Tree Automata Techniques and Applications

Hubert Comon Max Dauchet Rémi Gilleron
Florent Jacquemard Denis Lugiez Sophie Tison
Marc Tommasi

Contents

Acknowledgments

Many people gave substantial suggestions to improve the contents of this book. These are, in alphabetic order, Witold Charatonik, Zoltan Fülöp, Werner Kuich, Markus Lohrey, Jun Matsuda, Aart Middeldorp, Hitoshi Ohsaki, P. K. Manivannan, Masahiko Sakai, Helmut Seidl, Stephan Tobies, Ralf Treinen, Thomas Uribe, Sandor Vágvölgyi, Kumar Neeraj Verma, Toshiyuki Yamada.

Introduction

During the past few years, several of us have been asked many times about references on finite tree automata. On one hand, this is the witness of the liveness of this field. On the other hand, it was difficult to answer. Besides several excellent survey chapters on more specific topics, there is only one monograph devoted to tree automata by Gécseg and Steinby. Unfortunately, it is now impossible to find a copy of it and a lot of work has been done on tree automata since the publication of this book. Actually using tree automata has proved to be a powerful approach to simplify and extend previously known results, and also to find new results. For instance recent works use tree automata for application in abstract interpretation using set constraints, rewriting, automated theorem proving and program verification, databases and XML schema languages.

Tree automata have been designed a long time ago in the context of circuit verification. Many famous researchers contributed to this school which was headed by A. Church in the late 50's and the early 60's: B. Trakhtenbrot, J.R. Büchi, M.O. Rabin, Doner, Thatcher, etc. Many new ideas came out of this program. For instance the connections between automata and logic. Tree automata also appeared first in this framework, following the work of Doner, Thatcher and Wright. In the 70's many new results were established concerning tree automata, which lose a bit their connections with the applications and were studied for their own. In particular, a problem was the very high complexity of decision procedures for the monadic second order logic. Applications of tree automata to program verification revived in the 80's, after the relative failure of automated deduction in this field. It is possible to verify temporal logic formulas (which are particular Monadic Second Order Formulas) on simpler (small) programs. Automata, and in particular tree automata, also appeared as an approximation of programs on which fully automated tools can be used. New results were obtained connecting properties of programs or type systems or rewrite systems with automata.

Our goal is to fill in the existing gap and to provide a textbook which presents the basics of tree automata and several variants of tree automata which have been devised for applications in the aforementioned domains. We shall discuss only *finite tree* automata, and the reader interested in infinite trees should consult any recent survey on automata on infinite objects and their applications (See the bibliography). The second main restriction that we have is to focus on the operational aspects of tree automata. This book should appeal the reader who wants to have a simple presentation of the basics of tree automata, and to see how some variations on the idea of tree automata have provided a nice tool for solving difficult problems. Therefore, specialists of the domain probably know almost all the material embedded. However, we think that this book can

be helpful for many researchers who need some knowledge on tree automata. This is typically the case of a PhD student who may find new ideas and guess connections with his (her) own work.

Again, we recall that there is no presentation nor discussion of tree automata for infinite trees. This domain is also in full development mainly due to applications in program verification and several surveys on this topic do exist. We have tried to present a tool and the algorithms devised for this tool. Therefore, most of the proofs that we give are constructive and we have tried to give as many complexity results as possible. We don't claim to present an exhaustive description of all possible finite tree automata already presented in the literature and we did some choices in the existing menagerie of tree automata. Although some works are not described thoroughly (but they are usually described in exercises), we think that the content of this book gives a good flavor of what can be done with the simple ideas supporting tree automata.

This book is an open work and we want it to be as interactive as possible. Readers and specialists are invited to provide suggestions and improvements. Submissions of contributions to new chapters and improvements of existing ones are welcome.

Among some of our choices, let us mention that we have not defined any precise language for describing algorithms which are given in some pseudo algorithmic language. Also, there is no citation in the text, but each chapter ends with a section devoted to bibliographical notes where credits are made to the relevant authors. Exercises are also presented at the end of each chapter.

Tree Automata Techniques and Applications is composed of seven main chapters (numbered 1– 7). The first one presents tree automata and defines recognizable tree languages. The reader will find the classical algorithms and the classical closure properties of the class of recognizable tree languages. Complexity results are given when they are available. The second chapter gives an alternative presentation of recognizable tree languages which may be more relevant in some situations. This includes regular tree grammars, regular tree expressions and regular equations. The description of properties relating regular tree languages and context-free word languages form the last part of this chapter. In Chapter 3, we show the deep connections between logic and automata. In particular, we prove in full details the correspondence between finite tree automata and the weak monadic second order logic with k successors. We also sketch several applications in various domains.

Chapter 4 presents a basic variation of automata, more precisely automata with equality constraints. An equality constraint restricts the application of rules to trees where some subtrees are equal (with respect to some equality relation). Therefore we can discriminate more easily between trees that we want to accept and trees that we must reject. Several kinds of constraints are described, both originating from the problem of non-linearity in trees (the same variable may occur at different positions).

In Chapter 5 we consider automata which recognize sets of sets of terms. Such automata appeared in the context of set constraints which themselves are used in program analysis. The idea is to consider, for each variable or each predicate symbol occurring in a program, the set of its possible values. The program gives constraints that these sets must satisfy. Solving the constraints gives an upper approximation of the values that a given variable can take. Such an approximation can be used to detect errors at compile time: it acts exactly as

a typing system which would be inferred from the program. Tree set automata (as we call them) recognize the sets of solutions of such constraints (hence sets of sets of trees). In this chapter we study the properties of tree set automata and their relationship with program analysis.

Originally, automata were invented as an intermediate between function description and their implementation by a circuit. The main related problem in the sixties was the *synthesis problem*: which arithmetic recursive functions can be achieved by a circuit? So far, we only considered tree automata which accepts sets of trees or sets of tuples of trees (Chapter 3) or sets of sets of trees (Chapter 5). However, tree automata can also be used as a computational device. This is the subject of Chapter 6 where we study *tree transducers*.

Preliminaries

Terms

We denote by N the set of positive integers. We denote the set of finite strings over N by N^*. The empty string is denoted by ε.

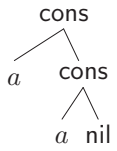
A ***ranked alphabet*** is a couple $(\mathcal{F}, Arity)$ where $\mathcal{F}$ is a finite set and $Arity$ is a mapping from $\mathcal{F}$ into N. The ***arity*** of a symbol $f \in \mathcal{F}$ is $Arity(f)$. The set of symbols of arity p is denoted by $\mathcal{F}_p$. Elements of arity $0, 1, \ldots p$ are respectively called constants, unary, $\ldots$, p-ary symbols. We assume that $\mathcal{F}$ contains at least one constant. In the examples, we use parenthesis and commas for a short declaration of symbols with arity. For instance, $f(,)$ is a short declaration for a binary symbol f.

Let $\mathcal{X}$ be a set of constants called ***variables***. We assume that the sets $\mathcal{X}$ and $\mathcal{F}_0$ are disjoint. The set $T(\mathcal{F}, \mathcal{X})$ of ***terms*** over the ranked alphabet $\mathcal{F}$ and the set of variables $\mathcal{X}$ is the smallest set defined by:

- $\mathcal{F}_0 \subseteq T(\mathcal{F}, \mathcal{X})$ and
- $\mathcal{X} \subseteq T(\mathcal{F}, \mathcal{X})$ and
- if $p \geq 1$, $f \in \mathcal{F}_p$ and $t_1, \ldots, t_p \in T(\mathcal{F}, \mathcal{X})$, then $f(t_1, \ldots, t_p) \in T(\mathcal{F}, \mathcal{X})$.

If $\mathcal{X} = \emptyset$ then $T(\mathcal{F}, \mathcal{X})$ is also written $T(\mathcal{F})$. Terms in $T(\mathcal{F})$ are called ***ground terms***. A term t in $T(\mathcal{F}, \mathcal{X})$ is ***linear*** if each variable occurs at most once in t.

Example 1. Let $\mathcal{F} = \{\mathsf{cons}(,), \mathsf{nil}, a\}$ and $\mathcal{X} = \{x, y\}$. Here cons is a binary symbol, nil and a are constants. The term $\mathsf{cons}(x, y)$ is linear; the term $\mathsf{cons}(x, \mathsf{cons}(x, \mathsf{nil}))$ is non linear; the term $\mathsf{cons}(a, \mathsf{cons}(a, \mathsf{nil}))$ is a ground term. Terms can be represented in a graphical way. For instance, the term $\mathsf{cons}(a, \mathsf{cons}(a, \mathsf{nil}))$ is represented by:



Terms and Trees

A finite ordered ***tree*** t over a set of labels E is a mapping from a prefix-closed set $\mathcal{P}os(t) \subseteq N^*$ into E. Thus, a term $t \in T(\mathcal{F}, \mathcal{X})$ may be viewed as a finite

ordered ranked tree, the leaves of which are labeled with variables or constant symbols and the internal nodes are labeled with symbols of positive arity, with out-degree equal to the arity of the label, *i.e.*a term $t \in T(\mathcal{F}, \mathcal{X})$ can also be defined as a partial function $t : N^* \to \mathcal{F} \cup \mathcal{X}$ with domain $\mathcal{P}os(t)$ satisfying the following properties:

(i) $\mathcal{P}os(t)$ is nonempty and prefix-closed.

(ii) $\forall p \in \mathcal{P}os(t)$, if $t(p) \in \mathcal{F}_n, n \geq 1$, then $\{j \mid pj \in \mathcal{P}os(t)\} = \{1, \ldots, n\}$.

(iii) $\forall p \in \mathcal{P}os(t)$, if $t(p) \in \mathcal{X} \cup \mathcal{F}_0$, then $\{j \mid pj \in \mathcal{P}os(t)\} = \emptyset$.

We confuse terms and trees, that is we only consider finite ordered ranked trees satisfying (i), (ii) and (iii). The reader should note that finite ordered trees with bounded rank k – *i.e.*there is a bound k on the out-degrees of internal nodes – can be encoded in finite ordered ranked trees: a label $e \in E$ is associated with k symbols $(e, 1)$ of arity 1, ..., (e, k) of arity k.

Each element in $\mathcal{P}os(t)$ is called a ***position***. A ***frontier position*** is a position p such that $\forall j \in N$, $pj \notin \mathcal{P}os(t)$. The set of frontier positions is denoted by $\mathcal{FP}os(t)$. Each position p in t such that $t(p) \in \mathcal{X}$ is called a ***variable position***. The set of variable positions of p is denoted by $\mathcal{VP}os(t)$. We denote by $\mathcal{H}ead(t)$ the ***root symbol*** of t which is defined by $\mathcal{H}ead(t) = t(\varepsilon)$.

SubTerms

A ***subterm*** $t|_p$ of a term $t \in T(\mathcal{F}, \mathcal{X})$ at position p is defined by the following:
- $\mathcal{P}os(t|_p) = \{j \mid pj \in \mathcal{P}os(t)\}$,
- $\forall q \in \mathcal{P}os(t|_p)$, $t|_p(q) = t(pq)$.

We denote by $t[u]_p$ the term obtained by replacing in t the subterm $t|_p$ by u.

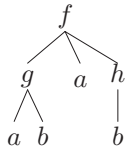
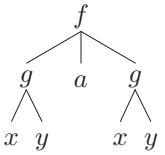
We denote by $\trianglerighteq$ the ***subterm ordering***, *i.e.*we write $t \trianglerighteq t'$ if t' is a subterm of t. We denote $t \triangleright t'$ if $t \trianglerighteq t'$ and $t \neq t'$.

A set of terms F is said to be ***closed*** if it is closed under the subterm ordering, *i.e.*$\forall t \in F$ $(t \trianglerighteq t' \Rightarrow t' \in F)$.

Functions on Terms

The ***size*** of a term t, denoted by $\|t\|$ and the ***height*** of t, denoted by $\mathcal{H}eight(t)$ are inductively defined by:
- $\mathcal{H}eight(t) = 0$, $\|t\| = 0$ if $t \in \mathcal{X}$,
- $\mathcal{H}eight(t) = 1$, $\|t\| = 1$ if $t \in \mathcal{F}_0$,
- $\mathcal{H}eight(t) = 1+\max(\{\mathcal{H}eight(t_i) \mid i \in \{1, \ldots, n\}\})$, $\|t\| = 1+\sum_{i\in\{1,\ldots,n\}} \|t_i\|$ if $\mathcal{H}ead(t) \in \mathcal{F}_n$.

Example 2. Let $\mathcal{F} = \{f(,,), g(,), h(), a, b\}$ and $\mathcal{X} = \{x, y\}$. Consider the terms

$t =$  ; $t' =$ 

The root symbol of t is f; the set of frontier positions of t is $\{11, 12, 2, 31\}$; the set of variable positions of t' is $\{11, 12, 31, 32\}$; $t|_3 = h(b)$; $t[a]_3 = f(g(a, b), a, a)$; $\mathcal{H}eight(t) = 3$; $\mathcal{H}eight(t') = 2$; $\|t\| = 7$; $\|t'\| = 4$.

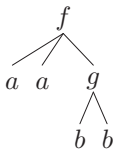
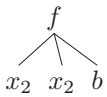
Substitutions

A ***substitution*** (respectively a ***ground substitution***) σ is a mapping from $\mathcal{X}$ into $T(\mathcal{F}, \mathcal{X})$ (respectively into $T(\mathcal{F})$) where there are only finitely many variables not mapped to themselves. The ***domain*** of a substitution σ is the subset of variables $x \in \mathcal{X}$ such that $\sigma(x) \neq x$. The substitution $\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ is the identity on $\mathcal{X} \setminus \{x_1, \ldots, x_n\}$ and maps $x_i \in \mathcal{X}$ on $t_i \in T(\mathcal{F}, \mathcal{X})$, for every index $1 \leq i \leq n$. Substitutions can be extended to $T(\mathcal{F}, \mathcal{X})$ in such a way that:

$$\forall f \in \mathcal{F}_n, \forall t_1, \ldots, t_n \in T(\mathcal{F}, \mathcal{X}) \quad \sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n)).$$

We confuse a substitution and its extension to $T(\mathcal{F}, \mathcal{X})$. Substitutions will often be used in postfix notation: $t\sigma$ is the result of applying σ to the term t.

Example 3. Let $\mathcal{F} = \{f(,,), g(,), a, b\}$ and $\mathcal{X} = \{x_1, x_2\}$. Let us consider the term $t = f(x_1, x_1, x_2)$. Let us consider the ground substitution $\sigma = \{x_1 \leftarrow a, x_2 \leftarrow g(b, b)\}$ and the substitution $\sigma' = \{x_1 \leftarrow x_2, x_2 \leftarrow b\}$. Then

$t\sigma = t\{x_1 \leftarrow a, x_2 \leftarrow g(b, b)\} =$  ; $t\sigma' = t\{x_1 \leftarrow x_2, x_2 \leftarrow b\} =$ 

Contexts

Let $\mathcal{X}_n$ be a set of n variables. A linear term $C \in T(\mathcal{F}, \mathcal{X}_n)$ is called a ***context*** and the expression $C[t_1, \ldots, t_n]$ for $t_1, \ldots, t_n \in T(\mathcal{F})$ denotes the term in $T(\mathcal{F})$ obtained from C by replacing variable x_i by t_i for each $1 \leq i \leq n$, that is $C[t_1, \ldots, t_n] = C\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$. We denote by $\mathcal{C}^n(\mathcal{F})$ the set of contexts over $(x_1, \ldots, x_n)$.

We denote by $\mathcal{C}(\mathcal{F})$ the set of contexts containing a single variable. A context is trivial if it is reduced to a variable. Given a context $C \in \mathcal{C}(\mathcal{F})$, we denote by C^0 the trivial context, C^1 is equal to C and, for $n > 1$, $C^n = C^{n-1}[C]$ is a context in $\mathcal{C}(\mathcal{F})$.

Chapter 1

Recognizable Tree Languages and Finite Tree Automata

In this chapter, we present basic results on finite tree automata in the style of the undergraduate textbook on finite automata by Hopcroft and Ullman [HU79]. Finite tree automata deal with finite ordered ranked trees or finite ordered trees with bounded rank. We discuss unordered and/or unranked finite trees in the bibliographic notes (Section 1.9). We assume that the reader is familiar with finite automata. Words over a finite alphabet can be viewed as unary terms. For instance a word abb over $A = \{a, b\}$ can be viewed as a unary term $t = a(b(b(\sharp)))$ over the ranked alphabet $\mathcal{F} = \{a(), b(), \sharp\}$ where $\sharp$ is a new constant symbol. The theory of tree automata arises as a straightforward extension of the theory of word automata when words are viewed as unary terms.

In Section 1.1, we define bottom-up finite tree automata where "bottom-up" has the following sense: assuming a graphical representation of trees or ground terms with the root symbol at the top, an automaton starts its computation at the leaves and moves upward. Recognizable tree languages are the languages recognized by some finite tree automata. We consider the deterministic case and the nondeterministic case and prove the equivalence. In Section 1.2, we prove a pumping lemma for recognizable tree languages. This lemma is useful for proving that some tree languages are not recognizable. In Section 1.3, we prove the basic closure properties for set operations. In Section 1.4, we define tree homomorphisms and study the closure properties under these tree transformations. In this Section the first difference between the word case and the tree case appears. Indeed, ecognizable word languages are closed under homomorphisms but recognizable tree languages are closed only under a subclass of tree homomorphisms: linear homomorphisms, where duplication of trees is forbidden. We will see all along this textbook that non linearity is one of the main difficulties for the tree case. In Section 1.5, we prove a Myhill-Nerode Theorem for tree languages and the existence of a unique minimal automaton. In Section 1.6, we define top-down tree automata. A second difference appears with the word case because it is proved that deterministic top-down tree automata are strictly less powerful than nondeterministic ones. The last section of the present chapter

gives a list of complexity results.

1.1 Finite Tree Automata

Nondeterministic Finite Tree Automata

A ***finite Tree Automaton*** (NFTA) over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ where Q is a set of (unary) states, $Q_f \subseteq Q$ is a set of final states, and Δ is a set of transition rules of the following type:

$$f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n)),$$

where $n \geq 0$, $f \in \mathcal{F}_n$, $q, q_1, \ldots, q_n \in Q$, $x_1, \ldots, x_n \in \mathcal{X}$.

Tree automata over $\mathcal{F}$ run on ground terms over $\mathcal{F}$. An automaton starts at the leaves and moves upward, associating along a run a state with each subterm inductively. Let us note that there is no initial state in a NFTA, but, when $n = 0$, *i.e.*when the symbol is a constant symbol a, a transition rule is of the form $a \to q(a)$. Therefore, the transition rules for the constant symbols can be considered as the "initial rules". If the direct subterms $u_1, \ldots, u_n$ of $t = f(u_1, \ldots, u_n)$ are labeled with states $q_1, \ldots, q_n$, then the term t will be labeled by some state q with $f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n)) \in \Delta$. We now formally define the move relation defined by a NFTA.

Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a NFTA over $\mathcal{F}$. The ***move relation*** $\to_{\mathcal{A}}$ is defined by: let $t, t' \in T(\mathcal{F} \cup Q)$,

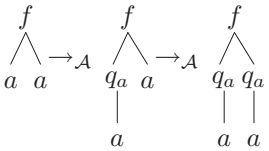
$$t \underset{\mathcal{A}}{\to} t' \Leftrightarrow \begin{cases} \exists C \in \mathcal{C}(\mathcal{F} \cup Q), \exists u_1, \ldots, u_n \in T(\mathcal{F}), \\ \exists f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n)) \in \Delta, \\ t = C[f(q_1(u_1), \ldots, q_n(u_n))], \\ t' = C[q(f(u_1, \ldots, u_n))]. \end{cases}$$

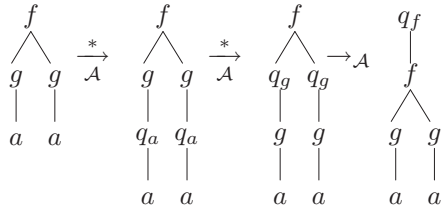
$\underset{\mathcal{A}}{\overset{*}{\to}}$ is the reflexive and transitive closure of $\to_{\mathcal{A}}$.

Example 4. Let $\mathcal{F} = \{f(,), g(), a\}$. Consider the automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = \{q_a, q_g, q_f\}$, $Q_f = \{q_f\}$, and Δ is the following set of transition rules:

$$\left\{ \begin{array}{rcl@{\qquad}rcl} a & \to & q_a(a) & g(q_a(x)) & \to & q_g(g(x)) \\ g(q_g(x)) & \to & q_g(g(x)) & f(q_g(x), q_g(y)) & \to & q_f(f(x,y)) \end{array} \right\}$$

We give two examples of reductions with the move relation $\to_{\mathcal{A}}$





A ground term t in $T(\mathcal{F})$ is ***accepted*** by a finite tree automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ if

$$t \xrightarrow[\mathcal{A}]{*} q(t)$$

for some state q in Q_f. The reader should note that our definition corresponds to the notion of nondeterministic finite tree automaton because our finite tree automaton model allows zero, one or more transition rules with the same left-hand side. Therefore there are possibly more than one reduction starting with the same ground term. And, a ground term t is accepted if there is one reduction (among all possible reductions) starting from this ground term and leading to a configuration of the form $q(t)$ where q is a final state. The tree language $L(\mathcal{A})$ ***recognized*** by $\mathcal{A}$ is the set of all ground terms accepted by $\mathcal{A}$. A set L of ground terms is ***recognizable*** if $L = L(\mathcal{A})$ for some NFTA $\mathcal{A}$. The reader should also note that when we talk about the set recognized by a finite tree automaton $\mathcal{A}$ we are referring to the specific set $L(\mathcal{A})$, not just any set of ground terms all of which happen to be accepted by $\mathcal{A}$. Two NFTA are said to be ***equivalent*** if they recognize the same tree languages.

Example 5. Let $\mathcal{F} = \{f(,), g(), a\}$. Consider the automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = \{q, q_g, q_f\}$, $Q_f = \{q_f\}$, and $\Delta =$

$$\begin{array}{{}l rcl \quad rcl}
\{ & a & \to & q(a) & g(q(x)) & \to & q(g(x)) \\
 & g(q(x)) & \to & q_g(g(x)) & g(q_g(x)) & \to & q_f(g(x)) \\
 & f(q(x), q(y)) & \to & q(f(x,y)) \quad \}. & & &
\end{array}$$

We now consider a ground term t and exhibit three different reductions of term t *w.r.t.*move relation $\to_{\mathcal{A}}$.

$$\begin{array}{llll}
t = g(g(f(g(a), a))) & \xrightarrow[\mathcal{A}]{*} & g(g(f(q_g(g(a)), q(a)))) & \\
t = g(g(f(g(a), a))) & \xrightarrow[\mathcal{A}]{*} & g(g(q(f(g(a), a)))) & \xrightarrow[\mathcal{A}]{*} \quad q(t) \\
t = g(g(f(g(a), a))) & \xrightarrow[\mathcal{A}]{*} & g(g(q(f(g(a), a)))) & \xrightarrow[\mathcal{A}]{*} \quad q_f(t)
\end{array}$$

The term t is accepted by $\mathcal{A}$ because of the third reduction. It is easy to prove that $L(\mathcal{A}) = \{g(g(t)) \mid t \in T(\mathcal{F})\}$ is the set of ground instances of $g(g(x))$.

The set of transition rules of a NFTA $\mathcal{A}$ can also be defined as a ground rewrite system, *i.e.*a set of ground transition rules of the form: $f(q_1, \ldots, q_n) \to q$. A move relation $\to_{\mathcal{A}}$ can be defined as before. The only difference is that,

now, we "forget" the ground subterms. And, a term t is accepted by a NFTA $\mathcal{A}$ if

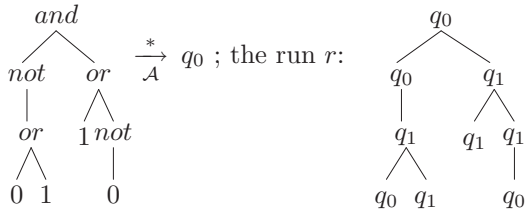
$$t \xrightarrow[\mathcal{A}]{*} q$$

for some final state q in Q_f. Unless it is stated otherwise, we will now refer to the definition with a set of ground transition rules. Considering a reduction starting from a ground term t and leading to a state q with the move relation, it is useful to remember the "history" of the reduction, *i.e.*to remember in which states the ground subterms of t are reduced. For this, we will adopt the following definitions. Let t be a ground term and $\mathcal{A}$ be a NFTA, a ***run*** r of $\mathcal{A}$ on t is a mapping $r : \mathcal{P}os(t) \to Q$ compatible with Δ, *i.e.*for every position p in $\mathcal{P}os(t)$, if $t(p) = f \in \mathcal{F}_n$, $r(p) = q$, $r(pi) = q_i$ for each $i \in \{1, \ldots, n\}$, then $f(q_1, \ldots, q_n) \to q \in \Delta$. A run r of $\mathcal{A}$ on t is ***successful*** if $r(\epsilon)$ is a final state. And a ground term t is accepted by a NFTA $\mathcal{A}$ if there is a successful run r of $\mathcal{A}$ on t.

Example 6. Let $\mathcal{F} = \{or(,), and(,), not(), 0, 1\}$. Consider the automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = \{q_0, q_1\}$, $Q_f = \{q_1\}$, and $\Delta =$

$$\begin{array}{{}l rcl rcl l} \{ & 0 & \to & q_0 & 1 & \to & q_1 \\ & not(q_0) & \to & q_1 & not(q_1) & \to & q_0 \\ & and(q_0, q_0) & \to & q_0 & and(q_0, q_1) & \to & q_0 \\ & and(q_1, q_0) & \to & q_0 & and(q_1, q_1) & \to & q_1 \\ & or(q_0, q_0) & \to & q_0 & or(q_0, q_1) & \to & q_1 \\ & or(q_1, q_0) & \to & q_1 & or(q_1, q_1) & \to & q_1 \quad \}. \end{array}$$

A ground term over $\mathcal{F}$ can be viewed as a boolean formula without variable and a run on such a ground term can be viewed as the evaluation of the corresponding boolean formula. For instance, we give a reduction for a ground term t and the corresponding run given as a tree



The tree language recognized by $\mathcal{A}$ is the set of true boolean expressions over $\mathcal{F}$.

NFTA with ϵ-rules

Like in the word case, it is convenient to allow ϵ-moves in the reduction of a ground term by an automaton, *i.e.*the current state is changed but no new symbol of the term is processed. This is done by introducing a new type of rules in the set of transition rules of an automaton. A ***NFTA with ϵ-rules*** is like a NFTA except that now the set of transition rules contains ground transition

rules of the form $f(q_1, \ldots, q_n) \to q$, and ϵ-rules of the form $q \to q'$. The ability to make ϵ-moves does not allow the NFTA to accept non recognizable sets. But NFTA with ϵ-rules are useful in some constructions and simplify some proofs.

Example 7. Let $\mathcal{F} = \{cons(,), s(), 0, nil\}$. Consider the automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = \{q_{\mathsf{Nat}}, q_{\mathsf{List}}, q_{\mathsf{List}*}\}$, $Q_f = \{q_{\mathsf{List}}\}$, and $\Delta =$

$$\left\{\begin{array}{rcl@{\qquad}rcl} 0 & \to & q_{\mathsf{Nat}} & s(q_{\mathsf{Nat}}) & \to & q_{\mathsf{Nat}} \\ nil & \to & q_{\mathsf{List}} & cons(q_{\mathsf{Nat}}, q_{\mathsf{List}}) & \to & q_{\mathsf{List}*} \\ q_{\mathsf{List}*} & \to & q_{\mathsf{List}}\}. & & & \end{array}\right.$$

The recognized tree language is the set of Lisp-like lists of integers. If the final state set Q_f is set to $\{q_{\mathsf{List}*}\}$, then the recognized tree language is the set of non empty Lisp-like lists of integers. The ϵ-rule $q_{\mathsf{List}*} \to q_{\mathsf{List}}$ says that a non empty list is a list. The reader should recognize the definition of an order-sorted algebra with the sorts Nat, List, and $\mathsf{List}*$ (which stands for the non empty lists), and the inclusion $\mathsf{List}* \subseteq \mathsf{List}$ (see Section 3.4.1).

Theorem 1 (The equivalence of NFTAs with and without ϵ-rules). *If L is recognized by a NFTA with ϵ-rules, then L is recognized by a NFTA without ϵ-rules.*

Proof. Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a NFTA with ϵ-rules. Consider the subset Δ_ϵ consisting of those ϵ-rules in Δ. We denote by *ϵ-closure(q)* the set of all states q' in Q such that there is a reduction of q into q' using rules in Δ_ϵ. We consider that $q \in$ *ϵ-closure(q)*. This computation is a transitive closure computation and can be done in $O(|Q|^3)$. Now let us define the NFTA $\mathcal{A}' = (Q, \mathcal{F}, Q_f, \Delta')$ where Δ' is defined by:

$$\Delta' = \{f(q_1, \ldots, q_n) \to q' \mid f(q_1, \ldots, q_n) \to q \in \Delta, q' \in \textit{ϵ-closure(q)}\}$$

Then it may be proved that $t \xrightarrow[\mathcal{A}]{*} q$ iff $t \xrightarrow[\mathcal{A}']{*} q$. □

Unless it is stated otherwise, we will now consider NFTA without ϵ-rules.

Deterministic Finite Tree Automata

Our definition of tree automata corresponds to the notion of nondeterministic finite tree automata. We will now define deterministic tree automata (DFTA) which are a special case of NFTA. It will turn out that, like in the word case, any language recognized by a NFTA can also be recognized by a DFTA. However, the NFTA are useful in proving theorems in tree language theory.

A tree automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ is ***deterministic*** (DFTA) if there are no two rules with the same left-hand side (and no ϵ-rule). Given a DFTA, there is at most one run for every ground term, *i.e.*for every ground term t, there is at most one state q such that $t \xrightarrow[\mathcal{A}]{*} q$. The reader should note that it is possible to define a tree automaton in which there are two rules with the same left-hand side such that there is at most one run for every ground term (see Example 8).

It is also useful to consider tree automata such that there is at least one run for every ground term. This leads to the following definition. A NFTA $\mathcal{A}$ is ***complete*** if there is at least one rule $f(q_1, \ldots, q_n) \to q \in \Delta$ for all $n \geq 0$, $f \in \mathcal{F}_n$, and $q_1, \ldots, q_n \in Q$. Let us note that for a complete DFTA there is exactly one run for every ground term.

Lastly, for practical reasons, it is usual to consider automata in which unnecessary states are eliminated. A state q is ***accessible*** if there exists a ground term t such that $t \xrightarrow[\mathcal{A}]{*} q$. A NFTA $\mathcal{A}$ is said to be ***reduced*** if all its states are accessible.

Example 8.

The automaton defined in Example 5 is reduced, not complete, and it is not deterministic because there are two rules of left-hand side $g(q(x))$. Let us also note (see Example 5) that at least two runs (one is successful) can be defined on the term $g(g(f(g(a), a)))$.

The automaton defined in Example 6 is a complete and reduced DFTA.

Let $\mathcal{F} = \{g(), a\}$. Consider the automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = \{q_0, q_1, q\}$, $Q_f = \{q_0\}$, and Δ is the following set of transition rules:

$$\begin{array}{llllll} \{ & a & \to & q_0 & g(q_0) & \to & q_1 \\ & g(q_1) & \to & q_0 & g(q) & \to & q_0 \\ & g(q) & \to & q_1 \}. & & & \end{array}$$

This automaton is not deterministic because there are two rules of left-hand side $g(q)$, it is not reduced because state q is not accessible. Nevertheless, one should note that there is at most one run for every ground term t.

Let $\mathcal{F} = \{f(,), g(), a\}$. Consider the automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined in Example 4 by: $Q = \{q_a, q_g, q_f\}$, $Q_f = \{q_f\}$, and Δ is the following set of transition rules:

$$\begin{array}{llllll} \{ & a & \to & q_a & g(q_a) & \to & q_g \\ & g(q_g) & \to & q_g & f(q_g, q_g) & \to & q_f \quad \}. \end{array}$$

This automaton is deterministic and reduced. It is not complete because, for instance, there is no transition rule of left-hand side $f(q_a, q_a)$. It is easy to define a deterministic and complete automaton $\mathcal{A}'$ recognizing the same language by adding a "dead state". The automaton $\mathcal{A}' = (Q', \mathcal{F}, Q_f, \Delta')$ is defined by: $Q' = Q \cup \{\pi\}$, $\Delta' = \Delta \cup$

$$\begin{array}{llllll} \{ & g(q_f) & \to & \pi & g(\pi) & \to & \pi \\ & f(q_a, q_a) & \to & \pi & f(q_a, q_g) & \to & \pi \\ & & \ldots & & f(\pi, \pi) & \to & \pi \quad \}. \end{array}$$

It is easy to generalize the construction given in Example 8 of a complete NFTA equivalent to a given NFTA: add a "dead state" π and all transition rules with right-hand side π such that the automaton is complete. The reader should note that this construction could be expensive because it may require $O(|\mathcal{F}| \times |Q|^{Arity(\mathcal{F})})$ new rules where $Arity(\mathcal{F})$ is the maximal arity of symbols in $\mathcal{F}$. Therefore we have the following:

Theorem 2. *Let L be a recognizable set of ground terms. Then there exists a complete finite tree automaton that accepts L.*

We now give a polynomial algorithm which outputs a reduced NFTA equivalent to a given NFTA as input. The main loop of this algorithm computes the set of accessible states.

Reduction Algorithm RED
input: NFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$
begin
Set *Marked* to $\emptyset$ /* *Marked* is the set of accessible states */
repeat
Set *Marked* to *Marked* $\cup \{q\}$
where
$f \in \mathcal{F}_n$, $q_1, \ldots, q_n \in$ *Marked*, $f(q_1, \ldots, q_n) \rightarrow q \in \Delta$
until no state can be added to *Marked*
Set Q_r to *Marked*
Set Q_{r_f} to $Q_f \cap$ *Marked*
Set Δ_r to $\{f(q_1, \ldots, q_n) \rightarrow q \in \Delta \mid q, q_1, \ldots, q_n \in \textit{Marked}\}$
output: NFTA $\mathcal{A}_r = (Q_r, \mathcal{F}, Q_{r_f}, \Delta_r)$
end

Obviously all states in the set *Marked* are accessible, and an easy induction shows that all accessible states are in the set *Marked*. And, the NFTA $\mathcal{A}_r$ accepts the tree language $L(\mathcal{A})$. Consequently we have:

Theorem 3. *Let L be a recognizable set of ground terms. Then there exists a reduced finite tree automaton that accepts L.*

Now, we consider the reduction of nondeterminism. Since every DFTA is a NFTA, it is clear that the class of recognizable languages includes the class of languages accepted by DFTAs. However it turns out that these classes are equal. We prove that, for every NFTA, we can construct an equivalent DFTA. The proof is similar to the proof of equivalence between DFAs and NFAs in the word case. The proof is based on the "subset construction". Consequently, the number of states of the equivalent DFTA can be exponential in the number of states of the given NFTA (see Example 10). But, in practice, it often turns out that many states are not accessible. Therefore, we will present in the proof of the following theorem a construction of a DFTA where only the accessible states are considered, *i.e.*the given algorithm outputs an equivalent and reduced DFTA from a given NFTA as input.

Theorem 4 (The equivalence of DFTAs and NFTAs). *Let L be a recognizable set of ground terms. Then there exists a DFTA that accepts L.*

Proof. First, we give a theoretical construction of a DFTA equivalent to a NFTA. Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a NFTA. Define a DFTA $\mathcal{A}_d = (Q_d, \mathcal{F}, Q_{df}, \Delta_d)$, as follows. The states of Q_d are all the subsets of the state set Q of $\mathcal{A}$. That is, $Q_d = 2^Q$. We denote by s a state of Q_d, *i.e.*$s = \{q_1, \ldots, q_n\}$ for some states

$q_1, \ldots, q_n \in Q$. We define

$$f(s_1, \ldots, s_n) \to s \in \Delta_d$$
$$\text{iff}$$
$$s = \{q \in Q \mid \exists q_1 \in s_1, \ldots, \exists q_n \in s_n, f(q_1, \ldots, q_n) \to q \in \Delta\}.$$

And Q_{df} is the set of all states in Q_d containing a final state of $\mathcal{A}$.

We now give an algorithmic construction where only the accessible states are considered.

Determinization Algorithm DET
input: NFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$
begin
/* A state s of the equivalent DFTA is in 2^Q */
Set Q_d to $\emptyset$; set Δ_d to $\emptyset$
repeat
Set Q_d to $Q_d \cup \{s\}$; Set Δ_d to $\Delta_d \cup \{f(s_1, \ldots, s_n) \to s\}$
where
$f \in \mathcal{F}_n$, $s_1, \ldots, s_n \in Q_d$,
$s = \{q \in Q \mid \exists q_1 \in s_1, \ldots, q_n \in s_n, f(q_1, \ldots, q_n) \to q \in \Delta\}$
until no rule can be added to Δ_d
Set Q_{df} to $\{s \in Q_d \mid s \cap Q_f \neq \emptyset\}$
output: DFTA $\mathcal{A}_d = (Q_d, \mathcal{F}, Q_{df}, \Delta_d)$
end

It is immediate from the definition of the determinization algorithm that $\mathcal{A}_d$ is a deterministic and reduced tree automaton. In order to prove that $L(\mathcal{A}) = L(\mathcal{A}_d)$, we now prove that:

$$(t \xrightarrow[\mathcal{A}_d]{*} s) \text{ iff } (s = \{q \in Q \mid t \xrightarrow[\mathcal{A}]{*} q\}).$$

The proof is an easy induction on the structure of terms.

- base case: let us consider $t = a \in \mathcal{F}_0$. Then, there is only one rule $a \to s$ in Δ_d where $s = \{q \in Q \mid a \to q \in \Delta\}$.
- induction step: let us consider a term $t = f(t_1, \ldots, t_n)$.
  - First, let us suppose that $t \xrightarrow[\mathcal{A}_d]{*} f(s_1, \ldots, s_n) \to_{\mathcal{A}_d} s$. By induction hypothesis, for each $i \in \{1, \ldots, n\}$, $s_i = \{q \in Q \mid t_i \xrightarrow[\mathcal{A}]{*} q\}$. States s_i are in Q_d, thus a rule $f(s_1, \ldots, s_n) \to s \in \Delta_d$ is added in the set Δ_d by the determinization algorithm and $s = \{q \in Q \mid \exists q_1 \in s_1, \ldots, q_n \in s_n, f(q_1, \ldots, q_n) \to q \in \Delta\}$. Thus, $s = \{q \in Q \mid t \xrightarrow[\mathcal{A}]{*} q\}$.
  - Second, let us consider $s = \{q \in Q \mid t = f(t_1, \ldots, t_n) \xrightarrow[\mathcal{A}]{*} q\}$. Let us consider the state sets s_i defined by $s_i = \{q \in Q \mid t_i \xrightarrow[\mathcal{A}]{*} q\}$. By induction hypothesis, for each $i \in \{1, \ldots, n\}$, $t_i \xrightarrow[\mathcal{A}_d]{*} s_i$. Thus

$s = \{q \in Q \mid \exists q_1 \in s_1, \ldots, q_n \in s_n, f(q_1, \ldots, q_n) \to q \in \Delta\}$. The rule $f(s_1, \ldots, s_n) \in \Delta_d$ by definition of the state set Δ_d in the determinization algorithm and $t \xrightarrow[\mathcal{A}_d]{*} s$.

□

Example 9. Let $\mathcal{F} = \{f(,), g(), a\}$. Consider the automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined in Example 5 by: $Q = \{q, q_g, q_f\}$, $Q_f = \{q_f\}$, and $\Delta =$

$$\begin{array}{llllll} \{ & a & \to & q & g(q) & \to & q \\ & g(q) & \to & q_g & g(q_g) & \to & q_f \\ & f(q,q) & \to & q \quad \}. & & & \end{array}$$

Given $\mathcal{A}$ as input, the determinization algorithm outputs the DFTA $\mathcal{A}_d = (Q_d, \mathcal{F}, Q_{df}, \Delta_d)$ defined by: $Q_d = \{\{q\}, \{q, q_g\}, \{q, q_g, q_f\}\}$, $Q_{df} = \{\{q, q_g, q_f\}\}$, and $\Delta_d =$

$$\begin{array}{llllll} & \{ & g(\{q\}) & \to & \{q, q_g\} & \\ & & g(\{q, q_g\}) & \to & \{q, q_g, q_f\} & \\ & & g(\{q, q_g, q_f\}) & \to & \{q, q_g, q_f\} & \} \\ \cup & \{ & f(s_1, s_2) & \to & \{q\} & \mid \quad s_1, s_2 \in Q_d \quad \}. \end{array}$$

We now give an example where an exponential blow-up occurs in the determinization process. This example is the same used in the word case.

Example 10. Let $\mathcal{F} = \{f(), g(), a\}$ and let n be an integer. And let us consider the tree language

$$L = \{t \in T(\mathcal{F}) \mid \text{the symbol at position } \underbrace{1 \ldots 1}_{n} \text{ is } f\}.$$

Let us consider the NFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = \{q, q_1, \ldots, q_{n+1}\}$, $Q_f = \{q_{n+1}\}$, and $\Delta =$

$$\begin{array}{llllll} \{ & a & \to & q & f(q) & \to & q \\ & g(q) & \to & q & f(q) & \to & q_1 \\ & g(q_1) & \to & q_2 & f(q_1) & \to & q_2 \\ & & \vdots & & & \vdots & \\ & g(q_n) & \to & q_{n+1} & f(q_n) & \to & q_{n+1} \quad \}. \end{array}$$

The NFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ accepts the tree language L, and it has $n + 2$ states. Using the subset construction, the equivalent DFTA $\mathcal{A}_d$ has 2^{n+1} states. Any equivalent automaton has to memorize the $n + 1$ last symbols of the input tree. Therefore, it can be proved that any DFTA accepting L has at least 2^{n+1} states. It could also be proved that the automaton $\mathcal{A}_d$ is minimal in the number of states (minimal tree automata are defined in Section 1.5).

If a finite tree automaton is deterministic, we can replace the transition relation Δ by a transition function δ. Therefore, it is sometimes convenient to consider a DFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \delta)$ where

$$\delta : \bigcup_n \mathcal{F}_n \times Q^n \to Q \ .$$

The computation of such an automaton on a term t as input tree can be viewed as an evaluation of t on finite domain Q. Indeed, define the labeling function $\hat{\delta} : T(\mathcal{F}) \to Q$ inductively by

$$\hat{\delta}(f(t_1, \ldots, t_n)) = \delta(f, \hat{\delta}(t_1), \ldots, \hat{\delta}(t_n)) \ .$$

We shall for convenience confuse δ and $\hat{\delta}$.

We now make clear the connections between our definitions and the language theoretical definitions of tree automata and of recognizable tree languages. Indeed, the reader should note that a complete DFTA is just a finite $\mathcal{F}$-algebra $\mathcal{A}$ consisting of a finite carrier $|\mathcal{A}| = Q$ and a distinguished n-ary function $f^{\mathcal{A}} : Q^n \to Q$ for each n-ary symbol $f \in \mathcal{F}$ together with a specified subset Q_f of Q. A ground term t is accepted by $\mathcal{A}$ if $\delta(t) = q \in Q_f$ where δ is the unique $\mathcal{F}$-algebra homomorphism $\delta : T(\mathcal{F}) \to \mathcal{A}$.

Example 11. Let $\mathcal{F} = \{f(,), a\}$ and consider the $\mathcal{F}$-algebra $\mathcal{A}$ with $|\mathcal{A}| = Q = Z_2 = \{0, 1\}$, $f^{\mathcal{A}} = +$ where the sum is formed modulo 2, $a^{\mathcal{A}} = 1$, and let $Q_f = \{0\}$. $\mathcal{A}$ and Q_f defines a DFTA. The recognized tree language is the set of ground terms over $\mathcal{F}$ with an even number of leaves.

Since DFTA and NFTA accept the same sets of tree languages, we shall not distinguish between them unless it becomes necessary, but shall simply refer to both as tree automata (FTA).

1.2 The Pumping Lemma for Recognizable Tree Languages

We now give an example of a tree language which is not recognizable.

Example 12. Let $\mathcal{F} = \{f(,), g(), a\}$. Let us consider the tree language $L = \{f(g^i(a), g^i(a)) \mid i > 0\}$. Let us suppose that L is recognizable by an automaton $\mathcal{A}$ having k states. Now, consider the term $t = f(g^k(a), g^k(a))$. t belongs to L, therefore there is a successful run of $\mathcal{A}$ on t. As k is the cardinality of the state set, there are two distinct positions along the first branch of the term labeled with the same state. Therefore, one could cut the first branch between these two positions leading to a term $t' = f(g^j(a), g^k(a))$ with $j < k$ such that a successful run of $\mathcal{A}$ can be defined on t'. This leads to a contradiction with $L(\mathcal{A}) = L$.

This (sketch of) proof can be generalized by proving a ***pumping lemma*** for recognizable tree languages. This lemma is extremely useful in proving that

certain sets of ground terms are not recognizable. It is also useful for solving decision problems like emptiness and finiteness of a recognizable tree language (see Section 1.7).

Pumping Lemma. *Let L be a recognizable set of ground terms. Then, there exists a constant $k > 0$ satisfying: for every ground term t in L such that $\mathcal{H}eight(t) > k$, there exist a context $C \in \mathcal{C}(\mathcal{F})$, a non trivial context $C' \in \mathcal{C}(\mathcal{F})$, and a ground term u such that $t = C[C'[u]]$ and, for all $n \geq 0$ $C[C'^n[u]] \in L$.*

Proof. Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a FTA such that $L = L(\mathcal{A})$ and let $k = |Q|$ be the cardinality of the state set Q. Let us consider a ground term t in L such that $\mathcal{H}eight(t) > k$ and consider a successful run r of $\mathcal{A}$ on t. Now let us consider a path in t of length strictly greater than k. As k is defined to be the cardinality of the state set Q, there are two positions $p_1 < p_2$ along this path such that $r(p_1) = r(p_2) = q$ for some state q. Let u be the ground subterm of t at position p_2. Let u' be the ground subterm of t at position p_1, there exists a non-trivial context C' such that $u' = C'[u]$. Now define the context C such that $t = C[C'[u]]$. Consider a term $C[C'^n[u]]$ for some integer $n > 1$, a successful run can be defined on this term. Indeed suppose that r corresponds to the reduction $t \xrightarrow[\mathcal{A}]{*} q_f$ where q_f is a final state of $\mathcal{A}$, then we have:

$$C[C'^n[u]] \xrightarrow[\mathcal{A}]{*} C[C'^n[q]] \xrightarrow[\mathcal{A}]{*} C[C'^{n-1}[q]] \dots \xrightarrow[\mathcal{A}]{*} C[q] \xrightarrow[\mathcal{A}]{*} q_f.$$

The same holds when $n = 0$. □

Example 13. Let $\mathcal{F} = \{f(,), a\}$. Let us consider the tree language $L = \{t \in T(\mathcal{F}) \mid |\mathcal{P}os(t)| \text{ is a prime number}\}$. We can prove that L is not recognizable. For all $k > 0$, consider a term t in L whose height is greater than k. For all contexts C, non trivial contexts C', and terms u such that $t = C[C'[u]]$, there exists n such that $C[C'^n[u]] \notin L$.

From the Pumping Lemma, we derive conditions for emptiness and finiteness given by the following corollary:

Corollary 1. *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a FTA. Then $L(\mathcal{A})$ is non empty if and only if there exists a term t in $L(\mathcal{A})$ with $\mathcal{H}eight(t) \leq |Q|$. Then $L(\mathcal{A})$ is infinite if and only if there exists a term t in $L(\mathcal{A})$ with $|Q| < \mathcal{H}eight(t) \leq 2|Q|$.*

1.3 Closure Properties of Recognizable Tree Languages

A ***closure property*** of a class of (tree) languages is the fact that the class is closed under a particular operation. We are interested in effective closure properties where, given representations for languages in the class, there is an algorithm to construct a representation for the language that results by applying the operation to these languages. Let us note that the equivalence between NFTA and DFTA is effective, thus we may choose the representation that suits

us best. Nevertheless, the determinization algorithm may output a DFTA whose number of states is exponential in the number of states of the given NFTA. For the different closure properties, we give effective constructions and we give the properties of the resulting FTA depending on the properties of the given FTA as input. In this section, we consider the Boolean set operations: union, intersection, and complementation. Other operations will be studied in the next sections. Complexity results are given in Section 1.7.

Theorem 5. *The class of recognizable tree languages is closed under union, under complementation, and under intersection.*

Union

Let L_1 and L_2 be two recognizable tree languages. Thus there are tree automata $\mathcal{A}_1 = (Q_1, \mathcal{F}, Q_{f1}, \Delta_1)$ and $\mathcal{A}_2 = (Q_2, \mathcal{F}, Q_{f2}, \Delta_2)$ with $L_1 = L(\mathcal{A}_1)$ and $L_2 = L(\mathcal{A}_2)$. Since we may rename states of a tree automaton, without loss of generality, we may suppose that $Q_1 \cap Q_2 = \emptyset$. Now, let us consider the FTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = Q_1 \cup Q_2$, $Q_f = Q_{f1} \cup Q_{f2}$, and $\Delta = \Delta_1 \cup \Delta_2$. The equality between $L(\mathcal{A})$ and $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ is straightforward. Let us note that $\mathcal{A}$ is nondeterministic and not complete, even if $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic and complete.

We now give another construction which preserves determinism. The intuitive idea is to process in parallel a term by the two automata. For this we consider a product automaton. Let us suppose that $\mathcal{A}_1$ and $\mathcal{A}_2$ are complete. And, let us consider the FTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = Q_1 \times Q_2$, $Q_f = Q_{f1} \times Q_2 \cup Q_1 \times Q_{f2}$, and $\Delta = \Delta_1 \times \Delta_2$ where

$$\Delta_1 \times \Delta_2 = \{f((q_1, q'_1), \ldots, (q_n, q'_n)) \to (q, q') \mid \\ f(q_1, \ldots, q_n) \to q \in \Delta_1 \; f(q'_1, \ldots, q'_n) \to q' \in \Delta_2\}$$

The proof of the equality between $L(\mathcal{A})$ and $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ is left to the reader, but the reader should note that the hypothesis that the two given tree automata are complete is crucial in the proof. Indeed, suppose for instance that a ground term t is accepted by $\mathcal{A}_1$ but not by $\mathcal{A}_2$. Moreover suppose that $\mathcal{A}_2$ is not complete and that there is no run of $\mathcal{A}_2$ on t, then the product automaton does not accept t because there is no run of the product automaton on t. The reader should also note that the construction preserves determinism, *i.e.*if the two given automata are deterministic, then the product automaton is also deterministic.

Complementation

Let L be a recognizable tree language. Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a complete DFTA such that $L(\mathcal{A}) = L$. Now, complement the final state set to recognize the complement of L. That is, let $\mathcal{A}^c = (Q, \mathcal{F}, Q_f^c, \Delta)$ with $Q_f^c = Q \setminus Q_f$, the DFTA $\mathcal{A}^c$ recognizes the complement of set L in $T(\mathcal{F})$.

If the input automaton $\mathcal{A}$ is a NFTA, then first apply the determinization algorithm, and second complement the final state set. This could lead to an exponential blow-up.

Intersection

Closure under intersection follows from closure under union and complementation because

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}.$$

where we denote by $\overline{L}$ the complement of set L in $T(\mathcal{F})$. But if the recognizable tree languages are defined by NFTA, we have to use the complementation construction, therefore the determinization process is used leading to an exponential blow-up. Consequently, we now give a direct construction which does not use the determinization algorithm. Let $\mathcal{A}_1 = (Q_1, \mathcal{F}, Q_{f1}, \Delta_1)$ and $\mathcal{A}_2 = (Q_2, \mathcal{F}, Q_{f2}, \Delta_2)$ be FTA such that $L(\mathcal{A}_1) = L_1$ and $L(\mathcal{A}_2) = L_2$. And, consider the FTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = Q_1 \times Q_2$, $Q_f = Q_{f1} \times Q_{f2}$, and $\Delta = \Delta_1 \times \Delta_2$. $\mathcal{A}$ recognizes $L_1 \cap L_2$. Moreover the reader should note that $\mathcal{A}$ is deterministic if $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic.

1.4 Tree Homomorphisms

We now consider tree transformations and study the closure properties under these tree transformations. In this section we are interested with tree transformations preserving the structure of trees. Thus, we restrict ourselves to tree homomorphisms. Tree homomorphisms are a generalization of homomorphisms for words (considered as unary terms) to the case of arbitrary ranked alphabets. In the word case, it is known that the class of regular sets is closed under homomorphisms and inverse homomorphisms. The situation is different in the tree case because whereas recognizable tree languages are closed under inverse homomorphisms, they are closed only under a subclass of homomorphisms, *i.e.*linear homomorphisms (duplication of terms is forbidden). First, we define tree homomorphisms.

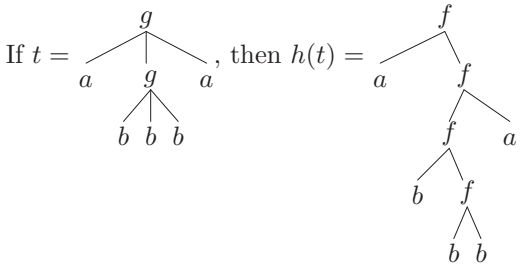
Let $\mathcal{F}$ and $\mathcal{F}'$ be two sets of function symbols, possibly not disjoint. For each $n > 0$ such that $\mathcal{F}$ contains a symbol of arity n, we define a set of variables $\mathcal{X}_n = \{x_1, \ldots, x_n\}$ disjoint from $\mathcal{F}$ and $\mathcal{F}'$.

Let $h_{\mathcal{F}}$ be a mapping which, with $f \in \mathcal{F}$ of arity n, associates a term $t_f \in T(\mathcal{F}', \mathcal{X}_n)$. The ***tree homomorphism*** $h : T(\mathcal{F}) \to T(\mathcal{F}')$ determined by $h_{\mathcal{F}}$ is defined as follows:

- $h(a) = t_a \in T(\mathcal{F}')$ for each $a \in \mathcal{F}$ of arity 0,
- $h(f(t_1, \ldots, t_n)) = t_f\{x_1 \leftarrow h(t_1), \ldots, x_n \leftarrow h(t_n)\}$

where $t_f\{x_1 \leftarrow h(t_1), \ldots, x_n \leftarrow h(t_n)\}$ is the result of applying the substitution $\{x_1 \leftarrow h(t_1), \ldots, x_n \leftarrow h(t_n)\}$ to the term t_f.

Example 14. Let $\mathcal{F} = \{g(,,), a, b\}$ and $\mathcal{F}' = \{f(,), a, b\}$. Let us consider the tree homomorphism h determined by $h_{\mathcal{F}}$ defined by: $h_{\mathcal{F}}(g) = f(x_1, f(x_2, x_3))$, $h_{\mathcal{F}}(a) = a$ and $h_{\mathcal{F}}(b) = b$. For instance, we have:



The homomorphism h defines a transformation from ternary trees into binary trees.

Let us now consider $\mathcal{F} = \{and(,), or(,), not(), 0, 1\}$ and $\mathcal{F}' = \{or(,), not(), 0, 1\}$. Let us consider the tree homomorphism h determined by $h_{\mathcal{F}}$ defined by: $h_{\mathcal{F}}(and) = not(or(not(x_1), not(x_2))$, and $h_{\mathcal{F}}$ is the identity otherwise. This homomorphism transforms a boolean formula in an equivalent boolean formula which does not contain the function symbol *and*.

A tree homomorphism is ***linear*** if for each $f \in \mathcal{F}$ of arity n, $h_{\mathcal{F}}(f) = t_f$ is a linear term in $T(\mathcal{F}', \mathcal{X}_n)$. The following example shows that tree homomorphisms do not always preserve recognizability.

Example 15. Let $\mathcal{F} = \{f(), g(), a\}$ and $\mathcal{F}' = \{f'(,), g(), a\}$. Let us consider the tree homomorphism h determined by $h_{\mathcal{F}}$ defined by: $h_{\mathcal{F}}(f) = f'(x_1, x_1)$, $h_{\mathcal{F}}(g) = g(x_1)$, and $h_{\mathcal{F}}(a) = a$. h is not linear. Let $L = \{f(g^i(a)) \mid i \geq 0\}$, then L is a recognizable tree language. $h(L) = \{f'(g^i(a), g^i(a)) \mid i \geq 0\}$ is not recognizable (see Example 12).

Theorem 6 (Linear homomorphisms preserve recognizability). *Let h be a linear tree homomorphism and L be a recognizable tree language, then $h(L)$ is a recognizable tree language.*

Proof. Let L be a recognizable tree language. Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a reduced DFTA such that $L(\mathcal{A}) = L$. Let h be a linear tree homomorphism from $T(\mathcal{F})$ into $T(\mathcal{F}')$ determined by a mapping $h_{\mathcal{F}}$.

First, let us define a NFTA $\mathcal{A}' = (Q', \mathcal{F}', Q'_f, \Delta')$. Let us consider a rule $r = f(q_1, \ldots, q_n) \to q$ in Δ and consider the linear term $t_f = h_{\mathcal{F}}(f) \in T(\mathcal{F}', \mathcal{X}_n)$ and the set of positions $\mathcal{P}os(t_f)$. We define a set of states $Q^r = \{q^r_p \mid p \in \mathcal{P}os(t_f)\}$, and we define a set of rules Δ_r as follows: for all positions p in $\mathcal{P}os(t_f)$

- if $t_f(p) = g \in \mathcal{F}'_k$, then $g(q^r_{p_1}, \ldots, q^r_{p_k}) \to q^r_p \in \Delta_r$,
- if $t_f(p) = x_i$, then $q_i \to q^r_p \in \Delta_r$,
- $q^r_\epsilon \to q \in \Delta_r$.

The preceding construction is made for each rule in Δ. We suppose that all the state sets Q^r are disjoint and that they are disjoint from Q. Now define $\mathcal{A}'$ by:

- $Q' = Q \cup \bigcup_{r \in \Delta} Q^r$,

- $Q'_f = Q_f$,
- $\Delta' = \bigcup_{r \in \Delta} \Delta_r$.

Second, we have to prove that $h(L) = L(\mathcal{A}')$.

$h(L) \subseteq L(\mathcal{A}')$. We prove that if $t \xrightarrow[\mathcal{A}]{*} q$ then $h(t) \xrightarrow[\mathcal{A}']{*} q$ by induction on the length of the reduction of ground term $t \in T(\mathcal{F})$ by automaton $\mathcal{A}$.

- Base case. Suppose that $t \to_{\mathcal{A}} q$. Then $t = a \in \mathcal{F}_0$ and $a \to q \in \Delta$. Then there is a reduction $h(a) = t_a \xrightarrow[\mathcal{A}']{*} q$ using the rules in the set $\Delta_{a \to q}$.
- Induction step.
  Suppose that $t = f(u_1, \ldots, u_n)$, then $h(t) = t_f\{x_1 \leftarrow h(u_1), \ldots, x_n \leftarrow h(u_n)\}$. Moreover suppose that $t \xrightarrow[\mathcal{A}]{*} f(q_1, \ldots, q_n) \to_{\mathcal{A}} q$. By induction hypothesis, we have $h(u_i) \xrightarrow[\mathcal{A}']{*} q_i$, for each i in $\{1, \ldots, n\}$. Then there is a reduction $t_f\{x_1 \leftarrow q_1, \ldots, x_n \leftarrow q_n\} \xrightarrow[\mathcal{A}']{*} q$ using the rules in the set $\Delta_{f(q_1,\ldots,q_n) \to q}$.

$h(L) \supseteq L(\mathcal{A}')$**.** We prove that if $t' \xrightarrow[\mathcal{A}']{*} q \in Q$ then $t' = h(t)$ with $t \xrightarrow[\mathcal{A}]{*} q$ for some $t \in T(\mathcal{F})$. The proof is by induction on the number of states in Q occurring along the reduction $t' \xrightarrow[\mathcal{A}']{*} q \in Q$.

- Base case. Suppose that $t' \xrightarrow[\mathcal{A}']{*} q \in Q$ and no state in Q apart from q occurs in the reduction. Then, because the state sets Q^r are disjoint, only rules of some Δ^r can be used in the reduction. Thus, t' is ground, $t' = h_{\mathcal{F}}(f)$ for some symbol $f \in \mathcal{F}$, and $r = f(q_1, \ldots, q_n) \to q$. Because the automaton is reduced, there is some ground term t with $\mathcal{H}ead(t) = f$ such that $t' = h(t)$ and $t \xrightarrow[\mathcal{A}]{*} q$.
- Induction step. Suppose that

$$t' \xrightarrow[\mathcal{A}']{*} v\{x'_1 \leftarrow q_1, \ldots, x'_m \leftarrow q_m\} \xrightarrow[\mathcal{A}']{*} q$$

  where v is a linear term in $T(\mathcal{F}', \{x'_1, \ldots, x'_m\})$, $t' = v\{x'_1 \leftarrow u'_1, \ldots, x'_m \leftarrow u'_m\}$, $u'_i \xrightarrow[\mathcal{A}']{*} q_i \in Q$, and no state in Q apart from q occurs in the reduction of $v\{x'_1 \leftarrow q_1, \ldots, x'_m \leftarrow q_m\}$ to q. The reader should note that different variables can be substituted by the same state. Then, because the state sets Q^r are disjoint, only rules of some Δ^r can be used in the reduction of $v\{x'_1 \leftarrow q_1, \ldots, x'_m \leftarrow q_m\}$ to q. Thus, there exists some linear term t_f such that $v\{x'_1 \leftarrow q_1, \ldots, x'_m \leftarrow q_m\} = t_f\{x_1 \leftarrow q_1, \ldots, x_n \leftarrow q_n\}$ for some symbol $f \in \mathcal{F}_n$ and $r = f(q_1, \ldots, q_n) \to q \in \Delta$. By induction hypothesis, there are terms $u_1, \ldots, u_m$ in L such that $u'_i = h(u_i)$ and $u_i \xrightarrow[\mathcal{A}]{*} q_i$ for each i in $\{1, \ldots, m\}$. Now consider the term $t = f(v_1, \ldots, v_n)$, where $v_i = u_i$ if x_i occurs in t_f and v_i is some term such that $v_i \xrightarrow[\mathcal{A}]{*} q_i$ otherwise (terms v_i always exist because $\mathcal{A}$ is reduced). We have

$h(t) = t_f\{x_1 \leftarrow h(v_1), \ldots, x_n \leftarrow h(v_n)\}$, $h(t) = v\{x'_1 \leftarrow h(u_1), \ldots, x'_m \leftarrow h(u_m)\}$, $h(t) = t'$. Moreover, by definition of the v_i and by induction hypothesis, we have $t \xrightarrow[\mathcal{A}]{*} q$. Note that if q_i occurs more than once, you can substitute q_i by any term satisfying the conditions. The proof does not work for the non linear case because you have to check that different occurrences of some state q_i corresponding to the same variable $x_j \in \mathcal{V}ar(t_f)$ can only be substituted by equal terms.

□

Only linear tree homomorphisms preserve recognizability. An example of a non linear homomorphism which transforms recognizable tree languages either in recognizable tree languages or in non recognizable tree languages is given in Exercise 6. For linear and non linear homomorphisms, we have:

Theorem 7 (Inverse homomorphisms preserve recognizability). *Let h be a tree homomorphism and L be a recognizable tree language, then $h^{-1}(L)$ is a recognizable tree language.*

Proof. Let h be a tree homomorphism from $T(\mathcal{F})$ into $T(\mathcal{F}')$ determined by a mapping $h_{\mathcal{F}}$. Let $\mathcal{A}' = (Q', \mathcal{F}', Q'_f, \Delta')$ be a complete DFTA such that $L(\mathcal{A}') = L$. We define a DFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ by $Q = Q' \cup \{s\}$ where $s \notin Q'$, $Q_f = Q'_f$ and Δ is defined by the following:

- for $a \in \mathcal{F}_0$, if $t_a \xrightarrow[\mathcal{A}']{*} q$ then $a \to q \in \Delta$;
- for $f \in \mathcal{F}_n$ where $n > 0$, if $t_f\{x_1 \leftarrow p_1, \ldots, x_n \leftarrow p_n\} \xrightarrow[\mathcal{A}']{*} q$ then $f(q_1, \ldots, q_n) \to q \in \Delta$ where $q_i = p_i$ if x_i occurs in t_f and $q_i = s$ otherwise;
- for $a \in \mathcal{F}_0$, $a \to s \in \Delta$;
- for $f \in \mathcal{F}_n$ where $n > 0$, $f(s, \ldots, s) \to s \in \Delta$.

The rule set Δ is computable. The proof of the equivalence $t \xrightarrow[\mathcal{A}]{*} q$ if and only if $h(t) \xrightarrow[\mathcal{A}']{*} q$ is left to the reader. □

It can be proved that the class of recognizable tree languages is the smallest non trivial class of tree languages closed by linear tree homomorphisms and inverse tree homomorphisms. Tree homomorphisms do not in general preserve recognizability, therefore let us consider the following problem: given as instance a recognizable tree language L and a tree homomorphism h, is the set $h(L)$ recognizable? To our knowledge it is not known whether this problem is decidable. The reader should note that if this problem is decidable, the problem whether the set of normal forms of a rewrite system is recognizable is easily shown decidable (see Exercises 6 and 12).

As a conclusion we consider different special types of tree homomorphisms. These homomorphisms will be used in the next sections in order to simplify some proofs and will be useful in Chapter 6. Let h be a tree homomorphism determined by $h_{\mathcal{F}}$. The tree homomorphism h is said to be:

- **ϵ-free** if for each symbol $f \in \mathcal{F}$, t_f is not reduced to a variable.

- ***symbol to symbol*** if for each symbol $f \in \mathcal{F}$, $\mathcal{H}eight(t_f) = 1$. The reader should note that with our definitions a symbol to symbol tree homomorphism is ϵ-free. A linear symbol to symbol tree homomorphism changes the label of the input symbol, possibly erases some subtrees and possibly modifies order of subtrees.
- ***complete*** if for each symbol $f \in \mathcal{F}_n$, $\mathcal{V}ar(t_f) = \mathcal{X}_n$.
- a ***delabeling*** if h is a complete, linear, symbol to symbol tree homomorphism. Such a delabeling only changes the label of the input symbol and possibly order of subtrees.
- ***alphabetic*** if for each symbol $f \in \mathcal{F}_n$, $t_f = g(x_1, \ldots, x_n)$, where $g \in \mathcal{F}'_n$.

As a corollary of Theorem 6, alphabetic tree homomorphisms, delabelings and linear, symbol to symbol tree homomorphisms preserve recognizability. It can be proved that for these classes of tree homomorphisms, given h and a FTA $\mathcal{A}$ such that $L(\mathcal{A}) = L$ as instance, a FTA for the recognizable tree language $h(L)$ can be constructed in linear time. The same holds for $h^{-1}(L)$.

Example 16. Let $\mathcal{F} = \{f(,), g(), a\}$ and $\mathcal{F}' = \{f'(,), g'(), a'\}$. Let us consider some tree homomorphisms h determined by different $h_{\mathcal{F}}$.

- $h_{\mathcal{F}}(f) = x_1$, $h_{\mathcal{F}}(g) = f'(x_1, x_1)$, and $h_{\mathcal{F}}(a) = a'$. h is not linear, not ϵ-free, and not complete.
- $h_{\mathcal{F}}(f) = g'(x_1)$, $h_{\mathcal{F}}(g) = f'(x_1, x_1)$, and $h_{\mathcal{F}}(a) = a'$. h is a non linear symbol to symbol tree homomorphism. h is not complete.
- $h_{\mathcal{F}}(f) = f'(x_2, x_1)$, $h_{\mathcal{F}}(g) = g'(x_1)$, and $h_{\mathcal{F}}(a) = a'$. h is a delabeling.
- $h_{\mathcal{F}}(f) = f'(x_1, x_2)$, $h_{\mathcal{F}}(g) = g'(x_1)$, and $h_{\mathcal{F}}(a) = a'$. h is an alphabetic tree homomorphism.

1.5 Minimizing Tree Automata

In this section, we prove that, like in the word case, there exists a unique minimal automaton in the number of states for a given recognizable tree language.

A Myhill-Nerode Theorem for Tree Languages

The ***Myhill-Nerode Theorem*** is a classical result in the theory of finite automata. This theorem gives a characterization of the recognizable sets and it has numerous applications. A consequence of this theorem, among other consequences, is that there is essentially a unique minimum state DFA for every recognizable language over finite alphabet. The Myhill-Nerode Theorem generalizes in a straightforward way to automata on finite trees.

An equivalence relation $\equiv$ on $T(\mathcal{F})$ is a ***congruence*** on $T(\mathcal{F})$ if for every $f \in \mathcal{F}_n$

$$u_i \equiv v_i \; 1 \leq i \leq n \Rightarrow f(u_1, \ldots, u_n) \equiv f(v_1, \ldots, v_n) \; .$$

It is of ***finite index*** if there are only finitely many $\equiv$-classes. Equivalently a congruence is an equivalence relation closed under context, *i.e.*for all contexts $C \in \mathcal{C}(\mathcal{F})$, if $u \equiv v$, then $C[u] \equiv C[v]$. For a given tree language L, let us define the congruence $\equiv_L$ on $T(\mathcal{F})$ by: $u \equiv_L v$ if for all contexts $C \in \mathcal{C}(\mathcal{F})$,

$$C[u] \in L \text{ iff } C[v] \in L.$$

We are now ready to give the Theorem:

Myhill-Nerode Theorem. *The following three statements are equivalent:*

(i) L *is a recognizable tree language*

(ii) L *is the union of some equivalence classes of a congruence of finite index*

(iii) the relation $\equiv_L$ *is a congruence of finite index.*

Proof.

- *(i)* $\Rightarrow$ *(ii)* Assume that L is recognized by some complete DFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \delta)$. We consider δ as a transition function. Let us consider the relation $\equiv_{\mathcal{A}}$ defined on $T(\mathcal{F})$ by: $u \equiv_{\mathcal{A}} v$ if $\delta(u) = \delta(v)$. Clearly $\equiv_{\mathcal{A}}$ is a congruence relation and it is of finite index, since the number of equivalence classes is at most the number of states in Q. Furthermore, L is the union of those equivalence classes that include a term u such that $\delta(u)$ is a final state.

- *(ii)* $\Rightarrow$ *(iii)* Let us denote by $\sim$ the congruence of finite index. And let us assume that $u \sim v$. By an easy induction on the structure of terms, it can be proved that $C[u] \sim C[v]$ for all contexts $C \in \mathcal{C}(\mathcal{F})$. Now, L is the union of some equivalence classes of $\sim$, thus we have $C[u] \in L$ iff $C[v] \in L$. Thus $u \equiv_L v$, and the equivalence class of u in $\sim$ is contained in the equivalence class of u in $\equiv_L$. Consequently, the index of $\equiv_L$ is lower than or equal to the index of $\sim$ which is finite.

- *(iii)* $\Rightarrow$ *(i)* Let Q_{min} be the finite set of equivalence classes of $\equiv_L$. And let us denote by $[u]$ the equivalence class of a term u. Let the transition function δ_{min} be defined by:

  $$\delta_{min}(f, [u_1], \ldots, [u_n]) = [f(u_1, \ldots, u_n)].$$

  The definition of δ_{min} is consistent because $\equiv_L$ is a congruence. And let $Q_{min_f} = \{[u] \mid u \in L\}$. The DFTA $\mathcal{A}_{min} = (Q_{min}, \mathcal{F}, Q_{min_f}, \delta_{min})$ recognizes the tree language L.

□

As a corollary of the Myhill-Nerode Theorem, we can deduce an other algebraic characterization of recognizable tree languages. This characterization is a reformulation of the definition of recognizability. A set of ground terms L is recognizable if and only if there exist a finite $\mathcal{F}$-algebra $\mathcal{A}$, an $\mathcal{F}$-algebra homomorphism $\phi : T(\mathcal{F}) \to \mathcal{A}$ and a subset A' of the carrier $|\mathcal{A}|$ of $\mathcal{A}$ such that $L = \phi^{-1}(A')$.

Minimization of Tree Automata

First, we prove the existence and uniqueness of the minimum DFTA for a recognizable tree language. It is a consequence of the Myhill-Nerode Theorem because of the following result:

Corollary 2. *The minimum DFTA recognizing a recognizable tree language L is unique up to a renaming of the states and is given by $\mathcal{A}_{min}$ in the proof of the Myhill-Nerode Theorem.*

Proof. Assume that L is recognized by some DFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \delta)$. The relation $\equiv_{\mathcal{A}}$ is a refinement of $\equiv_L$ (see the proof of the Myhill-Nerode Theorem). Therefore the number of states of $\mathcal{A}$ is greater than or equal to the number of states of $\mathcal{A}_{min}$. If equality holds, $\mathcal{A}$ is reduced, *i.e.*all states are accessible, because otherwise a state could be removed leading to a contradiction. Let q be a state in Q and let u be such that $\delta(u) = q$. The state q can be identified with the state $\delta_{min}(u)$. This identification is consistent and defines a one to one correspondence between Q and Q_{min}. □

Second, we give a minimization algorithm for finding the minimum state DFTA equivalent to a given reduced DFTA. We identify an equivalence relation and the sequence of its equivalence classes.

Minimization Algorithm MIN
input: complete and reduced DFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \delta)$
begin
  Set P to $\{Q_f, Q - Q_f\}$ /* P is the initial equivalence relation*/
  repeat
    $P' = P$
    /* Refine equivalence P in P' */
    $qP'q'$ if
      qPq' and
      $\forall f \in \mathcal{F}_n \forall q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n \in Q$
      $\delta(f(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_n)) P \delta(f(q_1, \ldots, q_{i-1}, q', q_{i+1}, \ldots, q_n))$
  until $P' = P$
  Set Q_{min} to the set of equivalence classes of P
  /* we denote by $[q]$ the equivalence class of state q *w.r.t.*P */
  Set δ_{min} to $\{(f, [q_1], \ldots, [q_n]) \to [f(q_1, \ldots, q_n)]\}$
  Set Q_{min_f} to $\{[q] \mid q \in Q_f\}$
  output: DFTA $\mathcal{A}_{min} = (Q_{min}, \mathcal{F}, Q_{min_f}, \delta_{min})$
end

The DFTA constructed by the algorithm $\mathcal{MIN}$ is the minimum state DFTA for its tree language. Indeed, let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ the DFTA to which is applied the algorithm and let $L = L(\mathcal{A})$. Let $\mathcal{A}_{min}$ be the output of the algorithm. It is easy to show that the definition of $\mathcal{A}_{min}$ is consistent and that $L = L(\mathcal{A}_{min})$. Now, by contradiction, we can prove that $\mathcal{A}_{min}$ has no more states than the number of equivalence classes of $\equiv_L$.

1.6 Top Down Tree Automata

The tree automata that we have defined in the previous sections are also known as bottom-up tree automata because these automata start their computation at the leaves of trees. In this section we define top-down tree automata. Such an automaton starts its computation at the root in an initial state and then simultaneously works down the paths of the tree level by level. The tree automaton accepts a tree if a run built up in this fashion can be defined. It appears that top-down tree automata and bottom-up tree automata have the same expressive power. An important difference between bottom-up tree automata and top-down automata appears in the question of determinism since deterministic top-down tree automata are strictly less powerful than nondeterministic ones and therefore are strictly less powerful than bottom-up tree automata. Intuitively, it is due to the following: tree properties specified by deterministic top-down tree automata can depend only on path properties. We now make precise these remarks, but first formally define top-down tree automata.

A nondeterministic ***top-down*** finite Tree Automaton (top-down NFTA) over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$ where Q is a set of states (states are unary symbols), $I \subseteq Q$ is a set of initial states, and Δ is a set of rewrite rules of the following type :

$$q(f(x_1, \ldots, x_n)) \to f(q_1(x_1), \ldots, q_n(x_n)),$$

where $n \geq 0$, $f \in \mathcal{F}_n$, $q, q_1, \ldots, q_n \in Q$, $x_1, \ldots, x_n \in \mathcal{X}$.

When $n = 0$, *i.e.*when the symbol is a constant symbol a, a transition rule of top-down NFTA is of the form $q(a) \to a$. A top-down automaton starts at the root and moves downward, associating along a run a state with each subterm inductively. We do not formally define the move relation $\to_{\mathcal{A}}$ defined by a top-down NFTA because the definition is easily deduced from the corresponding definition for bottom-up NFTA. The tree language $L(\mathcal{A})$ recognized by $\mathcal{A}$ is the set of all ground terms t for which there is an initial state q in I such that

$$q(t) \xrightarrow[\mathcal{A}]{*} t.$$

The expressive power of bottom-up and top-down tree automata is the same. Indeed, we have the following Theorem:

Theorem 8 (The equivalence of top-down and bottom-up NFTAs). *The class of languages accepted by top-down NFTAs is exactly the class of recognizable tree languages.*

Proof. The proof is left to the reader. **Hint.** Reverse the arrows and exchange the sets of initial and final states. □

Top-down and bottom-up tree automata have the same expressive power because they define the same classes of tree languages. Nevertheless they do not have the same behavior from an algorithmic point of view because nondeterminism can not be reduced in the class of top-down tree automata.

Proposition 1 (Top-down NFTAs and top-down DFTAs). *A top-down finite Tree Automaton $(Q, \mathcal{F}, I, \Delta)$ is deterministic (top-down DFTA) if there is one initial state and no two rules with the same left-hand side. Top-down DFTAs are strictly less powerful than top-down NFTAs, i.e.there exists a recognizable tree language which is not accepted by a top-down DFTA.*

Proof. Let $\mathcal{F} = \{f(,), a, b\}$. And let us consider the recognizable tree language $T = \{f(a, b), f(b, a)\}$. Now let us suppose there exists a top-down DFTA that accepts T, the automaton should accept the term $f(a, a)$ leading to a contradiction. Obviously the tree language $T = \{f(a, b), f(b, a)\}$ is recognizable by a finite union of top-down DFTA but there is a recognizable tree language which is not accepted by a finite union of top-down DFTA (see Exercise 2). □

1.7 Decision Problems and their Complexity

In this section, we study some decision problems and their complexity. The size of an automaton will be the size of its representation. More formally:

Definition 1. *Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ be a NFTA over $\mathcal{F}$. The size of a rule $f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n))$ is $arity(f) + 2$. The size of $\mathcal{A}$ noted $\|A\|$, is defined by:*

$$\|\mathcal{A}\| = |Q| + \sum_{f(q_1(x_1),\ldots,q_n(x_n))\to q(f(x_1,\ldots,x_n))\in\Delta} (arity(f) + 2).$$

We will work in the frame of RAM machines, with uniform measure.

Membership

Instance A ground term.

Answer "yes" if and only if the term is recognized by a given automaton.

Let us first remark that, in our model, for a given deterministic automaton, a run on a tree can be computed in $O(\|t\|)$. The complexity of the problem is:

Theorem 9. *The membership problem is ALOGTIME-complete.*

Uniform Membership

Instance A tree automaton and a ground term.

Answer "yes" if and only if the term is recognized by the given automaton.

Theorem 10. *The uniform membership problem can be decided in linear time for DFTA, in polynomial time for NFTA.*

Proof. In the deterministic case, from a term t and the automaton $\|\mathcal{A}\|$, we can compute a run in $O(\|t\| + \|\mathcal{A}\|)$. In the nondeterministic case, the idea is similar to the word case: the algorithm determinizes along the computation, *i.e.*for each node of the term, we compute the set of reached states. The complexity of this algorithm will be in $O(\|t\| \times \|\mathcal{A}\|)$.

□

The uniform membership problem has been proved LOGSPACE-complete for deterministic top-down tree automata, LOGCFL-complete for NFTA under log-space reductions. For DFTA, it has been proven LOGDCFL, but the precise complexity remains open.

Emptiness

Instance A tree automaton

Answer "yes" if and only if the recognized language is empty.

Theorem 11. *It can be decided in linear time whether the language accepted by a finite tree automaton is empty.*

Proof. The minimal height of accepted terms can be bounded by the number of states using Corollary 1; so, as membership is decidable, emptiness is decidable. Of course, this approach does not provide a practicable algorithm. To get an efficient algorithm, it suffices to notice that a NFTA accepts at least one tree if and only if there is an accessible final state. In other words, the language recognized by a **reduced** automaton is empty if and only if the set of final states is non empty. Reducing an automaton can be done in $O(|Q| \times \|\mathcal{A}\|)$ by the reduction algorithm given in Section 1.1. Actually, this algorithm can be improved by choosing an adequate data structure in order to get a linear algorithm (see Exercise 17). This linear least fixpoint computation holds in several frameworks. For example, it can be viewed as the satisfiability test of a set of propositional Horn formulae. The reduction is easy and linear: each state q can be associated with a propositional variable X_q and each rule $r : f(q_1, \ldots, q_n) \to q$ can be associated with a propositional Horn formula $F_r = X_q \vee \neg X_{q_1} \vee \cdots \vee \neg X_{q_n}$. It is straightforward that satisfiability of $\{F_r\} \cup \{\neg X_q / q \in Q_f\}$ is equivalent to emptiness of the language recognized by $(Q, \mathcal{F}, Q_f, \Delta)$. So, as satisfiability of a set of propositional Horn formulae can be decided in linear time, we get a linear algorithm for testing emptiness for NFTA. □

The emptiness problem is **P-complete** with respect to logspace reductions, even when restricted to deterministic tree automata. The proof can easily be done since the problem is very close to *the solvable path systems* problem which is known to be P-complete (see Exercise 18).

Intersection non-emptiness

Instance A finite sequence of tree automata.

Answer "yes" if and only if there is at least one term recognized by each automaton of the sequence.

Theorem 12. *The intersection problem for tree automata is EXPTIME-complete.*

Proof. By constructing the product automata for the n automata, and then testing non-emptiness, we get an algorithm in $O(\|\mathcal{A}_1\| \times \cdots \times \|\mathcal{A}_n\|)$. The proof of EXPTIME-hardness is based on simulation of an alternating linear space-bounded Turing machine. Roughly speaking, with such a machine and an input of length n can be associated polynomially n tree automata whose intersection corresponds to the set of accepting computations on the input. It is worth noting that the result holds for deterministic top down tree automata as well as for deterministic bottom-up ones. □

Finiteness

Instance A tree automaton

Answer "yes" if and only if the recognized language is finite.

Theorem 13. *Finiteness can be decided in polynomial time.*

Proof. Let us consider a NFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$. Deciding finiteness of $\mathcal{A}$ is direct by Corollary 1: it suffices to find an accepted term t *s.t.*$|Q| < \|t\| \leq 2*|Q|$. A more efficient way to test finiteness is to check the existence of a loop: the language is infinite if and only if there is a loop on some useful state, *i.e.*there exist an accessible state q and contexts C and C' such that $C[q] \xrightarrow[\mathcal{A}]{*} q$ and $C'[q] \xrightarrow[\mathcal{A}]{*} q'$ for some final state q'. Computing accessible and coaccessible states can be done in $O(|Q| \times \|\mathcal{A}\|)$ or in $O(\|\mathcal{A}\|)$ by using an ad hoc representation of the automaton. For a given q, deciding if there is a loop on q can be done in $O(\|\mathcal{A}\|)$. So, finiteness can be decided in $O(|Q| \times \|\mathcal{A}\|)$. □

Emptiness of the Complement

Instance A tree automaton.

Answer "yes" if and only if every term is accepted by the automaton

Deciding whether a deterministic tree automaton recognizes the set of all terms is polynomial for a fixed alphabet: we just have to check whether the automaton is complete (which can be done in $O(|\mathcal{F}| \times |Q|^{Arity(\mathcal{F})})$) and then it remains only to check that all accessible states are final. For nondeterministic automata, the following result proves in some sense that determinization with its exponential cost is unavoidable:

Theorem 14. *The problem whether a tree automaton accepts the set of all terms is EXPTIME-complete for nondeterministic tree automata.*

Proof. The proof of this theorem is once more based on simulation of a linear space bounded alternating Turing machine: indeed, the complement of the accepting computations on an input w can be coded polynomially in a recognizable tree language. □

Equivalence

Instance Two tree automata

Answer "yes" if and only if the automata recognize the same language.

Theorem 15. *Equivalence is decidable for tree automata.*

Proof. Clearly, as the class of recognizable sets is effectively closed under complementation and intersection, and as emptiness is decidable, equivalence is decidable. For two deterministic complete automata $\mathcal{A}_1$ and $\mathcal{A}_2$, we get by these means an algorithm in $O(\|\mathcal{A}_1\| \times \|\mathcal{A}_2\|)$. (Another way is to compare the minimal automata). For nondeterministic ones, this approach leads to an exponential algorithm. □

As we have proved that deciding whether an automaton recognizes the set of all ground terms is EXPTIME-hard, we get immediately:

Corollary 3. *The inclusion problem and the equivalence problem for NFTAs are EXPTIME-complete.*

Singleton Set Property

Instance A tree automaton

Answer "yes" if and only if the recognized language is a singleton set.

Theorem 16. *The singleton set property is decidable in polynomial time.*

Proof. There are several ways to get a polynomial algorithm for this property. A first one would be to first check non-emptiness of $L(\mathcal{A})$ and then "extract" from $\mathcal{A}$ a DFA $\mathcal{B}$ whose size is smaller than $\|\mathcal{A}\|$ and which accepts a single term recognized by $\mathcal{A}$. Then it remains to check emptiness of $L(\mathcal{A}) \cap \overline{L(\mathcal{B})}$. This can be done in polynomial time, even if $\mathcal{B}$ is non complete.

Another way is: for each state of a bottom-up tree automaton $\mathcal{A}$, compute, up to 2, the number $C(q)$ of terms leading to state q. This can be done in a straightforward way when $\mathcal{A}$ is deterministic; when $\mathcal{A}$ is non deterministic, this can be also done in polynomial time:

Singleton Set Test Algorithm
input: NFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$
begin
  Set $C(q)$ to 0, for every q in Q
  /* $C(q) \in \{0, 1, 2\}$ is the number, up to 2, of terms leading to state q */
  /* if $C(q) = 1$ then $T(q)$ is a representation of the accepted tree */
  repeat
    for each rule $f(q_1, \ldots, q_n) \to q \in \Delta$ do
      Case $\wedge_j C(q_j) >= 1$ and $C(q_i) = 2$ for some i: Set $C(q)$ to 2
      Case $\wedge_j C(q_j) = 1$ and $C(q) = 0$: Set $C(q)$ to 1, $T(q)$ to $f(q_1, ...q_n)$
      Case $\wedge_j C(q_j) = 1$, $C(q) = 1$ and $\mathit{Diff}(T(q), f(q_1, \ldots, q_n))$:
        Set $C(q)$ to 2
      Others null
      where $\mathit{Diff}(f(q_1, ..., q_n), g(q'_1, ..., q'_n))$ defined by:
        /* Diff can be computed polynomially by using memorization. */
        if $(f \neq g)$ then return true
        elseif $\mathit{Diff}(T(q_i), T(q'_i)$ for some q_i then return True
        else return False
  until C can not be changed
  output:
  /*$L(\mathcal{A})$ is empty */
  if $\wedge_{q \in Q_f}\ C(q) = 0$ **then** return False
  /* two terms in $L(\mathcal{A})$ accepted in the same state or two different states */
  elseif $\exists q \in Q_f\ C(q) = 2$ **then** return False
  elseif $\exists q, q' \in Q_f\ C(q) = C(q') = 1$ and $\mathit{Diff}(T(q), T(q'))$ **then** return False
  /* in all other cases $L(\mathcal{A})$ is a singleton set*/

else return True.
end

□

Other complexity results for "classical" problems can be found in the exercises. E.g., let us cite the following problem whose proof is sketched in Exercise 11

Ground Instance Intersection Problem

Instance A term t, a tree automaton $\mathcal{A}$.

Answer "yes" if and only if there is at least a ground instance of t which is accepted by $\mathcal{A}$.

Theorem 17. *The Ground Instance Intersection Problem for tree automata is P when t is linear, NP-complete when t is non linear and $\mathcal{A}$ deterministic, EXPTIME-complete when t is non linear and $\mathcal{A}$ non deterministic.*

1.8 Exercises

Starred exercises are discussed in the bibliographic notes.

Exercise 1. Let $\mathcal{F} = \{f(,), g(), a\}$. Define a top-down NFTA, a NFTA and a DFTA for the set $G(t)$ of ground instances of term $t = f(f(a, x), g(y))$ which is defined by $G(t) = \{f(f(a, u), g(v)) \mid u, v \in T(\mathcal{F})\}$. Is it possible to define a top-down DFTA for this language?

Exercise 2. Let $\mathcal{F} = \{f(,), g(), a\}$. Define a top-down NFTA, a NFTA and a DFTA for the set $M(t)$ of terms which have a ground instance of term $t = f(a, g(x))$ as a subterm, that is $M(t) = \{C[f(a, g(u))] \mid C \in \mathcal{C}(\mathcal{F}),\ u \in T(\mathcal{F})\}$. Is it possible to define a top-down DFTA for this language?

Exercise 3. Let $\mathcal{F} = \{g(), a\}$. Is the set of ground terms whose height is even recognizable? Let $\mathcal{F} = \{f(,), g(), a\}$. Is the set of ground terms whose height is even recognizable?

Exercise 4. Let $\mathcal{F} = \{f(,), a\}$. Prove that the set $L = \{f(t, t) \mid t \in T(\mathcal{F})\}$ is not recognizable. Let $\mathcal{F}$ be any ranked alphabet which contains at least one constant symbol a and one binary symbol $f(,)$. Prove that the set $L = \{f(t, t) \mid t \in T(\mathcal{F})\}$ is not recognizable.

Exercise 5. Prove the equivalence between top-down NFTA and NFTA.

Exercise 6. Let $\mathcal{F} = \{f(,), g(), a\}$ and $\mathcal{F}' = \{f'(,), g(), a\}$. Let us consider the tree homomorphism h determined by $h_{\mathcal{F}}$ defined by: $h_{\mathcal{F}}(f) = f'(x_1, x_2)$, $h_{\mathcal{F}}(g) = f'(x_1, x_1)$, and $h_{\mathcal{F}}(a) = a$. Is $h(T(\mathcal{F}))$ recognizable? Let $L_1 = \{g^i(a) \mid i \geq 0\}$, then L_1 is a recognizable tree language, is $h(L_1)$ recognizable? Let L_2 be the recognizable

tree language defined by $L_2 = L(\mathcal{A})$ where $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ is defined by: $Q = \{q_a, q_g, q_f\}$, $Q_f = \{q_f\}$, and Δ is the following set of transition rules:

$$\begin{array}{{}rcl@{\qquad}rcl}
\{ \quad a & \to & q_a & g(q_a) & \to & q_g \\
f(q_a, q_a) & \to & q_f & f(q_g, q_g) & \to & q_f \\
f(q_a, q_g) & \to & q_f & f(q_g, q_a) & \to & q_f \\
f(q_a, q_f) & \to & q_f & f(q_f, q_a) & \to & q_f \\
f(q_g, q_f) & \to & q_f & f(q_f, q_g) & \to & q_f \\
f(q_f, q_f) & \to & q_f \quad \}. & & &
\end{array}$$

Is $h(L_2)$ recognizable?

Exercise 7. Let $\mathcal{F}_1 = \{or(,), and(,), not(), 0, 1, x\}$. A ground term over $\mathcal{F}$ can be viewed as a boolean formula over variable x. Define a DFTA which recognizes the set of satisfiable boolean formulae over x. Let $\mathcal{F}_n = \{or(,), and(,), not(), 0, 1, x_1, \ldots, x_n\}$. A ground term over $\mathcal{F}$ can be viewed as a boolean formula over variables $x_1, \ldots, x_n$. Define a DFTA which recognizes the set of satisfiable boolean formulae over $x_1, \ldots, x_n$.

Exercise 8. Let t be a linear term in $T(\mathcal{F}, \mathcal{X})$. Prove that the set $G(t)$ of ground instances of term t is recognizable. Let R be a finite set of linear terms in $T(\mathcal{F}, \mathcal{X})$. Prove that the set $G(R)$ of ground instances of set R is recognizable.

Exercise 9. * Let R be a finite set of linear terms in $T(\mathcal{F}, \mathcal{X})$. We define the set $Red(R)$ of reducible terms for R to be the set of ground terms which have a ground instance of some term in R as a subterm.

1. Prove that the set $Red(R)$ is recognizable.
2. Prove that the number of states of a DFA recognizing $Red(R)$ can be at least 2^{n-1} where n is the size (number of nodes) of R. Hint: Consider the set reduced to the pattern $h(f(x_1, f(x_2, f(x_3), \ldots, (f(x_{p-1}, f(a, x_p)\cdots)$.
3. Let us now suppose that R is a finite set of **ground** terms. Prove that we can construct a DFA recognizing $Red(R)$ whose number of states is at most $n + 2$ where n is the number of different subterms of R.

Exercise 10. * Let R be a finite set of linear terms in $T(\mathcal{F}, \mathcal{X})$. A term t is inductively reducible for R if all the ground instances of term t are reducible for R. Prove that inductive reducibility of a linear term t for a set of linear terms R is decidable.

Exercise 11. *

We consider the following decision problem:

Instance t a term in $T(\mathcal{F}, \mathcal{X})$ and $\mathcal{A}$ a NFTA

Answer "yes" if and only if Yes, iff at least one ground instance of t is accepted by $|A$.

1. Let us first suppose that t is linear; prove that the property is P.
   Hint: a NFTA for the set of ground instances of t can ce computed polynomially (see Exercise 8
2. Let us now suppose that t is non linear but that $\mathcal{A}$ is deterministic.
   (a) Prove that the property is NP. Hint: we just have to guess a substitution of the variables of t by states.
   (b) Prove that the property is NP-hard.
   Hint: just consider a term t which represents a boolean formula and $\mathcal{A}$ a DFTA which accepts valid formulas.

3. Let us now suppose that t is non linear and that $\mathcal{A}$ is non deterministic.
   Prove that the property is $EXPTIME-$complete.
   Hint: use the EXPTIME-hardness of intersection non-emptiness.

Exercise 12. * We consider the following two problems. First, given as instance a recognizable tree language L and a tree homomorphism h, is the set $h(L)$ recognizable? Second, given as instance a set R of terms in $T(\mathcal{F}, \mathcal{X})$, is the set $Red(R)$ recognizable? Prove that if the first problem is decidable, the second problem is easily shown decidable.

Exercise 13. Let $\mathcal{F} = \{f(,), a, b\}$.

1. Let us consider the set of ground terms L_1 defined by the following two conditions:
   - $f(a, b) \in L_1$,
   - $t \in L_1 \Rightarrow f(a, f(t, b)) \in L_1$.

   Prove that the set L_1 is recognizable.
2. Prove that the set $L_2 = \{t \in T(\mathcal{F}) \mid |t|_a = |t|_b\}$ is not recognizable where $|t|_a$ (respectively $|t|_b$) denotes the number of a (respectively the number of b) in t.
3. Let L be a recognizable tree language over $\mathcal{F}$. Let us suppose that f is a commutative symbol. Let $C(L)$ be the congruence closure of set L for the set of equations $C = \{f(x, y) = f(y, x)\}$. Prove that $C(L)$ is recognizable.
4. Let L be a recognizable tree language over $\mathcal{F}$. Let us suppose that f is a commutative and associative symbol. Let $AC(L)$ be the congruence closure of set L for the set of equations $AC = \{f(x, y) = f(y, x); f(x, f(y, z)) = f(f(x, y), z)\}$. Prove that in general $AC(L)$ is not recognizable.
5. Let L be a recognizable tree language over $\mathcal{F}$. Let us suppose that f is an associative symbol. Let $A(L)$ be the congruence closure of set L for the set of equations $A = \{f(x, f(y, z)) = f(f(x, y), z)\}$. Prove that in general $A(L)$ is not recognizable.

Exercise 14. * Consider the *complement problem*:

- **Instance** A term $t \in T(\mathcal{F}, \mathcal{X})$ and terms $t_1, \ldots, t_n$,
- **Question** There is a ground instance of t which is not an instance of any t_i.

Prove that the complement problem is decidable whenever term t and all terms t_i are linear. Extend the proof to handle the case where t is a term (not necessarily linear).

Exercise 15. * Let $\mathcal{F}$ be a ranked alphabet and suppose that $\mathcal{F}$ contains some symbols which are commutative and associative. The set of ground AC-instances of a term t is the AC-congruence closure of set $G(t)$. Prove that the set of ground AC-instances of a linear term is recognizable. The reader should note that the set of ground AC-instances of a set of linear terms is not recognizable (see Exercice 13).

Prove that the *AC-complement problem* is decidable where the AC-complement problem is defined by:

- **Instance** A linear term $t \in T(\mathcal{F}, \mathcal{X})$ and linear terms $t_1, \ldots, t_n$,
- **Question** There is a ground AC-instance of t which is not an AC-instance of any t_i.

Exercise 16. * Let $\mathcal{F}$ be a ranked alphabet and $\mathcal{X}$ be a countable set of variables. Let S be a rewrite system on $T(\mathcal{F},\mathcal{X})$ (the reader is referred to [DJ90]) and L be a set of ground terms. We denote by $S^*(L)$ the set of reductions of terms in L by S and by $S(L)$ the set of ground S-normal forms of set L. Formally,

$$S^*(L) = \{t \in T(\mathcal{F}) \mid \exists u \in L\ u \xrightarrow{*} t\},$$

$$S(L) = \{t \in T(\mathcal{F}) \mid t \in IRR(S) \text{ and } \exists u \in L\ u \xrightarrow{*} t\} = IRR(S) \cap S^*(L)$$

where $IRR(S)$ denotes the set of ground irreducible terms for S. We consider the two following decision problems:

(1rst order reachability)

- **Instance** A rewrite system S, two ground terms u and v,
- **Question** $v \in S^*(\{u\})$.

(2nd order reachability)

- **Instance** A rewrite system S, two recognizable tree languages L and L',
- **Question** $S^*(L) \subseteq L'$.

1. Let us suppose that rewrite system S satisfies:

   (PreservRec) If L is recognizable, then $S^*(L)$ is recognizable.

   What can be said about the two reachability decision problems? Give a sufficient condition on rewrite system S satisfying (PreservRec) such that S satisfies (NormalFormRec) where (NormalFormRec) is defined by:

   (NormalFormRec) If L is recognizable, then $S(L)$ is recognizable.

2. Let $\mathcal{F} = \{f(,), g(), h(), a\}$. Let $L = \{f(t_1, t_2) \mid t_1, t_2 \in T(\{g(), h(), a\}\}$, and S is the following set of rewrite rules:

$$\left\{ \begin{array}{rcl@{\qquad}rcl} f(g(x),h(y)) & \to & f(x,y) & f(h(x),g(y)) & \to & f(x,y) \\ g(h(x)) & \to & x & h(g(x)) & \to & x \\ f(a,x) & \to & x & f(x,a) & \to & x \end{array} \right\}$$

   Are the sets L, $S^*(L)$, and $S(L)$ recognizable?

3. Let $\mathcal{F} = \{f(,), g(), h(), a\}$. Let $L = \{g(h^n(a)) \mid n \geq 0\}$, and S is the following set of rewrite rules:

$$\{ \quad g(x) \quad \to \quad f(x,x) \quad \}$$

   Are the sets L, $S^*(L)$, and $S(L)$ recognizable?

4. Let us suppose now that rewrite system S is linear and monadic, *i.e.*all rewrite rules are of one of the following three types:

$$\begin{array}{lcll} (1) & l & \to & a & , a \in \mathcal{F}_0 \\ (2) & l & \to & x & , x \in \mathcal{V}ar(l) \\ (3) & l & \to & f(x_1,\ldots,x_p) & , x_1,\ldots,x_p \in \mathcal{V}ar(l), f \in \mathcal{F}_p \end{array}$$

   where l is a linear term (no variable occurs more than once in t) whose height is greater than 1. Prove that a linear and monadic rewrite system satisfies (PreservRec). Prove that (PreservRec) is false if the right-hand side of rules of type (3) may be non linear.

Exercise 17. Design a linear-time algorithm for testing emptiness of the language recognized by a tree automaton:

Instance A tree automaton

Answer "yes" if and only if the language recognized is empty.

Hint: Choose a suitable data structure for the automaton. For example, a state could be associated with the list of the "adresses" of the rules whose left-hand side contain it (eventually, a rule can be repeated); each rule could be just represented by a counter initialized at the arity of the corresponding symbol and by the state of the right-hand side. Activating a state will decrement the counters of the corresponding rules. When the counter of a rule becomes null, the rule can be applied: the right-hand side state can be activated.

Exercise 18.
The Solvable Path Problem is the following:

Instance a finite set X and three sets $R \subset X \times X \times X, X_s \subset X$ and $X_t \subset X$.

Answer "yes" if and only if $X_t \cap A$ is non empty, where A is the least subset of X such that $X_s \subset A$ and if $y, z \in A$ and $(x, y, z) \in R$, then $x \in A$.

Prove that this $P - complete$ problem is log-space reducible to the emptiness problem for tree automata.

Exercise 19. A *flat automaton* is a tree automaton which has the following property: there is an ordering $\geq$ on the states and a particular state $q_\top$ such that the transition rules have one of the following forms:

1. $f(q_\top, \ldots, q_\top) \to q_\top$
2. $f(q_1, \ldots, q_n) \to q$ with $q > q_i$ for every i
3. $f(q_\top, \ldots, q_\top, q, q_\top, \ldots, q_\top) \to q$

Moreover, we assume that all terms are accepted in the state $q_\top$. (The automaton is called *flat* because there are no "nested loop").

Prove that the intersection of two flat automata is a finite union of automata whose size is linear in the sum of the original automata. (This contrasts with the construction of Theorem 5 in which the intersection automaton's size is the product of the sizes of its components).

Deduce from the above result that the intersection non-emptiness problem for flat automata is in NP (compare with Theorem 12).

1.9 Bibliographic Notes

Tree automata were introduced by Doner [Don65, Don70] and Thatcher and Wright [TW65, TW68]. Their goal was to prove the decidability of the weak second order theory of multiple successors. The original definitions are based on the algebraic approach and involve heavy use of universal algebra and/or category theory.

Many of the basic results presented in this chapter are the straightforward generalization of the corresponding results for finite automata. It is difficult to attribute a particular result to any one paper. Thus, we only give a list of some important contributions consisting of the above mentioned papers of Doner, Thatcher and Wright and also Eilenberg and Wright [EW67], Thatcher [Tha70], Brainerd [Bra68, Bra69], Arbib and Give'on [AG68]. All the results of this chapter and a more complete and detailed list of references can be found in the textbook of Gécseg and Steinby [GS84] and also in their recent survey [GS96]. For an overview of the notion of recognizability in general algebraic structures see Courcelle [Cou89] and the fundamental paper of Mezei and Wright [MW67].

In Nivat and Podelski [NP89] and [Pod92], the theory of recognizable tree languages is reduced to the theory of recognizable sets in an infinitely generated free monoid.

The results of Sections 1.1, 1.2, and 1.3 were noted in many of the papers mentioned above, but, in this textbook, we present these results in the style of the undergraduate textbook on finite automata by Hopcroft and Ullman [HU79]. Tree homomorphisms were defined as a special case of tree transducers, see Thatcher [Tha73]. The reader is referred to the bibliographic notes in Chapter 6 of the present textbook for detailed references. The reader should note that our proof of preservation of recognizability by tree homomorphisms and inverse tree homomorphisms is a direct construction using FTA. A more classical proof can be found in [GS84] and uses regular tree grammars (see Chapter 2).

Minimal tree recognizers and Nerode's congruence appear in Brainerd [Bra68, Bra69], Arbib and Give'on [AG68], and Eilenberg and Wright [EW67]. The proof we presented here is by Kozen [Koz92] (see also Fülöp and Vágvölgyi [FV89]). Top-down tree automata were first defined by Rabin [Rab69]. The reader is referred to [GS84] and [GS96] for more references and for the study of some subclasses of recognizable tree languages such as the tree languages recognized by deterministic top-down tree automata. An alternative definition of deterministic top-down tree automata was defined in [NP97] leading to "homogeneous" tree languages, also a minimization algorithm was given.

Some results of Sections 1.7 are "folklore" results. Complexity results for the membership problem and the uniform membership problem could be found in [Loh01]. Other interesting complexity results for tree automata can be found in Seidl [Sei89], [Sei90]. The EXPTIME-hardness of the problem of intersection non-emptiness is often used; this problem is close to problems of type inference and an idea of the proof can be found in [FSVY91]. A proof for deterministic top-down automata can be found in [Sei94b]. A detailed proof in the deterministic bottom-up case as well as some other complexity results are in [Vea97a], [Vea97b].

We have only considered finite ordered ranked trees. Unranked trees are used for XML Document Type Definitions and more generally for XML schema languages [MLM01]. The theory of unranked trees dates back to Thatcher. All the fundamental results for finite tree automata can be extended to the case of unranked trees and the methods are similar [BKMW01]. An other extension is to consider unordered trees. A general discussion about unordered and unranked trees can be found in the bibliographical notes of Section 4.

Numerous exercises of the present chapter illustrate applications of tree automata theory to automated deduction and to the theory of rewriting systems. These applications are studied in more details in Section 3.4. Results about tree automata and rewrite systems are collected in Gilleron and Tison [GT95]. Let S be a term rewrite system (see for example Dershowitz and Jouannaud [DJ90] for a survey on rewrite systems), if S is left-linear the set $IRR(S)$ of irreducible ground terms *w.r.t.*S is a recognizable tree language. This result first appears in Gallier and Book [GB85] and is the subject of Exercise 9. However not every recognizable tree language is the set of irreducible terms *w.r.t.*a rewrite system S (see Fülöp and Vágvölgyi [FV88]). It was proved that the problem whether, given a rewrite system S as instance, the set of irreducible terms is recognizable is decidable (Kucherov [Kuc91]). The problem of preservation of regularity by tree homomorphisms is not known decidable. Exercise 12 shows connections

between preservation of regularity for tree homomorphisms and recognizability of sets of irreducible terms for rewrite systems.

The notion of inductive reducibility (or ground reducibility) was introduced in automated deduction. A term t is S-inductively (or S-ground) reducible for S if all the ground instances of term t are reducible for S. Inductive reducibility is decidable for a linear term t and a left-linear rewrite system S. This is Exercise 10, see also Section 3.4.2. Inductive reducibility is decidable for finite S (see Plaisted [Pla85]). Complement problems are also introduced in automated deduction. They are the subject of Exercises 14 and 15. The complement problem for linear terms was proved decidable by Lassez and Marriott [LM87] and the AC-complement problem by Lugiez and Moysset [LM94].

The reachability problem is defined in Exercise 16. It is well known that this problem is undecidable in general. It is decidable for rewrite systems preserving recognizability, *i.e.*such that for every recognizable tree language L, the set of reductions of terms in L by S is recognizable. This is true for linear and monadic rewrite systems (right-hand sides have depth less than 1). This result was obtained by K. Salomaa [Sal88] and is the matter of Exercise 16. This is true also for linear and semi-monadic (variables in the right-hand sides have depth at most 1) rewrite systems, Coquidé et al. [CDGV94]. Other interesting results can be found in [Jac96] and [NT99].

Bibliography

[AD82] A. Arnold and M. Dauchet. Morphismes et bimorphismes d'arbres. *Theorical Computer Science*, 20:33–93, 1982.

[AG68] M. A. Arbib and Y. Give'on. Algebra automata I: Parallel programming as a prolegomena to the categorical approach. *Information and Control*, 12(4):331–345, April 1968.

[AKVW93] A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The complexity of set constraints. In E. Börger, Y. Gurevich, and K. Meinke, editors, *Proceedings of Computer Science Logic*, volume 832 of *Lecture Notes in Computer Science*, pages 1–17, 1993. Techn. Report 93-1352, Cornell University.

[AKW95] A. Aiken, D. Kozen, and E.L. Wimmers. Decidability of systems of set constraints with negative constraints. *Information and Computation*, 122(1):30–44, October 1995.

[AM78] M.A. Arbib and E.G. Manes. Tree transformations and semantics of loop-free programs. *Acta Cybernetica*, 4:11–17, 1978.

[AM91] A. Aiken and B. R. Murphy. Implementing regular tree expressions. In *Proceedings of the ACM conf. on Functional Programming Languages and Computer Architecture*, pages 427–447, 1991.

[AU71] A. V. Aho and J. D. Ullmann. Translations on a context-free grammar. *Information and Control*, 19:439–475, 1971.

[AW92] A. Aiken and E.L. Wimmers. Solving Systems of Set Constraints. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science* [IEE92], pages 329–340.

[Bak78] B.S. Baker. Generalized syntax directed translation, tree transducers, and linear space. *Journal of Comput. and Syst. Sci.*, 7:876–891, 1978.

[BGG97] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem.* Perspectives of Mathematical Logic. Springer Verlag, 1997.

[BGW93] L. Bachmair, H. Ganzinger, and U. Waldmann. Set constraints are the monadic class. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 75–83. IEEE Computer Society Press, 19–23 June 1993.

[BJ97] A. Bouhoula and J.-P. Jouannaud. Automata-driven automated induction. In *Proceedings, 12$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science* [IEE97].

[BKMW01] A. Brüggemann-Klein, M.Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Technical Report HKTUST-TCSC-2001-05, HKUST Theoretical Computer Science Center Research, 2001.

[Boz99] S. Bozapalidis. Equational elements in additive algebras. *Theory of Computing Systems*, 32(1):1–33, 1999.

[Boz01] S. Bozapalidis. Context-free series on trees. *ICOMP*, 169(2):186–229, 2001.

[BR82] Jean Berstel and Christophe Reutenauer. Recognizable formal power series on trees. *TCS*, 18:115–148, 1982.

[Bra68] W. S. Brainerd. The minimalization of tree automata. *Information and Control*, 13(5):484–491, November 1968.

[Bra69] W. S. Brainerd. Tree generating regular systems. *Information and Control*, 14(2):217–231, February 1969.

[BT92] B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In A. Finkel and M. Jantzen, editors, *9^{t}h Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 161–171, 1992.

[Büc60] J. R. Büchi. On a decision method in a restricted second order arithmetic. In Stanford Univ. Press., editor, *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science*, pages 1–11, 1960.

[CCC^{+}94] A.-C. Caron, H. Comon, J.-L. Coquidé, M. Dauchet, and F. Jacquemard. Pumping, cleaning and symbolic constraints solving. In *Proceedings, International Colloquium Automata Languages and Programming*, volume 820 of *Lecture Notes in Computer Science*, pages 436–449, 1994.

[CD94] H. Comon and C. Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, August 1994.

[CDGV94] J.-L. Coquide, M. Dauchet, R. Gilleron, and S. Vagvolgyi. Bottom-up tree pushdown automata : Classification and connection with rewrite systems. *Theorical Computer Science*, 127:69–98, 1994.

[CG90] J.-L. Coquidé and R. Gilleron. Proofs and reachability problem for ground rewrite systems. In *Proc. IMYCS'90*, Smolenice Castle, Czechoslovakia, November 1990.

[Chu62] A. Church. Logic, arithmetic, automata. In *Proc. International Mathematical Congress*, 1962.

[CJ97a] H. Comon and F. Jacquemard. Ground reducibility is EXPTIME-complete. In *Proceedings, 12$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science* [IEE97], pages 26–34.

[CJ97b] H. Comon and Y. Jurski. Higher-order matching and tree automata. In M. Nielsen and W. Thomas, editors, *Proc. Conf. on Computer Science Logic*, volume 1414 of *LNCS*, pages 157–176, Aarhus, August 1997. Springer-Verlag.

[CK96] A. Cheng and D. Kozen. A complete Gentzen-style axiomatization for set constraints. In *Proceedings, International Colloquium Automata Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 134–145, 1996.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[Com89] H. Comon. Inductive proofs by specification transformations. In *Proceedings, Third International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 76–91, 1989.

[Com95] H. Comon. Sequentiality, second-order monadic logic and tree automata. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 26–29 June 1995.

[Com98a] H. Comon. Completion of rewrite systems with membership constraints. Part I: deduction rules. *Journal of Symbolic Computation*, 25:397–419, 1998. This is a first part of a paper whose abstract appeared in Proc. ICALP 92, Vienna.

[Com98b] H. Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *Journal of Symbolic Computation*, 25:421–453, 1998. This is the second part of a paper whose abstract appeared in Proc. ICALP 92, Vienna.

[Cou86] B. Courcelle. Equivalences and transformations of regular systems–applications to recursive program schemes and grammars. *Theorical Computer Science*, 42, 1986.

[Cou89] B. Courcelle. *On Recognizable Sets and Tree Automata*, chapter Resolution of Equations in Algebraic Structures. Academic Press, m. Nivat and Ait-Kaci edition, 1989.

[Cou92] B. Courcelle. Recognizable sets of unrooted trees. In M. Nivat and A. Podelski, editors, *Tree Automata and Languages*. Elsevier Science, 1992.

[CP94a] W. Charatonik and L. Pacholski. Negative set constraints with equality. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 128–136. IEEE Computer Society Press, 4–7 July 1994.

[CP94b] W. Charatonik and L. Pacholski. Set constraints with projections are in NEXPTIME. In *Proceedings of the* 35^{th} *Symp. Foundations of Computer Science*, pages 642–653, 1994.

[CP97] W. Charatonik and A. Podelski. Set Constraints with Intersection. In *Proceedings,* 12^{th} *Annual IEEE Symposium on Logic in Computer Science* [IEE97].

[Dau94] M. Dauchet. Rewriting and tree automata. In H. Comon and J.-P. Jouannaud, editors, *Proc. Spring School on Theoretical Computer Science: Rewriting*, Lecture Notes in Computer Science, Odeillo, France, 1994. Springer Verlag.

[DCC95] M. Dauchet, A.-C. Caron, and J.-L. Coquidé. Reduction properties and automata with constraints. *Journal of Symbolic Computation*, 20:215–233, 1995.

[DGN+98] A. Degtyarev, Y. Gurevich, P. Narendran, M. Veanes, and A. Voronkov. The decidability of simultaneous rigid e-unification with one variable. In T. Nipkow, editor, *9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, 1998.

[DJ90] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter Rewrite Systems, pages 243–320. Elsevier, 1990.

[DM97] I. Durand and A. Middeldorp. Decidable call by need computations in term rewriting. In W. McCune, editor, *Proc. 14th Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 4–18. Springer Verlag, 1997.

[Don65] J. E. Doner. Decidability of the weak second-order theory of two successors. *Notices Amer. Math. Soc.*, 12:365–468, March 1965.

[Don70] J. E. Doner. Tree acceptors and some of their applications. *Journal of Comput. and Syst. Sci.*, 4:406–451, 1970.

[DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 242–248. IEEE Computer Society Press, 4–7 June 1990.

[DT92] M. Dauchet and S. Tison. Structural complexity of classes of tree languages. In M. Nivat and A. Podelski, editors, *Tree Automata and Languages*, pages 327–353. Elsevier Science, 1992.

[DTHL87] M. Dauchet, S. Tison, T. Heuillard, and P. Lescanne. Decidability of the confluence of ground term rewriting systems. In *Proceedings, Symposium on Logic in Computer Science*, pages 353–359. The Computer Society of the IEEE, 22–25 June 1987.

[DTT97] P. Devienne, J.-M. Talbot, and S. Tison. Solving classes of set constraints with tree automata. In G. Smolka, editor, *Proceedings of the 3^{th} International Conference on Principles and Practice of Constraint Programming*, volume 1330 of *Lecture Notes in Computer Science*, pages 62–76, oct 1997.

[Eng75] J. Engelfriet. Bottom-up and top-down tree transformations. a comparision. *Mathematical System Theory*, 9:198–231, 1975.

[Eng77] J. Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical System Theory*, 10:198–231, 1977.

[Eng78] J. Engelfriet. A hierarchy of tree transducers. In *Proceedings of the third Les Arbres en Algèbre et en Programmation*, pages 103–106, Lille, 1978.

[Eng82] J. Engelfriet. Three hierarchies of transducers. *Mathematical System Theory*, 15:95–125, 1982.

[ES78] J. Engelfriet and E.M. Schmidt. IO and OI II. *Journal of Comput. and Syst. Sci.*, 16:67–99, 1978.

[Esi83] Z. Esik. Decidability results concerning tree transducers. *Acta Cybernetica*, 5:303–314, 1983.

[EV91] J. Engelfriet and H. Vogler. Modular tree transducers. *Theorical Computer Science*, 78:267–303, 1991.

[EW67] S. Eilenberg and J. B. Wright. Automata in general algebras. *Information and Control*, 11(4):452–470, 1967.

[FSVY91] T. Frühwirth, E. Shapiro, M. Vardi, and E. Yardeni. Logic programs as types for logic programs. In *Proc. 6th IEEE Symp. Logic in Computer Science, Amsterdam*, pages 300–309, 1991.

[FV88] Z. Fülöp and S. Vágvölgyi. A characterization of irreducible sets modulo left-linear term rewiting systems by tree automata. Un type rr ??, Research Group on Theory of Automata, Hungarian Academy of Sciences, H-6720 Szeged, Somogyi u. 7. Hungary, 1988.

[FV89] Z. Fülöp and S. Vágvölgyi. Congruential tree languages are the same as recognizable tree languages–A proof for a theorem of D. kozen. *Bulletin of the European Association of Theoretical Computer Science*, 39, 1989.

[FV98] Z. Fülöp and H. Vögler. *Formal Models Based on Tree Transducers*. Monographs in Theoretical Computer Science. Springer Verlag, 1998.

[GB85] J. H. Gallier and R. V. Book. Reductions in tree replacement systems. *Theorical Computer Science*, 37(2):123–150, 1985.

[Gen97] T. Genet. Decidable approximations of sets of descendants and sets of normal forms - extended version. Technical Report RR-3325, Inria, Institut National de Recherche en Informatique et en Automatique, 1997.

[GJV98] H. Ganzinger, F. Jacquemard, and M. Veanes. Rigid reachability. In *Proc. ASIAN'98*, volume 1538 of *Lecture Notes in Computer Science*, pages 4–??, Berlin, 1998. Springer-Verlag.

[GMW97] H. Ganzinger, C. Meyer, and C. Weidenbach. Soft typing for ordered resolution. In W. McCune, editor, *Proc. 14th Conference on Automated Deduction*, volume 1249 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1997.

[Gou00] Jean Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *Proc. 15 IPDPS 2000 Workshops, Cancun, Mexico, May 2000*, volume 1800 of *Lecture Notes in Computer Science*, pages 977–984. Springer Verlag, 2000.

[GRS87] J. Gallier, S. Raatz, and W. Snyder. Theorem proving using rigid E-unification: Equational matings. In *Proc. 2nd IEEE Symp. Logic in Computer Science, Ithaca, NY*, June 1987.

[GS84] F. Gécseg and M. Steinby. *Tree Automata*. Akademiai Kiado, 1984.

[GS96] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer Verlag, 1996.

[GT95] R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24:157–176, 1995.

[GTT93] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints with negated subset relationships. In *Proceedings of the 34^{th} Symp. on Foundations of Computer Science*, pages 372–380, 1993. Full version in the LIFL Tech. Rep. IT-247.

[GTT99] R. Gilleron, S. Tison, and M. Tommasi. Set constraints and automata. *Information and Control*, 149:1 – 41, 1999.

[Gue83] I. Guessarian. Pushdowm tree automata. *Mathematical System Theory*, 16:237–264, 1983.

[Hei92] N. Heintze. *Set Based Program Analysis*. PhD thesis, Carnegie Mellon University, 1992.

[HJ90a] N. Heintze and J. Jaffar. A Decision Procedure for a Class of Set Constraints. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 42–51. IEEE Computer Society Press, 4–7 June 1990.

[HJ90b] N. Heintze and J. Jaffar. A finite presentation theorem for approximating logic programs. In *Proceedings of the 17^{th} ACM Symp. on Principles of Programming Languages*, pages 197–209, 1990. Full version in the IBM tech. rep. RC 16089 (#71415).

[HJ92] N. Heintze and J. Jaffar. An engine for logic program analysis. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science* [IEE92], pages 318–328.

[HL91] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems I. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 395–414. MIT Press, 1991. This paper was written in 1979.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 1979.

[IEE92] IEEE Computer Society Press. *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, 22–25 June 1992.

[IEE97] IEEE Computer Society Press. *Proceedings, 12*$^{\text{th}}$ *Annual IEEE Symposium on Logic in Computer Science*, 1997.

[Jac96] F. Jacquemard. Decidable approximations of term rewriting systems. In H. Ganzinger, editor, *Proceedings. Seventh International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, 1996.

[JM79] N. D. Jones and S. S. Muchnick. Flow Analysis and Optimization of LISP-like Structures. In *Proceedings of the 6th ACM Symposium on Principles of Programming Languages*, pages 244–246, 1979.

[Jon87] N. Jones. *Abstract interpretation of declarative languages*, chapter Flow analysis of lazy higher-order functional programs, pages 103–122. Ellis Horwood Ltd, 1987.

[Jr.76] William H. Joyner Jr. Resolution strategies as decision procedures. *Journal of the ACM*, 23(3):398–417, 1976.

[KFK97] Y. Kaji, T. Fujiwara, and T. Kasami. Solving a unification problem under constrained substitutions using tree automata. *Journal of Symbolic Computation*, 23(1):79–118, January 1997.

[Koz92] D. Kozen. On the Myhill-Nerode theorem for trees. *Bulletin of the European Association of Theoretical Computer Science*, 47:170–173, June 1992.

[Koz93] D. Kozen. Logical aspects of set constraints. In E. Börger, Y. Gurevich, and K. Meinke, editors, *Proceedings of Computer Science Logic*, volume 832 of *Lecture Notes in Computer Science*, pages 175–188, 1993.

[Koz95] D. Kozen. Rational spaces and set constraints. In *Proceedings of the 6th International Joint Conference on Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 42–61, 1995.

[Koz98] D. Kozen. Set constraints and logic programming. *Information and Computation*, 142(1):2–25, 1998.

[Kuc91] G. A. Kucherov. On relationship between term rewriting systems and regular tree languages. In R. Book, editor, *Proceedings. Fourth International Conference on Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 299–311, April 1991.

[Kui99] W. Kuich. Full abstract families of tree series i. In Juhani Karhumäki, Hermann A. Maurer, and Gheorghe Paun andy Grzegorz Rozenberg, editors, *Jewels are Forever*, pages 145–156. SV, 1999.

[Kui01] W. Kuich. Pushdown tree automata, algebraic tree systems, and algebraic tree series. *Information and Computation*, 165(1):69–99, 2001.

[KVW00] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching time model-checking. *Journal of the ACM*, 47(2):312–360, 2000.

[LD02] Denis Lugiez and Silvano DalZilio. Multitrees automata, presburger's constraints and tree logics. Technical Report 8, Laboratoire d'Informatique Fondamentale de Marseille, 2002.

[LM87] J.-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–318, September 1987.

[LM93] D. Lugiez and J.-L. Moysset. Complement problems and tree automata in AC-like theories. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *10th Annual Symposium on Theoretical Aspects of Computer Science*, volume 665 of *Lecture Notes in Computer Science*, pages 515–524, Würzburg, 25–27 February 1993.

[LM94] Denis Lugiez and Jean-Luc Moysset. Tree automata help one to solve equational formulae in ac-theories. *Journal of Symbolic Computation*, 18(4):297–318, 1994.

[Loh01] M. Lohrey. On the parallel complexity of tree automata. In *Proceedings of the 12th Conference on Rewriting and Applications*, pages 201–216, 2001.

[MGKW96] D. McAllester, R. Givan, D. Kozen, and C. Witty. Tarskian set constraints. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 138–141. IEEE Computer Society Press, 27–30 July 1996.

[Mis84] P. Mishra. Towards a Theory of Types in PROLOG. In *Proceedings of the 1st IEEE Symposium on Logic Programming*, pages 456–461, Atlantic City, 1984.

[MLM01] M. Murata, D. Lee, and M. Mani. Taxonomy of xml schema languages using formal language theory. In *In Extreme Markup Languages*, 2001.

[Mon81] J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, 1981.

[MS96] A. Mateescu and A. Salomaa. Aspects of classical language theory. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 175–246. Springer Verlag, 1996.

[Mur00] M. Murata. "Hedge Automata: a Formal Model for XML Schemata". Web page, 2000.

[MW67] J. Mezei and J. B. Wright. Algebraic automata and context-free sets. *Information and Control*, 11:3–29, 1967.

[Niv68] M. Nivat. *Transductions des langages de Chomsky*. Thèse d'etat, Paris, 1968.

[NP89] M. Nivat and A. Podelski. *Resolution of Equations in Algebraic Structures*, volume 1, chapter Tree monoids and recognizable sets of finite trees, pages 351–367. Academic Press, New York, 1989.

[NP93] J. Niehren and A. Podelski. Feature automata and recognizable sets of feature trees. In *Proceedings TAPSOFT'93*, volume 668 of *Lecture Notes in Computer Science*, pages 356–375, 1993.

[NP97] M. Nivat and A. Podelski. Minimal ascending and descending tree automata. *SIAM Journal on Computing*, 26(1):39–58, February 1997.

[NT99] T. Nagaya and Y. Toyama. Decidability for left-linear growing term rewriting systems. In M. Rusinowitch F. Narendran, editor, *10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, pages 256–270, Trento, Italy, 1999. Springer Verlag.

[Ohs01] Hitoshi Ohsaki. Beyond the regularity: Equational tree automata for associative and commutative theories. In *Proceedings of CSL 2001*, volume 2142 of *Lecture Notes in Computer Science*. Springer Verlag, 2001.

[Oya93] M. Oyamaguchi. NV-sequentiality: a decidable condition for call-by-need computations in term rewriting systems. *SIAM Journal on Computing*, 22(1):114–135, 1993.

[Pel97] N. Peltier. Tree automata and automated model building. *Fundamenta Informaticae*, 30(1):59–81, 1997.

[Pla85] D. A. Plaisted. Semantic confluence tests and completion method. *Information and Control*, 65:182–215, 1985.

[Pod92] A. Podelski. A monoid approach to tree automata. In Nivat and Podelski, editors, *Tree Automata and Languages, Studies in Computer Science and Artificial Intelligence 10*. North-Holland, 1992.

[PQ68] C. Pair and A. Quere. Définition et étude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968.

[Rab69] M. O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[Rab77] M. O. Rabin. *Handbook of Mathematical Logic*, chapter Decidable theories, pages 595–627. North Holland, 1977.

[Rao92] J.-C. Raoult. A survey of tree transductions. In M. Nivat and A. Podelski, editors, *Tree Automata and Languages*, pages 311–325. Elsevier Science, 1992.

[Rey69] J. C. Reynolds. Automatic Computation of Data Set Definition. *Information Processing*, 68:456–461, 1969.

[Sal73] A. Salomaa. *Formal Languages*. Academic Press, New York, 1973.

[Sal88] K. Salomaa. Deterministic tree pushdown automata and monadic tree rewriting systems. *Journal of Comput. and Syst. Sci.*, 37:367–394, 1988.

[Sal94] K. Salomaa. Synchronized tree automata. *Theorical Computer Science*, 127:25–51, 1994.

[Sei89] H. Seidl. Deciding equivalence of finite tree automata. In *Annual Symposium on Theoretical Aspects of Computer Science*, 1989.

[Sei90] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19, 1990.

[Sei92] H. Seidl. Single-valuedness of tree transducers is decidable in polynomial time. *Theorical Computer Science*, 106:135–181, 1992.

[Sei94a] H. Seidl. Equivalence of finite-valued tree transducers is decidable. *Mathematical System Theory*, 27:285–346, 1994.

[Sei94b] H. Seidl. Haskell overloading is DEXPTIME-complete. *Information Processing Letters*, 52(2):57–60, 1994.

[Sén97] G. Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *Lecture Notes in Computer Science*, pages 671–681, Bologna, Italy, 7–11 July 1997. Springer-Verlag.

[Sey94] F. Seynhaeve. Contraintes ensemblistes. Master's thesis, LIFL, 1994.

[Slu85] G. Slutzki. Alternating tree automata. *Theorical Computer Science*, 41:305–318, 1985.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. 5th ACM Symp. on Theory of Computing*, pages 1–9, 1973.

[Ste94] K. Stefansson. Systems of set constraints with negative constraints are nexptime-complete. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 137–141. IEEE Computer Society Press, 4–7 July 1994.

[SV95] G. Slutzki and S. Vagvolgyi. Deterministic top-down tree transducers with iterated look-ahead. *Theorical Computer Science*, 143:285–308, 1995.

[Tha70] J. W. Thatcher. Generalized sequential machines. *Journal of Comput. and Syst. Sci.*, 4:339–367, 1970.

[Tha73] J. W. Thatcher. Tree automata: an informal survey. In A.V. Aho, editor, *Currents in the theory of computing*, pages 143–178. Prentice Hall, 1973.

[Tho90] W. Thomas. *Handbook of Theoretical Computer Science*, volume B, chapter Automata on Infinite Objects, pages 134–191. Elsevier, 1990.

[Tho97] W. Thomas. Languages, automata and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–456. Springer Verlag, 1997.

[Tis89] S. Tison. Fair termination is decidable for ground systems. In *Proceedings, Third International Conference on Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pages 462–476, 1989.

[Tiu92] J. Tiuryn. Subtype inequalities. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science* [IEE92], pages 308–317.

[Tom92] M. Tommasi. Automates d'arbres avec tests d'égalité entre cousins germains. Mémoire de DEA, Univ. Lille I, 1992.

[Tom94] M. Tommasi. *Automates et contraintes ensemblistes*. PhD thesis, LIFL, 1994.

[Tra95] B. Trakhtenbrot. Origins and metamorphoses of the trinity: Logic, nets, automata. In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 26–29 June 1995.

[Tre96] R. Treinen. The first-order theory of one-step rewriting is undecidable. In H. Ganzinger, editor, *Proceedings. Seventh International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 276–286, 1996.

[TW65] J. W. Thatcher and J. B. Wright. Generalized finite automata. *Notices Amer. Math. Soc.*, 820, 1965. Abstract No 65T-649.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.

[Uri92] T. E. Uribe. Sorted Unification Using Set Constraints. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, New York, 1992.

[Vea97a] M. Veanes. On computational complexity of basic decision problems of finite tree automata. Technical report, Uppsala Computing Science Department, 1997.

[Vea97b] M. Veanes. *On simultaneous rigid E-unification*. PhD thesis, Computing Science Department, Uppsala University, Uppsala, Sweden, 1997.

[Zac79] Z. Zachar. The solvability of the equivalence problem for deterministic frontier-to-root tree transducers. *Acta Cybernetica*, 4:167–177, 1979.

Index