1)...(w \diamond h_n))^\omega (w \diamond h)((w \diamond h_1)...(w \diamond h_n))^\omega h_n) \\ u_g h &= x((w \diamond h_1)...(w \diamond h_n))^\omega h_n \text{ using } \mathbf{DA} \\ u_g h &= g \end{aligned}$$

□

Now consider contexts such that u_g and u_f are their images:

$$\begin{aligned} u_g &= \alpha((c_1 \diamond a_1)...(c_n \diamond a_n)) \\ u_f &= \alpha((c_{n+1} \diamond a_{n+1})...(c_m \diamond a_m)) \end{aligned}$$

For the sake of simplifying the notations we supposed that all the labels on the path from the root to the port were of arity 2, the proof however extends to arbitrary rank in a straightforward way. Since there exists no P-bad label, there exists $a \in A_0$ such that v is P-reachable from $v\alpha(c_1 \diamond a)$.

Claim 8.18. *For all i, v is P-reachable from $v\alpha(c_i \diamond a)$.*

Proof. This is proved using Equation (8.3). Since there exists no P-bad label there exists $b \in A_0$ such that v is P-reachable from $v\alpha(c_i \diamond b)\alpha(c_1 \diamond a)$. Therefore there exists a P-valid $x \in V$ such that:

$$\begin{aligned} v &= v\alpha(c_i \diamond b)\alpha(c_1 \diamond a)x \\ v &= v(\alpha(c_i \diamond b)\alpha(c_1 \diamond a)x)^\omega \\ v &= v\alpha(c_i \diamond a)\alpha(c_1 \diamond b)(\alpha(c_i \diamond b)\alpha(c_1 \diamond a)x)^\omega \text{ using (8.3)} \end{aligned}$$

It follows that v is P-reachable from $v\alpha(c_i \diamond a)$. □

We write:

$$\begin{aligned} u'_g &= \alpha(c_1 \diamond a)...(c_n \diamond a) \\ u'_f &= \alpha(c_{n+1} \diamond a)...(c_m \diamond a) \end{aligned}$$

Using Claim 8.18, it follows by definition of u'_g and u'_f that:

$$\begin{aligned} vu'_g u'_g u'_f u'_f x &= v \quad \text{for some } P\text{-valid } x \in V \\ v(u'_g u'_g u'_f u'_f x)^\omega &= v \end{aligned}$$

Hence by definition of u_g:

$$v(u'_g u'_g u'_f u'_f x)^\omega (u_g u'_g u_f u'_f x)^\omega f = vg$$

Notice that we have $u'_g u'_g \dashv u_g u'_g$ and $u'_f u'_f \dashv u_f u'_f$, therefore using Equation (8.4):

$$\begin{aligned} vg &= v(u'_g u'_g u'_f u'_f x)^\omega u_f u'_f x (u_g u'_g u_f u'_f x)^\omega f \\ vg &= v(u'_g u'_g u'_f u'_f x)^\omega f \qquad \text{by definition of } u_f \\ vg &= vf \end{aligned}$$

This completes the proof of Proposition 8.7.

Chapter 9

Boolean Combination of Existential First Order Formulas over Trees of Bounded Rank

In this chapter we provide a decidable characterization for the class of languages of trees of bounded rank definable by a boolean combination of $\Sigma_1(<_{\mathbf{v}})$ formulas.

A formula of BC-$\Sigma_1(<_{\mathbf{v}})$ formula is a boolean combination of formulas of the form:

$$\exists x_1 \dots \exists x_n \ \varphi$$

With φ a quantifier free FO($<_{\mathbf{v}}$) formula. More details about BC-$\Sigma_1(<_{\mathbf{v}})$ can be found in Chapter 6.

Recall that in Chapters 2 and 6 we quoted decidable characterizations for the corresponding classes in the settings of words and forests. In the word setting, the result was using monoids:

Theorem 2.8. ([Sim75]) *Fix L a regular word language. Then L is definable in BC-$\Sigma_1(<)$ iff its syntactic monoid M verifies for all $u, v \in M$:*

$$(uv)^\omega = (uv)^\omega u = v(uv)^\omega \tag{2.4}$$

The characterization on forests was using forests algebras together with the piece relation, $\preceq$, that we defined in Chapter 6:

Theorem 6.8. ([BSS08]) *Fix L a regular forest language. Then L is definable in BC-$\Sigma_1(<_{\mathbf{v}})$ iff its syntactic forest algebra using monoids (H, V) verifies for all $u, v \in V$ such that $v \preceq u$:*

$$h + g = g + h \quad \textit{for } h, g \in H \tag{6.6}$$

$$u^\omega = u^\omega v = v u^\omega \tag{6.7}$$

Notice that in particular, Theorem 6.8 generalizes Theorem 2.8. However it is impossible to express in BC-$\Sigma_1(<_{\mathbf{v}})$ that a tree is of rank k for some fixed integer k. Therefore Theorem 6.8 does not extend to trees of bounded rank in a simple way. We will see that restricting the maximal number of children in the trees actually leads to many problems.

Our characterization in the bounded rank setting contains an equation that is similar to Equation (6.7). However, we will see that we need to add two new equations. Note that these three equations use our adapted notion of the piece relation we introduced in Chapter 7. The "only if" direction of the characterization is proved using the same arguments as the proof of Theorem 6.8. The difficult part of the proof is the "if" direction.

We make an important restriction in this chapter. Our characterization only applies to trees of rank 2. It should be noted that in this case the extension to arbitrary rank is not straightforward. In Section 9.4, we will point out the difficulties tied to extending the proof to arbitrary rank k and discuss ideas for overcoming these difficulties. From now on all the trees we consider are of rank 2.

The chapter is organized as follows: in Section 9.1 we provide a characterization for BC-$\Sigma_1(<_{\mathbf{v}})$ by means of 2-algebras, we also compare this characterization with the one provided in [BSS08] for forests. Sections 9.2 and 9.3 are devoted to the proof of the characterization. In the last Section, Section 9.4, we discuss the case of trees of rank k for k greater than 2.

9.1 Characterization of BC-$\Sigma_1(<_{\mathbf{v}})$ Over Trees of Rank 2

As in the case of unranked trees in [BSS08] our characterization of BC-$\Sigma_1(<_{\mathbf{v}})$ relies on the notion of pieces we already used in Chapter 7 for the characterization of $\Delta_2(<_{\mathbf{v}})$. We recall this notion for trees of rank 2 below:

Pieces. Given an alphabet A of rank 2 and two trees s, t of rank 2 over A, we say that t is a *piece* of s iff there exists an injective morphism of the nodes of t to the nodes of s that preserves labels and the descendant relation, we write $s \preceq t$ (see Figure 9.1).

Since contexts and 2-contexts are trees with special leaves we can extend this definition to contexts and 2-contexts. Given two 2-contexts p, q, we say that q is a piece of p iff seen as trees $q \preceq p$ (see Figure 9.2).

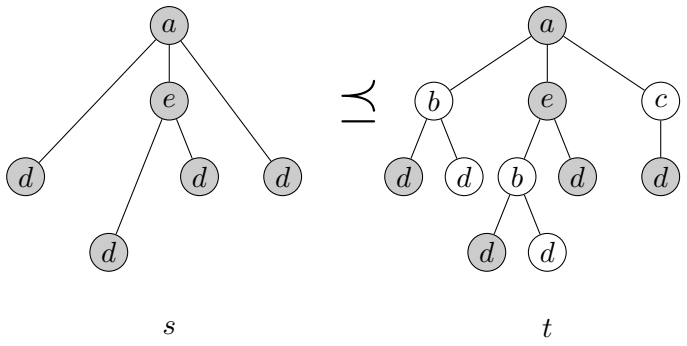


Figure 9.1: Illustration of the Piece Relation on trees.

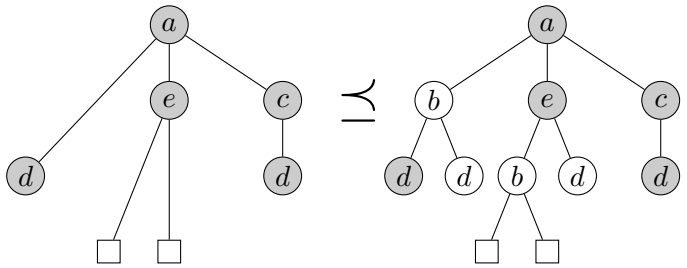


Figure 9.2: Illustration of the Piece Relation on 2-contexts.

Given a 2-algebra (H, W_1, W_2) and a morphism $\alpha : A^{\Delta} \to (H, W_1, W_2)$ we extend the notion of piece to elements of this 2-algebra. Given $h, g \in H$ we say that h is a *piece* of g and write $h \preceq g$ iff there exists two trees t, s such that $\alpha(t) = h$, $\alpha(s) = g$ and $t \preceq s$. Similarly, given $u, v \in W_i$ we say that u is a *piece* of v and write $u \preceq v$ if and only if there exists two i-contexts p, q such that $\alpha(p) = u$, $\alpha(q) = v$ and $p \preceq q$.

We are now ready to state the decidable characterization of BC-$\Sigma_1(<_{\mathbf{v}})$ over trees of rank 2. It uses the notion of 2-algebra. Recall that this notion is defined in Chapter 5.

Theorem 9.1. *Fix L a regular language of trees of rank 2. Then L is definable in BC-$\Sigma_1(<_{\mathbf{v}})$ iff its syntactic 2-algebra (H, V, W_2) verifies for all $u, u', v \in V$ and all $w, w' \in W_2$:*

$$u^{\omega}v = u^{\omega} = vu^{\omega} \quad v \preceq u \tag{9.1}$$

$$u^{\omega}w = u^{\omega}w' \quad \exists h \in H \ (w \diamond h), (w' \diamond h) \preceq u \tag{9.2}$$

$$(w \diamond h)u^\omega = (w \diamond h')u^\omega \quad \exists w' \in W_2 \ (w' \diamond h), (w' \diamond h) \preceq u \tag{9.3}$$

.

Notice that the first identity is the same as the one stated for the characterization of $BC - \Sigma_1(<_{\mathbf{v}})$ over forest algebras in [BSS08] for unranked trees. The only difference is in the definition of the piece relation which is more restrictive in our setting. However, this small difference leads to many problems. In the proof of Theorem 9.1, we often select a set of nodes in a tree and consider the tree formed by these nodes as a piece of the initial tree. While in the unranked setting this operation always yields a forest, in the setting of trees of rank 2, this does not yield a tree in the general case (see Figure 9.3). Therefore we need to be much more careful in the ranked setting when extracting pieces of the trees we consider. In particular we say that a set of nodes of a tree (resp. context, 2-context) *describes a valid piece* of the same tree (resp. context, 2-context) iff the restriction of the tree (resp. context, 2-context) to this set of nodes forms a valid tree (resp. context, 2-context). Because of this restriction on the notion of piece, we need to add Equations (9.2) and (9.3). notice they are consequences of (6.7) in the unranked setting. Finally, notice that we are missing an equivalent to Equation (6.6). This is because it is already assumed in k-algebra that the trees are unordered.

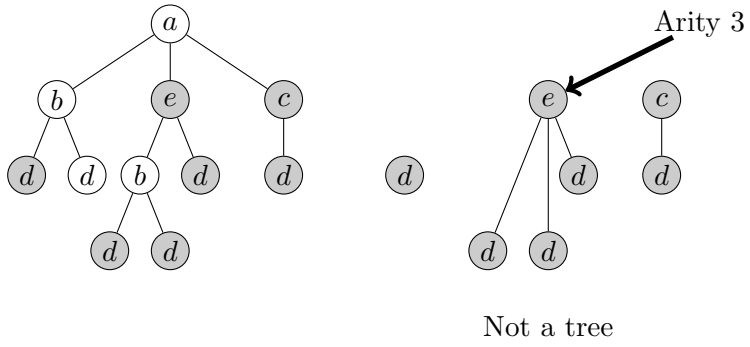


Figure 9.3: Set of nodes that do not describe a piece.

Notice that since the piece relation is computable (see Lemma 7.1), it follows from Theorem 9.1 that definability in BC-$\Sigma_1(<_{\mathbf{v}})$ is a decidable property of regular languages of trees of rank 2:

Corollary 9.2. *It is decidable whether a regular language of trees of rank* 2 *is definable in BC-*$\Sigma_1(<_{\mathbf{v}})$.

There are two directions to Theorem 9.1. We first consider the simpler "only if" direction. Equation 9.1 is identical to Equation (6.7) used in the corresponding characterization in the unranked setting. And in the unranked setting Equations (9.2) and (9.3) are simple consequences of 9.1. Using the same type of argument as in Chapter 7 for the characterization of $\Delta_2(<_{\mathbf{v}})$ in the bounded rank setting (see Proposition 7.4), we can show that the three identities are a necessary condition for being definable using a boolean combination of $\Sigma_1(<_{\mathbf{v}})$ formulas:

Proposition 9.3. *Let L be a regular tree language, if L is definable using a boolean combination of $\Sigma_1(<_{\mathbf{v}})$ formulas, then its syntactic 2-algebra verifies Equation* (9.1)*, Equation* (9.2) *and Equation* (9.3)*.*

Proof. It was proved in [BSS08] that Equation (9.1) is necessary for being definable in BC-$\Sigma_1(<_{\mathbf{v}})$ in the unranked setting. Any tree of rank 2 can be viewed as an unranked tree and the piece relation on trees of bounded rank is a restriction of the piece relation for unranked trees. Therefore this is also true for trees of rank 2. The argument is identical for Equations (9.2) and (9.3). □

The other direction is more difficult. We devote Section 9.2 and Section 9.3 to its proof. We prove that if the syntactic 2-algebra of a language verifies Equations (9.1),(9.2) and (9.3), then this language is definable using a boolean combination of $\Sigma_1(<_{\mathbf{v}})$ formulas. As for the unranked setting of [BSS08], this is done using an alternate definition of languages definable by boolean combination of $\Sigma_1(<_{\mathbf{v}})$ formulas, *piecewise testable languages*. We first give the definitions of piecewise testable languages in Section 9.2 and prove that they are exactly the languages definable by boolean combination of $\Sigma_1(<_{\mathbf{v}})$ formulas. Then, in Section 9.3, we use piecewise testable languages to prove Theorem 9.1.

9.2 Piecewise Testable Languages

A language is piecewise testable if its membership only depends on the pieces contained in the trees. More formally, a tree language L over A is piecewise testable iff if it can be defined as a boolean combination of languages of the form $\{t \mid s \preceq t\}$ for some tree s.

This definition is identical to the one given in [BSS08], however, the equality between piecewise testable languages and languages definable by a boolean combination of $\Sigma_1(<_{\mathbf{v}})$ formulas is a little more tedious in our setting. Indeed it remains simple to see that one can describe a piece using a $\Sigma_1(<_{\mathbf{v}})$ formula. However, the set of nodes selected by a $\Sigma_1(<_{\mathbf{v}})$ formula does not always describes a valid piece in the ranked setting. We solve this problem using a disjunction over all the smallest trees verifying this formula.

Proposition 9.4. *A tree language L over A is piecewise testable iff if it is definable by a boolean combination of $\Sigma_1(<_{\mathbf{v}})$ formulas.*

Proof. Given a tree s, we can define the language $\{t \mid s \preceq t\}$ with a $\Sigma_1(<_{\mathbf{v}})$ formula. Therefore, if a language is piecewise testable, it is definable by a boolean combination of $\Sigma_1(<_{\mathbf{v}})$ formulas.

For the other direction, we prove than any $\Sigma_1(<_{\mathbf{v}})$ formula is definable by a boolean combination of languages of the form $\{t \mid s \preceq t\}$. Take a $\Sigma_1(<_{\mathbf{v}})$ formula φ. It is of the form $\exists x_1 \cdots \exists x_n \Psi(x_1, \ldots, x_n)$ and let L be the language defined by φ. Consider the set of trees of depth at most n, which satisfy φ, we write this set $\{t_1, \ldots, t_m\}$. We prove that $L = \cup_{1 \leq i \leq m} \{t \mid t_i \preceq t\}$, which ends the proof.

Take $t \in \cup_{1 \leq i \leq m} \{t \mid t_i \preceq t\}$, there exists i such that $t_i \preceq t$. There exists an an injective mapping f of the nodes of t_i to the nodes of t that preserves the descendant relation and labels. By definition of t_i there exists positions $x_1, \ldots, x_n$ in t_i such that $t_i \models \varphi(x_1, \ldots, x_n)$. It follows that $t \models \varphi(f(x_1), \ldots, f(x_n))$. We have $t \in L$.

Take $t \in L$, then there exists positions $x_1, \ldots, x_n$ in t such that $t \models \varphi(x_1, \ldots, x_n)$. Consider t' the smallest piece of t such that the corresponding mapping f reaches all positions $x_1, \ldots, x_n$ in t. Since we took the smallest piece we have t' of depth smaller than n, therefore there exists i such that $t' = t_i$ and we have $t \in \{t \mid t_i \preceq t\}$. □

9.3 The Identities Are Sufficient

Fix a language L such that its syntactic 2-algebra verifies (9.1),(9.2) and (9.3). We show that L is piecewise testable. Using Proposition 9.4 this ends the proof of Theorem 9.1. We call α the syntactic morphism from L into its syntactic 2-algebra.

This result is a consequence of two properties that we state and prove below. Given two trees t and t' and an integer n we say that $t \cong_n t'$ if and only if t and t' have the same pieces of size n. $\cong_n$ is an equivalence relation and by definition each class is piecewise testable. Any piecewise testable language is an union of classes of $\cong_n$.

The first property is the following lemma taken from [BSS08]. The result remains true in our setting since it only relies on a pumping argument.

Lemma 9.5. ([BSS08]) *Let $n \in \mathbb{N}$, there exists $k \in \mathbb{N}$ such that for any two trees t, t' such that $t \cong_k t'$, there exists a tree s which verifies:*

$$s \preceq t \quad s \preceq t' \quad t \cong_n s \cong_n t'$$

We turn to the second property. It corresponds to a similar property in [BSS08], however the restriction on pieces in our setting makes it more technical.

Proposition 9.6. *There exists an integer $n \in \mathbb{N}$ such that (see Figure 9.4 for the second and third items):*

1. *For any context p, any tree t and any $b \in A_1$:*

$$p \cdot b \cdot t \cong_n p \cdot t \implies \alpha(p \cdot b \cdot t) = \alpha(p \cdot t)$$

2. *For any context p, any tree t, any $a \in A_2$ and any $c \in A_0$:*

$$p \cdot a \cdot (t, c) \cong_n p \cdot t \implies \alpha(p \cdot a \cdot (t, c)) = \alpha(p \cdot t)$$

3. *For any for any context p, any two trees t_1, t_2, any $a, a' \in A_2$ and any $c \in A_0$:*

$$p{\cdot}a'{\cdot}(a{\cdot}(t_1, t_2), c) \cong_n p{\cdot}a'{\cdot}(t_1, t_2) \implies \alpha(p{\cdot}a'{\cdot}(a{\cdot}(t_1, t_2), c)) = \alpha(p{\cdot}a'{\cdot}(t_1, t_2))$$

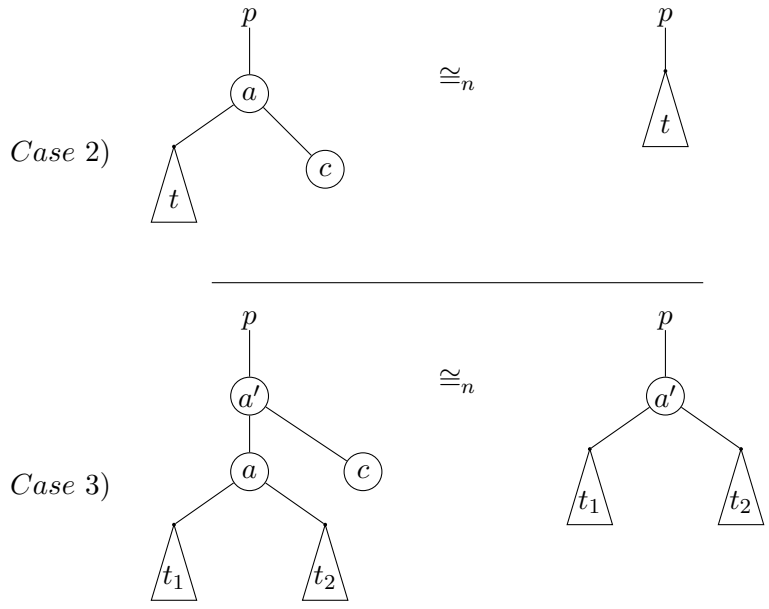


Figure 9.4: Illustration of the second and third properties in Proposition 9.6

Before proving Proposition 9.6, we show how to use it with Lemma 9.5 in order to end the proof of Theorem 9.1. Let n be as defined in Proposition 9.6. From

this n we get an integer k as defined in Lemma 9.5. We show that for any two trees t, t' we have $t \cong_k t' \Rightarrow \alpha(t) = \alpha(t')$. It follows that L is a union of classes of $\cong_k$ and therefore is piecewise testable, which ends the proof. Indeed, take two trees t, t' such that $t \cong_n t'$. From Lemma 9.5 we get a tree s such that $s \preceq t$, $s \preceq t'$ and $t \cong_n s \cong_n t'$. We conclude by showing the following lemma:

Lemma 9.7. *$\alpha(t) = \alpha(s)$ and $\alpha(t') = \alpha(s)$*

Proof. We only do the proof for $\alpha(t) = \alpha(s)$, the other item can be proved identically. We know that s is a piece of t therefore there exists a injective mapping from the nodes of s to the nodes of t that preserves labels and the descendant order. We say that a node x of t is *relevant* iff there exists a node y of s such that $f(y) = x$. We proceed by induction on the number of non relevant nodes in t. If all nodes of t are relevant then $s = t$ and the result follows. Otherwise we begin by proving two subresults that we will reuse several times in the proof:

Claim 9.8. *If t contains a non relevant inner node x of arity 1, we have $\alpha(t) = \alpha(s)$.*

Proof. We use the first item of Proposition 9.6. Let b be the label of x and p, t' be the context and tree such that $t = p \cdot b \cdot t'$. Let $t'' = p \cdot t'$, since x is not relevant, we have $s \preceq t'' \preceq t$. Therefore since $s \cong_n t$, we have $t'' \cong_n s$. Also t'' contains less relevant nodes than t, it follows by induction that $\alpha(t'') = \alpha(s)$. Finally we have $t'' \cong_n t$ and $t = pbt'$ and $t'' = pt'$, it follows from the first item of Proposition 9.6 that $\alpha(t'') = \alpha(t)$ and finally $\alpha(s) = \alpha(t)$. □

Claim 9.9. *If t contains a non relevant inner node x of arity 2, that has a non relevant child x' that is a leaf, we have $\alpha(t) = \alpha(s)$.*

Proof. We use the second item of Proposition 9.6. Let a be the label of x and c be the label of x'. Let p, t' such that $t = p \cdot a \cdot (t', c)$. Let $t'' = p \cdot t'$, since x and x' are not relevant, we have $s \preceq t'' \preceq t$. Therefore since $s \cong_n t$, we have $t'' \cong_n s$. Also t'' contains less relevant nodes than t, it follows by induction that $\alpha(t'') = \alpha(s)$. Finally we have $t'' \cong_n t$ and $t = pa(t', c)$ and $t'' = pt'$, it follows from the second item of Proposition 9.6 that $\alpha(t'') = \alpha(t)$ and finally $\alpha(s) = \alpha(t)$. □

Given this result we conclude the proof by distinguishing three cases. First we treat the case where s or t is a leaf. In a second case we treat the case where t has subtree that is not a leaf and contains only non relevant nodes. Finally we suppose that non of the first two cases hold.

Case 1: Either s or t is a leaf. Since we have $t \cong_n s$, both s and t are leaves. We get that $s = t$ and $alpha(s) = \alpha(t)$.

Case 2: There exists a subtree of t that is not a leaf and contains only non relevant nodes. In this case there exists a position x in t such that x is not relevant and all its descendants are leaves that are not relevant. Depending on the arity of x we are either in the setting of Claim 9.8 or Claim 9.9.

Case 3: We are neither in Case 1 nor Case 2. We know that s and t are not leaves and that positions x of t such that all nodes of the subtree at position x are not relevant are leaves. Since s is not a leaf t contains at least one relevant inner node. Also, since $s \neq t$ and s is a piece of t, t contains at least one non relevant node. We show that we can suppose that this node is an inner node. Indeed it is impossible for all the non relevant nodes of t to be leaves since s is a valid piece of t. Let y be an inner node of t such that:

- y is relevant and has an inner descendant node that is non relevant.
- No descendant of y verifies that property.

By choice of y, it has a non relevant child x that is an inner node. If x is of arity 1 we are in the setting of Claim 9.8. Suppose now that x is of arity 2. Since we are not in the previous case, we know that x contain relevant descendants (otherwise x would be a leaf). If x has a child whose subtree contains no relevant nodes then by hypothesis this child is a leaf and we are in the setting of Claim 9.9. Therefore, we suppose that both children of x contain relevant nodes. In this case we claim that y is of arity 2 and has a child y' whose subtree contains no relevant nodes and is therefore a leaf. Indeed x is non relevant and has two children that are relevant. Since y is relevant, the only way for s to be a valid piece of t is for the tree rooted at the sibling of x to be completely non relevant so there are two ports available below y in order to plug the two children of x, otherwise y would have at least three children in the piece (see Figure 9.5). We call a' the label of y, a the label of x and c the label of y'. We have $t = p \cdot a' \cdot (a \cdot (t_1, t_2), c)$, let $t'' = p \cdot a' \cdot (t_1, t_2)$. Since x and y' are not relevant, we have $s \preceq t'' \preceq t$. Therefore since $s \cong_n t$, we have $t'' \cong_n s$. Also t'' contains less relevant nodes than t, it follows by induction that $\alpha(t'') = \alpha(s)$. Finally we have $t'' \cong_n t$ and $t = p \cdot a' \cdot (a \cdot (t_1, t_2), c)$ and $t'' = p \cdot a' \cdot (t_1, t_2) t = pa(t', c)$, it follows from the item case of Proposition 9.6 that $\alpha(t'') = \alpha(t)$ and finally $\alpha(s) = \alpha(t)$.

□

It remains to prove Proposition 9.6. There are three items to prove. Notice that the first item is similar to the corresponding proposition in [BSS08] in the unranked setting. The second and third items are new. The three proofs follow the same proof structure as in the unranked setting. The first item shares the

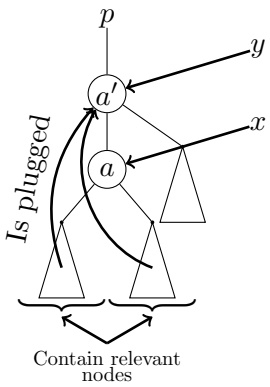


Figure 9.5: Third Case of Lemma 9.7

most similar proof. The most significant difference with the unranked setting is that we have to do a lot of technical work to ensure that we work with valid pieces. The second and third items need more work. We give a complete proof for the second item and then explain how to adapt the proof to the first and third items.

9.3.1 Second Item of Proposition 9.6

The proof structure is similar to the one used for the first item in the unranked setting. However we must be very careful to ensure that we consider only valid pieces at each step of the proof.

Fix p a context, t a tree, $a \in A_2$ and $c \in A_0$ as described in the second item of Proposition 9.6. We write $\alpha(t) = h$ and $v = \alpha(p)$. We write:

$$\begin{aligned} s &= p \cdot a \cdot (t, c) \\ s' &= p \cdot t \end{aligned}$$

Our goal is to prove that if $s \cong_n s'$ then $\alpha(s) = \alpha(s')$. For that we need to give a few definitions:

Good Decomposition. Consider some tree r and X a set of nodes of r, if X describes a valid piece of r we write $r[X]$ that piece. In that case, we say that $x, x' \in X$ is a *good decomposition* of r if:

- x has label a.
- In $r[X]$, x' is a child of x and is a leaf labeled with c.

- In $r[X]$, the subtree at the other child of x has type h.
- In $r[X]$, the context obtained by putting the port at x has type v.

Notice that this implies that if $x, x' \in X$ is a good decomposition of r then $\alpha(r[X]) = \alpha(s)$ and $\alpha(r[X - \{x, x'\}]) = \alpha(s')$.

Fractals A *fractal* is a bested sequence of good decompositions. More formally, consider a sequence of good decompositions $x_1, x'_1 \in X_1 \cdots x_l, x'_l \in X_l$. We say the sequence $x_1, x'_1 \in X_1 \cdots x_l, x'_l \in X_l$ is a fractal of length l iff:

$$X_i \subseteq X_{i+1} - \{x_{i+1}, x'_{i+1}\}$$

We call a *subfractal* a subsequence of a fractal. Notice that a subfractal is also a fractal.

We use the notion of fractal to conclude the proof of this case of Proposition 9.6. First we show that by choosing a big enough $n \in \mathbb{N}$, $s \cong_n t$ entails the existence of a fractal of any desired length. We then show that if there exists a big enough fractal then $\alpha(t) = \alpha(s)$. By combining the two results we conclude the proof of the second item of Proposition 9.6.

Lemma 9.10. *$\forall l \in \mathbb{N}$, there exists $n \in \mathbb{N}$ such that if $s' \cong_n s$ then there exists a fractal of length l inside s.*

Proof. The proof is similar to the one used in the unranked setting. We proceed by induction on l, by definition of s there exists a fractal of length 1 in s.

Assume now that there exists m such that if $s' \cong_m s$ then there exists a fractal of length l inside s. Using a pumping argument we can show that there exists an integer η such that if a tree T contains a fractal of length l then there exists a piece of that tree of size η that also contains a fractal of length l. We choose $n = max(\eta, m)$, suppose that $s' \cong_n s$, it follows from $n \geq m$ and from the induction hypothesis that s contains a fractal of length l. By choice of η it follows that there is a piece of s size η that contains a fractal of length l. By choice of n this piece can be found in s'. By definition of s we can complete this fractal into a fractal of length $l + 1$ in s by fixing X_{l+1} as the set of all node of s. □

Proposition 9.11. *There exist an integer l such that if there exists a tree r containing a fractal of length l we have:*

$$\alpha(t) = \alpha(s)$$

Combining Proposition 9.11 and Lemma 9.10 we get that there exists an integer n such that if $s \cong_n t$ then $\alpha(s) = \alpha(t)$ which concludes the proof of the second item of Proposition 9.6.

The rest of this section is devoted to the proof of Proposition 9.11. We proceed in a way similar to the one followed in the unranked case. We refine the notion of fractal, first with the notion of *tame fractal* and then with the notion of *monochromatic tame fractal*. We then prove that the existence of a big enough fractal entail the existence of a tame fractal and subsequently a monochromatic tame fractal of any desired length. Finally we prove that the existence of a big enough monochromatic tame fractal implies that $\alpha(t) = \alpha(s)$ which concludes the proof of Proposition 9.11. The main differences with the unranked case occur in the transition from fractals to tame fractals and the usage of monochromatic tame fractals to conclude. The transition from tame fractal to monochromatic tame fractals is identical to the unranked setting.

Tame Fractals The definition of tame fractals is more tedious than in the unranked setting. First in our setting we have two important nodes, x, x', to handle whereas there was only one in the unranked setting. Also, in our setting we need to add one extra property to the definition. This property is designed to ensure that the successive subsets of nodes given by the fractal describes valid pieces of the contexts we define.

Let r be a tree and consider $x_1 \in X_1 \cdots x_n \in X_n$ a fractal of length n inside r. There are three mutually exclusive types of tame fractals:

Bottom-up tame fractal We say this fractal is a *bottom-up tame fractal* iff there exists $n+1$ contexts $q, q_1, \ldots, q_n$ and a tree r' such that:

- $r = qq_n \cdots q_1 r'$.
- For all i, x_i, x'_i are in q_i.
- For $i < i'$, $q_i \cap X_{i'}$ is a valid piece of q_i.

Top-down tame fractal We say this fractal is a *top-down tame fractal* iff there exists $n+1$ contexts $q, q_1, \ldots, q_n$ and a tree r' such that:

- $r = qq_1 \cdots q_n$.
- For all i, x_i, x'_i are in q_i.
- For $i < i'$, $q_i \cap X_{i'}$ is a valid piece of q_i.

Scattered top-down tame fractal We say this fractal is a *scattered top-down tame fractal* iff there exists $2n+2$ contexts $q, q', q_1, \ldots, q_n, \tilde{q}_1, \ldots, \tilde{q}_n$ and a tree r' such that:

- $r = qq_1 \cdots q_n q' \tilde{q}_n \cdots \tilde{q}_1 r'$.
- For all i, x_i is in q_i.
- For all i, x'_i is in $\tilde{q}_i$.
- For $i < i'$, $q_i \cap X_{i'}$ is a valid piece of q_i and $\tilde{q}_i \cap X_{i'}$ is a valid piece of $\tilde{q}_i$.

A fractal is tame iff it is either a bottom-up tame fractal, a top-down tame fractal or a scattered top-down tame fractal. The three categories of tame fractals are depicted in Figure 9.6. Notice that the last item of all categories allows us to write $q_i[X_{i'}]$ and $\tilde{q}_i[X_{i'}]$ for $i < i'$.

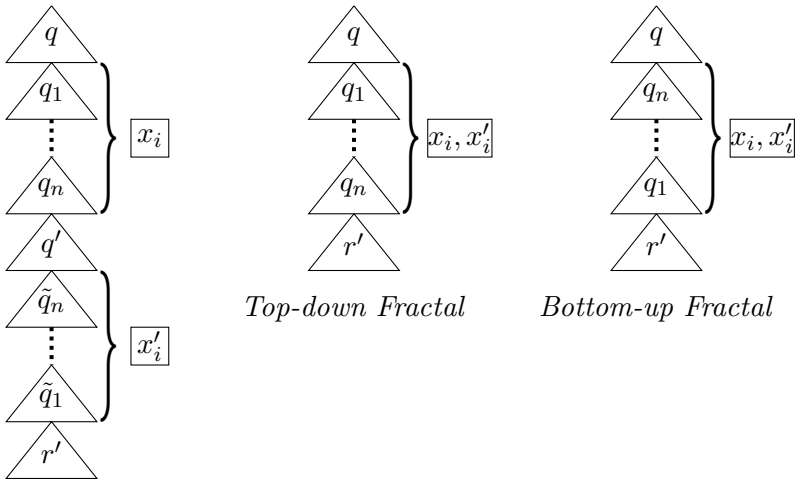


Figure 9.6: Tame Fractals

We call *skeleton* of a tame fractal the sequence of nodes that goes from the root of r to the root of r'.

Lemma 9.12. *$\forall l \in \mathbb{N}$, $\exists n \in \mathbb{N}$ such that given a tree r, if r contains a fractal of length n then r contains a tame fractal of length l.*

Proof. The proof of this result is more tedious than in the unranked setting. Indeed in our setting we have to manipulate more nodes and we have to guarantee

the last condition of tameness which did no exist in the unranked setting. We use the following claim taken from [BSS08]:

Claim 9.13. *Let $m \in \mathbb{N}$. There exists an integer $f(m)$ such that for every tree r, and every set Y of at least $f(m)$ nodes there is a decomposition $r = qq_1 \cdots q_m r'$ where each q_i contains at least a node of Y.*

Proof. The proof is similar to the one in the unranked setting. However, in our setting contexts that have the port in the root are not allowed. Because of this we need to use the hypothesis that the rank of the trees is bounded with rank k in order to conclude.

Let Z be the set of nodes in r which are closest common ancestors of two distinct nodes in Y. We call the *degree* of $z \in Z$ the number of nodes $z' \in Z \cup Y$ such that all nodes in the path between z and z' are outside Z. We show that the degree of a node $z \in Z$ is bounded by k. Indeed, Z contains all the nodes that are closest common ancestors of two nodes of Z. Therefore for each child of z there can be at most one descendant z' of this child such that all nodes in the path between z and z' are outside Z. Since z is at most of arity k, it follows that the degree of z is bounded by k.

Let $m' = max(k+1, m)$, we choose $f(m) = (m')^{m'}$. Two cases may hold, either there exists a node in $Z \cup Y$ with degree m', or $Z \cup Y$ contains a chain of length m'. Since the first case is impossible, the second case holds and gives the decomposition. □

We use this claim to finish the proof of Lemma 9.12. For any integer m we write $g(m) = f((3m)^2)$ with f as defined in Claim 9.13 (notice that by definition $f(m) > m$). We choose $n = g(g(l+1))$, let $y_1, y_1' \in Y_1 \cdots y_n, y_n' \in Y_n$ be a fractal of length n. We show that we can extract a tame subfractal of length l. We construct this subfractal in two steps. In the first step, we use Claim 9.13 to construct a subfractal that admits a monotone decomposition satisfying the first item of tameness. Given this decomposition we then get a skeleton that we use to construct our final decomposition. In the second step we extract a last subfractal and construct the associated decomposition.

We use Claim 9.13 with Y the set of nodes y_i. We get a decomposition of length $m = (3f(l+1))^2$ such that each context contains a node y_i for some i. While this decomposition might not be monotone we can extract a monotone subsequence of length $3f(l+1)$. We get a new fractal $x_1, x_1' \in X_1 \cdots x_m x_m' \in X_m$ along with a decomposition $r = qq_1q_2 \cdots q_m r'$ or $r = qq_mq_{m-1} \cdots q_1 r'$ such that for each i, x_i is in q_i. This decomposition defines a skeleton, which is the sequence of nodes going from the root of r to the root of r'.

We finish the proof by distinguishing two cases depending on whether our subfractal is bottom-up or top-down.

Case 1: The Fractal is Bottom-up. For each good decomposition in the fractal, we define a special node that we call x_i''. For every i, x_i'' is x_i if x_i is on the skeleton. Otherwise x_i'' is the closest ancestor of x_i on the skeleton that is in X_{i+1} (x_i'' exists because both x_i and x_{i+1} are in X_{i+1} and their closest common ancestor must be by construction on the skeleton). Notice that since by hypothesis our extracted fractal is bottom-up, x_i'' is either an ancestor of x_{i+1}'' or equal to x_{i+1}''.

We want to build a new decomposition $r = pp_m \cdots p_1 r''$ that verifies all the conditions of a bottom-up tame fractal. The idea is that we want to to choose x_i'' as root of the p_i for all i. The problem is that we might have $x_i'' = x_{i+1}''$ for some i. We show that we can extract a subfractal of big enough length such that it does not happen:

Claim 9.14. *We can extract a subfractal of length $f(l+1)$ such that for all i, $x_i'' \neq x_{i+1}''$.*

Proof. We show that for all i it is impossible that $x_i'' = x_{i+1}'' = x_{i+2}''$. It then follows that is sufficient to extract a subfractal by keeping a node every two nodes x_i. Since the original fractal was of length $3f(l+1)$ we get a fractal of length greater than $f(l+1)$ which satisfies the desired property.

Suppose that for some i, $x_i'' = x_{i+1}'' = x_{i+2}''$. Let x'' be this node. Since the fractal is bottom-up this puts x'' in or above q_{i+2}. Therefore since by hypothesis, $x_i \in q_i$ and $x_{i+1} \in q_{i+1}$, we have $x_i \neq x_i''$ and $x_{i+1} \neq x_{i+1}''$, by definition it follows that neither x_i nor x_{i+1} are on the skeleton. Also we have $x_{i+2}'' = x_i'' \in X_{i+1}$ therefore $x_{i+2}'' \neq x_{i+2}$ and x_{i+2} is not on the skeleton either (we depict this situation in Figure 9.7). By definition there is no node between x_{i+1} and x'' that is in X_{i+2} and on the skeleton. Therefore, x'' is the first common ancestor of x_i and x_{i+1} in $r[X_{i+2}]$. Since it is also an ancestor of the node x_{i+2}, this gives x'' arity 3 which is impossible.

□

Since $f(l+1) > l$, it follows from Claim 9.14 that we have a subfractal $z_1, z_1' \in Z_1 \cdots z_l z_l' \in Z_l$ of length$l$ such that for all i, $z_i'' \neq z_{i+1}''$. We construct a decomposition $r = pp_l \cdots p_1 r''$ as follows: for all i we choose x_i'' as the root of p_i. This fixes the port of p_i as x_{i-1}'' (we choose the port of p_1 as some leaf below z_1 that is in Z_2 and different of z_1').

By construction we have for all i $z_i \in p_i$. Also the port of p_i is in Z_i therefore it cannot be between z_i and z_i' which are parent and child in Z_i. Therefore for all i $z_i' \in p_i$. Finally, since both the root and the port of p_i are in Z_{i+1}, for all $i' > i$, $Z_{i'}$ describes a valid piece of p_i. Therefore the fractal is a bottom-up tame fractal of length l and we are done.

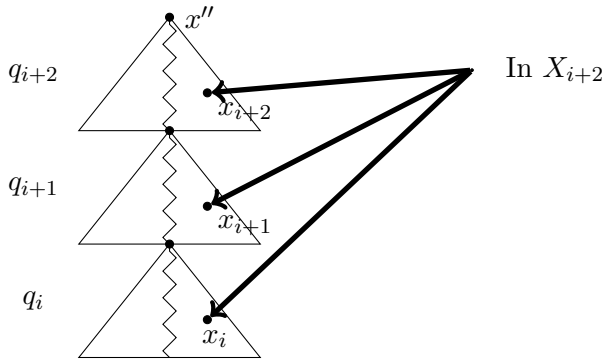


Figure 9.7: Illustration of Case 1

Case 2: The Fractal is Top-down As we did in the previous case, for each good decomposition in the fractal, we define a special node that we also call x_i''. However the definition is slightly different from the one of the previous case. For every i, x_i'' is x_i if x_i is on the skeleton. Otherwise x_i'' is the closest ancestor of x_i on the skeleton that is in X_i (x_i'' exists because both x_i and x_{i-1} are in X_i and their closest common ancestor is on the skeleton). Notice that since by hypothesis our extracted fractal is top-down, x_i'' is either an descendant of x_{i+1}'' or equal to x_{i+1}''.

We want to build a new decomposition $r = pp_1 \cdots p_m r''$ that verifies either the conditions of a top-down tame fractal or the conditions of a scattered top-down tame fractal. As in the previous case, the idea is that we want to to choose x_i'' as root of the p_i for all i. Again the problem is that we might have $x_i'' = x_{i+1}''$ for some i. We solve this with a result similar to Claim 9.14:

Claim 9.15. *We can extract a subfractal of length $f(l+1)$ such that for all i, $x_i'' \neq x_{i+1}''$.*

Proof. We show that for all i it is impossible that $x_i'' = x_{i+1}'' = x_{i+2}'' = x_{i+3}''$. It then follows that it is sufficient to extract a subfractal by keeping a node x_i out of every three nodes. Since the original fractal was of length $3f(l+1)$ we get a fractal of length greater than $f(l+1)$ which satisfies the desired property.

Suppose that for some i, $x_i'' = x_{i+1}'' = x_{i+2}'' = x_{i+3}''$. Let x'' be that node. As we did for Claim 9.14 we show that this implies arity 3 for x'', which is impossible. Since the fractal is top-down x'' is in or above q_i. Therefore since by hypothesis, $x_{i+3} \in q_{i+3}$,$x_{i+2} \in q_{i+2}$ and $x_{i+1} \in q_{i+1}$, x_{i+3},x_{i+2} and x_{i+1} are not on the skeleton. Since x'' is is the first ancestor of x_{i+3} that is on the skeleton, it follows that x_{i+3},x_{i+2} and x_{i+1} must be in three separate children of x'' in $r[X_{i+3}]$ (see Figure 9.8). This gives x'' arity 3 which is impossible.

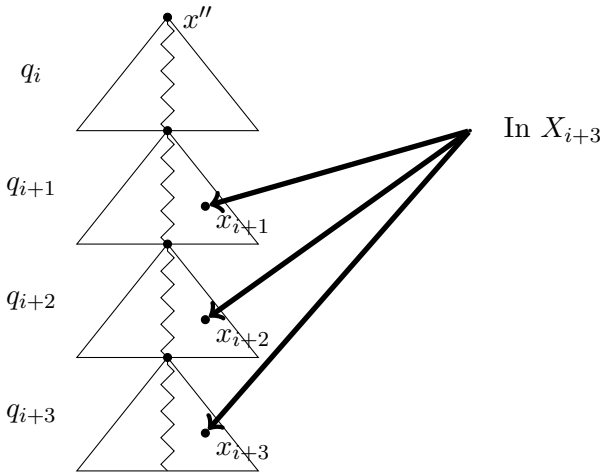


Figure 9.8: Illustration of Case 2

□

It follows from Claim 9.14 that we have a subfractal $z_1, z'_1 \in Z_1 \cdots z_{f(l+1)} z'_{f(l+1)} \in Z_{f(l+1)}$ of length $f(l+1)$ such that for all i, $z''_i \neq z''_{i+1}$. We construct a decomposition $r = pp_1 \cdots p_{f(l+1)} r''$ as follows: for all i we choose x''_i as the root of p_i. This fixes the port of p_i as x''_{i+1} (we choose the port of $p_{f(l+1)}$ as some arbitrary leaf below $z_{f(l+1)}$).

By construction we have for all i $z_i \in p_i$ and since both the root and the port of p_i are in Z_{i+1}, for all $i' > i$, $Z_{i'}$ describes a valid piece of p_i. The problem is that z'_i might not be in p_i for all i.

We finish the proof by ordering the nodes z'_i. Since we have a fractal of length $f(l+1)$. We can redo the whole process with the nodes z'_i replacing the nodes z_i and get a new subfractal of length $l+1$. This fractal $z_1, z'_1 \in Z_1 \cdots z_{l+1} z'_{l+1} \in Z_{l+1}$ is either top-down with a decomposition $r = \tilde{p}\tilde{p}_1 \cdots \tilde{p}_{l+1} r'''$ or bottom-up with a decomposition $r = \tilde{p}\tilde{p}_{l+1} \cdots \tilde{p}_1 r'''$. And for all i, $z'_i \in \tilde{p}_i$ and for all $i' > i$ $Z_{i'}$ describes a valid piece of $\tilde{p}_i$. We now distinguish two subcases depending on whether this new subfractal is bottom-up or top-down.

Subcase 2.a: The Subfractal is Top-down. We show that the fractal is actually a top-down tame fractal. Indeed by construction, for all i the root of $\tilde{p}_i$ is in Z_i (see the construction of x''_i in Case 2). Therefore since in $r[Z_i]$ z'_i is a child of z_i, this root cannot be between them. It follows that for all i z_i in $\tilde{p}_i$. The fractal is a top-down tame fractal of length $l+1$ which means that we have a top-down subfractal of length l.

Subcase 2.b: The Subfractal is Bottom-up. We show that the fractal is actually a scattered top-down tame fractal. Recall that we still have our initial top-down decomposition $r = pp_1 \cdots p_{l+1}r''$ such that for all i, $z_i \in p_i$ and for all $i' > i$, $Z_{i'}$ describes a valid piece of p_i. We show that the root of p_{l+1} is an ancestor of the port of $\tilde{p}_{l+1}$. Indeed $z'_{l+1} \in \tilde{p}_{l+1}$ is a descendant of $z_{l+1} \in p_{l+1}$.

Now fix p' as the context with the same root as p_{l+1} and the same port as $\tilde{p}_{l+1}$. By construction we get $r = pp_1 \cdots p_l p' \tilde{p}_l \cdots \tilde{p}_1 r'''$. We have a fractal of length l such that for all i, $z_i \in p_i$ and $z'_i \in \tilde{p}_i$ and for $i < i'$, $Z_{i'}$ describes a valid piece of p_i and $\tilde{p}_i$. The fractal is a scattered top-down tame fractal of length l.

This finishes the proof of Lemma 9.12.

□

Monochromatic Tame Fractals The definition of monochromatic tame fractals is similar to the one given in the unranked setting. However we need to modify it slightly so that it corresponds to our notion of tame fractal. Fix r a tree and consider $x_1, x'_1 \in X_1 \cdots x_n, x'_n \in X_n$ a tame fractal inside r. Let $q, q_1, \ldots, q_n, q', \tilde{q}_1, \ldots, \tilde{q}_n$ and r' be as in the definition of tame fractals. We consider two cases depending on the nature of the tame fractal. If the fractal is bottom-up, for any l_1, l_2, l_3 such that $0 \leq l_1 < l_2 < l_3 \leq l + 1$ we write:

$$\begin{array}{ll} u_{l_1,l_2,l_3} = \alpha(q_{l_2} \cdots q_{l_1+1}) & \text{for } l_3 = n+1 \\ u_{l_1,l_2,l_3} = \alpha((q_{l_2} \cdots q_{l_1+1})[X^{l_3}]) & \text{for } l_3 \leq n \end{array}$$

If the fractal is top-down, for any l_1, l_2, l_3 such that $0 \leq l_1 < l_2 < l_3 \leq n + 1$ we write:

$$\begin{array}{ll} u_{l_1,l_2,l_3} = \alpha(q_{l_1+1} \cdots q_{l_2}) & \text{for } l_3 = n+1 \\ u_{l_1,l_2,l_3} = \alpha((q_{l_1+1} \cdots q_{l_2})[X^{l_3}]) & \text{for } l_3 \leq n \\ \tilde{u}_{l_1,l_2,l_3} = \alpha(\tilde{q}_{l_2} \cdots \tilde{q}_{l_1+1}) & \text{for } l_3 = n+1 \\ \tilde{u}_{l_1,l_2,l_3} = \alpha((\tilde{q}_{l_2} \cdots \tilde{q}_{l_1+1})[X^{l_3}]) & \text{for } l_3 \leq n \end{array}$$

Notice that these are well defined because of the last hypothesis in the definition of tame fractals (For $i' > i$ $X_{i'}$ describes a valid piece of $q_i, \tilde{q}_i$). Also notice that the $\tilde{u}_{l_1,l_2,l_3}$ are only defined in the case of a scattered top-down tame fractal.

We say the tame fractal is monochromatic iff there exists $u, \tilde{u} \in V$ such that for all $0 \leq l_1 < l_2 < l_3 \leq l + 1$ and all $0 \leq l'_1 < l'_2 < l'_3 \leq l + 1$:

$$\begin{array}{ccccc} u_{l_1,l_2,l_3} & = & u_{l'_1,l'_2,l'_3} & = & u \\ \tilde{u}_{l_1,l_2,l_3} & = & \tilde{u}_{l'_1,l'_2,l'_3} & = & \tilde{u} \end{array}$$

Lemma 9.16. *$\forall l \in \mathbb{N}$, $\exists n \in \mathbb{N}$ such that for all trees r, if r contains a tame fractal of length n then r contains a monochromatic tame fractal of length l.*

Proof. This is proved using a combinatorial argument using Ramsey Theorem. The proof is identical to the one given in [BSS08]. □

Combining Lemma 9.12 and Lemma 9.16 we get that the existence of a big enough fractal implies the existence of a monochromatic tame fractal of any desired length. Using this result we conclude the proof of Proposition 9.11 by showing that the existence of a big enough monochromatic fractal implies that $\alpha(s') = \alpha(s)$. Recall that ω is the number such that for any $v \in V$ in the syntactic k-algebra of l, v^ω is idempotent.

Lemma 9.17. *If there exists a monochromatic tame fractal of length $l = \omega + 1$, then $\alpha(s) = \alpha(s')$.*

Proof. Consider r a tree such that there exists a monochromatic tame fractal $x_1 \in X_1 \cdots x_l \in X_l$ in r. We consider three cases depending on the type of tame fractal.

Case1: Bottom-up tame fractal. Consider $r = qq_lq_{l-1}\cdots q_1r'$, the decomposition of the fractal. Since $x_l \in X_l$ is a good decomposition, by definition of s we have:

$$\begin{aligned}
\alpha(s) &= \alpha(r[X_l]) \\
\alpha(s) &= \alpha((qq_l)[X_l]) \cdot u_{l-2,l-1,l} \cdots u_{0,1,l} \cdot \alpha(r'[X_l]) \\
\alpha(s) &= \alpha((qq_l)[X_l]) \cdot u^\omega \cdot \alpha(r'[X_l]) \text{ by monochromaticity and by choice of } l
\end{aligned}$$

Also by definition of s' we have (since $x_l, x'_l \in q_l$):

$$\begin{aligned}
\alpha(s') &= \alpha(r[X_l - \{x_l, x'_l\}]) \\
\alpha(s') &= \alpha((qq_l)[X_l - \{x_l, x'_l\}]) \cdot u_{l-2,l-1,l} \cdots u_{0,1,l} \cdot \alpha(r'[X_l]) \\
\alpha(s') &= \alpha((qq_l)[X_l - \{x_l, x'_l\}]) \cdot u^\omega \cdot \alpha(r'[X_l]) \text{ by monochromaticity and by choice of } l
\end{aligned}$$

By monochromaticity, we know that $\alpha(q_l) = u$. Therefore if X_l describes a valid piece of q_l we get $\alpha(q_l[X_l]) \preceq u$ and $\alpha(q_l[X_l - \{x_l, x'_l\}]) \preceq u$. We then conclude using Equation (9.1) that:

$$\alpha(s) = \alpha(s') = \alpha(q[X_l])u^\omega\alpha(r'[X_l])$$

Now suppose that X_l do not describe a valid piece of q_l. We depict this situation in 9.9.

If $x_l \in p$ then so is x'_l. Therefore we have:

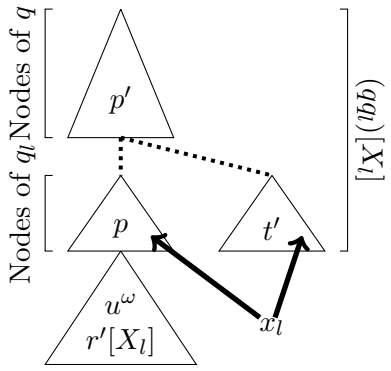


Figure 9.9: X_l does not describe a valid piece of q_l in Case 1 of Lemma 9.17

$$\begin{aligned}\alpha(s) &= (\alpha(p') \diamond \alpha(t')) \cdot \alpha(p) \cdot u^\omega \cdot \alpha(r'[X_l]) \\ \alpha(s') &= (\alpha(p') \diamond \alpha(t')) \cdot \alpha(p[X_l - \{x_l, x_l'\}]) \cdot u^\omega \cdot \alpha(r'[X_l])\end{aligned}$$

By definition $\alpha(p) \preceq \alpha(q_l) = u$. Using Equation (9.1) we get $\alpha(s') = \alpha(s)$. If x_l is in t' then so is x_l'. Therefore we have:

$$\begin{aligned}\alpha(s) &= (\alpha(p') \diamond \alpha(t')) \cdot \alpha(p) \cdot u^\omega \cdot \alpha(r'[X_l]) \\ \alpha(s') &= (\alpha(p') \diamond \alpha(t'[X_l - \{x_l, x_l'\}])) \cdot \alpha(p) \cdot u^\omega \cdot \alpha(r'[X_l])\end{aligned}$$

By definition there exists some $w \in W_2$ such that $(w \diamond \alpha(t')) \preceq \alpha(q_l) = u$. Therefore Equation (9.3) yields $\alpha(s') = \alpha(s)$. This concludes this case.

Case 2: Top-down tame fractal. Consider $r = qq_1q_2 \cdots q_lr'$, the decomposition of the fractal. Since $x_l, x_l' \in X_l$ is a good decomposition, by choice of l and by monochromaticity, we get:

$$\begin{aligned}\alpha(s) &= \alpha(r[X_l]) &&= \alpha(q[X_l]) \cdot u^\omega \cdot \alpha((q_lr')[X^l]) \\ \alpha(s') &= \alpha(r[X_l - \{x_l, x_l'\}]) &&= \alpha(q[X_l]) \cdot u^\omega \cdot \alpha((q_lr')[X_l - \{x_l, x_l'\}])\end{aligned}$$

As in the previous case, we assume first that if X_l describes a valid piece of q_l. By monochromaticity we have $\alpha(q_l) = u$. Therefore $\alpha(q_l[X_l]) \preceq u$ and $\alpha(q_l[X_l - \{x_l, x_l'\}]) \preceq u$. We then conclude using Equation (9.1) that:

$$\alpha(s) = \alpha(s') = \alpha(q[X_l])u^\omega\alpha(r'[X_l])$$

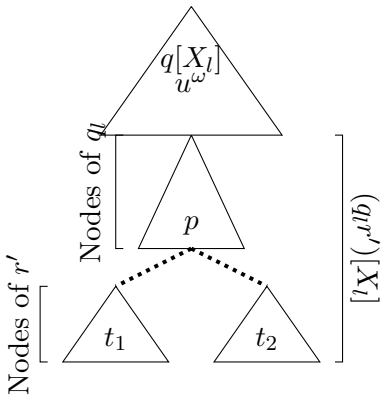


Figure 9.10: X_l does not describe a valid piece of q_l in Case 2 of Lemma 9.17

If X_l does not describe a valid piece of q_l we depict the situation in Figure 9.10. We have:

$$\begin{array}{rcl} \alpha(s) & = & \alpha(q[X_l]) \cdot u^\omega \cdot \alpha(p)[X^l]) \cdot (\alpha(t_1[X_l]), \alpha(t_2[X_l])) \\ \alpha(s') & = & \alpha(q[X_l]) \cdot u^\omega \cdot \alpha(p)[X_l - \{x_l, x_l'\}]) \cdot (\alpha(t_1[X_l]), \alpha(t_2[X_l])) \end{array}$$

By definition there exists some $h \in H$ such that $(\alpha(p) \diamond h) \preceq \alpha(q_l) = u$. Therefore Equation (9.2) yields $\alpha(s) = \alpha(s')$. This concludes this case.

Case 3: Scattered top-down tame fractal. Consider $r = qq_1q_2 \cdots q_l q' \tilde{q}_l \ldots \tilde{q}_1 r'$, the decomposition of the fractal. We have:

$$\begin{array}{rcl} \alpha(s) & = & \alpha(q[X_l]) \cdot u^\omega \cdot \alpha((q_l q' \tilde{q}_l)[X^l]) \cdot \tilde{u}^\omega \cdot \alpha(r'[X_l]) \\ \alpha(s') & = & \alpha(q[X_l]) \cdot u^\omega \cdot \alpha((q_l q' \tilde{q}_l)[X_l - \{x_l, x_l'\}]) \cdot \tilde{u}^\omega \cdot \alpha(r'[X_l]) \end{array}$$

This is done using the same arguments as Case 1 for the bottom-part and the same arguments as Case 2 for the top part. We conclude that $\alpha(s') = \alpha(s)$. □

9.3.2 Third Item of Proposition 9.6

The proof is very close to the one we provided for the second case of Proposition 9.6. The difference is contained in the new definitions for good decompositions and tame fractals. The sequence of Lemmas is identical and the proofs are also done using the same arguments. In this section, we provide the adapted definition for the notion of fractal.

Fix p a context, t_1, t_2 two trees, $a, a' \in A_2$ and $c \in A_0$ as described in the third item of Proposition 9.6. We write $\alpha(t_1) = h$, $\alpha(t_2) = g$ and $v = \alpha(p)$. We write:

$$\begin{aligned} s &= p \cdot a' \cdot (a \cdot (t_1, t_2), c) \\ s' &= p \cdot a' \cdot (t_1, t_2) \end{aligned}$$

Good Decomposition We define a new notion of good decomposition that corresponds to this case. Consider some tree r and X a set of nodes of r, if X describes a valid piece of r we write $r[X]$ that piece. In that case, we say that $x, x' \in X$ is a *good decomposition* of r if:

- x has label a.
- In $r[X]$, x' is a sibling of x and is a leaf labeled with c.
- In $r[X]$, the children of x have types h and g.
- In $r[X]$, the 2-context obtained by putting the port at x and its sibling has type $v\alpha(a')$

Fractals The definition is identical to the previous case. A *fractal* is a sequence of good decompositions. More formally, consider a sequence of good decompositions $x_1, x'_1 \in X_1 \cdots x_l, x'_l \in X_l$. We say the sequence $x_1, x'_1 \in X_1 \cdots x_l, x'_l \in X_l$ is a fractal of length l iff:

$$X_i \subseteq X_{i+1} - \{x_{i+1}, x'_{i+1}\}$$

Tame Fractals The definition of tame fractals is symmetric with the one we gave for the second case of Proposition 9.6. Let r be a tree and consider $x_1 \in X_1 \cdots x_n \in X_n$ a fractal of length n inside r.

Top-down tame fractal We say this fractal is a *top-down tame fractal* iff there exists $n + 1$ contexts $q, q_1, \ldots, q_n$ and a tree r' such that:

- $r = qq_1 \cdots q_n$.
- For all i, x_i, x'_i are in q_i.
- For $i < i'$, $X_{i'}$ describes a valid piece of q_i.

Bottom-up tame fractal We say this fractal is a *bottom-up tame fractal* iff there exists $n+1$ contexts $q, q_1, \ldots, q_n$ and a tree r' such that:

- $r = qq_n \cdots q_1 r'$.
- For all i, x_i, x'_i are in q_i.
- For $i < i'$, $X_{i'}$ describes a valid piece of q_i.

Scattered bottom-up tame fractal We say this fractal is a *scattered bottom-up tame fractal* iff there exists $2n+2$ contexts $q, q', q_1, \ldots, q_n, \tilde{q}_1, \ldots, \tilde{q}_n$ and a tree r' such that:

- $r = q\tilde{q}_1 \cdots \tilde{q}_n q' q_n \cdots q_1 r'$.
- For all i, x_i is in q_i.
- For all i, x'_i is in $\tilde{q}_i$.
- For $i < i'$, $X_{i'}$ describes a valid piece of q_i and $\tilde{q}_i$.

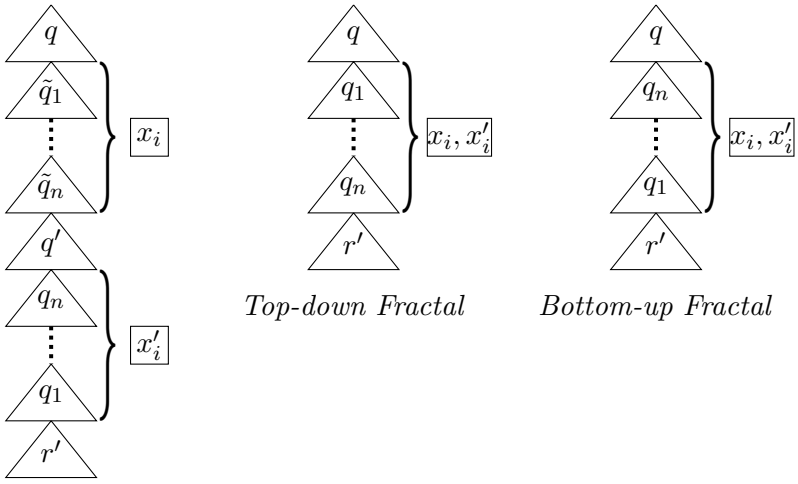


Figure 9.11: Tame Fractals

The rest of the proof is identical to the one of the second item. We use the sane sequence of lemmas and the same proofs.

9.3.3 First Item of Proposition 9.6

This case is the simplest one. It has the closest definitions to the unranked setting. In this section, we provide the adapted definitions for the notion of fractal.

Fix p a context, $b \in A_1$ and t a tree as described in the first item of Proposition 9.6. We write $\alpha(t) = h$ and $v = \alpha(p)$. We write:

$$\begin{aligned} s &= p \cdot b \cdot t \\ s' &= p \cdot t \end{aligned}$$

Good Decomposition We define a new notion of good decomposition that corresponds to this case. Consider some tree r and X a set of nodes of r, if X describes a valid piece of r we write $r[X]$ that piece. In that case, we say that $x \in X$ is a *good decomposition* of r if:

- x has label b.
- In $r[X]$, the subtree at the child of x has type h.
- In $r[X]$, the context obtained by putting the port at x has type v.

Fractals The definition is identical to the previous case. A *fractal* is a sequence of good decompositions. More formally, consider a sequence of good decompositions $x_1 \in X_1 \cdots x_l \in X_l$. We say the sequence $x_1 \in X_1 \cdots x_l \in X_l$ is a fractal of length l iff:

$$X_i \subseteq X_{i+1} - \{x_{i+1}\}$$

Tame Fractals The definition of tame fractals is simpler to the ones for the second and third items. Since we have less nodes to manipulate, we do not need the scattered fractal category. Let r be a tree and consider $x_1 \in X_1 \cdots x_n \in X_n$ a fractal of length n inside r.

Top-down tame fractal We say this fractal is a *top-down tame fractal* iff there exists $n+1$ contexts $q, q_1, \ldots, q_n$ and a tree r' such that:

- $r = qq_1 \cdots q_n$.
- For all i, x_i is in q_i.
- For $i < i'$, $X_{i'}$ describes a valid piece of q_i.

Bottom-up tame fractal We say this fractal is a *bottom-up tame fractal* iff there exists $n+1$ contexts $q, q_1, \ldots, q_n$ and a tree r' such that:

- $r = qq_n \cdots q_1 r'$.
- For all i, x_i is in q_i.
- For $i < i'$, $X_{i'}$ describes a valid piece of q_i.

The rest of the proof is identical to the one of the second item. We use the sane sequence of lemmas and the same proofs.

9.4 Discussion

Our characterization in Theorem 9.1 is restricted to trees of rank 2. For trees of rank k for some arbitrary k, we conjecture the following theorem:

Conjecture 9.18. *Fix L a regular language of trees of rank k. L is definable in BC-$\Sigma_1(<_{\mathbf{v}})$ iff its syntactic k-algebra $(H, V, W_2, \ldots, W_k)$ verifies for all $u, u', v \in V$, all $w, w' \in W_2$ and all $\tilde{w} \in W_3$:*

$$u^\omega v = u^\omega = v u^\omega \quad v \preceq u \tag{9.1}$$

$$u^\omega w = u^\omega w' \quad \exists h \in H \ (w \diamond h), (w' \diamond h) \preceq u \tag{9.2}$$

$$(w \diamond h) u^\omega = (w \diamond h') u^\omega \quad \exists w' \in W_2 \ (w' \diamond h), (w' \diamond h) \preceq u \tag{9.3}$$

$$\tilde{w} \cdot (u^\omega h, g, u'^\omega h') = \tilde{w} \cdot (u^\omega h, g', u'^\omega h') \quad \begin{array}{c} \exists w \in W_2 \ (w \diamond g) \preceq u \\ \exists w' \in W_2 \ (w' \diamond g') \preceq u' \end{array} \tag{9.4}$$

Notice that this characterization uses the same equations as Theorem 9.1 and a new equation, Equation (9.4). Equation (9.4) is specific to the trees of rank k for $k \geq 3$ since it uses an element of W_3. The main problem for proving Conjecture 9.18 would to prove an adapted version of Proposition 9.6 which we state below:

Conjecture 9.19. *There exists an integer $n \in \mathbb{N}$ such that for: (the construction is depicted in Figure 9.12):*

- $1 \leq \iota \leq \mu \leq k$
- $0 \leq \iota_1, \iota_2 \leq i$, $0 \leq \mu_1, \mu_2 \leq j$ *such that* $\iota_1 + \iota_2 = \iota$ *and* $\mu_1 + \mu_2 + 1 = \mu$, $\iota_2 = \mu_1 + 1$
- $a \in A_j$, $b \in A_i$

- $a_1, \ldots, a_{\mu_1}, b_1, \ldots, b_{\iota_1} \in A_0$
- $t_1, \ldots, t_{\mu_2 - 1}, s_1, \ldots, s_{\iota_2}$ *some trees*
- p *a context*

If we write $t = pa(b(b_1, \ldots, b_{\iota_1}, s_1, \ldots, s_{\iota_2}), a_1, \ldots, a_{\mu_1}, t_1, \ldots, t_{\mu_2})$ *and* $s = pa(s_1, \ldots, s_{\iota_2}, t_1, \ldots, t_{\mu_2})$*, we have:*

$$t \cong_n s \implies \alpha(t) = \alpha(s)$$

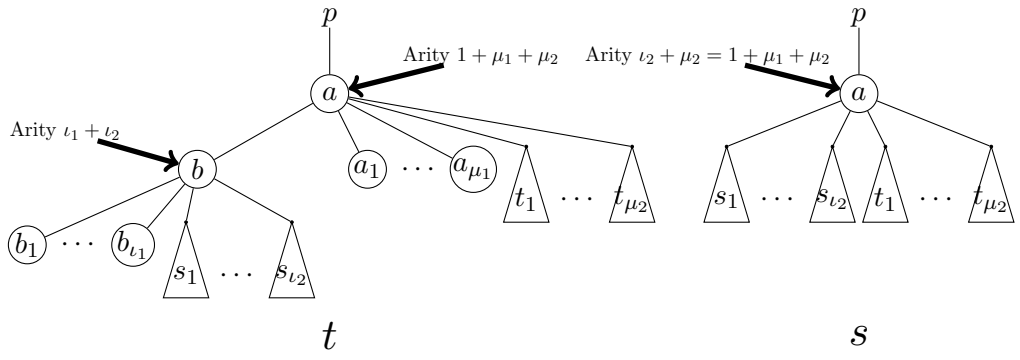


Figure 9.12: Illustration of the trees s and t in Conjecture 9.19

The problem for proving Conjecture 9.19 is that it involves much more nodes than Proposition 9.6. In particular this leads to a definition of good decomposition that is much more complicated. As well as a notion of tame fractal which involves more categories. This makes the proof of Lemma 9.12 much more technical. Finally one of the categories of tame fractal involves using Equation (9.4) in the proof of Lemma 9.16.

Chapter 10

First Order Logic with Two Variables over Unranked Trees

In this chapter, our main goal is to provide a characterization for the logic $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ on forests. We briefly recall the definition of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$, more details can be found in Chapter 6. $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ is the restriction of $\mathrm{FO}(<_{\mathbf{h}}, <_{\mathbf{v}})$ to only two variables that may be reused. It uses the predicates $<_{\mathbf{v}}$ for the ancestor relation and $<_{\mathbf{h}}$ for the following sibling relation. It is stated in Theorem 6.2 that $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ has the same expressive power as the temporal logic $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$. $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ navigates in a forest using two "vertical" modalities, one for going to some ancestor node (F^{-1}) and one for going to some descendant node (EF), and two "horizontal" modalities for going to some following sibling ($\mathrm{F}_{\mathbf{h}}$) or some preceding sibling ($\mathrm{F}_{\mathbf{h}}^{-1}$). We will use Theorem 6.2 throughout the chapter and consider $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ or $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ formulas depending on convenience.

Recall that in Chapter 3, we presented a decidable characterization for the restriction of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ to the setting of words:

Theorem 3.2. ([TW98]) *A regular word language L over an alphabet A is definable in $F + F^{-1}$ iff its syntactic monoid M verifies, for all $u, v \in M$:*

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \tag{2.2}$$

Also, in Chapter 6 we quoted a decidable characterization for the temporal logic $EF + F^{-1}$ on forests. The logic $EF + F^{-1}$ is less expressive than $\mathrm{FO}^2(<_{\mathbf{v}})$ which is the restriction of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ to the vertical predicate $<_{\mathbf{v}}$. This characterization uses forest algebras and a specific relation $\dashv$ that we defined in Chapter 6. The relation $\dashv$ is defined as follows: from some context, a smaller context can be built by deleting the subtrees hanging on the path leading from the root to the port.

Theorem 6.3. ([Boj07b]) *Fix L a regular forest language. L is definable in $EF + F^{-1}$ iff its syntactic forest algebra using monoids verifies:*

$$h + g = g + h \quad \forall h, g \in H \tag{6.1}$$

$$h + h = h \quad \forall h \in H \tag{6.2}$$

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \quad \forall u, v \in V \tag{2.2}$$

$$\begin{gathered} \forall u_1, u_2, v_1, v_2 \in V \quad \textit{such that } u_1 \dashv u_2 \textit{ and } v_1 \dashv v_2 \\ (u_1 v_1)^\omega (u_2 v_2)^\omega = (u_1 v_1)^\omega u_1 v_2 (u_2 v_2)^\omega \end{gathered} \tag{6.3}$$

Our characterization shares some similarities with these two results. However, it is also very different. A first difference would be that for $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$, we will need to consider forest algebras using semigroups rather than forest algebras using monoids. Recall that forest algebras using semigroups are restricted to strict contexts (see Chapter 5 for more details).

Not surprisingly, Equation (2.2) is still part of our characterization of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. It states states both the horizontal and vertical semigroups H and V of the syntactic forest algebra of the language must satisfy Equation (2.2). However, because we consider forest algebras using semigroups, the vertical statement of Equation (2.2) is less powerful than for Theorem 6.3 since it is restricted to strict contexts.

The other equations used in Theorem 6.3 are no longer true for languages definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. because of the order $<_{\mathbf{h}}$, languages definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ are not closed under bisimulation. Therefore neither Equation (6.1) nor Equation (6.2) are true. Finally we give an example of language that is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ but does not verify (6.3). Consider the following $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ formula:

$$\varphi = \forall x \ (\neg(\exists y \ (P_a(y) \wedge x <_{\mathbf{h}} y)) \vee \exists y \ (x <_{\mathbf{v}} y \wedge \exists x \ (y <_{\mathbf{h}} x \wedge P_b(x))))$$

The formula φ recognizes the language L of forests such that all nodes that have a following sibling labeled with a, have an ancestor which has a following sibling labeled with b. Consider the contexts $c_1 = c \cdot \Box, c_2 = c \cdot \Box + b$ and $d_1 = c \cdot \Box, d_2 = c \cdot \Box + a$. For all integer k, the $(c_1 d_1)^k (c_2 d_2)^k a \in L$ while $(c_1 d_1)^k c_1 d_2 (c_2 d_2)^k a \notin L$. Therefore L does not verify (6.3).

Besides using (2.2) on the two semigroups, our characterization uses a third extra property. The statement of (2.2) on H can be seen as a reflection of the horizontal expressive power of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$, and the statement of (2.2) on V

as a reflection of its vertical expressive power. Our third property mixes both expressive powers at the same time. We call it *closure under saturation* and we do not know yet whether it is implied by the previous identities or not.

It is immediate from the word case that being definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ implies that the vertical and horizontal semigroups of the syntactic forest algebra satisfy the required identity. That closure under saturation is also necessary is proved via a classical, but tedious, Ehrenfeucht-Fraïssé game argument. As usual, the main difficulty is to show that the closure conditions are sufficient. In order to do so, as it is standard when dealing with FO^2 (see Chapters 3 and 8), we introduce a notion of reachability relations for comparing elements of the syntactic algebra. However, in our case, we parametrize these relations with a set of forbidden patterns: the contexts authorized for going from one type to another type cannot use any of the forbidden patterns. We are then able to perform an induction using this set of forbidden patterns, thus refining our comparison relations more and more until they become trivial.

In Section 10.1 we define several notions that will be involved in both the statement and the proof of our characterization. In particular we give a definition of the notion of patterns we consider and of the reachability order we use. In Section 10.2, we state our characterization and discuss its relation with Theorems 3.2 and 6.3. Sections 10.3 and 10.4 are devoted to the proof of our characterization. In Section 10.3 we prove that closure under saturation is a necessary property of languages definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. In Section 10.4, we prove that any language verifying the characterization is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$.

It turns out that our proof technique applies for various horizontal predicates. In the Section 10.5 we show how to adapt the characterization obtained for $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ in order to obtain decidable characterizations for $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$, $\mathrm{FO}^2(s, <_{\mathbf{v}})$ and $EF + F^{-1}(S)$. Finally, in Section 10.6, we discuss the decidability of closure under saturation. Note that we do not provide a full proof for the decidability of saturation as this is ongoing work.

Note that all forest algebras we consider in this chapter are forest algebras using semigroups. Also all contexts we consider are strict. Recall that this means that the port of the context has no sibling.

10.1 Preliminaries

In this section we define the main ingredient necessary to our characterization of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. In the first part we describe how we will handle the horizontal behavior of the logic in our proof. In the second part we define orders over H and V that will play a key role in our characterization. We assume fixed a language L recognized by a forest algebra (H, V) via a morphism α.

10.1.1 Horizontal behavior and k-SMTypes.

As mentioned in the introduction we will constantly be working with sequences of sibling nodes. Actually we will be working with the sequence of their labels. For technical reasons, we also include to the label of a node the label of its child if it has a unique child that is a leaf. We now make this notion more precise.

Shallow Multicontexts A multicontext is defined as for context but has several ports. The *arity* of a multicontext is the number of its ports. A multicontext is said to be *shallow* if each of its trees is either a single leaf a, a single node with a port below $b(\Box)$ or, a tree of the form $b(a)$ where b is a node and a a leaf (see Figure 10.1). Let x be a node of a forest s. Let $x_1, \cdots, x_l$ be the sequence of siblings of x, including x. Let $t_1, \cdots, t_l$ be the subtrees of s rooted at those nodes. The *shallow multicontext of x in s* is the sequence $p_1, \cdots, p_l$ such that $p_i := a$ if $t_i = a$, $p_i := b(a)$ if $t_i = b(a)$, and $p_i := b(\Box)$ otherwise. A shallow multicontext c *occurs* in a forest s iff there exists a node x of s such that c is the shallow multicontext of x in s. Given a shallow multicontext c of arity n and a sequence T of n forests, $c[T]$ denotes the forest obtained after placing each tree in the sequence T at their corresponding place in the sequence of ports of c.

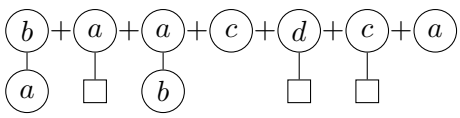


Figure 10.1: Illustration of a typical shallow multicontext

k-SMTypes As expected we will only manipulate shallow multicontexts modulo $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ definability. Intuitively, $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ treats a shallow multicontext as a string whose letters are either a, $b(a)$, or $b(\Box)$. More formally, consider the alphabet A_+ containing the letters a for all $a \in A$, the letters $b(\Box)$ for all $b \in A$ and the letters $b(a)$ for all $a, b \in A$. We see a shallow multicontext p as a string over the alphabet A_+. Now consider $\mathrm{FO}^2(<)$, the first order logic restricted to two variables on strings using the predicate $<$ for the following position relation. For each positive integer k and any two shallow multicontexts p and p', we write $p \equiv_k p'$ for the fact that p and p' seen as words over A_+ agree on all sentences of $\mathrm{FO}^2(<)$ of quantifier rank k. We denote by k-SMTypes the equivalence classes of this relation. Notice $\equiv_k$ has finite index and that each equivalence class of $\equiv_k$ is definable in $\mathrm{FO}^2(<)$.

Let P be a set of k-SMTypes - this set will play the role of forbidden patterns in our proof - a forest t is said to be *P-valid* if no shallow multicontext of a class

in P occur in t. Similarly we define the notion of P-valid context. We extend the notion of P-validity to forest types of H. We say $h \in H$ is P-valid iff there exists a P-valid forest t such that $h = \alpha(t)$. Similarly for $v \in V$. By definition P-validity is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$.

Lemma 10.1. *For any k and any set P of k-SMTypes, the language of P-valid forests is definable in* $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$.

(X,k)-PosTypes We also need to denote specific positions within k-SMTypes, again modulo definability in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. For this we will use the notion of (X,k)-PosTypes. Consider a set $X \subseteq H$, X will later be a parameter in our induction. Again we see a shallow multicontext as a word over the alphabet A_+. For technical reasons, rather than using the full power of $\mathrm{FO}^2(<)$ to describe our positions, we use a weaker logic $\mathrm{FO}^2_X(<))$. The logic $\mathrm{FO}^2_X(<))$ is weaker in testing labels, intuitively, it works like $\mathrm{FO}^2(<)$ on shallow multicontexts but it cannot distinguish the symbol $b(\Box)$ from the symbol $b(a)$ whenever $\alpha(a) \notin X$. More formally, $\mathrm{FO}^2_X(<))$ is limited in the unary predicates it can use to test labels. $\mathrm{FO}^2_X(<))$ can use a for all $a \in A$, $b(a)$ when $\alpha(a) \in X$, or $b(\Box) \vee b(a)$ when $\alpha(a) \notin X$.

Given two nodes x and x' of t we write $x \cong_{k,X} x'$ if the shallow multicontext of x and the shallow multicontext of x' seen as words over A_+ satisfy the same formulas of $\mathrm{FO}^2_X(<))$ of quantifier depth at most k, with one free variable denoting respectively the position x and x'. We only consider classes of of $\cong_{k,X}$ such that $b(\Box)(x)$ holds for some $b \in A$. We denote by (X,k)-PosTypes the equivalence classes of this relation. Notice that $\cong_{k,X}$ has finite index.

Given a (X,k)-PosType δ and a k-SMType τ, we say that δ is *compatible with* τ if all shallow multicontexts $p \in \tau$ contain a position $x \in \delta$.

The following lemma is immediate from the definitions.

Lemma 10.2. *For any k-SMType τ and compatible (X,k)-PosType δ, there is a formula $\psi_{\tau,\delta}$ of $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}^{-1}_{\mathbf{h}})$ such that for any forest s, the set of nodes of s satisfying $\psi_{\tau,\delta}$ is exactly the set of nodes of s in δ and whose shallow multicontext is in τ.*

10.1.2 P-reachability

We define P-reachability, which is an order on elements of H. P-reachability will play a major role as a parameter of our induction.

As before, P denotes a set of k-SMTypes for some k. A forest type $h \in H$ is said to be P-reachable from the forest type h' if there exists a P-valid context type v such that $h = vh'$. Two forest types are P-equivalent if they are mutually P-reachable.

Notice that the notion of P-reachability becomes finer when P increases: if $P \subseteq P'$ then P'-reachable implies P-reachable.

When we have at least one shallow multicontext of arity at least 2 outside of P then P-reachability restricted to P-valid forest types contains a unique maximal class. In that case we say that P is *branching* and we write H_P this maximal class:

Claim 10.3. *If there is a shallow multicontext of arity at least 2 outside of P then there is a unique maximal class regarding P-reachability restricted to P-valid forest types.*

Proof. Take p outside of P and of arity $n \geq 2$. Given $h, h' \in H$ that are P-valid, consider t and t' two P-valid trees such that $\alpha(t) = h$ and $\alpha(t') = h'$. Consider the ordered set T of n P-valid forests containing copies of t and t', with at least one copy of t and one copy of t'. Now $\alpha(p[T])$ is P-reachable from both h and h'. The result follows. □

Similarly we define P-reachability over V. Given two contexts $u, v \in V$ we say that v is *P-reachable* from u whenever there is a P-valid context type c such that $v = u \cdot c$.

10.1.3 Antichain Composition Principle

We shall make use of the following composition lemma. This lemma is essentially taken from [Boj07b]. We reuse notations from [Boj07b].

A formula of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ with one free variable is called *antichain* if in every forest, the set of nodes where it holds forms an antichain, i.e. a set (not necessarily maximal) of nodes pairwise incomparable with respect to the descendant relation and to the following sibling relation. This is a semantic property, and may not be apparent just by looking at the syntax of the formula.

We fix (i) an antichain formula φ, (ii) disjoint tree languages $L_1, \cdots, L_n$ and (iii) leaves of label $a_1, \cdots, a_n$. Given a forest s, we define the forest $s[(L_1, \varphi) \to a_1, \cdots, (L_n, \varphi) \to a_n]$ as follows. For each node x of s such that $s, x \models \varphi(x)$, we determine the unique i such the forest language L_i contains the subforest of x. If such an i exists, we remove the subforest of x (including x and its siblings), and replace it by a leaf labeled with a_i. Since φ is antichain, this can be done simultaneously for all x. Note that the formula φ may also depend on ancestors of x, while the languages L_i only talk about the subtree at x. A simple argument, similar to the one given in [Boj07b] for $EF + F^{-1}$, and restricting the navigation inside or outside suitable definable regions, shows:

Lemma 10.4. *[Antichain Composition Lemma] Let φ, $L_1, \cdots, L_n$ and $a_1, \cdots, a_n$ be as above. If $L_1, \cdots, L_n$ and K are languages definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$, then so is $\{t \mid t[(L_1, \varphi) \to a_1, \cdots, (L_n, \varphi) \to a_n] \in K\}$.*

10.2 Characterization of $\mathbf{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$

In this section we state our main theorem, which is the decidable characterization of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. This characterization is given using two equations over the syntactic forest algebra of the language and a property we call *saturation*. We begin with the definition of saturation.

Saturation Fix some integer k and a language L recognized by a forest algebra (H, V) via a morphism α. We first define the notion of a saturated context. Consider some branching set of forbidden k-SMTypes P. By Claim 10.3 there exists a unique maximal class H_P regarding P-reachability. We write $\bar{H}_p = H - H_P$. Intuitively a context is saturated if it is P-valid and contains one representative for each k-SMType $\tau \notin P$ and compatible $(\bar{H}_P, k)$-PosType. More formally, a context Δ is said to be *P-saturated* if (i) it is P-valid (ii) for each P-valid k-SMType τ, and each each compatible $(\bar{H}_P, k)$-PosType δ, there exists a node x, occurring in the *skeleton* of Δ (i.e. the path from the root of Δ to its port), such that $x \in \delta$ and the shallow multicontext of x in Δ is in τ.

We say that a tree language L is *closed under k-saturation* if for all branching set P of k-SMTypes, for all context Δ that is P-saturated, for all P-valid tree t, for all P-valid shallow multicontext p, for all position x of p and for all sequence T of P-valid forests whose types are in H_P, we have:

$$\alpha(\Delta)^\omega \alpha(t) = \alpha(\Delta)^\omega \alpha(p[T, x]) \alpha(\Delta)^\omega \alpha(t) \tag{10.1}$$

were $p[T, x]$ is the context formed from p by placing the forests of T at the corresponding ports of p except for the port at position x. A language is closed under saturation if it is closed under k-saturation for some k.

The main result.

Theorem 10.5. *A regular forest language L is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ iff its syntactic forest algebra using semigroups (H, V) verifies the following properties:*

a) H verifies the equation

$$\omega(h + g) + g + \omega(h + g) = \omega(h + g) \tag{10.2}$$

b) V verifies the equation

$$(uv)^\omega v (uv)^\omega = (uv)^\omega \tag{10.3}$$

c) L is closed under saturation.

As we already said It turns out that (10.2) and (10.3) above are exactly the identities characterizing, over strings, definability in $\mathrm{FO}^2(<)$. We give this equations new names in chapter so we do not confuse the horizontal one with the vertical one.

Theorem 3.2. ([TW98]) *A regular language L over an alphabet A is definable in $F + F^{-1}$ iff its syntactic monoid M verifies, for all $u, v \in M$:*

$$(uv)^\omega = (uv)^\omega v (uv)^\omega \tag{2.2}$$

Recall that $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ can express the fact that a forest is a tree and, for each k, that a tree has rank k, hence Theorem 10.5 also apply for regular unranked tree languages and regular languages of trees of bounded rank.

There are two directions that need to be proved for obtaining Theorem 10.5. In Section 10.3 we prove that the properties listed in the statement of Theorem 10.5 are necessary for having definability in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ using an Ehrenfeucht-Fraïssé argument. In Section 10.4 we prove the most difficult direction of Theorem 10.5, the properties imply definability in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$.

10.3 Correctness of the Properties

In this section we prove that the properties stated in Theorem 10.5 are necessary for being definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. Meaning that any language L definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ verifies Equations (10.2) and (10.3) and is closed under saturation.

It is simple to see that Equations (10.2) and (10.3) are necessary. The proof is identical to the one used in [TW98] for the string case. We thus concentrate on proving that saturation is necessary using a classical, but tedious, Ehrenfeucht-Fraïssé argument:

Lemma 10.6. *A forest language definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ is closed under saturation.*

Before proving Lemma 10.6, we give some definitions that will play a key role in its proof. We call the *root* of a forest, the root of the leftmost tree in the forest. Recall that the *skeleton* of a context is the path of node that goes from the port to the root that is an ancestor of the port. We define a new notion that we call the *extended skeleton* of a context. The extended skeleton is the set of nodes composed of the skeleton itself and all nodes that are siblings of a node on the skeleton. Both notions are illustrated in Figure 10.2.

The rest of this section is devoted to the proof of Lemma 10.6.

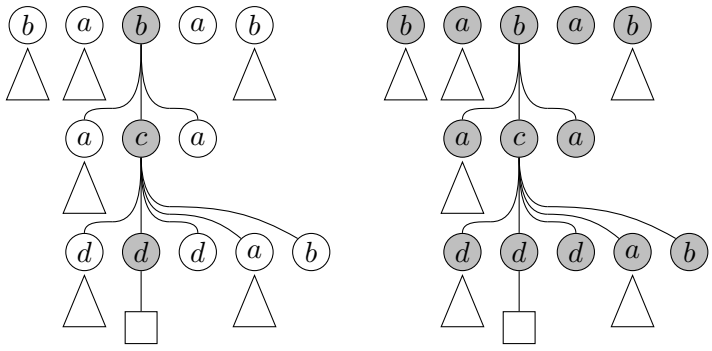


Figure 10.2: Illustration of the notions of skeleton and extended skeleton.

Given Theorem 6.2, we adopt an $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ point of view as the corresponding game is slightly simpler. Assume L is definable in $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ and is recognized by the forest algebra (H, V) via some morphism α. Let k be the nesting depth of the navigational modalities used in the formula recognizing L, we show that L is closed under k-saturation.

The proof is an Ehrenfeucht-Fraïssé argument. The definition of the game corresponding to $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ is standard. There are two players, Duplicator and Spoiler, the board consists in two forests and both players agree on the number of moves in advance. At any time there is one pebble placed on a node of each of the two forests and the corresponding nodes have the same label. At the beginning of the game the two pebbles are placed on the root of the leftmost tree of each forests. At each step Spoiler moves one of the pebble, either to some ancestor of its current position, or to some descendant or to some left or right sibling. Duplicator must respond by moving the other pebble in the same direction to a node of the same label. If Duplicator cannot move then Spoiler wins.

Given P, Δ, p, x, t and T as in the definition of k-saturation, let $u = \alpha(\Delta)$, $h = \alpha(t)$ and $v = \alpha(p[T, x])$.

We exhibit two forests S and S' such that $\alpha(S) = u^\omega h$ and $\alpha(S') = u^\omega v u^\omega h$ and such that Duplicator has a winning strategy for the k-move game described above when playing on S and S'.

A classical argument then shows that this implies that no formula of $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ whose nesting depth of its navigational modalities is less than k can distinguish between the two forests. This implies that $u^\omega h = u^\omega v u^\omega h$ as desired.

Our agenda is as follows. In Section 10.3.1 we define the two forests S and S' on which we will play. Finally in Section 10.3.2 we give the winning strategy for

Duplicator in the k-move game on S and S'.

10.3.1 Definition of the forests S and S'.

By definition of k-saturation P is branching and by Claim 10.3 we let $H_P = \{h_1, \cdots, h_l\}$ be the maximal P-equivalence class. We fix $\{a_1, \ldots, a_l\}$ a set of labels such that for all i, $\alpha(a_i) = h_i$.

Since P is branching, there exists a shallow multicontext q_0 outside of P with arity greater than 2. We denote by V_i the context obtained from q_0 by placing the forest a_i into all the ports of q_0 except for the last one.

Because H_P is the maximal class relative to P-reachability, for each $i \leq l$, there exists a P-valid context U_i' such that $h_i = \alpha(U_i')\alpha(V_l \cdots V_1)u^\omega h$. We write U_i the context obtained from $U_i' \cdot V_l \cdots V_0$ by replacing all maximal subforests of forest type h_i with the letter a_i. For all i we write $u_i = \alpha(U_i)$. The contexts U_i have the two following properties:

- U_i is P-valid,
- for all i, $u_i u^\omega h = h_i$,
- for all i, the context U_i contains all labels $a_1, \ldots, a_l$ as subforests. Therefore it contains at least one subforests for any forest types in H_P.

Recall that by definition of k-saturation, for each shallow multicontext q occurring in U_i and each position x in q that is not a leaf, there exists a q' occurring in Δ such that $(q, x) \cong_k (q', x')$ where x' mark the position in q' occurring in the skeleton of Δ.

We now construct by induction on j contexts Δ_j and $U_{i,j}$, and trees $T_{i,j}$ such that for all i, we have $\alpha(\Delta_j) = u$, $\alpha(U_{i,j}) = u_i$ and $\alpha(T_{i,j}) = h_i$.

We initialize the process by setting for all $i \leq l$:

- $U_{i,0} := U_i$,
- Δ_0 is obtained from Δ by replacing all subforests of type h_i with a_i.
- $T_{i,0} = U_{i,0} \cdot (\Delta_0)^\omega \cdot t$.

By construction we have $\alpha(\Delta_0) = u$, $\alpha(U_{i,0}) = u_i$ and $\alpha(T_{i,0}) = u_i u^\omega h = h_i$ as desired.

For $j > 0$, the inductive step of the construction is done as follows for all $i \leq l$:

- $U_{i,j}$ is formed from U_i by replacing each subforest of type h_ℓ that is a child of a node of the extended skeleton by $T_{\ell,j-1}$ (see Figure 10.3),

- Δ_j is formed from Δ by replacing each subforest of type h_ℓ by $T_{\ell,j-1}$,
- $T_{i,j} = U_{i,j} \cdot \Gamma_j$.

where Γ_j is the forest:

$$\Gamma_j = (\Delta_j)^\omega \cdots (\Delta_0)^\omega \cdot t$$

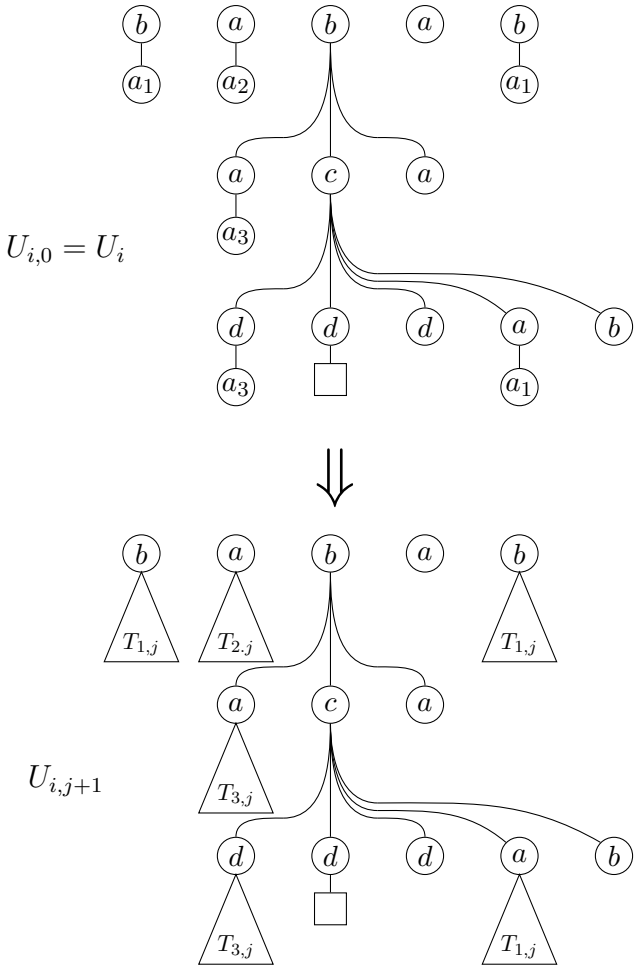


Figure 10.3: Illustration of the construction of $U_{i,j}$ from U_i: each subforest of type h_ℓ in U_i is replaced by $T_{\ell,j-1}$.

By induction we have $\alpha(U_{i,j}) = \alpha(U_i) = u_i$, $\alpha(\Delta_j) = \alpha(\Delta) = u$ and $\alpha(T_{i,j}) = u_i u^\omega h = h_i$ as required. Moreover Δ_j remains P-saturated and each $U_{i,j}$ contains a copy of $T_{\ell,j-1}$ for all $\ell \leq l$.

Let $m = 2^k$ and let P_m be the context formed from $p[T, x]$ by replacing all maximal subforests of type h_i by $T_{i,m-1}$. In particular, each forest of T is replaced by the appropriate $T_{\ell,m}$. The construction is similar to the construction depicted in Figure 10.3.

Finally let:

$$\begin{aligned} S &:= (\Delta_m)^{(m+1)\omega} \cdot P_m \cdot \Gamma_m \\ S' &:= (\Delta_m)^{(m+1)\omega} \cdot \Gamma_m \end{aligned} \tag{10.4}$$

The following claim then conclude the proof of Lemma 10.6.

Claim 10.7. *Duplicator has a winning strategy for the k-move game between S and S'.*

10.3.2 The winning strategy

Proof of Claim 10.7. We give a winning strategy for Duplicator. In order to be able to formulate this strategy we need further definitions.

Given two nodes x and y and a number n we denote by $x \equiv_n^{\mathbf{h}} y$ the fact that Duplicator has a winning strategy in the n-move game played on the words formed by the sequence of labels of the siblings of x and of y, starting from the position x on one side and y on the other side.

The *nesting level* of a node x of S or S' is the minimal number ℓ such that x belongs to a context Δ_ℓ or $U_{i,\ell}$. We set the nesting level of the nodes that in any copy of t to 0. We also set the nesting level for the nodes of P_m that are not in any tree $T_{i,m-1}$ for some $i \leq l$ to m. The notion of nesting level is illustrated in Figure 10.4.

Recall that because of the construction of the context $U_{i,n}$, a node of nesting level n always has, for all $\ell \leq l$, a descendant that is at the root of a forest $T_{\ell,n-1}$.

The *skeleton* of S (or of S') is the path of S (of S') going from its root to the port of each copy of Δ_m.

The *upward number* of a node $x \in S$ (or $x \in S'$) is the number of occurrences of Δ_m in the path from x to the root of S (see Figure 10.5).

We now state a property $\mathcal{P}(n)$ that depends on an integer n, two nodes $x \in S$ and $y \in S'$, possibly two nodes $\hat{x} \in S$ and $\hat{y} \in S'$ that are ancestors of respectively x and y and possibly two nodes $\bar{x} \in S$ and $\bar{y} \in S$ that are on the skeleton.

We then show that when $\mathcal{P}(n+1)$ holds on a game starting at x, y, then Duplicator can play one move while enforcing $\mathcal{P}(n)$. As it is easy to see that $\mathcal{P}(k)$ holds for the leftmost roots of S and S', this will conclude the proof of Claim 10.7.

The inductive property $\mathcal{P}(n)$ states that $\hat{x}$ is defined iff $\hat{y}$ is defined and, whenever they are defined, they have nesting level $\geq n-1$ and their parents have

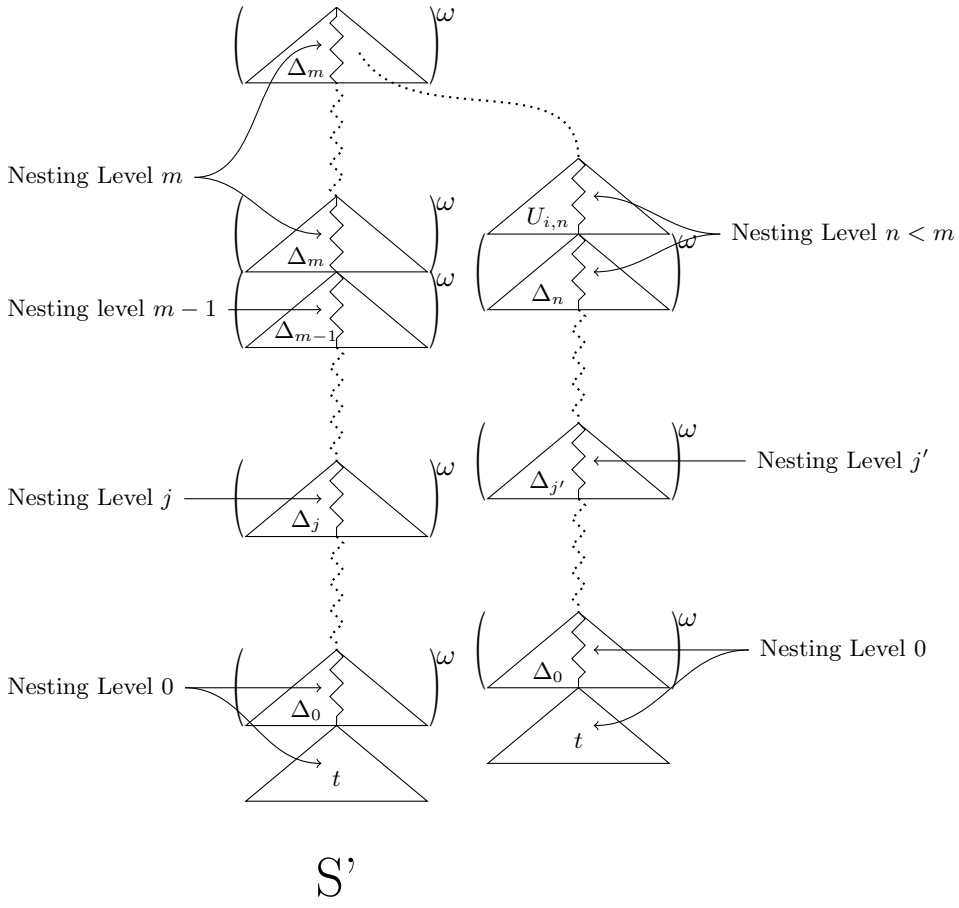


Figure 10.4: Illustration of the notion of nesting level in the proof of Claim 10.7.

nesting level $\geq n$. Also they both have an upward number $> n$. Moreover it requires the disjunction of the following three cases:

1. $\hat{x}$ and $\hat{y}$ are defined. In this case Duplicator has a winning strategy in the n-move game played on the subforest of $\hat{x}$ and the subforest of $\hat{y}$, starting at positions x and y.

2. $\hat{x}$ and $\hat{y}$ are undefined and the upward number of x and y are $\leq n$. In this case x and y are at the same relative position from the root in their respective forest (recall that by construction S and S' are isomorphic up to m copies of Δ_m).

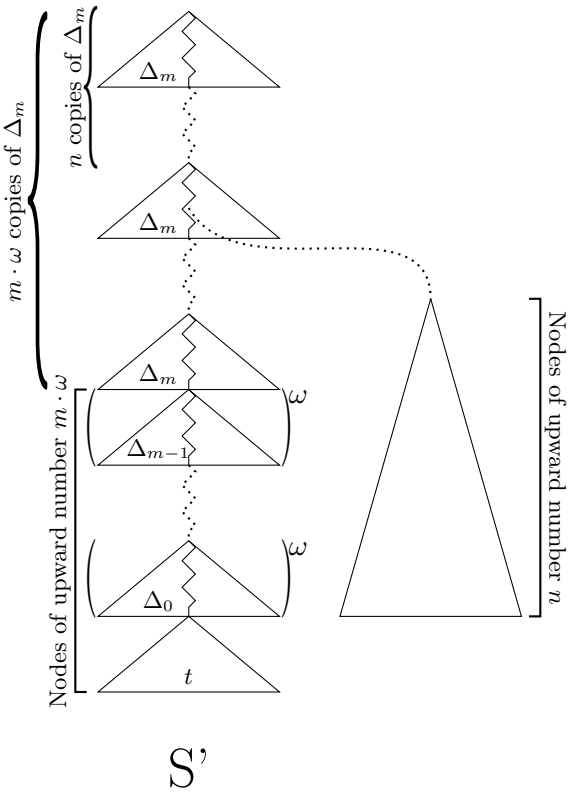


Figure 10.5: Illustration of the notion of upward number in the proof of Claim 10.7.

3. $\hat{x}$ and $\hat{y}$ are undefined, the upward number of x and y is $> n$. In this case x and y are of nesting level $> n$, moreover $x \equiv_n^{\mathbf{h}} y$.

Assume we are in a situation where $\mathcal{P}(n+1)$ holds. We sketch how Duplicator can play while enforcing $\mathcal{P}(n)$. The strategy depends on why $\mathcal{P}(n+1)$ holds.

Case 1

$\hat{x}$, and therefore also $\hat{y}$, are defined.

• If Spoiler moves to a node that is in the subforest of $\hat{x}$ then Duplicator simply respond using the strategy provided by Item (1) of $\mathcal{P}(n+1)$. $\hat{x}$ and $\hat{y}$ remain unchanged.

• We now assume that Spoiler moves to an ancestor x' of $\hat{x}$.

If the upward number of x' is $\leq n$. Notice that the only ancestors of x having a smaller upward number than x are on the skeleton. Therefore x' is on the skeleton.

Remember that, by Item (1) of $\mathcal{P}(n+1)$, $\hat{y}$ has upward number $> (n+1)$. Therefore the copy y' of x' in the other forest, which has upward number $\leq n$, is an ancestor of $\hat{y}$. Duplicator then selects y' as her answer while satisfying Item (2) of $\mathcal{P}(n)$. In this case $\hat{x}$ and $\hat{y}$ now become undefined.

If the upward number of x' is $> n$. By saturation of Δ_m, there is a node z in the skeleton Δ_m such that $x' \equiv_n^{\mathbf{h}} z$. By hypothesis the upward number of y is larger than the upward number of $\hat{y}$ which is $> (n+1)$. Hence we can find above y an occurrence of Δ_m of upward number $> n$. Duplicator answers by the copy of z in this occurrence of Δ_m. By construction x, y have upward numbers $> n$. Moreover they are ancestors of $\hat{x}$ and $\hat{y}$ who had nesting level $> n+1$ by $\mathcal{P}(n+1)$. Hence they have nesting level $> n$ and Item (3) of $\mathcal{P}(n)$ is satisfied. In this case $\hat{x}$ and $\hat{y}$ now become undefined.

Case 2

The upward number of x is $\leq n+1$.

• If Spoiler moves up or horizontally, Duplicator simply copy Spoiler's move and Item (2) of $\mathcal{P}(n)$ remains true if we end up with an upward number $\leq n$ otherwise Item (3) trivially hold. If Spoiler moves down to some node with an upward number $\leq n$, the we are still in the part of the forests that are identical in both S and S' and Duplicator simply copy Spoiler's move. As above, Item (2) of $\mathcal{P}(n)$ is true if we end up with an upward number $\leq n$ otherwise Item (3) trivially hold.

• Assume now that Spoiler moves to some descendant x' of x that has upward number equal to $n+1$. Recall that x and y are at the same relative position from the root in their respective forest. Therefore there exists a node y' below y that is at the same position as x'. Duplicator chooses y' as her answer. If x', y' are of nesting level $> n$ then Item (3) trivially hold. If x', y' are of nesting level $\leq n$, they are both in a tree $T_{i,m-1}$ for some i at an isomorphic position. We choose $\hat{x}$ and $\hat{y}$ as the roots of the copies of each tree $T_{i,m-1}$ in their respective forests. $\mathcal{P}(n)$ holds for Item (1).

• Assume now that Spoiler moves to some descendant x' of x that has upward number $> n+1$. Notice that the only ancestors of x' that have upward number smaller than $n+1$ are on the skeleton. Therefore x, which has upward number $\leq n+1$, is on the skeleton and has nesting level m. By hypothesis x and y are at the same relative position from the root in their respective forest. Therefore y also has nesting level m. We distinguish between two cases depending on the nesting level of x'. Intuitively, if x' has small nesting level we will conclude by enforcing Item (1) and if x' has large nesting level we will conclude by enforcing Item (3).

If x' has nesting level $> n$. By saturation of Δ_m, there is node z in the skeleton

of Δ_m such that $x' \equiv_n^{\mathbf{h}} z$. By hypothesis the nesting level of y is $m > (n+1)$. Hence we can find below y an occurrence of Δ_j with $j > n$. Duplicator answers by the copy of z in this occurrence of Δ_j and Item (3) of $\mathcal{P}(n)$ is satisfied. In this case $\hat{x}$ and $\hat{y}$ remain undefined.

If x' has nesting level $\leq n$. By definition, x' is inside a forest $T_{i,n}$ for some i such that $0 \leq i \leq l$. We set $\hat{x}$ as the root of this forest. By hypothesis y is of nesting level $m > (n+1)$. Recall that this implies that for all ℓ such that $0 \leq \ell \leq l$ y has a tree $T_{\ell,n}$ as descendant. In particular y has a forest $T_{i,n}$ as descendant. We set $\hat{y}$ as the root of this forest. Since the subforests of $\hat{x}$ and $\hat{y}$ are identical, Duplicator answers in the subforest of $\hat{y}$ at an isomorphic position y' to the position x' in the subforest of $\hat{x}$. By definition $\hat{x}$ and $\hat{y}$ are of nesting level n and Duplicator has a winning strategy (which consists in playing the isomorphism) in the n-move game played on the subforest of $\hat{x}$ and the subforest of $\hat{y}$, and starting at positions x' and y' . Hence Item (1) of $\mathcal{P}(n)$ is satisfied.

Case 3

The upward number of x is $> (n+1)$.

• If Spoiler moves horizontally, Duplicator moves according to the winning strategy provided by $\equiv_{(n+1)}^{\mathbf{h}}$ and Item (3) of $\mathcal{P}(n)$ remains true.

• If Spoiler moves up to some node x'.

If the upward number of x' is $\leq n$, then Duplicator answers by the copy of x', y', in the other forest while satisfying Item (2) of $\mathcal{P}(n)$. Note that the upward number of y is $> n$. Therefore y' of upward number $\leq n$ is indeed an ancestor of y. In this case $\hat{x}$ and $\hat{y}$ remain undefined.

If the upward number of x' is $> n$. By saturation of Δ_m, there is a node z in the skeleton of Δ_m such that $x' \equiv_n^{\mathbf{h}} z$. By hypothesis the upward number of y is $> n+1$. Hence we can find above y an occurrence of Δ_m of upward number $n+1$. Duplicator answers by the copy of z in this occurrence of Δ_m and Item (3) of $\mathcal{P}(n)$ is satisfied. In this case $\hat{x}$ and $\hat{y}$ remain undefined.

• If Spoiler moves down to some node x'.

If x' has nesting level $> n$. By saturation of Δ_{n+1}, there is a node z in the skeleton of Δ_{n+1} such that $x' \equiv_n^{\mathbf{h}} z$. By Item (3) the nesting level of $y > (n+1)$. Hence we can find below y an occurrence of Δ_{n+1}. Duplicator answers by the copy of z in this occurrence of Δ_{n+1} and Item (3) of $\mathcal{P}(n)$ is satisfied. In this case $\hat{x}$ and $\hat{y}$ remain undefined.

If x' has nesting level $\leq n$. By construction, x' is inside a forest $T_{i,n}$ for some i such that $0 \leq i \leq l$. We set $\hat{x}$ as the root of this forest. By hypothesis y is of nesting level $> (n+1)$. Recall that this implies that for all ℓ such that $0 \leq \ell \leq l$ y has a forest $T_{\ell,n}$ as descendant. In particular y has a forest $T_{i,n}$ as descendant. We set $\hat{y}$ as the root of this forest. Since the subforests of $\hat{x}$ and $\hat{y}$ are identical,

Duplicator answers in the subforest of $\hat{y}$ at an isomorphic position y' to the position x' in the subforest of $\hat{x}$. By definition $\hat{x}$ and $\hat{y}$ are of nesting level n and Duplicator has a winning strategy (which consists as playing the isomorphism) in the n-move game played on the subforest of $\hat{x}$ and the subforest of $\hat{y}$, and starting at positions x' and y' . Item (1) of $\mathcal{P}(n)$ is satisfied.

This concludes the proof of Claim 10.7 and therefore the proof of Lemma 10.6.

10.4 Sufficientness of the Properties

In all this section we fix a regular forest language L that is recognized by a forest algebra (H, V) via a morphism α. We assume that L is closed under saturation and that H and V verify Equations (10.2) and (10.3). We will show that L is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$, concluding the proof of Theorem 10.5.

We assume that for each forest type $h \in H$ there is a tree consisting of a single leaf node that has h for forest type via α. This simplifies the notations in the proof with no harm in the generality of the result.

As mentioned before, we will work with k-SMTypes for some fixed integer k. We start by defining a suitable k.

Lemma 10.8. *There exists a number k'' such that whenever p and p' are shallow multicontext with the same k''-SMType then for all forest s we have $\alpha(p[\bar{s}]) = \alpha(p'[\bar{s}])$, where $p[\bar{s}]$ is the forest constructed from p by placing s at each port of p.*

Proof. This is a consequence of Theorem 3.2 and the fact that H satisfies Equation (10.2). Consider words over H as alphabet and the natural morphism $\beta : H^+ \to H$. Since H verifies (10.2), which is (2.2) stated with additive notations, it follows from Theorem 3.2 that for every $h \in H$, $\beta^{-1}(h)$ is definable using a formula of $\mathrm{FO}^2(<)$. We choose k'' as the maximal rank of all these formulas.

Take p and p' with the same k''-SMType and s some forest. Let $t_1, \ldots, t_n$ be the sequence of trees occurring in $p[\bar{s}]$ and $t'_1, \ldots, t'_{n'}$ be the sequence of trees occurring in $p'[\bar{s}]$. For all i let $h_i = \alpha(t_i)$ and $h'_i = \alpha(t'_i)$. As $p \equiv_{k''} p'$ the words $h_1 \ldots h_n$ and $h'_1 \ldots h'_{n'}$ satisfy the same formulas of $\mathrm{FO}^2(<)$ of rank k'' over the alphabet H. By our choice of k'' it follows that $\beta(h_1 \ldots h_n) = \beta(h'_1 \ldots h'_{n'})$. Therefore $\alpha(p[\bar{s}]) = \alpha(p'[\bar{s}])$. □

As L is closed under saturation, there is an integer k' such that L is closed under k'-saturation. Take k as the maximum between k' and k''. Notice that L remains closed under k-saturation and that the conclusion of Lemma 10.8 remains true when replacing k'' by the bigger number k.

The proof of Theorem 10.5 is done by induction using an inductive hypothesis that is stated in the proposition below. One of the parameters is a subset X of H. The following definition is adapted from [Boj07b]. We already used a similar

notion in Chapter 8 for the proof of the charcaterization of $EF + F^{-1}$ in the setting of trees of bounded rank. A forest s is said to be *X-trimmed* if the only subforests of s that have a forest type in X are single leaves. We say that a forest language L is *definable modulo X* if there is a definable forest language L' that agrees with L over X-trimmed forests. For each $h \in H$, $v \in V^1$ and each set of k-SMTypes let $L^P_{v,h} = \{t \mid v \cdot \alpha(t) = h \text{ and } t \text{ is } P\text{-valid}\}$.

Our goal in this section is to show that:

Proposition 10.9. *$\forall h \in H, v \in V^1$ and $X \subseteq H$, and P a set of k-SMTypes, $L^P_{v,h}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ modulo X.*

Notice that in the statement of Proposition 10.9 we use V^1 instead of V. The added neutral element is called the *empty context type*. We can then complete the proof of Theorem 10.5 by applying Proposition 10.9 for all $h \in \alpha(L)$ with v the empty context type, and P, X empty sets.

In the rest of this section we only care about P-valid forests and hence we implicitly ignore the forest types $h \in H$ such that $\alpha^{-1}(h)$ contains no P-valid forests.

In the rest of this section we prove Proposition 10.9, we work by induction on three parameters that we now define.

We say that u is at distance one from u' if u is P-reachable from u', u' is not P-reachable from u and, for any context u'' that is P-reachable from u', u is also P-reachable from u''. The transitive closure of this relation defines the distance between contexts. The *P-depth* of v is then the distance between v and the empty context type.

We next define an order on sets of k-SMTypes. For each k-SMType τ, its *X-number* is the number of (X, k)-PosTypes compatible with τ. For each set P of k-SMTypes the *n-index of P* is the number of k-SMTypes in P of X-number n. The *X-index of P* is then the sequence of its n-indexes ordered by decreasing n. We write $P_1 <_X P_2$ if the X-index of P_1 is strictly smaller than the X-index of P_2.

We work by induction on the three following parameters, given below in their order of importance:

- $|X|$
- the X-index of P
- the P-depth of v

We consider three main cases: In the first case we suppose that all shallow multicontexts that are not in P have arity 0 or 1. In this case we will show that we can treat our forests as words and Proposition 10.9 follows from known results

over words. Therefore as soon as we are not in the first case we denote by H_P the unique maximal class relative to P-reachability as guaranteed by Claim 10.3.

Our second case assumes that there exists a P-valid forest whose forest type is neither in X nor in H_P. In this case we will conclude by induction either by adding forest types in X or forbidden patterns in P while increasing its X-index.

In the remaining case, $H \setminus X$ is reduced to H_P on P-valid forests. We will then show that we can increase the X-index of P, or increase the P-depth of v or make use of closure under saturation of L to show that v must be constant and hence $L^P_{v,h}$ is trivially definable.

10.4.1 Case 1: All shallow multicontexts outside of P have arity 0 or 1

We show that in this case we can treat our forests as words and use the known results on words. Any P-valid forest t is therefore of the form:

$$c_1 \cdots c_k s$$

where k is possible 0 and the $c_1, \cdots, c_k$ are P-valid shallow multicontexts of arity 1 and s a P-valid shallow multicontext of arity 0. For each $u \in V^1$ and $g \in H$, consider the languages:

$$M_{u,g} = \{t \quad | \quad t = c_1 \cdots c_k \cdot s \text{ is } P\text{-valid}, \\ \alpha(c_1 ... c_k) = u, \text{ and } \alpha(s) = g\}$$

Notice that $L^P_{v,h}$ is the union of those languages where $vug = h$. We show that for any u and g, $M_{u,g}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ modulo X. This will conclude this case.

Let $\{\tau_1, ..., \tau_n\}$ be the set of k-SMTypes not in P of arity 1. From Lemma 10.8 it follows that any two contexts of type τ_i have the same image in V by α. Let $\{v_1, ..., v_n\}$ be those context types. Let $\Gamma = \{d_1, ..., d_n\}$ be a word alphabet and define a morphism $\beta : \Gamma^* \to V$ by $\beta(d_i) = v_i$.

Since V verifies Equation (10.3), for each $v \in V$ there is a $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ formula φ_v such that the words of Γ^* satisfying φ_v are the words of type v under β. From each such formula φ_v we construct an $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ formula Ψ_v by replacing atomic formula $P_{d_i}(x)$ with a formula that tests if the k-SMType at position x is τ_i (recall that by Lemma 10.2 this is expressible in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$). Since s contains only leaves or trees of depth one, we can also easily express in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ that $\alpha(s) = g$ by just investigating the labels occurring in s. By putting all this together we get that $M_{u,g}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ modulo X.

This proves Proposition 10.9 for this case. In the rest of this section we assume the existence of a k-SMType of arity 2 outside of P and therefore, by Claim 10.3, the existence of a unique maximal P-reachable class H_P.

10.4.2 Case 2: There exists a P-valid forest whose forest type is neither in X nor in H_P.

Let t be such a P-valid forest. Fix a class G of mutually P-reachable forest types such that the forest type of t is reachable from any forest type of G, $G \not\subseteq X$, and G is P-minimal with the previous two properties. In other words G is just above X according to P-reachability, and is not in H_P by hypothesis.

Our agenda for this case is as follows. First, we show that being a subforest whose forest type is in G can be detected in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ as it only depends on the presence or absence of certain k-SMTypes. Note that our hypothesis then guarantees that there exists at least one k-SMType whose presence forces that the corresponding forest has a forest type outside G.

Then, intuitively, we split the forest into two parts: one containing all the nodes inside a subforest whose forest type is in G and another one containing all the other nodes. As G is minimal there is no node of the second kind that is a descendant of a node of the first kind. For the first part of the forest, we can add to P the k-SMTypes that are forbidden for having a forest type in G and use our induction hypothesis in order to get an $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ formula giving the precise forest type of G of a forest having a forest type in G. For the second part of the forest we can add G in X and use our induction hypothesis in order to get an $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ formula describing this part. We then conclude using the Antichain Composition Principle. This case illustrated in Figure 10.6.

We start by showing that membership in G can be detected in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$.

Lemma 10.10. *There is a formula $\varphi(x) \in \mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ such that for any P-valid and X-trimmed tree t the set of nodes x such that the subforest of x has forest type in G is exactly the set of nodes at which φ holds.*

Proof. This lemma is proved using Equations (10.2) and (10.3). We show that a subforest has a forest type in G iff it does not contain certain k-SMTypes. Since we can detect those forbidden k-SMTypes using a formula of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$, the result will follow. The proof relies on the following claim:

Claim 10.11. *Take a shallow multicontext p of arity n and take two sequences T and T' of n P-valid forests of forest type in G. We have:*

$$\alpha(p[T]) \in G \;\Leftrightarrow\; \alpha(p[T']) \in G$$

Proof. We use Equation (10.3) to prove this claim. We write $T = (t_1, ..., t_n)$ and $T' = (t'_1, ..., t'_n)$. For $i \in [1, n]$ we write c_i the context obtained from $p[T']$ by replacing t'_i by a port and t'_j by t_j for $j > i$. Notice that by hypothesis on p, T and T', c_i is P-valid. We write $u_i = \alpha(c_i)$, and show that:

$$u_i\alpha(t_i) \in G \;\Leftrightarrow\; u_i\alpha(t'_i) \in G \tag{10.5}$$

Let $g = \alpha(t_i)$ and $g' = \alpha(t'_i)$ and suppose that $u_i g \in G$, we show that $u_i g' \in G$. By symmetry this will prove (10.5). As G is closed under mutual P-reachability, it is enough to show that $u_i g'$ is mutually P-reachable from g'. By definition $u_i g'$ is P-reachable from g'. Therefore it remains to show that g' is P-reachable from $u_i g'$. From $u_i g \in G$ we get that g' is P-reachable from $u_i g$. As g and g' are both in G they are mutually P-reachable. Therefore we have two P-valid contexts c and c' such that $g' = \alpha(c)u_i g$ and $g = \alpha(c')g'$. A little bit of algebra and Equation (10.3) yields:

$$
\begin{aligned}
g' &= \alpha(c)u_i\alpha(c')g' \\
g' &= (\alpha(c)u_i\alpha(c'))^{\omega+1}g' \\
g' &= \alpha(c)u_i(\alpha(c')\alpha(c)u_i)^{\omega}\alpha(c')g' \\
g' &= \alpha(c)u_i(\alpha(c')\alpha(c)u_i)^{\omega}\alpha(c)u_i(\alpha(c')\alpha(c)u_i)^{\omega}\alpha(c')g' \quad \text{using Equation (10.3)} \\
g' &= (\alpha(c)u_i\alpha(c'))^{\omega}\alpha(c)u_i(\alpha(c)u_i\alpha(c'))^{\omega+1}g' \\
g' &= (\alpha(c)u_i\alpha(c'))^{\omega}\alpha(c)u_i g' \\
g' &= \alpha((cc_ic')^{\omega}c)u_i g'
\end{aligned}
$$

as cc_ic' is P-valid, g' is P-reachable from $u_i g'$ and (10.5) is proved.

For concluding the proof of the claim, notice that by construction $\alpha(p[T]) = u_1\alpha(t_1)$. From Equation (10.5) we obtain $u_i\alpha(t_i) \in G$ iff $u_i\alpha(t'_i) \in G$. Notice that by construction $u_i\alpha(t'_i) = u_{i+1}\alpha(t_{i+1})$. Now, again from Equation (10.5), we have $u_{i+1}\alpha(t_{i+1}) \in G$ iff $u_{i+1}\alpha(t'_{i+1}) \in G$. Altogether this gives $u_i\alpha(t_i) \in G$ iff $u_{i+1}\alpha(t_{i+1}) \in G$. Finally by construction we also have $u_n\alpha(t'_n) = \alpha(p[T'])$. By putting all this together we obtain $\alpha(p[T]) \in G$ iff $\alpha(p[T']) \in G$ as desired. □

A shallow multicontext p of arity n is said to be H-good if for some sequence T of n P-valid forests of forest type in G we have $\alpha(p[T]) \in G$. From the previous claim we know that this definition does not depend on the choice of T. A shallow multicontext p that is not H-good is said to be H-bad. It turns out that this distinction between good and bad shallow multicontexts characterizes membership in G.

Claim 10.12. *Let t be a P-valid X-trimmed forest. Then we have $\alpha(t) \in G$ iff t contains only H-good shallow multicontexts.*

Proof. Suppose that $\alpha(t) \notin G$, we show that t contains an H-bad shallow multicontext. Let s be a subforest of t such that $\alpha(s) \notin G$ and $s = p[T]$ where p is a shallow multicontext and T a sequence of forests of forest type in G (possibly empty if p is of arity 0). The existence of such a subforest s is ensured by the fact that $\alpha(t) \notin G$, that G is P-minimal and that t is X-trimmed. By Claim 10.11 p is an H-bad shallow multicontext and it is contained in t. □

It follows from this claim that in order to check whether a subforest is of forest type in G, it is sufficient to check whether it contains an H-bad shallow multicontexts or not. It remains to show that this can be expressed in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. For this we show that the set of H-good shallow multicontexts is a union of k-SMTypes.

Claim 10.13. *Let p and p' be two shallow multicontexts of the same k-SMType. Then we have p is H-good iff p' is H-good.*

Proof. Suppose that p is H-good and of arity n. We show that p' is H-good. Let n' be the arity of p', by Claim 10.11 it is sufficient to prove that there exists a sequence of n' forests T' of forest type in G such that $\alpha(p'[T']) \in G$. Let t be a forest such that $\alpha(t) \in G$ and T be the sequence of n copies of t and T' the sequence of n' copies of t. As $p \equiv_k p'$, because $k \geq k''$, by Lemma 10.8 we get $\alpha(p'[T']) = \alpha(p[T])$. Since p is H-good, $\alpha(p[T]) \in G$, therefore $\alpha(p'[T']) \in G$. □

This last claim concluded the proof of Lemma 10.10. □

We now aim at applying Lemma 10.4, the antichain formula being essentially the one given by Lemma 10.10. The next two lemmas show that the appropriate languages are definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$.

Lemma 10.14. *$L^P_{v,h}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ modulo $X \cup G$.*

Proof. By induction on $|X|$ in Proposition 10.9 we get that $L^{\emptyset}_{v,h}$ is definable modulo $X \cup G$. Notice that if a language of P-valid forests is definable modulo X it is also definable modulo $X \cup G$. By combining the two we get that $L^P_{v,h}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ modulo $X \cup G$. □

Lemma 10.15. *For any $g \in G$, $L^P_{v,g}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ modulo X.*

Proof. Let P' be the set of H-bad k-SMTypes (recall the definition in the proof of Lemma 10.10). Because G is not H_P, there exists at least a H-bad k-SMType and hence P' is not empty. We also know from the proof of Lemma 10.10 that forests that have a forest type in G do not contain any k-SMTypes in P'. Therefore for any $g \in G$, $L^P_{v,g} = L^{(P \cup P')}_{v,g}$. Notice that the X-index of $P \cup P'$ is strictly higher than the X-index of P. Hence, by induction on the X-index of P in Proposition 10.9, $L^{(P \cup P')}_{v,g}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. □

We are now ready to give the final argument which is depicted in Figure 10.6. Let φ be the formula which holds at a node x of a tree t iff x is in L_G, there is no node between the root of t and x in L_G and x has no left sibling. From Lemma 10.10, φ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and by definition it is an antichain formula. By Lemma 10.14, there exists a language K definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$

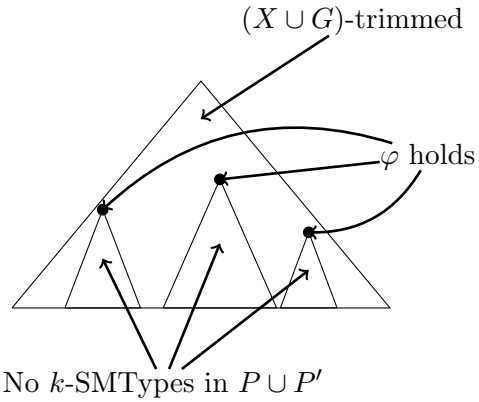


Figure 10.6: Illustration of the Antichain Composition Lemma for Case 2. The marked nodes are the topmost nodes in G and P' is the set of H-bad k-SMTypes

that agrees with $L^P_{v,h}$ on $(X \cup G)$-trimmed forests. Assume $G = \{g_1, \cdots, g_l\}$. For any $i \leq l$, let a_i be a leaf node such that $\alpha(a_i) = g_i$. By Lemma 10.15 for any $i \leq l$, there exists a language L_i definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ that agrees with L^P_{v,g_i} over X-trimmed forests. Hence from the Antichain Composition Lemma, Lemma 10.4, we have that $K' = \{t \mid t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k] \in K\}$ is also definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. By definition of K and the L_i, K' agrees with $L^P_{v,h}$ on X-trimmed and hence $L^P_{v,h}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ modulo X. This concludes the proof of Proposition 10.9 for this case.

10.4.3 Case 3: $H \setminus X$ is reduced to H_P on P-valid forests

A context type u *P-preserves* v if v is P-reachable from vu. A context c *P-preserves* v if $\alpha(c)$ preserves v.

We then distinguish two subcases. In the first subcase we assume that there is a k-SMType $\tau \notin P$ and a compatible (X, k)-PosType δ such that no matter what forests we place in the shallow multicontexts of τ, leaving a port at a position in δ, the resulting context does not *P-preserve* v. In this subcase we will split again the forest into two parts, conclude on each part by induction and combine everything together using the Antichain Composition Principle.

In the remaining subcase, we will use closure under saturation to conclude that $L^P_{v,h}$ is trivial.

Formally, we say that a k-SMType τ is *P-bad for* v if $\tau \notin P$ and there exists a compatible (X, k)-PosType δ such that for any shallow multicontext $p \in \tau$ of arity n, any position x of p in δ and any sequence T of $n-1$ forests of forest type in H_p, the context $p[T, x]$ does not P-preserve v.

We distinguish two subcases.

Subcase 1: There exists a k-SMType τ which is P-bad for v.

We fix a $\tau \notin P$ of maximal X-number that is P-bad for a (X,k)-PosType δ. Let $p \in \tau$ and x a position in p of type δ.

Given two elements h and h' of H, we say that h is v^+-equivalent to h' if for all context type u P-reachable from v such that v is not P-reachable from u (hence the P-depth of u is strictly higher that the P-depth of v) we have $uh = uh'$.

Lemma 10.16. *Each v^+-equivalence class is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ modulo X.*

Proof. This is immediate by induction on the P-depth of v in Proposition 10.9. □

Intuitively, we want to approximate the subtree below a v-bad position by its v^+-equivalence class. When doing this we may reintroduce shallow multicontexts that were forbidden by P. But fortunately the X-index of P will increase when doing so.

Let $p \in \tau$. Let $x_1, \cdots, x_l$ be all the positions of p of (X,k)-PosType δ. Let $b(\Box)$ be the label of all the x_i in p. Let $\hat{P}$ be the set of all the shallow multicontexts constructed from p by replacing at all the positions x_i, $b(\Box)$ by $b(a_i)$, for some arbitrary choice of $a_i \notin X$. Let Δ be the set of k-SMTypes τ' of all the shallow multicontexts in $\hat{P}$. Let P' be $(P \cup \{\tau\}) \setminus \Delta$.

Lemma 10.17. *The set $L^{P'}_{v,h}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ modulo X.*

Proof. We show that $P' > P$, the result follows by induction on the X-index of P in Proposition 10.9.

More precisely, we show that any $\tau' \in \Delta$ is of X-number strictly smaller than the X-number of τ. This gives the desired result.

By definition of Δ there exists $p \in \tau$ and $p' \in \tau'$ such that p' can be obtained from p by replacing $b(\Box)$ with subtrees of the form $b(a)$ where $\alpha(a) \notin X$. Consider a position x' of p' of label $b'(\Box)$. By construction the corresponding position x of p has the same label. By the definition of the logic used for defining (X,k)-PosTypes, which cannot distinguish $b(\Box)$ from $b(a)$ if $\alpha(a) \notin X$, x and x' must have the same (X,k)-PosType. Hence any (X,k)-PosType compatible with τ' is also compatible with τ. Moreover, by construction of Δ, δ is no longer compatible with τ'. As τ had a maximal X-number, $P' > P$.

□

Based on the above lemmas, we conclude this case of Proposition 10.9 as follows. Consider the property that holds at a node y of a tree t if the k-SMType of the shallow multicontext at y is in τ and its (X,k)-PosType in δ and there is no node between the root of t and y satisfying this property. By Lemma 10.1 this

property is expressible by a formula $\varphi(y)$ of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and it is antichain by definition. We also know that each such position y has the same label, say b.

Let $\gamma_1, \cdots, \gamma_k$ be all the equivalence classes of the v^+-equivalence relation. For each such class γ_i, consider the set of trees $\{b(t) \mid t \in \gamma_i\}$. Thanks to Lemma 10.16, for each such set there exists L_i definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ that agrees with it on X-trimmed trees. For any $i = 1, \cdots, k$, let h_i be an arbitrarily chosen forest type in the class γ_i, and let a_i be a leaf label whose forest type is h_i.

By Lemma 10.17, there exists K definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ that agrees with $L^{P'}_{v,h}$ on X-trimmed trees. Hence we can apply the Antichain Composition Lemma (see Figure 10.7) and have that $\{t \mid t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k] \in K\}$ is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$.

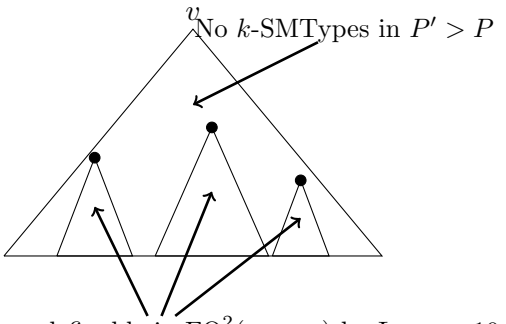


Figure 10.7: Illustration of the Antichain Composition Lemma for Subcase 1. The marked nodes are the topmost nodes whose shallow multicontext are in τ.

We conclude by showing that $L^P_{v,h} = \{t \mid t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k] \in K\}$ over X-trimmed trees. It follows that $L^P_{v,h}$ is definable modulo X. This is a simple consequence of the following two lemmas.

Lemma 10.18. *For any P-valid X-trimmed tree t, $t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k]$ is P'-valid.*

Proof. This follows from the construction of P' and the definition of φ. □

Lemma 10.19. *For any X-trimmed tree t, $v\alpha(t) = v\alpha(t[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k])$.*

Proof. The proof goes by induction on the number of occurrences of τ in t and the number of nodes y of (X, k)-PosType δ in each occurrence p of τ. If there is no occurrence of τ, this is immediate as the substitution does nothing.

Consider a node y of a shallow multicontext p such that $p \in \tau$ and y is in δ and no node above y satisfies that property. Let b be the label of y as specified by δ.

Let s be the subforest below y in t and i such that $\alpha(s) \in \gamma_i$. Let c be the context formed from t by placing a port at y. Let d the context formed from c by removing all the strict ancestors of y. By the definition of P-bad and the choice of y, $\alpha(d)$ does not P-preserve v. We write t' the tree constructed from t by replacing the subtree at y by $b(a_i)$. By construction $t'[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k]$ is $ca_i[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k]$. By induction hypothesis we have that $v\alpha(t') = v\alpha(t'[(L_1, \varphi) \to a_1, \cdots, (L_k, \varphi) \to a_k])$. Therefore it remains to show that $v\alpha(t') = v\alpha(cs)$. We first claim that v is not P-reachable from $v\alpha(cd)$. This is a consequence of Equation (10.3), suppose that v is P-reachable from $v\alpha(cd)$, then there exists a P-valid u such that $v = v\alpha(cd)u$. From there we get the following sequence of equalities:

$$\begin{aligned}
&v = v\alpha(cd)u \\
&v = v(\alpha(c)\alpha(d)u)^\omega \\
&v = v(\alpha(c)\alpha(d)u)^\omega\alpha(d)(\alpha(c)\alpha(d)u)^\omega \qquad \text{using (10.3)} \\
&v = v\alpha(d)(\alpha(c)\alpha(d)u)^\omega
\end{aligned}$$

This implies that $\alpha(d)$ P-preserves v, which we know to be false. Let then $u = v\alpha(cd)$. From the above v is not P-reachable from u but, as t is P-valid, u is P-reachable from v. Hence $u\alpha(s) = u\alpha(a_i)$ by definition of v^+-equivalence. This implies the desired result. □

Subcase 2: There is no k-SMType τ which is P-bad for v.

Using closure under saturation, we show that in this case, v is P-preserved by a context that is constant over P-valid trees. This implies that $L^P_{v,h}$ contains no P-valid trees or all of them and is therefore definable in $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$.

By hypothesis, for each $\tau \notin P$ and each compatible (X, k)-PosType δ, there exists a shallow multicontext $p \in \tau$ and a position $x \in \delta$ of p such that there exists a sequence T of P-valid X-trimmed forests such that the context $p[T, x]$, P-preserves v. For each pair (τ, δ), we fix such a context $p[T, x]$. Let Δ be the context defined as the concatenation of all those contexts. By construction, Δ^ω is P-valid and P-preserves v. By construction Δ^ω is also saturated.

Using closure under saturation we show that Δ^ω is constant on P-valid trees. Let h_1 and h_2 be two elements of H_P and t_1, t_2 be two forests such that $\alpha(t_1) = h_1$ and $\alpha(t_2) = h_2$. We want to show that $\alpha(\Delta)^\omega h_1 = \alpha(\Delta)^\omega h_2$.

Consider a P-valid shallow multicontext p of arity at least 2, and two positions x, y of p and a sequence T of P-valid forests. Let T be an arbitrary sequence of P-valid forests with types in H_P. Let $p[T, x, y]$ be the multicontext of arity 2 constructed from p by placing the two ports in x and y and placing the forests of T for the other ports. Let $p^+[T, x]$ be the context constructed from $p[T, x, y]$ by

placing $\Delta^\omega t_2$ at the port denoted by y. Let $p^+[T, y]$ be the context constructed from $p[T, x, y]$ by placing $\Delta^\omega t_1$ at the port denoted by x.

Then we have:

$$\begin{aligned}
\alpha(\Delta)^\omega h_1 &= \alpha(\Delta)^\omega \alpha(p^+[T, x])\alpha(\Delta)^\omega h_1 && \text{using (10.1)} \\
&= \alpha(\Delta)^\omega \alpha(p^+[T, y])\alpha(\Delta)^\omega h_2 && \text{by construction} \\
&= \alpha(\Delta)^\omega h_2 && \text{using (10.1)}
\end{aligned}$$

And we are done with the last case.

10.5 Other Logics

Using the same proof structure we can obtain the decidability of several other logics that differ with $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ only in the horizontal modalities. This is done by adapting Equation (10.2) restricting H, together with the notion of k-SMType and the notion of (X, k)-PosType, and therefore the notion of saturation. We briefly give the definitions of these logics, see Chapter 6 for more details. We illustrate this with the predicates $\mathrm{S}^{\neq}$, S, $\mathrm{X}_{\mathbf{h}}$ and $\mathrm{X}_{\mathbf{h}}^{-1}$ but we believe that other modalities could be considered, assuming the induced logic over words has a decidable characterization.

The predicate $\mathrm{S}^{\neq}\varphi$ holds at x if φ holds at some sibling of x distinct from x. It is a shorthand for $\mathrm{F}_{\mathbf{h}}\varphi \vee \mathrm{F}_{\mathbf{h}}^{-1}\varphi$. The predicate $\mathrm{S}\varphi$ holds at x if φ holds at some sibling of x including x. It is a shorthand for $\varphi \vee \mathrm{S}^{\neq}\varphi$. The predicates $\mathrm{X}_{\mathbf{h}}$ and $\mathrm{X}_{\mathbf{h}}^{-1}$ are the usual next sibling and previous sibling modalities.

We provide decidable characterizations for three logics using these modalities. The first one, $EF + F^{-1}(S)$, is the weakest one and cannot be equivalently defined as a simple fragment of first order logic like $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. This is because languages defined by a formula of $EF + F^{-1}(S)$ are closed under bisimulation. Fragments of first order logic allow quantification over incomparable nodes which rules out closure under bisimulation.

The second one, $EF + F^{-1}(S^{\neq})$, can be equivalently defined using a first order formalism. Consider the binary predicate $s(x, y)$ that holds when x and y are siblings. In terms of expressive power $EF + F^{-1}(S^{\neq})$ is equivalent to $\mathrm{FO}^2(s, <_{\mathbf{v}})$, the two variable fragment of first order logic using the predicates s, and $<_{\mathbf{v}}$, see Theorem 6.2.

Finally, we consider $EF + F^{-1}(\mathrm{X}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}, \mathrm{X}_{\mathbf{h}}^{-1}, \mathrm{F}_{\mathbf{h}}^{-1})$, which is an extension of the logic $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$. This logic can be equivalently defined using a first order formalism. Consider the binary predicate $\mathrm{Succ}_{\mathbf{h}}(x, y)$ that holds when y is the next sibling of x. In terms of expressive power $EF + F^{-1}(\mathrm{X}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}, \mathrm{X}_{\mathbf{h}}^{-1}, \mathrm{F}_{\mathbf{h}}^{-1})$ is equivalent to $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$, the two variable fragment of first order logic using the predicates $\mathrm{Succ}_{\mathbf{v}}$, $<_{\mathbf{v}}$, and $<_{\mathbf{v}}$, see Theorem 6.2.

The three characterizations require Equation (10.3) for V as before. They also require that H satisfy the known characterization of the fragment of LTL induced by the horizontal modalities. Moreover they require closure under saturation of the language with a notion of saturation modified in order to use the appropriate notion k-SMType and (X, k)-PosType relative to the horizontal expressive power of the logic.

For each of three logics mentioned above, we first define an adapted notion of k-SMType and (X, k)-PosType such that Lemma 10.1, Lemma 10.2 holds for the logic under investigation. This in particular yields a new notion of saturation. Then we state the characterization of the logic. Notice then that aside from the definition of the integer k through Lemma 10.8 and the Antichain Composition Principle, the proof of Theorem 10.5 is fully parametrized by the notions of k-SMType and (X, k)-PosType. As the Antichain Composition Principle is rather straightforward, we will therefore not give a full proof for these new characterizations and only focus on the choice of the integer k.

We begin with $EF + F^{-1}(S)$ and $EF + F^{-1}(S^{\neq})$. Because the horizontal expressive power of these logics is very weak, the main proofs for these two logics are actually much simpler than for $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ as we will see that there is no need to parametrize the notions of k-SMType and (X, k)-PosType with an integer k.

10.5.1 $EF + F^{-1}(S)$

$EF + F^{-1}(S)$ is the weakest logic we consider. As we said before it cannot be defined as a two variable fragment of first order logic.

We begin with the definition of the new notions of k-SMType and (X, k)-PosType. As we did for $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$, we see a shallow multicontext as a string whose letters are either a, $b(a)$, or $b(\Box)$.

SMTypes[S] The straightforward way to adapt the notion of k-SMType would be to say that for any two shallow multicontexts p and p', we have $p \equiv_k p'$ iff p and p' agree on all sentences of $EF + F^{-1}(S)$ of nesting depth of modalities k. But horizontally $EF + F^{-1}(S)$ can only check the set of trees appearing in the shallow multicontext. Therefore, for any $k \geq 1$, $\equiv_k$ is the same as $\equiv_1$ and there is no need to parametrize the notion of SMType[S] with an integer k. We denote by SMTypes[S] the equivalence classes of the relation $\equiv_1$. We immediately have:

Lemma 10.20. *For any set P of SMTypes[S], the language of P-valid forests is definable in $EF + F^{-1}(S)$.*

X-PosTypes[S] Let $X \subseteq H$. Recall that the notion of (X, k)-PosTypes was defined in order to be able to point position within k-SMTypes. However since

there is no order on siblings due to the weakness of $EF + F^{-1}(S)$, two positions with the same label in a SMType[S] are indistinguishable. Therefore we do not need to parametrize the notion of X-PosType[S] with an integer k. Consider x a node in some forest and let b be its label. The X-PosType[S] of a node x is b if x is a leaf, $b(a)$ is x has a single leaf child labeled with a such that $\alpha(a) \in X$ and $b(\Box)$ otherwise[1]. Altogether we have by definition:

Lemma 10.21. *For any SMType[S] τ and compatible X-PosType[S] δ, there is a formula $\psi_{\tau,\delta}$ of $EF + F^{-1}(S)$ such that for any forest s, the set of nodes of s satisfying $\psi_{\tau,\delta}$ is exactly the set of nodes of s in δ and whose shallow multicontext is in τ.*

Recall the definition of saturation given in Section 10.2. The notion of S-saturation is obtained identically after replacing k-SMType with SMType[S] and (X, k)-PosType with X-PosType[S].

These new definitions yield the appropriate characterization.

Theorem 10.22. *A regular forest language L is definable in $EF + F^{-1}(S)$ iff its syntactic forest algebra (H, V) verifies:*

a) H verifies the equations

$$2h = h \text{ and } f + g = g + f \tag{10.6}$$

b) V verifies Equation (10.3)

$$(uv)^\omega v (uv)^\omega = (uv)^\omega$$

c) L is closed under S-saturation.

Aside from the initial choice of the integer k which is no longer necessary here, the proof is identical[2] to the one we gave for Theorem 10.5. Lemma 10.8 is replaced by the following result:

Lemma 10.23. *Take p and p' that have the same SMType[S] then for all forest s we have $\alpha(p[\bar{s}]) = \alpha(p'[\bar{s}])$, where $p[\bar{s}]$ is the forest constructed from p by placing s at each port of p.*

[1]Note that in this case, it is actually possible to simplify the proof and remove all parameters on X-PosTypes[S] by fixing $X = H$

[2]Notice that the first case of the proof is much simpler. Since the logic is closed under bisimulation, for every shallow multicontext of arity 1 outside of P there exists a shallow multicontext of arity 2 with the same SMType[S]. Therefore since P is a union of SMType[S], if there exists a shallow multicontext of arity 1 outside of P then there also exists a shallow multicontext of arity 2 outside of P. It follows that Case 1 can be reformulated: All shallow multicontexts outside of P have arity 0.

Proof. Since p and p' that have the same SMType[S] the forests $p[\bar{s}]$ and $p'[\bar{s}]$ contain the same trees but possibly with a different number of occurrences. It follows from (10.6) that $\alpha(p[\bar{s}]) = \alpha(p'[\bar{s}])$. □

10.5.2 $EF + F^{-1}(S^{\neq})$

As said before $EF + F^{-1}(S^{\neq})$ has the same expressive power than $\mathrm{FO}^2(s, <_\mathbf{v})$.

SMTypes[$\mathrm{S}^{\neq}$] Like in the previous case the straightforward way to adapt the notion of k-SMType would be to say that for any two shallow multicontexts p and p', we have $p \equiv_k p'$ iff p and p' agree on all sentences of $EF + F^{-1}(S^{\neq})$ of modal depth k. Now horizontally $EF + F^{-1}(S^{\neq})$ can only check the number of each tree that appears in the shallow multicontext up to threshold 2. Therefore, for any $k \geq 2$, $\equiv_k$ is the same as $\equiv_2$ and there is no need to parametrize the notion of SMType[$\mathrm{S}^{\neq}$] with an integer k. We denote by SMTypes[$\mathrm{S}^{\neq}$] the equivalence classes of the relation $\equiv_2$.

Lemma 10.24. *For any set P of SMTypes[$S^{\neq}$], the language of P-valid forests is definable in $EF + F^{-1}(S^{\neq})$.*

X-PosTypes[$\mathrm{S}^{\neq}$] Let $X \subseteq H$. As before, to the expressive weakness of $EF + F^{-1}(S^{\neq})$, two positions with the same label in a SMType[$\mathrm{S}^{\neq}$] are indistinguishable. Therefore we do not need to parametrize the notion of X-PosType[$\mathrm{S}^{\neq}$] with an integer k. Consider x a node in some forest and let b be its label. The X-PosType[$\mathrm{S}^{\neq}$] of a node x is b if x is a leaf, $b(a)$ is x has a single leaf child labeled with a such that $\alpha(a) \in X$ and $b(\Box)$ otherwise[3].

Lemma 10.25. *For any SMType[$S^{\neq}$] τ and compatible X-PosType[$S^{\neq}$] δ, there is a formula $\psi_{\tau,\delta}$ of $EF + F^{-1}(S^{\neq})$ such that for any forest s, the set of nodes of s satisfying $\psi_{\tau,\delta}$ is exactly the set of nodes of s in δ and whose shallow multicontext is in τ.*

As in the previous case, replacing these notion in the definition of saturation yields a new notion of saturation that we call $S^{\neq}$-saturation. We now have:

Theorem 10.26. *A regular forest language L is definable in $EF + F^{-1}(S^{\neq})$ iff its syntactic forest algebra (H, V) verifies:*

a) H verifies the equations

$$3h = 2h \text{ and } f + g = g + f \tag{10.7}$$

[3]Again, it is actually possible to simplify the proof in this case and remove all parameters on X-PosTypes[$\mathrm{S}^{\neq}$] by fixing $X = H$

b) V verifies Equation (10.3)

$$(uv)^\omega v (uv)^\omega = (uv)^\omega$$

c) L is closed under $S^{\neq}$-saturation.

Aside from the initial choice of the integer k which is no longer necessary here, the proof is identical to the one we gave for Theorem 10.5. Lemma 10.8 is replaced by the following result:

Lemma 10.27. *Take p and p' that have the same SMType[$S^{\neq}$] then for all forest s we have $\alpha(p[\bar{s}]) = \alpha(p'[\bar{s}])$, where $p[\bar{s}]$ is the forest constructed from p by placing s at each port of p.*

Proof. Since p and p' that have the same SMType[$\mathrm{S}^{\neq}$] the forests $p[\bar{s}]$ and $p'[\bar{s}]$ contain the same trees with the same number of occurrences up to threshold 2. It follows from (10.7) that $\alpha(p[\bar{s}]) = \alpha(p'[\bar{s}])$. □

10.5.3 $\mathbf{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$

The characterization for $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ is very close to our characterization of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. Indeed, $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ is more expressive than $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ and we can no longer simplify the notion of k-SMType and (X,k)-PosType as we did in the previous two cases.

k-SMTypes[$\mathbf{X_h}$] Consider $\mathrm{FO}^2(<, Succ)$, the first order logic restricted to two variables on words using the predicate $<$ for the following node relation and the predicate Succ for the next node relation. For each positive integer k and any two shallow multicontexts p and p', we write $p \equiv_k p'$ the fact that p and p' seen as words agree on all sentences of $\mathrm{FO}^2(<, Succ)$ of quantifier rank k. We denote by k-SMTypes[$\mathrm{X}_{\mathbf{h}}$] the equivalence classes of this relation.

Lemma 10.28. *For any set P of k-SMTypes[$X_{\mathbf{h}}$], the language of P-valid forests is definable in* $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$.

(X,k)-PosTypes[$\mathbf{X_h}$] Let $X \subseteq H$ and consider the logic $\mathrm{FO}^2_X(\mathrm{Succ}, <)$ for denoting positions on shallow multicontexts that cannot distinguish $b(\Box)$ from $b(a)$ whenever $\alpha(a) \notin X$.

Given two nodes x and x' of t we write $x \cong_{k,X} x'$ if the shallow multicontext of x and the shallow multicontext of x' seen as words satisfy the same formulas of $\mathrm{FO}^2_X(\mathrm{Succ}, <)$ of quantifier depth at most k, with one free variable denoting respectively the position x and x'. We denote by (X,k)-PosTypes[$\mathrm{X}_{\mathbf{h}}$] the equivalence classes of this relation and we only consider (X,k)-PosTypes[$\mathrm{X}_{\mathbf{h}}$] such that $P_{b(\Box)}(x)$ holds for some b.

Lemma 10.29. *For any k, any $X \subset H$, and any k-SMType[$X_{\mathbf{h}}$] τ and compatible (X,k)-PosType[$X_{\mathbf{h}}$] δ, there is a formula $\psi_{\tau,\delta}$ of $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ such that for any forest s, the set of nodes of s satisfying $\psi_{\tau,\delta}$ is exactly the set of nodes of s in δ and whose shallow multicontext is in τ.*

As in the previous case, replacing these notion in the definition of saturation yields a new notion of saturation that we call $X_{\mathbf{h}}$-saturation. We now have:

Theorem 10.30. *A regular forest language L is definable in $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}})$ iff its syntactic forest algebra (H,V) verifies:*

a) H verifies for all $f,g \in H$, for all $e \in H$ such that $e^2 = e$:

$$\omega(e+f+e+g+e)+g+\omega(e+f+e+g+e) = \omega(e+f+e+g+e) \quad (10.8)$$

b) V verifies Equation (10.3)

$$(uv)^\omega v (uv)^\omega = (uv)^\omega$$

c) L is closed under $X_{\mathbf{h}}$-saturation.

Recall that Equation (10.8) is exactly the equattion that characterizes $\mathrm{FO}^2(<, Succ)$ in the word setting:

Theorem 3.9. ([TW98, Alm96]) *A regular language L over an alphabet A is definable in $\mathrm{FO}^2(<, Succ)$ iff its syntactic semigroup S verifies, for all $u,v,e \in S$ with e idempotent:*

$$(eueve)^\omega = (eueve)^\omega v (eueve)^\omega \quad (2.3)$$

Aside from the initial choice of the integer k. The proof is identical to the one we gave for Theorem 10.5. We explain here how to choose the integer k. We prove the following Lemma that corresponds to Lemma 10.8:

Lemma 10.31. *There exists a number k'' such that whenever p and p' have the same k''-SMType[$X_{\mathbf{h}}$] then for all forest s we have $\alpha(p[\bar{s}]) = \alpha(p'[\bar{s}])$, where $p[\bar{s}]$ is the forest constructed from p by placing s at each port of p.*

Proof. This is a consequence of Theorem 3.9 and the fact that H verifies Equation (10.8). The proof is identical to the one we provided for Lemma 10.8 replacing Theorem 3.2 by Theorem 3.9 and k-SMType with k-SMType[$\mathrm{X}_{\mathbf{h}}$]. □

10.6 Discussion

10.6.1 Decidability

One major consequence of Theorem 10.5, Theorem 10.22, Theorem 10.26 and Theorem 10.30 is that in order to decide definability of a language L in the corresponding logics one just needs to check if the syntactic forest algebra of L verifies the equations given in the corresponding characterization and if L is closed under the corresponding notion of saturation.

Recall first that the syntactic forest algebra (H, V) of a regular language L can be computed from any automaton recognizing L. Then, by testing all possible combinations, it is straightforward to check whether H and V satisfy (10.3) and (10.2).

The case of saturation is more complicated. We first show that, when k is fixed, one can decide whether a regular language L is closed under k-saturation. This is a consequence of the Pumping Lemma. Recall the definition: We need to check that for all branching set P of k-SMTypes, for all context Δ that is P-saturated, for all P-valid tree t, for all P-valid shallow multicontext p, for all position x of p and for all sequences T of P-valid forests whose types are in H_P, we have:

$$\alpha(\Delta)^\omega\alpha(t) = \alpha(\Delta)^\omega\alpha(p[T,x])\alpha(\Delta)^\omega\alpha(t)$$

were $p[T,x]$ is the context formed from p by placing the forests of T at the corresponding ports of p except for the port at position x. From the Pumping Lemma it follows that only finitely many context Δ, finitely many tree t, finitely many shallow multicontext p and finitely many sequence of forest T needs to be considered. As k is fixed, all the other quantifications also range over finite sets hence a brute force approach testing all possibilities yields an algorithm for testing closure under k-saturation.

Therefore, in order to decide if a language L is closed under saturation it is sufficient to bound the number k such that L may be closed under k-saturation. This is easy for $EF+F^{-1}(S)$ and $EF+F^{-1}(S^{\neq})$ since by definitions the notions of S-saturation and $S^{\neq}$-saturation do not depend on an integer k (see Section 10.5). This yields the following corollary:

Corollary 10.32. *It is decidable whether a forest language is definable in $EF + F^{-1}(S)$. It decidable whether a language is definable in $EF + F^{-1}(S^{\neq})$.*

Note that the brute force algorithm we described yields an awful complexity with several nested exponential for the decision problem. We don't know yet whether this can be improved. Recall that the complexity of the same problem

for the corresponding logics over words is polynomial is the size of the syntactic monoid.

We believe that it also possible to bound the number k such that a language is k-saturated for the notions of saturation corresponding to $EF + F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$ and $EF + F^{-1}(\mathrm{X}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}, \mathrm{X}_{\mathbf{h}}^{-1}, \mathrm{F}_{\mathbf{h}}^{-1})$. Using a tedious pumping argument we believe we that can show that if a language if closed under saturation then it is closed under k-saturation for k computable from the size of the syntactic forest algebra of the language and the size of the alphabet. This is ongoing work. Therefore we conjecture the following result:

Conjecture 10.33. *It is decidable whether a regular forest language is definable in $EF+F^{-1}(\mathrm{F}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}^{-1})$. It is decidable whether a regular forest language is definable in $EF + F^{-1}(\mathrm{X}_{\mathbf{h}}, \mathrm{F}_{\mathbf{h}}, \mathrm{X}_{\mathbf{h}}^{-1}, \mathrm{F}_{\mathbf{h}}^{-1})$.*

10.6.2 Further remarks

It would be interesting to incorporate the vertical successor in our proofs and obtain a decidable characterization for $\mathrm{FO}^2(<_{\mathbf{h}}, \mathit{Succ}_{\mathbf{h}}, <_{\mathbf{v}}, \mathit{Succ}_{\mathbf{v}})$, over trees. This would yield a decidable characterization of the navigational core of XPath. But this seems to require new ideas.

It would also be interesting to obtain an equivalent decidable characterization of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$ without using the cumbersome notion of saturation. For instance it is not clear whether the notion of confusion introduced in [BSW09] can be used as a replacement. We leave this as an open problem.

Our proof technique requires that the logic can at least express the fact that two nodes are siblings. In particular it does not apply to $\mathrm{FO}_2(<_{\mathbf{v}})$. We leave as an open problem to find a decidable characterization for $\mathrm{FO}_2(<_{\mathbf{v}})$.

Chapter 11

Locally Testable Languages over Trees of Bounded Rank and Unranked Trees

In this chapter we concentrate on LT for the setting of trees of binary trees and on ILT,ALT for the setting of unordered unranked trees. Note that while we restrict ourselves to binary trees for the setting of trees of bounded rank, the purpose of this restriction is only to simplify notations. All results we present in this chapter for binary trees extend to trees of rank k for any arbitrary k in a straightforward way.

We will not discuss the setting of forests. However all results presented on unranked unordered trees extend in a straightforward way to forests with an adapted notion of ALT and ILT.

Recall that we already presented two decidable characterizations for LT in the setting of words. One was algebraic and was first introduced in [BS73, McN74]. The second one was using a specific formalism. We quote this result below:

Theorem 4.1. *Consider L, a regular language over some alphabet A, the following properties are equivalent:*

1. *L is in* LT.
2. *L is tame.*
3. *The syntactic semigroup S of L verifies, for all $u, v, e \in S$ such that e is an idempotent:*

$$eueue = eue \tag{4.1}$$

$$eueve = eveue \tag{4.2}$$

In the case of binary trees, we were not able to obtain a reasonably simple set of identities for characterizing LT similar to the ones of Theorem 4.1. Nevertheless we show:

Theorem 11.1. *It is decidable whether a regular binary tree language is in* LT.

Rather than exhibiting a set of identities we prove Theorem 11.1 using a method similar to what we mentioned in the discussion of Chapter 4 in our study of the word setting. We will bound the size of the expected κ such that the language is LT-κ. Our strategy for proving Theorem 11.1 is as follows. In a first step we provide necessary conditions for a language to be in LT. These conditions are an extension to trees of the notion of tame languages we provided for words in Section 4. In a second step we show that if a language L verifies those necessary conditions then we can compute from an automaton recognizing L a number κ such that if L is in LT then L is κ-locally testable. The last step is simple and show that once κ is fixed, it is decidable whether a regular language is κ-locally testable. This last step follows immediately from the fact that once κ is fixed, there are only finitely many κ-locally testable languages and hence one can enumerate them and test whether L is equivalent to one of them or not.

Given a regular language L, testing whether L is in LT is then done by: (1) compute from L the κ of the second step and (2) test whether L is κ-locally testable using the third step. The first step implies that this algorithm is correct.

Before starting providing the proof details we note that there exists examples showing that the necessary conditions are not sufficient. Such an example will be provided in Section 11.4. We also note that the problem of finding κ whenever such a κ exists is a special case of the delay theorem mentioned in the introduction. In the case of LT, the delay theorem says that if a finite state automaton A recognizes a language in LT then this language must be κ-locally testable for a κ computable from A. This theorem was proved over words in [Str85] and can be used in order to decide whether a regular language is in LT as explained in [Boj07a]. We were not able to prove such a general theorem for trees. Our second step can be seen as a particular case of the delay theorem for languages satisfying the conditions provided by the first step. Note that a formalisation of this argument as a "delay theorem" can be found in [Str10].

In the setting of unranked unordered trees, the results for ILT and ALT will be different. We will prove that membership of a regular language in ALT is decidable using arguments similar to the binary tree case. However, for ILT we provide a set of identities that characterizes the class. Recall that all the unranked trees we consider in this chapter are unordered.

The chapter is organized as follows: Sections 11.1 and 11.2 are devoted to binary trees. In Section 11.1 we exhibit our necessary conditions for a tree language to be in LT. In Section 11.2 we show that for the languages satisfying the necessary conditions the expected size of the neighborhoods can be computed, hence concluding the decidability of the characterization. The case of unranked unordered trees is treated in Section 11.3. In this section we explain how to extend the results of Sections 11.1 and 11.2 to the setting of unranked unordered trees.

11.1 Tame Languages of Binary Trees

In this section we exhibit necessary conditions for a regular language of binary trees to be in LT. These conditions will play a crucial role in our decision algorithm. Notice that if we restrict ourselves to words this conditions correspond to the notion of tame word langauges we defined in Chapter 4. These conditions are expressed using the same formalism as the one used in [BS09] for characterizing LTT.

Guarded operations. Let t be a tree, and x, x' be two nodes of t such that x and x' are not related by the descendant relationship. The *horizontal swap* of t at nodes x and x' is the tree t' constructed from t by replacing $t|_x$ with $t|_{x'}$ and vice-versa, see Figure 11.1 (left). A horizontal swap is said to be *k-guarded* if x and x' have the same k-type.

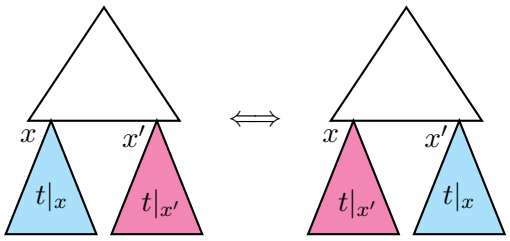


Figure 11.1: Horizontal Swap

Let t be a tree and x, y, z be three nodes of t such that x, y, z are not related by the descendant relationship and such that $t|_x = t|_y$. The *horizontal transfer* of t at x, y, z is the tree t' constructed from t by replacing $t|_y$ with a copy of $t|_z$, see Figure 11.2 (right). A horizontal transfer is k-guarded if x, y, z have the same k-type.

Let t be a tree of root a, and x, y, z be three nodes of t such that y is a descendant of x and z is a descendant of y. The *vertical swap* of t at x, y, z is the tree t' constructed from t by swapping the context between x and y with the

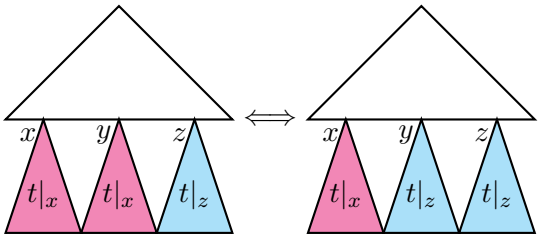


Figure 11.2: Horizontal Transfer

context between y and z, see Figure 11.3 (left). More formally let $C = t[a, x]$, $\Delta_1 = t[x, y]$, $\Delta_2 = t[y, z]$ and $T = t|_z$. We then have $t = C \cdot \Delta_1 \cdot \Delta_2 \cdot T$. The tree t' is defined as $t' = C \cdot \Delta_2 \cdot \Delta_1 \cdot T$. A vertical swap is *$k$-guarded* if x, y, z have the same k-type.

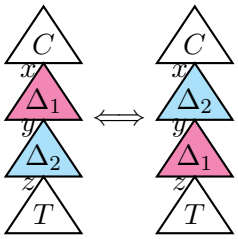


Figure 11.3: Vertical Swap

Let t be a tree of root a, and x, y, z be three nodes of t such that y is a descendant of x and z is a descendant of y such that $\Delta = t[x, y] = t[y, z]$. The *vertical stutter* of t at x, y, z is the tree t' constructed from t by removing the context between x and y, see Figure 11.4 (right). A vertical stutter is k-guarded if x, y, z have the same k-type.

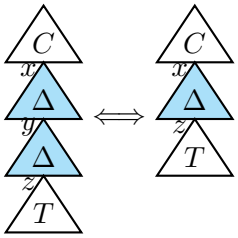


Figure 11.4: Vertical Stutter

Let L be a tree language and k be a number. If X is any of the four constructions above, horizontal or vertical swap, or vertical stutter or horizontal transfer, we say that L is *closed under k-guarded X* if for every tree t and every tree t' constructed from t using k-guarded X then t is in L iff t' is in L. Notice that being closed under k-guarded X implies being closed under k'-guarded X for $k' > k$. An important observation is that each of the k-guarded operation does not affect the set of $(k+1)$-types occurring in the trees.

If L is closed under all the k-guarded operations described above, we say that L is *k-tame.* A language is said to be *tame* if it is k-tame for some k.

The following simple result shows that tameness is a necessary condition for LT.

Proposition 11.2. *If L is in* LT *then L is tame.*

Proof. Assume L is in LT then there is a κ such that L is κ-locally testable. We show that L is κ-tame. This is a straightforward consequence of the fact that all the κ-guarded operations above preserve $(\kappa+1)$-types and hence preserve κ-types. □

A simple pumping argument shows that if L is tame then it is k-tame for k bounded by a polynomial in the size of the minimal deterministic bottom-up tree automata recognizing L. Notice that once k is fixed, a brute force algorithm can check whether L is k-tame or not. Therefore the proposition below implies that tameness is decidable. However for deciding LT we will only need the bound k_0 given by the proposition.

Proposition 11.3. *Given a regular language L and A the minimal deterministic bottom-up tree automata recognizing L, we have L is tame iff L is k_0-tame for $k_0 = |A|^3 + 1$.*

Proof. We prove that if X is one of the four operations that defines tameness, then if L is closed under k-guarded X for $k > k_0$, then L is closed under k_0-guarded X. This will imply that if L is k-tame then it is k_0-tame.

Consider the case of k-guarded horizontal transfer and assume L is closed under k-guarded horizontal transfers. We show that L is closed under k_0-guarded horizontal transfers. Let t be a tree and x, y, z three nodes of t having the same k_0-type and not related by the descendant relation such that $t|_x = t|_y$. We need to show that replacing $t|_y$ by a copy of $t|_z$ does not affect membership into L.

We do this in three steps, first we transform t by pumping in parallel in the subtrees of x, y and z until x, y, z have the same k-type, then we use the closure of L under k-guarded horizontal transfer in order to replace $t|_y$ by a copy of $t|_z$, and finally we backtrack the initial pumping phase in order to recover the initial subtrees.

We let $t_1 = t|_x$ and $t_2 = t|_z$ and we assume for now on that $t_1 \neq t_2$. By *position* we denote a string w of $\{0,1\}^*$. A position w is *realized* in a tree t if there is a node x of t such that if $x_1, \cdots, x_n = x$ is the sequence of nodes in the path from the root of t to x then for all $i \leq n$ the i^{th} bit of w is zero if x_i is a left child and it is one if x_i is a right child. We order positions by first comparing their respective length and then using the lexicographical order.

By hypothesis t_1 and t_2 are identical up to depth at least k_0. Let w be the first position such that t_1 and t_2 differ at that position. That can be either because w is realized in t_1 but not in t_2, or vice versa, or w is realized in both trees but the labels of the corresponding nodes differ. We know that the length n of w is strictly greater than k_0. If $n > k$, we are done with the first phase. We assume now that $n \leq k$.

Consider the run r of A on t. The run assigns a state q to each node of t. From r we assign to each position $w' < w$ a pair of states (q, q') such that q is the state given by r at the corresponding node in t_1 while q' is the state given by r at the corresponding node in t_2. Because $n > k_0 > |A|^2$, there must be two prefixes w_1 and w_2 of w that were assigned the same pair of states. Consider the context $C_1 = t_1[v_1, v_2]$ where v and v' are the nodes of t_1 at position w_1 and w_2 and the context $C_2 = t_2[v'_1, v'_2]$ where v and v' are the nodes of t_2 at position w_1 and w_2. Without affecting membership in L, we can therefore at the same time duplicate C_1 in the two copies of t_1 rooted at x and y and C_2 in the copy of t_2 rooted at z.

Let t'_1 and t'_2 be the subtrees of the resulting tree, rooted respectively at x and z. The reader can verify that t'_1 and t'_2 now differ at a position strictly greater than w.

Performing this repeatedly, we eventually arrive in a situation where the subtree t'_1 rooted at x and y agree up to depth k with the subtree rooted at z. We can now apply k-guarded horizontal transfer and replace one occurrence of t'_1 by a copy of t'_2. We can then replace t'_1 by t_1 and both copies of t'_2 by t_2 without affecting membership into L.

The other operations are done similarly. For the horizontal swap, we pump the subtrees at positions x and x' simultaneously, which is possible because $k_0 > |A|^2$. For vertical swap, we pump the subtrees at the positions x, y and z simultaneously, and that requires $k_0 > |A|^3$. Finally, in for vertical stutter, we pump the subtrees at the positions x, y and z simultaneously, which again requires $k_0 > |A|^3$. □

11.2 Deciding LT for Binary Trees

In this section we show that it is decidable whether a regular tree language is in LT. This is done by showing that if a regular language L is in LT then there is a κ computable from an automata recognizing L such that L is in fact κ-

locally testable. Recall that once this κ is computed the decision procedure simply enumerates all the finitely many κ-locally testable languages and tests whether L is one of them.

Assume L is in LT. By Proposition 11.2, L is tame. Even more, from Proposition 11.3, one can effectively compute a k such that L is k-tame. Hence Theorem 11.1 follows from the following proposition.

Proposition 11.4. *Assume L is a k-tame regular tree language then L is in* LT *iff L is κ-locally testable where κ is computable from k.*

Recall that for each k the number of k-types is finite. Let β_k be this number. Proposition 11.4 is an immediate consequence of the following proposition.

Proposition 11.5. *Let L be a k-tame regular tree language. Set $\kappa = \beta_k + k + 1$. Then for all $l > \kappa$ and any two trees t, t' if $t \simeq_\kappa t'$ then there exists two trees T, T' with*

1. *$t \in L$ iff $T \in L$*
2. *$t' \in L$ iff $T' \in L$*
3. *$T \simeq_l T'$*

Proof of Proposition 11.4 using Proposition 11.5. Assume L is k-tame and let κ be defined as in Proposition 11.5. We show that L is in LT iff L is κ-locally testable. Assume L is in LT. Then L is l-locally testable for some $l \in \mathbb{N}$. We show that L is actually κ-locally testable. For this it suffices to show that for any pair of trees t and t', if $t \simeq_\kappa t'$ then $t \in L$ iff $t' \in L$. Let T and T' be the trees constructed for l from t and t' by Proposition 11.5. We have $T \simeq_l T'$ and therefore $T \in L$ iff $T' \in L$. As we also have $t \in L$ iff $T \in L$ and $t' \in L$ iff $T' \in L$, the proposition is proved. □

Before proving Proposition 11.5 we need some extra terminology. A non-empty context C occurring in a tree t is a *loop of k-type τ* if the k-type of its root and the k-type of its port is τ. A non-empty context C occurring in a tree t is a k-loop if there is some k-type τ such that C is a loop of k-type τ. Given a context C we call the path from the root of C to its port the *principal path of C*. Finally, the result of the *insertion* of a k-loop C at a node x of a tree t is a tree T such that if $t = D \cdot t|_x$ then $T = D \cdot C \cdot t|_x$. Typically an insertion will occur only when the k-type of x is τ and C is a loop of k-type τ. In this case the k-types of the nodes initially from t and of the nodes of C are unchanged by this operation.

Proof of Proposition 11.5. Suppose that L is k-tame. We start by proving two lemmas that will be useful in the construction of T and T'. Essentially these

lemmas show that even though being k-tame does not imply being $(k+1)$-locally testable (recall the remark after Theorem 11.1) some of the expected behavior of $(k+1)$-locally testable languages can still be derived from being k-tame. The first lemma shows that given a tree t, without affecting membership into L, we can replace a subtree of t containing only $(k+1)$-types occurring elsewhere in t by any other subtree satisfying this property and having the same k-type as root. The second lemma shows the same result for contexts by showing that a k-loop can be inserted in a tree t without affecting membership into L as soon as all the $(k+1)$-types of the k-loop were already present in t. After proving these lemmas we will see how to combine then for constructing T and T'.

Lemma 11.6. *Assume L is k-tame. Let $t = Ds$ be a tree where s is a subtree of t. Let s' be another tree such that the roots of s and s' have the same k-type.*

If $s \preccurlyeq_{k+1} D$ and $s' \preccurlyeq_{k+1} D$ then $Ds \in L$ iff $Ds' \in L$.

Proof. We start by proving a special case of the Lemma when s' is actually another subtree of t. We will use repeatedly this particular case in the proof.

Claim 11.7. *Assume L is k-tame. Let t be a tree and let x, y be two nodes of t not related by the descendant relationship and with the same k-type. We write $s = t|_x$, $s' = t|_y$ and C the context such that $t = Cs$. If $s \preccurlyeq_{k+1} C$ then $Cs \in L$ iff $Cs' \in L$.*

Proof. The proof is done by induction on the depth of s and make a crucial use of k-guarded horizontal transfer.

Assume first that s is of depth less than k. Since x and y have the same k-type, we have $s = s'$ and the result follows.

Assume now that s is of depth greater than k.

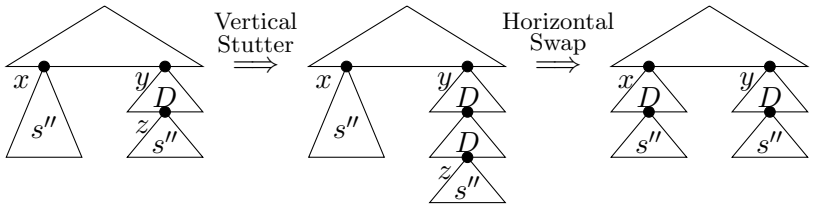
Let τ be the $(k+1)$-type of x. We assume that s is a tree of the form $a(s_1, s_2)$. Notice that the k-type of the roots of s_1 and s_2 are completely determined by τ. Since $s \preccurlyeq_{k+1} C$, there exists a node z in C of type τ. We write $s'' = t|_z$.

We consider several cases depending on the relationship between x, y and z. We first consider the case where x and z are not related by the descendant relationship, then we reduce the other cases to this case.

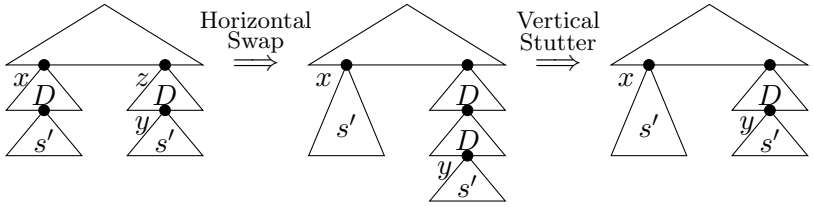
Assume that x and z are not related by the descendant relationship. Since s'' is of type τ, it is of the form $a(s_1'', s_2'')$ where the roots of s_1'' and s_2'' have the same k-type than respectively the roots of s_1 and s_2. By hypothesis all the $(k+1)$-types of s_1 and s_2 already appear in C and hence we can apply the induction hypothesis to replace s_1 by s_1'' and s_2 by s_2'' without affecting membership into L. Notice that the resulting tree is Cs'', that $t = Cs \in L$ iff $Cs'' \in L$, and that Cs'' contains two copies of the subtree s'', one at position x and one at position z. We now show that we can derive Cs' from Cs'' using k-guarded operations. Since L is k-tame

it will follow that that $Cs'' \in L$ iff $Cs' \in L$ and thus $Cs \in L$ iff $Cs' \in L$. Let $t'' = Cs''$ and we distinguish between three cases depending on the relationship between z and y in t'':

1. If z is descendant of y, let $D = t''[y, z]$ and notice that $s' = Ds''$. Since x, y and z have the same k-type, we use k-guarded vertical stutter to duplicate D and a k-guarded horizontal swap to move the new copy of D at position x (see the picture below). The resulting tree is Cs' as desired.



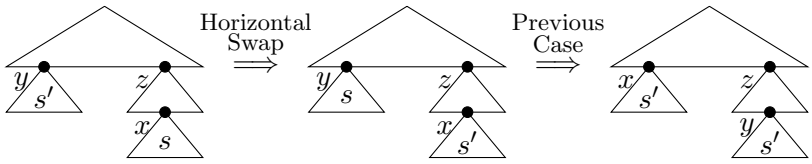
2. If z is an ancestor of y, let $D = t''[z, y]$ and notice that $s'' = Ds'$. Since y and x have the same k-type, we use k-guarded horizontal swap followed by a k-guarded vertical stutter to delete the copy of D (see the picture below). The resulting tree is Cs' as desired.



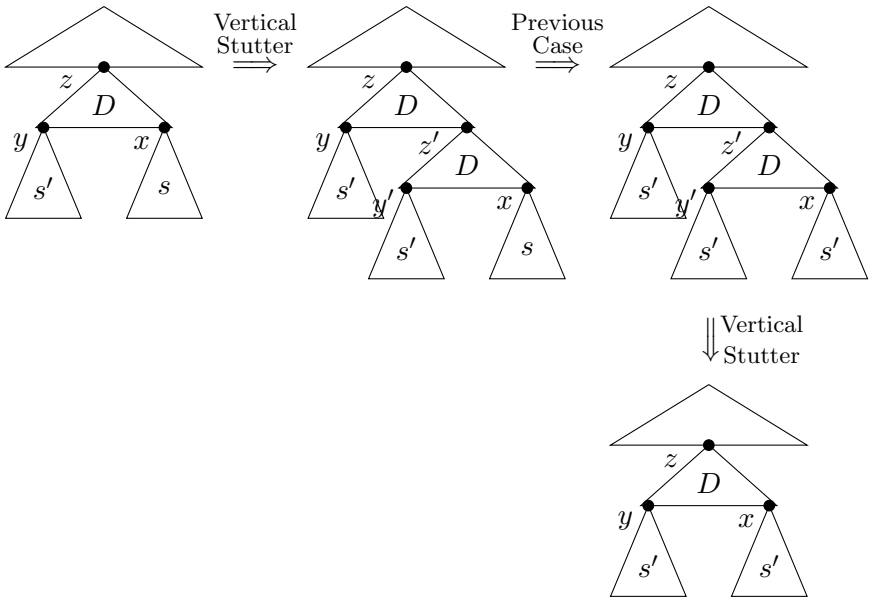
3. If z and y are not related by the descendant relation, then x, y and z have the same k-type, x, y, z that are not related by the descendant relationship and $t''|_x = t''|_z$. We use k-guarded horizontal transfer to replace $t''|_x$ with $t''|_y$ as depicted below.

This concludes the case where x and z are not related by the descendant relationship in t. We are left with the case where x is a descendant of z (recall that z is outside s and therefore not a descendant of x). We reduce this problem to the previous case by considering two subcases:

- If y, z are not related by the descendant relationship, we use a k-guarded horizontal swap to replace s by s' and vice versa. This reverses the roles of x and y and as y and z are not related by the descendant relationship we can apply the previous case.



- If z is an ancestor of both x and y we use k-guarded vertical stutter to duplicate the context between z and x. This introduces a new node z' of type τ that is not related to y by the descendant relationship and we are back with the previous case.



□

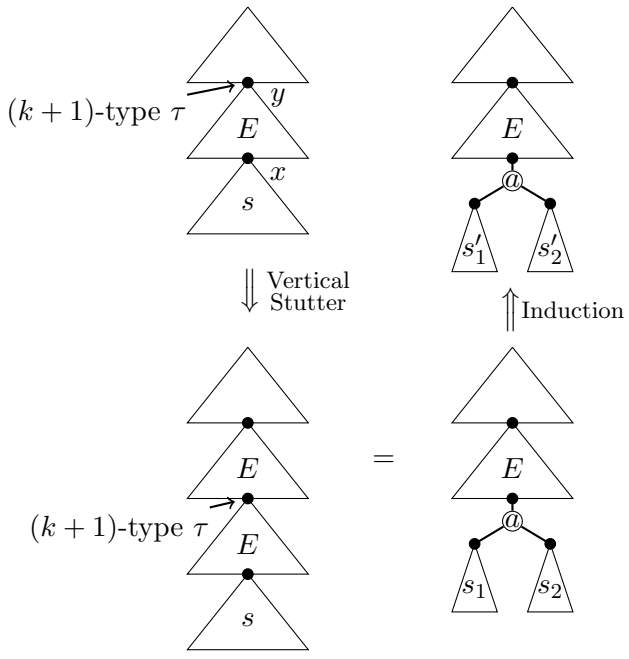
We now turn to the proof of Lemma 11.6. The proof is done by induction on the depth of s'. The idea is to replace s with s' node by node.

Assume first that s' is of depth less than k. Then because the k-type of the roots of s and s' are equal, we have $s = s'$ and the result follows.

Assume now that s' is of depth greater than k.

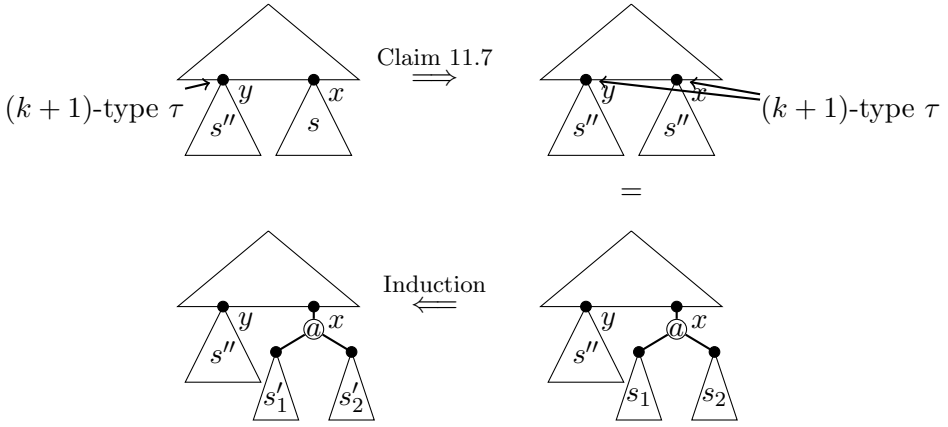
Let x be the node of t corresponding to the root of s. Let τ be the $(k+1)$-type of the root of s'. We assume that s' is a tree of the form $a(s'_1, s'_2)$. Notice that the k-type of the roots of s'_1 and s'_2 are completely determined by τ. By hypothesis $s' \preccurlyeq_{k+1} D$, hence there exists a node y in D of type τ. We consider two cases depending on the relationship between x and y.

- If y is an ancestor of x, let E be $t[y, x]$ and notice that x and y have the same k-type. This case is depicted below. Hence applying a k-guarded vertical stutter we can duplicate E obtaining the tree DEs. Because L is k-tame, $DEs \in L$ iff $t = Ds \in L$. Now the root of Es in DES is of type τ and therefore of the form $a(s_1, s_2)$ where the roots of s_1 and s_2 have the same k-type than respectively the roots of s'_1 and s'_2. By construction all the $(k+1)$-types of s_1 and s_2 already appear in D and hence we can apply the induction hypothesis to replace s_1 by s'_1 and s_2 by s'_2 without affecting membership into L. Altogether this gives the desired result.



- Assume now that x and y are not related by the descendant relationship. This case is depicted below. Let s'' be the subtree of Ds rooted at y. By

hypothesis all the $(k+1)$-types of s are already present in D and the roots of s and s'' have the same k-type. Hence we can apply Claim 11.7 and we have $Ds \in L$ iff $Ds'' \in L$. Now the root of s'' is by construction of type τ. Hence s'' if of the form $a(s_1, s_2)$ where s_1 and s_2 have all their $(k+1)$-types appearing in D and their roots have the same k-type as respectively s'_1 and s'_2. Hence by induction s_1 can be replaced by s'_1 and s_2 by s'_2 without affecting membership into L. Altogether this gives the desired result.



□

We now prove a similar result for k-loops.

Lemma 11.8. *Assume L is k-tame. Let t be a tree and x a node of t of k-type τ. Let t' be another tree such that $t \simeq_{k+1} t'$ and C be a k-loop of type τ in t'. Consider the tree T constructed from t by inserting a copy of C at x. Then $t \in L$ iff $T \in L$.*

Proof. The proof is done in two steps. First we use the k-tame property of L to show that we can insert a k-loop C' at x in t such that the principal path of C is the same as the principal path of C'. By this we mean that there is a bijection from the principal path of C' to the principal path of C that preserves the child relation and $(k+1)$-types. In a second step we replace one by one the subtrees hanging from the principal path of C' with the corresponding subtrees in C.

First some terminology. Given two nodes x, y of some tree T, we say that x is a **l**-ancestor of y if y is a descendant of the left child of x. Similarly we define **r**-ancestorship.

Consider the context C occurring in t'. Let $y_0, \cdots, y_n$ be the nodes of t' on the principal path of C and $\tau_0, \cdots, \tau_n$ be their respective $(k+1)$-type. For $0 \leq i < n$, set c_i to **l** if y_{i+1} is a left child of y_i and **r** otherwise.

From t we construct using k-guarded swaps and k-vertical stutters a tree t_1 such that there is a sequence of nodes $x_0, \cdots, x_n$ in t_1 with for all $0 \leq i < n$, x_i is of type τ_i and x_i is an c_i-ancestor of x_{i+1}. The tree t_1 is constructed by induction on n. If $n = 0$ then this is a consequence of $t \simeq_{k+1} t'$ that one can find in t a node of type τ_0. Consider now the case $n > 0$. By induction we have constructed from t a tree t'_1 such that $x_0, \cdots, x_{n-1}$ is an appropriate sequence in t'_1. By symmetry it is enough to consider the case where y_n is the left child of y_{n-1}. Because all k-guarded operations preserve $(k+1)$-types, we have $t \simeq_{k+1} t'_1$ and hence there is a node x of t'_1 of type τ_n. If x_{n-1} is a $\mathbf{l}$-ancestor of x then we are done. Otherwise consider the left child x' of x_{n-1} and notice that because y_n is a child of y_{n-1} and x_{n-1} has the same $(k+1)$-type than y_{n-1} then x', y_n and x have the same k-type.

We know that x is not a descendant of x'. There are two cases. If x and x' are not related by the descendant relationship then by k-guarded swaps we can replace the subtree rooted in x' by the subtree rooted in x and we are done. If x is an ancestor of x' then the context between x and x' is a k-loop and we can use k-guarded vertical stutter to duplicate it. This places a node having the same $(k+1)$-type as x as the left child of x_{n-1} and we are done.

This concludes the construction of t_1. From t_1 we construct using k-guarded swaps and k-guarded vertical stutter a tree t_2 such that there is a path $x_0, \cdots, x_n$ in t_2 with for all $0 \leq i < n$, x_i is of type τ_i.

Consider the sequence $x_0, \cdots, x_n$ obtained in t_1 from the previous step. Recall that the k-type of x_0 is the same as the k-type of x_n. Hence using k-guarded vertical stutter we can duplicate in t_1 the context rooted in x_0 and whose port is x_n. Let t'_1 the resulting tree. We thus have two copies of the sequence $x_0, \cdots, x_n$ that we denote by the *top copy* and the *bottom copy*. Assume x_i is not a child of x_{i-1}. By symmetry it is enough to consider the case where x_{i-1} is a $\mathbf{l}$-ancestor of x_i. Notice then that the context between the left child of x_{i-1} and x_i is a k-loop. Using k-guarded vertical swap we can move the top copy of this context next to its bottom copy. Using k-guarded vertical stutter this extra copy can be removed. We are left with an instance of the initial sequence in the bottom copy, while in the top one x_i is a child of x_{i-1}. This construction is depicted in figure 11.5.

Repeating this argument yields the desired tree t_2.

Consider now the context $C' = t_2[x_0, x_n]$. It is a loop of k-type τ. Let T' be the tree constructed from t by inserting C' at x.

Claim 11.9. *$T' \in L$ iff $t \in L$.*

Proof. Consider the sequence of k-guarded swaps and k-guarded vertical stutter that was used in order to obtain t_2 from t. Because L is k-tame, $t \in L$ iff $t_2 \in L$.

We can easily identify the nodes of t with the nodes of T' outside of C'. Consider the same sequence of k-guarded operations applied to T'. Observe that this yields a tree T_2 corresponding to t_2 with possibly several extra copies of C'.

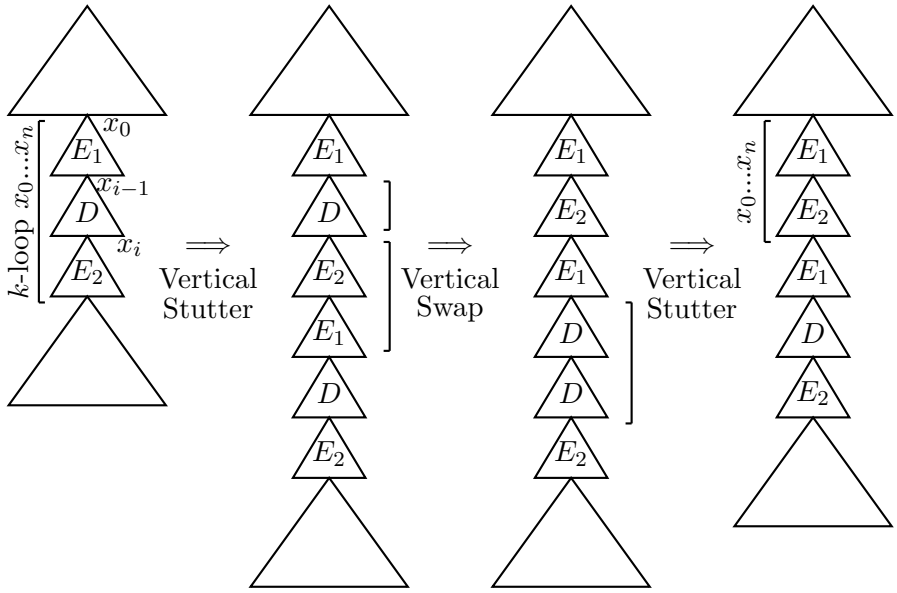


Figure 11.5: The construction of t_2, eliminating the context D between x_{i-1} and x_i

With appropriates k-guarded swaps, all the extra copies can be brought together and using k-guarded vertical stutter only one copy remains resulting in t_2. Hence $T' \in L$ iff $t_2 \in L$ and the claim is proved. See figure 11.6. □

It remains to show that $T' \in L$ iff $T \in L$. By construction of T' we have $C' \preccurlyeq_{k+1} t$. Consider now a node x_i in the principal path of C'. Let T_i be the subtree branching out the principal path of C at y_i and T'_i be the subtree branching out the principal path of C' at x_i. By construction x_i and y_i are of $(k+1)$-type τ_i. Therefore the roots of T_i and T'_i have the same k-type. Because $C' \preccurlyeq_{k+1} t$ all the $(k+1)$-types of T'_i already appear in the part of T' outside of C'. By hypothesis we also have $T_i \preccurlyeq_{k+1} t$. Hence we can apply Lemma 11.6 and replacing T'_i with T_i does not affect membership into L. A repeated use of that Lemma eventually shows that $T' \in L$ iff $T \in L$. □

We now turn to the construction of T and T' and prove Proposition 11.4.

Recall that the number of k-types is β_k. Therefore, by choice of κ, in every branch of a κ-type one can find at least one k-type that is repeated. This provides many k-loops that can be used using Lemma 11.8 for obtaining bigger types.

Take $l > \kappa$, we build T and T' from t and t' by inserting k-loops in t and t' without affecting their membership in L using Lemma 11.8.

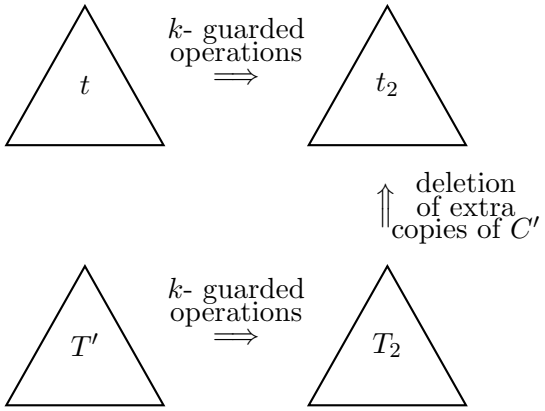


Figure 11.6: Relation with t_2

Let $B = \{\tau_0, ..., \tau_n\}$ be the set of k-types τ such that there is a loop of k-type τ in t or in t'. For each $\tau \in B$ we fix a context C_τ as follows. Because $\tau \in B$ there is a context C in t or t' that is a loop of k-type τ. For each $\tau \in B$, we fix arbitrarily such a C and set C_τ as $\underbrace{C \cdot \ldots \cdot C}_{l}$, l concatenations of the context C. Notice that the path from the root of C_τ to its port is then bigger than l.

We now describe the construction of T from t. The construction of T' from t' is done similarly. The tree T is constructed by simultaneously inserting, for all $\tau \in B$, a copy of the context C_τ at all nodes of t of type τ.

We now show that T and T' have the desired properties.

The first and second properties, $t \in L$ iff $T \in L$ and $t' \in L$ iff $T' \in L$, essentially follow from Lemma 11.8. We only show that $t \in L$ iff $T \in L$, the second property is proved symmetrically. We view T as if it was constructed from t using a sequence of insertions of some context C_τ for $\tau \in B$. We write $s_0, ..., s_m$ the sequence of intermediate trees with $s_0 = t$ and $s_m = T$. We call C_i the context inserted to get s_i from s_{i-1}. We show by induction on i that (i) $s_i \simeq_{k+1} t$ and (ii) $s_i \in L$ iff $s_{i+1} \in L$. This will imply $t \in L$ iff $T \in L$ as desired. (i) is clear for $i = 0$. We show that for all i (i) implies (ii). Recall that C_i is constructed by l copies of a k-loop present either in t or in t'. Let s be this tree and let s' be the tree constructed from s by duplicating the k-loop l times. Hence s' is a tree containing C_i and by construction $s' \simeq_{k+1} s$. Because $t \simeq_\kappa t'$ with $\kappa > k + 1$ and $s_i \simeq_{k+1} t$ we have $s' \simeq_{k+1} s_i$. By Lemma 11.8 this implies that $s_{i+1} \in L$ iff $s_i \in L$. By construction we also have $s_{i+1} \simeq_{k+1} s_i$ and the induction step is proved.

We now show the third property:

Lemma 11.10. $T \simeq_l T'$

Proof. We need to show that $T \preccurlyeq_l T'$, $T' \preccurlyeq_l T$ and that the roots of T and T' have the same l-type. It will be convenient for proving this to view the nodes of T as the union of the nodes of t plus some nodes coming from the k-loops that were inserted. To do this more formally, if x is a node of t of k-type not in B, then x is identified with the corresponding node of T. If x is a node of t whose k-type is in B then x is identified in T with the port of the copy of C_τ that was inserted at node x. We start with the following claim.

Claim 11.11. *Take two nodes x in t and x' in t', such that x and x' have the same κ-type. Let y and y' be the corresponding nodes in T and T'. Then y and y' have the same l-type.*

Proof. Let ν the κ-type of x and x'. Consider a branch of ν of length κ. By the choice of κ we know that in this branch one can find two nodes z and z' with the same k-types τ, with z an ancestor of z' and such that the k-type τ of z is determined by ν (z is at distance $\geq k$ from the leaves of ν). Hence τ is in B. Note that because the k-type of z is included in ν, the presence of a node of type ν induces the presence of a node of type τ at the same relative position than z. Hence a copy of C_τ is done simultaneously at the same place relative of y and y' during the construction of T and T'. Because this is true for all branches of ν and because all C_τ have depth at least l, then y and y' have the same l-type. □

From claim 11.11 it follows that the roots of T and T' have the same l-type. By symmetry we only show that $T \preccurlyeq_l T'$. Let y be a node of T and μ be its l-type. We show that there exists $y' \in T'$ with type μ. We consider two cases:

- y is not a node of a loop inserted during the construction of T. Let x be the corresponding position in t and let ν be its κ-type. Since $t \simeq_\kappa t'$, there is a node x' of t' of type ν. Let y' be the node of T' corresponding to y'. By Claim 11.11 y and y' have the same l-type.

- y is a node inside a copy of C_τ inserted to construct T. Let x be the node of t where this loop was inserted. Let ν be the κ-type of x (the k-type of x is τ). Since $t \simeq_\kappa t'$, there is a node x' of t' of type ν. Since $\kappa > k$, x and x' have the same k-type, a copy of C_τ was also inserted in t' at position x' during the construction of T'. From Claim 11.11, x and x', when view as nodes of T and T' have the same l-type. Let y' be the node of T' in the copy of C_τ inserted at x' that corresponds to the position y. Since y and y' are ancestors of x and x' that have the same l-type, and that the context from y to x is the same as the context from y' to x', then y and y' must have the same l-type.

□

This concludes the proof of Proposition 11.4. □

11.3 Unranked Unordered Trees

In this section we consider unranked unordered trees with labels in A. In such trees, each node may have an arbitrary number of children but no order is assumed on these children. In particular even if a node has only two children we can not necessarily distinguish the left child from the right child.

Our goal is to adapt the result of the previous section and provide a decidable characterization of locally testable languages of unranked unordered trees. Since we work directly with unordrered trees, we redefine the notion on regular languages over unordered trees.

In this section by *regular language* we mean definable in the logic MSO using only the child predicate and unary predicates for the labels of the nodes. There is also an equivalent automata model that we briefly describe next. A tree automaton A over unordered unranked trees consists essentially of a finite set of states $Q = \{q_1, \cdots, q_k\}$, an integer m denoted as the *counter threshold* in the sequel, and a transition function δ associating a unique state to any pair consisting of a label and a tuple $(q_1, \gamma_1) \cdots (q_k, \gamma_k)$ where $\gamma_i \in \{= i \mid i < m\} \cup \{\geq m\}$. The meaning is straightforward via bottom-up evaluation: A node of label **a** get assigned a state q if for all i, the number of its children, up to threshold m, that were assigned state q_i is as specified by δ. In the sequel we assume without loss of generality that all our tree automata are deterministic.

We recall the definitions of LT over unranked unordered trees. In the unranked tree case, there are several natural definitions of LT. Recall the definition of k-type: the k-type of a node x is the isomorphism type of the subtree induced by the descendant of x at distance at most k from x. With unranked trees this definition generates infinitely many k-types. We therefore introduce a more flexible notion of type, (k,l)-type, based on one extra parameter l restricting the horizontal information. It is defined by induction on k. Consider an unordered tree t and a node x of t. For $k = 0$, the (k,l)-type of x is just the label of x. For $k > 0$ the (k,l)-type of x is the label of x together with, for each $(k-1,l)$-type, the number, up to threshold l, of children of x of this type. The reader can verify that over binary trees, the $(k,2)$-type and the k-type of x always coincide. As in the previous section we say that two trees are (k,l)-equivalent, and denote this using $\simeq_{(k,l)}$, if they have the same occurrences of (k,l)-types and their roots have the same (k,l)-type. We also use $t \preccurlyeq_{(k,l)} t'$ to denote the fact that all (k,l)-types of t also occur in t'.

Based on this new notion of type, we define two notions of locally testable languages. The most expressive one, denoted ALT (A for *Aperiodic*), is defined as

follows. A language L is in (κ, λ)-ALT if it is a union of (κ, λ)-equivalence classes. A language L is in ALT if there is a k and a λ such that L is in (κ, λ)-ALT.

The second one, denoted ILT in the sequel (I for *Idempotent*), assumes $\lambda = 1$: A language L is in ILT if there is a κ such that L is a union of $(\kappa, 1)$-equivalence classes.

The main result of this section is that we can decide membership for both ILT and ALT.

Theorem 11.12. *It is decidable whether a regular unranked unordered tree language is* ILT. *It is decidable whether a regular unranked unordered tree language is* ALT.

Tameness The notion of k-tame is defined as in Section 11.1 using the same k-guarded operations requiring that the swaps nodes have identical k-type. We also define a notion of (k, l)-tame which corresponds to our new notion of (k, l)-type. Consider the four operations of tameness defined in Section 11.1, a horizontal swap is said to be (k, l)-guarded if x and x' have the same (k, l)-type, a horizontal transfer is (k, l)-guarded if x, y, z have the same (k, l)-type, a vertical swap is (k, l)-guarded if x, y, z have the same (k, l)-type and a vertical stutter is (k, l)-guarded if x, y, z have the same (k, l)-type. Let L be a regular unranked unordered tree language and let m be the counting threshold of the minimal automaton recognizing L, we say that L is (k, l)-tame iff it is closed under (k, l)-guarded horizontal swap, horizontal transfer, vertical swap and vertical stutter and $l > m$[1]. We first prove that over unordered trees being k-tame is the same as being (k, l)-tame.

Proposition 11.13. *Let L be an unordered unranked regular tree language, then for all integer k, L is k-tame iff there exists l such that L is (k, l)-tame. Furthermore, such a l can be computed from any automaton recognizing L.*

Proof. If there exists l such that L is (k, l)-tame then L is obviously k-tame. Suppose that L is k-tame, and let m be the counting threshold of the minimal automaton A recognizing L, we show that there exists l' such that L is closed under (k, l')-guarded operations. This implies the result as one can then take $l = max(m + 1, l')$.

We need to show that L is closed under (k, l')-guarded vertical swap, vertical stutter, horizontal swap and horizontal transfer. The proof is similar to the proof of *Proposition 1* in [BS09]. We will use the following claim which is proved in [BS09] using a simple pumping argument:

[1]We assume $l > m$ in order to make the statements of the results similar to those used in the binary setting.

Claim 11.14. *[BS09] For every tree automaton A there is a number l', computable from A, such that for every k if a tree t_1 is (k, l')-equivalent to a tree t_2, then there are trees t_1', t_2' with t_1' and t_2' k-equivalent such that A reaches the same state on t_i' as on t_i for $i = 1, 2$.*

We use this claim to prove that L is closed under horizontal transfer. Let l' be the number computed from A by Claim 11.14. We prove that L is closed under (k, l')-guarded horizontal transfer. Consider a tree t and three nodes x, y, z of t not related by the descendant relationship and such that $t|_x = t|_y$ and such that x, y and z have the same (k, l)-type. Let t' be the horizontal transfer of t at x, y, z. Let $t_1 = t|_x$ and $t_2 = t|_z$ and t_1', t_2' obtained from t_1, t_2 using Claim 11.14. Let s be the tree obtained from t by replacing $t|_x$ and $t|_y$ with t_1' and $t|_z$ with t_2', and let s' be the tree obtained from t' by replacing $t'|_x$ with t_1' and $t'|_y$ and $t'|_z$ with t_2. From Claim 11.14 it follows that $t \in L$ iff $s \in L$ and $t' \in L$ iff $s' \in L$. Since L is k-tame, it is closed under k-guarded horizontal transfer, therefore we have $s \in L$ iff $s' \in L$, it follows that $t \in L$ iff $t' \in L$.

The closure under horizontal swap is proved using the same claim. The proof for vertical swap and vertical stutter use a claim similar to Claim 11.14 but for contexts: For every tree automaton A there is a number l computable from A such that for every k if the context C_1 is (k, l)-equivalent to the context C_2 (by this we mean that their roots have the same (k, l)-type), then there are contexts C_1', C_2' with C_1' k-equivalent to C_2' such that C_i' induces the same function on the states of A as C_i for $i = 1, 2$. □

From this lemma we know that a regular language over unranked unordered trees is tame iff it is k-tame for some k iff it is (k, l)-tame for some k, l. Moreover, as in the binary setting, if a regular language is tame then it is (k, l)-tame for some k and l computable from an automaton recognizing L. The bound on k can be obtained by a straightforward adaptation of Proposition 11.3. The bound on l then follows from Proposition 11.13. Hence we have:

Proposition 11.15. *Let L be a regular language and let A be its minimal deterministic bottom-up tree automata, we have L is tame iff L is (k_0, l_0)-tame for $k_0 = |A|^3 + 1$ and some l_0 computable from A.*

11.3.1 Decision of ALT

We now turn to the proof of Theorem 11.12. We begin with the proof for ALT as both the decision procedure and its proof are obtained as in the case of binary trees. Assuming tameness we obtain a bound on κ and λ such that a language is in ALT iff it is in (κ, λ)-ALT. Once κ and λ are known, it is easy do decide if a language is (κ, λ)-ALT since the number of such languages is finite. The

bounds on κ and λ are obtained following the same proof structure as in the binary cases, essentially replacing k-tame by (k,l)-tame, but with several technical modifications. Therefore, we only sketch the proofs below and only detail the new technical material. Our goal is to prove the following result.

Proposition 11.16. *Assume L is a (k,l)-tame regular tree language and let A be its minimal automaton. Then L is in* ALT *iff L is in* (κ,λ)-ALT *where κ and λ are computable from k, l and A.*

Notice that for each k,l the number of (k,l)-types is finite, let $\beta_{k,l}$ be this number. Proposition 11.16 is now a simple consequence of the following proposition.

Proposition 11.17. *Let L be a (k,l)-tame regular tree language and let A be the minimal automaton recognizing L. Set $\lambda = |A|l+1$ and $\kappa = \beta_{k,l}+k+1$. Then for all $\kappa' > \kappa$, all $\lambda' > \lambda$ and any two trees t, t' if $t \simeq_{(\kappa,\lambda)} t'$ then there exists two trees T, T' with*

1. *$t \in L$ iff $T \in L$*
2. *$t' \in L$ iff $T' \in L$*
3. *$T \simeq_{(\kappa',\lambda')} T'$.*

Before proving Proposition 11.17 we adapt the extra terminology we used in the proof of Proposition 11.5 to the unranked setting. A non-empty context C occurring in a tree t is a *loop of (k,l)-type τ* if the (k,l)-type of its root and the (k,l)-type of its port is τ. A non-empty context C occurring in a tree t is a (k,l)-loop if there is some (k,l)-type τ such that C is a loop of (k,l)-type τ. Given a context C we call the path from the root of C to its port the *principal path of C*. Finally, the result of the *insertion* of a (k,l)-loop C at a node x of a tree t is a tree T such that if $t = D \cdot t|_x$ then $T = D \cdot C \cdot t|_x$. Typically an insertion will occur only when the (k,l)-type of x is τ and C is a loop of (k,l)-type τ. In this case the (k,l)-types of the nodes initially from t and of the nodes of C are unchanged by this operation.

Proof of Proposition 11.17. Suppose that L is (k,l)-tame. Like we did for the proof of the binary case we first prove two lemmas that are crucial for the construction of T and T'. They show that subtrees can be replaced and contexts can be inserted as soon as this does not change the (k+1,l)-equivalence class of the tree. They are direct adaptations of the corresponding lemmas for the ranked setting: Lemmas 11.6 and 11.8. We start with subtrees.

Lemma 11.18. *Assume L is (k,l)-tame. Let $t = Ds$ be a tree where s is a subtree of t. Let s' be another tree such that the roots of s and s' have the same (k,l)-type.*

If $s \preccurlyeq_{(k+1,l)} D$ and $s' \preccurlyeq_{(k+1,l)} D$ then $Ds \in L$ iff $Ds' \in L$.

Proof sketch. As in the binary setting the proof is done by first proving a restricted version where s' is actually another subtree of t. Before doing that, we state a new claim, specific to the unranked setting, that will be useful later for solving the induction bases of our proofs. In the binary setting, two trees that had the same k-type at their root and were of depth smaller than k were equal. This obviously does not extend to unranked trees and (k,l)-types. However it is simple to see that equality can be replaced by indistinguishable by the minimal tree automaton recognizing L.

Claim 11.19. *Let A a tree automaton and m be its counting threshold. Let t and t' be two trees of depth smaller than k and whose roots have the same (k,m)-type. Then t and t' evaluate to the same state of A.*

Proof. This is done by induction on k. If $k = 0$, t and t' are leaves, it follows from their $(0,m)$-type that $t = t'$.

Otherwise we know that t and t' have the same (k,m)-type at their root therefore they have the same root label. Let s and s' be two trees that are children of the root of t or of t' and have the same $(k-1,m)$-type at their root. The depth of s and s' is smaller than $k-1$, therefore by induction hypothesis s and s' evaluate to the same state of A. Now, because the roots of t and t' have the same (k,m)-type, for each $(k-1,m)$-type τ, they have the same number of children of type τ up to threshold m. From the previous remark this implies that for each state q of A, they have the same number of children in state q up to threshold m. It follows from the definition of A that t and t' evaluate to the same state of A. □

We are now ready to state and prove the lemma in the restricted case.

Claim 11.20. *Assume L is (k,l)-tame. Let t be a tree and let x, y be two nodes of t not related by the descendant relationship and with the same (k,l)-type. We write $s = t|_x$, $s' = t|_y$ and C the context such that $t = Cs$. If $s \preccurlyeq_{(k+1,l)} C$ then $Cs \in L$ iff $Cs' \in L$.*

Proof sketch. This proof only differs from its binary tree counterpart Claim 11.7 in the details of the induction step. It is done by induction on the depth of s.

Assume first that s is of depth less than k. Since x and y have the same (k,l)-type and since $l \geq m$ it follows from Claim 11.19 that s and s' evaluate the the same state on the automaton A recognizing L. Hence we can replace s with s' without affecting membership in L.

Assume now that s is of depth greater than k.

Let τ be the $(k+1,l)$-type of x. We write $s_1, ..., s_n$ the children of s and a the label of its root. Since $s \preccurlyeq_{(k+1,l)} C$, there exists a node z in C of type τ. We write $s'' = t|_z$.

We now do a case analysis depending on the descendant relationships between x, y and z. As for binary trees, all cases reduce to the case when x and z are not related by the descendant relationship by simple (k,l)-tameness operations. Therefore we only consider this case here.

Assume that x and z are not related by the descendant relationship. We show only that $Cs \in L$ iff $Cs'' \in L$. The proof that $Cs' \in L$ iff $Cs'' \in L$ is then done exactly as for binary trees.

Since x and z are of same $(k+1,l)$-type τ, the roots of s' and s'' have the same label a. Let $s''_1, \ldots, s''_{n'}$ be the children of the root of s'', like in the binary case we want to replace the trees $s_1, \ldots, s_n$ with these children by induction since the depth of the trees $s_1, \ldots, s_n$ is smaller than the depth of s. Unfortunately for each (k,l)-type τ_i, the number of trees whose root has type τ_i among the children of x and among the children of z might not be the same. However we know that in this case both numbers are greater than l. We overcome this difficulty in two steps, first we modify the children of x, without affecting membership in L, so that if s_i and s_j have the same (k,l)-type then $s_i = s_j$, then we use the fact that $l > m$ in order to delete or duplicate children of x until for each (k,l)-type τ_i the number of trees of root of type τ_i among the children of x and among the children of z is the same. By definition of A, this does not affect membership into L. Finally we replace the s_i by the s''_i by induction as in the binary case.

For the first step notice that any of the s_i is by definition of depth smaller than s therefore by induction hypothesis we can replace it with any of its siblings having the same (k,l)-type at its root without affecting membership in L. □

We now turn to the proof of Lemma 11.18 in its general statement. The proof is done by induction on the depth of s'. The idea is to replace s with s' node by node.

Assume first that s' is of depth smaller than k. Then because the (k,l)-types of the roots of s and s' are the same we are in the hypothesis of Claim 11.19 and it follows that s and s' evaluate to the same state of A. The result follows.

Assume now that s' is of depth greater than k.

Let x be the node of t corresponding to the root of s. Let τ be the $(k+1,l)$-type of the root of s'. In the binary tree case we used a sequence of tame operations to reduce the problem to the case where x has $(k+1,l)$-type τ. Using the same operations we can also reduce the problem to this case in the unranked setting. Then we use the induction hypothesis to replace the children of x by the children of the root of s'. Like in the proof of Claim 11.20, the problem is that the

number of children might not match but this is solved exactly as in the proof of Claim 11.20. □

As in the binary tree case, we now prove a result similar to Lemma 11.18 but for (k,l)-loops.

Lemma 11.21. *Assume L is (k,l)-tame. Let t be a tree and x a node of t of (k,l)-type τ. Let t' be another tree such that $t \simeq_{(k+1,l)} t'$ and C be a (k,l)-loop of type τ in t'. Consider the tree T constructed from t by inserting a copy of C at x. Then $t \in L$ iff $T \in L$.*

Proof sketch. The proof is done using the same structure as Lemma 11.8 for the binary case. First we use the (k,l)-tame property of L to show that we can insert a (k,l)-loop C' at x in t such that the principal path of C is the same as the principal path of C'. By this we mean that there is a bijection from the principal path of C' to the principal path of C that preserves the child relation and $(k+1,l)$-types. In a second step we replace one by one the subtrees hanging from the principal path of C' with the corresponding subtrees in C.

Let T' be the tree resulting from inserting C' at position x. We do not detail the first step as it is done using exactly the same sequence of tame operations we used for this step in the proof of Lemma 11.8. This yields: $t \in L$ iff $T' \in L$. We turn to the second step showing that $T' \in L$ iff $T \in L$.

By construction of T' we have $C' \preccurlyeq_{(k+1,l)} t$. Consider now a node x'_i in the principal path of C' and x_i the corresponding node in C. Like in the binary tree case we replace the subtrees branching out of the principal path of C' with the corresponding trees branching out of the principal path of C using Lemma 11.18. As in the previous proof, the problem is that the numbers of children might not match. This is solved exactly as in the proof of Lemma 11.18. □

We now turn to the construction of T and T' and prove Proposition 11.17.

The construction is similar to the one we did in the binary tree case, we insert (k,l)-loops in t and t' using Lemma 11.21 for obtaining bigger types. However inserting loops only affect the depth of the types. Therefore we need to do extra work in order to also increase the width of the types.

Assuming $t \simeq_{(k,l)} t'$ we first we construct two intermediate trees T_1 and T'_1 that have the following properties:

- $t \in L$ iff $T_1 \in L$
- $t' \in L$ iff $T'_1 \in L$
- $T_1 \simeq_{(\kappa',\lambda)} T'_1$

This construction is the same as in the binary tree setting so we only briefly describe it. Let $B = \{\tau_0, ..., \tau_n\}$ be the set of (k,l)-types τ such that there is a loop of (k,l)-type τ in t or in t'. For each $\tau \in B$ we fix a context C_τ as follows. Because $\tau \in B$ there is a context C in T_1 or T_1' that is a loop of (k,l)-type τ. For each $\tau \in B$, we fix arbitrarily such a C and set C_τ as $\underbrace{C \cdot \ldots \cdot C}_{\kappa'}$, κ' concatenations of the context C. Notice that the path from the root of C_τ to its port is then bigger than κ'.

T_1 is constructed from t as follows (the construction of T_1' from t' is done similarly). The tree T_1 is constructed by simultaneously inserting, for all $\tau \in B$, a copy of the context C_τ at all nodes of t of type τ. By Lemma 11.21 it follows that $t \in L$ iff $T_1 \in L$ and $t' \in L$ iff $T_1' \in L$. Using the same proof as of Proposition 11.5 for the binary tree setting, we obtain $T_1 \simeq_{(\kappa',\lambda)} T_1'$.

We now describe the construction of T from T_1, the construction of T' from T_1' is done similarly. It will be convenient for us to view the nodes of T_1 as the union of the nodes of t plus some extra nodes coming from the loops that were inserted.

Let n be the maximum arity of a node of T_1 or of T_1'. We duplicate subtrees in T_1 and T_1' as follows. Let x be a node of T_1, that is not in a loop we inserted when constructing T_1 from t. For each $(\kappa'-1,\lambda)$-type τ, if x has more than λ children of type τ we duplicate one of the corresponding subtrees until x has exactly n children of type τ in total. This is possible without affecting membership into L because $\lambda > m|A|$. Indeed, because $\lambda > m|A|$, for at least one state q of A, there exists more than m subtrees of x of type τ for which A assigns that state q at their root, and by definition of A any of these subtrees can be duplicated without affecting membership into L. The tree T is constructed from T_1 by repeating this operation for any node x of T_1 coming from t. By construction we have $T_1 \in L$ iff $T \in L$ and therefore $t \in L$ iff $T \in L$. The same construction starting from T_1' yields a tree T' such that $t' \in L$ iff $T' \in L$.

We now show that $T \simeq_{\kappa'} T'$, it follows that $T \simeq_{(\kappa',\lambda')} T'$ and this concludes the proof.

Lemma 11.22. $T \simeq_{\kappa'} T'$

Proof. We need to show that $T \preccurlyeq_{\kappa'} T'$, $T' \preccurlyeq_{\kappa'} T$ and that the roots of T and T' have the same κ'-type.

Recall that in T_1 we distinguished between two kinds of nodes, those coming from t and those coming from the loops that were inserted during the construction of T_1 from t. We do the same distinction in T by assuming that a node generated after a duplication gets the same kind as its original copy.

Recall the definition of B and of C_τ for $\tau \in B$ that was used for defining T_1 and T_1' from t and t'.

As for the binary tree case it suffices to show that for any node of T coming from t there is a node of T' coming from t' and having the same κ'-type. Hence the result follows from the claim below that is an adaptation of Claim 11.11.

Claim 11.23. *Take two nodes x in t and x' in t', such that x and x' have the same (κ, λ)-type. Let z and z' be the corresponding nodes in T and T'. Then z and z' have the same κ'-type.*

Proof. Let x and x' be two nodes of t and t' with the same (κ, λ)-type. Let x_1 and x'_1 be the corresponding nodes in T_1 and T'_1. The same proof as Claim 11.11 for the binary tree case shows that x_1 and x'_1 have the same (κ', λ)-type.

Let y be a child of x. Let y_1 be the node corresponding to y in T_1. Notice now that the (κ', λ)-type of y_1 in T_1 is completely determined by the $(\kappa - 1, \lambda)$-type ν of y in t. Indeed, by choice of κ, during the construction of T_1, a loop of type $\tau \in B$ will be inserted between y and any descendant of y at distance at most $\beta_{(k,l)} - 1$ from y. As $\kappa > \beta_{(k,l)} + k$, the relative positions below y where such a C_τ is inserted can be read from ν. As the depth of any C_τ is greater than κ', from ν we can compute exactly the descendants of y_1 in T_1 up to depth κ'. Hence ν determines the (κ', λ)-type of y_1.

It follows that two children of x_1 or of x'_1 have the same (κ', λ)-types iff they had the same $(\kappa - 1, \lambda)$-types in t or in t'.

We now construct an isomorphism between the κ'-type of z and the one of z'. Let d be the maximal distance between z and a node that is a descendant of z where a loop was inserted during the construct of T from t. We construct our isomorphism by induction on d.

If $d = 0$ then the (k, l)-type of z is in B and as z and z' have the same (κ', λ)-type with $\kappa' > k$, the (k, l)-type of z' is the same as the one of z'. Therefore the subtrees rooted at z and z' are equal up to depth κ' as they all start with a copy of C_τ and we are done.

Otherwise, as z and z' have the same (κ', λ)-type their roots must have the same labels. Consider now a $(\kappa' - 1, \lambda)$-type μ. By construction of T and T', z and z' must have the same number of occurrences of children of type μ. Indeed from the type these numbers must match if one of them is smaller than λ and by construction they are equal to n otherwise. Hence we have a bijection from the children of z of type μ and the children of z' of type μ. From the text above we know that the (κ', λ)-type of these nodes is determined by the $(\kappa - 1, \lambda)$-type of their copy in t or in t'. Because x and x' have the same (κ, λ)-type, the corresponding $(\kappa - 1, \lambda)$-types are all equal and hence all the nodes of type μ actually have the same (κ', λ)-type. By induction they are isomorphic up to depth κ' and we are done. □

From Claim 11.22 the lemma follows as in the proof of Lemma 11.10 for binary trees. □

This concludes the proof of Proposition 11.17. □

11.3.2 Decision of ILT

In the idempotent case we can completely characterize ILT using closure properties. We show that membership in ILT corresponds to tameness together with an extra closure property denoted *horizontal stutter* reflecting the idempotent behavior. A tree language L is closed under horizontal stutter iff for any tree t and any node x of t, replacing $t|_x$ with two copies of $t|_x$ does not affect membership into L. Theorem 11.12 for ILT is a consequence of the following theorem.

Theorem 11.24. *A regular unordered tree language is in* ILT *iff it is tame and closed under horizontal stutter.*

Proof. It is simple to see that tameness and closure under horizontal stutter are necessary conditions. We prove that they are sufficient. Take a regular tree language L and suppose that L is tame and closed under horizontal stutter. Then there exists k and l such that L is (k, l)-tame. We show that if $t \simeq_{(k+1,1)} t'$ then $t \in L$ iff $t' \in L$. It follows that L is in ILT. We first show a simple lemma stating that if two trees contain the same $(k+1, 1)$-types, then we can pump them without affecting membership in L into two trees that contain the same $(k + 1, l)$-types.

Lemma 11.25. *Let L closed under horizontal stutter and let s and s' two trees such that $s \simeq_{(k+1,1)} s'$. Then there exists two trees S and S' such that:*

- *$s \in L$ iff $S \in L$.*
- *$s' \in L$ iff $S' \in L$.*
- *$S \simeq_{(k+1,l)} S'$*

Proof. S is constructed from s via a bottom-up procedure. Let x be a node of s. For each subtree rooted at a child of x, we duplicate it l times using horizontal stutter. This does not affect membership into L. After performing this for all nodes x of s we obtain a tree S with the desired properties. □

Let T and T' be constructed from t and t' using Lemma 11.25. Let $T_1, \ldots, T_n$ the children of the root of T and $T'_1, \ldots, T'_{n'}$ the children of the root of T'. Let T'' be the tree whose root is the same as T and T' and whose children is the sequence of trees $T_1, \ldots, T_n, T'_1, \ldots, T'_{n'}$. We show that $T'' \in L$ iff $T \in L$ and $T'' \in L$ iff

$T' \in L$. It will follow that $T \in L$ iff $T' \in L$ and by Lemma 11.25 that $t \in L$ iff $t' \in L$ which ends the proof.

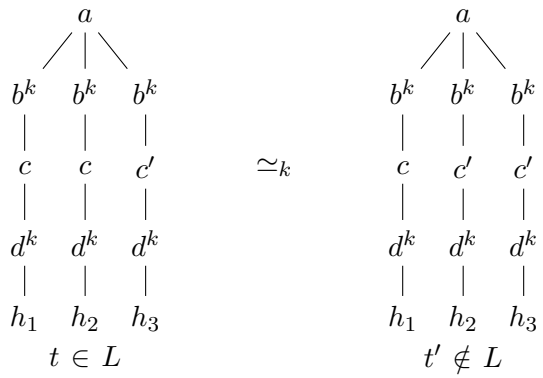
To show that $T'' \in L$ iff $T \in L$ we use horizontal stutter and Lemma 11.18. As the roots of T and T' have the same $(k+1, l)$-type, for each T'_i, there exists a tree T_j such that its root has the same (k, l)-type as T'_i. Fix such a pair (i, j). Let S be the tree obtained by duplicating T_j in T, by closure under horizontal stutter $T \in L$ iff $S \in L$. But now $S = DT_j$ for some context D such that $T \preccurlyeq_{(k+1,l)} D$. Altogether we have that: the roots of T'_i and T_j have the same (k, l)-type (by choice if i and j), $T'_i \preccurlyeq_{(k+1,l)} D$ (as $T'_i \preccurlyeq_{(k+1,l)} T'$ and $T \simeq_{(k+1,l)} T'$) and $T_j \preccurlyeq_{(k+1,l)} D$ (as T_j is part of T hence of D). We can therefore apply Lemma 11.18 and $DT'_i \in L$ iff $DT_j \in L$.

Repeating this argument for all i eventually yield the tree T''. This proves that $T'' \in L$ iff $T \in L$. By symmetry we also have $T'' \in L$ iff $T' \in L$ which concludes the proof. □

11.4 Tameness is not sufficient

Over strings tameness characterizes exactly LT as vertical swap and vertical stutter are exactly the extensions to trees of the known equations for LT Over trees this is no longer the case, in this section we provide an example of a language that is tame but not LT. For simplifying the presentation we assume that nodes may have between 0 to three children; this can easily be turned into a binary tree language. All trees in our language L have the same structure consisting of a root of label **a** from which exactly three sequences of nodes with only one child (strings) are attached. The trees in L have therefore exactly three leaves, and those must have three distinct labels among $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$. The labels of two of the branches, not including the root and the leaf, must form a sequence in the language $\mathbf{b}^*\mathbf{cd}^*$. The third branch must form a sequence in the language $\mathbf{b}^*\mathbf{c'd}^*$. We assume that **b**, **c**, **c'** and **d** are distinct labels. Note that the language does not specify which leaf label among $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\}$ is attached to the branch containing **c'**.

The reader can verify that L is 1-tame. We show that L is not in LT. For all integer k, the two trees t and t' depicted below are such that $t \in L$, $t' \notin L$, while $t \simeq_k t'$.



11.5 Discussion

We have shown a decidable characterization for the class of locally testable regular tree languages both for ranked trees and unranked unordered trees.

Complexity The decision procedure for deciding membership in LT as described in this thesis requires a time which is a tower of several exponentials in the size of the deterministic minimal automaton recognizing L. This is most likely not optimal. In comparison, over strings, membership in LT can be performed in polynomial time [Pin05]. Essentially our procedure requires two steps. The first step shows that if a regular language L is in LT then it is κ-locally testable for some κ computable from the minimal deterministic automaton A recognizing L. The κ obtained in Proposition 11.4 is doubly exponential in the size of A. In comparison, over strings, this κ can be shown to be polynomial. For trees we did not manage to get a smaller κ but we have no example where even one exponential would be necessary.

Our second step tests whether L is κ-locally testable once κ is fixed. This is easy to do using a brute force algorithm that requires several exponentials in κ. It is likely that this can be optimized but we didn't investigate this direction.

However for unranked unordered trees we have seen in Theorem 11.24 that in the case of ILT it is enough to test for tameness. The naive procedure for deciding tameness is exponential in the size of A. But the techniques presented in [BS09] for the case of LTT, easily extend to the closure properties of tameness, and provide an algorithm running in time polynomial in the size of A. Hence membership in ILT can be tested in time polynomial in the size of of the minimal deterministic bottom-up tree automaton recognizing the language.

Open problem In the case of unranked *ordered* trees, we believe that tameness together with a property that essentially say that the horizontal navigation is in

LT should provide a characterization for an intuitive notion of LT. Note that in this setting it is no longer clear whether tameness is decidable or not. We leave the case of ordered trees as an open problem.

Conclusion

Our main goal in this thesis is to get a better understanding of the expressive power of first-order logic in the setting of trees. In particular, the main open problem in this research area is to obtain decidable characterizations for $\mathrm{FO}(<_{\mathbf{v}})$ and $\mathrm{FO}(<_{\mathbf{h}}, <_{\mathbf{v}})$ using the ancestor and following sibling relations. We leave this main problem open. However, we were able to obtain decidable characterizations for several fragments of FO. This fragments are obtained by restricting the expressive power of FO in specific ways. First we presented decidable characterizations for logics obtained by restricting the quantifications, such as $\Delta_2(<_{\mathbf{v}})$, or the number variables allowed, such as $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. Also, we provided a decidable characterization of LT which, in particular, restricts the predicates that may be used in FO to only the successor relation.

A question that is raised from these results is why we are still unable to obtain a decidable characterization for FO. A good place to look would be to investigate which algebraic formalism is the right one for this purpose. We already used several formalisms and it is not cleat that they are powerful enough for FO. Moreover, none of them unify all the other formalisms. For instance, we are still unable to express the tameness property we use for the characterization of LT, using forest algebras. Another example is the notion of closure under saturation we use in the characterization of $\mathrm{FO}^2(<_{\mathbf{h}}, <_{\mathbf{v}})$. This property involves several parameters that are outside the syntactic forest algebra of the language. In particular, we do know how to express this property using only the syntactic forest algebra of the language.

This leads to two possible research directions. First, our knowledge of forest algebra might be insufficient. We would need to further investigate this formalism and devise new tools in order, for instance, to be able express properties such as saturation.

Another possible direction would be the definition of a new algebraic formalism that would be better suited for expressing our closure properties. For example saturation relies heavily on branching in the trees. An adapted formalism, should be able to express these branching properties in a simple way. Another possibility for finding a new formalism is to investigate the expressive power of fragments of

FO that are slightly more expressive than the logics for which we already have decidable decidable characterizations. For example obtaining a decidable characterization for $\mathrm{FO}^2(<_{\mathbf{h}}, Succ_{\mathbf{h}}, <_{\mathbf{v}}, Succ_{\mathbf{v}})$ would involve obtaining a better understanding on the child relation in trees. Therefore, it could lead to the definition of an algebra that would be better in expressing this relation and could be used to express closure properties for fragments such as LT. Similarly, studying a notion of locally testable languages of unranked ordered trees, may lead to a better understanding on how to express horizontal closure properties. Another motivation for searching for a new formalism is to unify all known formalism in a simple way.

Bibliography

[Alm96] Jorge Almeida. A syntactical proof of locality of **DA**. *International Journal of Algebra and Computation*, 6:165–177, 1996.

[Boj07a] Mikołaj Bojańczyk. A new algorithm for testing if a regular language is locally threshold testable. *Information Processing Letters*, 104(3):91–94, 2007.

[Boj07b] Mikołaj Bojańczyk. Two-way unary temporal logic over trees. In *Proceedings of the 22th Annual IEEE Symposium on Logic in Computer Science (LICS'07)*, pages 121–130, 2007.

[BP89] Danièle Beauquier and Jean-Éric Pin. Factors of words. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP'89)*, pages 63–79, 1989.

[BS73] Janusz A. Brzozowski and Imre Simon. Characterizations of locally testable languages. *Discrete Mathematics*, 4:243–271, 1973.

[BS08] Mikołaj Bojańczyk and Luc Segoufin. Tree languages defined in first-order logic with one quantifier alternation. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP'08)*, pages 233–245, 2008.

[BS09] Michael Benedikt and Luc Segoufin. Regular Languages Definable in FO and FOmod. *ACM Transactions of Computational Logic*, 11(1), 2009.

[BSS08] Mikołaj Bojańczyk, Luc Segoufin, and Howard Straubing. Piecewise testable tree languages. In *Proceedings of the 23th Annual IEEE Symposium on Logic in Computer Science (LICS'08)*, pages 442–451, 2008.

[BSW09] Mikołaj Bojańczyk, Howard Straubing, and Igor Walukiewicz. Wreath products of forest algebras, with applications to tree logics. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science (LICS'09)*, pages 255–263, 2009.

[BW06] Mikołaj Bojańczyk and Igor Walukiewicz. Characterizing EF and EX tree logics. *Theoretical Computer Science*, 358, 2006.

[BW07] Mikołaj Bojańczyk and Igor Walukiewicz. Forest algebras. In *Automata and Logic: History and Perspectives*, pages 107–132. Amsterdam University Press, 2007.

[Bü60] Julius Richard Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

[DG08] Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, volume 2, pages 261–306. Amsterdam University Press, 2008.

[DGK08] Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words. *International Journal of Foundations of Computer Science*, 19(3):513–548, 2008.

[ÉI08a] Zoltán Ésik and Szabolcs Iván. Products of tree automata with an application to temporal logic. *Fundamenta Informaticae*, 82:61–78, 2008.

[ÉI08b] Zoltán Ésik and Szabolcs Iván. Some varieties of finite tree automata related to restricted temporal logics. *Fundamenta Informaticae*, 82:79–103, 2008.

[Ési06] Zoltán Ésik. Characterizing CTL-like logics on finite trees. *Theoretical Computer Science*, 356:136–142, 2006.

[EVW02] Kousha Etessami, Moshe Y. Vardi, and Thomas Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

[ÉW05] Zoltán Ésik and Pascal Weil. Algebraic characterization of regular tree languages. *Theoretical Computer Science*, 340:291–321, 2005.

[Mar05] Maarten Marx. First order paths in ordered trees. In *Proceedings of the 10th International Conference in Database Theory (ICDT'05)*, pages 114–128, 2005.

[McN74] Robert McNaughton. Algebraic decision procedures for local testability. *Mathematical Systems Theory*, 8:60–76, 1974.

[Pin96] Jean-Éric Pin. Logic, semigroups and automata on words. *Annals of Mathematics and Artificial Intelligence*, 16:343–384, 1996.

[Pin05] Jean-Éric Pin. The expressive power of existential first order sentences of büchi's sequential μ calculus. *Discrete Mathematics*, 291:155–174, 2005.

[Pla08] Thomas Place. Characterization of logics over ranked tree languages. In *Proceedings of the 17th Annual EACSL Conference on Computer Science Logic (CSL'08)*, pages 401–415, 2008.

[PS09] Thomas Place and Luc Segoufin. A decidable characterization of locally testable tree languages. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP'09)*, pages 285–296, 2009.

[PS10] Thomas Place and Luc Segoufin. Deciding definability in $FO_2(<)$ (or XPath) on trees. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS'10)*, pages 1–1, 2010.

[PW97] Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30, 1997.

[Sch65] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8, 1965.

[Sim75] Imre Simon. Piecewise testable events. *Automata Theory and Formal Languages*, pages 214–222, 1975.

[Str85] Howard Straubing. Finite semigroup varieties of the form V*D. *Journal of Pure and Applied Algebra*, 36(1):53–94, 1985.

[Str10] Howard Straubing. Forest categories. *Unpublished manuscript*, 2010.

[Til87] Bret Tilson. Categories as algebra: an essential ingredient in the theory of monoids. *Journal of Pure and Applied Algebra*, 48:83–198, 1987.

[TT02] Pascal Tesson and Denis Thérien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages*, pages 475–500, 2002.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2, 1968.

[TW85] Denis Thérien and Alex Weiss. Graph congruences and wreath products. *Journal of Pure and Applied Algebra*, 36:205–215, 1985.

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC'98)*, pages 234–240, 1998.

[Wil96] Thomas Wilke. An algebraic characterization of frontier testable tree languages. *Theoretical Computer Science*, 154(1):85–106, 1996.