Warsaw University
Faculty of Mathematics, Informatics and Mechanics

Mikołaj Bojańczyk

Decidable Properties of Tree Languages

PhD Thesis

Supervisor
dr hab. Igor Walukiewicz
CNRS,
Laboratoire Bordelais de Recherche en Informatique,
Bordeaux, France

June, 2004

Abstract

The first part of the thesis concerns problems related to the question: "when can a regular tree language be defined in first-order logic?" Characterizations in terms of automata of first-order logic and the related chain logic are presented. A decidable property of tree automata called confusion is introduced; it is conjectured that a regular tree language can be defined in chain logic if and only if its minimal automaton does not contain confusion. Furthermore, polynomial time algorithms are presented that decide if a given regular tree language can be defined in any one of the temporal branching logics TL[EX], TL[EF] and TL[EX, EF].

In the second part of the thesis, an extension MSOL+$\mathbb{B}$ of monadic second-order logic over infinite trees is considered, where a new quantifier $\mathbb{B}$ is added. Using this quantifier, one can express properties such as: "there exist bigger and bigger sets satisfying ..." An automata-theoretic investigation of the quantifier is conducted, yielding decidable satisfiability for two fragments of MSOL+$\mathbb{B}$. These results are then applied to a decision problem stemming from the μ-calculus.

Keywords: Tree automata, definability, monadic second-order logic, μ-calculus, temporal logic.

ACM Classification: D.2.4, F.1.1, F.3.1

Contents

Chapter 1

Introduction

This thesis concerns issues on the boundary of logic and the theory of tree automata. It consists of two parts. The first concerns the definability problem for languages of finite trees. The subject of the second part is an extension of monadic second-order logic by a cardinality quantifier.

The first part includes Chapters 2 and 3 and is concerned with finite trees. In it, we investigate what structural properties of tree automata correspond to certain logically defined classes of tree languages. The aim is to produce algorithms that, given as input a tree automaton, decide if the corresponding tree language can be defined in a particular logic. The logics considered are first-order logic, chain logic and several simple branching temporal logics. For the branching temporal logics, the aim of providing algorithms is achieved, while only partial progress is made in the cases of chain and first-order logic.

The second part of the thesis, consisting of Chapters 4 and 5, is devoted the study of a new quantifier that extends monadic second-order logic over infinite trees with the ability to say that certain families of sets contain sets of unbounded size. An investigation into automata theoretic aspects of this quantifier is conducted, yielding algorithms for satisfiability of formulas that use the quantifier in a certain restricted fashion. These results are applied to solve the decision problem: "given a formula of the modal μ-calculus with backward modalities, decide if it can be satisfied in some finite structure."

The rest of this introduction is meant to be a cursory summary of both parts of the thesis. However, before we proceed to the subject of the first part – the definability problem for tree languages – we begin with a short detour into word languages.

An established decision problem for regular word languages is the *definability problem*, which for a given class $\mathcal{X}$ of regular word languages is defined as follows:

Decide if an input regular language belongs to the class $\mathcal{X}$.

A celebrated problem of this kind concerns the class of star-free languages, i.e. regular word languages that can be defined by a regular expression without the Kleene star (but with complementation). This class was characterized by Schützenberger in the groundbreaking paper [53], in which it is shown that a language is star-free if and only if its syntactic semigroup is aperiodic, i.e. does not contain a nontrivial group. Since the syntactic semigroup of a regular word language is a finite object that can be effectively computed, Schützenberger's theorem implies that the definability problem for star-free languages is decidable. For instance, the syntactic semigroup of the language $(aa)^*$ is the two-element group $\mathbb{Z}_2$, and therefore this language is not star-free.

The definability problem seems particularly interesting, however, when a logical approach to regular languages is considered. In this approach, a word is represented as an appropriate relational structure. The domain of this structure is the set of positions in the word, which is assumed to be linearly ordered, and for every possible letter there is a corresponding unary predicate. For instance, the structure appropriate for the word *abba* is:

$$\langle \quad \{1,2,3,4\} \quad , \quad \leq \quad , \quad \underline{a} = \{1,4\} \quad , \quad \underline{b} = \{2,3\} \quad \rangle \ .$$

Using this representation, with a logical formula one may identify the language of words which satisfy it; for instance, the formula $\forall x.a(x)$ corresponds to the language a^*.

A fundamental result here is the Büchi Theorem [9] (proved also, independently, by Elgot [14] and Trakhtenbrot [66]), which says that a word language is regular if and only if it can be defined in monadic second-order logic. This logic is an extension of first-order logic where quantification over sets of elements (in this case, sets of positions in the word) is also allowed.

By this theorem, every sublogic of monadic second-order logic corresponds to a subclass of the class of regular languages. Logic becomes thus a natural source for an abundance of definability problems. A particularly notable example is:

Is a given regular word language definable in first-order logic?

By a result of McNaughton and Papert [40], a word language is first-order definable if and only if it is star-free, therefore the above problem is decidable thanks to Schützenberger's theorem.

This result has opened up a thriving research field [58, 47]. Among the many other logics for which the definability problem is decidable are first-order logic limited to two variables [60] and first-order logic with $\leq$ replaced by the successor relation [61]. A slightly different setting is provided by temporal logic, where definability has been shown decidable for various language

classes defined by limiting the type or nesting of modalities allowed [70]. Although intensive research continues, one can safely say that we have by now attained a solid understanding of the relationship between logic and regular word languages.

Unfortunately, the same cannot be said for tree languages. Much theory of regular languages carries over to the tree case in a straightforward manner, including the correspondence between monadic second-order logic and regular languages [59]. In the subject of definability, however, instances of progress have been few and far between. In particular, decidability of the problem:

Is a given regular tree language definable in first-order logic?

remains, despite attempts at resolution, open to this day. In a sense, Chapter 2 of this thesis is a chronicle of one such failed attempt.

Much of the success in the word case has been due to automata-theoretic and algebraic methods. It seems that this is also the way to go in the tree case. That is why, in Chapter 2, we devote our energies to the development and analysis of two such tools: wordsum automata and cascade product of tree automata.

A wordsum automaton is a bottom-up tree automaton which runs a word automaton on all the branches of an input tree. It calculates the set of possible states that the underlying word automaton can assume after reading *some* branch of the tree. In particular, by looking at the state of the wordsum automaton one can determine if all (or some) branches of the tree are accepted by the underlying word automaton.

Cascade product, on the other hand, is an operation that takes two tree automata $\mathcal{A}$ and $\mathcal{B}$ and returns a new tree automaton $\mathcal{B} \circ \mathcal{A}$. This automaton runs on an input tree first the automaton $\mathcal{A}$ and then lets the automaton $\mathcal{B}$ read both the labeling of the original tree and the states assumed in the run of $\mathcal{A}$.

By using the notions of wordsum automaton and cascade product, we obtain several characterizations of logically defined language classes. One of these says that a tree language is first-order definable if and only if it is recognized by a cascade product of aperiodically wordsum automata (these are wordsum automata whose underlying word automaton corresponds to a first-order definable word language). Other characterizations are related to a hierarchy of formulas within the branching temporal logic CTL* .

In the second part of Chapter 2, we turn our attention to a tree logic that sits between first-order logic and monadic second-order logic: chain logic. This logic, introduced by Thomas in [62], is obtained from monadic-second order logic by limiting set quantification to chains. We consider the definability problem for chain logic and, although failing to prove it decidable, we do

suggest a conjecture. This conjecture states that a language is definable in chain logic if and only if it is nonconfusing, i.e. is recognized by an automaton that does not contain a special type of pattern (called confusion). At the very least, confusion is a convenient method of showing that a language is not definable in chain logic, since a chain-definable language can be proved not to contain confusion.

The confusion conjecture also admits an interesting application of the cascade product: it is shown that the class of automata that do not contain confusion is closed under homomorphic image and cascade product. This closure result implies, in particular, that nonconfusing languages are closed under chain quantification and the boolean operations.

In the next chapter, research on the definability problem is continued. We cut back our ambitions a bit, and instead of first-order logic, we consider temporal logics where only two operators are allowed: EF, corresponding to "there exists a node below the current one"; and EX, corresponding to "there exists a successor". For instance, the formula

$$\mathsf{EF}(\mathsf{EX}a \wedge \mathsf{EX}b)$$

is satisfied in trees that contain a node with one successor labeled by a and the other successor labeled by b. Three logics are considered. These are called TL[EX], TL[EF] and TL[EX, EF] and account for the three possible combinations of allowing and disallowing the operators EF and EX. For each of these logics, the definability problem is researched.

Consider first the definability problem for the logic TL[EX]. It turns out that a language recognized by a tree automaton is TL[EX]-definable if and only if the acceptance of a tree by this automaton depends only upon nodes whose distance from the root is bounded by some fixed threshold. Since this threshold, if it exists, can be bounded by the size of the automaton, the definability problem is decidable for TL[EX].

The characterization above is rather straightforward and has already been known in the literature for some time [46]. The original contribution in this thesis are characterizations for the other two logics TL[EF] and TL[EX, EF], which show the corresponding definability problems to be decidable. In fact, if the input automaton is assumed to be deterministic, the appropriate algorithms run in polynomial time.

This concludes the first part of the thesis, which is about the definability problem. In the first part the trees are assumed to be finite and the logics we considered are fragments of monadic second-order logic. In the second part, we shift our attention to infinite trees. The logic concerned is still monadic second-order logic, but this time we are more interested in its extensions rather than its fragments.

A fundamental result of Rabin [50] says that a language of infinite trees is regular if and only if it is definable in monadic second-order logic. Due to the expressive power of monadic second-order logic over infinite trees, Rabin's theorem has been the basis for many decidability results [17, 63, 64, 27]. There are properties, however, that cannot be defined even in this powerful logic. Consider, for instance, the following one:

> There are chains of a-labeled nodes of any finite size, but none of infinite size.

The second part of this property – the absence of an infinite chain of a-labeled nodes – can easily be expressed in monadic second-order logic over infinite trees. This is in contrast with the first part though, which can be shown to be inexpressible using monadic-second order logic. This is really a shame, since cardinality constraints such as the one above have several interesting applications (some of which are discussed in Chapter 4).

For this reason, we consider an extension MSOL+$\mathbb{B}$ of monadic second-order logic, where a new second-order quantifier $\mathbb{B}$ is added. In this extension, a formula $\mathbb{B}X.\psi(X)$ is satisfied in the trees that admit a finite bound on the possible size of sets satisfying $\psi(X)$.

In the particular example regarding chains of a-labeled nodes considered above, an appropriate MSOL+$\mathbb{B}$ sentence would be the conjunction

$$\varphi \quad = \quad \neg\mathbb{B}X.\ \psi(X) \quad \wedge \quad \forall X.(\psi(X) \Rightarrow \exists x \in X\ \forall y \in X.\ y \leq x)$$

where the formula $\psi(X)$ says that X is a chain of a-labeled nodes:

$$\psi(X) = \forall x, y \in X.\ [a(x) \wedge (x \leq y \vee y \leq x)]$$

In the sentence φ, the first conjunct says that no finite bound on the size of a-labeled chains can be found, while the second one says that every such chain has a maximal element and is therefore finite.

In Chapter 4, we embark on an investigation of the logic MSOL+$\mathbb{B}$. The decision problem considered here is satisfiability, i.e. the question whether a given formula is satisfied in some tree. Although we fail to prove satisfiability decidable for all of MSOL+$\mathbb{B}$, we manage to identify two fragments where the problem is decidable. These two fragments are interesting enough to include several important applications (and the formula φ above).

One of these applications is the subject of the next and final chapter of the thesis, Chapter 5, where the modal μ-calculus with backward modalities [67, 36, 3] is considered. A characteristic trait of this logic is that it does not have the finite model property: it defines formulas which are satisfiable

only in infinite structures. What makes the modal μ-calculus with backward modalities remarkable, however, is that – contrary to most other formalisms without a finite model property such as first-order logic over arbitrary relational structures – it is a computationally manageable logic. In particular, it is known to have decidable satisfiability [67]. These considerations stimulate us to consider the following decision problem:

> Is a given formula of the modal μ-calculus with backward modalities satisfiable in some finite structure?

Instead of working directly with the μ-calculus, we use alternating two-way automata over graphs. This is an automata formalism [56, 43] closely related to the μ-calculus. In particular, for every formula of the modal μ-calculus with backward modalities there exists an alternating two-way automaton that accepts exactly the same graphs [43, 42, 67]. Therefore the finite satisfiability problem for the μ-calculus mentioned above can be reduced to the analogous question for two-way alternating automata on graphs.

Although the alternating two-way automata we use work over arbitrary graphs, in the proof we limit our attention to graphs which resemble trees, i.e. two-way tree unravelings. An alternating two-way automaton cannot distinguish between a graph and its two-way tree unraveling, therefore one can consider only the latter type of graphs where all sorts of tree-automata techniques can be applied.

However, the tree unraveling of a graph is necessarily an infinite tree. The question is: how can we tell, looking at this infinite tree, if it originated from a finite graph? It turns out that this can be done using a cardinality-related property definable in MSOL+$\mathbb{B}$: a tree can be "wound back" into a finite structure if and only if there is a finite bound on the cardinality of certain paths. What is more, the appropriate MSOL+$\mathbb{B}$ sentence belongs to one of the fragments from Chapter 4 that have decidable satisfiability, and therefore the finite satisfiability problem for the two-way μ-calculus is shown to be decidable.

We would like to conclude this introduction with some remarks on the structure of the thesis. Each of the chapters is fairly self-contained; a reader with some background in tree automata can probably read them separately (perhaps an exception here is Chapter 5 which relies on some concepts from Chapter 4). There is no global "Preliminaries" section; definitions are introduced more or less where they are first used. This implies that definitions of the most basic concepts are concentrated in the first sections of the next chapter. Finally, what may well be the most interesting part of the thesis, is not what it contains, but what it does *not* contain. For this reason, at the

end of each chapter an attempt is made to summarize the problems which are left open and invite further research.

Acknowledgements. First and foremost, I would like to thank my supervisor Igor Walukiewicz. I am only slowly beginning to realize the extraordinary extent of his help; both the quantity and quality of which have burdened me with a debt that I will never repay. In particular, the next two chapters are simply the result of our collaboration. I would also like to thank Damian Niwiński and Thomas Colcombet for their helpful comments on draft versions of the thesis.

Chapter 2

Cascade Products of Tree Automata

2.1 Introduction

Many logical formalisms developed for branching structures can be defined as fragments of monadic second-order logic over trees. This is the case for the temporal logic CTL* [19, 16], and consequently all of its fragments such as CTL [11]. The same also goes for first order-logic and its fragments, such as the restriction to two variables and the restriction which only uses the successor relation.

With such a large variety of logical formalisms, one would like to understand how they are related to each other in terms of expressibility. We would like to answer questions such as: "can a given tree language be defined in first-order logic?" or "what type of properties cannot be defined in CTL*?" A good answer to such a question is an algorithm deciding if a tree language is definable in a particular logic. In such a case the *definability problem* for the logic is said to be decidable. But less definitive answers may be of some value, too – for instance one might be interested in providing sufficient conditions for a language to be *not* definable in a given logic.

This is the subject that we will undertake in this chapter and the next one. We shall try to characterize some tree logics within the framework of regular tree languages. In this chapter, we tackle "expressive" logics, such as first-order logic or chain logic. We present characterizations of these two logics, using a version of cascade product for tree automata. However, these results do not yield any algorithms for deciding first-order or chain definability, although a conjecture pertinent to the latter is presented. In Chapter 3, on the other hand, we consider much simpler logics, obtained by

using only the branching temporal modalities EF and EX. The simplicity of the subject matter is traded for stronger results: effective procedures are presented for deciding if a regular tree language can be defined in any of the logics TL[EX], TL[EF] and TL[EX, EF].

Due to the remarkable robustness of regular languages, the input to a procedure for the definability problem – a regular tree language – can be presented in one of many ways: by a formula of monadic second-order logic, by one of a number of equivalent variants of tree automata, by a regular expression, etc. Although effective procedures exist which translate between these formalisms, one should seek a framework that is convenient for studying and formulating structural properties. For this, we search for inspiration in a wide body of work done on analogous problems for word languages.

The definability problem for word languages has a rich and eventful history. A celebrated early result is Schützenberger's Theorem [53], which says that a word language is star-free if and only if its syntactic semigroup is aperiodic, i.e. does not contain a nontrivial group. This, together with McNaughton's and Papert's [40] result that star-free word languages are exactly the first-order definable ones, yields a decidable characterization of first-order logic over finite words. Decidable characterizations are also known for, to name but a few: linear temporal logic, its fragments obtained by restricting the types of modalities used or the nesting that is allowed; also restrictions of first-order logic where only the successor relation is used (or only two variables), etc. We direct the reader to the surveys [58, 47, 70] for more information. The important point here is that all of these characterizations are in terms of properties of the syntactic semigroup, or – more or less equivalently – of the minimal deterministic automaton.

The tree analogue to the syntactic semigroup seems to be the *syntactic algebra*, where binary operators are allowed (in the case of binary trees). On the automata side, the tree analogue to a deterministic finite automaton seems to be a *bottom-up* deterministic tree automaton. Of the two approaches, we choose here to work with the automata one, although an algebraic framework would yield the same results.

For bottom-up tree automata, cartesian product, determinization and projection work the same way as for word automata. We would venture to say, however, that the most important analogous property is the existence of *syntactic automata*. As in the word case, for every regular tree language L there is a unique minimal bottom-up automaton recognizing it, which can be treated as a canonical representation of L. This syntactic automaton is a close analogue to the one in the case of words: it is a homomorphic image of every other automaton recognizing L, its states represent equivalence classes of the context-indistinguishability relation, etc.

We start with first-order logic and consider the question [62, 31, 48]:

> What property of the syntactic automaton corresponds to first-order definability of a regular tree language?

This is a notoriously difficult question and we immediately dispel any undue expectations: no serious progress is made regarding decidability. We do, however, propose an automata-theoretic framework for the problem, which puts several previous results in a fresh perspective and opens new avenues of research.

As might be expected, tree languages bring new difficulties which did not appear in the word case. One the one hand, some languages cannot be defined in first-order logic despite satisfying a rough tree equivalent of aperiodicity, cf. boolean expressions evaluating to true. On the other hand, languages which posses periodic-like behaviour can be defined in first-order logic, using the auxiliary structure provided by a binary tree. An example of the second type is presented in Section 2.3.3: the set of trees where every path is of even length. The three-state syntactic automaton of this language seems – at first glance – to use "periodic" counting modulo two. However, the language can be shown to be first-order definable.

Instead of working with first-order formulas directly, we use the result of Hafer and Thomas [30] that over binary trees, first-order logic defines the same class of languages as CTL* [19, 16]. Within CTL* we distinguish a hierarchy $\Pi_1, \Sigma_1, \Pi_2, \Sigma_2, \ldots$ accounting for the nesting depth of the E operator. We then set out to relate this hierarchy with tree automata.

We start in Section 2.4 with some simple results regarding low levels of this hierarchy, presenting decidable characterizations of languages in Π_1 and Σ_1. These are languages recognized by deterministic (respectively codeterministic) top-down automata where certain aperiodicity constraints are satisfied. Perhaps interestingly, this is something different than saying: deterministic (resp. codeterministic) and in first-order logic. Unfortunately, already the boolean combinations B_1 of languages in Π_1 and Σ_1 prove too difficult and we leave the decidability of the class B_1 open, although we do present a corresponding class of automata – aperiodic wordsum automata.

In the next section, Section 2.5, we extend these automata characterizations to higher levels of the hierarchy using a concept of *cascade product* of tree automata. The cascade product of two automata $\mathcal{B}$ and $\mathcal{A}$ is an automaton $\mathcal{B} \circ \mathcal{A}$ which first runs $\mathcal{A}$ on the input tree and then lets $\mathcal{B}$ read both the original tree and the run of $\mathcal{A}$. This operation is devised as an automata-theoretic analogue of formula composition and as such is a tree generalization of cascade product of sequential transducers [2, 12, 13, 33]. A similar operation for trees can also be found in [21]. Using the cascade

product operation, we show that a regular language L is first-order definable if and only if it can be recognized by a cascade product of aperiodic wordsum automata. Unfortunately, we do not know whether this reflects some decidable structural property of the syntactic automaton of L, which is only known to be a homomorphic image of such a cascade product.

In Section 2.5.4, we use the cascade product to prove that the E-nesting hierarchy in CTL* over finite binary trees is infinite. Since a similar result is already known [55], this section serves mainly to show how the cascade product is a convenient tool for nondefinability proofs; being a sort of automata-theoretic equivalent of the Ehrenfeucht-Fraïsé game technique used in logic.

In the last sections we shift our attention to chain logic [62]. This logic is obtained from MSOL by restricting the second-order quantification from arbitrary sets to linearly ordered ones, i.e. chains. The same techniques as for first-order logic show that a language is definable in chain logic if and only if it can be recognized by a cascade product of arbitrary – not necessarily aperiodic – wordsum automata. Equivalently, a language is definable in chain logic if and only if it can be recognized by a cascade product of syntactic automata of deterministic languages. This characterization supports the intuition that chain logic is a half-way point between first-order logic and monadic second-order logic, a point where periodicity is allowed on the "vertical axis" corresponding to paths, but not on the "horizontal axis" corresponding to branching. One should, however, be careful with taking this metaphor too far, as testified by the even-depth language mentioned previously, or by a language due to Potthoff [48] which is definable in chain logic and simultaneously aperiodic with respect to contexts, yet is not definable in first-order logic.

Finally, in Section 2.6, we introduce a structural property of tree automata called *confusion* and conjecture that a language is definable in chain logic if and only if its syntactic automaton does not contain confusion. It is decidable whether or not an automaton contains confusion, therefore if the conjecture were to be true, the class of tree languages definable in chain logic would be decidable. We do show that languages with confusion are not chain definable, the open question is whether a language not expressible in chain logic must necessarily contain confusion.

We conclude the chapter by providing some arguments in favour of the confusion conjecture. Arguably the most compelling piece of evidence is presented in Section 2.6.3, where it is shown that languages without confusion are closed under boolean operations and chain quantification. It is in this context that the cascade product shows its worth. The definition of non-confusing automata has a combinatorial character, yet closure under chain quantification has more of a logical character. Due to its hybrid nature, the

cascade product serves here as a sort of bridge between the two worlds: for instance, chain quantification over a nonconfusing language corresponds to a homomorphic image of the cascade product of two nonconfusing automata.

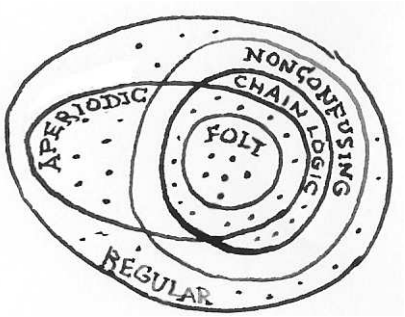


Figure 2.1: Language classes considered in this chapter

2.2 The Word Case

We begin the chapter by briefly and informally surveying some results relating logic and automata over words. We cite here two classic correspondences: between regular word languages and word languages definable in monadic second-order logic; and between aperiodic word languages and word languages definable in first-order logic.

A *deterministic word automaton* is a tuple $\mathcal{A} = \langle Q, \Sigma, q_I, \delta \rangle$, where Q is a finite set of *states*, Σ is a finite *input alphabet*, $q_I \in Q$ is the *initial state* and $\delta : Q \times \Sigma \to Q$ is the *transition function*. The *run* of this automaton over a word $w = a_0 \cdots a_n \in \Sigma^*$ is the unique sequence of states $\rho_w = q_0 \cdots q_{n+1}$ such that $q_0 = q_I$ and for all $i < n$, the state q_{i+1} is $\delta(q_i, a_i)$. Given an *accepting condition*, or set of states $F \subseteq Q$, the language $L(\mathcal{A}, F)$ is the set of words whose run ends in a state from F. In such a case $\mathcal{A}$ is said to *recognize* this language, and the language is called *regular* .

Another way of defining regular word languages is by using the syntactic equivalence relation of Myhill and Nerode. Given a language L, the (bidirectional) *syntactic equivalence relation* identifies two words $w, w' \in \Sigma^*$ if

$$u \cdot w \cdot v \in L \text{ iff } u \cdot w' \cdot v \in L \qquad \text{for all } u, v \in \Sigma^*.$$

An *L-type* is an equivalence class of this relation. A standard result says that a language is regular if and only if it has finitely many types. The syntactic equivalence is a congruence with respect to concatenation, hence it makes sense to write $\alpha \cdot \beta$ for two types α and β of L.

Yet another way of looking at regular word languages is by using a logical approach. Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ be a finite alphabet and $w = a_0 \cdots a_m$ a finite word over Σ. This word is represented as a relational structure

$$\underline{w} = (\mathrm{dom}(w), \leq^w, \underline{\sigma_1}^w, \ldots, \underline{\sigma_n}^w).$$

The domain of the structure $\underline{w}$ is the set of positions $\mathrm{dom}(w) = \{0, \ldots, m\}$ in the word w, the binary relation $\leq^w$ represents the natural order and the unary relations $\underline{\sigma_i}^w$ represent the letters according to $\underline{\sigma_i}^w = \{j : a_j = \sigma_i\}$. The signature of this structure is denoted as Sig_Σ.

A language of finite words $L \subseteq \Sigma^*$ is *definable in monadic second-order logic* if there is a monadic second-order sentence over the signature Sig_Σ which is true in exactly the words from L. We use MSOLW to define the class of word languages definable in monadic second-order logic. A classic result [9, 14, 66] states that this class is equal to the class of regular word languages.

Within the framework of monadic second-order logic, one naturally distinguishes the fragment which does not quantify over sets, i. e. first-order logic. Akin to monadic second-order definability, a word language is *first-order definable* if the defining formula can be found already in first-order logic. We write FOLW to denote the class of first-order definable word languages.

A property of automata corresponding to first-order definability is as follows. A regular language is *periodic*, if for some type α,

$$\alpha \cdot \alpha \neq \alpha \quad \text{but} \quad \alpha^n = \alpha \text{ for some } n \geq 2 \ .$$

A language is *aperiodic* if it is not periodic. A celebrated result due to Schützenberger [53] and McNaughton and Papert [40] says that a language is first-order definable if and only if it is aperiodic. In particular the following problem is decidable: "is a given regular word language first-order definable". This is due to the fact that checking aperiodicity is decidable – in fact it is in PSPACE with respect to the size of a recognizing deterministic automaton [57].

2.2.1 LTL and Kamp's Theorem

In this section we introduce the syntax and semantics of linear temporal logic (LTL). We then cite the famous theorem of Kamp, which states that word languages definable in LTL are exactly the first-order definable ones.

Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ be a finite alphabet. Formulas of *linear temporal logic over* Σ are defined by the following grammar:

$$\mathcal{V} ::= \mathcal{V} \wedge \mathcal{V} \mid \mathcal{V} \vee \mathcal{V} \mid \neg\mathcal{V} \mid \mathcal{V} \mathsf{U} \mathcal{V} \mid \sigma_1 \mid \cdots \mid \sigma_n.$$

We define the semantics by giving a translation that assigns to every LTL formula ψ a corresponding first-order formula $[\![\psi]\!]_{\mathrm{FOL}}$ with one free variable. This translation is defined by induction:

- $[\![\sigma]\!]_{\mathrm{FOL}}(x) = \underline{\sigma}(x)$ for $\sigma \in \Sigma$;
- $[\![\phi \wedge \psi]\!]_{\mathrm{FOL}}(x) = [\![\phi]\!]_{\mathrm{FOL}}(x) \wedge [\![\psi]\!]_{\mathrm{FOL}}(x)$, similarly for $\neg, \vee$;
- $[\![\phi \mathsf{U} \psi]\!]_{\mathrm{FOL}}(x) = \exists y > x.\ [\![\psi]\!]_{\mathrm{FOL}}(y) \wedge \forall z.\ x < z < y \Rightarrow [\![\phi]\!]_{\mathrm{FOL}}(z)$.

Note that our semantics of the U operator are *strict*, i.e. we do not require ϕ to hold in the node x.

An LTL formula ψ is *satisfied* in a word w if the first position of the word model $\underline{w}$ satisfies the first-order formula $[\![\psi]\!]_{\mathrm{FOL}}$. A formula *defines* a language L if it is satisfied in exactly the words belonging to L.

Sometimes the syntax of LTL is extended by additional operators F, G and X – called *finally*, *globally* and *next*, respectively – which we consider here to be the following abbreviations:

$$\mathsf{F}\varphi = \top \mathsf{U} \varphi \qquad \mathsf{G}\varphi = \neg \mathsf{F} \neg \varphi \qquad \mathsf{X}\varphi = \bot \mathsf{U} \varphi\ .$$

By definition, every LTL-definable language is in FOLW. By Kamp's theorem [34, 26], the converse also holds.

2.3 Finite Trees: Basic Definitions

We now focus our attention on tree languages. In this section, we introduce some basic concepts regarding automata and logic for tree languages, extending the definitions for the word case. We then establish an analogous logical framework, along with the correspondence of monadic second-order logic and regular languages. Finally, as a tree analogue to LTL, we present the branching temporal logic CTL*.

2.3.1 Regular Tree Languages and Tree Automata

In this section we introduce the basic definitions regarding trees and tree automata. We will consider finite trees here, although some of our results carry over to the infinite case.

A *finite A-sequence* is a function $\boldsymbol{a} : \{0, \ldots, n\} \to A$, while an *infinite A-sequence* is a function $\boldsymbol{a} : \mathbb{N} \to A$. We use boldface letters to denote sequences. Given a function $f : A \to B$ and an A-sequence $\boldsymbol{a}$, the function composition $f \circ \boldsymbol{a}$ is also a well defined B-sequence. Often we will forsake the

functional notation and write $\boldsymbol{a}_i$ instead of $\boldsymbol{a}(i)$. The length $|\boldsymbol{a}| \in \mathbb{N} \cup \{\infty\}$ of a sequence is the size of its domain. We use A^* to denote the set of finite A-sequences and $A^{\mathbb{N}}$ for the set of infinite A-sequences. The concatenation of two sequences $\boldsymbol{a}$ and $\boldsymbol{b}$, denoted by $\boldsymbol{a} \cdot \boldsymbol{b}$, is defined in the usual fashion.

An *unlabeled tree* is a finite nonempty prefix-closed subset u of $\{0,1\}^*$ which contains a word $x \cdot 0$ if and only if it contains the word $x \cdot 1$. Let Σ be some finite set, called an *alphabet*. A Σ-*tree* is a function $t : u \to \Sigma$, where u is an unlabeled tree, which is called the *domain of* t and denoted $\mathrm{dom}(t)$. We write $\mathrm{Trees}(\Sigma)$ to denote the set of Σ-trees. A *tree language over* Σ is any set of Σ-trees. A *node* of a tree is any element of its domain. We order nodes of a tree using the prefix relation $\leq$. In every tree we distinguish the empty sequence ε, called *root*, which is the least element with respect to $\leq$. A node v of t is a *leaf* if it is $\leq$-maximal. Nodes that are not leaves are called *inner* nodes. Given two Σ-trees s and t and a node $v \in \mathrm{dom}(t)$, the substitution $t[v := s]$ is the Σ-tree defined:

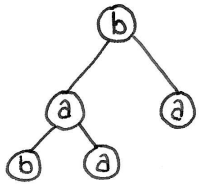


Figure 2.2: A tree

$$t[v := s](w) = \begin{cases} s(u) & \text{if } w = v \cdot u \text{ for some } u \in \mathrm{dom}(s) \\ t(w) & \text{otherwise} \end{cases}$$

The *composition* of a Σ-tree s and a Γ-tree t is the $\Sigma \times \Gamma$-tree $s \hat{} t$ defined $(s \hat{} t)(v) = (s(v), t(v))$. For this composition to be defined, both trees need to have the same domain.

Let $*$ be some letter not occurring in the alphabet Σ. A Σ-*multicontext* is a $\Sigma \cup \{*\}$-tree where the label $*$ may occur only in the leaves. We will denote multicontexts using letters C, D. Any leaf labeled by $*$ is called a *hole*; we write $\mathrm{holes}(C)$ to denote the set of holes in a multicontext C. Let C be a multicontext with $v_1, \ldots, v_n$ being its holes written from left to right. Given Σ-trees $t_1, \ldots, t_n$, we write $C[t_1, \ldots, t_n]$ to denote the tree obtained by plugging each tree t_i into the respective hole v_i. We will sometimes treat a letter $\sigma \in \Sigma$ as the unique multicontext with holes in the nodes 0 and 1 and the label σ in the root. Hence $\sigma[s,t]$ is the unique tree with σ in the root and the trees s and t in the left and right sons.

A multicontext C with only one hole v is called a *context* and is denoted $C[]$. The composition of two contexts $C[]$ and $D[]$ is the naturally defined context $C[D[]]$. By $C^n[]$ we denote the n-fold composition of $C[]$.

Definition 2.3.1 A *deterministic bottom-up tree automaton* over Σ-trees is a tuple $\mathcal{A} = \langle Q, \Sigma, q_I, \delta \rangle$, where Q is finite set of *states*, Σ is a finite *input*

alphabet, $q_I \in Q$ is the *initial* state and $\delta \in Q^2 \times \Sigma \to Q$ is the *transition function*.

This automaton starts below the leaves in the initial state (which are called the bottom of the tree) and then proceeds towards the root according to the transition function. Formally, with an automaton $\mathcal{A}$ and a Σ-tree t we associate the *run tree* $t^{\mathcal{A}} : \mathrm{dom}(t) \to Q$ defined inductively:

$$t^{\mathcal{A}}(v) = \begin{cases} \delta(q_I, q_I, t(v)) & \text{if } v \text{ is a leaf;} \\ \delta(t^{\mathcal{A}}(v{\cdot}0), t^{\mathcal{A}}(v{\cdot}1), t(v)) & \text{otherwise.} \end{cases}$$

A tree t is said to *evaluate* to the state $t^{\mathcal{A}}(\varepsilon)$. Given a set $F \subseteq Q$ of *accepting states*, we write $L(\mathcal{A}, F)$ to denote the language of Σ-trees which evaluate to a state in F. Such a language is *recognized* by $\mathcal{A}$ and is said to be *regular*.

The *syntactic equivalence* of a language $L \subseteq \mathrm{Trees}(\Sigma)$ is the equivalence relation $\simeq_L$ which identifies two Σ-trees s and t if

$$C[s] \in L \;\text{ iff }\; C[t] \in L \qquad \text{for all } \Sigma\text{-contexts } C.$$

Equivalence classes of the syntactic equivalence of L are called *L-types*, and the set of L-types is denoted as $\mathrm{Types}(L)$. A language is regular if and only if it has a finite number of types. The syntactic equivalence relation is a congruence in the sense that for all $\sigma \in \Sigma$,

$$s \simeq_L s' \text{ and } t \simeq_L t' \quad \Rightarrow \quad \sigma[s,t] \simeq_L \sigma[s',t'].$$

Hence it makes sense to write expressions like $\sigma[\alpha, \beta]$, with σ a letter and α, β types. For every regular language there is a minimal bottom-up deterministic automaton recognizing it, called its *syntactic automaton*. The states of this automaton are types of the language and the transition function is defined accordingly.

Let L be a regular tree language over Σ and $\mathcal{A}$ an automaton of state space Q over Σ-trees. For a Σ-multicontext C we define in the natural way the following evaluations:

$$\begin{aligned} C[\nu] \in Q \quad &\text{where} \quad \nu : \mathrm{holes}(C) \to Q; \\ C[\nu] \in \mathrm{Types}(L) \quad &\text{where} \quad \nu : \mathrm{holes}(C) \to \mathrm{Types}(L); \\ C[\nu] \in \mathrm{Trees}(\Sigma) \quad &\text{where} \quad \nu : \mathrm{holes}(C) \to \mathrm{Trees}(\Sigma) \end{aligned}$$

which describe how the multicontext C affects states of $\mathcal{A}$, trees and types of L respectively.

Definability

Given a class $\mathcal{L}$ of regular languages, we will be interested in the *definability* decision problem for $\mathcal{L}$:

> Given a regular language L represented by its syntactic automaton, decide if L belongs to $\mathcal{L}$.

If the above problem is decidable, we say the class $\mathcal{L}$ is decidable.

2.3.2 Logic

In this section we introduce the logical approach to tree languages. We start by considering languages definable in monadic second-order logic and languages definable in first-order logic.

Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ be a finite alphabet. With a Σ-tree t we associate the logical structure

$$\underline{t} = (\mathrm{dom}(t), S_0^t, S_1^t, \leq^t, \underline{\sigma_1}^t, \ldots, \underline{\sigma_n}^t)$$

which is defined like in the case of words, except that now we have two new unary relations S_0^t and S_1^t, which denote the sets of left and right sons:

$$S_0^t = \mathrm{dom}(t) \cap \{0,1\}^* \cdot 0 \quad \text{and} \quad S_1^t = \mathrm{dom}(t) \cap \{0,1\}^* \cdot 1 \ .$$

We denote the signature of such a structure by Sigt_Σ.

A tree language $L \subseteq \mathrm{Trees}(\Sigma)$ belongs to the class *MSOLT* of *monadic second-order definable tree languages* if and only if there is a sentence ϕ of monadic second-order logic over the signature Sigt_Σ which is satisfied in exactly the trees belonging to L. If ϕ is actually a first-order formula, then L also belongs to the class *FOLT* of *first-order definable tree languages*. For finite trees, a theorem of Thatcher and Wright [59] shows that a tree language is regular if and only if it is in MSOLT.

Tree Aperiodicity

We would like to end this section by noting that the natural analogue of aperiodicity for trees does not characterize the first-order tree languages.

Definition 2.3.2 A regular tree language $L \subseteq \mathrm{Trees}(\Sigma)$ is *periodic* if there exists a Σ-context and an L-type α such that

$$C[\alpha] \neq \alpha \quad \text{but} \quad C^n[\alpha] = \alpha \text{ for some } n \geq 2 \ . \tag{2.1}$$

The language is *aperiodic* if it is not periodic.

The first example of an aperiodic tree language which is not first-order definable was presented by Heuter in [31]. A simple example of this sort of language follows:

Example: Consider the set $L \subseteq \text{Trees}(\{0, 1, \vee, \wedge\})$ of trees that represent well-formed boolean expressions that evaluate to 1. This language has three types: expressions evaluating to 0, expressions evaluating to 1 and malformed expressions. Consider now a context $C[]$. The function on types induced by $C[]$ is monotone on the types 0 and 1 and returns a malformed expression when given a malformed expression. Therefore, no context satisfies condition (2.1) and the language L is aperiodic. In Section 2.6.1 will prove that this language is not first-order definable. $\square$

2.3.3 CTL* and FOLT

In this section we present the branching time temporal logic CTL* [19, 16]. We consider this logic because it is equivalent to first-order logic over trees, but has a structure more convenient for automata.

Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ be a finite alphabet and consider the following grammar, with two mutually recursive variables $\mathcal{V}$ and $\mathcal{W}$:

$$\begin{aligned} \mathcal{V} &\rightarrow \mathcal{V} \wedge \mathcal{V} \mid \mathcal{V} \vee \mathcal{V} \mid \neg\mathcal{V} \mid \mathcal{V}\mathsf{U}\mathcal{V} \\ \mathcal{W} &\rightarrow \mathsf{E}\mathcal{V} \mid \mathsf{A}\mathcal{V} \mid \mathcal{W} \wedge \mathcal{W} \mid \mathcal{W} \vee \mathcal{W} \mid \neg\mathcal{W} \mid \sigma_1 \mid \cdots \mid \sigma_n \mid S_0 \mid S_1 \end{aligned}$$

Formulas derived from the symbol $\mathcal{V}$ are called *path formulas* and formulas derived from the symbol $\mathcal{W}$ are called *node formulas*. In path formulas, we will also use the abbreviations F, G and X introduced in the section on LTL. The set of CTL* formulas over Σ is defined to be the set of node formulas.

Analogously to the case for LTL, we define the semantics using a mapping into first-order logic. This mapping assigns to each node formula a formula of one free variable and to each path formula a formula of two free variables as described below.

Intuitively, a node formula ψ corresponds to a formula $[\![\psi]\!]_{\text{FOL}}(x)$, which describes the subtree rooted in the node x, while a path formula ψ corresponds to a formula $[\![\psi]\!]_{\text{FOL}}(x, y)$ which describes properties of the path from the node x to a leaf y, along with the relevant subtrees.

For node formulas, the translation is:

$$\begin{aligned} [\![\mathsf{E}\psi]\!]_{\text{FOL}}(x) &= \exists y \geq x.[\![\psi]\!]_{\text{FOL}}(x, y) \wedge \forall z.\neg(z > y) \\ [\![\mathsf{A}\psi]\!]_{\text{FOL}}(x) &= [\![\neg\mathsf{E}\neg\psi]\!]_{\text{FOL}} \\ [\![\psi \wedge \varphi]\!]_{\text{FOL}}(x) &= [\![\psi]\!]_{\text{FOL}}(x) \wedge [\![\varphi]\!]_{\text{FOL}}(x), \text{ similarly for } \vee \text{ and } \neg \\ [\![R]\!]_{\text{FOL}}(x) &= R(x) \text{ for } R \in \Sigma \cup \{S_0, S_1\} \end{aligned}$$

We call node formulas of the form $\mathsf{E}\varphi$ *existential formulas* and node formulas of the form $\mathsf{A}\varphi$ *universal formulas*. For path formulas the translation is the same as for LTL:

$$\begin{aligned}
[\![\psi \mathsf{U} \varphi]\!]_{\mathrm{FOL}}(x,y) &= \exists z.\, [x < z \leq y \wedge [\![\varphi]\!]_{\mathrm{FOL}}(z,y) \wedge \\
&\qquad \wedge \forall u.\, x < u < z \Rightarrow [\![\psi]\!]_{\mathrm{FOL}}(u,y)] \\
[\![\psi \wedge \varphi]\!]_{\mathrm{FOL}}(x,y) &= [\![\psi]\!]_{\mathrm{FOL}}(x,y) \wedge [\![\varphi]\!]_{\mathrm{FOL}}(x,y), \text{ similarly for } \vee \text{ and } \neg
\end{aligned}$$

A CTL* formula ψ is *satisfied* in a Σ-tree t, which is written $t \models \psi$, if the first-order formula $[\![\psi]\!]_{\mathrm{FOL}}$ is true in the root of t. A CTL* formula *defines* a tree language L if it is satisfied in exactly the trees from L. Our very definition shows that CTL* definable languages are in FOLT. It turns out, that the converse is also true – a generalization of Kamp's Theorem also holds for CTL* :

Theorem 2.3.3
FOLT and CTL* define the same class of tree languages.

This theorem – allowing for different syntax – has been proved for infinite trees by Hafer and Thomas in [30]. The simpler case of finite trees can be shown using the same proof technique.

A CTL* Hierarchy

We define here a hierarchy of CTL* formulas which accounts for nesting of the E and A operators. Let $\Pi_0 = \Sigma_0$ be the set of node formulas containing neither E nor A. The ensuing levels of the hierarchy for $i > 0$ are defined by induction:

- B_i is the set of boolean combinations of Σ_i and Π_i formulas.
- Σ_i is the set of existential node formulas whose every proper node subformula belongs to B_{i-1}.
- Π_i is the set of universal node formulas whose every proper node subformula belongs to B_{i-1}.
- Δ_i is the set of Σ_i formulas for which an equivalent formula in Π_i exists.

Abusing the notation, we will also use B_i, Σ_i, Π_i and Δ_i to denote the language classes corresponding to the appropriate formula types. This hierarchy of languages is depicted in Figure 2.3 along with the obvious inclusions and equalities.

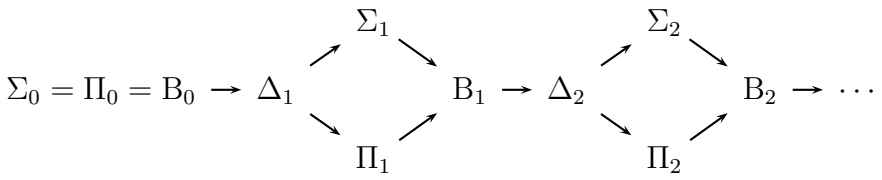


Figure 2.3: The hierarchy in CTL* with arrows denoting inclusion

An Example

Before we proceed with a more systematic exposition, we attempt here to whet the reader's appetite using an example that shows how insidiously the structure of a CTL* formula can be hidden in a language.

Every language in Π_1 is deterministic (see the next section for a definition) and first-order definable. What may be surprising is that the reverse inclusion does not hold.

Lemma 2.3.4 There exists a deterministic and first-order definable language outside Π_1.

Proof
Consider the language:

$$L = \{t \in \mathrm{Trees}(\{\sigma\}) : \text{every leaf of } t \text{ is at even depth}\} \ .$$

Obviously this language is deterministic. Using the algorithm from Lemma 2.4.5, one can prove that L is not in Π_1. The tricky part is that L is first-order definable.

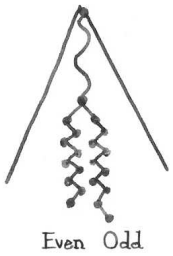
In order to show this, we will produce a CTL* formula defining the complement of L. Given a node v, the *middle path from* v is the path which starts in v with a left turn and alternately does left and right turns. The following CTL* formula states that the middle path is of odd length:

$$\psi_1 \quad = \quad \mathsf{E}[\mathsf{X}S_0 \wedge \mathsf{G}(S_1 \Rightarrow \neg \mathsf{X}S_1) \wedge \mathsf{G}(S_0 \Rightarrow \mathsf{X}S_1)]$$

A similar formula ψ_0 for even lengths can be written. We say a tree is *mixed* if it contains paths of both even and odd lengths. No mixed tree belongs to L. If a tree is not mixed then all of its leaves are at even depth if and only if the tree satisfies ψ_0.

The key idea is that the following conditions are equivalent:

- a tree is mixed;
- there are two siblings whose subtrees are non-mixed and which satisfy ψ_0 and ψ_1 respectively;
- there are two siblings which satisfy ψ_0 and ψ_1 respectively.



Hence the following CTL* formula defines the complement of L:

$$\psi_1 \vee \mathsf{EF} \bigvee_{i \in \{0,1\}} (\mathsf{EX}[\psi_0 \wedge S_i]) \wedge (\mathsf{EX}[\psi_1 \wedge S_{1-i}]).$$

□

2.4 Low Levels of the Hierarchy

In this section we consider the classes Δ_1, Π_1, Σ_1 and B_1, and characterize them in terms of tree automata. Using these characterizations, we prove that the first three are decidable. We are, however, unable to provide a decision procedure for the class B_1.

Deterministic Top-Down Automata

Before proceeding, we briefly consider *top-down* tree automata, which traverse the tree in a direction opposite to that of bottom-up automata: they start in the root and go down into the leaves. Although *nondeterministic* top-down automata define the same class of languages as bottom-up automata, this is no longer true for their deterministic variant. For instance, the language "there exists an a-labeled node" is not recognizable by any deterministic top-down automaton.

A precise definition is as follows. A *deterministic top-down tree automaton* is a tuple

$$\mathcal{D} = \langle Q, \Sigma, q_I, \delta \rangle \ ,$$

with Q being the set of *states*, Σ being the *input alphabet*, $q_I \in Q$ the *initial state* and the *transition function* δ being of the form $\delta : Q \times \Sigma \to Q^2$. The *run* of the automaton $\mathcal{D}$ over a Σ-tree t is the tree

$$t^{\mathcal{D}} : \mathrm{dom}(t) \to Q$$

defined by induction as follows: the root is labeled with the initial state q_I, and if a node $v \in \mathrm{dom}(t)$ is labeled with q, then its sons $v \cdot 0$ and $v \cdot 1$ are

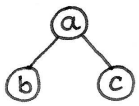
labeled with the first and second components of $\delta(q, t(v))$ respectively. Note that this run does not depend on the leaf labels.

Given a set of states $F \subseteq Q$, called the *accepting condition*, the language $L(\mathcal{D}, F)$ is the set of Σ-trees t where for every leaf v the value $\delta(t^{\mathcal{D}}(v), t(v))$ belongs to $F \times F$. Such a language is said to be *deterministic*. We use DET to denote the class of deterministic languages. As noted above, not all regular languages are deterministic. Moreover, the class DET is not closed under boolean operations (except intersection), hence DET membership cannot be decided simply by looking at the syntactic automaton.

Another way of looking at a deterministic top-down automaton is that it verifies if *every* path in the given tree satisfies some property. Given an alphabet Σ, the *trace alphabet of* Σ, denoted Σ_t, is the set $\Sigma \times \{0, 1\}$. A *trace* of a Σ-tree t is any sequence

$$(\sigma_0, a_0), \ldots, (\sigma_n, a_n) \in \Sigma_t^*$$

such that $a_0 \cdots a_{n-1}$ is a leaf in t and $\sigma_i = t(a_0 \cdots a_{i-1})$ for all $i \in [0, n]$. The set of traces of a tree t is denoted $\mathrm{tr}(t)$. For instance, the tree:



has the following traces: $\{(a,0)\cdot(b,0), (a,0)\cdot(b,1), (a,1)\cdot(c,0), (a,1)\cdot(c,1)\}$. The set of traces of a language L is

$$\mathrm{tr}(L) = \bigcup_{t \in L} \mathrm{tr}(t).$$

Fact 2.4.1 The set of traces of a regular tree language is a regular word language and can be effectively computed.

Proof
Consider a language $L = L(\mathcal{D}, F)$ recognized by a top-down deterministic automaton $\mathcal{D} = \langle Q, \Sigma, q_I, \delta, F \rangle$. We assume without loss of generality that every state is used in some accepting run. The language $\mathrm{tr}(L)$ is recognized by the deterministic word automaton whose state space is Q, whose initial state is q_I and whose transition function δ' is defined:

$$\delta'(q, (\sigma, d)) = q_d \quad \text{iff} \quad \delta(q, \sigma) = (q_0, q_1) \qquad \text{for } d \in \{0, 1\} \ .$$

□

Fact 2.4.2 A tree language L is deterministic iff $L = \{t : \mathrm{tr}(t) \subseteq \mathrm{tr}(L)\}$.

Proof
Follows from the fact that the state of a top-down deterministic automaton in a node v is uniquely determined by the trace leading to the node v. □

Aperiodically Deterministic Automata and Π_1

A language is *aperiodically deterministic* if it is deterministic and $\mathrm{tr}(L)$ is an aperiodic word language. We use DETAP to denote the class of such languages. Here we show that this class coincides with Π_1 and is decidable. Hence also the decidability of the classes Σ_1 and Δ_1.

The *tree completion* of a word $w \in \Sigma_t^*$ is any nonempty set $A \subseteq \Sigma_t^*$ such that for some Σ-tree t,

$$A = \{v \in \mathrm{tr}(t) : w \text{ is a prefix of } v\}.$$

Given a language $K \subseteq \Sigma_t^*$, a word $w \in \Sigma_t^*$ can be *K-completed* if some tree completion of w is a subset of K.

Fact 2.4.3 Let $K \subseteq \Sigma_t^*$ be a first-order definable word language. The set of words that can be K-completed is also first-order definable.

Proof
Let $\mathcal{A}$ be an aperiodic word automaton recognizing K. Whether or not a word can be completed depends only on the state of $\mathcal{A}$ to which it is evaluated. Hence the set of completable words is also recognized by $\mathcal{A}$. □

Lemma 2.4.4 A tree language L is in Π_1 iff it is aperiodically deterministic.

Proof
If $\mathrm{tr}(L)$ is aperiodic, then it is equivalent to some LTL formula, therefore the right to left implication follows from Fact 2.4.2.

The left to right implication requires some more effort. Let L be of the form $\mathsf{A}\psi$ and let K be the aperiodic word language defined by ψ. The tree language L is deterministic, since it is recognized by a top-down deterministic automaton which verifies whether every trace is in K. By construction, a tree in L has all traces in K, hence $\mathrm{tr}(L) \subseteq K$. Let $M = K \setminus \mathrm{tr}(L)$. In general the language M need not be empty, but every word $w \in M$ must satisfy the following property:

$$t \notin L \text{ for all } t \text{ such that } w \in \mathrm{tr}(t).$$

This is equivalent to the property that some prefix of w cannot be K-completed. By Fact 2.4.3, this is property is first-order definable. Hence $\mathrm{tr}(L)$ belongs to FOLW as the difference of two languages in FOLW. □

Lemma 2.4.5 It is decidable if a regular language is in Π_1.

Proof
By Lemma 2.4.4, a language is in Π_1 if and only if it is deterministically aperiodic. But this is decidable, by Facts 2.4.2 and 2.4.1 and the decidability of aperiodicity for word languages. □

Corollary 2.4.6 The classes Σ_1, Π_1 and Δ_1 are decidable.

Wordsum Automata and B_1

In this section we introduce wordsum automata. A wordsum automaton is a bottom-up tree automaton which runs a word automaton on all of the traces of the tree (starting with the leaves). We show that languages recognized by these automata coincide with the class B_1.

Definition 2.4.7 Let $\mathcal{A} = \langle Q, \Sigma_t, q_I, \delta \rangle$ be a word automaton over the trace alphabet Σ_t of some alphabet Σ. The *wordsum automaton over* $\mathcal{A}$ is the bottom-up tree automaton

$$\mathrm{WS}(\mathcal{A}) = \langle P(Q), \Sigma, \{q_I\}, \delta' \rangle$$

whose transition function δ' is defined

$$\delta'(R_0, R_1, \sigma) = \bigcup_{i \in \{0,1\}} \{\delta(q, (a, i)) : q \in R_i\} \ .$$

We use WS to denote the class of wordsum automata and WSAP to denote the class of *aperiodically wordsum automata*, i.e. wordsum automata obtained from an aperiodic word automaton. Abusing the notation slightly, we also use WS and WSAP for the classes of languages recognized by these automata.

Lemma 2.4.8 A language is in WSAP if and only if it is a boolean combination of languages in DETAP. A language is in WS if and only if it is a boolean combination of languages in DET.

Proof
We will only prove the second part of the statement, the proof of the first being analogous.

First we show that a boolean combination of deterministic languages is in WS. Let L be a boolean combination of deterministic languages $L_1, \ldots, L_n$. Deterministic languages are obviously wordsum languages, hence there exist wordsum automata $\mathrm{WS}(\mathcal{A}_1), \ldots, \mathrm{WS}(\mathcal{A}_n)$ recognizing these languages. Consider now the word automaton $\mathcal{A}$ obtained by taking the cartesian product

of all the word automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$. One can easily show that the wordsum automaton $\mathrm{WS}(\mathcal{A})$ recognizes L.

For the other inclusion, let L be a language in WS. This means that $L = L(\mathcal{B}, F)$ for some wordsum automaton $\mathcal{B} = \mathrm{WS}(\mathcal{A})$ and some family F of subsets of the state space Q of $\mathcal{A}$. To end the proof, we will use the following obvious claim, which shows that if the acceptance condition is a powerset, then a wordsum language is deterministic:

Claim 2.4.8.1 For every $R \subseteq Q$, the language $L(\mathcal{B}, P(R))$ is deterministic.

Let $R_1, \ldots, R_m$ be the subsets of Q in the set F. Since L is the sum of the languages $L(\mathcal{B}, \{R_i\})$, the proof of Lemma 2.4.8 follows from Claim 2.4.8.1 and the following equality:

$$L(\mathcal{B}, \{R_i\}) = L(\mathcal{B}, P(R_i)) \setminus \bigcup_{q \in R_i} L(\mathcal{B}, P(R_i \setminus \{q\})) \ .$$

□

Corollary 2.4.9 A language belongs to B_1 if and only if it is recognizable by an aperiodic wordsum automaton.

Unfortunately, we do not know if the classes WS and WSAP are decidable.

2.5 Cascade Product

In this section we introduce the cascade product of tree automata. Using this notion, we present two characterizations: we show that first-order logic over trees corresponds to cascade products of aperiodically wordsum automata; and that chain logic corresponds to cascade products of arbitrary wordsum automata. We then prove that the hierarchy from Figure 2.3 is infinite.

2.5.1 Cascade Product

From now on, we will only be using bottom-up deterministic automata. The cascade product $\mathcal{B} \circ \mathcal{A}$ defined here formalizes the idea that the automaton $\mathcal{B}$ reads the output of the automaton $\mathcal{A}$ on an input tree.

Definition 2.5.1 Let $\mathcal{A}$ be an automaton over Σ-trees of state space Q and let $\mathcal{B}$ be an automaton over $\Sigma \times Q$-trees of state space R. A *cascade product* $\mathcal{B} \circ \mathcal{A}$ is an automaton over Σ-trees of state space $R \times Q$ such that

$$t^{\mathcal{B} \circ \mathcal{A}} = (t \hat{} t^{\mathcal{A}})^{\mathcal{B}} \hat{} t^{\mathcal{A}} \qquad \text{for all } \Sigma\text{-trees } t.$$

Intuitively, a run of $\mathcal{B} \circ \mathcal{A}$ can be decomposed into two parts: first, $\mathcal{A}$ runs on the tree and additionally labels each node with a state; then $\mathcal{B}$ runs over the tree equipped with the additional labeling. Cascade product of tree automata is an analogue of cascade composition of sequential transducers [2, 12, 13, 33] and also of wreath product of transformation semigroups from the celebrated Krohn-Rhodes Theorem [39].

A cascade product evaluates trees the same way as the automaton whose initial state is $(q_I^{\mathcal{B}}, q_I^{\mathcal{A}})$ and whose transition function is defined:

$$\delta^{\mathcal{B}\circ\mathcal{A}}((q_0, r_0), (q_1, r_1), a) = (\delta^{\mathcal{B}}(q_0, q_1, (a, r)), r) \quad \text{where } r = \delta^{\mathcal{A}}(r_0, r_1).$$

Although cascade product does not expand the power of automata beyond regular tree languages, it does introduce a certain structure. Using this structure, we will characterize logically defined classes of tree languages.

Given $n \geq 1$, the n-th cascade power of a class of automata X is defined:

$$X^n = \{\mathcal{A}_1 \circ \cdots \circ \mathcal{A}_n : \mathcal{A}_i \in X \text{ for all } i \leq n\}.$$

By convention, the class X^0 consists of all automata with one state. We use $\mathrm{CP}(X, n)$ to denote the class of languages recognized by automata in X^n and use the abbreviation

$$\mathrm{CP}(X) = \bigcup_{n\in\mathbb{N}} \mathrm{CP}(X, n).$$

In our characterizations, we will also be using deterministic languages. However, since these are defined using top-down automata, while the cascade product uses bottom-up automata, we use the class SDET of syntactic (bottom-up) automata of languages in DET. Note that not every language recognized by an automaton in SDET is in DET, nor do the classes SDET and WS coincide (cf. the syntactic automaton of the language "there is an a-labeled node and there is a b-labeled node", which is in WS but not in SDET). The class SDETAP is the aperiodic version of SDET.

Fact 2.5.2 $\mathrm{CP}(\mathrm{WS}) = \mathrm{CP}(\mathrm{SDET})$ and $\mathrm{CP}(\mathrm{WSAP}) = \mathrm{CP}(\mathrm{SDETAP})$.

Proof
The right-to-left inclusions in both equalities are obvious. We will only show the left-to-right inclusion for the first equality, the other one being analogous.

We will show that $\mathrm{WS} \subseteq \mathrm{CP}(\mathrm{SDET})$, essentially proving that cascade product can simulate boolean operations. Let L belong to WS. By Lemma 2.4.8, L is a boolean combination of languages $L_1, \ldots, L_n \in \mathrm{DET}$. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n \in \mathrm{SDET}$ be the syntactic automata of these languages, with state spaces $Q_1, \ldots, Q_n$. For $i \leq n$, let $K_i \in \mathrm{DET}$ be the language

$$\{t\hat{}t_{i-1}\hat{}\cdots\hat{}t_1 : t \in L_i \text{ and } t_j \in \mathrm{Trees}(Q_j) \text{ for j} < i\}.$$

Let $\mathcal{B}_i \in \mathrm{SDET}$ be the syntactic automaton of the language K_i. Without loss of generality, we can assume that the state space of $\mathcal{B}_i$ is the same as that of $\mathcal{A}_i$. One can easily verify that the cascade product $\mathcal{B}_n \circ \cdots \circ \mathcal{B}_1$ recognizes the language L. □

2.5.2 Tree Automata for First-Order Logic

In this section we present the cascade product characterization of first-order definable tree languages. This characterization is obtained by following a straightforward translation of CTL* formulas into automata.

We say a CTL* formula *defines a state* q in an automaton $\mathcal{A}$ if it is satisfied in exactly the trees that evaluate to q.

Lemma 2.5.3 For $n \geq 1$, $\mathrm{CP}(\mathrm{WSAP}, n) \subseteq \mathrm{B}_n$.

Proof
We will prove that every state of an automaton in WSAP^n is defined by some formula in B_n. The proof is by induction on n, the case for $n = 1$ following from Corollary 2.4.9.

For $n > 1$, consider an automaton $\mathcal{B} \circ \mathcal{A} \in \mathrm{WSAP}^n$, with $\mathcal{A} \in \mathrm{WSAP}^{n-1}$ an automaton of state space Q and $\mathcal{B}$ a wordsum automaton over the alphabet $\Sigma \times Q$. By induction assumption, for every state $r \in Q$, there is a formula $\psi_r \in \mathrm{B}_{n-1}$ defining r in $\mathcal{A}$ and for every state q of $\mathcal{B}$, there is a formula $\varphi_q \in \mathrm{B}_1$ defining q in $\mathcal{B}$.

Let $\hat{\varphi}_q$ be the formula obtained from φ_q be replacing every occurrence of a letter $(\sigma, r) \in \Sigma \times Q$ by the formula $\sigma \wedge \psi_r$. A simple verification shows that for every state r of $\mathcal{A}$ and q of $\mathcal{B}$, the formula $\psi_r \wedge \hat{\varphi}_q$ belongs to B_n and defines (q, r) in $\mathcal{B} \circ \mathcal{A}$. □

The following lemma can be proved by an induction on k using a similar technique to the one used in the proof of Lemma 2.4.8:

Lemma 2.5.4 For every finite set of languages $\mathcal{X} \subseteq \mathrm{CP}(\mathrm{WS}, k)$, there is an automaton in WS^k recognizing simultaneously all languages in $\mathcal{X}$.

Lemma 2.5.5 For $n \geq 1$, $\mathrm{B}_n \subseteq \mathrm{CP}(\mathrm{WSAP}, n)$.

Proof
Induction on n. The case of $n = 1$ follows from Corollary 2.4.9. Consider some $n > 1$ and assume that the statement is true for $n - 1$. Let $\varphi \in \mathrm{B}_n$ be a formula over Σ with $\Psi \subseteq \mathrm{B}_{n-1}$ being all its node subformulas.

By induction assumption and Lemma 2.5.4, there is an automaton $\mathcal{A} \in \mathrm{WSAP}^{n-1}$ of state space Q recognizing all the formulas in Ψ. Let $\hat{\varphi} \in \mathrm{B}_1$ be the formula over the alphabet $\hat{\Sigma} = \Sigma \times P(\Psi)$ obtained from φ by replacing

$$\psi \in \Psi \text{ with } \bigvee_{\sigma} \bigvee_{\Gamma:\psi\in\Gamma} (\sigma, \Gamma) \qquad \text{and} \qquad \sigma \in \Sigma \text{ with } \bigvee_{\Gamma} (\sigma, \Gamma).$$

This formula, instead of using subformulas from Ψ, consults the value of an appropriate letter. Since $\hat{\varphi}$ belongs to B_1, there is by Corollary 2.4.9 a wordsum automaton $\mathcal{B} \in \mathrm{WSAP}$ recognizing it. Let $\mathcal{C}$ be the wordsum automaton over $\Sigma \times Q$-trees obtained from $\mathcal{B}$ by treating a letter $(\sigma, q) \in \Sigma \times Q$ as the letter

$$(\sigma, \{\psi \in \Psi : \text{some tree evaluating to } q \text{ satisfies } \psi\})$$

One can now verify that the cascade product $\mathcal{C} \circ \mathcal{A}$ recognizes ψ. □

Putting together Lemmas 2.5.3 and 2.5.5 we obtain:

Corollary 2.5.6 For $n \geq 1$, $\mathrm{B}_n = \mathrm{CP}(\mathrm{WSAP}, n)$.

Applying Lemma 2.4.5 and Fact 2.5.2 to this corollary yields:

Theorem 2.5.7
FOLT = CP(WSAP) = CP(SDETAP).

In particular, every first-order definable tree language can be recognized a cascade product of automata from a decidable class. (SDETAP).

2.5.3 Chain Logic

In the previous section, we proved that first-order logic coincides with cascade products of deterministically aperiodic languages. This naturally brings the question: what class of languages coincides with cascade products of arbitrary deterministic languages? In this section we prove that this class has an elegant logical characterization: a language is in CP(SDET) if and only if it is definable in chain logic.

Recall that a set of nodes is a *chain* if it is totally ordered by the relation $\leq$. Formulas of *chain logic* (CL) have the same syntax as monadic second-order logic, but the semantics are different in that the second-order quantifiers are restricted to chains. Chain logic was introduced by Thomas in [62].

Fact 2.5.8 ([62]) FOLT $\subsetneq$ CL $\subsetneq$MSOLT.

Proof
The inclusions are obvious. The language "the length of the leftmost path is even" is in CL and not in FOLT, which can be proved using the Ehrenfeucht-Fraïsé game technique. Boolean expressions that evaluate to true are definable in monadic second-order logic but are not definable in CL (see the example in Section 2.6.1). □

We would like to remark here that *antichain logic*, where quantification is restricted to sets of pairwise incomparable nodes, is strong enough to define all regular tree languages [49].

Before we prove the announced equivalence between CL and cascade products of wordsum automata, we need to define a concept of chain quantification for arbitrary tree languages, not only those obtained from formulas. Given a Σ-tree t and a set $A \subseteq \mathrm{dom}(t)$, we define t_A to be the unique $\{0,1\}$-tree of the same domain as t which has a 1 in exactly the nodes from A. Let L be a tree language over $\Sigma \times \{0,1\}$. The tree language $\exists^c.L$ is:

$$\{t \in \mathrm{Trees}(\Sigma) : t\hat{}t_A \in L \text{ for some chain } A \subseteq \mathrm{dom}(t)\}$$

We are now in a position to prove the periodic equivalent of Theorem 2.5.7:

Theorem 2.5.9
CL = CP(WS) = CP(SDET).

Proof
The second equality was shown in Fact 2.5.2 and we therefore only consider the first one. The right-to-left inclusion can be shown using the same technique as for first-order logic. For the other inclusion, the only nontrivial step is showing the chain quantification:

$$L \in \mathrm{CP(WS)} \quad \Rightarrow \quad \exists^c L \in \mathrm{CP(WS)}.$$

Let then $L \subseteq \mathrm{Trees}(\Sigma \times \{0,1\})$ be a language recognized by an automaton $\mathcal{A}$ which belongs, for some k, to the class WS^k. We will show that $\exists^c L$ belongs to CP(WS). This is done by using a cascade product of several intermediate automata.

One first constructs an automaton $\mathcal{A}^\perp$ over Σ-trees that simulates $\mathcal{A}$ assuming that the $\{0,1\}$ component is always 0. One can prove by induction on k that $\mathcal{A}^\perp$ can be assumed to be in WS^k.

Then, using cascade product, an automaton $\mathcal{B}$ is run over $\mathcal{A}^\perp$, whose state in v says what states are assumed by $\mathcal{A}^\perp$ in the nodes $v \cdot 0$ and $v \cdot 1$. Such an automaton can be found in WS.

Finally, an automaton $\mathcal{C}$ guesses a path π and a labeling of it with $\{0,1\}$. On this path, $\mathcal{C}$ simulates the automaton $\mathcal{A}$ by using $\mathcal{B}$ to read the states

of $\mathcal{A}^{\perp}$ on siblings of nodes on the path. Guessing a path and evaluating something regular on it can be done by a wordsum automaton.

This means that the language $\exists^{c} L$ is recognized by the automaton $\mathcal{C} \circ \mathcal{B} \circ \mathcal{A}^{\perp}$ and hence belongs to CP(WS). □

Seeing how similar the characterizations of FOLT and CL are, one ask ask the natural question:

Under what conditions is a CL language in FOLT?

This turns out to be a rather tricky issue. One example of the inherent difficulties is Lemma 2.3.4.

Another example is related to tree aperiodicity, cf. Definition 2.3.2. One might suppose that an aperiodic CL language would be in FOLT; yet this has been disproved by Potthoff in [48], where a tree language is shown which is both aperiodic and chain-definable, yet not first-order logic definable.

2.5.4 Cascade Product Hierarchies

In this section we show that the CTL* hierarchy depicted in Fig. 2.3 is infinite. Although this result – that the E-nesting hierarchy is infinite – is already known in the literature [55], we choose to include the proof for two reasons. First, it shows how cascade product can be used as an automata-theoretic equivalent of the Ehrenfeucht-Fraïsé game technique. Second, the same proof can be used to show that an analogous hierarchy for chain logic is also infinite.

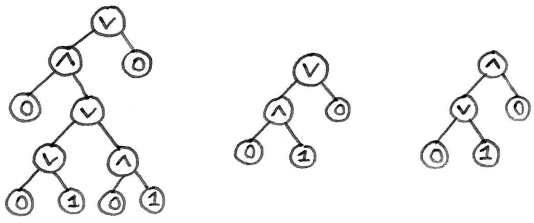


Figure 2.4: A tree with 3-alternation, a tree in $E_2^{\vee}$ and a tree in $E_2^{\wedge}$.

For the proof, we return to the boolean expressions already mentioned in the example in Section 2.3.2. A $\{\wedge, \vee, 0, 1\}$-tree belongs to the set E of *expression encodings* if the leaves are labeled with 0 or 1 and the inner nodes with $\wedge$ or $\vee$. Within E, we distinguish the set L of expression encodings which evaluate to 1. A tree in E is said to have *n-alternation* if it contains a path with at least n occurrences of either $\vee \cdot \wedge$ or $\wedge \cdot \vee$. We write $E_n^{\vee} \subseteq E$ to denote the set of trees which do not contain n-alternation and have $\vee$ in

the root. Similarly we define $E_n^\wedge$. All the $E_n^\wedge$ and $E_n^\vee$ languages are in Π_1. Finally, consider the languages:

$$L_n^\vee = E_n^\vee \cap L \qquad L_n^\wedge = E_n^\wedge \cap L.$$

Of course, L is a union of all the $L_n^\vee$ and $L_n^\wedge$ languages. The following obvious fact can be proved by induction:

Fact 2.5.10 $L_n^\vee \in \Sigma_n$ and $L_n^\wedge \in \Pi_n$

Our aim is to show that the languages $L_n^\vee$ and $L_n^\wedge$ witness the infinity of the E-nesting hierarchy in CTL* . We first present a lemma that will be used several times in proofs which show by induction that a cascade product of wordsum automata cannot recognize some language.

We say an automaton $\mathcal{A}$ *distinguishes* two trees s and t if it evaluates them to different states. In Lemma 2.5.11, we are going to describe runs of a cascade product $\mathcal{B} \circ \mathcal{A}$ over multicontexts whose holes are plugged with combinations of two trees s and t that are not distinguished by $\mathcal{A}$.

Consider a multicontext C and a valuation ν of its holes which uses only the trees s and t. By assumption, the $\mathcal{A}$ component of $\mathcal{B} \circ \mathcal{A}$ labels the multicontext C with the same states regardless of the valuation ν. Therefore, the only part of $\mathcal{B} \circ \mathcal{A}$ that depends on the actual valuation ν is the $\mathcal{B}$ component. Moreover, if $\mathcal{B}$ is a wordsum automaton then only particular properties of the valuation ν can be recognized, where, intuitively, every hole of C is treated independently. This reasoning is formalized in the following Lemma:

Lemma 2.5.11 Let $\mathcal{C}$ be an automaton over Σ-trees of the form

$$\mathcal{C} = \mathrm{WS}(\mathcal{B}) \circ \mathcal{A} \ .$$

Consider a Σ-multicontext C of holes V and two Σ-trees s and t which are not distinguished by $\mathcal{A}$. There exists a set A and a function $\rho : V \times \{s,t\} \to A$ such that for every valuation $\nu : V \to \{s,t\}$, the state to which the automaton $\mathcal{C}$ evaluates the tree $C[\nu]$ depends only upon the value

$$\bigcup_{v \in V} \rho(v, \nu(v)) \quad \subseteq A \ .$$

The size of the set A depends only on the automaton $\mathcal{C}$, not the context C.

Proof
Given a Σ-multicontext C and two valuations $\nu, \mu : V \to \{s,t\}$, from the indistinguishability of s and t by $\mathcal{A}$ it follows that

$$(C[\nu])^{\mathcal{A}}(v) = (C[\mu])^{\mathcal{A}}(v) \qquad \text{for all } v \in \mathrm{dom}(C) \setminus \mathrm{holes}(C). \tag{2.2}$$

Hence it makes sense to talk about a tree $\hat{C}$ of domain $\mathrm{dom}(C) \setminus \mathrm{holes}(C)$ defined $\hat{C}(v) = (C[\nu])^{\mathcal{A}}(v)$ without specifying the valuation ν. Consider the states of $\mathcal{C}$ to which it evaluates the trees s and t:

$$s^{\mathcal{C}}(\varepsilon) = (R, r) \quad t^{\mathcal{C}}(\varepsilon) = (S, s) \qquad \text{where} \quad s = r \quad R, S \subseteq Q \ .$$

The equation $r = s$ follows from (2.2). We will use the sets R and S to define the function ρ.

Let Q be the state space of the word automaton $\mathcal{A}$. We set A to be $P(Q)$. Given a state $q \in Q$, let $f_v(q)$ be the state assumed by the word automaton $\mathcal{B}$ after starting in q and reading the trace in the tree $\hat{C}$ from the hole v to the root. Using the functions f_v and the sets R and S from the previous paragraph, we define the function $\rho : V \times \{s, t\} \to A$ as follows:

$$\rho(v, u) = \begin{cases} f_v(R) & \text{if } u = s; \\ f_v(S) & \text{otherwise.} \end{cases}$$

Having thus defined ρ one can easily verify, using the definitions of wordsum automata and cascade product, that the state to which $\mathcal{C}$ evaluates a tree $C[\nu]$ indeed depends only on the sum in the statement of the lemma. $\square$

Lemma 2.5.12 For all $n \geq 0$, the languages $L^{\vee}_{n+1}, L^{\wedge}_{n+1}$ are not in B_n.

Proof

The proof is by induction on n. We will show that given an automaton $\mathcal{A} \in \mathrm{WS}^n$, there exist two trees $s_n, t_n \in E_n$ such that s_n belongs to $L^{\vee}_{n+1}$ and t_n does not; however both are indistinguishable by $\mathcal{A}$. Simultaneously, we will show the analogous statement for $L^{\wedge}_{n+1}$. By Corollary 2.5.6, these two invariants yield the statement of the lemma.

The case of $n = 0$ is obvious. Assume then that we have already proved the statement for $n \geq 0$, and consider $n + 1$. We will only prove the case of $L^{\vee}_{n+1}$, the case of $L^{\wedge}_{n+1}$ being symmetric.

Consider then an automaton in WS^{n+1} of the form

$$\mathcal{C} = \mathrm{WS}(\mathcal{B}) \circ \mathcal{A},$$

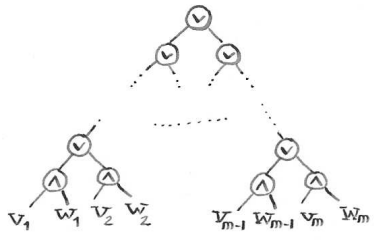
where $\mathcal{A} \in \mathrm{WS}^n$. We will find two trees s_{n+1} and t_{n+1} which are indistinguishable by $\mathcal{C}$ and such that

$$s_{n+1} \in L^{\vee}_{n+2} \qquad t_{n+1} \in E_{n+2} \setminus L^{\vee}_{n+2} \ .$$

Consider first the trees from the induction assumption, which are indistinguishable by the automaton $\mathcal{A}$:

$$s_n \in L^{\wedge}_{n+1} \qquad t_n \in E_{n+1} \setminus L^{\wedge}_{n+1}$$

Consider then the following balanced multicontext C of sufficiently large depth d with $2m = 2^d$ holes $V = \{v_1, w_1, \ldots, v_m, w_m\}$:



By Lemma 2.5.11, there exists a set A and a function $\rho : V \times \{s_n, t_n\} \to A$ such that for every valuation $\nu : V \to \{s_n, t_n\}$, the state to which the automaton $\mathcal{C}$ evaluates the tree $C[\nu]$ depends only upon the value

$$R_\nu = \bigcup_{v \in V} \rho(v, \nu(v)) \quad \subseteq A \ . \tag{2.3}$$

If the depth d was large enough, we can find two elements $a, b \in A$ and three distinct indices $i, j, k \in \{1, \ldots, m\}$ such that

$$\begin{aligned} \rho(v_i, s_n) = \rho(v_j, s_n) = \rho(v_k, s_n) = a \ , \\ \rho(v_i, t_n) = \rho(v_j, t_n) = \rho(v_k, t_n) = b \ . \end{aligned}$$

Consider then the two valuations:

$$\begin{aligned} \nu_1(v) = s_n \quad &\text{iff} \quad v \in \{v_i, v_j, w_j\} \\ \nu_2(v) = s_n \quad &\text{iff} \quad v \in \{v_i, w_j\}. \end{aligned}$$

The valuations differ only on the position v_j. However, since the value $\rho(v_j, *)$ is either a or b, and both these elements are already included in the sum R_ν from (2.3), the value R_ν and, consequently, the state of $\mathcal{C}$ does not differentiate between the two valuations ν_1 and ν_2. This is testified by the equation

$$\bigcup_{v \in V} \rho(v, \nu_1(v)) = \bigcup_{v \in V} \rho(v, \nu_2(v)) = \{a, b\} \cup \bigcup_{v \in V \setminus \{v_i, v_j, v_k\}} \rho(v, \nu_1(v)) \ .$$

Since the expression corresponding to $C[\nu_1]$ evaluates to true and the expression corresponding to $C[\nu_2]$ evaluates to false, we can choose these as the trees s_{n+1} and t_{n+1}, completing thus the proof of the lemma. □

Lemma 2.5.13 For all $n > 0$, $\Pi_n \neq \Sigma_n$

Proof
We will prove that $L_n^\vee \notin \Pi_n$ and $L_n^\wedge \notin \Sigma_n$, which, together with Fact 2.5.10, gives the statement of the lemma. We will only do the case of $L_n^\wedge \notin \Sigma_n$, the proof of $L_n^\vee \notin \Pi_n$ being symmetric.

Let $\mathsf{E}\varphi$ be a formula in Σ_n and consider the set $\Psi \subseteq \mathrm{B}_{n-1}$ of all the node subformulas of φ. Using the previous lemma, Corollary 2.5.6 and Lemma 2.5.4 we can find two trees $s \in L_n^\wedge$, $t \in E_n \setminus L_n^\wedge$ which satisfy the same formulas from Ψ. But then, from the CTL* semantics, it follows that $\mathsf{E}\varphi$ cannot recognize $L_n^\wedge$, since either one of the following cases must hold:

$$\wedge[s,t] \models \mathsf{E}\varphi \quad \text{or} \quad \wedge[t,s] \models \mathsf{E}\varphi \quad \text{or} \quad \wedge[s,s] \not\models \mathsf{E}\varphi.$$

□

Lemma 2.5.14 For all $n > 0$, $\mathrm{B}_n \subsetneq \Delta_{n+1}$.

Proof
An $\{a, \vee, \wedge, 0, 1\}$-tree t is *n-well-formed* if all nodes on the leftmost path are

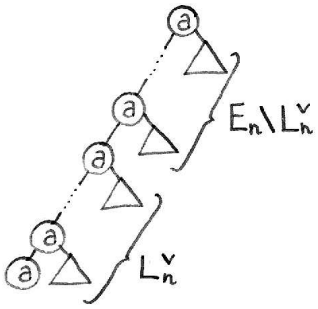


Figure 2.5: A tree in K_n

labeled by a and all subtrees outside the leftmost path belong to E_n. Being n-well-formed can be verified by a Π_1 formula φ. Consider the language K_n consisting of n-well-formed trees t such that for all j satisfying $0^j \cdot 1 \in \mathrm{dom}(t)$, the following implication holds (see Fig. 2.5):

$$t|_{0^j \cdot 1} \notin L_n^\vee \quad \Rightarrow \quad t|_{0^k \cdot 1} \notin L_n^\vee \quad \text{for all } k \leq j \ .$$

We claim that K_n witnesses the strictness of $\mathrm{B}_n \subsetneq \Delta_{n+1}$, i. e.

$$K_n \notin \mathrm{B}_n \quad \text{and} \quad K_n \in \Delta_{n+1} \qquad \text{for all } n > 0.$$

The second part is simple. Let $\psi \in \Sigma_k$ be a formula defining $L_k^\vee$ obtained from Fact 2.5.10. One can prove that there is a formula $\psi^1 \in \Sigma_k$ such that for every tree t:

$$t \models \psi^1 \quad \text{iff} \quad t|_1 \models \psi.$$

The language K_n can then be equivalently defined by both formulas written below and therefore belongs to Δ_{n+1}:

$$\mathsf{E}[\varphi \wedge \mathsf{X}(\mathsf{G}S_0) \wedge \mathsf{G}(\psi^1 \Rightarrow \mathsf{G}\psi^1)] \in \Sigma_{n+1};$$
$$\mathsf{A}[\varphi \wedge \mathsf{X}(\mathsf{G}S_0) \Rightarrow \mathsf{G}(\psi^1 \Rightarrow \mathsf{G}\psi^1)] \in \Pi_{n+1}.$$

We will now prove that $K_n \notin \mathrm{CP}(\mathrm{WS}, n)$, which implies $K_n \notin \mathrm{B}_n$. Let us assume, for the sake of contradiction, that K_n is recognized by an automaton

$$\mathcal{C} = \mathrm{WS}(\mathcal{B}) \circ \mathcal{A}$$

with $\mathcal{A} \in \mathrm{WS}^{n-1}$. Let $s \in L_n^\vee$ and $t \in E_n \setminus L_n^\vee$ be trees, obtained from Lemma 2.5.12, which are indistinguishable by $\mathcal{A}$. Let $N \in \mathbb{N}$ be a sufficiently large number and consider a multicontext C defined:

$$C(0^k) = a \text{ for } k \leq N \qquad C(0^k \cdot 1) = * \text{ for } k < N.$$

By Lemma 2.5.11, there exists a set A and a function $\rho : V \times \{s, t\} \to A$ such that for every valuation $\nu : V \to \{s, t\}$, the state to which the automaton $\mathcal{C}$ evaluates the tree $C[\nu]$ depends only upon the value

$$R_\nu = \bigcup_{v \in V} \rho(v, \nu(v)) \quad \subseteq A \ .$$

By inspecting the construction of ρ in the proof of Lemma 2.5.11, one can show using pumping that there is some m such that if $N > 4m$,

$$\rho(0^k \cdot 1, u) = \rho(0^{k+m} \cdot 1, u) \quad \text{for all } k \in \{m, \ldots, 2m\} \text{ and } u \in \{s, t\}. \quad (2.4)$$

Consider then two valuations ν_1 and ν_2 defined below:

$$\nu_1(0^k \cdot 1) = t \quad \text{iff} \quad k \in \{0, \ldots, m\}$$
$$\nu_2(0^k \cdot 1) = t \quad \text{iff} \quad k \in \{0, \ldots, m, 2m\}.$$

By (2.4), the sets R_{ν_1} and R_{ν_2} are equal, hence the automaton $\mathcal{C}$ does not distinguish between the trees $C[\nu_1]$ and $C[\nu_2]$. However, since the first belongs to K_n and the second one does not, $\mathcal{C}$ does not recognize the language K_n. □

Theorem 2.5.15
All the inclusions of the hierarchy depicted in Fig. 2.3 are strict.

Proof
The strictness of the inclusion $\mathrm{B}_{n-1} \subseteq \Delta_n$ was shown in Lemma 2.5.14. Since any of the below equalities

$$\Pi_n = \mathrm{B}_n \qquad \Sigma_n = \mathrm{B}_n \qquad \Delta_n = \Pi_n \qquad \Delta_n = \Sigma_n$$

would imply, by the symmetry of Π_n and Σ_n, the equality $\Pi_n = \Sigma_n$, all the remaining inclusions in Fig. 2.3 must be strict by Lemma 2.5.13. □

Note that the same proof would yield an analogue of Theorem 2.5.15 for an corresponding hierarchy in CL. As a side note we would like to remark the perhaps surprising result that for all k, the language: "there exist exactly k nodes labeled by a" is in $\mathrm{CP}(\mathrm{WS}, 2)$.

2.6 Confusion Conjecture

In this section we present a conjecture regarding definability in chain logic. This conjecture says that a language is in CL if and only if its syntactic automaton does not contain a certain type of pattern, called confusion. One of the implications in this conjectured equivalence is proved in Section 2.6.1, where it is shown that languages which contain confusion are not in CL. The validity of the other implication remains open, but, in Sections 2.6.2 and 2.6.3, some arguments in its favour are presented.

2.6.1 The Conjecture

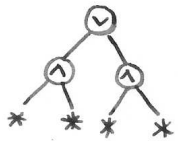
Consider an alphabet Σ and a regular language L over this alphabet. Let Γ be a set of L-types and C a multicontext. By $\mathrm{Val}(C, \Gamma)$ we denote the set of all possible valuations $\mathrm{holes}(C) \to \Gamma$. Any subset $\mathcal{X} \subseteq \mathrm{Val}(C, \Gamma)$ is called a *constraint*, the *trivial constraint* being the whole set $\mathrm{Val}(C, \Gamma)$. Given a type $\alpha \in \Gamma$ and a hole v, the pair (α, v) is $\mathcal{X}$*-legal* if there is some valuation $\nu \in \mathcal{X}$ which assigns α to v.

Definition 2.6.1 Let L be a language and Γ a set of at least two L-types. Let C be a multicontext along with a set $\mathcal{X} \subseteq \mathrm{Val}(C, \Gamma)$ of valuations. The pair $(C, \mathcal{X})$ is Γ*-confusion* if

1. For every valuation $\nu \in \mathcal{X}$, the type $C[\nu]$ belongs to Γ;
2. For every type $\beta \in \Gamma$ and every $\mathcal{X}$-legal pair (α, v), there is a valuation $\nu \in \mathcal{X}$ such that $\nu(v) = \alpha$ and $C[\nu] = \beta$.

A language without confusion is called *nonconfusing*. We use NC to denote the class of nonconfusing languages. Intuitively, confusion means that in the multicontext C, there is no hole which, when plugged with a legal type, "fixes" the type of the tree in the sense of narrowing the set of possible types, just like a 1 fixes the type of a tree of $\vee$'s and a 0 fixes the type of a tree of $\wedge$'s.

Example: Recall the language L of properly formed boolean expressions evaluating to 1 defined in Section 2.5.4. The set of L-types is $\{0, 1, \perp\}$, $\perp$ being the set of malformed trees. The language L contains confusion, since the following multicontext is $\{0, 1\}$-confusing under the trivial constraint:



Note that this is a minimal confusing multicontext. □

In a moment, we will show in Lemma 2.6.4 that a language with confusion cannot be definable in chain-logic. By our failure to find a nonconfusing language beyond CL, we propose:

Conjecture 2.6.2 (Confusion conjecture) CL=NC.

We would like to remark that we have not even found a language where more than two types are needed for Γ, nor one where a nontrivial constraint is necessary.

We now proceed to show that a confusing language cannot be in CL. The notion of an $\mathcal{X}$-legal pair is extended to valuations in the following manner. Let $\mathcal{X} \subseteq \mathrm{Val}(C, \Gamma)$. A valuation $\mu : W \to \Gamma$ is said to be $\mathcal{X}$-legal if it can be expanded to a valuation in $\mathcal{X}$. In particular, W needs to be a set of holes in C. The pair $(C, \mathcal{X})$ is called *i-fold Γ-confusion* if for every type β in Γ, every $\mathcal{X}$-legal valuation of domain no bigger than i can be expanded to a valuation ν in $\mathcal{X}$ such that $C[\nu] = \beta$.

Lemma 2.6.3 If there is Γ-confusion, then there is i-fold Γ-confusion for every $i \geq 1$.

Proof
The proof is by induction on i. For $i = 1$ we take the Γ-confusion $(C, \mathcal{X})$ from the assumption. Consider some $i \geq 1$ and let $(C^i, \mathcal{X}^i)$ be the i-fold confusion from the induction assumption. To construct $(C^{i+1}, \mathcal{X}^{i+1})$ we take C and substitute C^i for each of the holes of C. The constraint $\mathcal{X}^{i+1}$ needs

to be defined as a sort of composition of the constraints $\mathcal{X}$ and $\mathcal{X}^i$. With a valuation $\nu : \mathrm{holes}(C^{i+1}) \to \Gamma$ we associate:

- For every hole v of C, the valuation $\nu|v : \mathrm{holes}(C^i) \to \Gamma$, which assigns to w the type $\nu(v \cdot w)$.
- The valuation $\nu|\varepsilon : \mathrm{holes}(C) \to \Gamma$, which assigns to v the type of the tree C^{i+1} in the node v (this is the type $C[\nu|v]$).

The constraint $\mathcal{X}^{i+1}$ is defined to be the set of those valuations ν such that both $\nu|\varepsilon$ belongs to $\mathcal{X}$ and all the $\nu|v$ valuations belong to $\mathcal{X}^i$.

We will verify now that $(C^{i+1}, \mathcal{X}^{i+1})$ is indeed $(i+1)$-fold Γ-confusion. Consider any valuation μ which is $\mathcal{X}^{i+1}$-legal and whose domain consists of at most i holes of C^{i+1}. If these holes are chosen in more than one copy of C^i then every copy of C^i can have arbitrary type, so C^{i+1} can have arbitrary type. If all the holes are chosen in one copy of C^i, then the type of this copy may be determined to some type α. But this is only one hole v of the multicontext C. Moreover, the pair (α, v) is $\mathcal{X}$-legal by assumption that μ is $\mathcal{X}^{i+1}$-legal. As the remaining holes are unconstrained, C^{i+1} may have arbitrary type. □

Lemma 2.6.4 A confusing language is not in CL.

Proof
Let L be a language containing Γ-confusion $(C, \mathcal{X})$, with $\Gamma = \{\alpha_1, \ldots, \alpha_m\}$. We prove by induction on k that for every automaton $\mathcal{C}$ in WS^k, we can define a set of trees $T^k = \{t_1, \ldots, t_m\}$ such that

$$t_1^{\mathcal{C}}(\varepsilon) = \cdots = t_m^{\mathcal{C}}(\varepsilon) \quad \text{and} \quad \mathrm{type}(t_i) = \alpha_i \text{ for all } i \leq m.$$

We will show that no automaton $\mathcal{C}$ in WS^k can recognize L. This is of course sufficient to show that the language L is not in CL. The case $k = 0$ is obvious. For the induction step, consider some level $k \geq 0$ and an automaton

$$\mathcal{C} = \mathrm{WS}(\mathcal{B}) \circ \mathcal{A} \qquad \text{with } \mathcal{A} \in \mathrm{WS}^k.$$

in WS^{k+1}. We assume that, using the induction assumption, we have constructed the set of trees $T^k = \{t_1, \ldots t_m\}$ with the required properties for the automaton $\mathcal{A}$. We are going to show how to construct the set T^{k+1} for the automaton $\mathcal{C}$. Let n be a number bigger than the size of the set A from Lemma 2.5.11 appropriate for the automaton $\mathcal{C}$. Consider the n-fold amplification $(C, \mathcal{X})$ of the Γ-confusion we have assumed to exist. Let V be the set of holes in C.

By an obvious generalization Lemma 2.5.11 from the two trees s and t to the m trees in T^k, there exists a function $\rho : V \times T^k \to A$ such that for every valuation $\nu : V \to T^k$, the state to which the automaton $\mathcal{C}$ evaluates the tree $C[\nu]$ depends only upon the value

$$R_\nu = \bigcup_{v \in V} \rho(v, \nu(v)) \quad \subseteq A \ .$$

Let W be a subset of V. With a valuation $\mu : W \to T^k$ we associate the set

$$R_\mu = \bigcup_{v \in W} \rho(v, \mu(v)) \quad \subseteq A \ .$$

Consider a set W of no more than n elements and an $\mathcal{X}$-legal valuation $\mu : W \to T^k$ whose every expansion $\nu : V \to T^k$ satisfies $R_\mu = R_\nu$. Such W and μ can be found by successively adding elements to W. But then, no matter what expansion ν of μ is chosen, the automaton $\mathcal{C}$ evaluates the tree $C[\nu]$ to the same state.

However, since only n holes are fixed in the valuation μ and since C is n-fold confusion, by Lemma 2.6.3 we can find for every $i \in \{1, \ldots, m\}$ an expansion $\nu_i \in \mathcal{X}$ of the valuation μ such that the type of the tree $C[\nu_i]$ is α_i. This goes to show that the set

$$T^{k+1} = \{C[\nu_1], \ldots, C[\nu_m]\}$$

satisfies the desired properties. □

Lemma 2.6.5 NC membership is decidable.

Proof
The general idea is that the existence of confusion can be expressed in monadic second-order logic. However, we need to do a little coding to express the constraint $\mathcal{X}$.

Let C be a multicontext of holes V. With a set of types Γ and a set $A \subseteq \Gamma \times V$, we associate the the following constraint $\mathcal{X}(A, \Gamma)$:

$$\{\nu \in \Gamma^V : C[\nu] \in \Gamma \text{ and } (\nu(v), v) \in A \text{ for all } v \in V\}$$

This is the maximal constraint which ensures that the result is in Γ but the legal pairs are in A.

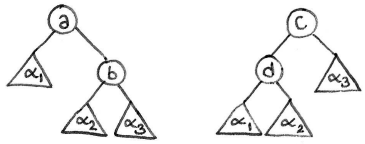
If $(C, \mathcal{X})$ is confusion and A is the set of $\mathcal{X}$-legal pairs, then $(C, \mathcal{X}(A, \Gamma))$ is also confusion. Therefore a language contains confusion if and only if for some context C, some set of types Γ and some $A \subseteq \Gamma \times \text{holes}(V)$, the pair $(C, \mathcal{X}(A, \Gamma))$ is Γ-confusion. But this can be expressed in monadic second-order logic. □

2.6.2 Evidence in Favor of the Confusion Conjecture

In the previous section it was shown that a confusing language cannot be in CL. However, the validity of the reverse implication, and consequently of the confusion conjecture, remains open. The remainder of Section 2.6 is devoted to presenting some evidence in its favor. By doing an exhaustive search, once can show that the confusion conjecture is true when restricted to languages with two types. In a moment we will also give a more involved class of languages satisfying the conjecture, i.e. frontier languages. Finally, in Section 2.6.3, we will prove that the class NC of nonconfusing languages is closed under some of the same operations as CL, in particular chain quantification and boolean operations.

We now proceed to show that the conjecture is true for so-called frontier languages. The *frontier* of a Σ-tree t is the sequence $\mathrm{fr}(t) \in \Sigma^*$ of labels in the leaves of t, read from left to right. The frontier $\mathrm{fr}(L)$ of a word language L is the set of Σ-trees whose frontier belongs to L. Such a set of trees is called a *frontier language*.

Note that a tree language L is a frontier language if and only if for all L-types $\alpha_1, \alpha_2, \alpha_3$ and all letters a, b, c, d, the types of the two trees below are the same:



It follows that being a frontier language is decidable and that the regular tree languages which are frontier languages are exactly the frontier languages obtained from regular word languages.

Lemma 2.6.6 If $L \in$ FOLW, then $\mathrm{fr}(L) \in$ FOLT.

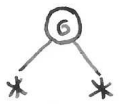
Proof
This follows from the fact that the lexicographic ordering of leaves is first-order definable in a tree. □

Lemma 2.6.7 If $L \notin$ FOLW, then $\mathrm{fr}(L)$ is a confusing language.

Proof
If L is not first-order definable, then by Schützenberger's Theorem it is periodic, i.e. there is a word type α such that

$$\alpha \cdot \alpha \neq \alpha \quad \text{but} \quad \alpha^n = \alpha \text{ for some } n \geq 2 \ .$$

Since the types of the tree language $\mathrm{fr}(L)$ are isomorphic to the types of the word language L, one can easily verify that for any letter σ, the following multicontext is $\{\alpha, \ldots, \alpha^{n-1}\}$-confusing with the trivial constraint:



□

Corollary 2.6.8 For frontier languages, FOLT=CL=NC.

2.6.3 NC Closure Properties

In this section we demonstrate that the class NC is closed under chain quantification and boolean operations. In order to prove this result, we show that nonconfusing automata are closed under cascade product and homomorphic images.

Let $\mathcal{A}$ be an automaton over Σ-trees and R a subset of its states. The notion of R-confusion $(C, \mathcal{X})$ in the automaton is defined analogously to confusion over types. Since there is a one-to-one correspondence between types and states of the syntactic automaton, we obtain:

Fact 2.6.9 A language is nonconfusing if and only if its syntactic automaton is nonconfusing.

Homomorphic Images

In this section we show that a homomorphic image of a nonconfusing automaton is also nonconfusing.

Definition 2.6.10 Given two automata $\mathcal{A}$ and $\mathcal{B}$ over the same alphabet Σ, we say a function h assigning states of $\mathcal{B}$ to states of $\mathcal{A}$ is an *automaton homomorphism* if for all states q, q_0, q_1 of $\mathcal{A}$ and all letters $\sigma \in \Sigma$,

$$\delta^{\mathcal{B}}(h(q_0), h(q_1), \sigma) = h(\delta^{\mathcal{A}}(q_0, q_1, \sigma)).$$

The automaton $\mathcal{B}$ is called a *homomorphic image* of $\mathcal{B}$.

Homomorphic images are important due to the following property, which is proved the same way as the analogous property of word automata:

Fact 2.6.11 If $\mathcal{A}$ recognizes L, then the syntactic automaton of L is a homomorphic image of $\mathcal{A}$.

Since we will be simultaneously using valuations over different sets of states, different contexts and different automata, we are going to use following notation to avoid some misunderstandings in the proofs to come. Let $\mathcal{A}$ be an automaton and R a set of its states. Given a multicontext C, an $(\mathcal{A}, C, R)$-*valuation* is any function $\nu : \mathrm{holes}(C) \to R$.

Let us fix for the rest of this section an automaton $\mathcal{B}$ over Σ which is a homomorphic image of some automaton $\mathcal{A}$ over Σ via a homomorphism

$$h : Q^{\mathcal{A}} \to Q^{\mathcal{B}} \ .$$

Let $\sim$ be the equivalence relation on $Q^{\mathcal{A}}$ identifying states with the same image under h. We extend the relation $\sim$ onto valuations, writing $\mu \sim \nu$ if $h \circ \mu = h \circ \nu$. Let R be a subset of $Q^{\mathcal{A}}$. A set of $(\mathcal{A}, C, R)$-valuations $\mathcal{X}$ is said to *respect* (R, h) if every two $\sim$-equivalent $(\mathcal{A}, C, R)$-valuations either both belong to $\mathcal{X}$ or both are outside $\mathcal{X}$.

The following definition is a technical generalization of the notion of confusion. Apart from additionally requiring that $\mathcal{X}$ respect (R, h), item 2 of Definition 2.6.1 is weakened by only requiring the states obtained in the root of the multicontext to be specified up to the equivalence relation $\sim$. The definition is tailored to go through the induction in Lemma 2.6.14.

Definition 2.6.12 Let R be a set of at least two states of $\mathcal{A}$, and C a multicontext along with a set $\mathcal{X}$ of $(\mathcal{A}, C, R)$-valuations. The pair $(C, \mathcal{X})$ is called *R-pseudoconfusion* if

1. For every valuation $\nu \in \mathcal{X}$, the state $C[\nu]$ belongs to R;
2. For every state $r \in R$ and every $\mathcal{X}$-legal pair (q, v), there is a valuation $\nu \in \mathcal{X}$ such that $\nu(v) = q$ and $C[\nu] \sim r$;
3. The constraint $\mathcal{X}$ respects (R, h).

Lemma 2.6.13 If $\mathcal{B}$ contains confusion, then $\mathcal{A}$ contains pseudoconfusion.

Proof
Assume that for some subset S of the state space of $\mathcal{B}$, the pair $(C, \mathcal{Y})$ is S-confusion in $\mathcal{B}$. Let $R = h^{-1}(S)$ and let $\mathcal{X}$ be the set of those valuations ν such that $h \circ \nu \in \mathcal{Y}$. One can easily verify that $(C, \mathcal{X})$ is R-pseudoconfusion in $\mathcal{A}$. □

The following is the key technical lemma regarding pseudoconfusion:

Lemma 2.6.14 If $\mathcal{A}$ contains pseudoconfusion, then it contains confusion.

Proof
Induction on the size of the set of states R for which the pseudoconfusion holds. Let $(C, \mathcal{X})$ be R-pseudoconfusion in $\mathcal{A}$. The base case is where $|R| = |h(R)|$. In this case $C[\nu] \sim r$ implies $C[\nu] = r$ and hence the pseudoconfusion is actually confusion.

Consider then pseudoconfusion with R having more than $|h(R)|$ elements. Let $v_1, \ldots, v_n$ be the holes in C. It may be the case that $(C, \mathcal{X})$ is R-confusion and we are done. Otherwise there must be a state $p \in R$ and some hole $v_i \in \{v_1, \ldots, v_n\}$ such that the pair (p, v_i) is $\mathcal{X}$-legal and the set

$$S = \{C[\nu] : \nu \in \mathcal{X} \text{ and } \nu(v_i) = p\}$$

is a proper subset of R (it must be a subset by definition of pseudoconfusion). We fix this index i for the rest of the proof.

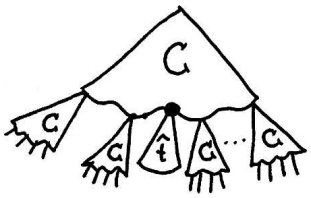


Figure 2.6: The multicontext E

Since $(C, \mathcal{X})$ is R-pseudoconfusion in $\mathcal{A}$, for every state $r \in R$ there is a valuation $\nu \in \mathcal{X}$ such that $\nu(v_i) = p$ and the state $C[\nu] \in S$ is $\sim$-equivalent to r, i.e. has the some image under h as r. Therefore,

$$h(R) \quad = \quad h(S).$$

Let $\hat{t}$ be a tree which evaluates to p under $\mathcal{A}$ and let D be the multicontext obtained from C by substituting $\hat{t}$ for v_i while leaving the other holes open. Finally, let E be the multicontext obtained from D by substituting C into all the holes. The holes in E, the set of which we denote by V, are

$$\{v_j \cdot v_k : j \in I, k \in J\} \qquad \text{with } J = \{1, \ldots, n\} \text{ and } I = J \setminus \{i\}. \tag{2.5}$$

We will show that the multicontext E, along with a certain constraint $\mathcal{Y}$, is S-pseudoconfusion. This constraint $\mathcal{Y}$ will be defined by requiring certain parts of a $(\mathcal{A}, E, R)$-valuation to conform to the original constraint $\mathcal{X}$. The precise definition follows.

As in the proof of Lemma 2.6.3, with an $(\mathcal{A}, E, R)$-valuation μ we associate two types of derived valuation: an $(\mathcal{A}, C, R)$-valuation $\nu|j$ defined for

every $j \in I$ and an $(\mathcal{A}, D, R)$-valuation $\nu|\varepsilon$. These are defined:

$$\begin{array}{rcll} \nu|j(v_k) & = & \nu(v_j \cdot v_k) & \text{for } k \in J; \\ \nu|\varepsilon(v_k) & = & C[\nu|k] & \text{for } k \in I. \end{array}$$

Given an $(\mathcal{A}, D, R)$ valuation ν, let ν^C be the $(\mathcal{A}, C, R)$ valuation obtained from ν by additionally mapping v_i to the state p which "narrows down" C. The aforementioned constraint $\mathcal{Y}$ is the following set of $(\mathcal{A}, E, S)$-valuations:

$$\mathcal{Y} = \{\nu : (\nu|\varepsilon)^C \in \mathcal{X} \text{ and } \nu|j \in \mathcal{X} \text{ for all } j \in I\}.$$

We are now going show that $(E, \mathcal{Y})$ is S-pseudoconfusion in $\mathcal{A}$. For this we need to prove the three items from Definition 2.6.12.

3. First we show that $\mathcal{Y}$ respects (S, h). Let ν, μ be two $(\mathcal{A}, E, S)$-valuations. If $\nu \sim \mu$, then all the derived valuations $\nu|\varepsilon$, $\mu|\varepsilon$ and $\nu|i$, $\mu|i$ are respectively $\sim$-equivalent. Since $\mathcal{X}$ respects (R, h), either both ν and μ belong to the set $\mathcal{Y}$ or both are outside it.

1. We need to show that $E[\nu] \in S$ for all $\nu \in \mathcal{Y}$. Let $\nu \in \mathcal{Y}$ and consider $E[\nu]$. By the assumption that ν belongs to $\mathcal{Y}$, the states in the nodes v_j all belong to R for $j \in I$. Since the node v_i is fixed with p, and the valuation $(\nu|\varepsilon)^C$ belongs to $\mathcal{X}$, we obtain $E[\nu] \in S$ by definition of the set S.

2. Let (q, v) be a $\mathcal{Y}$-legal pair and r a state in S. We need to show that there exists an $(\mathcal{A}, E, S)$-valuation $\mu \in \mathcal{Y}$ such that

$$\begin{array}{rcll} \mu(v) & = & q, \quad \text{and} & (2.6) \\ E[\mu] & \sim & r. & (2.7) \end{array}$$

   By (2.5), the hole v is of the form $v_j \cdot v_k$ for some $j \in I$ and $k \in J$, which we fix for the rest of the proof. By definition of the set S, there is some $(\mathcal{A}, D, R)$-valuation ν such that $D[\nu] = r$ and ν^C belongs to $\mathcal{X}$. We are going to try to expand this valuation into the valuation μ for the big multicontext E. In the process however, the value $E[\mu]$ may change to some other state, yet will remain in S and will still be $\sim$-equivalent to r.

   By assumption on the multicontext C, for every $m \in I$, there exists a $(\mathcal{A}, C, R)$-valuation $\nu_m \in \mathcal{X}$ such that:

$$R \ni C[\nu_m] \sim \nu(v_m). \quad (2.8)$$

Moreover, the ν_j valuation can be assumed to satisfy

$$\nu_j(v_k) \quad = \quad q. \tag{2.9}$$

Since $h(S) = h(R)$ and $\mathcal{X}$ respects (R, h), we may also assume that all these valuations are actually $(\mathcal{A}, C, S)$-valuations.

We have started with the valuation ν and created based on it some valuations ν_m for $m \in I$. Unfortunately, these valuations do not compose correctly, since in (2.8) only $\sim$-equivalence between the output of ν_m and the input of ν is postulated. That is why we come back to the valuation ν and modify it into one which does compose with the valuations ν_m: an $(\mathcal{A}, D, S)$-valuation $\hat{\nu}$ defined

$$\hat{\nu}(v_m) = C[\nu_m] \qquad \text{for } m \in I.$$

Since $\mathcal{X}$ respects (R, h) and ν^C belongs to $\mathcal{X}$, then also $(\hat{\nu})^C$ belongs to $\mathcal{X}$, by (2.8). Since h is a homomorphism, $h(C[\hat{\nu}]) = h(C[\nu]) = h(r)$, hence

$$C[\hat{\nu}] \quad \sim \quad r. \tag{2.10}$$

We are now ready to define the desired $(\mathcal{A}, E, S)$-valuation μ:

$$\mu(v_l \cdot v_m) = \nu_l(v_m) \qquad \text{for all } l \in I, m \in J.$$

We show that μ belongs to $\mathcal{Y}$ and satisfies equations (2.6) and (2.7). By definition, $\mu|m = \nu_m$ for $m \in I$ and $\mu|\varepsilon = \hat{\nu}$. Since $(\hat{\nu})^C$ and all the ν_m valuations belong to $\mathcal{X}$, we have $\mu \in \mathcal{Y}$. By (2.9), μ satisfies (2.6). By construction of μ and E, we have $E[\mu] = C[\hat{\nu}]$, hence (2.7), follows from (2.10).

□

Since the automata $\mathcal{A}$ and $\mathcal{B}$ and the homomorphism h were picked arbitrarily, we obtain from Lemmas 2.6.13 and 2.6.14 the following corollary:

Corollary 2.6.15 The homomorphic image of a nonconfusing automaton is nonconfusing.

Products

In this section we consider cascade and cartesian products of nonconfusing automata.

Lemma 2.6.16 If both $\mathcal{A}$ and $\mathcal{B}$ are nonconfusing, then so is $\mathcal{B} \circ \mathcal{A}$.

Proof
We will prove that if $\mathcal{B} \circ \mathcal{A}$ contains confusion, then either one of $\mathcal{A}$ or $\mathcal{B}$ does too. Let $(C, \mathcal{X})$ be Q-confusion in $\mathcal{B} \circ \mathcal{A}$, with $Q \subseteq Q^{\mathcal{B}} \times Q^{\mathcal{A}}$ being a subset of the state space of $\mathcal{B} \circ \mathcal{A}$. Consider two cases:

- The projection of Q onto $Q^{\mathcal{A}}$ has more than one element. Let $\mathcal{Y}$ be the projection onto $Q^{\mathcal{A}}$ of the constraint $\mathcal{X}$. Then $(C, \mathcal{Y})$ is confusing for $\mathcal{A}$, since the first element in the automaton $\mathcal{B} \circ \mathcal{A}$ behaves as in the automaton $\mathcal{A}$.

- $Q = R \times \{q\}$ for some $q \in Q^{\mathcal{A}}$ and $R \subseteq Q^{\mathcal{B}}$. Let D be the $\Sigma \times Q^{\mathcal{A}}$-multicontext obtained from C by additionally labeling each node with the state assumed by $\mathcal{A}$ provided that in the holes of C state q is assumed. It is now easy to check that D along with the obvious constraint give R-confusion in $\mathcal{B}$.

□

We omit the obvious proof of the following fact:

Fact 2.6.17 The cartesian product of two nonconfusing automata is nonconfusing.

Closure of NC

Now we are ready to show that the class NC is closed under the basic operations definable in CL. Note that the following theorem would be a straightforward corollary if the confusion conjecture were true.

Theorem 2.6.18
NC is closed under boolean operations and chain quantification.

Proof
The syntactic automaton of a boolean combination of languages is, by Fact 2.6.11, a homomorphic image of the cartesian product of syntactic automata of these languages, which proves the first part of the theorem.

For the second part, consider a nonconfusing language L. By the proof of Theorem 2.5.9, the language $\exists^c L$ is recognized by an automaton

$$\mathcal{C} \circ \mathcal{B} \circ \mathcal{A}^{\perp}$$

where $\mathcal{A}^{\perp}$ is a nonconfusing automaton obtained from the syntactic automaton $\mathcal{A}$ of L, while $\mathcal{C} \circ \mathcal{B}$ is a nonconfusing automaton representing the chain

quantifier. By Lemma 2.6.16, the automaton $\mathcal{C} \circ \mathcal{B} \circ \mathcal{A}^{\perp}$ is nonconfusing. By Fact 2.6.11, the syntactic automaton of $\exists^c L$ is also nonconfusing, hence the language is in NC. $\square$

2.7 Open Problems

In this section we would like to recall the most important open problems concerning this chapter. Naturally, a decidable characterization of either first-order logic over trees or chain logic would considered a breakthrough. However, a seemingly simpler problem remains to be resolved: is it decidable whether a language is recognized by an (aperiodic) wordsum automaton? Finally, it remains to be seen whether the definition of confusion can be limited to sets of two types or trivial constraints.

Chapter 3

Logics with the Operators EX and EF

We continue in this chapter our study of the definability problem for tree logics. The reader might recall that the characterizations in the previous chapter did not yield decision procedures for first-order or chain logic definability. In this chapter, we consider some fragments of first-order logic which are simple enough to make the definability problem decidable.

The fragments in question are restrictions of CTL* where the only modalities allowed are EX (there is a successor) and EF (there is a descendant). Apart from perhaps being a step towards solving the first-order definability problem, these logics may be of some independent interest. In some cases the model-checking problem for them is easier than for CTL* (and even CTL); for instance when the model is given by a BPP [22] or by a a push-down system [68]. The operators EX and EF are also closely related to the path operators of XPath [25, 24].

We prove the definability problem decidable for three logics: TL[EX], TL[EF] and TL[EX, EF]. These are built using the eponymous operators along with boolean connectives. Our decision procedures use a sort of forbidden pattern characterization that is expressed in terms of types of the given language. The resulting algorithms are polynomial in the number of types. If, on the other hand, we assume that the input is a CTL formula or a nondeterministic tree automaton, then definability is EXPTIME-complete.

The plan of this chapter is as follows. After a preliminary section we briefly state a characterization of the logic TL[EX]. This is very similar to a characterization of modal logic presented in the literature [46], so we mention the result mostly for completeness. In the next two sections, we characterize the logics TL[EF] and TL[EX, EF] respectively. Maybe counterintuitively, the argument for the weaker logic is longer. In the penultimate section we sum-

marize the results, showing how they imply decidability algorithms. Finally, we justify our characterizations by pointing out why the forbidden patterns known from the word case do not adapt directly to the tree case.

3.1 EX+EF Formulas

EX+EF formulas are CTL* formulas which use boolean connectives, letter symbols and where the modalities U, E and A are allowed only in the forms EX (exists next) and EF (exists finally).

Although formula satisfaction was already defined in the section on CTL*, we repeat the definition here for this restricted case:

- A tree satisfies the formula a if its root is labeled by a;
- Satisfaction for boolean operations is defined in the standard way;
- A tree satisfies $\mathsf{EX}\varphi$ if one the subtrees rooted in its sons satisfies φ;
- A tree satisfies $\mathsf{EF}\varphi$ if it has a proper subtree that satisfies φ.

Observe that the modality EF has strict semantics here: the appropriate subtree has to be proper. The formula $\mathsf{AX}\varphi$ is used as an abbreviation of $\neg\mathsf{EX}\neg\varphi$, while $\mathsf{AG}\varphi$ is used as an abbreviation of $\neg\mathsf{EF}\neg\varphi$.

Given a set $\mathcal{G}$ of EX+EF formulas, we say that a tree language is $\mathcal{G}$-*definable* if there exists a formula in $\mathcal{G}$ that defines it. Given a set of modalities $\mathcal{M} \subseteq \{\mathsf{EF}, \mathsf{EX}\}$, we write TL($\mathcal{M}$) to denote the set of formulas constructed using boolean operations, letter constants and modalities from $\mathcal{M}$. We will be considering three instances in this chapter: TL[EX], TL[EF] and TL[EX, EF].

3.2 TL[EX]

In this section we state a characterization of TL[EX]-definable languages. We do this for the sake of completeness since the characterization is essentially the same as in [46].

Definition 3.2.1 Two trees are *identical up to depth* k if they are the same when restricted to $\{0,1\}^{\leq k}$. We say that a language L is *dependent on depth* k if every two trees which are identical up to depth k have the same L-type.

A context is *nontrivial* if its hole is not in the root.

Definition 3.2.2 Let L be a language and let α, β be two distinct L-types. We say that the language L contains an $\{\alpha, \beta\}$-*loop* if for some nontrivial context $C[]$, both $C[\alpha] = \alpha$ and $C[\beta] = \beta$ hold.

Theorem 3.2.3
For a regular language L, the following conditions are equivalent:

1. *L is* TL[EX]*-definable;*
2. *For some $k \in \mathbb{N}$, L is dependent on depth k;*
3. *L does not have an $\{\alpha, \beta\}$-loop for any two L-types α, β .*

Proof
The equivalence of the first two conditions is obvious, as is the implication from 2 to 3. To end the proof of the theorem, we will show that if the language L is not dependent on any depth k, then a loop can be found.

Let $k > |\text{Types}(L)|^2$ and assume that L is not dependent on depth k. This means there are trees s and t which are identical up to depth k but have different types. Let $v_1, \ldots, v_n$ be all the nodes of depth k in the tree s (or equivalently in t). We define a sequence of trees $s = s_0, \ldots, s_n = t$ which gradually morphs from the tree s to the tree t:

$$s_0 = s \qquad \text{and} \qquad s_{i+1} = s_i[v_i := t|_{v_i}] \ .$$

Since the trees s_0 and s_n have different types, there must be some $i \in \{1, \ldots, n\}$ such that s_{i-1} and s_i have different types. These two latter trees differ only below the node v_i. Let $w_0 < \ldots < w_{k-1}$ be all the ancestors of the node v_i. Given $j < k$, let

$$\alpha_j = \mathit{type}(s_{i-1}|_{w_j}) \qquad \text{and} \qquad \beta_j = \mathit{type}(s_i|_{w_j}).$$

Since the node v_i is at depth $k > |\text{Types}(L)|^2$, there must be some two indices $j < k$ such that the equalities $\alpha_j = \alpha_k$ and $\beta_j = \beta_k$ hold. Since the types of s_{i-1} and s_i are distinct, so are the types α_j and β_j. But this means that the part of s_{i-1} whose root is in w_j and whose hole is in w_k provides an $\{\alpha_j, \beta_j\}$-loop. □

3.3 TL[EF]

In this section we show a characterization of TL[EF]-definable languages. This is the most involved section of the chapter, with a long technical proof.

Before we can formulate the main theorem (Theorem 3.3.2) we need some auxiliary definitions. We start with the key definition in this section: that of a delayed type.

Given a Σ-tree t and a letter $a \in \Sigma$, we write $t\langle a\rangle$ to denote the tree obtained from t by relabeling the root with the letter a. With every Σ-tree t we associate its *delayed type*, which is the function:

$$\mathit{dtype}_L(t) : \Sigma \to \mathrm{Types}(L) \qquad \text{defined} \quad \mathit{dtype}_L(t)(a) = \mathit{type}_L(t\langle a\rangle).$$

Note that the delayed type of a tree does not depend on the letter labeling its root. We will denote delayed types using the letters x, y, z. We write $(x, a) \trianglelefteq_L y$ if there is a tree of delayed type y having a subtree of type $x(a)$. We also write $x \trianglelefteq_L y$ if $(x, a) \trianglelefteq_L y$ for some $a \in \Sigma$. This relation is a quasiorder but not necessarily a partial order, since it may not be antisymmetric.

For delayed types x, y and letters $a, b \in \Sigma$, we write $\mathit{dtype}_L(x, a, y, b)$ for the delayed type which assigns to a letter c the type $c[x(a), y(b)]$. In other words, this is the delayed type of a tree whose left and right subtrees have types $x(a)$ and $y(b)$ respectively. The set of *neutral letters* of a delayed type x is the set

$$N_x^L = \{a : x = \mathit{dtype}_L(x, a, x, a)\}.$$

This set may be empty.

Definition 3.3.1 A Σ-language L is *EF-admissible* if it is a regular tree language such that all delayed types x, y and all letters $a, c \in \Sigma$ satisfy:

P1 The relation $\trianglelefteq_L$ on delayed types is a partial order;

P2 $\mathit{dtype}_L(x, a, y, b) = \mathit{dtype}_L(x, a, y, b')$ for all $b, b' \in N_y^L$;

P3 if $(x, a) \trianglelefteq_L y$ then $\mathit{dtype}_L(x, a, y, c) = \mathit{dtype}_L(y, c, y, c)$;

P4 $\mathit{dtype}_L(x, a, y, c) = \mathit{dtype}_L(y, c, x, a)$.

Another important concept used in Theorem 3.3.2 is that of typeset dependency. The *typeset* of a Σ-tree t is the set

$$\mathrm{TS}_L(t) = \{\mathit{type}_L(t|_w) : w \in \mathrm{dom}(t) \setminus \{\varepsilon\}\} \ .$$

Note that the type of the tree itself is not necessarily included in its typeset. We say that a language L is *typeset dependent* if the delayed type of a tree depends only on its typeset.

Our characterization of TL[EF] is presented in the following theorem:

Theorem 3.3.2
For a regular language L, the following conditions are equivalent:

1. *L is* TL[EF]*-definable,*
2. *L is typeset dependent,*
3. *L is EF-admissible.*

The proof of this theorem is long and will be spread across the next three sections; the implications $1 \Rightarrow 2$, $2 \Rightarrow 3$ and $3 \Rightarrow 1$ being proved in Sections 3.3.1, 3.3.2 and 3.3.3 respectively. For the remainder of Section 3.3 we assume that an alphabet Σ along with a Σ-language L are fixed, hence we will omit the L qualifier from the notation, writing for instance $\trianglelefteq$ instead of $\trianglelefteq_L$.

3.3.1 A TL[EF]-Definable Language Is Typeset Dependent

In this section, we will show that the language L is typeset dependent using the assumption that it is defined by some TL[EF] formula ψ.

Definition 3.3.3 By $cl(\psi)$ we denote the smallest set of formulas that contains ψ and is closed under negations and subformulas.

It is not difficult to see that the type of a tree is determined by the set of those formulas from $cl(\psi)$ which it satisfies (although this correspondence need not be injective). Our first step is to show that for the delayed type, even less information is sufficient

Definition 3.3.4 An *existential formula* is a formula of the form $\mathsf{EF}\varphi$. The *signature* $\mathrm{Sig}(t)$ of a tree t is the set of existential formulas from $cl(\psi)$ that it satisfies.

Lemma 3.3.5 The signature of a tree determines its delayed type.

Proof
Take two trees s and t with the same signatures. For a given letter $a \in \Sigma$, an easy induction on formula size shows that for all $\varphi \in cl(\psi)$:

$$s\langle a\rangle \models \varphi \qquad \text{iff} \qquad t\langle a\rangle \models \varphi.$$

This is due to the fact that the modality EX is strict. Since the two trees $s\langle a\rangle$ and $t\langle a\rangle$ satisfy the same formulas from $cl(\psi)$, their types must be the

same. As the choice of the letter a was arbitrary, this implies that the trees s and t have the same delayed types. $\square$

Given two trees t_0, t_1 and a letter $a \in \Sigma$, we write $\mathrm{Sig}(t_0, t_1)$ instead of $\mathrm{Sig}(a[t_0, t_1])$. This notation is unambiguous since $\mathrm{Sig}(a[t_0, t_1])$ does not depend on the letter a.

Given two types α and β, we denote by $\mathit{dtype}(\alpha, \beta)$ the delayed type which assigns to a letter a the type $a[\alpha, \beta]$. A type α is *reachable* from a type β, denoted $\beta \preccurlyeq \alpha$, if $C[\beta] = \alpha$ holds for some context $C[]$. This relation is a quasiorder and we use $\approx$ for the accompanying equivalence relation. The following simple lemma is given without a proof:

Lemma 3.3.6 If t' is a subtree of t, then $\mathrm{Sig}(t', s) \subseteq \mathrm{Sig}(t, s)$. If $\alpha \preccurlyeq \beta$ then $\mathit{dtype}(\alpha, \beta) = \mathit{dtype}(\beta, \beta)$.

The following lemma shows that for TL[EF]-definable languages, the relation $\approx$ is a congruence with respect to the function $\mathit{dtype}(\alpha, \beta)$:

Lemma 3.3.7 If $\alpha_0 \approx \beta_0$ and $\alpha_1 \approx \beta_1$ then $\mathit{dtype}(\alpha_0, \alpha_1) = \mathit{dtype}(\beta_0, \beta_1)$.

Proof
Since a TL[EF]-definable language satisfies $\mathit{dtype}(\alpha, \beta) = \mathit{dtype}(\beta, \alpha)$, it is sufficient to prove the case where $\beta_1 = \alpha_1$. Let $C[]$ be a context such that $C[\alpha_0] = \beta_0$ and let $D[]$ be a context such that $D[\beta_0] = \alpha_0$. All these contexts exist by assumption. Let s_0 be a tree of type α_0 and let s_1 be a tree of type α_1. Consider the two sequences of trees $\{s_i\}_{i \geq 0}$ and $\{t_i\}_{i \geq 0}$ defined by induction as follows:

$$\begin{array}{rcll} s_0 & = & s_0; & \\ t_i & = & C[s_i] & \text{for } i \geq 0; \\ s_i & = & D[t_{i-1}] & \text{for } i \geq 1. \end{array}$$

By a simple induction one can prove that for all $i \geq 0$,

$$\mathit{type}(s_i) = \alpha_0 \quad \text{and} \quad \mathit{type}(t_i) = \beta_0 \ .$$

By Lemma 3.3.6, for all $i \geq 0$

$$\mathrm{Sig}(s_i, s_1) \subseteq \mathrm{Sig}(t_i, s_1) \subseteq \mathrm{Sig}(s_{i+1}, s_1) \ .$$

Since there are only finitely many signatures, there must be some $i > 0$ such that $\mathrm{Sig}(s_i, s_1) = \mathrm{Sig}(t_i, s_1)$. Consequently, by Lemma 3.3.5, the delayed types $\mathit{dtype}(\alpha_0, \alpha_1)$ and $\mathit{dtype}(\beta_0, \alpha_1)$ are equal. $\square$

We are now ready to show that the language L is typeset dependent. Let s and t be two trees with the same typeset. If this typeset is empty,

then both trees have one node and, consequently, the same delayed type. Otherwise once can consider the following four types, which describe the sons of s and t:

$$\alpha_0 = \mathit{type}(s|_0) \qquad \alpha_1 = \mathit{type}(s|_1) \qquad \beta_0 = \mathit{type}(t|_0) \qquad \beta_1 = \mathit{type}(t|_1).$$

We need to prove that $\mathit{dtype}(\beta_0, \beta_1) = \mathit{dtype}(\alpha_0, \alpha_1)$. By assumption that the typesets of s and t are equal, both β_0 and β_1 occur in nonroot nodes of s and both α_0 and α_1 occur in nonroot nodes of t. Thus $\beta_0 \preccurlyeq \alpha$ holds for some $\alpha \in \{\alpha_0, \alpha_1\}$ and similarly for β_1, α_0 and α_1. The result follows from the following case analysis:

- $\beta_0, \beta_1 \preccurlyeq \alpha$ for some $\alpha \in \{\alpha_0, \alpha_1\}$. By assumption we must have $\alpha \preccurlyeq \beta$ for some $\beta \in \{\beta_0, \beta_1\}$. Hence $\alpha \approx \beta$. By Lemma 3.3.7 we get $\mathit{dtype}(\alpha, \alpha) = \mathit{dtype}(\beta, \beta)$. As $\beta_0, \beta_1 \preccurlyeq \alpha \preccurlyeq \beta$, from Lemma 3.3.6 we obtain $\mathit{dtype}(\beta_0, \beta_1) = \mathit{dtype}(\beta, \beta)$. Similarly one proves the equality $\mathit{dtype}(\alpha_1, \alpha_2) = \mathit{dtype}(\alpha, \alpha)$.
- $\alpha_0, \alpha_1 \preccurlyeq \beta$ for some $\beta \in \{\beta_0, \beta_1\}$. As in the case above.
- A short analysis reveals that if neither of the above holds then $\beta_0 \preccurlyeq \alpha_i \preccurlyeq \beta_0$ and $\beta_1 \preccurlyeq \alpha_{1-i} \preccurlyeq \beta_1$ for some $i \in \{0, 1\}$. Therefore $\beta_0 \approx \alpha_i$ and $\beta_1 \approx \alpha_{1-i}$ and an application of Lemma 3.3.7 yields the desired result.

3.3.2 A Typeset Dependent Language Is EF-Admissible

This step of the proof consists of verifying that all the properties P1 to P4 are satisfied if the language is typeset dependent.

Lemma 3.3.8 L satisfies the property P1.

Proof
Condition P1 states that $\trianglelefteq$ is a partial order on delayed types. The relation $\trianglelefteq$ is obviously transitive and reflexive. We will show that $x \trianglelefteq y \trianglelefteq x$ implies that the delayed types x and y are equal. Assume then that $x \trianglelefteq y \trianglelefteq x$. In this case, for arbitrary n we can find a tree t with nodes $v_1 \leq w_1 \leq v_2 \leq w_2 \leq \cdots \leq w_n$ such that for all $i \leq n$:

$$\mathit{dtype}(t|_{v_i}) = x \quad \text{and} \quad \mathit{dtype}(t|_{w_i}) = y.$$

Naturally, for all $0 \leq i < n$,

$$\mathrm{TS}(t|_{v_i}) \subseteq \mathrm{TS}(t|_{w_i}) \subseteq \mathrm{TS}(t|_{v_{i+1}}).$$

If we take n to be bigger than the number of types in L then we can find some i such that $\mathrm{TS}(t|_{v_i}) = \mathrm{TS}(t|_{w_i})$, which implies $x = y$. □

Lemma 3.3.9 L satisfies the property P2.

Proof
Condition P2 states that if b, b' are neutral letters for a delayed type y, then the delayed types $dtype(x, a, y, b)$ and $dtype(x, a, y, b')$ are equal. To show that this condition is satisfied, we define by induction a sequence of trees $t_0, t_1, \ldots$ in the following manner. For t_0 we take some tree of delayed type y with b in the root, while t_{i+1} is defined as $b[b'[t_i, t_i], b'[t_i, t_i]]$. Because b and b' are neutral letters for the delayed type y, all the trees t_i have delayed type y. Moreover, for some $j > 0$, the typesets of the trees t_j and $b'[t_j, t_j]$ are equal.

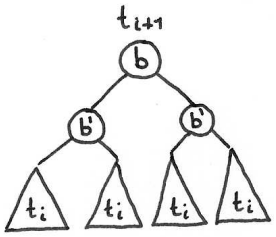


Figure 3.1: The tree t_i.

Consider a tree t of delayed type x and with the label a in the root. Let s and s' be trees such that:

$$\begin{array}{ll} s|_0 = t & s|_1 = t_j; \\ s'|_0 = t & s'|_1 = b'[t_j, t_j]. \end{array}$$

By assumption on t_j and $b'[t_j, t_j]$, the trees s and s' have the same typesets. Since L is typeset dependent, their delayed types are equal. Therefore,

$$dtype(x, a, y, b) = dtype(s) = dtype(s') = dtype(x, a, y, b').$$

□

The last two properties P3 and P4 are obviously satisfied in every typeset dependent language.

3.3.3 A EF-Admissible Language Is TL[EF]-Definable

We now proceed to the most difficult part of the proof, where a defining TL[EF] formula is found based only on the assumption that the properties P1 to P4 are satisfied. We start by stating a key property of EF-admissible languages which shows the importance of neutral letters.

Lemma 3.3.10 If the delayed type of a tree t is y, then its every proper subtree with delayed type y has the root label in N_y.

Proof
Consider some proper subtree $t|_v$ of delayed type y and its root label $b = t(v)$. Let w be the brother of the node v and let z, c be its delayed type and label, respectively. Obviously $(z, c) \trianglelefteq y$. By property P3 we get $\mathit{dtype}(y, b, z, c) = \mathit{dtype}(y, b, y, b)$ and consequently $\mathit{dtype}(y, b, y, b) \trianglelefteq y$. As $\trianglelefteq$ is a partial order by P1 and since $y \trianglelefteq \mathit{dtype}(y, b, y, b)$ holds by definition, we get $\mathit{dtype}(y, b, y, b) = y$. Hence b belongs to N_y. □

Note that if the trees t and $t|_v$ have delayed type y, then so does the tree $t|_w$ for any $w < v$, because $\trianglelefteq$ is a partial order. In particular, the above lemma says that nodes with delayed type y form cones whose non-root elements have labels in N_y.

Formulas Defining Delayed Types

A delayed type x is *definable* if there is some TL[EF] formula θ_x true in exactly the trees of delayed type x. A set A of delayed types is *downward closed* if it contains every delayed type $\trianglelefteq$-smaller than an element of A.

The construction of the θ_x formulas will proceed by induction on the $\trianglelefteq$ order. The first step is the following lemma:

Lemma 3.3.11 Let x be a delayed type and let $A \not\ni x$ be a downward closed set of definable delayed types. There is a TL[EF] formula fork_x^A such that:

$$t \vDash \mathit{fork}_x^A \quad \text{iff} \quad \mathit{dtype}(t) = x \text{ and for all } w > \varepsilon,\ \mathit{dtype}(t|_w) \in A.$$

We postpone the technical proof of this Lemma until Section 3.3.4. Meanwhile, we will use this lemma to construct a formula θ_x defining x. For the rest of Section 3.3.3 we fix the delayed type x and assume that every delayed type $y \triangleleft x$ is definable by a formula θ_y.

The first case is when x has no neutral letters. Let $x-$ denote the set $\{y : y \triangleleft x\}$. By Lemma 3.3.10, in a tree of delayed type x both sons have delayed types in $x-$, since there are no neutral letters for x. In this case we can set

$$\theta_x = \mathit{fork}_x^{x-} \ . \tag{3.1}$$

The correctness of this definition follows immediately from Lemma 3.3.11.

The definition of θ_x is more involved when the set of neutral letters for x is not empty. The rest of Section 3.3.3 is devoted to this case.

Consider first the following formula:

$$\theta_{\not\trianglelefteq} = \big(\mathsf{EF} \bigvee \{b \wedge \theta_y : y \trianglelefteq x \wedge (y,b) \not\trianglelefteq x\}\big) \vee \bigvee \{fork_y^{x-} : y \not\trianglelefteq x\}$$

The intention of this formula is to spell out evident cases when the delayed type of a node cannot be x. The first disjunct says that there is a descendant with a delayed type and a label that prohibit its ancestors to have type x. The second disjunct says that the type of the node is not x but the types of all descendants are $\trianglelefteq x$. This formula works correctly, however, only when some assumptions about the tree are made. These assumptions use the following definition: a tree t satisfies the property $\mathrm{OK}_x(t)$ if

$$dtype(t) \triangleleft x \qquad \text{or} \qquad dtype(t) = x \text{ and } t(\varepsilon) \in N_x \ .$$

Lemma 3.3.12 Let t be a tree where $\mathrm{OK}_x(t|_v)$ holds for all $v > \varepsilon$. This tree satisfies $\theta_{\not\trianglelefteq}$ if and only if $dtype(t) \not\trianglelefteq x$.

Proof
The left to right implication was already discussed and follows from the assumptions on the θ_y formulas used in $\theta_{\not\trianglelefteq}$ and from Lemma 3.3.11.

For the right to left implication, let $dtype(t) = dtype(y,b,z,c)$ with y, b, z, c describing delayed types and labels of the nodes 0 and 1 which correspond to the left and right sons of the root. We consider three cases:

- $y = z = x$. This is impossible because $\mathrm{OK}_x(t|_0)$ and $\mathrm{OK}_x(t|_1)$ hold, so the labels a, b must belong to N_x, and thus $dtype(t) = x$.

- $y = x$ and $z \triangleleft x$. Since $\mathrm{OK}_x(t|_0)$ holds, the label b belongs to N_x. If the inequality $(z,c) \trianglelefteq x$ were true (which is not necessarily implied by our assumption that $z \triangleleft x$), then by property P3 we would have

  $$dtype(t) = dtype(y,b,z,c) = dtype(x,b,z,c) = dtype(x,b,x,b) = x \ ,$$

  a contradiction with $dtype(t) \not\trianglelefteq x$. Therefore we have $(z,c) \not\trianglelefteq x$ and hence the first disjunct of $\theta_{\not\trianglelefteq}$ holds. The case where $z = x$ and $y \triangleleft x$ is symmetric.

- $y, z \triangleleft x$. In this case the second disjunct in the definition of $\theta_{\not\trianglelefteq}$ must hold by Lemma 3.3.11.

□

Let $\theta_{\triangleleft x}$ stand for $\bigvee_{y \triangleleft x} \theta_y$ and consider the formula

$$\varphi_x = \theta_{\triangleleft x} \vee (\neg\theta_{\not\trianglelefteq} \wedge \bigvee \{a : a \in N_x\}) \ .$$

This formula will be used to express the $\mathrm{OK}_x(t)$ property. We use AG^* as the non-strict version of AG, i.e. $\mathsf{AG}^*\varphi$ is an abbreviation for the formula $\varphi \wedge \mathsf{AG}\varphi$.

Lemma 3.3.13 A tree t satisfies $\mathsf{AG}^*\varphi_x$ iff $\mathrm{OK}_x(t|_v)$ holds for all $v \geq \varepsilon$.

Proof
By induction on the depth of the tree t.

$\Rightarrow$ If t satisfies φ_x because it satisfies $\theta_{\lhd x}$, then obviously $\mathrm{OK}_x(t|_v)$ holds for all $v \geq \varepsilon$. Otherwise we have

$$t(\varepsilon) \in N_x \qquad \text{and} \qquad t \not\models \theta_{\not x} \ .$$

By induction assumption, $\mathrm{OK}_x(t|_v)$ holds for all $v > \varepsilon$. But then, by Lemma 3.3.12, $\mathit{dtype}(t) \trianglelefteq x$. This, together with $t \not\models \theta_{\lhd x}$ gives $\mathit{dtype}(t) = x$ and hence $\mathrm{OK}_x(t)$.

$\Leftarrow$ Let t be such that $\mathrm{OK}_x(t|_v)$ holds for all $v \geq \varepsilon$. By induction assumption, we have $\mathsf{AG}\varphi_x$. We need to prove that t satisfies φ_x. If $\mathit{type}(t) \lhd x$ holds, then t satisfies $\theta_{\lhd x}$ and we are done. Otherwise, as $\mathrm{OK}_x(v)$ holds, $\mathit{dtype}(t) = x$ and $t(\varepsilon) \in N_x$. Hence, by Lemma 3.3.12, t satisfies the second disjunct in φ_x.

□

Since the type of a tree can be computed from its delayed type and root label, the following lemma ends the proof that every EF-admissible language is TL[EF]definable:

Lemma 3.3.14 Every delayed type is definable.

Proof
By induction on the depth of a delayed type x in the order $\trianglelefteq$. If x has no neutral letters then the defining formula θ_x is as in (3.1). Otherwise, we set the defining formula to be

$$\theta_x = \neg\theta_{\lhd x} \wedge \neg\theta_{\not x} \wedge \mathsf{AG}\varphi_x \ .$$

Let us show why θ_x has the required properties. By Lemma 3.3.13,

$$t \models \mathsf{AG}\varphi_x \qquad \text{iff} \qquad \mathrm{OK}_x(t|_w) \text{ for all } w > \varepsilon. \tag{3.2}$$

If $t \models \theta_x$ then we get $\mathit{dtype}(t) = x$ using Lemma 3.3.12 and (3.2). For the other direction, if $\mathit{dtype}(t) = x$ then clearly $\neg\theta_{\lhd x}$ holds in t. By Lemma 3.3.10, $\mathrm{OK}_x(t|_w)$ holds for all $w > \varepsilon$, therefore t satisfies $\mathsf{AG}\varphi_x$ by (3.2), and then the formula $\neg\theta_{\not x}$ holds by Lemma 3.3.12. □

3.3.4 A *fork* Formula

Recall Lemma 3.3.11 which was used in Section 3.3.3, but not proved there:

Lemma 3.3.11 Let x be a delayed type and let $A \not\ni x$ be a downward closed set of definable delayed types. There is a TL[EF] formula $fork_x^A$ such that:

$$t \vDash fork_x^A \quad \text{iff} \quad dtype(t) = x \text{ and for all } w > \varepsilon,\ dtype(t|_w) \in A.$$

The rest of Section 3.3.4 is devoted to a proof of this lemma. We fix a delayed type x and a downward closed set of delayed types A. We assume that $x \notin A$ and that all the delayed types in A are definable. The *fork* formula will be composed out of a vast number of auxiliary formulas, which we describe here:

$$\begin{aligned}
\theta_{\lhd y} &= \bigvee_{z \lhd y} \theta_z && \text{for } y \in A \cup \{x\};\\
\theta_{\trianglelefteq y} &= \bigvee_{z \trianglelefteq y} \theta_z && \text{for } y \in A;\\
\theta_y^b &= \mathsf{EF}(\theta_y \wedge b) \wedge \mathsf{AG}(\theta_y \Rightarrow (b \vee N_y)) && \text{for } y \in A, b \in \Sigma;\\
\theta^{\trianglelefteq y} &= \mathsf{AG} \bigvee \{\theta_z \wedge c : (z,c) \trianglelefteq y\} && \text{for } y \in A;\\
\varphi_y^b &= \theta_y^b \wedge \mathsf{AG}\theta_{\trianglelefteq y} \wedge \mathsf{AG}(\theta_{\lhd y} \Rightarrow \theta^{\trianglelefteq y}) && \text{for } y \in A, b \in \Sigma.
\end{aligned}$$

Observe that these formulas are well defined because we have assumed that all delayed types in A are definable, hence the appropriate θ_y formulas exist.

We now proceed to prove that the above formulas have certain desired properties. For a delayed type $y \in A$ and a letter $b \in \Sigma$, we say that the pair (y, b) is *sufficient* if $dtype(y, b, y, b)$ is our fixed delayed type x. Given a delayed type y, we define the following equivalence relation $\sim_y$ over Σ:

$$a \sim_y b \qquad \text{iff} \qquad a = b \quad \text{or} \quad a, b \in N_y\ .$$

Lemma 3.3.15 If (y, b) is sufficient, a tree satisfying φ_y^b has delayed type x.

Proof
Let t be a tree that satisfies φ_y^b. First we will show that some node $w \in \{0,1\}$ must have delayed type y and a label $\sim_y$-equivalent to b. Let Y be the set of nodes $w > \varepsilon$ with delayed type y; this set is not empty because t satisfies $\mathsf{EF}(\theta_y \wedge b)$. Since $\trianglelefteq$ is a partial order and $\mathsf{AG}\theta_{\trianglelefteq y}$ holds, every nonroot node between the root and a node in Y is also in Y. If the letter b belongs to N_y then, by $\mathsf{AG}(\theta_y \Rightarrow (b \vee N_y))$, we have a node in $\{0,1\}$ with a label in N_y,

hence $\sim_y$-equivalent to b. If $b \notin N_y$ then by $\mathsf{EF}(\theta_y \wedge b)$, there is a node in Y with label b. This must be one of 0 or 1 as Lemma 3.3.10 says that all nodes in $Y \setminus \{0,1\}$ must have labels in N_y.

Now we can prove that t has delayed type x. If both sons have delayed type y and labels $\sim_y$-equivalent to b then, by property P2, t is of delayed type x. If the brother of w has delayed type y but a label c that is not $\sim_y$-equivalent to b then c must belong to N_y, because $\mathsf{AG}(\theta_y \Rightarrow (b \vee N_y))$ holds. By definition of N_y we have $(y,c) \trianglelefteq y$. By property P3 we get $\mathit{dtype}(y,b,y,c) = \mathit{dtype}(y,b,y,b) = x$. The last case is when the brother of w is of delayed type z and has letter c such that $(z,c) \trianglelefteq y$ (because $\theta^{\trianglelefteq y}$ holds). By property P3 $\mathit{dtype}(y,b,z,c) = \mathit{dtype}(y,b,y,b) = x$. □

Given two delayed types $y, z \in A$ and letters $b, c \in \Sigma$, we define

$$\varphi_{(y,b,z,c)} = \begin{cases} \varphi_y^b & \text{if } (z,c) \trianglelefteq y; \\ \varphi_z^c & \text{if } (y,b) \trianglelefteq z \text{ and not the above;} \\ \theta_y^b \wedge \theta_z^c \wedge \mathsf{AG}(\theta_{\trianglelefteq y} \vee \theta_{\trianglelefteq z}) & \text{otherwise.} \end{cases}$$

Lemma 3.3.16 If $\mathit{dtype}(y,b,z,c) = x$ and a tree t satisfies $\varphi_{(y,b,z,c)}$, then its delayed type is x.

Proof
If $(z,c) \trianglelefteq y$ then by property P3, the pair (y,b) is sufficient and the lemma follows from Lemma 3.3.15. Similarly if $(y,b) \trianglelefteq z$. It remains to consider the case when

$$(z,c) \not\trianglelefteq y \qquad \text{and} \qquad (y,b) \not\trianglelefteq z\ . \tag{3.3}$$

Let $u_1 \neq \varepsilon$ be a node where $\theta_y \wedge b$ holds and $u_2 \neq \varepsilon$ a node where $\theta_z \wedge c$ holds. Such nodes exists since t satisfies both θ_y^b and θ_z^c. Let $w_1 \in \{0,1\}$ be the first letter of u_1 and let $w_2 \in \{0,1\}$ be the first letter of u_2. By (3.3), $t|_{w_1} \not\models \theta_{\trianglelefteq z}$, while $t|_{w_2} \not\models \theta_{\trianglelefteq y}$. Hence it must be the case that

$$t|_{w_1} \models \theta_{\trianglelefteq y} \quad \text{and} \quad t|_{w_2} \models \theta_{\trianglelefteq z}\ .$$

In particular, $w_1 \neq w_2$. By a reasoning similar to the one in Lemma 3.3.15, one shows that

$$\begin{aligned} \mathit{dtype}(t|_{w_1}) &= y \quad \text{and} \quad t(w_1) \sim_y b \\ \mathit{dtype}(t|_{w_2}) &= z \quad \text{and} \quad t(w_2) \sim_z c\ . \end{aligned}$$

By property P2, the delayed type of the tree t is x. □

Lemma 3.3.17 Let y, b be the delayed type and label of $t|_0$, similarly for z, c and $t|_1$. If $\mathit{dtype}(y,b,z,c) = x$ and $y, z \in A$ then $t \models \theta_{(y,b,z,c)}$.

Proof
If $(z,c) \trianglelefteq y$ then $z \trianglelefteq y$ and an easy analysis shows that t satisfies φ^b_y and hence also $\theta_{(y,b,z,c)}$. A similar reasoning shows that if $(y,b) \trianglelefteq z$ then t satisfies $\theta_{(y,b,z,c)}$. The last case is when $(z,c) \not\trianglelefteq y$ and $(y,b) \not\trianglelefteq z$. But then t satisfies the formula $\theta^b_y \wedge \theta^c_z \wedge \mathsf{AG}(\theta_{\trianglelefteq y} \vee \theta_{\trianglelefteq z})$. □

But Lemmas 3.3.16 and 3.3.17 are exactly what we need to show that the *fork* formula defined below satisfies the properties postulated in Lemma 3.3.11:

$$fork^A_x = (\mathsf{AG} \bigvee_{y \in A} \theta_y) \wedge \bigvee \{\varphi_{(y,b,z,c)} : x = dtype(y,b,z,c)\}$$

3.4 TL[EX, EF]

The last logic we consider in this chapter is TL[EX, EF]. As in the previous sections, we will present a characterization of TL[EX, EF]-definable languages. For the rest of the section we fix an alphabet Σ along with a Σ-language L and will henceforth omit the L qualifier from notation.

Recall the type reachability quasiorder $\preccurlyeq$ along with its accompanying equivalence relation $\approx$, which were defined on p. 55. The $\approx$-equivalence class of a type α is called here its *strongly connected component* and is denoted $\mathrm{SCC}_L(\alpha)$. We extend the relation $\preccurlyeq$ to SCCs by setting:

$$\begin{array}{lll} \Delta \preccurlyeq \Gamma & \text{if} & \alpha \preccurlyeq \beta \text{ for some } \alpha \in \Delta \text{ and } \beta \in \Gamma; \\ \alpha \preccurlyeq \Gamma & \text{if} & \alpha \preccurlyeq \beta \text{ for some } \beta \in \Gamma. \end{array}$$

We use the standard notational shortcuts, writing $\Delta \prec \Gamma$ when $\Delta \preccurlyeq \Gamma$ but not $\Gamma = \Delta$; similarly for $\alpha \prec \Gamma$.

Let Γ be some SCC and let $k \in \mathbb{N}$. The (Γ,k)-*view* of a tree t is the tree $view(\Gamma,k,t)$ whose domain is the set of nodes in t at depth at most k and where a node v is labeled by:

- $t(v)$ if v is at depth smaller than k;
- $type(t|_v)$ if v is at depth k and $type(t|_v) \prec \Gamma$;
- ? otherwise.

Let $views(\Gamma,k)$ denote the set of possible (Γ,k)-views. The intuition behind the (Γ,k)-view of t is that it gives exact information about the tree t for types which are $\preccurlyeq$ smaller than Γ, while for other types it just says "I don't know". The following definition describes languages where this information is sufficient to pinpoint the type within the strongly connected component Γ.

Definition 3.4.1 Let $k \in \mathbb{N}$. The language L is (Γ, k)-solvable if every two trees s and t with types in Γ and the same (Γ, k) view have the same type. The language is *k-solvable* if it is (Γ, k)-solvable for every SCC Γ and it is *SCC-solvable* if it is k-solvable for some k.

It turns out that SCC-solvability is exactly the property which characterizes the TL[EX, EF]-definable languages:

Theorem 3.4.2
A regular language is TL[EX, EF]*-definable if and only if it is SCC-solvable.*

The proof of this theorem will be presented in the two subsections that follow.

3.4.1 An SCC-solvable Language is TL[EX, EF]-Definable

In this section we show that one can write TL[EX, EF] formulas which compute views. Then, using these formulas and the assumption that L is SCC-solvable, the type of a tree can be found.

Fix some k such that L is k-solvable. Let $views(\alpha)$ be the set of possible (Γ, k)-views that can be assumed in a tree of type $\alpha \in \Gamma$. By assumption on L being k-solvable, we have:

Fact 3.4.3 Let t be a tree such that $type(t) \preccurlyeq \alpha$. The type of t is α if and only if its $(\mathrm{SCC}(\alpha), k)$-view belongs to the set $views(\alpha)$.

The following lemma states that views can be computed using the logic TL[EX, EF].

Lemma 3.4.4 Suppose that for every type $\beta \prec \Gamma$, there is a TL[EX, EF] formula θ_β defining it. Then for every $i \in \mathbb{N}$ and every $s \in views(\Gamma, i)$ there is a formula ψ_s satisfied in exactly the trees whose (Γ, i)-view is s.

Proof
By induction on i. □

We define below a set of views which certainly cannot appear in a tree with a type in a strongly connected component Γ:

$$\begin{aligned}\mathrm{Bad}(\Gamma) \;=\; & \{a[s,t] : s \in views(\alpha), t \in views(\beta), \text{where } \alpha, \beta \preccurlyeq \Gamma, a[\alpha,\beta] \not\preccurlyeq \Gamma\} \cup \\ & \cup\{t : type(t) \not\preccurlyeq \Gamma \text{ and } \mathrm{dom}(t) = \{\varepsilon\}\}\end{aligned}$$

Observe that $\mathrm{Bad}(\Gamma)$ is a set of $(\Gamma, k+1)$-views. The following lemma shows that the above cases are essentially the only ones.

Lemma 3.4.5 For a tree t and an SCC Γ, the following equivalence holds:

$$\mathit{type}(t) \not\preccurlyeq \Gamma \quad \text{iff} \quad \mathit{view}(\Gamma, k+1, t|_v) \in \mathrm{Bad}(\Gamma) \text{ for some } v \in \mathrm{dom}(t).$$

Proof
Both implications follow easily from Fact 3.4.3 if one considers the maximal possible node v satisfying the right hand side. □

The following lemma completes the proof that L is TL[EX, EF]-definable.

Lemma 3.4.6 Every type of L is TL[EX, EF]-definable.

Proof
The proof is by induction on depth of the type in the quasiorder $\preccurlyeq$. Consider a type α and its SCC Γ. By induction assumption, for all types $\beta \prec \Gamma$, there is a formula θ_β which is satisfied in exactly the trees of type β. Using the θ_β formulas and Lemma 3.4.4 we construct the following TL[EX, EF] formula (recall that AG^* is the non-strict version of AG defined on page 60):

$$\theta_\Gamma = \mathsf{AG}^* \bigwedge_{t \in \mathrm{Bad}(\Gamma)} \neg\psi_t.$$

By Lemma 3.4.5, a tree t satisfies θ_Γ if and only if $\mathit{type}(t) \preccurlyeq \Gamma$. Finally, the formula θ_α is defined:

$$\theta_\alpha = \theta_\Gamma \wedge \bigvee_{t \in \mathit{views}(\alpha)} \psi_t.$$

The correctness of this construction follows from Fact 3.4.3. □

3.4.2 A TL[EX, EF]-Definable Language is SCC-Solvable

In this section, we are going to show that a language which is not SCC-solvable is not TL[EX, EF]-definable. For this, we introduce an appropriate Ehrenfeucht-Fraïsé game, called the *EX+EF game*, which characterizes trees indistinguishable by TL[EX, EF]-formulas.

The game is played over two trees and by two players, Spoiler and Duplicator. The intuition is that in the *k-round EX+EF game*, the player Spoiler tries to differentiate the two trees using k moves.

The precise definition is as follows. At the beginning of the k-round game, with $k \geq 0$, the players are faced with two trees t_0 and t_1. If these have different root labels, Spoiler wins. If they have the same root labels and $k = 0$, Duplicator wins; otherwise the game continues. Spoiler first picks one

of the trees t_i, with $i \in \{0,1\}$. Then he chooses whether to make an EF or EX move. If he chooses to make EF move, he needs to choose some non-root node $v \in \mathrm{dom}(t_i)$ and Duplicator must respond with a non-root node $w \in \mathrm{dom}(t_{1-i})$ of the other tree. If Spoiler chooses to make an EX move, he picks a son $v \in \{0,1\}$ of the root in t_i and Duplicator needs to pick the same son $w = v$ in the other tree. If a player cannot find an appropriate node in the relevant tree, this player immediately looses. Otherwise the trees $t_i|_v$ and $t_{1-i}|_w$ become the new position and the $(k-1)$-round game is played.

The *operator nesting depth* of a formula is defined by induction in the natural fashion. Formulas that correspond to letters have depth zero, the depth of a boolean combination is the maximal depth of the formulas involved, while applying EX or EF to a formula increases the depth by one.

Lemma 3.4.7 Duplicator wins the k-round EX+EF game over t_0 and t_1 iff t_0 and t_1 satisfy the same EX+EF formulas of operator nesting depth k.

Proof
A standard proof by induction on k. The case of $k = 0$ is obvious. Let us assume that we have proved the statement for some k and consider $k+1$.

Consider first the left to right implication. We show that if a formula φ distinguishes the trees t_0 and t_1, then a winning strategy for Spoiler can be found. If φ distinguishes the trees t_0 and t_1, then one of its subformulas of the form $\mathsf{EX}\psi$ or $\mathsf{EF}\psi$ distinguishes them too. Let us consider the case of $\mathsf{EF}\psi$ and assume without loss of generality that $\mathsf{EF}\psi$ holds only in t_0. This means that there is a nonroot node v_0 in the tree t_0 such that

$$t_0|_{v_0} \models \psi \qquad \text{and} \qquad t_1|_{v_1} \not\models \psi \text{ for all nonroot nodes } v_1 \text{ of } t_1$$

The winning strategy for Spoiler is, of course, to pick an EF move, the tree t_0 and the vertex v_0. Since ψ is of operator nesting depth k, no matter what vertex v_1 Duplicator picks, Spoiler has – by induction assumption – a winning strategy in the k-round game over the trees $t_0|_{v_0}$ and $t_1|_{v_1}$. A similar argument is used when the distinguishing formula is of the form $\mathsf{EX}\psi$.

For the right to left implication, we show how to write a distinguishing formula ψ of nesting depth $k+1$ based on the assumption that Spoiler wins the $k+1$-round game. Consider a winning strategy of Spoiler in this game. We assume without loss of generality that Spoiler chooses the tree t_0 to make his move. Two cases need be considered. The first is when Spoiler chooses an EF move and a subtree $t_0|_{v_0}$. Since his strategy is winning, for every possible choice of a node v_1 in the tree t_1, the k-round game over the trees $t_0|_{v_0}$ and $t_1|_{v_1}$ can be won by Spoiler. By induction assumption this means that for

every node v_1 in the tree t_1, there is a formula ψ_{v_1} of nesting depth k such that

$$t_0|_{v_0} \models \psi_{v_1} \qquad \text{and} \qquad t_1|_{v_1} \not\models \psi_{v_1} \ .$$

Note that in order to have ψ_{v_1} satisfied in $t_0|_{v_0}$ and not in $t_1|_{v_1}$, we may have negated the formula from the induction assumption. Let ψ be a conjunction of all the ψ_{v_1} formulas for all choices of v_1. The appropriate formula that distinguishes the trees t_0 and t_1 is then $\mathsf{EF}\psi$. A similar reasoning is used for EX. □

For two types $\alpha, \beta \in \Gamma$ we define an (α, β)-*context* to be a multicontext C such that there are two valuations of its holes $\nu_\alpha, \nu_\beta : V \to \Gamma$ giving the types $C[\nu_\alpha] = \alpha$ and $C[\nu_\beta] = \beta$. The *hole depth* of a multicontext C is the minimal depth of a hole in C. A multicontext C is *k-bad for an SCC Γ* if it has hole depth at least k and is an (α, β)-context for two different types $\alpha, \beta \in \Gamma$.

Lemma 3.4.8 L is not SCC-solvable if and only if for some SCC Γ and every $k \in \mathbb{N}$, it contains multicontexts which are k-bad for Γ.

Proof
A k-bad context exists for Γ if and only if L is not (Γ, k)-solvable. □

The following lemma concludes the proof that no TL[EX, EF] formula can recognize a language which is not SCC-solvable:

Lemma 3.4.9 If L is not SCC-solvable then for every k there are trees $s \in L$ and $t \notin L$ such that Duplicator wins the k-round EX+EF game over s and t.

Proof
Take some $k \in \mathbb{N}$. If L is not SCC-solvable then, by Lemma 3.4.8, there is a multicontext C which is k-bad for some SCC Γ. Let $V = \{v_1, \ldots, v_n\}$ be the holes of C, let $\nu_\alpha, \nu_\beta : V \to \Gamma$ be the appropriate valuations and $\alpha = C[\nu_\alpha], \beta = C[\nu_\beta]$ the resulting types. We will use this multicontext to find trees $s \in L$ and $t \notin L$ such that Duplicator wins the k-round EX+EF game over s and t.

Since all the types used in the valuations ν_α and ν_β come from same SCC, there are contexts $C_1^\alpha[], \ldots, C_n^\alpha[]$ and $C_1^\beta[], \ldots, C_n^\beta[]$ such that

$$C_i^\alpha[\alpha] = \nu_\beta(v_i) \qquad C_i^\beta[\beta] = \nu_\alpha(v_i) \qquad \text{for all } i \in \{1, \ldots, n\}.$$

This means there are two contexts D^α and D^β with n holes each, such that: 1) D^α and D^β agree over nodes of depth less than k; 2) when all holes of D^α are plugged with β, we get the type α; and 3) when all holes of D^β

are plugged with α, we get the type β. These are obtained by plugging the appropriate "translators" $C_i^\alpha[]$ and $C_i^\beta[]$ into the holes of the multicontext C. Let t_0 be some tree of type α. The trees t_j for $j > 0$ are defined by induction as follows:

$$t_{2i+1} = D^\beta[\overbrace{t_{2i}, \ldots, t_{2i}}^{n \text{ times}}] \qquad t_{2i+2} = D^\alpha[\overbrace{t_{2i+1}, \ldots, t_{2i+1}}^{n \text{ times}}].$$

By an obvious induction, all the trees t_{2i} have type α and all the trees t_{2i+1} have type β. As $\beta \neq \alpha$, there exists a context $D[]$ such that $D[\alpha] \in L$ and $D[\beta] \notin L$ (or the other way round).

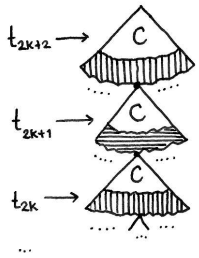


Figure 3.2: The tree t_{2i+2}.

To finish the proof of the lemma, we will show that Duplicator wins the k-round EX+EF game over the trees

$$s = D[t_{2k+2}] \qquad \text{and} \qquad t = D[t_{2k+1}] \ .$$

The winning strategy for Duplicator is obtained by following an invariant. This invariant is a disjunction of three properties, one of which always holds when the i-round game is about to be played:

1. The two trees are identical;
2. The two trees are $s|_v$ and $t|_v$ for some $|v| \leq k - i$;
3. The two trees are $t_m|_v$ and $t_{m-2}|_v$ for

$$m \geq k + i + 1 \quad \text{and} \quad \begin{cases} v \in \mathrm{dom}(D^\alpha) & \text{if } m \text{ is even;} \\ v \in \mathrm{dom}(D^\beta) & \text{if } m \text{ is odd.} \end{cases}$$

The invariant holds at the beginning of the first round, due to 2, and one can verify that Duplicator can play in such a way that it is satisfied in all rounds. Item 2 of the invariant will be preserved in the initial fragment of the game when only EX moves are made, then item 3 will hold until either the game ends or item 1 begins to hold. □

3.5 Decidability

In this section we round up the results by showing that our characterizations are decidable.

Theorem 3.5.1
It is decidable in time polynomial in the number of types if a language is:

- TL[EX]-*definable;*
- TL[EF]-*definable;*
- TL[EX, EF]-*definable.*

Proof
Using a simple dynamic algorithm, one can compute in polynomial time all tuples $(\alpha, \beta, \alpha', \beta')$ such that for some context $C[]$, $C[\alpha] = \alpha'$ and $C[\beta] = \beta'$. Using this, we can find in polynomial time:

- Whether L contains an $\{\alpha, \beta\}$-loop;
- The $\preccurlyeq_L$ and $\approx_L$ relations on types.

Since the delayed type of a tree depends only on the types of its immediate subtrees, the number of delayed types is polynomial in the number of types. The relation $\trianglelefteq_L$ on delayed types can then be computed in polynomial time from the relation $\preccurlyeq_L$. Having the relations $\preccurlyeq_L$ and $\trianglelefteq_L$, one can check in polynomial time if L is EF-admissible.

This, along with the characterizations from Theorems 3.2.3 and 3.3.2, proves decidability for TL[EX] and TL[EF]. The remaining logic is TL[EX, EF].

By Theorem 3.4.2, it is enough to show that SCC-solvability is decidable. In order to do this, we give an algorithm that detects if a given SCC Γ admits bad multicontexts of arbitrary size, cf. Lemma 3.4.8. Fix an SCC Γ. We define by induction a sequence B^i of subsets of $\Gamma \times \Gamma$.

- $B^0 = \Gamma \times \Gamma$.
- $(\alpha, \beta) \in B^{i+1}$ if $(\alpha, \beta) \in B^i$ and either
  - there is a pair $(\alpha', \beta') \in B^i$, a type $\gamma \prec \Gamma$ and a letter $a \in \Sigma$ such that $\mathit{type}(a[\alpha', \gamma]) = \alpha$ and $\mathit{type}(a[\beta', \gamma]) = \beta$; or
  - there are pairs $(\alpha', \beta'), (\alpha'', \beta'') \in B^i$ and a letter $a \in \Sigma$ such that $\mathit{type}(a[\alpha', \alpha'']) = \alpha$ and $\mathit{type}(a[\beta', \beta'']) = \beta$

The sequence B^i is decreasing so it reaches a fix-point B^∞ in no more than $|\Gamma|^2$ steps. The following lemma yields the algorithm for TL[EX, EF]:

Lemma 3.5.2 Γ admits bad multicontexts of arbitrary size iff $B^\infty \neq \emptyset$.

For the left-to-right implication suppose that B^∞ is not empty. By induction on k we show that for every k and $(\alpha, \beta) \in B^\infty$, we can construct (α, β)-context of hole depth k. Take $(\alpha, \beta) \in B^\infty$. We have one of the two cases from the definition above. The first is when there are a pair $(\alpha', \beta') \in B^\infty$, a type $\gamma' \prec \Gamma$ and a letter $a \in \Sigma$ such that $\mathit{type}(a[\alpha', \gamma']) = \alpha$ and $\mathit{type}(a[\beta', \gamma']) = \beta$. By induction assumption we have an (α', β')-context C' of hole depth $k-1$. Using this multicontext, we construct the multicontext $a[C', s]$, where s is a tree of type γ. It is a required (α, β)-context of hole depth k. The other case is similar.

For the right-to-left implication we show that if $(\alpha, \beta) \in B^i - B^{i+1}$ then all (α, β)-contexts have hole depth bounded by i. This is also done by induction on i.

□

Corollary 3.5.3 If the input is a CTL formula or a nondeterministic tree automaton, all of the problems in Theorem 3.5.1 are EXPTIME-complete.

Proof
Since, in both cases, the types can be computed in time at most exponential in the input size, the EXPTIME membership follows immediately from Theorem 3.5.1. For the lower bound, we will use an argument analogous to the one in [69], reducing the EXPTIME-hard universality problems for both CTL [23] and nondeterministic automata [54] to any of these problems.

We will only show here the EXPTIME-hardness of the problem:

Is a given CTL formula equivalent to one in TL[EF]? (*)

Let ψ be a CTL formula over some alphabet Σ. By [23], the question whether ψ is satisfied in all Σ-trees is EXPTIME-hard. We show the EXPTIME-hardness of the problem (*) by presenting a formula φ which is definable in the logic TL[EF] if and only if the formula ψ is true in all Σ-trees. This formula is obtained by using ψ and some fixed formula ϕ (say, $\mathsf{E}(a\mathsf{U}b)$) not definable in TL[EF]:

$$\varphi = \mathsf{EX}(S_0 \wedge \psi) \vee \mathsf{EX}(S_1 \wedge \phi)$$

First we show that if ψ is true in all Σ-trees, then φ is definable in the logic TL[EF]. But this is simple: if ψ is true in all Σ-trees, then, by its first

disjunct, φ is true in all Σ-trees with more than one node. This language is defined by the TL[EF] formula $\mathsf{EX}\top$.

Finally, we need to prove that if ψ is not true in all Σ-trees, then φ is not definable in TL[EF]. Let us assume for the sake of contradiction that φ is equivalent to some TL[EF] formula θ. Let Ψ be the set of all subformulas of θ. By assumption that the formula ϕ is not definable in TL[EF], there exist two trees t_1 and t_2 that satisfy the same formulas in Ψ, but one satisfies ϕ and the other does not (otherwise an appropriate boolean combination of formulas in Ψ would be equivalent to ϕ). Therefore exchanging t_1 with t_2 in any subtree does not affect the satisfaction of θ. Let s be a tree that does not satisfy ψ, obtained by the assumption on ψ not being satisfied in all Σ-trees. One can easily verify that for any letter $a \in \Sigma$,

$$a[s, t_1] \models \varphi \qquad a[s, t_2] \not\models \varphi \ ,$$

but either both these trees satisfy θ or both do not. □

3.6 Why Forbidden Patterns Do not Work

In the survey [70], one can find decidable characterizations for several fragments of LTL. These fragments can be seen as the word equivalents of the logics TL[EX], TL[EF] and TL[EX, EF] considered here. One naturally asks the question: how are the word and tree cases related? In the case of the logic TL[EX], the loop characterization from Theorem 3.2.3 is an exact analogue of the characterization corresponding to the fragment TL[X] of LTL.

For the two remaining logics, however, the word and tree cases diverge. This section is devoted to showing why.

Case of TL[EF]**.** First we need to introduce the appropriate definitions for words. The delayed type of a word w is the function which assigns to a letter a the type of the word $a \cdot w$. Two word types α, β are in the same SCC if there are word types α', β' such that $\alpha' \cdot \alpha = \beta$ and $\beta' \cdot \beta = \alpha$. In [70] it is shown that a word language is definable using only the modality F if and only if the delayed type of a word is determined by the SCC of its type.

Hence a natural question: is a tree language TL[EF]-definable if and only if the delayed type of a tree is determined by the SCCs of its two sons? This can understood in two ways: the ordered pair of SCCs, or the set of SCCs. The first idea can be immediately disproved, for instance using the language "there is an a in the left subtree". The idea that uses sets requires a more elaborate example, which is presented here.

	a	b
$x_\top$	$\top$	$\top$
x_a	α	γ
x_b	γ	β
x_γ	γ	γ

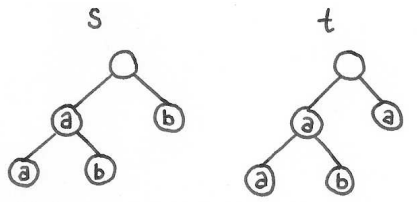


Figure 3.3: The delayed types of L along with the trees s and t

Consider the $\{a, b\}$-language L defined by the formula (which uses the non-strict versions of AG and EF):

$$\psi = \mathsf{EF}^*[\mathsf{EX}(\mathsf{AG}^* a) \wedge \mathsf{EX}((\mathsf{EF}^* a) \wedge (\mathsf{EF}^* b))]$$

A tree t satisfies this formula if it contains two nodes v and w which are siblings and $t|_v$ contains only a's, while $t|_w$ contains both a's and b's. This language has four types α, β, γ and $\top$, which are defined by the formulas:

$$\alpha = \mathsf{AG}^* a; \quad \beta = \mathsf{AG} b^*; \quad \gamma = (\mathsf{EF}^* a) \wedge (\mathsf{EF}^* b) \wedge \neg\psi; \quad \top = \psi.$$

Each of these types is its own strongly connected component. There are also four delayed types $\{x_\top, x_a, x_b, x_\gamma\}$, which are defined in Figure 3.3.

Consider now the two trees s and t drawn in Figure 3.3 (we do not specify the root letters since we are interested only in delayed types). These trees show that the language L is not typeset dependent and therefore not in TL[EF], since:

$$\mathrm{TS}(s) = \mathrm{TS}(t) = \{\alpha, \beta, \gamma\} \quad \text{but} \quad x_\gamma = \mathit{dtype}(s) \neq \mathit{dtype}(t) = x_\top.$$

However, if the SCCs of both sons are known, then the types of both sons are known and hence so is the delayed type of the tree.

Case of TL[EX, EF]**.** In [70] it is shown that a word language is definable using the modalities F and X if and only if one cannot find two distinct types α, β in the same SCC and a nonempty word w such that:

$$\alpha = w \cdot \alpha \quad \text{and} \quad \beta = w \cdot \beta. \tag{3.4}$$

We will show that a straightforward generalization of this condition obtained by considering contexts instead of words does not work in the tree case. Consider the language K over the alphabet $\{a, b, c\}$ defined by the formula

$$\psi = \mathsf{AG}^* \bigvee_{\sigma \neq \tau \in \{a,b,c\}} \psi_{\sigma,\tau} \qquad \text{where} \quad \psi_{\sigma,\tau} = [\mathsf{EX}\top \Rightarrow (\sigma \wedge \mathsf{A}(\sigma \mathsf{U} \tau))] \ .$$

This language consists of those trees where for every node v, all the minimal nodes in the set $\{w : w > v \text{ and } t(w) \neq t(v)\}$ have the same label. The ten types of the language are:

$$\bot = \neg\psi; \qquad \alpha_\sigma = \sigma \wedge \mathsf{AX}\bot; \qquad \beta_{\sigma,\tau} = \psi \wedge \psi_{\sigma,\tau} \qquad \text{for all } \sigma \neq \tau \in \{a, b, c\}.$$

There are five SCCs in this language: an SCC Γ containing the six $\beta_{\sigma,\tau}$ types, while each of the remaining types is its own SCC.

The language K is not k-solvable for any $k \in \mathbb{N}$, and hence is not definable in TL[EX, EF]. We will show, however, that if in the definition (3.4) one considers nontrivial contexts instead of nonempty words w, the resulting condition on tree languages is satisfied by K. This goes to show that in a bad multicontext one sometimes requires the use of more than one hole.

Let $C[]$ be a nontrivial context, i.e. one with the hole not in the root. We will show that one cannot find two distinct types in the SCC Γ such that

$$C[\gamma_1] = \gamma_1 \qquad \text{and} \qquad C[\gamma_2] = \gamma_2. \tag{3.5}$$

Let σ be the letter in the parent v of the hole, and let w be the brother of the hole. Let $V = \{u : u \geq w \text{ and } C(u) \neq \sigma\}$. If $V = \emptyset$, or nodes in V have two different labels, then $C[\gamma] = \bot$ for all types γ. Otherwise, let τ be the unique label of all nodes in V. This means that for any tree t, the type in v of $C[t]$ is either $\bot$ or $\beta_{\sigma,\tau}$, which proves that (3.5) cannot be satisfied.

3.7 Open Problems

The question of definability for the logics TL[EX], TL[EF] and TL[EX, EF] has been pretty much closed in this chapter. One possible continuation are logics where instead of EF we use the non-strict modality EF^*. The resulting logics are weaker than their strict counterparts (for instance the language $\mathsf{EF}a$ is not definable in TL[EF^*]) and therefore decidability of the their definability problems can be investigated.

Another question is what happens if we enrich these logics with past quantification (there exists a point in the past)? This question is particularly relevant in the case of TL[EX, EF], since the resulting logic coincides with first-order logic with two variables (where the signature contains $\leq$ and two *binary* successor relations).

Finally, there is the logic CTL. This logic is similar to the ones considered in this chapter in that it too is *cascade bounded*, i.e. for some n, every language definable in CTL is recognized by a cascade product of n-state automata. In the case of CTL, $n \leq 4$. Providing a decidable characterization of CTL would

be valuable achievement, since this is a widely used logic. Note that on words CTL collapses to LTL and hence first-order logic, so such a characterization would to subsume first-order definability for words.

Chapter 4

A Bounding Quantifier

In this chapter and the next one, we change the setting: instead of finite trees, we consider infinite ones and instead of definability, we consider satisfiability. The only thing in common with the first chapters is that we will still be considering monadic second-order logic and regular languages.

When the tree is infinite, many sorts of cardinality constraints can be expressed using monadic second-order logic. Some constraints, however, fall outside monadic second-order logic, such as: "there exist bigger and bigger sets such that..." or "there is a bound on the size of sets such that...". In this chapter we present decision procedures for an extension of MSOL where such cardinality-bounding properties are definable.

The need for cardinality constraints occurs naturally in applications. For instance, a graph that is interpreted in the full binary tree using monadic second-order logic is known to have bounded tree-width if and only if it does not contain bigger and bigger complete bipartite subgraphs [4]. Another example: a formula of the two-way μ-calculus [67] is satisfied in some finite model if and only if it is satisfied in some tree model in which there is a bound on the size of certain sets [5]; this problem is the subject of the next chapter. Sometimes boundedness is an object of interest in itself, cf. [8], where pushdown games with the bounded stack condition are considered.

In light of these examples, it seems worthwhile to consider the logic MSOL+$\mathbb{B}$ obtained from MSOL by adding a new quantifier $\mathbb{B}$, which expresses properties like the ones just mentioned. Let $\psi(X)$ be a formula defining some property of a set X in a labeled infinite tree. The formula $\mathbb{B}X.\psi(X)$ is satisfied in those trees t where there is a finite bound – which might depend on t – on the size of sets satisfying this property. We also consider the dual quantifier $\mathbb{U}$, which states that there is no finite bound.

Adding new constructions to MSOL has a long history. A notable early example is a paper of Elgot and Rabin [15], where the authors investigated

what predicates can be added to the structure $\langle \mathbb{N}, \leq \rangle$ while preserving decidability of its monadic second-order theory. Among the positive examples they gave are monadic predicates representing the sets $\{i! : i \in \mathbb{N}\}$, $\{i^k : i \in \mathbb{N}\}$ and $\{k^i : i \in \mathbb{N}\}$. This line of research was recently continued by Carton and Thomas in [10], where the list was extended by so called *morphic* predicates which include, for instance, the set of Fibonacci numbers.

A construction similar to our bounding quantifier can be found in [35], where Klaedtke and Ruess consider extending MSOL on trees and words with cardinality constraints of the form:

$$|X_1| + \cdots + |X_m| < |Y_1| + \cdots + |Y_n|.$$

Although MSOL with these cardinality constraints is in general undecidable, the authors show a decision procedure for a fragment of the logic, where, among other restrictions, quantification is allowed only over finite sets. Interestingly, the bounding quantifier $\mathbb{B}$ is definable using cardinality constraints, although it does fall outside the aforementioned fragment and cannot be described using the techniques of Klaedtke and Ruess:

$$\mathbb{B}X.\psi(X) \qquad \text{iff} \qquad \exists Y.\text{Finite}(Y) \wedge \forall X.(\psi(X) \Rightarrow |X| < |Y|) \ .$$

Finally, a quantifier $\exists^{\mathfrak{c}}$ that also deals with cardinality can be formulated based on the results of Niwiński in [44]. It is not the size of sets satisfying $\psi(X)$, but the number of such sets that is quantified in this case, however. More precisely, a binary tree t satisfies $\exists^{\mathfrak{c}}X.\psi(X)$ if there are continuum sets satisfying $\psi(X)$. This quantifier, it turns out, is definable in MSOL, and thus its unrestricted use retains decidability.

Our bounding quantifier $\mathbb{B}$, however, is *not* definable in MSOL. Using the bounding quantifier, one can define nonregular languages and hence the question: is satisfiability of MSOL+$\mathbb{B}$ formulas decidable? In this chapter we investigate this question and, while being unable to provide an exhaustive answer, we present decision procedures for two nontrivial fragments of MSOL+$\mathbb{B}$.

This investigation leads us to identify a class of tree languages, new to our knowledge, which we call quasiregular tree languages. A set of infinite trees is *L-quasiregular* if it coincides with the regular language L over the set of regular trees and, moreover, is the sum of some family of regular tree languages. The intuition behind an L-quasiregular language is that it is a slight non-regular variation over the language L, yet in most situations it behaves the same way.

On the one hand, quasiregular languages are simple enough to have decidable emptiness: an L-quasiregular language is nonempty if and only if L is

nonempty. On the other hand, quasiregular languages are powerful enough to allow nontrivial applications of the bounding quantifier: they are closed under bounding quantification, existential quantification, conjunction and disjunction. This yields the decidability result in Theorem 4.2.12:

> The satisfiability problem for *existential bounding formulas*, i. e. ones built from arbitrary MSOL formulas by application of $\mathbb{B}$, $\exists$, $\wedge$ and $\vee$ is decidable.

Unfortunately quasiregular languages do not capture all of MSOL+$\mathbb{B}$. For instance, they are not closed under complementation, hence Theorem 4.2.12 gives little insight into properties that use the dual quantifier $\mathbb{U}$.

For this reason, we also conduct a separate analysis of the $\mathbb{U}$ quantifier (note that satisfiability for $\mathbb{U}$ is related to *validity* for $\mathbb{B}$). By inspection of an underlying automaton, we prove Theorem 4.2.20:

> Satisfiability is decidable for formulas of the form $\mathbb{U}X.\psi$, where ψ is MSOL.

We are, however, unable to extend this result in a fashion similar to Theorem 4.2.12, by allowing for non-trivial nesting.

The plan of this chapter is as follows. After some preliminaries in Section 4.1, we introduce the quantifier in Section 4.2. In Sections 4.2.1 and 4.2.2 we prove decidability for bounding existential formulas, while in Section 4.2.3 we prove decidability for formulas which use the unbounding quantifier next to an MSOL formula.

4.1 Preliminaries

In this section we define the basic notions used in the chapter: infinite trees, regular languages of infinite trees and regular trees. The definitions are straightforward extensions of the ones for finite trees formulated in Chapter 2.

Let Σ be some finite set, called the *alphabet*. An *infinite Σ-tree* is a function $t : \{0,1\}^* \to \Sigma$. Therefore, all infinite trees have the same domain. We denote the set of infinite Σ-trees by $\mathrm{Trees}^\infty(\Sigma)$. An *infinite tree language over* Σ is any subset of $\mathrm{Trees}^\infty(\Sigma)$. Since we will only consider infinite trees in this chapter and the next one, we will omit the word infinite and simply write Σ-tree and tree language.

A node is any element of $\{0,1\}^*$. The definitions of a subtree $t|_v$ and the composition $t\hat{}s$ of two trees are the same as for finite trees. A *regular tree* is a tree with finitely many distinct subtrees; the class of all regular trees is

denoted by REG. An *infinite path* is any infinite sequence of nodes $\boldsymbol{\pi}$ such that:

$$\boldsymbol{\pi}_0 = \varepsilon \quad \boldsymbol{\pi}_1 = \boldsymbol{\pi}_0 \cdot a_0 \quad \boldsymbol{\pi}_2 = \boldsymbol{\pi}_1 \cdot a_1 \quad \cdots \qquad a_i \in \{0,1\} \ .$$

Given two nodes $v < w$, we define the set $\mathrm{Bet}(v,w)$ of elements between v and w as $v \cdot \{0,1\}^* \setminus w \cdot \{0,1\}^*$.

Let Σ be an alphabet and $*$ a letter outside Σ. A Σ-*context* is any $\Sigma \cup \{*\}$-tree C where the label $*$ occurs only once, in a position called the *hole* of C. We don't require this position to be a leaf, since there are no leaves in an infinite tree, but all nodes below the hole are going to be irrelevant to the context. The *domain* $\mathrm{dom}(C)$ of a context C is the set of nodes that are not below or equal to the hole. Given a Σ-tree t and a context $C[]$ whose hole is v, the tree $C[t]$ is defined by:

$$C[t](w) = \begin{cases} t(u) & \text{if } w = v \cdot u \text{ for some } u \in \{0,1\}^*; \\ C(w) & \text{otherwise.} \end{cases}$$

The composition of two contexts C and D is the unique context $C \cdot D$ such that $(C \cdot D)[t] = C[D[t]]$ holds for all trees t. We do not use multicontexts for infinite trees.

4.1.1 Nondeterministic Tree Automata and Regular Tree Languages

As in the case of finite trees, regular languages of infinite trees can be defined both using automata and monadic second-order logic. The two approaches are briefly described in this section.

Definition 4.1.1 [Parity condition] A sequence $\boldsymbol{a} \in A^{\mathbb{N}}$ of numbers belonging to some finite set of natural numbers A is said to satisfy the *parity condition* if the smallest number occurring infinitely often in $\boldsymbol{a}$ is even.

A *nondeterministic tree automaton with the parity condition* is a tuple

$$\mathcal{A} = \langle Q, \Sigma, q_I, \delta, \Omega \rangle$$

where Q is a finite set of *states*, Σ is the finite *input alphabet*, $q_I \in Q$ is the *initial state*, $\delta \subseteq Q \times \Sigma \times Q \times Q$ is the *transition relation* and $\Omega : Q \to \mathbb{N}$ is the *ranking function*. Elements of the finite image $\Omega(Q)$ are called *ranks*. A *run* of $\mathcal{A}$ over a Σ-tree t is any Q-tree ρ such that

$$\langle \rho(v), t(v), \rho(v \cdot 0), \rho(v \cdot 1) \rangle \in \delta \qquad \text{for every } v \in \{0,1\}^*.$$

The run ρ is *accepting* if for every infinite path $\boldsymbol{\pi}$, the sequence of ranks $\Omega \circ \rho \circ \boldsymbol{\pi}$ satisfies the parity condition. The automaton *accepts a tree t from state $q \in Q$* if there is some accepting run with state q labeling the root. A tree is *accepted* if it is accepted from the initial state q_I. The *language of $\mathcal{A}$*, denoted $L(\mathcal{A})$, is the set of trees accepted by $\mathcal{A}$; such a language is said to be *regular*. An automaton is *nonempty* if and only if its language is.

We say two trees s and t are *equivalent for an automaton $\mathcal{A}$*, which is denoted $s \simeq_{\mathcal{A}} t$, if for every state q of $\mathcal{A}$, the tree s is accepted from q if and only if the tree t is. If the trees s and t are equivalent for $\mathcal{A}$, then they cannot be distinguished by a context, i.e. for every context $C[]$, the tree $C[s]$ is accepted by $\mathcal{A}$ if and only if the tree $C[t]$ is.

We now proceed to define the logical approach to regular languages of infinite trees. Consider an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$. As in the finite tree case, with a Σ-tree, we associate a relational structure

$$\underline{t} = \langle \{0,1\}^*, S_0, S_1, \leq, \underline{\sigma}_1^t, \ldots, \underline{\sigma}_n^t \rangle.$$

The relations are interpreted as follows: S_0 is the set of left sons $\{0,1\}^*0$, S_1 is the set of right sons $\{0,1\}^*1$, $\leq$ is the prefix ordering, while $\underline{\sigma}_i^t$ is the set of nodes that are labeled by the letter σ_i.

With a sentence ψ of monadic second-order logic we associate the language $L(\psi)$ of trees t such that $\underline{t}$ satisfies ψ. Such a language is said to be *MSOL-definable*. A famous result of Rabin [50] says that a language of infinite trees is MSOL-definable if and only if it is regular.

4.2 The Bounding Quantifier

The logic MSOL+$\mathbb{B}$ is obtained from MSOL by adding two quantifiers: the *bounding quantifier* $\mathbb{B}$, and its dual *unbounding quantifier* $\mathbb{U}$, which we define here using infinitary disjunction and conjunction:

$$\mathbb{B}^i X.\varphi := \forall X.(\varphi(X) \Rightarrow |X| < i) \qquad \mathbb{U}^i X.\varphi := \exists X.(\varphi(X) \wedge |X| \geq i)$$

$$\mathbb{B}X.\ \varphi := \bigvee_{i \in \mathbb{N}} \mathbb{B}^i X.\varphi \qquad \mathbb{U}X.\ \varphi := \bigwedge_{i \in \mathbb{N}} \mathbb{U}^i X.\varphi$$

MSOL+$\mathbb{B}$ defines strictly more languages than MSOL (see Fact 4.2.13), hence it is interesting to consider decidability of the following problem:

Is a given formula of MSOL+$\mathbb{B}$ satisfiable in some infinite tree?

The remainder of this chapter is devoted to this question. Although unable to provide a decision procedure for the whole logic, we do identify two decidable

fragments. The first, existential bounding formulas, is proved decidable in Sections 4.2.1 and 4.2.2, while the second, formulas of the form $\mathbb{U}X.\psi$ with ψ in MSOL, is proved decidable in Section 4.2.3. Before concluding the chapter, we overview in Section 4.3 some of the possible applications of these results.

4.2.1 Quasiregular Tree Languages

Before we proceed with the proof of Theorem 4.2.12, we define the concept of a quasiregular tree language. We then demonstrate some simple closure properties of quasiregular tree languages and, in Section 4.2.2, show that quasiregular tree languages are closed under bounding quantification. These closure properties, along with the decidable nonemptiness of quasiregular languages, yield the decision procedure for existential bounding formulas found in Theorem 4.2.12.

For technical reasons, we will find it henceforth convenient to work on trees where the alphabet is the powerset $P(\Sigma)$ of some set Σ. The same results would hold for arbitrary alphabets, but the notation would be more cumbersome. By $\mathrm{Val}(\Sigma)$ we denote the set of $P(\Sigma)$-trees. Elements of the set Σ will be treated as set variables, the intuition being that a tree in $\mathrm{Val}(\Sigma)$ represents a valuation of the variables in Σ. Given a tree $t \in \mathrm{Val}(\Sigma)$ and a set $F \subseteq \{0,1\}^*$, the tree

$$t[X := F] \in \mathrm{Val}(\Sigma \cup \{X\})$$

is defined by adding the element X to the labels of all nodes in F and removing it, if necessary, from all the other nodes.

Bounding quantification for an arbitrary tree language $L \subseteq \mathrm{Val}(\Sigma)$ is defined as follows. A tree $t \in \mathrm{Val}(\Sigma \setminus \{X\})$ belongs to the language $\mathbb{B}X.L$ if there is some finite bound on the size of sets F such that the tree $t[X := F]$ belongs to L.

We now give the key definition of a quasiregular tree language.

Definition 4.2.1 (Quasiregular Language) Let L be a regular tree language. A tree language K is *L-quasiregular* if

- $K \cap \mathrm{REG} = L \cap \mathrm{REG}$, and
- K is the union of some family of regular tree languages

A tree language is *quasiregular* if it is L-quasiregular for some regular language L. For the rest of Section 4.2.1 we will use the letter L for regular languages and the letter K for quasiregular ones.

Lemma 4.2.2 If K is L-quasiregular, then $K \subseteq L$.

Proof
Let $\{L_i\}_{i \in I}$ be the family of regular tree languages whose union is K. We will show that each language L_i is a subset of L. Indeed, over regular trees L_i is a subset of L, since K and L agree over regular trees. This implies the inclusion $L_i \subseteq L$ for arbitrary trees, since otherwise the regular language $L_i \setminus L$ would be nonempty and therefore, by Rabin's Basis Theorem [51], contain a regular tree. □

The following easy fact shows that emptiness is decidable for quasiregular tree languages given an appropriate presentation:

Fact 4.2.3 If K is L-quasiregular, then K is nonempty iff L is nonempty.

Proof
If L is nonempty, then it contains by Rabin's Basis Theorem a regular tree and hence K must contain this same tree. The other implication follows from Lemma 4.2.2. □

In particular, every nonempty quasiregular language contains a regular tree. For a variable X we define the *projection function* Π_X which given a tree returns the tree with X removed from all the labels. Projection is the tree language operation corresponding to existential quantification, as testified by the following equation:

$$L(\exists X.\psi) = \Pi_X(L(\psi)).$$

A set $F \subseteq \{0,1\}^*$ is *regular* if the unique tree $t[X := F] \in \mathrm{Val}(\{X\})$ is regular. Equivalently, F is regular if it is a regular word language. The following is a standard result:

Lemma 4.2.4 If a regular tree t belongs to the projection $\Pi_X(L)$ of a regular language L, then $t[X := F]$ belongs to L for some regular set F.

Proof
Since t is a regular tree, the set $\{t\}$ is a regular tree language and so is $\Pi_X^{-1}(\{t\})$. Therefore the intersection $L \cap \Pi_X^{-1}(\{t\})$ is regular and nonempty and, by Rabin's Basis Theorem, contains some regular tree. Obviously, the X component in this tree must be a regular set. □

Now we are ready to show some basic closure properties of quasiregular languages:

Lemma 4.2.5 Quasiregular languages are closed under projection, intersection and union.

Proof
The cases of intersection and union are trivial; we will only do the proof for projection. Let K be L-quasiregular. We will show that the projection $\Pi_X(K)$ is $\Pi_X(L)$-quasiregular. First we prove that $\Pi_X(K)$ is the union of a family of regular languages. By assumption, K is the union some family of regular tree languages $\{L_i\}_{i\in I}$. But then

$$\Pi_X(K) = \Pi_X(\bigcup_{i\in I} L_i) = \bigcup_{i\in I} \Pi_X(L_i)$$

and, since regular tree languages are closed under projection, $\Pi_X(K)$ is the union of some family of regular tree languages.

We also need to show that for every regular tree t,

$$t \in \Pi_X(K) \qquad \text{iff} \qquad t \in \Pi_X(L).$$

The left to right implication follows from Lemma 4.2.2. The right to left implication follows from the fact that if $t \in \Pi_X(L)$ then, by Lemma 4.2.4, for some regular set F, $t[X = F] \in L$. Since the tree $t[X = F]$ is regular, it also belongs to K and hence t belongs to $\Pi_X(K)$.

□

4.2.2 Closure Under Bounding Quantification

In this section, we show that quasiregular tree languages are closed under application of the bounding quantifier. This, together with the closure properties described in Lemma 4.2.5, yields a decision procedure for the fragment of MSOL+$\mathbb{B}$ that nests $\mathbb{B}$ along with existential quantification, conjunction and disjunction.

Recall that a *chain* is any set of nodes that is linearly ordered by $\leq$. We say that a chain C is a *trace path* of a set of nodes F if the set $F \cap \mathrm{Bet}(v, u)$ is nonempty for all nodes $v < w$ in C.

Lemma 4.2.6 A set of at least 3^n nodes has a trace path of size n. An infinite set has an infinite trace path.

Proof
We prove the finite case by induction, strengthening the statement by requiring the least element of the trace path to be the greatest lower bound of the set of nodes F in question. The base case of $n = 0$ is trivial. Assume therefore, that the statement has been proved for n and consider a set F of cardinality at least 3^{n+1}. Let v be the greatest lower bound of F. There

must be at least 3^n elements of F below some successor w of v, therefore an n-element trace path $C \subseteq w \cdot \{0,1\}^*$ of F can be found. The $n+1$-element chain $C \cup \{v\}$ must also be a trace path of F, since the set $\mathrm{Bet}(v,w) \cap F$ is nonempty by assumption on v.

The infinite case follows by an infinite iteration of the above process. $\Box$

Let $t \in \mathrm{Val}(\Sigma)$ and $L \subseteq \mathrm{Val}(\Sigma \cup \{X\})$. An (L,X)-*bad chain* in the tree t is an infinite chain C whose every finite subset is a trace path of some set F such that $t[X := F] \in L$. The set of trees containing no (L,X)-bad chain is denoted by $\mathbb{C}X.L$. Bad chains have the desirable property of being MSOL definable, as testified by:

Lemma 4.2.7 If L is regular then $\mathbb{C}X.L$ is regular.

Proof
Since L is regular, then $L = L(\varphi)$ for some MSOL formula φ. Using φ and some standard coding, we will describe an MSOL formula expressing $\mathbb{C}X.L$. First we define a formula $\mathrm{Chain}(X)$ which expresses that X is a chain:

$$\mathrm{Chain}(X) \;=\; \forall x \forall y.\ (x \in X \wedge y \in X) \Rightarrow (x \leq y \vee x \geq y)\ .$$

Before we express infinity for arbitrary sets, we first provide an MSOL formula describing infinite chains:

$$\begin{array}{rcl} \mathrm{InfChain}(X) & = & \mathrm{Chain}(X) \wedge (\exists x.\ x \in X) \wedge \\ & & \wedge (\forall x.\ x \in X \Rightarrow \exists y.\ (y \in X \wedge y > x))\ . \end{array}$$

By Lemma 4.2.6, a finite set is one that does not contain an infinite trace path, which can be expressed in MSOL as follows:

$$\begin{array}{rcl} \mathrm{TrPath}(Y,X) & = & \mathrm{Chain}(Y) \wedge \forall x \forall y.\ [x \in Y \wedge y \in Y \wedge (x < y)] \Rightarrow \\ & & \Rightarrow \exists z.\ (z \in X \wedge x \leq z \wedge y \not\leq z)\ ; \\ \mathrm{Finite}(X) & = & \neg \exists Y.\ \mathrm{InfChain}(Y) \wedge \mathrm{TrPath}(Y,X)\ . \end{array}$$

Finally, the language $\mathbb{C}X.L$ is defined by the formula:

$$\neg \exists Y.\ \mathrm{InfChain}(Y) \wedge \forall Z.\ (Z \subseteq Y \wedge \mathrm{Finite}(Z)) \Rightarrow \exists X.\ \varphi(X) \wedge \mathrm{TrPath}(Z,X)$$

$\Box$

Lemma 4.2.8 If K is L-quasiregular then $\mathrm{REG} \cap \mathbb{C}X.K = \mathrm{REG} \cap \mathbb{C}X.L$.

Proof
This follows from the fact that for a finite (and therefore regular) node set F and a regular tree t, the tree $t[X := F]$ is regular, and hence belongs to L if and only if it belongs to K. □

Lemma 4.2.9 Let L be a regular tree language. A regular tree that belongs to $\mathbb{C}X.L$ also belongs to $\mathbb{B}X.L$.

Proof
Consider a regular tree t with m distinct subtrees. Let $\mathcal{A}$ being some automaton recognizing L with k being the index of the relation $\simeq_{\mathcal{A}}$. Setting n to be $3^{k \cdot m+1}$, we will show that if t does not belong to the language $\mathbb{B}^n X.L$, then an (L, X)-bad chain must exist.

Consider indeed a set F of at least n nodes such that the tree $t[X := F]$ belongs to L. By Lemma 4.2.6, this set has a trace path with more than $k \cdot m$ nodes. Let $v < w$ be two nodes on this trace path such that

$$t[X := F]|_v \simeq_{\mathcal{A}} t[X := F]|_w \qquad \text{and} \qquad t|_v = t|_w \ .$$

Such two nodes exist by virtue of the trace path's size. Moreover, since v and w are on the trace path, the intersection $F \cap \mathrm{Bet}(v, w)$ is nonempty. Let $u \in \{0, 1\}^*$ be such that $w = v \cdot u$.

We claim that the chain $\{v \cdot u^i : i \in \mathbb{N}\}$ is a bad chain. For this, we will show that for every $i \in \mathbb{N}$, the subchain $C_i = \{v \cdot u^j : j \leq i\}$ can be expanded to a set F_i satisfying $t[X := F_i] \in L$.

This is done by pumping i times the part of the set F between v and w. Consider the following partition of F:

$$F_1 = \{u' : u' < v\} \cap F \qquad F_2 = \mathrm{Bet}(v, w) \cap F \qquad F_3 = \{u' : u' > w\} \cap F$$

One can easily check that the following set F_i contains the subchain C_i:

$$F_i \quad = \quad F_1 \quad \cup \quad \bigcup_{j \in \{0,\ldots,i\}} v \cdot u^j \cdot v^{-1} \cdot F_2 \quad \cup \quad v \cdot u^i \cdot v^{-1} \cdot F_3.$$

Moreover, since for all $j \in \{0, \ldots, i\}$, the equivalence

$$t[X := F_j]_{v \cdot u^j} \simeq_{\mathcal{A}} t[X := F]|_w$$

holds, the tree $t[X := F_i]$ belongs to L. □

Using the Lemma 4.2.9 above, we can show that quasiregular tree languages are closed under application of the bounding quantifier.

Lemma 4.2.10 If K is quasiregular, then so is $\mathbb{B}X.K$.

Proof
If K is quasiregular, then it is a union $\bigcup_{i\in I} L_i$ of some family of regular tree languages. Therefore $\mathbb{B}X.\ K$ is also a union regular tree languages:

$$\mathbb{B}X.K = \bigcup_{i\in I}\bigcup_{j>0} \mathbb{B}^j X.\ L_i.$$

Let L be such that K is L-quasiregular. We will show:

$$\mathbb{C}X.L \cap \mathrm{REG} = \mathbb{B}X.K \cap \mathrm{REG}.$$

The right to left inclusion follows from Lemma 4.2.8 and the simple inclusion $\mathbb{B}X.K \subseteq \mathbb{C}X.K$. The left to right inclusion follows from Lemma 4.2.9. □ The following lemma provides an effective assesment of size of sets in a formula with the bounding quantifier.

Lemma 4.2.11 Let K be L-quasiregular, L being recognized by some nondeterministic automaton $\mathcal{A}$. Then $\mathbb{B}X.K$ is nonempty if and only if $\mathbb{B}^n X.K$ is nonempty, where n is doubly exponential in the size of $\mathcal{A}$.

Proof
By Lemma 4.2.10, the language $\mathbb{B}X.K$ is $\mathbb{C}X.L$-quasiregular. Since $\mathbb{C}X.L$ is recognized by an automaton whose size is polynomial in the size of $\mathcal{A}$, it contains, by Rabin's Basis Theorem, a regular tree t whose regularity is also polynomial in the size of $\mathcal{A}$. By the proof of Lemma 4.2.10, the tree t belongs to $\mathbb{B}^n X.K$, where n is the constant from Lemma 4.2.9, which is exponential in the regularity of t and the (exponential) index of the relation $\simeq_{\mathcal{A}}$. □

Putting together the closure properties of quasiregular tree languages proved in this and the previous section, we obtain:

Theorem 4.2.12
The satisfiability problem for existential bounding formulas*, i.e. ones built from arbitrary MSOL formulas by application of* $\mathbb{B}$, $\exists$, $\wedge$ *and* $\vee$ *is decidable.*

Proof
By Lemmas 4.2.5 and 4.2.10, the language $L(\psi)$ of an existential bounding formula is L-quasiregular for some effectively obtained regular tree language L. By Fact 4.2.3, the emptiness of $L(\psi)$ is equivalent to the emptiness of L. □

Unfortunately, we cannot hope to extend the quasiregular tree language approach to decide all possible nestings of the bounding quantifier, as certified by the following Fact:

Fact 4.2.13 Even for regular L, $\neg\mathbb{B}X.L$ is not necessarily quasiregular.

Proof
The language L in question is obtained from a formula ψ with free variables X and Y. This formula states that Y contains no infinite subchains and that X is a subchain of Y.

In a regular tree $t \in \mathrm{Val}(\{X, Y\})$ with n distinct subtrees, a subchain of Y can be of size at most n – otherwise Y has an infinite subchain and ψ does not hold. Therefore $\neg\mathbb{B}X.L(\psi)$ is a nonempty language without a regular tree and cannot be quasiregular. □

4.2.3 The Unbounding Quantifier

In this section we present a procedure which, given a regular language $L \subseteq \mathrm{Val}(\Sigma)$ and a variable $X \in \Sigma$, decides whether the language $\mathbb{U}X.L$ is nonempty. This implies that satisfiability is decidable for formulas of the form $\mathbb{U}X.\psi(X)$, where ψ is in MSOL. Unfortunately, we are unable to extend this decision procedure to accommodate nesting, the way we did in Theorem 4.2.12. On the other hand though, the procedure runs in polynomial time.

In order to help the reader's intuition a bit, we will begin our analysis by debunking a natural, yet false, idea: for every regular language L there is some $n \in \mathbb{N}$ such that the language $\mathbb{U}X.L$ is nonempty if and only if the language $\mathbb{U}^n X.L$ is.

The intuition behind this idea would be that a pumping process should inflate arbitrarily the set F once it has reached some threshold size. The problem, however, is that a tree may contain labels which are not part of the set F, and the pumping might violate this labeling. A suitable counterexample is the following language $L \subseteq \mathrm{Val}(\{X, Y\})$:

> X is a subset of Y and Y is a finite set.

Obviously the language $\mathbb{U}X.L$ is empty, yet for every $n \in \mathbb{N}$, the language $\mathbb{U}^n X.L$ is nonempty. We will have to bear such issues in mind in the proofs below, taking care that we pump only the part of the labeling corresponding to X.

Let us fix a set Σ, a regular language $L \subseteq \mathrm{Val}(\Sigma)$ and a variable $X \in \Sigma$ for the rest of this section. We will use $\hat{\Sigma}$ to denote the set $\Sigma \setminus \{X\}$. Analogously to the "language" definition of $\mathbb{B}X.L$ in Section 4.2.1, we say a tree $t \in \mathrm{Val}(\hat{\Sigma})$ belongs to $\mathbb{U}X.L$ if there is *no* finite bound on the size of sets F such that $t[X := F] \in L$.

An infinite sequence of nodes $\boldsymbol{v}$ is *increasing* if $\boldsymbol{v}_i < \boldsymbol{v}_{i+1}$ holds for all $i \in \mathbb{N}$. A family of node sets $\mathcal{F}$ is *traced* by an increasing sequence $\boldsymbol{v}$ of

nodes if for all $i \in \mathbb{N}$,

$$|F \cap \mathrm{Bet}(\boldsymbol{v}_i, \boldsymbol{v}_{i+1})| \geq i \qquad \text{for some } F \in \mathcal{F} \ .$$

Lemma 4.2.14 A family that contains sets of unbounded size is traced.

Proof
Consider a family $\mathcal{F}$ with sets of unbounded size. Given a node v, let $\mathcal{F}|_v$ be the family $\{F \cap v \cdot \{0,1\}^* : F \in \mathcal{F}\}$. Let G be the set of nodes v such that $\mathcal{F}|_v$ contains sets of unbounded size. The set G is infinite and downward closed, hence it contains a chain H. We are going to pick an increasing sequence of nodes $\boldsymbol{v}$ from this chain H in such a way that for all $i \geq 1$, the set $\mathrm{Bet}(\boldsymbol{v}_{i-1}, \boldsymbol{v}_i)$ contains at least i nodes of some set $F \in \mathcal{F}$.

We choose $\boldsymbol{v}_0 = \varepsilon$. Let us assume that $\boldsymbol{v}_0, \ldots, \boldsymbol{v}_n$ have been defined and that the above conditions are satisfied for $i \leq n$. Let $F \in \mathcal{F}$ be a set that contains at least n nodes below v_n. Such a set exists by the assumption $\boldsymbol{v}_n \in H$. The claim immediately follows from the fact that there must be some node $\boldsymbol{v}_{n+1} \in H$ such that at least n elements of F fall within $\mathrm{Bet}(\boldsymbol{v}_n, \boldsymbol{v}_{n+1})$. $\square$

We fix now some nondeterministic parity automaton recognizing L:

$$\mathcal{A} = \langle Q, \Sigma, A, q_I, \delta, \Omega \rangle.$$

Without loss of generality, we assume that every state $q \in Q$ is used in some accepting run. The rest of this section is devoted to an analysis this automaton and to establishing a structural property equivalent to the nonemptiness of the language $\mathbb{U}X.L$.

A *descriptor* is any element of $Q \times \Omega(Q) \times Q$. With a $P(\Sigma)$-context C we associate the set *Trans*(C) consisting of those descriptors (q, m, r) such that there is a run of $\mathcal{A}$ that starts in the root of C in state q and:

- The (finite) path of the run that ends in the hole of C uses states of rank at least m and ends in the state r.
- All the (infinite) paths of the run that do not go through the hole of C satisfy the parity condition.

The intuition is that *Trans*(C) describes the possible runs of $\mathcal{A}$ which go through the context C. The compositions of two descriptors and then of two sets of descriptors are defined below (descriptors which do not agree on the state p do not compose):

$$\begin{array}{rcll} (q,n,p) \cdot (p,m,r) & = & (q, \min(n,m), r) & \text{(descriptor)} \\ X \cdot Y & = & \{x \cdot y : x \in X, y \in Y\} & \text{(set of descriptors).} \end{array}$$

The descriptor set of the context composition $C \cdot D$ can be computed from the composition of the descriptor sets of the contexts C and D:

$$\mathit{Trans}(C \cdot D) = \mathit{Trans}(C) \cdot \mathit{Trans}(D). \tag{4.1}$$

We will also be using descriptors of $P(\hat{\Sigma})$-contexts. For a $P(\hat{\Sigma})$-context C and $k \in \mathbb{N}$, we define $\mathit{Trans}_k(C)$ to be the set

$$\bigcup_{F:|F|\geq k} \mathit{Trans}(C[X := F]).$$

A *schema* is a pair $R = (R^\bullet, R^\circ)$ of descriptor sets. A schema is meant to describe a $P(\hat{\Sigma})$-context, the intuition being that the R° descriptors can be obtained from any sets F, while the $R^\bullet$ descriptors are obtained from "large" sets. A $P(\hat{\Sigma})$-context C is said to *k-realize* a schema R if $R^\circ \subseteq \mathit{Trans}_0(C)$ and $R^\bullet \subseteq \mathit{Trans}_k(C)$. The composition $R \cdot S$ of two schemas $R = (R^\bullet, R^\circ)$ and $S = (S^\bullet, S^\circ)$ is defined to be the schema

$$R \cdot S = (R^\bullet \cdot S^\circ \cup R^\circ \cdot S^\bullet, R^\circ \cdot S^\circ).$$

The following obvious fact describes how composition of schemas corresponds to composition of $P(\hat{\Sigma})$-contexts.

Fact 4.2.15 Let R and S be schemas which are respectively k-realized by $P(\hat{\Sigma})$-contexts C and D. The schema $R \cdot S$ is k-realized by the context $C \cdot D$.

We now proceed to define the notion of an infinitary sequence. The intuition here is that an infinitary sequence exhibits the existence of a tree belonging to $\mathbb{U}X.L$, which is obtained by composing all the contexts in $\boldsymbol{C}$:

Definition 4.2.16 A sequence of schemas $\boldsymbol{R}$ is *infinitary* if both

- There is a sequence of $P(\hat{\Sigma})$-contexts $\boldsymbol{C}$ such that for every $n \in \mathbb{N}$ the schema $\boldsymbol{R}_n$ is n-realized by the context $\boldsymbol{C}_n$; and
- For some fixed state $r \in Q$ and all $n \in \mathbb{N}$, there is a state sequence $\boldsymbol{q}$ with $\boldsymbol{q}_0 = r$ such that:
  1. For $i < n$, $(\boldsymbol{q}_i, m, \boldsymbol{q}_{i+1}) \in \boldsymbol{R}_i^\circ$ for some rank m;
  2. For $i = n$, $(\boldsymbol{q}_i, m, \boldsymbol{q}_{i+1}) \in \boldsymbol{R}_i^\bullet$ for some rank m;
  3. For $i > n$, $(\boldsymbol{q}_i, m, \boldsymbol{q}_{i+1}) \in \boldsymbol{R}_i^\circ$ for some even rank m.

Lemma 4.2.17 $\mathbb{U}X.L$ is nonempty iff there exists an infinitary sequence.

Proof
Consider first the right to left implication. Let $\boldsymbol{R}$ be the infinitary sequence with $\boldsymbol{C}$ and $r \in Q$ being the appropriate sequence of contexts and starting state from Definition 4.2.16. Let $t \in \mathrm{Val}(\hat{\Sigma})$ be the infinite composition of all successive contexts in $\boldsymbol{C}$:

$$t = \boldsymbol{C}_0 \cdot \boldsymbol{C}_1 \cdot \boldsymbol{C}_2 \cdot \cdots$$

Let D be a context such that $(q_I, n, r) \in \mathit{Trans}(D)$ for some $n \in \Omega(Q)$. This context exists by our assumption on $\mathcal{A}$ not having useless states. Using the properties of the sequence $\boldsymbol{R}$ postulated in Definition 4.2.16, one can easily verify that the tree $D[t]$ belongs to $\mathbb{U}X.L$.

For the left to right implication, consider a tree t in $\mathbb{U}X.L$. From this tree we will extract an infinitary sequence. Consider the family of node sets

$$\{F \subseteq \{0,1\}^* : t[X := F] \in L\}.$$

By assumption on t, this family contains sets of unbounded size. Therefore, by Lemma 4.2.14, it is traced by some increasing sequence $\boldsymbol{v}$. Consider the sequence $\boldsymbol{C}$ of contexts, where $\boldsymbol{C}_i$ is obtained from the tree $t|_{\boldsymbol{v}_i}$ by placing the hole in the node corresponding to $\boldsymbol{v}_{i+1}$. One can verify that the sequence of schemas

$$\boldsymbol{R}_i = (\mathit{Trans}(\boldsymbol{C}_i), \mathit{Trans}_i(\boldsymbol{C}_i)),$$

along with $r = q_I$, is infinitary. □

Although infinitary sequences characterize the unboundedness of L, they are a little hard to work with. That is why we use a special type of infinitary sequence, which nonetheless remains equivalent to the general case (cf. Lemma 4.2.19). Consider a very simple schema R which consists of two loops in R° and a connecting descriptor in $R^\bullet$:

$$R = (R^\circ, R^\bullet) \qquad \text{where} \quad R^\circ = \{(q,k,q),(p,m,p)\} \quad \text{and} \quad R^\bullet = \{(q,n,p)\}.$$

We say the pair of states (q,p) used above is *inflatable* if the sequence $\boldsymbol{R}$ constantly equal R is infinitary, for some choice of ranks k, m and n. Note that in this case, the rank m must be even.

Lemma 4.2.18 The inflatable pairs can be computed in polynomial time.

Proof
For $i \in \mathbb{N}$, consider the set A_i of triples

$$\langle (q_1,n_1,p_1),(q_2,n_2,p_2),(q_3,n_3,p_3)\rangle \in (Q \times \Omega(Q) \times Q)^3$$

such that for some $P(\hat{\Sigma})$-context C:

- $(q_1, n_1, p_1), (q_3, n_3, p_3) \in \mathit{Trans}_0(C)$;
- $(q_2, n_2, p_2) \in \mathit{Trans}_i(C)$.

Using a dynamic algorithm, the set A_i can be computed in time polynomial on i and the size of the state space Q. By a pumping argument, one can show that the pair (q, p) is inflatable if and only if

$$\langle (q, n_1, q), (q, n_2, p), (p, n_3, p) \rangle \in A_{|Q|+1} \qquad \text{for some even } n_3.$$

□

We now proceed to show Lemma 4.2.19, which shows that one can consider inflatable pairs instead of arbitrary infinitary sequences.

Lemma 4.2.19 There is an infinitary sequence iff there is an inflatable pair.

Proof
An inflatable pair is by definition obtained from an infinitary sequence, hence the right to left implication. For the other implication, consider an infinitary sequence $\boldsymbol{R}$ along with the appropriate sequence of contexts $\boldsymbol{C}$. With every two indices $i < j$, we associate the schema $R[i, j]$ obtained by composing the schemas $\boldsymbol{R}_i \cdot \cdots \cdot \boldsymbol{R}_{j-1}$. Since there is a finite number of schemas, by Ramsey's Theorem [52] there is a schema R and a set of indices $I = \{i_1 < i_2 < \cdots\} \subseteq \mathbb{N}$ such that $R[i, j] = R$ for every $i < j$ in I. Naturally, in this case $R \cdot R = R$ and, by Fact 4.2.15, the sequence constantly equal R is an infinitary sequence which realizes the sequence of contexts $\boldsymbol{D}$ defined

$$\boldsymbol{D}_j = \boldsymbol{C}_{i_j} \cdot \cdots \cdot \boldsymbol{C}_{i_{j+1}}.$$

We will now show how to extract an inflatable pair from this sequence. Let $q \to p$ be the relation holding for those states $q, p \in Q$ such that (q, m, p) belongs to R° for some rank m. Since $R \cdot R = R$, the relation $\to$ is transitive. Let $(q, m, p) \in R^\bullet$ be a descriptor used for infinitely many n in clause 2 of Definition 4.2.16. We claim:

- $r \to q'$, $q' \to q'$ and $q' \to q$, for some $q' \in Q$. This follows from transitivity of $\to$ if we take n in Definition 4.2.16 to be big enough to find a loop.
- $p \to p'$ and $(p', m, p') \in R^\circ$ for some $p' \in Q$ and even rank m. This is done as above.

Consider finally the sequence of contexts $\boldsymbol{E}$ defined

$$\boldsymbol{E}_i = \boldsymbol{D}_i \cdot \boldsymbol{D}_{i+1} \cdot \boldsymbol{D}_{i+2} \ .$$

By Fact 4.2.15, for all $i \in \mathbb{N}$ the schema $R \cdot R \cdot R = R$ is i-realized by the context $\boldsymbol{E}_i$. One can easily verify that the sequence $\boldsymbol{E}$ witnesses the fact that (q', p') is an inflatable pair. □

From Lemmas 4.2.17, 4.2.18 and 4.2.19 we immediately obtain:

Theorem 4.2.20
Satisfiability is decidable for formulas of the form $\mathbb{U}X.\psi$, where ψ is MSOL.

Note that the appropriate algorithm is in fact polynomial in the size of a parity automaton recognizing ψ.

4.3 Applications

Before we conclude this chapter, we briefly and informally overview three possible applications. We would like to emphasize that in none of these cases does using the bounding quantifier give *new* results, it only simplifies proofs of existing ones.

The first application comes from graph theory. Sometimes a graph $G = (V, E)$ can be interpreted in the unlabeled full binary tree $\{0,1\}^*$ via a pair of monadic second-order formulas: a formula $\alpha(x)$ true for the nodes used to represent a vertex from V and a formula $\beta(x, y)$ representing the edge relation E. In [4], Barthelmann showed that such a graph $G(\alpha, \beta)$ is of bounded tree-width if and only if there is a fixed bound on the size n of full bipartite subgraphs $K_{n,n}$ of $G(\alpha, \beta)$. Given two sets $F, G \subseteq \{0,1\}^*$ one can express using MSOL that they represent the left and right parts of a bipartite subgraph. The property that there exist bigger and bigger sets F, G encoding a bipartite graph can then, after some effort, be expressed as a formula of the form $\mathbb{U}Z.\psi(Z)$, where the unboundedness of only a single set Z is required. The validity of such a formula in the unlabeled tree can be verified using either one of the Theorems 4.2.12 and 4.2.20, hence we obtain conceptually simple decidability proof for the problem [4]: "does a graph represented in the full binary tree have bounded tree-width?"

Another application is in deciding the winner in a certain type of pushdown game. A *pushdown game* is a two-player game obtained from a *pushdown graph*. The vertices of such a graph are the configurations $(q, \gamma) \in Q \times \Gamma^*$ of a pushdown automaton of state space Q and stack alphabet Γ, while the edges represent the transitions. The game is obtained by adding a

partition of Q into states Q_0 of player 0 and states Q_1 of player 1, along with a *winning condition*, or set of plays in $(Q \times \Gamma^*)^{\mathbb{N}}$ that are winning for the player 0. In [8], the authors consider the *bounded stack* winning condition, where a play is winning for player 0 if there is a fixed finite bound on the size of the stacks appearing in it. Using a natural interpretation of the pushdown game in a binary tree, the fact that player 0 wins the game from a fixed position v is equivalent to the satisfiability of a formula

$$\exists S_0 \; \forall S_1 \; \mathbb{B}X.\; \psi(S_0, S_1, X, v) \;,$$

in which $\psi(S_0, S_1, X, v)$ says that the stack represented by X appears in the unique play starting in node v and concordant with the strategies S_0 and S_1. We are able to quantify over strategies due to memoryless determinacy of the relevant game. Moreover, by a closer inspection of the game, one can show that $\forall S_1$ can be shifted inside the $\mathbb{B}$ quantifier, yielding an existential bounding formula whose satisfiability is decidable by Theorem 4.2.12.

Finally, the bounding quantifier can be applied to the following decision problem [6, 5]: "Is a given formula ψ of the modal μ-calculus with backward modalities satisfiable in some finite structure?" This problem and its relation with the bounding quantifier are the subject of Chapter 5 of this thesis.

4.4 Open Problems

Whether or not satisfiability is decidable for the whole logic MSOL+$\mathbb{B}$ remains an open problem. Short of giving a full decision procedure for satisfiability in MSOL+$\mathbb{B}$, one could also ask: can the procedure for the unbounding quantifier be extended to allow some nesting? Can the two procedures in Theorems 4.2.12 and 4.2.20 be somehow combined? Moreover, a better complexity assessment for Theorem 4.2.12 would be welcome.

Chapter 5

Finite Satisfiability

5.1 Introduction

The satisfiability decision problem – the question whether a sentence of a given logic can possibly be satisfied in some structure – is a cornerstone of logic in computer science. There is, however, a related decision problem – one we call *finite satisfiability* – which has received considerably less attention. This is the question whether a sentence is satisfied in some *finite* structure.

In a logic where every satisfiable sentence is also satisfiable in some finite structure, finite satisfiability is equivalent to satisfiability and there is no reason to consider the two as separate problems. Such a logic is said to have the *finite model property*. Although whether or not a logic has the finite model property is an important question, it is not the one we are interested in here. Our point of departure is a logic which is known *not* to have the finite model property; for such a logic we are interested in finite satisfiability as a decision problem whose decidability or complexity can be investigated.

For the finite satisfiability problem to be decidable, a logic cannot be too powerful; for the problem to be meaningful, it needs to be sufficiently powerful to express satisfiable sentences which are not finitely satisfiable. These are rather stringent requirements. On the one hand, logics without the finite model property tend to be difficult, cf. first-order logic over arbitrary relational structures, where both satisfiability and finite satisfiability are undecidable [65]. On the other hand, logics with decidable satisfiability tend to have the finite model property, eg. modal logic, the modal μ-calculus[37] and the Guarded Fragment of first-order logic[1, 32].

In recent years, a number of formalisms have appeared which simultaneously have decidable satisfiability and do not have the finite model property: two-way alternating tree automata, the modal μ-calculus with back-

ward modalities and guarded fixed point logic.

In this chapter we consider the finite satisfiability problem for the first two formalisms. We prove that it is decidable for alternating two-way automata over finite graphs. Then, using a well-established correspondence between alternating automata and the modal μ-calculus, we transfer this result to the modal μ-calculus with backward modalities.

The decidability proof uses properties similar to the ones discussed in Chapter 4. Indeed, finite satisfiability of a given alternating two-way automaton is reduced to the emptiness of an effectively obtained quasiregular language. Hence decidability of the finite satisfiability problem follows from Theorem 4.2.12.

We now proceed to give a more detailed overview of the chapter. In this overview we will introduce the logical formalisms considered in the chapter, state the main results and elaborate further on the connection between finite satisfiability and the bounding quantifier.

Alternating Tree Automata and the μ-Calculus

Alternating two-way tree automata with the parity condition were used by Vardi [67] to decide satisfiability for the modal μ-calculus with backward modalities. The latter is a logic which augments Kozen's modal μ-calculus [36, 3] with two additional modalities $\Diamond^-$ and $\Box^-$. These moalities have the following meaning: a formula $\Diamond^-\varphi$ states that φ is true in some predecessor of the current state, while $\Box^-\varphi$ states that φ is true in all predecessors of the current state.

Analogously to the μ-calculus with backward modalities, the two-way alternating tree automaton mentioned in the previous paragraph can, apart from the usual forward moves of one-way alternating automata, make moves which go backward, i. e. toward the root of the tree.

The correspondence between two-way alternating automata and the μ-calculus with backward modalities used by Vardi fits into a well-known connection between alternating automata and the μ-calculus that was discovered by Muller and Schupp [43, 42]. In both cases – when backward modalities are considered (Vardi) and when they are not (Muller and Schupp) – there exists a linear translation which assigns to a μ-calculus formula an equivalent alternating automaton. A linear translation in the other direction can also be presented, assuming a vectorial representation of fix-point formulas.

Apart from the μ-calculus, two-way alternating automata have also been used by Graedel and Walukiewicz in [29] to decide satisfiability for guarded fixed point logic, a logic obtained by adding fixed point operators to the Guarded Fragment of Andréka, Neméti and van Benthem [1].

The Finite Satisfiability Problem

There is an interesting common denominator of guarded fixed point logic, the μ-calculus with backward modalities and two-way alternating automata: none of these have a finite model property.

Modal logic and even Kozen's modal μ-calculus have the finite model property; the μ-calculus with backward modalities does not, as witnessed by the sentence

$$\nu X.(\Diamond^+ X \wedge \mu Y.\Box^- Y).$$

A similar situation occurs in the Guarded Fragment: guarded fixed point logic does not have the finite model property, contrary to the Guarded Fragment without fixed point operators [28].

Since the alternating two-way tree automata of Vardi work on infinite trees by definition, there is little sense in speaking about a finite model property. However, if the definition is tweaked a little and one considers two-way alternating automata on arbitrary graphs, finite or infinite, nonempty automata which are not finitely satisfiable can be found.

A formalism that, like the ones above, does not have the finite model property admits the *finite satisfiability problem* mentioned above:

> Is a given sentence/automaton finitely satisfiable?

In this chapter we present algorithms deciding the problem both for alternating two-way automata on graphs and the modal μ-calculus with backward modalities. The case of guarded fixed point logic is left open.

The finite satisfiability problem for two-way alternating automata was the subject of the paper [6], which presented a doubly exponential decision procedure for two-way alternating automata, where not the full parity condition, but the more restricted *Büchi condition* was used. That result is improved here in two ways: we solve the finite satisfiability problem for the general parity condition, and we do it in singly exponential time, which turns out to be optimal:

Theorem 5.2.22
The finite satisfiability problem for alternating two-way automata on graphs is EXPTIME *complete*

Using this result and the linear translation to and from the μ-calculus, we prove that the finite satisfiability problem for the μ-calculus with backward modalities is also EXPTIME complete.

Relationship with the Bounding Quantifier

Although our alternating two-way automata work on graphs, the proof of Theorem 5.2.22 deals mostly with trees: a finite graph is represented as an infinite tree – its two-way tree unraveling. Of course, not all infinite trees represent finite graphs – for this a special condition, called *bounded signature*, must be satisfied. Therefore, the finite satisfiability problem may be reduced to the emptiness problem for the language consisting of trees with bounded signature. Although this language is not regular, it can be expressed using an existential bounding formula from Theorem 4.2.12. Hence the decidability of the problem.

Unfortunately, although using the bounding quantifier gives a simple decidability proof, it does not yield the optimal complexity. That is why decidability is reproved directly in Section 5.2.7 without using the bounding quantifier. This is done by reducing finite satisfiability to emptiness for some other, effectively found, regular language. This approach is more complicated, requiring some technical automata manipulations, but yields the optimal EXPTIME complexity.

5.2 Two-Way Alternating Automata

In this section, we introduce two-way alternating automata and prove that their corresponding finite satisfiability problem is decidable. In the other section of the chapter, Section 5.3, this result will be applied to the μ-calculus with backward modalities.

5.2.1 Parity Games

We are going to define the semantics of two-way alternating automata on graphs by using a game approach. Therefore, before proceeding, we give in this section a short introduction to parity games.

Definition 5.2.1 (Parity game) A *parity game* is a tuple

$$\mathbb{G} = \langle V_\exists, V_\forall, E, v_s, \Omega \rangle$$

where $V_\exists$ and $V_\forall$ are disjoint sets of *positions*, the *ranking* function Ω assigns to every position in $V = V_\forall \cup V_\forall$ one of a finite number of *ranks* $\{0, \ldots, N\}$, $E \subseteq V \times V$ is the set of *edges*, and $v_s \in V$ is some fixed *starting position*.

The game is played by two players, $\exists$ and $\forall$. At each moment of the game there is a current position v from V, starting with the position v_s. If

the current position belongs to the set $V_\forall$, then the player $\forall$ chooses a new position w such that the edge (v, w) belongs to the set E. Otherwise, the new position is chosen in the same way, but by the player $\exists$. The play may reach a position from which there are no outgoing edges, in this case the player to whose set this position belongs loses for a lack of possible moves. Otherwise, the infinite sequence of position ranks assumed in the play is considered. If it satisfies the parity condition (cf. Definition 4.1.1), the player $\exists$ is declared the winner; otherwise the player $\forall$ is declared the winner.

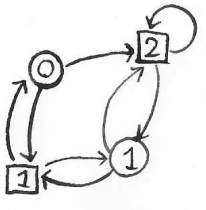


Figure 5.1: A parity game. Square positions are for $\forall$, round ones for $\exists$. No matter what position is picked as the starting one, the player $\forall$ wins.

A *strategy* for the player $i \in \{\forall, \exists\}$ is a mapping $s : V^*V_i \to V$ such that for each sequence $v_0 \cdots v_j \in V^*V_i$, there is an edge in E from v_j to the position $s(v_0 \cdots v_j)$. Given strategies $s_\forall, s_\exists$ for both players, a unique play $\boldsymbol{\rho}(s_\forall, s_\exists) \in V^\mathbb{N} \cup V^*$ consistent with these strategies is defined in the natural fashion. A strategy for player i is *winning* if for all strategies of the other other player, the resulting play is winning for player i.

A strategy is *memoryless* if the choice $s(v_0 \cdots v_j)$ depends solely upon the position v_j; we identify such a strategy with a mapping $V \to V$. A fundamental result [18, 41] says we only need consider memoryless strategies:

Theorem 5.2.2 (Memoryless determinacy theorem)
In every parity game one of the players has a winning strategy, which is, moreover, memoryless.

5.2.2 Two-way Alternating Automata

We have now sufficiently prepared the ground for introducing two-way alternating automata on graphs. These are a slight generalization of two-way alternating automata on infinite trees, which were studied by Vardi in [67] as a tool for deciding satisfiability in the modal μ-calculus with backward modalities. As opposed to "normal" one-way alternating automata, two-way automata can also move backward across edges.

In this chapter, when speaking of graphs, we mean *labeled graphs with a starting vertex*. Such a graph is a tuple

$$G = \langle V, E, \Sigma, e, v_s \rangle$$

where V is the set of *vertices*, $E \subseteq V \times V$ is the set of *edges*, the *labeling* is a function $e : V \to \Sigma$ and $v_s \in V$ is the distinguished *starting vertex*. We assume that the set Σ of *labels* is finite.

Let $W \subseteq V$ be a vertex set containing the starting vertex. The *subgraph of G induced by W* is the graph obtained from G by reducing the vertex set to W and restricting the edges and labeling accordingly. Given a vertex v of G, we write $G_{v_s:=v}$ for the graph where v is chosen as the new starting position.

Definition 5.2.3 (Two-way alternating automaton) A *two-way alternating automaton* on Σ-labeled graphs is a tuple:

$$\mathcal{A} = \langle Q_\exists, Q_\forall, q_I, \Sigma, \delta, \Omega \rangle$$

where $Q_\exists, Q_\forall$ are disjoint finite sets of *states*, $q_I \in Q = Q_\exists \cup Q_\forall$ is called the *initial state*, Σ is the finite *input alphabet* and Ω is a function assigning to each state a natural number called its *rank*. The *transition function* δ is of the form

$$\delta : Q \times \Sigma \to P(\{0, +, -\} \times Q).$$

Intuitively, whether or not the automaton $\mathcal{A}$ accepts a graph G depends on the outcome of a certain game played by two players $\forall$ and $\exists$. In the game, the players alternately guide the automaton's control, which at all times is placed over single vertex of the graph G and assumes one of the states in Q. The game starts in the initial state q_I in the distinguished starting vertex of G. Afterward, the automaton moves around the graph, the player deciding which move to make being chosen depending on whether the current state is in $Q_\exists$ or $Q_\forall$.

The set of possible moves depends on the value that is assigned by δ to the current state and the label in G of the current position. A choice of the form $(q, +)$ (respectively $(q, -)$) means the controlling player needs to change the state to q and move to some position along a forward (respectively backward) edge. Choosing $(q, 0)$ does not change the position, only the state of the automaton. The winner is determined as in the parity game.

The precise definition is as follows. Given a labeled graph G and a two-way alternating automaton $\mathcal{A}$:

$$G = \langle V, E, \Sigma, e, v_s \rangle \qquad \mathcal{A} = \langle Q_\exists, Q_\forall, q_I, \Sigma, \delta, \Omega \rangle,$$

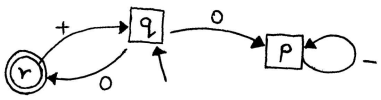
a parity game $\mathbb{G}(\mathcal{A}, G)$ is defined as follows:

- The positions for the player $\exists$ are the pairs in $Q_\exists \times V$, while the positions for the player $\forall$ are the pairs $Q_\forall \times V$;
- The starting position is the pair (q_I, v_s);
- There is an edge from the position (q, v) to the position (r, w) if:
  - $(0, r)$ belongs to $\delta(q, e(v))$ and $v = w$; or
  - $(-, r)$ belongs to $\delta(q, e(v))$ and (w, v) belongs to E; or
  - $(+, r)$ belongs to $\delta(q, e(v))$ and (v, w) belongs to E.
- The rank of a position (q, v) is the rank $\Omega(q)$.

Definition 5.2.4 (Acceptance by the automaton) We say that the automaton $\mathcal{A}$ *accepts a graph* G *under the strategy* s if s is a winning strategy for player $\exists$ in the game $\mathbb{G}(\mathcal{A}, G)$. Such a strategy is called *accepting* and $\mathcal{A}$ is said *accept the graph* G.

By Theorem 5.2.2, we assume without loss of generality that accepting strategies are memoryless. We conclude this section with an example of an automaton which accepts an infinite graph, yet rejects all finite ones.

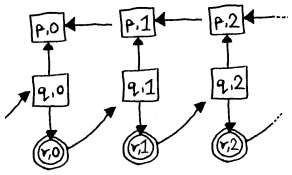
Example: Consider the following two-way alternating automaton $\mathcal{A}$ over a one letter input alphabet $\{\sigma\}$:



Since the alphabet has only one letter, we only draw the directions next to the transitions. The initial state is q, which is signified by the incoming arrow. In the picture, round states belong to $Q_\exists$ and square ones belong to $Q_\forall$; while states with a double border have rank 0 and states with a single border have rank 1.

Consider moreover the following graph, with all vertices labeled by σ:

The game $\mathbb{G}(\mathcal{A}, G)$ looks as follows (round positions are for the player $\exists$, square ones are for the player $\forall$; the starting position is $(q, 0)$):



In this game, all finite plays end in $(p, 0)$ while the only infinite one visits all the (r, i) vertices. Both types of play are winning for the player $\exists$. Hence the automaton $\mathcal{A}$ is nonempty because it accepts the graph G.

We will now show that $\mathcal{A}$ accepts no finite graph. Let G be a graph accepted by $\mathcal{A}$ and consider a vertex v. If the position (p, v) is winning for player $\exists$ in the game $\mathbb{G}(\mathcal{A}, G)$, then there is no infinite sequence of vertices $\boldsymbol{\pi}$ such that

$$\boldsymbol{\pi}_0 = v \quad \text{and} \quad (\boldsymbol{\pi}_{n+1}, \boldsymbol{\pi}_n) \text{ is an edge in } G \qquad \text{for all } \in \mathbb{N}. \tag{5.1}$$

On the other hand, if the position (q, v) is winning for player $\exists$ there must be an infinite sequence of vertices $\boldsymbol{\pi}$ such that

$$\boldsymbol{\pi}_0 = v \quad \text{and} \quad (\boldsymbol{\pi}_n, \boldsymbol{\pi}_{n+1}) \text{ is an edge in } G \qquad \text{for all } \in \mathbb{N}$$

and the positions $(q, \boldsymbol{\pi}_i)$ (and consequently $(p, \boldsymbol{\pi}_i)$) are all winning for the player $\exists$. The graph G cannot be finite, since the path $\boldsymbol{\pi}$ would yield a cycle and hence a path of the form (5.1), a contradiction. $\square$

5.2.3 The Finite Graph Problem

The example in Section 5.2.2 is a motivation for the following *finite satisfiability problem for two-way alternating automata:*

> Does a given alternating two-way automaton on graphs accept some finite graph?

It is crucial here that our alternating automata are two-way. Indeed, using a technique akin to [20], one can prove the following (a one-way alternating automaton is one where the image of the transition function is restricted to subsets of $\{0, +\} \times Q$):

Theorem 5.2.5 (One-way finite model property)
Every nonempty one-way alternating automaton accepts some finite graph.

5.2.4 Tree Unraveling

In this section we define a two-way tree unraveling operation that out of an arbitrary graph creates a tree-like one which is moreover indistinguishable by an alternating two-way automaton. Let us fix for the rest of Section 5.2 a two-way alternating automaton whose state space $Q_\exists \cup Q_\forall$ we will refer to as Q:

$$\mathcal{A} = \langle Q_\exists, Q_\forall, q_I, \Sigma, \delta, \Omega \rangle$$

For the sake of technical simplicity, we assume without loss of generality that every state of the automaton has an outgoing 0-transition, hence all plays in the games $\mathbb{G}(\mathcal{A}, G)$ are necessarily infinite.

Definition 5.2.6 (Trace) Let s be a strategy for player $\exists$ in the game $\mathbb{G}(\mathcal{A}, G)$ and let $\boldsymbol{\rho}$ be a play consistent with this strategy. Any contiguous – finite or infinite – subsequence of $\boldsymbol{\rho}$ is called a *trace* in (G, s).

Note that every position in a trace must be reachable from the starting position in the game $\mathbb{G}(\mathcal{A}, G)$. Traces will be denoted using the letter $\boldsymbol{\tau}$. We use $\mathrm{Traces}(G, s)$ to denote the set of all traces in (G, s). A trace $\boldsymbol{\tau}$ is a *sub-trace* of $\boldsymbol{\tau}'$, written as $\boldsymbol{\tau} \sqsubseteq \boldsymbol{\tau}'$, if $\boldsymbol{\tau}$ is a contiguous subsequence of $\boldsymbol{\tau}'$.

The following just rephrases acceptance of automata in terms of traces:

Fact 5.2.7 The automaton $\mathcal{A}$ accepts a graph G under the strategy s if and only if the ranks assumed in every infinite trace in (G, s) satisfy the parity condition.

A *two-way tree* is any graph which becomes a tree when directions of the edges are forgotten. The starting vertex of a two-way tree, which can be chosen arbitrarily, is called the *root*. In a two-way tree, a vertex v is a *successor* of a vertex w if there is an edge between v and w and every two-way path from v to the root leads through w. Note that the successor relation depends on the choice of the root. We denote two-way trees using the letter T.

Let T be a two-way tree. Given a vertex v of this tree, we use $T|_v$ to denote the set of vertices which can be connected to the root only via two-way paths leading through v. For $i \in \mathbb{N}$, we denote by $T|^i$ the set of vertices in the tree T that can be connected to the root with a two-way path of length at most i.

A *two-way path* in a graph G is sequence of vertices $\boldsymbol{\pi}$ where for all $i \in \mathbb{N}$, either $(\boldsymbol{\pi}_i, \boldsymbol{\pi}_{i+1})$ or $(\boldsymbol{\pi}_{i+1}, \boldsymbol{\pi}_i)$ is an edge in G. The *two-way tree unraveling* of a graph G is the two-way tree $un(G)$ obtained from G as follows. The vertex set is the set of finite two-way paths in G that begin in the starting

vertex of G. The label of such a two-way path is taken to be the label (in G) of its last vertex. If there is an edge from v to w in G, we include in $un(G)$ an edge from $\boldsymbol{\pi}\cdot v$ to its extension $\boldsymbol{\pi}\cdot v\cdot w$. If there is an edge from w to v, we include an edge from $\boldsymbol{\pi}\cdot v\cdot w$ to its prefix $\boldsymbol{\pi}\cdot v$. In particular, it is possible to have edges in both directions. The root of the unraveling is set to be the two-way path of length one that contains only the starting vertex of G. See Figure 5.2 for an example of a two-way tree unraveling.

We also define the canonical projection Π which maps a two-way tree vertex $\boldsymbol{\pi}\cdot v$ onto the vertex v of the original graph. Abusing the notation, we also call Π the function which to a trace in the unwinding $un(G)$ assigns the corresponding trace in G.

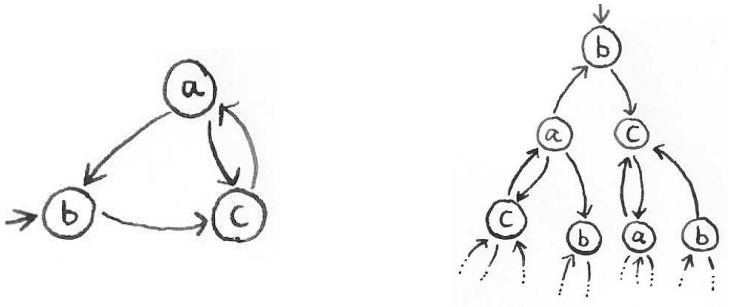


Figure 5.2: A graph and its unraveling

Strategies in the game $\mathbb{G}(\mathcal{A}, G)$ can be transferred to the game in the tree unraveling in the following manner. We define the *strategy unraveling* of a strategy s in the game $\mathbb{G}(\mathcal{A}, G)$ to be the following strategy $un(s)$ in the game $\mathbb{G}(\mathcal{A}, un(G))$:

$$un(s)(q, \boldsymbol{\pi}\cdot v) = (r, \boldsymbol{\pi}\cdot v\cdot w) \text{ where } s(q, v) = (r, w).$$

Lemma 5.2.8 $\boldsymbol{\Pi}(\mathrm{Traces}(un(G), un(s))) = \mathrm{Traces}(G, s)$

Proof (sketch)
It is enough to prove the lemma for traces starting in the starting position of the game $\mathbb{G}(\mathcal{A}, G)$. The proof is by induction on the length of the trace. □

An immediate Corollary of Fact 5.2.7 and Lemma 5.2.8 is:

Corollary 5.2.9 $\mathcal{A}$ accepts the graph G under a strategy s if and only if it accepts the unwinding $un(G)$ under the strategy $un(s)$.

5.2.5 Signature

Our approach to the finite satisfiability problem involves what we call an automaton signature. This is the technical concept behind the key Lemma 5.2.10, which allows us to transfer the finite satisfiability problem from graphs and alternating automata to tree languages and the logic MSOL+$\mathbb{B}$ from the previous chapter.

Consider a graph G accepted by $\mathcal{A}$ under some strategy s. For a trace $\boldsymbol{\tau} \in \mathrm{Traces}(G, s)$ and $k \in \mathbb{N}$, we denote by $|\boldsymbol{\tau}|_k$ the number of those positions in $\boldsymbol{\tau}$ where rank k is assumed and which are before the first occurrence of a rank smaller than k. Intuitively, $|\boldsymbol{\tau}|_k$ is a measure of the trace $\boldsymbol{\tau}$ which says how many states of rank k appear at the beginning of $\boldsymbol{\tau}$ without being "canceled" by some state of smaller rank. We will be interested in the value of $|\boldsymbol{\tau}|_k$ only when k is an odd rank.

Given an odd rank k, we associate with a position (q, v) in the game $\mathbb{G}(\mathcal{A}, G)$ its *k-signature*, i.e. the upper bound on possible values of $|\boldsymbol{\tau}|_k$ for traces $\boldsymbol{\tau}$ that start in (q, v). Note that the signature depends on the particular strategy s for the player $\exists$. This value, which can be finite or infinite, is denoted $\mathrm{Sig}^k_{G,s}(q, v)$. If the context is clear as to what graph and strategy are concerned, we simply write Sig^k.

We say (G, s) has a signature *bounded* by $N \in \mathbb{N}$ if the k-signature of every position is bounded by N for all odd ranks k. Such a graph G is said to have a *bounded signature*. Note that a graph may have many signatures, corresponding to many strategies; we require only one of them to be bounded here. The following lemma characterizes automata that accept finite graphs in terms of signatures and two-way trees:

Lemma 5.2.10 $\mathcal{A}$ accepts a finite graph if and only if some two-way tree has a bounded signature.

The proof of this lemma is long and will be spread across the next two subsections.

A Finite Graph Yields a Two-Way Tree with Bounded Signature

The two-way tree in question will simply be the unraveling of any finite graph accepted by $\mathcal{A}$. First we state two auxiliary lemmas.

Lemma 5.2.11 A finite graph accepted by $\mathcal{A}$ has a bounded signature.

Proof
We will prove that this signature can be bounded by the product of the size of the graph's vertex set V and the state space of $\mathcal{A}$. If the contrary were to

hold, there would be an accepting strategy s and a trace $\boldsymbol{\tau} \in \mathrm{Traces}(G, s)$ such that for some odd rank k,

$$|\boldsymbol{\tau}|_k > |V||Q| \ .$$

Without loss of generality we assume that the trace $\boldsymbol{\tau}$ does not contain states of rank less than k, otherwise we can consider the appropriate prefix. Since the trace $\boldsymbol{\tau}$ contains more than $|V||Q|$ states of rank k, some game position (q, v) corresponding to a state q of rank k must occur twice in $\boldsymbol{\tau}$. But this means that the player $\forall$ can close a loop containing this position. Since the strategy s is memoryless, there exists a play consistent with s which violates the parity condition, a contradiction with the assumption that s is an accepting strategy. □

Lemma 5.2.12 Tree unwinding does not increase the signature, i. e.

$$Sig^k_{un(G),un(s)}(q, \boldsymbol{\pi}\cdot v) \leq Sig^k_{G,s}(q, v)$$

Proof
This follows from Lemma 5.2.8. □

Having these two lemmas, the left to right implication in Lemma 5.2.10 follows easily. Indeed, assume that the automaton $\mathcal{A}$ accepts a finite graph G. By Lemma 5.2.9, it also accepts the unraveling $un(G)$ under the strategy $un(s)$. Moreover, for every vertex $\boldsymbol{\pi}\cdot v$ of the unraveling, every state $q \in Q$ and every odd rank k, we have the two inequalities:

$$Sig^k_{un(G),un(s)}(q, \boldsymbol{\pi}\cdot v) \leq Sig^k_{G,s}(q, v) \leq |V||Q|$$

The first one is due to Lemma 5.2.12, the second due to Lemma 5.2.11.

A Two-Way Tree with Bounded Signature Yields a Finite Graph

This direction is more involved. We will show that a two-way tree with bounded signature can be "looped" into a finite graph. Let G be a graph and s a strategy for $\exists$ in the game $\mathbb{G}(\mathcal{A}, G)$. We say a position (r, w) of the game is an *s-sequel* of the position (q, v) if it is possible to move in one step from the position (q, v) to the position (r, w) in the game $\mathbb{G}(\mathcal{A}, G)$, i.e. $(q, v)(r, w)$ belongs to $\mathrm{Traces}(G, s)$.

Let k be an odd rank. A *k-pseudosignature* is a function σ^k that assigns to every position in $\mathbb{G}(\mathcal{A}, G)$ an integer from a finite set $\{0, \ldots, N\}$ such that:

- If $\Omega(q) > k$ then $\sigma^k(q, v) \geq \sigma^k(r, w)$ for all s-sequels (r, w) of (q, v);

- If $\Omega(q) = k$ then $\sigma^k(q, v) > \sigma^k(r, w)$ for all s-sequels (r, w) of (q, v).

A *pseudosignature* for (G, s) is a system of k-pseudosignatures for all odd ranks. We say a graph G *admits a pseudosignature* if for some strategy s, there is a pseudosignature for (G, s).

Lemma 5.2.13 $\mathcal{A}$ accepts a finite graph if and only if this graph admits a pseudosignature.

Proof
For the left to right implication it is sufficient to notice that the signature is a pseudosignature. For the other direction, one shows that a pseudosignature majorizes the signature, so that each state with odd rank appears at most N times before a state of smaller rank state appears. □

The following lemma completes the proof of the right to left implication in Lemma 5.2.10.

Lemma 5.2.14 If a pseudosignature exists for some two-way tree then one exists for a finite graph.

Proof
Let s be a strategy which admits a pseudosignature σ in a two-way tree T. By removing vertices not used by the strategy s and duplicating some subtrees, we can assume that the two-way tree T has exactly $|Q|$ successors for each vertex. Hence, we can associate with T a bijection of the vertices V of T and the set of sequences Q^*:

$$\psi_T : V \to Q^* \tag{5.2}$$

such that for all vertices v of T, every successor w of v is assigned a sequence $\psi_T(v) \cdot q$, for some $q \in Q$. In the same spirit, we can encode a memoryless strategy s in $\mathbb{G}(\mathcal{A}, T)$ as a mapping

$$\varphi_s^T : V \to (Q \times \{-1, 0, 1, \ldots, |Q|\})^{Q_\exists} \tag{5.3}$$

which says what moves the player $\exists$ makes depending on the vertex of T and the state of the automaton. The neighbors are encoded by a number, where -1 is the ancestor, 0 stands for not moving and the positive numbers encode the successors according to the bijection ψ_T.

With every vertex v of T we shall associate two pieces of information constituting its *description*: the value $\varphi_s^T(v)$ and the values $\sigma^k(q, v)$ for all states q and odd ranks k. Since a pseudosignature has a finite image by definition, there exists a finite number of vertex descriptions. We can thus

find a number i such that all vertex descriptions in the set $T|^{i+1}$ appear already in the set $T|^i$.

Let $f : T|^{i+1} \to T|^i$ be any function that restricted to the set $T|^i$ is the identity mapping and which assigns to every vertex v from the set $T|^{i+1}$ a vertex of the same description. Such a function exists by assumption on i.

Using this function, we will construct the finite graph from the statement of the lemma. We set the vertex set to be $T|^i$. The starting vertex and labeling are inherited from the original two-way tree. The edge set of the new graph is:

$$\{(f(v), f(w)) : (v, w) \text{ is an edge in } T\} .$$

An easy verification shows that the original pseudosignature is also a pseudosignature for this graph. □

5.2.6 Applying Signatures to Decidability

In this section we present an existential bounding formula which expresses the fact that a two-way tree is accepted with a bounded signature. Since, by Lemma 5.2.10, the automaton $\mathcal{A}$ accepts a finite graph if and only if there is some two-way tree with a bounded signature, we can use the decision procedure from Theorem 4.2.12 to show decidability of the finite satisfiability problem.

Let G be a graph and s a strategy for the player $\exists$ in the game $\mathbb{G}(\mathcal{A}, G)$. Let k be an odd rank and let $\boldsymbol{\tau} \in \text{Traces}(G, s)$ be a trace where no state has rank smaller than k. A vertex v is a *bad point for* $\boldsymbol{\tau}$ if for some state q of rank k, the position (q, v) occurs in $\boldsymbol{\tau}$. A *bad point set* is a set of bad points for some trace. The following immediately follows from the definition of a signature:

Lemma 5.2.15 A graph has bounded signature if and only if the cardinality of bad point sets is bounded.

We will now show that in the case of two-way trees, the above property can be expressed using the bounding quantifier, indeed by an existential bounding formula. Before we do so, however, we need to introduce some encodings, since the bounding quantifier works over infinite trees, not two-way trees (which, technically, are graphs). For this, it will be convenient to consider, instead of infinite binary trees defined in the previous chapter, the more general *Q-ary infinite trees*. Such a tree is a mapping $Q^* \to \Sigma$. The definitions of regular languages, monadic second-order logic, etc. are extended in the natural manner from binary trees to Q-ary ones. All the

results from Chapter 4, in particular Theorem 4.2.12, remain valid for Q-ary trees.

Let T be a two-way tree whose every vertex has exactly Q successors. Using the ψ_T mapping defined in (5.2), we associate with a two-way tree T the tree

$$\tilde{T} : Q^* \to \Sigma \times \{\uparrow, \downarrow, \updownarrow\}$$

defined in the natural manner, with the first component encoding the label of a vertex and the second one saying what type of edges link it with its parent.

In a similar manner, using the mapping (5.3), a memoryless strategy s in the game $\mathbb{G}(\mathcal{A}, T)$ is encoded as the tree $\tilde{s} = \varphi_s^T \circ \psi_T^{-1}$. A set of vertices W in the two-way tree T is encoded using a tree $r_W : Q^* \to \{0, 1\}$ which assigns 1 exactly to the vertices in $\psi_T(W)$. If W is a bad point set, the tree r_W is said to *encode bad points*.

Lemma 5.2.16 The set L of trees $\tilde{T}\hat{}\tilde{s}\hat{}r$, where r encodes bad points in (T, s) is a regular tree language.

Proof
This set is recognized by an automaton which guesses an automaton trace $\boldsymbol{\tau} \in \text{Traces}(T, s)$ and confirms if all the vertices represented by r are bad points of $\boldsymbol{\tau}$. The trace $\boldsymbol{\tau}$ can be guessed by a finite automaton, since without loss of generality only traces which visit every vertex at most $|Q|$ times need be considered. □

Lemma 5.2.17 The set K of encodings of two-way trees that, under some strategy, have a bound on the cardinality of bad points sets, is defined by an existential bounding formula.

Proof
This set is obtained from the regular tree language L in the previous lemma by applying the bounding quantifier and existential quantification over strategies. □

Corollary 5.2.18 It is decidable whether a two-way alternating automaton accepts some finite graph.

Proof
Let $\mathcal{A}$ be a two-way alternating automaton. By Lemma 5.2.10, $\mathcal{A}$ accepts some finite graph if and only if some two-way tree T has a bounded signature. By Lemma 5.2.15 this is equivalent to the fact that T has a bound on the

cardinality of bad point sets. By Lemma 5.2.17, this is equivalent to $\tilde{T}$ belonging to the the tree language K from Lemma 5.2.17. Hence $\mathcal{A}$ accepts some finite graph if and only if K is nonempty. Nonemptiness, in turn, is decidable by Theorem 4.2.12. □

We also provide a bound of some sort on the size of graphs accepted by two-way alternating automata.

Fact 5.2.19 If $\mathcal{A}$ accepts some finite graph then it accepts a graph whose size is elementary in the size of $|\mathcal{A}|$.

Proof
This follows from Lemma 4.2.11 and the fact that all the automata constructions used here are at most exponential. □

5.2.7 EXPTIME Completeness

Although the bounding quantifier gives a conceptually simple decision procedure for the finite satisfiability problem, it does not unfortunately yield the optimal complexity. In this section, we conduct a direct analysis of the problem in order to prove that it is EXPTIME complete.

Let T be a two-way tree, s a strategy for $\exists$ in the game $\mathbb{G}(\mathcal{A}, T)$ and consider an infinite sequence of positions in this game. This sequence is a *forward bad trace in* (T, s) if it is an infinite trace in Traces(T, s) which violates the parity condition. It is a *backward bad trace in* (T, s) if its states violate the parity condition and it has infinitely many prefixes that, when reversed, are traces in Traces(T, s). We say (T, s) is *trace finite* if it contains no forward bad trace and no backward bad trace.

Lemma 5.2.20 If $\tilde{T}\hat{}\tilde{s}$ is a regular tree, then (T, s) has a bounded signature if and only if it is trace finite.

Proof
Obviously, if (T, s) is not trace finite then it cannot have a bounded signature. For the other direction, we will prove that if (T, s) does not have a bounded signature, then it is contains a bad trace.

One can prove, using an argument similar to the one in the proof of Lemma 4.2.9, that if bad point sets are of arbitrary size, there must exist a state q of odd rank k and a trace

$$(q, v) \cdot \tau \cdot (q, w)$$

with all ranks at least k, such that the subtrees

$$(\tilde{T}\hat{}\tilde{s})|_v \qquad (\tilde{T}\hat{}\tilde{s})|_w$$

are equal and, moreover, either $v < w$ or $w < v$. But this guarantees the existence of a bad trace: if $v < w$ then it is a forward bad trace, while if $w < v$ it is a backward bad trace. □

Lemma 5.2.21 The set L of trees $\tilde{T}\hat{}\tilde{s}$ such that (T, s) is trace finite can be recognized by an automaton whose state space is exponential in $|Q|$ and whose acceptance condition is polynomial in $|Q|$.

Proof
For purposes of this proof, we call an automaton *small* if its state space and acceptance condition are polynomial in $|Q|$ (the alphabet is allowed to be exponential). The disjunction of two small automata is also small.

The automaton recognizing the language L will be obtained from two small automata $\mathcal{C}$ and $\mathcal{D}$ via sum, complementation and projection. Before describing $\mathcal{C}$ and $\mathcal{D}$, we will first present the construction which makes out of them the automaton in the statement of the lemma.

For every pair (T, s) a unique labeling $l_{T,s}$ is defined, whose alphabet Γ is exponential in the size of Q:

$$l_{T,s} : Q^* \to \Gamma.$$

Intuitively, this labeling says what positions are reachable and what are the possible loops. The labeling is such that the small automaton $\mathcal{C}$ accepts a tree $\tilde{T}\hat{}\tilde{s}\hat{}l$ if and only if $l \neq l_{T,s}$. Moreover, having access to such a labeling, the small automaton $\mathcal{D}$ accepts the complement of L. More precisely, $\mathcal{D}$ accepts $\tilde{T}\hat{}\tilde{s}\hat{}l_{T,s}$ if and only if $\tilde{T}\hat{}\tilde{s}$ does not belong to L. Consequently, the language L can be characterized as:

$$L = \{\tilde{T}\hat{}\tilde{s} : \text{ both } \mathcal{C} \text{ and } \mathcal{D} \text{ reject } \tilde{T}\hat{}\tilde{s}\hat{}l \text{ for some } l\}$$

Therefore the language L is the projection of the complement of a language recognized by small automata. By the complexity of complementation for parity automata established in [17], an automaton for L of exponential state space and a polynomial acceptance condition can be found.

Having presented the construction, we now proceed to describe in more detail the labeling $l_{T,s}$ and the small automata $\mathcal{C}$ and $\mathcal{D}$. The labeling $l_{T,s}$ is the composition of the following labellings:

$$\begin{aligned} a_k, b_k : &\quad Q^* \to P(Q \times Q) \qquad k \text{ is an odd rank} \\ c : &\quad Q^* \to P(Q) \end{aligned}$$

which, intuitively, say what are the possible loops and which positions are reachable. Formally, we define the labellings to be correct, i. e. rejected by $\mathcal{C}$, if and only if:

- A pair (q, r) belongs to the set in the label $a_k(v)$ if and only if there is a trace from (q, v) to (r, v) which does not use states of rank less than k; the pair additionally belongs to $b_k(v)$ if a state of rank k is used in the trace. In particular, the pair (q, q) belongs to $a_k(v)$ if the rank of k is at least k and it also belongs to $b_k(v)$ if this rank is exactly k.
- $q \in c(v)$ if and only if (q, v) is reachable in (T, s)

Writing the small automaton $\mathcal{C}$ which accepts erroneous labellings is done by simulating an alternating automaton using a nondeterministic one. We omit the description here and refer interested readers to the paper of Vardi [67].

The small automaton $\mathcal{D}$ which rejects L having access to the labeling $l_{T,s}$ is defined to be the disjunction

$$\mathcal{D} = \bigvee (\mathcal{B}_k \vee \mathcal{F}_k)$$

over all odd ranks k of two types of small automata:

- $\mathcal{F}_k$, which accepts if there exists a forward bad trace whose smallest rank occurring infinitely often is k; and
- $\mathcal{B}_k$, which accepts if there exists a backward bad trace whose smallest rank occurring infinitely often is k.

Let us describe here only the small automaton $\mathcal{F}_k$, the automaton for backward traces $\mathcal{B}_k$ being defined in a similar manner. An infinite trace may contain a finite number of positions, in which case there is a vertex v and a state q such that

$$q \in c(v) \qquad \text{and} \qquad (q, q) \in b_k(v) \ .$$

This can be easily verified. Otherwise, the infinite trace can be decomposed into infinitely many finite subtraces

$$\boldsymbol{\tau} = \overbrace{(q_{i_0}, v_{i_0}), \ldots, (q_{i_1-1}, v_{i_1-1})}^{\boldsymbol{\tau}^0}, \overbrace{(q_{i_1}, v_{i_1}), \ldots, (q_{i_2-1}, v_{i_2-1})}^{\boldsymbol{\tau}^1}, \ldots$$

where for all $j \in \mathbb{N}$, the trace $\boldsymbol{\tau}^j$ begins and ends in the same vertex v_{i_j}, which is a successor of the vertex $v_{i_{j-1}}$ corresponding to the previous trace.

If the trace is a bad forward trace whose least rank occurring infinitely often is k, only ranks greater or equal to k appear in the subtraces $\boldsymbol{\tau}^i$ after some point $m \in \mathbb{N}$; moreover a state of rank k is assumed in infinitely many of these subtraces.

The automaton $\mathcal{F}_k$ follows this characterization by guessing an infinite path $\boldsymbol{\pi}$ in $\mathrm{dom}(\tilde{T})$, a point $m \in \mathbb{N}$ on this path, and two labellings $\boldsymbol{q}, \boldsymbol{r} : \mathrm{dom}(\boldsymbol{\pi}) \to Q$ of this path such that:

- The m-th position $(\boldsymbol{q}_m, \boldsymbol{\pi}_m)$ is reachable, i. e. belongs to the label $c(\boldsymbol{\pi}_i)$;
- For all $i \geq m$, there is a loop $(\boldsymbol{q}_i, \boldsymbol{r}_i) \in a_k(\boldsymbol{\pi}_i)$;
- Infinitely often this loop contains the rank k, i. e. $(\boldsymbol{q}_i, \boldsymbol{r}_i) \in b_k(\boldsymbol{\pi}_i)$;
- For all $i \geq m$, the strategy s admits a move from $(\boldsymbol{r}_i, \boldsymbol{\pi}_i)$ to $(\boldsymbol{q}_{i+1}, \boldsymbol{\pi}_{i+1})$.

□

Theorem 5.2.22
The finite satisfiability problem for alternating two-way automata on graphs is EXPTIME*-complete*

Proof
By Lemma 5.2.10, a two-way alternating automaton is finitely satisfiable if and only if the tree language

$$K = \{\tilde{T}\hat{}\tilde{s} : \text{ the signature of } (T, s) \text{ is bounded}\}$$

is nonempty. Obviously K is the sum of a family of regular languages:

$$K = \bigcup_{i \in \mathbb{N}} \{\tilde{T}\hat{}\tilde{s} : \text{ the signature of } (T, s) \text{ is bounded by } i\}$$

This and Lemma 5.2.20 show that K is L-quasiregular, where L is the language from Lemma 5.2.21. But then, by Fact 4.2.3:

$$K \neq \emptyset \qquad \text{iff} \qquad L \neq \emptyset.$$

Recall that the language L was recognized by an automaton of exponential state space, yet with only a polynomial acceptance condition. By [17], emptiness for nondeterministic parity tree automata can be tested in time $O((mn)^{3n})$, where m is the number of states, while n is the number of ranks in the acceptance condition of the automaton. Hence the EXPTIME membership.

For the hardness result, by Theorem 5.2.5, asking whether a one-way alternating automaton accepts some finite graph is equivalent to asking whether it accepts any graph at all. But emptiness for one-way alternating automata is well known to be EXPTIME-hard, for instance due to a reduction from the satisfiability problem for propositional dynamic logic [23]. □

5.3 The μ-Calculus

In this section we consider the modal μ-calculus with backward modalities. We prove that for every formula of this calculus, an equivalent alternating two-way automaton of the same size can be found; hence the finite satisfiability problem for the μ-calculus with backward modalities is EXPTIME-complete by a reduction to the automata case.

As was the case with monadic second-order logic, we use in formulas variables from some infinite set $\mathbb{V}$. For purposes of the μ-calculus we do not differentiate between types of variables.

Definition 5.3.1 (Formulas of the calculus) Let $\Sigma = \{P_1, \ldots, P_n\}$ be a finite set of *atomic propositions*. The set $\mathcal{F}_\Sigma$ of *formulas of the modal μ-calculus with backward modalities over* Σ is defined by the following grammar:

$$\begin{aligned} \mathcal{G} \rightarrow \;& \Sigma \mid \neg\Sigma \mid \mathbb{V} \mid \mathcal{G} \wedge \mathcal{G} \mid \mathcal{G} \vee \mathcal{G} \mid \\ & \Box^- \mathcal{G} \mid \Box^+ \mathcal{G} \mid \Diamond^- \mathcal{G} \mid \Diamond^+ \mathcal{G} \mid \\ & \mu\mathbb{V}.\mathcal{G} \mid \nu\mathbb{V}.\mathcal{G} \end{aligned}$$

We respectively call μ and ν the *least* and *greatest fix-point* operators. We call $\Box^+$ and $\Diamond^+$ the *forward universal modality* and the *forward existential modality*, respectively; while $\Box^-$ and $\Diamond^-$ are respectively called the *backward universal modality* and the *backward existential modality*. Without the two backward modalities $\Box^-$ and $\Diamond^-$, the logic is just the modal μ-calculus introduced by Kozen in [36, 3].

In this chapter, a μ-calculus formula is interpreted in a graph. It would be possible to use *Kripke structures*, where edges can be additionally labeled by some finite set of *actions*. This is, for instance, the approach taken by Vardi in [67]. Although our results would also hold for this generalization, for the sake of simplicity we stick to structures where there is only one type of edge.

Let Σ be a set of atomic propositions and consider a $P(\Sigma)$-labeled graph

$$G = \langle V, E, P(\Sigma), v_s, e \rangle$$

along with a *valuation* $\eta : V \to P(\mathbb{V})$. Given a formula $\psi \in \mathcal{F}_\Sigma$, the satisfaction relation $G, \eta \models \psi$ is defined below. Intuitively, $G, \eta \models \psi$ holds if ψ is true in the starting vertex v_s of the graph G, with the graph labeling e and the valuation η defining what atomic propositions and propositional variables are true in which vertex. The formal inductive definition is:

- $G, \eta \models P$ if $P \in e(v_s)$, where P is an atomic proposition;
- $G, \eta \models X$ if $X \in \eta(v_s)$, where X is a propositional variable;
- $G, \eta \models \psi_1 \wedge \psi_2$ if $G, \eta \models \psi_1$ and $G, \eta \models \psi_2$;
- $G, \eta \models \psi_1 \vee \psi_2$ if $G, \eta \models \psi_1$ or $G, \eta \models \psi_2$;
- $G, \eta \models \Diamond^+ \psi$ if $G_{v_s:=v}, \eta \models \psi$ for some $v \in V$ such that $(v_s, v) \in E$;
- $G, \eta \models \Diamond^- \psi$ if $G_{v_s:=v}, \eta \models \psi$ for some $v \in V$ such that $(v, v_s) \in E$;
- $G, \eta \models \Box^+ \psi$ if $G_{v_s:=v}, \eta \models \psi$ for all $v \in V$ such that $(v_s, v) \in E$;
- $G, \eta \models \Box^- \psi$ if $G_{v_s:=v}, \eta \models \psi$ for all $v \in V$ such that $(v, v_s) \in E$.

The semantics of the fix-point operators need some more explanation. Given a valuation η and a variable X, a formula ψ can be seen as a function ψ_η^X in the powerset lattice $P(V)$:

$$\psi_\eta^X(W) \quad = \quad \{v : G_{v_s:=v}, \eta[X := W] \models \psi\} \qquad \text{for } W \in P(V) \ .$$

Since formulas contain negation only next to atomic propositions, such a mapping is monotone and, by the Knaster-Tarski Theorem, has least and greatest fixed points, which are used to define the operators μ and ν:

- $G, \eta \models \mu X.\psi$ if v_s belongs to the least fixed point of ψ_η^X;
- $G, \eta \models \nu X.\psi$ if v_s belongs to the greatest fixed point of ψ_η^X.

If ψ is a sentence, i. e. contains no free variables, then the satisfaction relation $\models$ does not depend on the valuation η and we can, without risking ambiguity, write $G \models \psi$ instead of $G, \eta \models \psi$.

Example: Consider the following formula ψ over the empty set of atomic propositions:

$$\psi = \nu X.\ (\Diamond^+ X \wedge \mu Y.\ \Box^- Y).$$

An easy verification shows that a graph G satisfies

$$\mu Y.\ \Box^- Y$$

if and only if there is no infinite sequence of vertices $\boldsymbol{\pi}$ such that $\boldsymbol{\pi}_0$ is the starting vertex and there is an edge from $\boldsymbol{\pi}_{i+1}$ to $\boldsymbol{\pi}_i$ for all $i \in \mathbb{N}$. Moreover, the graph satisfies the whole formula ψ if and only if there is an infinite path $\boldsymbol{\pi}$ starting in the starting vertex, all of whose vertices satisfy the formula $\mu Y.\ \Box^- Y$. Hence, this formula is equivalent to the automaton from the example in Section 5.2.2 and is therefore satisfiable yet not finitely satisfiable. $\Box$

As in the case of alternating two-way automata, this example motivates the following decision problem:

> Is a given formula of the modal μ-calculus with backward modalities satisfied in some finite structure?

The remainder of this chapter is devoted to applying the result on alternating automata (Theorem 5.2.22) to prove decidability of this problem.

5.3.1 Automata on Models

In this section, we sketch the correspondence between the μ-calculus with backward modalities and alternating two-way automata. Let Σ be a finite set of atomic propositions and consider a sentence $\psi \in \mathcal{F}_\Sigma$. Without loss of generality, we assume that every variable is bound at most once in ψ. We will construct an alternating two-way automaton $\mathcal{A}_\psi$ on graphs such that for all $P(\Sigma)$-labeled graphs G,

$$\mathcal{A}_\psi \text{ accepts } G \qquad \text{iff} \qquad G \text{ satisfies } \psi\ .$$

This automaton is obtained directly from the structure of ψ. Let $cl(\psi)$ denote the smallest set of formulas that contains ψ and is closed under subformulas.

$$\text{if } \sigma X.\varphi(X) \in cl(\psi) \quad \text{then} \quad \varphi(\sigma X.\varphi(X)) \in cl(\psi) \qquad \text{for } \sigma \in \{\mu, \nu\}.$$

Within $cl(\psi)$ we distinguish two subsets: $cl_\exists(\psi)$, which consists of formulas of the form $\psi_1 \vee \psi_2$, $\Diamond^+\psi$ and $\Diamond^-\psi$; and $cl_\forall(\psi)$ – the remaining formulas. The alternating two-way automaton on $P(\Sigma)$-labelled graphs

$$\mathcal{A}_\psi = \langle Q_\exists, Q_\forall, q_I, P(\Sigma), \delta, \Omega \rangle$$

is defined as follows:

- The state sets are $Q_\exists = cl_\exists(\psi) \cup \{q_\top\}$ and $Q_\forall = cl_\forall(\psi) \cup \{q_\bot\}$, with $q_\top$ and $q_\bot$ being some new states;
- The initial state q_I is ψ;

- The input alphabet is $P(\Sigma)$;
- The transition function δ is defined as follows:
  - $\delta(q_\top, \Gamma) = \{(0, q_\top)\}$, $\delta(q_\bot, \Gamma) = \{(0, q_\bot)\}$;
  - For $P \in \Sigma$, $\delta(P, \Gamma) = \{(0, q_\top)\}$ if $P \in \Gamma$, $\{(0, q_\bot)\}$ otherwise;
  - For $P \in \Sigma$, $\delta(\neg P, \Gamma) = \{(0, q_\top)\}$ if $P \notin \Gamma$, $\{(0, q_\bot)\}$ otherwise;
  - $\delta(\psi_1 \vee \psi_2, \Gamma) = \delta(\psi_1 \wedge \psi_2, \Gamma) = \{(0, \psi_1), (0, \psi_2)\}$;
  - For $\sigma \in \{\mu, \nu\}$, $\delta(\sigma X.\varphi(X), \Gamma) = \{(0, \varphi(\varphi(X)))\}$;
  - For $X \in \mathit{Var}$, $\delta(X, \Gamma) = \{(0, \sigma X.\varphi(X))\}$, where $\sigma X.\varphi(X)$ is the unique formula in $cl(\psi)$ where the variable X is bound;
  - For $k \in \{-, +\}$, $\delta(\Diamond^k \varphi, \Gamma) = \delta(\Box^k \varphi, \Gamma) = \{(k, \varphi)\}$;
- The ranking function Ω is fixed so as to satisfy the following conditions:
  - $\Omega(q_\top)$ is even, while $\Omega(q_\bot)$ is odd;
  - If the variable Y occurs freely in the formula $\sigma X.\psi_1(X)$ then $\Omega(\sigma X.\psi_1(X)) > \Omega(\sigma Y.\psi_2(Y))$;
  - Fix-point formulas have even rank if and only if the operator is ν;
  - The other formulas have a rank bigger than all fix-point formulas.

The following Lemma follows from a game characterization of the μ-calculus that can be found in [45]:

Lemma 5.3.2 A graph satisfies ψ if and only if it is accepted by $\mathcal{A}_\psi$.

Theorem 5.3.3
The finite satisfiability problem for the μ-calculus with backward modalities is EXPTIME*-complete.*

Proof
Since the size of the automaton $\mathcal{A}_\psi$ is linear on the size of the formula ψ, the EXPTIME upper bound follows immediately from Lemma 5.3.2 and Theorem 5.2.22. The lower bound can be obtained in a similar way to the one in Theorem 5.2.22: a formula that does not use backward modalities is satisfiable if and only if it is finitely satisfiable [20], and satisfiability for such formulas is EXPTIME-hard [38, 17, 20]. □

5.4 Open Problems

The techniques used in this section could conceivably be applied to formalisms other than the μ-calculus and alternating automata. As mentioned in the introduction, two-way alternating automata were used in the paper [29] to decide the satisfiability of formulas in the Guarded Fragment extended with fixed points. Perhaps the bounding quantifier might be applied to solving the open question whether finite satisfiability is decidable for the Guarded Fragment with fixed points.

Bibliography

[1] H. Andréka, J. van Benthem, and I. Neméti. Modal logics and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

[2] M. Arbib, editor. *Algebraic Theory of Machines, Languages and Semigroups*. Academic Press, New York, 1968.

[3] A. Arnold and D. Niwiński. *Rudiments of μ-calculus*. Elsevier, 2001.

[4] K. Barthelmann. When can an equational simple graph be generated by hyperedge replacement? In *MFCS*, volume 1450 of *Lecture Notes in Computer Science*, pages 543–552, 1998.

[5] M. Bojańczyk. Two-way alternating automata and finite models. In *International Colloquium on Automata, Languages and Programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 833–844, 2002.

[6] M. Bojańczyk. The finite graph problem for two-way alternating automata. *Theoretical Computer Science*, 298(3):511–528, 2003.

[7] M. Bojańczyk and I. Walukiewicz. Characterizing EF and EX tree logics. To Appear.

[8] A. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with unboundedness and regular conditions. In *Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Science*, pages 88–99, 2003.

[9] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundl. Math.*, 6:66–92, 1960.

[10] O. Carton and W. Thomas. The monadic theory of morphic infinite words and generalizations. In *Mathematical Foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 275–284, 2000.

[11] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs: Workshop*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, 1981.

[12] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974.

[13] S. Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, New York, 1976.

[14] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the AMS*, 98:21–52, 1961.

[15] C. C. Elgot and M. O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *Journal of Symbolic Logic*, 31:169–181, 1966.

[16] E. A. Emerson and J. Y. Halpern. 'Sometimes' and 'not never' revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[17] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. In *Foundations of Computer Science*, pages 328–337, 1988.

[18] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Foundations of Computer Science*, pages 368–377, 1991.

[19] E. A. Emerson and P. A. Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, 1984.

[20] E. A. Emerson and R. S. Streett. An automaton theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81:249–264, 1989.

[21] Z. Esik and P. Weil. On certain logically defined tree languages. In *Foundations of Software Technology and Theoretical Computer Science*, volume 2914 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 2003.

[22] J. Esparza. Decidability of model-checking for infinite-state concurrent systems. *Acta Informatica*, 34:85–107, 1997.

[23] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[24] M. Franceschet, L. Afanasiev, M. de Rijke, and M. Marx. CTL model checking for processing simple XPath queries. In *Temporal Presentation and Reasoning*.

[25] C. Koch G. Gottlob. Monadic queries over tree-structured data. In *Logic in Computer Science*, pages 189–202, 2002.

[26] D. Gabbay, A. Pnuelli, S. Shelach, and J. Stavi. On the temporal analysis of fairness. In *Principles of Programming Languages*, pages 163–173, 1980.

[27] E. Grädel. Decision procedures for guarded logics. In *Automated Deduction*, volume 1632 of *Lecture Notes in Computer Science*, 1999.

[28] E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 1999.

[29] E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *Logic in Computer Science*, pages 45–54, 1999.

[30] T. Hafer and W. Thomas. Computation tree logic CTL and path quantifiers in the monadic theory of the binary tree. In *International Colloquium on Automata, Languages and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 260–279, 1987.

[31] U. Heuter. First-order properties of trees, star-free expressions, and aperiodicity. In *Symposium on Theoretical Aspects of Computer Science*, volume 294 of *Lecture Notes in Computer Science*, pages 136–148, 1988.

[32] I. Hodkinson. Loosely guarded fragment has finite model property. *Studia Logica*, 70(2):205–240, 2002.

[33] W. Holcombe. *Algebraic Automata Theory*. Number 1 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1982.

[34] J. A. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Univ. of California, Los Angeles, 1968.

[35] F. Klaedtke and H. Ruess. Parikh automata and monadic second–order logics with linear cardinality constraints. Technical Report 177, Institute of Computer Science at Freiburg University, 2002.

[36] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[37] D. Kozen. A finite model theorem for the propositional μ-calculus. *Studia Logica*, 47(3):234–241, 1988.

[38] D. Kozen and R. Parikh. A decision procedure for the modal μ-calculus. In *Logic of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 313–325, 1983.

[39] K. Krohn and J. Rhodes. The algebreaic theory of machinesi. *Transactions of the AMS*, 116:450–464, 1965.

[40] R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.

[41] A. Mostowski. Games with forbidden positions. Technical report, University of Gdask, 1991.

[42] D. E. Muller, A. Saoudi, and P. E. Shupp. Weak alternating automata give a simple explanation why most temporal and dynamic logics are decidable in exponential time. In *Logic in Computer Science*, pages 422–427, 1988.

[43] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[44] D. Niwiński. On the cardinality of sets of infinite trees recognizable by infinite automata. In *Mathematical Foundations of Computer Science*, volume 520 of *Lecture Notes in Computer Science*, 1991.

[45] D. Niwiński. μ-calculus via games. In *Computer Science Logic*, volume 2471 of *Lecture Notes in Computer Science*, pages 27–43, 2002.

[46] M. Otto. Eliminating recursion in the μ-calculus. In *Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 531–540, 1999.

[47] J. Pin. Logic, semigroups and automata on words. *Annals of Mathematics and Artificial Intelligence*, 16:343–384, 1996.

[48] A. Potthoff. First-order logic on finite trees. In *Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 125–139, 1995.

[49] A. Potthoff and W. Thomas. Regular tree languages without unary symbols are star-free. In *Fundamentals of Computation Theory*, volume 710 of *Lecture Notes in Computer Science*, pages 396–405, 1993.

[50] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–23, 1969.

[51] M. O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Providence, RI, 1972.

[52] F. P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, 30:264–285, 1930.

[53] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

[54] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal of Computing*, 19:424–437, 1990.

[55] S. Shamir, O. Kupferman, and E. Shamir. Branching-depth hierarchies. In *Expressiveness in Concurrency*, Electronic Notes in Theoretical Computer Science, 2000.

[56] G. Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41:305–318, 1985.

[57] J. Stern. Complexity of some problems from the theory of automata. *Information and Computation*, 66:163–176, 1985.

[58] H. Straubing. *Finite Automata, Formal Languages, and Circuit Complexity*. Birkhäuser, Boston, 1994.

[59] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[60] D. Thérien and T. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *ACM Symposium on the Theory of Computing*, pages 256–263, 1998.

[61] W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25:360–375, 1982.

[62] W. Thomas. Logical aspects in the study of tree languages. In *Colloquium on Trees and Algebra in Programming*, pages 31–50, 1984.

[63] W. Thomas. Automata on infinite objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–192. Elsevier and MIT Press, 1990.

[64] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.

[65] B. A. Trakthenbrot. The impossibility of an algorithm for the decision problem for finite domains (russian). *Doklady Academii Nauk SSSR*, 70:569–572, 1950.

[66] B. A. Trakthenbrot. Finite automata and the logic of monadic second order predicates(russian). *Doklady Academii Nauk SSSR*, 140:326–329, 1961.

[67] M. Vardi. Reasoning about the past with two-way automata. In *International Colloquium on Automata, Languages and Programming*, number 1443 in Lecture Notes in Computer Science, pages 628–641, 1998.

[68] I. Walukiewicz. Model checking CTL properties of pushdown systems. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138, 2000.

[69] I. Walukiewicz. Deciding low levels of tree-automata hierarchy. In *Workshop on Logic, Language, Information and Computation*, volume 67 of *Electronic Notes in Theoretical Computer Science*, 2002.

[70] T. Wilke. Classifying discrete temporal properties. In *Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46, 1999.

Index