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.

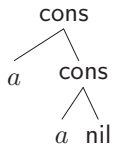
A ***ranked alphabet*** is a couple $(\mathcal{F}, Arity)$ where $\mathcal{F}$ is a finite set and $Arity$ is a mapping from $\mathcal{F}$ into N. The ***arity*** of a symbol $f \in \mathcal{F}$ is $Arity(f)$. The set of symbols of arity p is denoted by $\mathcal{F}_p$. Elements of arity $0, 1, \ldots p$ 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.

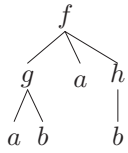
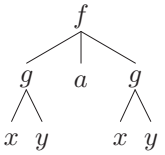
We denote by $\trianglerighteq$ the ***subterm ordering***, *i.e.*we write $t \trianglerighteq t'$ if t' is a subterm of t. We denote $t \triangleright t'$ if $t \trianglerighteq t'$ and $t \neq t'$.

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 $\{11, 12, 31, 32\}$; $t|_3 = h(b)$; $t[a]_3 = f(g(a, b), a, a)$; $\mathcal{H}eight(t) = 3$; $\mathcal{H}eight(t') = 2$; $\|t\| = 7$; $\|t'\| = 4$.

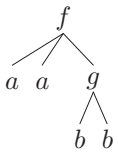
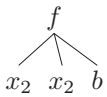
Substitutions

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 7

Alternating Tree Automata

7.1 Introduction

Complementation of non-deterministic tree (or word) automata requires a determinization step. This is due to an asymmetry in the definition. Two transition rules with the same left hand side can be seen as a single rule with a disjunctive right side. A run of the automaton on a given tree has to choose some member of the disjunction. Basically, determinization gathers the disjuncts in a single state.

Alternating automata restore some symmetry, allowing both disjunctions and conjunctions in the right hand sides. Then complementation is much easier: it is sufficient to exchange the conjunctions and the disjunction signs, as well as final and non-final states. In particular, nothing similar to determinization is needed.

This nice formalism is more concise. The counterpart is that decision problems are more complex, as we will see in Section 7.5.

There are other nice features: for instance, if we see a tree automaton as a finite set of monadic Horn clauses, then moving from non-deterministic to alternating tree automata consists in removing a very simple assumption on the clauses. This is explained in Section 7.6. In the same vein, removing another simple assumption yields *two-way* alternating tree automata, a more powerful device (yet not more expressive), as described in Section 7.6.3.

Finally, we also show in Section 7.2.3 that, as far as emptiness is concerned, tree automata correspond to alternating word automata on a single-letter alphabet, which shows the relationship between computations (runs) of a word alternating automaton and computations of a tree automaton.

7.2 Definitions and Examples

7.2.1 Alternating Word Automata

Let us start first with alternating *word automata*.

If Q is a finite set of states, $\mathcal{B}^+(Q)$ is the set of positive propositional formulas over the set of propositional variables Q. For instance, $q_1 \wedge (q_2 \vee q_3) \wedge (q_2 \vee q_4) \in \mathcal{B}^+(\{q_1, q_2, q_3, q_4\})$.

Alternating word automata are defined as deterministic word automata, except that the transition function is a mapping from $Q \times A$ to $\mathcal{B}^+(Q)$ instead of being a mapping from $Q \times A$ to Q. We assume a subset Q_0 of Q of *initial states* and a subset Q_f of Q of *final states*.

Example 59. Assume that the alphabet is $\{0,1\}$ and the set of states is $\{q_0, q_1, q_2, q_3, q_4, q'_1, q'_2\}$, $Q_0 = \{q_0\}$, $Q_f = \{q_0, q_1, q_2, q_3, q_4\}$ and the transitions are:

$q_0 0$	$\rightarrow$	$(q_0 \wedge q_1) \vee q'_1$	$q_0 1$	$\rightarrow$	q_0
$q_1 0$	$\rightarrow$	q_2	$q_1 1$	$\rightarrow$	true
$q_2 0$	$\rightarrow$	q_3	$q_2 1$	$\rightarrow$	q_3
$q_3 0$	$\rightarrow$	q_4	$q_3 1$	$\rightarrow$	q_4
$q_4 0$	$\rightarrow$	true	$q_4 1$	$\rightarrow$	true
$q'_1 0$	$\rightarrow$	q'_1	$q'_1 1$	$\rightarrow$	q'_2
$q'_2 0$	$\rightarrow$	q'_2	$q'_2 1$	$\rightarrow$	q'_1

A *run* of an alternating word automaton $\mathcal{A}$ on a word w is a finite tree ρ labeled with $Q \times \mathbb{N}$ such that:

- The root of ρ is labeled by some pair $(q, 0)$.
- If $\rho(p) = (q, i)$ and i is strictly smaller than the length of w, $w(i+1) = a$, $\delta(q, a) = \phi$, then there is a set $S = \{q_1, \ldots, q_n\}$ of states such that $S \models \phi$, positions $p1, \ldots, pn$ are the successor positions of p in ρ and $\rho(pj) = (q_j, i+1)$ for every $j = 1, ...n$.

The notion of satisfaction used here is the usual one in propositional calculus: the set S is the set of propositions assigned to true, while the propositions not belonging to S are assumed to be assigned to false. Therefore, we have the following:

- there is no run on w such that $w(i+1) = a$ for some i, $\rho(p) = (q, i)$ and $\delta(q, i) =$ false
- if $\delta(q, w(i+1)) =$ true and $\rho(p) = (q, i)$, then p can be a leaf node, in which case it is called a *success node*.
- All leaf nodes are either success nodes as above or labeled with some (q, n) such that n is the length of w.

A run of an alternating automaton is *successful* on w if and only if

- all leaf nodes are either success nodes or labeled with some (q, n), where n is the length of w, such that $q \in Q_f$.
- the root node $\rho(\epsilon) = (q_0, 0)$ with $q_0 \in Q_0$.

Example 60. Let us come back to Example 59. We show on Figure 7.1 two runs on the word 00101, one of which is successful.

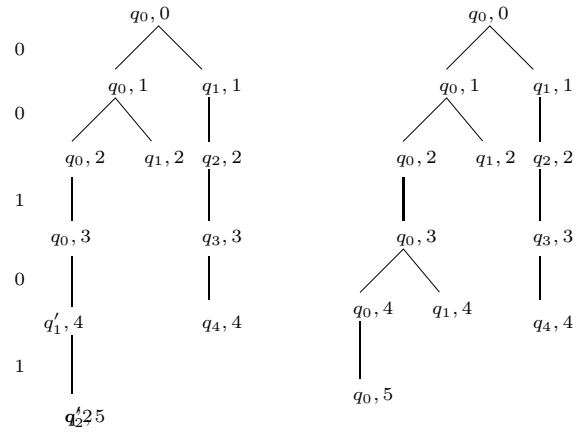


Figure 7.1: Two runs on the word 00101 of the automaton defined in Example 59. The right one is successful.

Note that non-deterministic automata are the particular case of alternating automata in which only disjunctions (no conjunctions) occur in the transition relation. In such a case, if there is a succesful run on w, then there is also a successful run, which is a string.

Note also that, in the definition of a run, we can always choose the set S to be a minimal satisfaction set: if there is a successful run of the automaton, then there is a successful one in which we always choose a minimal set S of states.

7.2.2 Alternating Tree Automata

Now, let us switch to alternating tree automata: the definitions are simple adaptations of the previous ones.

Definition 14. *An* alternating tree automaton *over $\mathcal{F}$ is a tuple $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$ where Q is a set of states, $I \subseteq Q$ is a set of initial states and Δ is a mapping from $Q \times \mathcal{F}$ to $\mathcal{B}^+(Q \times \mathbb{N})$ such that $\Delta(q, f) \in \mathcal{B}^+(Q \times \{1, \ldots, Arity(f)\})$ where $Arity(f)$ is the arity of f.*

Note that this definition corresponds to a *top-down* automaton, which is more convenient in the alternating case.

Definition 15. *Given a term $t \in T(\mathcal{F})$ and an alternating tree automaton $\mathcal{A}$ on $\mathcal{F}$, a* run *of $\mathcal{A}$ on t is a tree ρ on $Q \times \mathbb{N}^*$ such that $\rho(\varepsilon) = (q, \varepsilon)$ for some state q and*

> *if $\rho(\pi) = (q, p)$, $t(p) = f$ and $\delta(q, f) = \phi$, then there is a subset $S = \{(q_1, i_1), \ldots, (q_n, i_n)\}$ of $Q \times \{1, \ldots, Arity(f)\}$ such that $S \models \phi$, the successor positions of π in ρ are $\{\pi 1, \ldots, \pi n\}$ and $\rho(\pi \cdot j) = (q_j, p \cdot i_j)$ for every $j = 1..n$.*

A run ρ is successful *if $\rho(\varepsilon) = (q, \varepsilon)$ for some initial state $q \in I$.*

Note that (completely specified) non-deterministic top-down tree automata are the particular case of alternating tree automata. For a set of non-deterministic rules $q(f(x_1,\ldots,x_n)) \to f(q_1(x_1),\ldots,q_n(x_n))$, $Delta(q,f)$ is defined by:

$$\Delta(q,f) = \bigvee_{(q_1,\ldots,q_n)\in S} \bigwedge_{i=1}^{Arity(f)} (q_i,i)$$

Example 61. Consider the automaton on the alphabet $\{f(,),a,b\}$ whose transition relation is defined by:

Δ	f	a	b
q_2	$[((q_1,1)\wedge(q_2,2))\vee((q_1,2)\wedge(q_2,1))]\wedge(q_4,1)$	true	false
q_1	$((q_2,1)\wedge(q_2,2))\vee((q_1,2)\wedge(q_1,1))$	false	true
q_4	$((q_3,1)\wedge(q_3,2))\vee((q_4,1)\wedge(q_4,2))$	true	true
q_3	$((q_3,1)\wedge(q_2,2))\vee((q_4,1)\wedge(q_1,2))\wedge(q_5,1))$	false	true
q_5	false	true	false

Assume $I=\{q_2\}$. A run of the automaton on the term $t = f(f(b,f(a,b)),b)$. is depicted on Figure 7.2.

In the case of a non-deterministic top-down tree automaton, the different notions of a run coincide as, in such a case, the run obtained from Definition 15 on a tree t is a tree whose set of positions is the set of positions of t, possibly changing the ordering of sons.

Words over an alphabet A can be seen as trees over the set of unary function symbols A and an additional constant $\#$. For convenience, we read the words from right to left. For instance, $aaba$ is translate into the tree $a(b(a(a(\#))))$. Then an alternating word automaton $\mathcal{A}$ can be seen as an alternating tree automaton whose initial states are the final states of $\mathcal{A}$, the transitions are the same and there is additional rules $\delta(q^0,\#) = \text{true}$ for the initial state q^0 of $\mathcal{A}$ and $\delta(q,\#) = \text{false}$ for other states.

7.2.3 Tree Automata versus Alternating Word Automata

It is interesting to remark that, guessing the input tree, it is possible to reduce the emptiness problem for (non-deterministic, bottom-up) tree automata to the emptiness problem for an alternating word automaton on a single letter alphabet: assume that $\mathcal{A} = (Q,\mathcal{F},Q_f,\Delta)$ is a non-deterministic tree automaton, then construct the alternating word automaton on a one letter alphabet $\{a\}$ as follows: the states are $Q\times\mathcal{F}$, the initial states are $Q_f\times\mathcal{F}$ and the transition rules:

$$\delta((q,f),a) = \bigvee_{f(q_1,\ldots,q_n)\to q\in\Delta} \bigwedge_{i=1}^{n} \bigvee_{f_j\in\mathcal{F}} ((q_i,f_j),i)$$

Conversely, it is also possible to reduce the emptiness problem for an alternating word automaton over a one letter alphabet $\{a\}$ to the emptiness problem of non-deterministic tree automata, introducing a new function symbol for each conjunction; assume the formulas in disjunctive normal form (this can

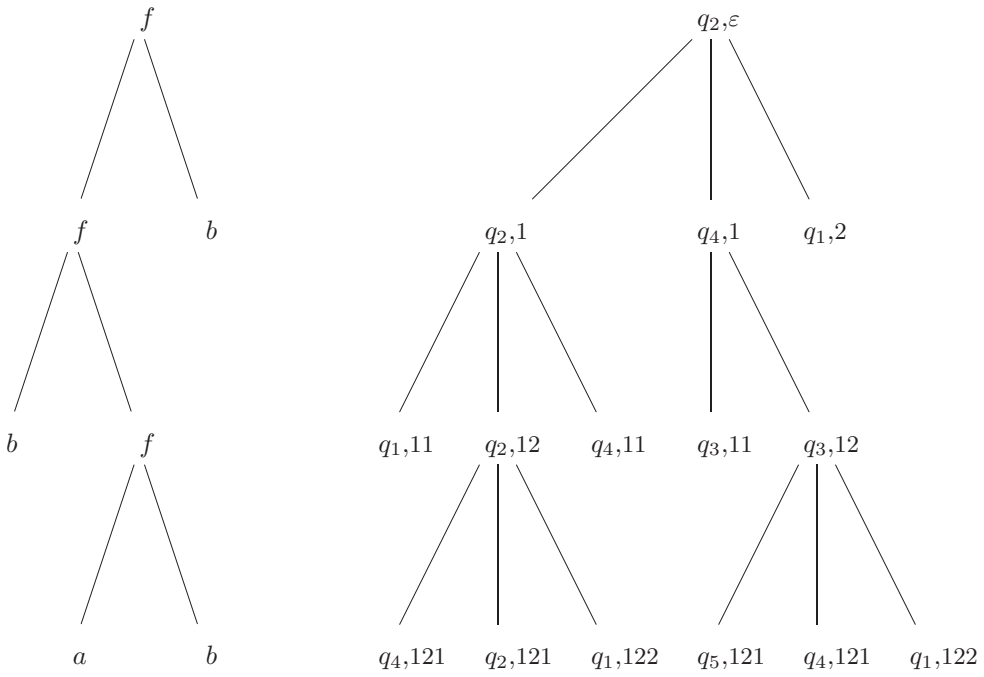


Figure 7.2: A run of an alternating tree automaton

be assumed w.l.o.g, see Exercise 82), then replace each transition $\delta(q,a) = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{k_i}(q_{i,j},i)$ with $f_i(q_{i,1},\ldots,q_{i,k_i}) \to q$.

7.3 Closure Properties

One nice feature of alternating automata is that it is very easy to perform the Boolean operations (for short, we confuse here the automaton and the language recognized by the automaton). First, we show that we can consider automata with only one initial state, without loss of generality.

Lemma 10. *Given an alternating tree automaton $\mathcal{A}$, we can compute in linear time an automaton $\mathcal{A}'$ with only one initial state and which accepts the same language as $\mathcal{A}$.*

Proof. Add one state q^0 to $\mathcal{A}$, which will become the only initial state, and the transitions:

$$\delta(q^0,f) = \bigvee_{q\in I} \delta(q,f)$$

□

Proposition 47. *Union, intersection and complement of alternating tree automata can be performed in linear time.*

Proof. We consider w.l.o.g. automata with only one initial state. Given $\mathcal{A}_1$ and $\mathcal{A}_2$, with a disjoint set of states, we compute an automaton $\mathcal{A}$ whose states are those of $\mathcal{A}_1$ and $\mathcal{A}_2$ and one additional state q^0. Transitions are those of $\mathcal{A}_1$ and $\mathcal{A}_2$ plus the additional transitions for the union:

$$\delta(q^0,f) = \delta_1(q_1^0,f) \vee \delta_2(q_2^0,f)$$

where q_1^0, q_2^0 are the initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. For the intersection, we add instead the transitions:

$$\delta(q^0,f) = \delta_1(q_1^0,f) \wedge \delta_2(q_2^0,f)$$

Concerning the complement, we simply exchange $\wedge$ and $\vee$ (resp. true and false) in the transitions. The resulting automaton $\widetilde{\mathcal{A}}$ will be called the *dual automaton* in what follows.

The proof that these constructions are correct for union and intersection are left to the reader. Let us only consider here the complement.

We prove, by induction on the size of t that, for every state q, t is accepted either by $\mathcal{A}$ or $\widetilde{\mathcal{A}}$ in state q and not by both automata.

If t is a constant a, then $\delta(q,a)$ is either true or false. If $\delta(q,a) =$ true, then $\widetilde{\delta}(q,a) =$ false and t is accepted by $\mathcal{A}$ and not by $\widetilde{\mathcal{A}}$. The other case is symmetric.

Assume now that $t = f(t_1,\ldots,t_n)$ and $\delta(q,f) = \phi$. Let S be the set of pairs (q_j,i_j) such that t_{i_j} is accepted from state q_j by $\mathcal{A}$. t is accepted by $\mathcal{A}$, iff $S \models \phi$. Let $\widetilde{S}$ be the complement of S in $Q \times [1..n]$. By induction hypothesis, $(q_j,i) \in \widetilde{S}$ iff t_i is accepted in state q_j by $\widetilde{\mathcal{A}}$.

We show that $\widetilde{S} \models \widetilde{\phi}$ iff $S \not\models \phi$. ($\widetilde{\phi}$ is the dual formula, obtained by exchanging $\wedge$ and $\vee$ on one hand and true and false on the other hand in ϕ). We

show this by induction on the size of ϕ: if ϕ is true (resp. false), then $S \models \phi$ and $\widetilde{S} \not\models \widetilde{\phi}$ (resp. $\widetilde{S} = \emptyset$) and the result is proved. Now, let, ϕ be, *e.g.*, $\phi_1 \wedge \phi_2$. $S \not\models \phi$ iff either $S \not\models \phi_1$ or $S \not\models \phi_2$, which, by induction hypothesis, is equivalent to $\widetilde{S} \models \widetilde{\phi_1}$ or $\widetilde{S} \models \widetilde{\phi_2}$. By construction, this is equivalent to $\widetilde{S} \models \widetilde{\phi}$. The case $\phi = \phi_1 \vee \phi_2$ is similar.

Now t is accepted in state q by $\mathcal{A}$ iff $S \models \phi$ iff $\widetilde{S} \not\models \widetilde{\phi}$ iff t not accepted in state q by $\widetilde{\mathcal{A}}$. □

7.4 From Alternating to Deterministic Automata

The expressive power of alternating automata is exactly the same as finite (bottom-up) tree automata.

Theorem 54. *If $\mathcal{A}$ is an alternating tree automaton, then there is a finite deterministic bottom-up tree automaton $\mathcal{A}'$ which accepts the same language. $\mathcal{A}'$ can be computed from $\mathcal{A}$ in deterministic exponential time.*

Proof. Assume $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$, then $\mathcal{A}' = (2^Q, \mathcal{F}, Q_f, \delta)$ where $Q_f = \{S \in 2^Q \mid S \cap I \neq \emptyset\}$ and δ is defined as follows:

$$f(S_1, \ldots, S_n) \rightarrow \{q \in Q \mid S_1 \times \{1\} \cup \ldots \cup S_n \times \{n\} \models \Delta(q, f)\}$$

A term t is accepted by $\mathcal{A}'$ in state S iff t is accepted by $\mathcal{A}$ in all states $q \in S$. This is proved by induction on the size of t: if t is a constant, then t is accepted in all states q such that $\Delta(q, t) = \text{true}$. Now, if $t = f(t_1, \ldots, t_n)$ we let $S_1, \ldots, S_n$ are the set of states in which $t_1, \ldots, t_n$ are respectively accepted by $\mathcal{A}$. t is accepted by $\mathcal{A}$ in a state q iff there is $S_0 \subseteq Q \times \{1, \ldots, n\}$ such that $S_0 \models \Delta(q, f)$ and, for every pair $(q_i, j) \in S_0$, t_j is accepted in q_i. In other words, t is accepted by $\mathcal{A}$ in state q iff there is an $S_0 \subseteq S_1 \times \{1\} \cup \ldots \cup S_n \times \{n\}$ such that $S_0 \models \Delta(q, f)$, which is in turn equivalent to $S_1 \times \{1\} \cup \ldots \cup S_n \times \{n\} \models \Delta(q, f)$. We conclude by an application of the induction hypothesis. □

Unfortunately the exponential blow-up is unavoidable, as a consequence of Proposition 47 and Theorems 14 and 11.

7.5 Decision Problems and Complexity Issues

Theorem 55. *The emptiness problem and the universality problem for alternating tree automata are DEXPTIME-complete.*

Proof. The DEXPTIME membership is a consequence of Theorems 11 and 54.

The DEXPTIME-hardness is a consequence of Proposition 47 and Theorem 14. □

The membership problem (given t and $\mathcal{A}$, is t accepted by $\mathcal{A}$?) can be decided in polynomial time. This is left as an exercise.

7.6 Horn Logic, Set Constraints and Alternating Automata

7.6.1 The Clausal Formalism

Viewing every state q as a unary predicate symbol P_q, tree automata can be translated into Horn clauses in such a way that the language recognized in state q is exactly the interpretation of P_q in the least Herbrand model of the set of clauses.

There are several advantages of this point of view:

- Since the logical setting is declarative, we don't have to distinguish between top-down and bottom-up automata. In particular, we have a definition of bottom-up alternating automata for free.
- Alternation can be expressed in a simple way, as well as push and pop operations, as described in the next section.
- There is no need to define a run (which would correspond to a proof in the logical setting)
- Several decision properties can be translated into decidability problems for such clauses. Typically, since all clauses belong to the *monadic fragment*, there are decision procedures e.g. relying on ordered resolution strategies.

There are also weaknesses: complexity issues are harder to study in this setting. Many constructive proofs, and complexity results have been obtained with tree automata techniques.

Tree automata can be translated into Horn clauses. With a tree automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ is associated the following set of Horn clauses:

$$P_q(f(x_1, \ldots, x_n)) \leftarrow P_{q_1}(x_1), \ldots, P_{q_n}(x_n)$$

if $f(q_1, \ldots, q_n) \to q \in \Delta$. The language accepted by the automaton is the union of interpretations of P_q, for $q \in Q_f$, in the least Herbrand model of clauses.

Also, alternating tree automata can be translated into Horn clauses. Alternation can be expressed by variable sharing in the body of the clause. Consider an alternating tree automaton $(Q, \mathcal{F}, I, \Delta)$. Assume that the transitions are in disjunctive normal form (see Exercise 82). With a transition $\Delta(q, f) = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k_i} (q_j, i_j)$ is associated the clauses

$$P_q(f(x_1, \ldots, x_n)) \leftarrow \bigwedge_{j=1}^{k_i} P_{q_j}(x_{i_j})$$

We can also add ϵ-transitions, by allowing clauses

$$P(x) \leftarrow Q(x)$$

In such a setting, automata with equality constraints between brothers, which are studied in Section 4.3, are simply an extension of the above class of Horn clauses, in which we allow repeated variables in the head of the clause.

Allowing variable repetition in an arbitrary way, we get alternating automata with contraints between brothers, a class of automata for which emptiness is decidable in deterministic exponential time. (It is expressible in Löwenheim's class with equality, also called sometimes the *monadic class*).

Still, for tight complexity bounds, for closure properties (typically by complementation) of automata with equality tests between brothers, we refer to Section 4.3. Note that it is not easy to derive the complexity results obtained with tree automata techniques in a logical framework.

7.6.2 The Set Constraints Formalism

We introduced and studied general set constraints in Chapter 5. Set constraints and, more precisely, *definite set constraints* provide with an alternative description of tree automata.

Definite set constraints are conjunctions of inclusions

$$e \subseteq t$$

where e is a set expression built using function application, intersection and variables and t is a term set expression, constructed using function application and variables only.

Given an assignment σ of variables to subsets of $T(\mathcal{F})$, we can interpret the set expressions as follows:

$$\begin{array}{rcl} [\![f(e_1,\ldots,e_n)]\!]_\sigma & \stackrel{\text{def}}{=} & \{f(t_1,\ldots,t_n) \mid t_i \in [\![e_i]\!]_\sigma\} \\ [\![e_1 \cap e_2]\!]_\sigma & \stackrel{\text{def}}{=} & [\![e_1]\!]_\sigma \cap [\![e_2]\!]_\sigma \\ [\![X]\!]_\sigma & \stackrel{\text{def}}{=} & X\sigma \end{array}$$

Then σ is a solution of a set constraint if inclusions hold for the corresponding interpretation of expressions.

When we restrict the left members of inclusions to variables, we get another formalism for alternating tree automata: such set constraints have always a least solution, which is accepted by an alternating tree automaton. More precisely, we can use the following translation from the alternating automaton $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$: assume again that the transitions are in disjunctive normal form (see Exercise 82) and construct, the inclusion constraints

$$f(X_{1,f,q,d},\ldots,X_{n,f,q,d}) \subseteq X_q$$

$$\bigcap_{(q',j)\in d} X_{q'} \subseteq X_{j,f,q,d}$$

for every $(q, f) \in Q \times \mathcal{F}$ and d a disjunct of $\Delta(q, f)$. (An intersection over an empty set has to be understood as the set of all trees).

Then, the language recognized by the alternating tree automaton is the union, for $q \in I$, of $X_q\sigma$ where σ is the least solution of the constraint.

Actually, we are constructing the constraint in exactly the same way as we constructed the clauses in the previous section. When there is no alternation, we get an alternative definition of non-deterministic automata, which corresponds to the algebraic characterization of Chapter 2.

Conversely, if all right members of the definite set constraint are variables, it is not difficult to construct an alternating tree automaton which accepts the least solution of the constraint (see Exercise 85).

7.6.3 Two Way Alternating Tree Automata

Definite set constraints look more expressive than alternating tree automata, because inclusions

$$X \subseteq f(Y, Z)$$

cannot be directly translated into automata rules.

We define here *two-way tree automata* which will easily correspond to definite set constraints on one hand and allow to simulate, e.g., the behavior of standard pushdown word automata.

It is convenient here to use the clausal formalism in order to define such automata. A clause

$$P(u) \leftarrow P_1(x_1), \ldots, P_n(x_n)$$

where u is a linear, non-variable term and $x_1, \ldots, x_n$ are (not necessarily distinct) variables occurring in u, is called a *push clause*. A clause

$$P(x) \leftarrow Q(t)$$

where x is a variable and t is a linear term, is called a *pop clause*. A clause

$$P(x) \leftarrow P_1(x), \ldots, P_n(x)$$

is called an *alternating clause* (or an *intersection clause*).

Definition 16. *An* alternating two-way tree automaton *is a tuple* $(Q, Q_f, \mathcal{F}, C)$ *where* Q *is a finite set of unary function symbols,* Q_f *is a subset of* Q *and* C *is a finite set of clauses each of which is a push clause, a pop clause or an alternating clause.*

Such an automaton *accepts* a tree t if t belongs to the interpretation of some $P \in Q_f$ in the least Herbrand model of the clauses.

Example 62. Consider the following alternating two-way automaton on the alphabet $\mathcal{F} = \{a, f(,)\}$:

$$\begin{array}{lrcl}
1. & P_1(f(f(x_1, x_2), x_3)) & \leftarrow & P_2(x_1), P_2(x_2), P_2(x_3) \\
2. & P_2(a) & & \\
3. & P_1(f(a, x)) & \leftarrow & P_2(x) \\
4. & P_3(f(x, y)) & \leftarrow & P_1(x), P_2(y) \\
5. & P_4(x) & \leftarrow & P_3(x), P_1(x) \\
6. & P_2(x) & \leftarrow & P_4(f(x, y)) \\
7. & P_1(y) & \leftarrow & P_4(f(x, y))
\end{array}$$

The clauses 1,2,3,4 are push clauses. Clause 5 is an alternating clause and clauses 6,7 are pop clauses.

If we compute the least Herbrand model, we successively get for the five first steps:

step	1	2	3	4	5
P_1		$f(a,a), f(f(a,a),a)$			a
P_2	a				$f(a,a)$
P_3			$f(f(a,a),a), f(f(f(a,a),a),a)$		
P_4				$f(f(a,a),a)$	

These automata are often convenient in expressing some problems (see the exercises and bibliographic notes). However they do not increase the expressive power of (alternating) tree automata:

Theorem 56. *For every alternating two-way tree automaton, it is possible to compute in deterministic exponential time a tree automaton which accepts the same language.*

We do not prove the result here (see the bibliographic notes instead). A simple way to compute the equivalent tree automaton is as follows: first flatten the clauses, introducing new predicate symbols. Then saturate the set of clauses, using ordered resolution (*w.r.t.*subterm ordering) and keeping only non-subsumed clauses. The saturation process terminates in exponential time. The desired automaton is obtained by simply keeping only the push clauses of this resulting set of clauses.

Example 63. Let us come back to Example 62 and show how we get an equivalent finite tree automaton.

First flatten the clauses: Clause 1 becomes

$$\begin{array}{lrcl} 1. & P_1(f(x,y)) & \leftarrow & P_5(x), P_2(y) \\ 8. & P_5(f(x,y)) & \leftarrow & P_2(x), P_2(y) \end{array}$$

Now we start applying resolution;

$$\begin{array}{lrrcl} \text{From } 4+5: & 9. & P_4(f(x,y)) & \leftarrow & P_1(x), P_2(y), P_1(f(x,y)) \\ \text{Form } 9+6: & 10. & P_2(x) & \leftarrow & P_1(x), P_2(y) \\ \text{From } 10+2: & 11. & P_2(x) & \leftarrow & P_1(x) \end{array}$$

Clause 11 subsumes 10, which is deleted.

$$\begin{array}{lrrcl} \text{From } 9+7: & 12. & P_1(y) & \leftarrow & P_1(x), P_2(y) \\ \text{From } 12+1: & 13. & P_1(y) & \leftarrow & P_2(y), P_5(x), P_2(z) \\ \text{From } 13+8: & 14. & P_1(y) & \leftarrow & P_2(y), P_2(x_1), P_2(x_2), P_2(z) \end{array}$$

Clause 14. can be simplified and, by superposition with 2. we get

$$\text{From } 14+2: \quad 15. \quad P_1(y) \quad \leftarrow \quad P_2(y)$$

At this stage, from 11. and 15. we have $P_1(x) \leftrightarrow P_2(x)$, hence, for simplicity, we will only consider P_1, replacing every occurrence of P_2 with P_1.

$$\begin{array}{lrrcl} \text{From } 1+5: & 16. & P_4(f(x,y)) & \leftarrow & P_3(f(x,y)), P_5(x), P_1(y) \\ \text{From } 1+9: & 17. & P_4(f(x,y)) & \leftarrow & P_1(x), P_1(y), P_5(x) \\ \text{From } 2+5: & 18. & P_4(a) & \leftarrow & P_3(a) \\ \text{From } 3+5: & 19. & P_4(f(a,x)) & \leftarrow & P_3(f(a,x)), P_1(x) \\ \text{From } 3+9: & 20. & P_4(f(a,x)) & \leftarrow & P_1(x), P_1(a) \\ \text{From } 2+20: & 21. & P_4(f(a,x)) & \leftarrow & P_1(x) \end{array}$$

Clause 21. subsumes both 20 and 19. These two clauses are deleted.

$$\begin{array}{lrcl}
\text{From } 5+6: & 22. & P_1(x) & \leftarrow & P_3(f(x,y)), P_1(f(x,y)) \\
\text{From } 5+7: & 23. & P_1(y) & \leftarrow & P_3(f(x,y)), P_1(f(x,y)) \\
\text{From } 16+6: & 24. & P_1(x) & \leftarrow & P_3(f(x,y)), P_5(x), P_1(y) \\
\text{From } 23+1: & 25. & P_1(y) & \leftarrow & P_3(f(x,y)), P_5(x), P_1(y)
\end{array}$$

Now every new inference yields a redundant clause and the saturation terminates, yielding the automaton:

$$\begin{array}{rrcl}
1. & P_1(f(x,y)) & \leftarrow & P_5(x), P_2(y) \\
2. & P_1(a) & & \\
3. & P_1(f(a,x)) & \leftarrow & P_1(x) \\
4. & P_3(f(x,y)) & \leftarrow & P_1(x), P_1(y) \\
8. & P_5(f(x,y)) & \leftarrow & P_1(x), P_1(y) \\
11. & P_1(x) & \leftarrow & P_1(y) \\
15. & P_2(x) & \leftarrow & P_1(x) \\
21. & P_4(f(a,x)) & \leftarrow & P_1(x)
\end{array}$$

Of course, this automaton can be simplified: P_1 and P_2 accept all terms in $T(\mathcal{F})$.

It follows from Theorems 56, 55 and 11 that the emptiness problem (resp. universality problems) are DEXPTIME-complete for two-way alternating automata.

7.6.4 Two Way Automata and Definite Set Constraints

There is a simple reduction of two-way automata to definite set constraints:

A push clause $P(f(x_1,\ldots,x_n)) \leftarrow P_1(x_{i_1}),\ldots,P_n(x_{i_n})$ corresponds to an inclusion constraint

$$f(e_1,\ldots,e_n) \subseteq X_P$$

where each e_j is the intersection, for $i_k = j$ of the variables X_{P_k}. A (conditional) pop clause $P(x_i) \leftarrow Q(f(x_1,\ldots,x_n)), P_1(x_1),\ldots P_k(x_k)$ corresponds to

$$f(e_1,\ldots,e_n) \cap X_Q \subseteq f(\top,\ldots,X_P,\top,\ldots)$$

where, again, each e_j is the intersection, for $i_k = j$ of the variables X_{P_k} and $\top$ is a variable containing all term expressions. Intersection clauses $P(x) \leftarrow Q(x), R(x)$ correspond to constraints

$$X_Q \cap X_R \subseteq X_P$$

Conversely, we can translate the definite set constraints into two-way automata, with additional restrictions on some states. We cannot do better since a definite set constraint could be unsatisfiable.

Introducing auxiliary variables, we only have to consider constraints:

1. $f(X_1,\ldots,X_n) \subseteq X$,
2. $X_1 \cap \ldots \cap X_n \subseteq X$,
3. $X \subseteq f(X_1,\ldots,X_n)$.

The first constraints are translated to push clauses, the second kind of constraints is translated to intersection clauses. Consider the last constraints. It can be translated into the pop clauses:

$$P_{X_i}(x_i) \leftarrow P_X(f(x_1, \ldots, x_n))$$

with the provision that all terms in P_X are headed with f.

Then the procedure which solves definite set constraints is essentially the same as the one we sketched for the proof of Theorem 56, except that we have to add unit negative clauses which may yield failure rules

---

Example 64. Consider the definite set constraint

$$f(X,Y) \cap X \subseteq f(Y,X), \quad f(a,Y) \subseteq X, \quad a \subseteq Y, \quad f(f(Y,Y),Y) \subseteq X$$

Starting from this constraint, we get the clauses of Example 62, with the additional restriction

$$26.\ \neg P_4(a)$$

since every term accepted in P_4 has to be headed with f.

If we saturate this constraint as in Example 63, we get the same clauses, of course, but also negative clauses resulting from the new negative clause:

$$\text{From } 26 + 18\ \ 27.\ \neg P_3(a)$$

And that is all: the constraint is satisfiable, with a minimal solution described by the automaton resulting from the computation of Example 63.

7.6.5 Two Way Automata and Pushdown Automata

Two-way automata, though related to pushdown automata, are quite different. In fact, for every pushdown automaton, it is easy to construct a two-way automaton which accepts the possible contents of the stack (see Exercise 86). However, two-way tree (resp. word) automata have the same expressive power as standard tree (resp. word) automata: they only accept regular languages, while pushdown automata accept context-free languages, which strictly contain regular languages.

Note still that, as a corollary of Theorem 56, the language of possible stack contents in a pushdown automaton is regular.

7.7 An (other) example of application

Two-way automata naturally arise in the analysis of cryptographic protocols. In this context, terms are constructed using the function symbols $\{\_\}\_$ (binary encryption symbols), $<\_,\_>$ (pairing) and constants (and other symbols which are irrelevant here). The so-called Dolev-Yao model consists in the deduction rules of Figure 7.3, which express the capabilities of an intruder. For simplicity, we only consider here symmetric encryption keys, but there are similar rules for public key cryptosystems. The rules basically state that an intruder can encrypt

$$\text{Pairing}\quad \frac{u \quad v}{< u, v >} \qquad \text{Encryption}\quad \frac{u \quad v}{\{u\}_v}$$

$$\text{Unpairing L}\quad \frac{< u, v >}{u} \qquad \text{Unpairing R}\quad \frac{< u, v >}{v}$$

$$\text{Decryption}\quad \frac{\{u\}_v \quad v}{u}$$

Figure 7.3: The Dolev-Yao intruder capabilities

a known message with a known key, can decrypt a known message encrypted with k, provided he knows k and can form and decompose pairs.

It is easy to construct a two-way automaton which, given a regular set of terms R, accepts the set of terms that can be derived by an intruder using the rules of Figure 7.3 (see Exercise 87).

7.8 Exercises

Exercise 82. Show that, for every alternating tree automaton, it is possible to compute in polynomial time an alternating tree automaton which accepts the same language and whose transitions are in disjunctive normal form, *i.e.*each transition has the form

$$\delta(q, f) = \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k_i} (q_j, l_j)$$

Exercise 83. Show that the membership problem for alternating tree automata can be decided in polynomial time.

Exercise 84. An alternating automaton is *weak* if there is an ordering on the set of states such that, for every state q and every function symbol f, every state q' occurring in $\delta(q, f)$ satisfies $q' \leq q$.

Prove that the emptiness of weak alternating tree automata is in PTIME.

Exercise 85. Given a definite set constraint whose all right hand sides are variables, show how to construct (in polynomial time) k alternating tree automata which accept respectively $X_1\sigma, \ldots, X_k\sigma$ where σ is the least solution of the constraint.

Exercise 86. A *pushdown* automaton on words is a tuple $(Q, Q_f, A, \Gamma, \delta)$ where Q is a finite set of states, $Q_f \subseteq Q$, A is a finite alphabet of input symbols, Γ is a finite alphabet of stack symbols and δ is a transition relation defined by rules: $qa \xrightarrow{w} q'$ and $qa \xrightarrow{w^{-1}} q'$ where $q, q' \in Q$, $a \in A$ and $w, w' \in \Gamma^*$.

A configuration is a pair of a state and a word $\gamma \in \Gamma^*$. The automaton may move when reading a, from (q, γ) to (q', γ') if either there is a transition $qa \xrightarrow{w} q'$ and $\gamma' = w \cdot \gamma$ or there is a transition $qa \xrightarrow{w^{-1}} q'$ and $\gamma = w \cdot \gamma'$.

1. Show how to compute (in polynomial time) a two-way automaton which accepts w in state q iff the configuration (q, w) is reachable.
2. This can be slightly generalized considering alternating pushdown automata: now assume that the transitions are of the form: $qa \xrightarrow{w} \phi$ and $qa \xrightarrow{w^{-1}} \phi$ where $\phi \in \mathcal{B}^+(Q)$. Give a definition of a run and of an accepted word, which is consistent with both the definition of a pushdown automaton and the definition of an alternating automaton.
3. Generalize the result of the first question to alternating pushdown automata.
4. Generalize previous questions to tree automata.

Exercise 87. Given a finite tree automaton A over the alphabet $\{a, \{\_\}\_, < \_, \_ >\}$, construct a two-way tree automaton which accepts the set of terms t which can be deduced by the rule of Figure 7.3 and the rule

$$\frac{}{t} \quad \text{If } t \text{ is accepted by } A$$

7.9 Bibliographic Notes

Alternation has been considered for a long time as a computation model, e.g. for Turing machines. The seminal work in this area is [CKS81], in which the relationship between complexity classes defined using (non)-deterministic machines and alternating machines is studied.

Concerning tree automata, alternation has been mainly considered in the case of infinite trees. This is especially useful to keep small representations of automata associated with temporal logic formulas, yielding optimal model-checking algorithms [KVW00].

Two-way automata and their relationship with clauses have been first considered in [FSVY91] for the analysis of logic programs. They also occur naturally in the context of definite set constraints, as we have seen (the completion mechanisms are presented in, e.g., [HJ90a, CP97]), and in the analysis of cryptographic protocols [Gou00].

There several other definitions of two-way tree automata. We can distinguish between two-way automata which have the same expressive power as regular languages and what we refer here to pushdown automata, whose expressive power is beyond regularity.

Decision procedures based on ordered resolution strategies could be found in [Jr.76].

Alternating automata with contraints between brothers define a class of languages expressible in Löwenheim's class with equality, also called sometimes the *monadic class*. See for instance [BGG97].

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