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.

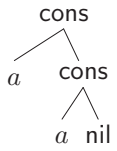
A ***ranked alphabet*** is a couple $(\mathcal{F}, Arity)$ where $\mathcal{F}$ is a finite set and $Arity$ is a mapping from $\mathcal{F}$ into N. The ***arity*** of a symbol $f \in \mathcal{F}$ is $Arity(f)$. The set of symbols of arity p is denoted by $\mathcal{F}_p$. Elements of arity $0, 1, \ldots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

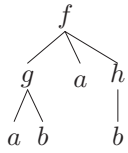
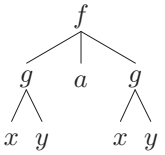
We denote by $\trianglerighteq$ the ***subterm ordering***, *i.e.*we write $t \trianglerighteq t'$ if t' is a subterm of t. We denote $t \triangleright t'$ if $t \trianglerighteq t'$ and $t \neq t'$.

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

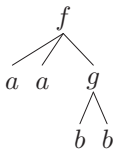
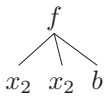
Substitutions

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 4

Automata with Constraints

4.1 Introduction

A typical example of a language which is not recognized by a finite tree automaton is the set of terms $\{f(t,t) \mid t \in T(\mathcal{F})\}$. The reason is that the two sons of the root are recognized independently and only a fixed finite amount of information can be carried up to the root position, whereas t may be arbitrarily large. Therefore, as seen in the application section of the previous chapter, this imposes some linearity conditions, typically when automata techniques are applied to rewrite systems or to sort constraints. The shift from linear to non linear situations can also be seen as a generalization from tree automata to DAG (directed acyclic graphs) automata. This is the purpose of the present chapter: how is it possible to extend the definitions of tree automata in order to carry over the applications of the previous chapter to (some) non-linear situations?

Such an extension has been studied in the early 80's by M. Dauchet and J. Mongy. They define a class of automata which (when working in a top-down fashion) allow duplications. Considering bottom-up automata, this amounts to check equalities between subtrees. This yields the *RATEG class* . This class is not closed under complement. If we consider its closure, we get the class of automata with equality and disequality constraints. This class is studied in Section 4.2.

Unfortunately, the emptiness problem is undecidable for the class RATEG (and hence for automata with equality and disequality constraints).
Several decidable subclasses have been studied in the literature. The most remarkable ones are

- The class of *automata with constraints between brothers* which, roughly, allows equality (or disequality) tests only between positions with the same ancestors. For instance, the set of terms $f(t,t)$ is recognized by such an automaton. This class is interesting because all properties of tree automata carry over this extension and hence most of the applications of tree automata can be extended, replacing linearity conditions with such restrictions on non-linearities.

  We study this class in Section 4.3.

- The class of *reduction automata* which, roughly, allows arbitrary disequal-

ity constraints but only a fixed finite amount of equality constraints on each run of the automaton. For instance the set of terms $f(t,t)$ also belongs to this class. Though closure properties have to be handled with care (with the definition sketched above, the class is not closed by complement), reduction automata are interesting because for example the set of irreducible terms (*w.r.t.*an arbitrary, possibly non-linear rewrite system) is recognized by an reduction automaton. Then the decidability of ground reducibility is a direct consequence of emptiness decidability for reduction automata. There is also a logical counterpart: the *reducibility theory* which is presented in the linear case in the previous chapter and which can be shown decidable in the general case using a similar technique.

Reduction automata are studied in Section 4.4.

We also consider in this chapter automata with arithmetic constraints. They naturally appear when some function symbols are assumed to be associative and commutative (AC). In such a situation, the sons of an AC symbol can be permuted and the relevant information is then the number of occurrences of the same subtree in the multisets of sons. These integer variables (number of occurrences) are subject to arithmetic constraints which must belong to a decidable fragment of arithmetic in order to keep closure and decidability properties.

4.2 Automata with Equality and Disequality Constraints

4.2.1 The Most General Class

An ***equality constraint*** (resp. a ***disequality constraint***) is a predicate on $T(\mathcal{F})$ written $\pi = \pi'$ (resp. $\pi \neq \pi'$) where $\pi, \pi' \in \{1, \ldots, k\}^*$. Such a predicate is satisfied on a term t, which we write $t \models \pi = \pi'$, if $\pi, \pi' \in \mathcal{P}os(t)$ and $t|_\pi = t|_{\pi'}$ (resp. $\pi \neq \pi'$ is satisfied on t if $\pi = \pi'$ is not satisfied on t).

The satisfaction relation $\models$ is extended as usual to any Boolean combination of equality and disequality constraints. The empty conjunction and disjunction are respectively written $\bot$ (false) and $\top$ (true).

An ***automaton with equality and disequality constraints*** is a tuple $(Q, \mathcal{F}, Q_f, \Delta)$ where $\mathcal{F}$ is a finite ranked alphabet, Q is a finite set of states, Q_f is a subset of Q of ***finite states*** and Δ is a set of transition rules of of the form

$$f(q_1, \ldots, q_n) \xrightarrow{c} q$$

where $f \in \mathcal{F}$, $q_1, \ldots, q_n, q \in Q$, and c is a Boolean combination of equality (and disequality) constraints. The state q is called ***target state*** in the above transition rule.

We write for short AWEDC the class of automata with equality and disequality constraints.

Let $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta) \in AWEDC$. The move relation $\rightarrow_\mathcal{A}$ is defined by as for NFTA modulo the satisfaction of equality and disequality constraints: let $t, t' \in F(\mathcal{F} \cup Q, X)$, then $t \rightarrow_\mathcal{A} t'$ if and only

there is a context $C \in \mathcal{C}(\mathcal{F} \cup Q)$ and some terms $u_1, \ldots, u_n \in T(\mathcal{F})$

there exists $f(q_1, \ldots, q_n) \xrightarrow{c} q \in \Delta$

$t = C[f(q_1(u_1), \ldots, q_n(u_n)]$ and $t' = C[q(f(u_1, \ldots, u_n))]$

$C[f(u_1, \ldots, u_n)] \models c$

$\rightarrow_{\mathcal{A}}^*$ is the reflexive and transitive closure of $\rightarrow_{\mathcal{A}}$. As in Chapter 1, we usually write $t \rightarrow_{\mathcal{A}}^* q$ instead of $t \rightarrow_{\mathcal{A}}^* q(t)$.

An automaton $\mathcal{A} \in AWEDC$ ***accepts*** (or ***recognizes***) a ground term $t \in T(\mathcal{F})$ if $t \rightarrow_{\mathcal{A}}^* q$ for some state $q \in Q_f$. More generally, we also say that $\mathcal{A}$ accepts t in state q iff $t \rightarrow_{\mathcal{A}}^* q$ (acceptance by $\mathcal{A}$ is the particular case of acceptance by $\mathcal{A}$ in a final state).

A ***run***) is a mapping ρ from $\mathcal{P}os(t)$ into Δ such that:

- $\rho(\Lambda) \in Q_f$
- if $t(p) = f$ and the target state of $\rho(p)$ is q, then there is a transition rule $f(q_1, \ldots, q_n) \xrightarrow{c} q$ in Δ such that for all $1 \leq i \leq n$, the target state of $\rho(pi)$ is q_i and $t|_p \models c$.

Note that we do not have here exactly the same definition of a run as in Chapter 1: instead of the state, we keep also the rule which yielded this state. This will be useful in the design of an emptiness decision algorithm for non-deterministic automata with equality and disequality constraints.

The ***language accepted*** (or ***recognized***) by an automaton $\mathcal{A} \in AWEDC$ is the set $L(\mathcal{A})$ of terms $t \in T(\mathcal{F})$ that are accepted by $\mathcal{A}$.

Example 37. Balanced complete binary trees over the alphabet f (binary) and a (constant) are recognized by the AWEDC $(\{q\}, \{f, a\}, \{q\}, \Delta)$ where Δ consists of the following rules:

$$\begin{array}{ll} r_1: & a \rightarrow q \\ r_2: & f(q, q) \xrightarrow{1=2} q \end{array}$$

For example, $t = f(f(a, a), f(a, a))$ is accepted. The mapping which associates r_1 to every position p of t such that $t(p) = a$ and which associates r_2 to every position p of t such that $t(p) = f$ is indeed a successful run: for every position p of t such that $t(p) = f$, $t|_{p\cdot 1} = t_{p\cdot 2}$, hence $t|_p \models 1 = 2$.

Example 38. Consider the following AWEDC: $(Q, \mathcal{F}, Q_f, \Delta)$ with $\mathcal{F} = \{0, s, f\}$ where 0 is a constant, s is unary and f has arity 4, $Q = \{q_n, q_0, q_f\}$, $Q_f = \{q_f\}$, and Δ consists of the following rules:

$$\begin{array}{rcl@{\qquad}rcl} 0 & \rightarrow & q_0 & s(q_0) & \rightarrow & q_n \\ s(q_n) & \rightarrow & q_n & f(q_0, q_0, q_n, q_n) & \xrightarrow{3=4} & q_f \\ f(q_0, q_0, q_0, q_0) & \rightarrow & q_f & f(q_0, q_n, q_0, q_n) & \xrightarrow{2=4} & q_f \\ f(q_f, q_n, q_n, q_n) & \xrightarrow{14=4 \wedge 21=12 \wedge 131=3} & q_f & & & \end{array}$$

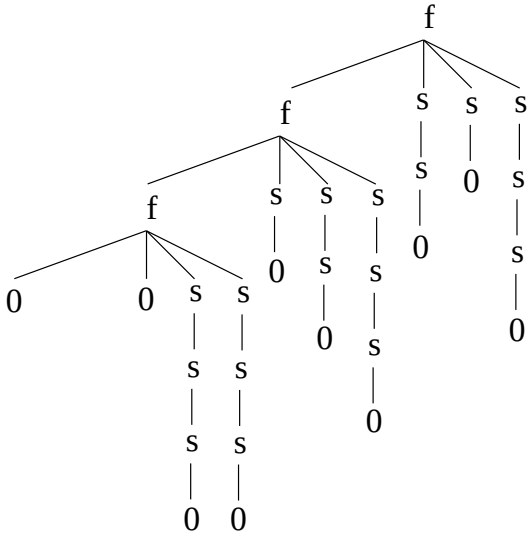


Figure 4.1: A computation of the sum of two natural numbers

This automaton computes the sum of two natural numbers written in base one in the following sense: if t is accepted by $\mathcal{A}$ then[1] $t = f(t_1, s^n(0), s^m(0), s^{n+m}(0))$ for some t_1 and $n, m \geq 0$. Conversely, for each $n, m \geq 0$, there is a term $f(t_1, s^n(0), s^m(0), s^{n+m}(0))$ which is accepted by the automaton.

For instance the term depicted on Figure 4.1 is accepted by the automaton. Similarly, it is possible to design an automaton of the class AWEDC which "computes the multiplication" (see exercises)

In order to evaluate the complexity of operations on automata of the class AWEDC, we need to precise a representation of the automata and estimate the space which is necessary for this representation.

The ***size*** of is a Boolean combination of equality and disequality constraints is defined by induction:

- $\|\pi = \pi'\| \stackrel{\text{def}}{=} \|\pi \neq \pi'\| \stackrel{\text{def}}{=} |\pi| + |\pi'|$ ($|\pi|$ is the length of π)
- $\|c \wedge c'\| \stackrel{\text{def}}{=} \|c \vee c'\| \stackrel{\text{def}}{=} \|c\| + \|c'\| + 1$
- $\|\neg c\| \stackrel{\text{def}}{=} \|c\|$

Now, deciding whether $t \models c$ depends on the representation of t. If t is represented as a directed acyclic graph (a DAG) with maximal sharing, then this can be decided in $O(\|c\|)$ on a RAM. Otherwise, it requires to compute first this representation of t, and hence can be computed in time at most $O(\|t\| \log \|t\| + \|c\|)$.

[1] $s^n(0)$ denotes $\underbrace{s(\ldots s}_{n}(0)$

¿From now on, we assume, for complexity analysis, that the terms are represented with maximal sharing in such a way that checking an equality or a disequality constraint on t can be completed in a time which is independent of $\|t\|$.

The ***size*** of an automaton $\mathcal{A} \in AWEDC$ is

$$\|\mathcal{A}\| \stackrel{\text{def}}{=} |Q| + \sum_{f(q_1,\ldots,q_n) \xrightarrow{c} q \in \Delta} n + 2 + \|c\|$$

An automaton $\mathcal{A}$ in AWEDC is ***deterministic*** if for every $t \in T(\mathcal{F})$, there is at most one state q such that $t \xrightarrow[\mathcal{A}]{*} q$. It is ***complete*** if for every $t \in T(\mathcal{F})$ there is at least one state q such that $t \xrightarrow[\mathcal{A}]{*} q$.

When every constraint is a tautology, then our definition of automata reduces to the definition of Chapter 1. However, in such a case, the notions of determinacy do not fully coincide, as noticed in Chapter 1, page 21.

Proposition 18. *Given $t \in T(\mathcal{F})$ and $\mathcal{A} \in AWEDC$, deciding whether t is accepted by $\mathcal{A}$ can be completed in polynomial time (linear time for a deterministic automaton).*

Proof. Because of the DAG representation of t, the satisfaction of a constraint $\pi = \pi'$ on t can be completed in time $O(|\pi| + |\pi'|)$. Thus, if $\mathcal{A}$ is deterministic, the membership test can be performed in time $O(\|t\| + \|\mathcal{A}\| + MC)$ where $MC = \max(\|c\| \mid c \text{ is a constraint of a rule of } \mathcal{A})$. If $\mathcal{A}$ is nondeterministic, the complexity of the algorithm will be $O(\|t\| \times \|\mathcal{A}\| \times MC)$. □

4.2.2 Reducing Non-determinism and Closure Properties

Proposition 19. *For every automaton $\mathcal{A} \in AWEDC$, there is a complete automaton $\mathcal{A}'$ which accepts the same language as $\mathcal{A}$. The size $\|\mathcal{A}'\|$ is polynomial in $\|\mathcal{A}\|$ and the computation of $\mathcal{A}'$ can be performed in polynomial time (for a fixed alphabet). If $\mathcal{A}$ is deterministic, then $\mathcal{A}'$ is deterministic.*

Proof. The proof is the same as for Theorem 2: we add a trash state and every transition is possible to the trash state. However, this does not keep the determinism of the automaton. We need the following more careful computation in order to preserve the determinism.

We also add a single trash state $q_\perp$. The additional transitions are computed as follows: for each function symbol $f \in \mathcal{F}$ and each tuple of states (including the trash state) $q_1, \ldots, q_n$, if there is no transition $f(q_1, \ldots, q_n) \xrightarrow{c} q \in \Delta$, then we simply add the rule $f(q_1, \ldots, q_n) \to q_\perp$ to Δ. Otherwise, let $f(q_1, \ldots, q_n) \xrightarrow{c_i} s_i$ $(i = 1, ..m)$ be all rules in Δ whose left member is $f(q_1, \ldots, q_n)$. We add the rule $f(q_1, \ldots, q_n) \xrightarrow{c'} q_\perp$ to Δ, where $c' \stackrel{\text{def}}{=} \neg \bigvee_{i=1}^{m} c_i$. □

Proposition 20. *For every automaton $\mathcal{A} \in AWEDC$, there is a deterministic automaton $\mathcal{A}'$ which accepts the same language as $\mathcal{A}$. $\mathcal{A}'$ can be computed in exponential time and its size is exponential in the size of $\mathcal{A}$. Moreover, if $\mathcal{A}$ is complete, then $\mathcal{A}'$ is complete.*

Proof. The construction is the same as in Theorem 4: states of $\mathcal{A}'$ are sets of states of $\mathcal{A}$. Final states of $\mathcal{A}'$ are those which contain at least one final state of $\mathcal{A}$. The construction time complexity as well as the size $\mathcal{A}'$ are also of the same magnitude as in Theorem 4. The only difference is the computation of the constraint: if $S_1, \ldots, S_n, S$ are sets of states, in the deterministic automaton, the rule $f(S_1, \ldots, S_n) \xrightarrow{c} S$ is labeled by a constraint c defined by:

$$c = \Big(\bigwedge_{q\in S} \bigvee_{\substack{f(q_1,\ldots,q_n) \xrightarrow{c_r} q\in\Delta \\ q_i\in S_i i\le n}} c_r\Big) \wedge \Big(\bigwedge_{q\notin S} \bigwedge_{\substack{f(q_1,\ldots,q_n) \xrightarrow{c_r} q\in\Delta \\ q_i\in S_i i\le n}} \neg c_r\Big)$$

Let us prove that t is accepted by $\mathcal{A}$ in states $q_1, \ldots, q_k$ (and no other states) if and only if there t is accepted by $\mathcal{A}'$ in the state $\{q_1, \ldots, q_k\}$:

$\Rightarrow$ Assume that $t \xrightarrow[\mathcal{A}]{n} q_i$ (*i.e.*$t \xrightarrow[\mathcal{A}]{*} q_i$ in n steps), for $i = 1, \ldots, k$. We prove, by induction on n, that

$$t \xrightarrow[\mathcal{A}']{n} \{q_1, \ldots, q_k\}.$$

If $n = 1$, then t is a constant and $t \to S$ is a rule of $\mathcal{A}'$ where $S = \{q \mid a \xrightarrow[\mathcal{A}]{} q\}$.

Assume now that $n > 1$. Let, for each $i = 1, \ldots, k$,

$$t = f(t_1, \ldots, t_p) \xrightarrow[\mathcal{A}]{m} f(q_1^i, \ldots, q_p^i) \xrightarrow[\mathcal{A}]{} q_i$$

and $f(q_1^i, \ldots, q_p^i) \xrightarrow{c_i} q_i$ be a rule of $\mathcal{A}$ such that $t \models c_i$. By induction hypothesis, each term t_j is accepted by $\mathcal{A}'$ in the states of a set $S_j \supseteq \{q_j^1, \ldots, q_j^k\}$. Moreover, by definition of $S = \{q_1, \ldots, q_k\}$, if $t \xrightarrow[\mathcal{A}]{*} q'$ then $q' \in S$. Therefore, for every transition rule of $\mathcal{A}$ $f(q_1', \ldots, q_p') \xrightarrow{c'} q'$ such that $q' \notin S$ and $q_j \in S_j$ for every $j \le p$, we have $t \not\models c'$. Then t satisfies the above defined constraint c.

$\Leftarrow$ Assume that $t \xrightarrow[\mathcal{A}']{n} S$. We prove by induction on n that, for every $q \in S$, $t \xrightarrow[\mathcal{A}]{n} q$.

If $n = 1$, then S is the set of states q such that $t \xrightarrow[\mathcal{A}]{} q$, hence the property.

Assume now that

$$t = f(t_1, \ldots, t_p) \xrightarrow[\mathcal{A}']{n} f(S_1, \ldots, S_p) \xrightarrow[\mathcal{A}']{} S.$$

Let $f(S_1, \ldots, S_p) \xrightarrow{c} S$ be the last rule used in this reduction. Then $t \models c$ and, by definition of c, for every state $q \in S$, there is a rule $f(q_1, \ldots, q_n) \xrightarrow{c_r} q \in \Delta$ such that $q_i \in S_i$ for every $i \le n$ and $t \models c_r$. By induction hypothesis, for each i, $t_i \xrightarrow[\mathcal{A}']{m_i} S_i$ implies $t_i \xrightarrow[\mathcal{A}]{m_i} q_i$ $(m_i < n)$ and hence $t \xrightarrow[\mathcal{A}]{n} f(q_1, \ldots, q_p) \xrightarrow[\mathcal{A}]{} q$.

Thus, by construction of the final states set, a ground term t is accepted by $\mathcal{A}'$ iff t is accepted by $\mathcal{A}$.

Now, we have to prove that $\mathcal{A}'$ is deterministic indeed. Assume that $t \xrightarrow[\mathcal{A}]{*} {}'S$ and $t \xrightarrow[\mathcal{A}']{*} S'$. Assume moreover that $S \neq S'$ and that t is the smallest term (in size) with the property of being recognized in two different states. Then there exists $S_1, \ldots, S_n$ such that $t \xrightarrow[\mathcal{A}']{*} f(S_1, \ldots, S_n)$ and such that $f(S_1, \ldots, S_n) \xrightarrow{c} S$ and $f(S_1, \ldots, S_n) \xrightarrow{c'} S'$ are transition rules of $\mathcal{A}'$, wit $t \models c$ and $t \models c'$. By symmetry, we may assume that there is a state $q \in S$ such that $q \notin S'$. Then, by definition, there are some states $q_i \in S_i$, for every $i \leq n$, and a rule $f(q_1, \ldots, q_n) \xrightarrow{c_r} q$ of $\mathcal{A}$ where c_r occurs positively in c, and is therefore satisfied by t, $t \models c_r$. By construction of the constraint of $\mathcal{A}'$, c_r must occur negatively in the second part of (the conjunction) c'. Therefore, $t \models c'$ contradicts $t \models c_r$. □

Example 39. Consider the following automaton on the alphabet $\mathcal{F} = \{a, f\}$ where a is a constant and f is a binary symbol: $Q = \{q, q_\perp\}$, $Q_f = \{q\}$ and Δ contains the following rules:

$$\begin{array}{ccc} a \to q & f(q,q) \xrightarrow{1=2} q & f(q,q) \to q_\perp \\ f(q_\perp, q) \to q_\perp & f(q, q_\perp) \to q_\perp & f(q_\perp, q_\perp) \to q_\perp \end{array}$$

This is the (non-deterministic) complete version of the automaton of Example 37.

Then the deterministic automaton computed as in the previous proposition is given by:

$$\begin{array}{cc} a \to \{q\} & f(\{q\},\{q\}) \xrightarrow{1=2\wedge\perp} \{q\} \\ f(\{q\},\{q\}) \xrightarrow{1=2} \{q, q_\perp\} & f(\{q\},\{q\}) \xrightarrow{1\neq 2} \{q_\perp\} \\ f(\{q\},\{q_\perp\}) \to \{q_\perp\} & f(\{q_\perp\},\{q\}) \to \{q_\perp\} \\ f(\{q_\perp\},\{q_\perp\}) \to \{q_\perp\} & f(\{q, q_\perp\},\{q\}) \xrightarrow{1=2\wedge\perp} \{q\} \\ f(\{q, q_\perp\},\{q\}) \xrightarrow{1=2} \{q, q_\perp\} & f(\{q, q_\perp\},\{q_\perp\}) \to \{q_\perp\} \\ f(\{q, q_\perp\},\{q, q_\perp\}) \xrightarrow{1=2} \{q, q_\perp\} & f(\{q, q_\perp\},\{q\}) \xrightarrow{1\neq 2} \{q_\perp\} \\ f(\{q\},\{q, q_\perp\}) \xrightarrow{1=2} \{q, q_\perp\} & f(\{q\},\{q, q_\perp\}) \xrightarrow{1\neq 2} \{q_\perp\} \\ f(\{q, q_\perp\},\{q\}) \xrightarrow{1\neq 2} \{q_\perp\} & f(\{q\},\{q, q_\perp\}) \xrightarrow{1=2\wedge\perp} \{q\} \\ f(\{q, q_\perp\},\{q, q_\perp\}) \xrightarrow{1=2\wedge\perp} \{q\} & f(\{q_\perp\},\{q, q_\perp\}) \to \{q_\perp\} \end{array}$$

For instance, the constraint $1{=}2\wedge\perp$ is obtained by the conjunction of the label of $f(q,q) \xrightarrow{1=2} q$ and the negation of the constraint labelling $f(q,q) \to q_\perp$, (which is $\top$).

Some of the constraints, such as $1{=}2\wedge\perp$ are unsatisfiable, hence the corresponding rules can be removed. If we finally rename the two accepting states $\{q\}$ and $\{q, q_\perp\}$ into a single state q_f (this is possible since by replacing one of these states by the other in any left hand side of a transition rule, we get

another transition rule), then we get a simplified version of the deterministic automaton:

$$\begin{array}{cc} a \to q_f & f(q_f, q_f) \xrightarrow{1=2} q_f \\ f(q_f, q_f) \xrightarrow{1\neq 2} q_\perp & f(q_\perp, q_f) \to q_\perp \\ f(q_f, q_\perp) \to q_\perp & f(q_\perp, q_\perp) \to q_\perp \end{array}$$

Proposition 21. *The class AWEDC is effectively closed by all Boolean operations. Union requires linear time, intersection requires quadratic time and complement requires exponential time. The respective sizes of the AWEDC obtained by these construction are of the same magnitude as the time complexity.*

Proof. The proof of this proposition can be obtained from the proof of Theorem 5 (Chapter 1, pages 28–29) with straightforward modifications. The only difference lies in the product automaton for the intersection: we have to consider conjunctions of constraints. More precisely, if we have two AWEDC $\mathcal{A}_1 = (Q_1, \mathcal{F}, Q_{f1}, \Delta_1)$ and $\mathcal{A}_1 = (Q_2, \mathcal{F}, Q_{f2}, \Delta_2)$, we construct an AWEDC $\mathcal{A} = (Q_1 \times Q_2, \mathcal{F}, Q_{f1} \times Q_{f2}, \Delta)$ such that if $f(q_1, \ldots, q_n) \xrightarrow{c} q \in \Delta_1$ and $f(q'_1, \ldots, q'_n) \xrightarrow{c'} q' \in \Delta_2$, then $f((q_1, q'_1), \ldots, (q_n, q'_n)) \xrightarrow{c \wedge c'} (q, q') \in \Delta$. The AWEDC $\mathcal{A}$ recognizes $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$. □

4.2.3 Undecidability of Emptiness

Theorem 32. *The emptiness problem for AWEDC is undecidable.*

Proof. We reduce the Post Correspondence Problem (PCP). If $w_1, \ldots, w_n$ and $w'_1, \ldots, w'_n$ are the word sequences of the PCP problem over the alphabet $\{a, b\}$, we let $\mathcal{F}$ contain h (ternary), a, b (unary) and 0 (constant). Lets recall that the answer for the above instance of the PCP is a sequence $i_1, \ldots, i_p$ (which may contain some repetitions) such that $w_{i_1} \ldots w_{i_p} = w'_{i_1} \ldots w'_{i_p}$.

If $w \in \{a, b\}^*$, $w = a_1 \ldots a_k$ and $t \in T(\mathcal{F})$, we write $w(t)$ the term $a_1(\ldots(a_k(t))\ldots) \in T(\mathcal{F})$.

Now, we construct $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta) \in AWEDC$ as follows:

- Q contains a state q_v for each prefix v of one of the words w_i, w'_i (including q_{w_i} and $q_{w'_i}$ as well as 3 extra states: q_0, q and q_f. We assume that a and b are both prefix of at least one of the words w_i, w'_i. $Q_f = \{q_f\}$.

- Δ contains the following rules:

$$\begin{array}{rcll} a(q_0) & \to & q_a & b(q_0) \to q_b \\ a(q_v) & \to & q_{a\cdot v} & \text{if } q_v, q_{a\cdot v} \in Q \\ b(q_v) & \to & q_{b\cdot v} & \text{if } q_v, q_{b\cdot v} \in Q \\ a(q_{w_i}) & \to & q_a & b(q_{w_i}) \to q_b \\ a(q_{w'_i}) & \to & q_a & b(q_{w'_i}) \to q_b \end{array}$$

Δ also contains the rules:

$$\begin{array}{rcl} 0 & \rightarrow & q_0 \\ h(q_0, q_0, q_0) & \rightarrow & q \\ h(q_{w_i}, q, q_{w'_i}) & \xrightarrow{1\cdot 1^{|w_i|}=2\cdot 1 \wedge 3\cdot 1^{|w'_i|}=2\cdot 3} & q \\ h(q_{w_i}, q, q_{w'_i}) & \xrightarrow{1\cdot 1^{|w_i|}=2\cdot 1 \wedge 3\cdot 1^{|w'_i|} \wedge 1=3} & q_f \end{array}$$

The rule with left member $h(q_0, q_0, q_0)$ recognizes the beginning of a Post sequence. The rules with left members $h(q_{w_i}, q.q_{w'_i})$ ensure that we are really in presence of a successor in the PCP sequences: the constraint expresses that the subterm at position 1 is obtained by concatenating some w_i with the term at position $2 \cdot 1$ and that the subterm at position 3 is obtained by concatenating w'_i (with the same index i) with the subterm at position $2 \cdot 3$. Finally, entering the final state is subject to the additional constraint $1 = 3$. This last constraint expresses that we went thru two identical words with the w_i sequences and the w'_i sequences respectively. (See Figure 4.2).

The details that this automaton indeed accepts the solutions of the PCP are left to the reader.

Then the language accepted by $\mathcal{A}$ is empty if and only if the PCP has a solution. Since PCP is undecidable, emptiness of $\mathcal{A}$ is also undecidable. □

4.3 Automata with Constraints Between Brothers

The undecidability result of the previous section led to look for subclasses which have the desired closure properties, contain (properly) the classical tree automata and still keep the decidability of emptiness. This is the purpose of the class AWCBB:

An automaton $\mathcal{A} \in AWEDC$ is an ***automaton with constraints between brothers*** if every equality (resp disequality) constraint has the form $i = j$ (resp. $i \neq j$) where $i, j \in \mathbb{N}_+$.

AWCBB is the set automata with constraints between brothers.

Example 40. The set of terms $\{f(t, t) \mid t \in T(\mathcal{F})\}$ is accepted by an automaton of the class AWCBB, because the automaton of Example 37 is in AWCBB indeed.

4.3.1 Closure Properties

Proposition 22. *AWCBB is a stable subclass of AWEDC w.r.t.Boolean operations (union, intersection, complementation).*

Proof. It is sufficient to check that the constructions of Proposition 21 preserve the property of being a member of AWCBB. □

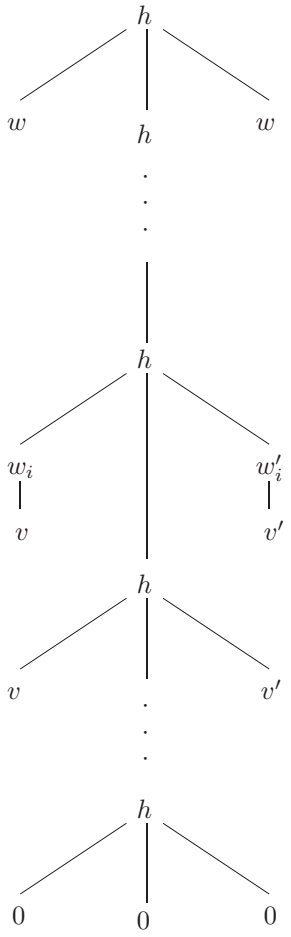


Figure 4.2: An automaton in AWEDC accepting the solutions of PCP

Recall that the time complexity of each such construction is the same in AWEDC and in the unconstrained case: union and intersection are polynomial, complementation requires determinization and is exponential.

4.3.2 Emptiness Decision

To decide emptiness we would like to design for instance a "cleaning algorithm" as in Theorem 11. As in this result, the correctness and completeness of the marking technique relies on a pumping lemma. Is there an analog of Lemma 1 in the case of automata of the class AWCBB?

There are additional difficulties. For instance consider the following example.

Example 41. $\mathcal{A}$ contains only one state and the rules

$$\begin{array}{lcl} a & \to & q \\ b & \to & q \end{array} \qquad\qquad f(q,q) \xrightarrow{1\neq 2} q$$

Now consider the term $f(f(a,b),b)$ which is accepted by the automaton. $f(a,b)$ and b yield the same state q. Hence, for a classical finite tree automaton, we may replace $f(a,b)$ with b and still get a term which is accepted by $\mathcal{A}$. This is not the case here since, replacing $f(a,b)$ with b we get the term $f(b,b)$ which is not accepted. The reason of this phenomenon is easy to understand: some constraint which was satisfied before the pumping is no longer valid after the pumping.

Hence the problem is to preserve the satisfaction of constraints along term replacements. First, concerning equality constraints, we may see the terms as DAGs in which each pair of subterms which is checked for equality is considered as a single subterm referenced in two different ways. Then replacing one of its occurrences automatically replaces the other occurrences and preserves the equality constraints. This is what is formalized below.

Preserving the equality constraints. Let t be a term accepted by the automaton $\mathcal{A}$ in AWCBB. Let ρ be a run of the automaton on t. With every position p of t, we associate the conjunction $cons(p)$ of atomic (equality or disequality) constraints that are checked by $\rho(p)$ and satisfied by t. More precisely: let $\rho(p) = f(q_1,\ldots,q_n) \xrightarrow{c'} q$; $cons(p) \stackrel{\text{def}}{=} decomp(c',p)$ where $decomp(c',p)$ is recursively defined by: $decomp(\top,p) \stackrel{\text{def}}{=} \top$, $decomp(c_1 \wedge c_2, p) \stackrel{\text{def}}{=} decomp(c_1,p) \wedge decomp(c_2,p)$ and $decomp(c_1 \vee c_2, p) = decomp(c_1,p)$ if $t|_p \models c_1$, $decomp(c_1 \vee c_2, p) = decomp(c_2,p)$ otherwise. We can show by a simple induction that $t|_p \models cons(p)$.

Now, we define the equivalence relation $=_t$ on the set of positions of t as the least equivalence relation such that:

- if $i = j \in cons(p)$, then $p \cdot i =_t p \cdot j$
- if $p =_t p'$ and $p \cdot \pi \in \mathcal{P}os(t)$, then $p \cdot \pi =_t p' \cdot \pi$

Note that in the last case, we have $p' \cdot \pi \in \mathcal{P}os(t)$. Of course, if $p =_t p'$, then $t|_p = t|_{p'}$ (but the converse is not necessarily true). Note also (and this is a property of the class AWCBB) that $p =_t p'$ implies that the lengths of p and p' are the same, hence, if $p \neq p'$, they are incomparable *w.r.t.*the prefix ordering. We can also derive from this remark that each equivalence class is finite (we may assume that each equality constraint of the form $i = i$ has been replaced by $\top$).

Example 42. Consider the automaton whose transition rules are:

$$\begin{array}{llllllll} r1: & f(q,q) & \to & q & r2: & a & \to & q \\ r3: & f(q,q) & \xrightarrow{1=2} & q_f & r4: & b & \to & q \\ r5: & f(q,q_f) & \to & q_f & r6: & f(q_f,q) & \to & q_f \\ r7: & f(q,q_f) & \to & q & r8: & f(q_f,q) & \to & q \end{array}$$

Let $t = f(b, f(f(f(a,a), f(a,b)), f(f(a,a), f(a,b))))$. A possible run of $\mathcal{A}$ on t is $r5(r4, r3(r1(r1(r2,r2), r1(r2,r5)), r8(r3(r2,r2), r1(r2,r5))))$ Equivalence classes of positions are:

$$\{\Lambda\}, \{1\}, \{2\}, \{21, 22\}, \{211, 221\}, \{212, 222\},$$
$$\{2111, 2211, 2112, 2212\}, \{2121, 2221\}, \{2122, 2222\}$$

Let us recall the principle of pumping, for finite bottom-up tree automata (see Chapter 1). When a ground term $C[C'[t]]$ (C and C' are two contexts) is such that t and $C'[t]$ are accepted in the same state by a NFTA $\mathcal{A}$, then every term $C[C'^n[t]]$ ($n \geq 0$) is accepted by $\mathcal{A}$ in the same state as $C[C'[t]]$. In other words, any $C[C'^n[t]] \in L(\mathcal{A})$ may be reduced by pumping it up to the term $C[t] \in L(\mathcal{A})$. We consider here a position p (corresponding to the term $C'[t]$) and its equivalence class $[\![p]\!]$ modulo $=_t$. The simultaneous replacement on $[\![p]\!]$ with t in u, written $u[t]_{[\![p]\!]}$, is defined as the term obtained by successively replacing the subterm at position p' with t for each position $p' \in [\![p]\!]$. Since any two positions in $[\![p]\!]$ are incomparable, the replacement order is irrelevant. Now, a ***pumping*** is a pair $(C[C'[t]]_p, C[C'^n[t]]_{[\![p]\!]})$ where $C'[t]$ and t are accepted in the same state.

Preserving the disequality constraints. We have seen on Example 41 that, if t is accepted by the automaton, replacing one of its subterms, say u, with a term v accepted in the same state as u, does not necessary yield an accepted term. However, the idea is now that, if we have sufficiently many such candidates v, at least one of the replacements will keep the satisfaction of disequality constraints.

This is the what shows the following lemma which states that minimal accepted terms cannot contain too many subterms accepted in the same state.

Lemma 5. *Given any total simplification ordering, a minimal term accepted by a deterministic automaton in AWCBB contains at most $|Q| \times N$ distinct subterms where N is the maximal arity of a function symbol and $|Q|$ is the number of states of the automaton.*

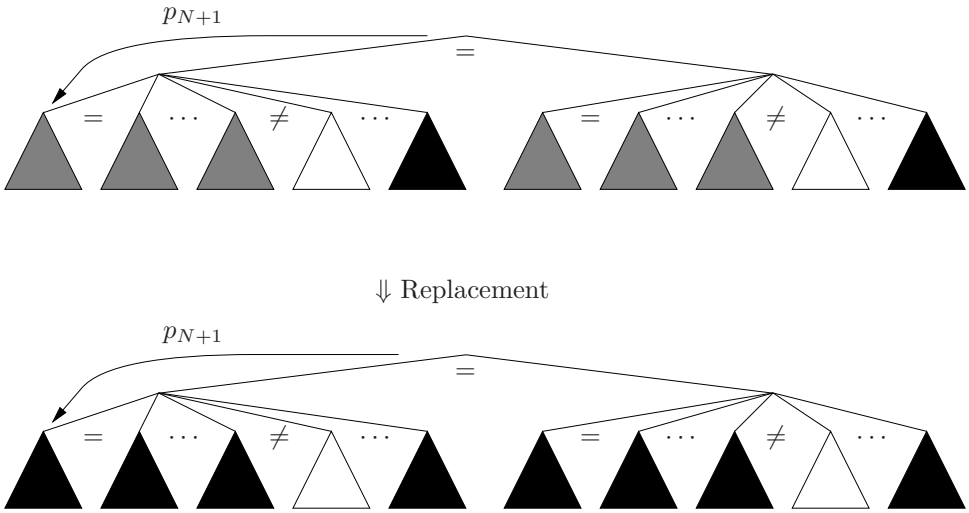


Figure 4.3: Constructing a smaller term accepted by the automaton

Proof. If ρ is a run, let $\tau\rho$ be the mapping from positions to states such that $\tau\rho(p)$ is the target state of $\rho(p)$.

If t is accepted by the automaton (let ρ be a successful run on t) and contains at least $1 + |Q| \times N$ distinct subterms, then there are at least $N + 1$ positions $p_1, \ldots, p_{N+1}$ such that $\tau\rho(p_1) = \ldots = \tau\rho(p_{N+1})$ and $t|_{p_1}, \ldots, t|_{p_{N+1}}$ are distinct. Assume for instance that $t|_{p_{N+1}}$ is the largest term (in the given total simplification ordering) among $t|_{p_1}, \ldots, t|_{p_{N+1}}$. We claim that one of the terms $v_i \stackrel{\text{def}}{=} t[t|_{p_i}]_{[\![p_{N+1}]\!]}$ $(i \leq N)$ is accepted by the automaton.

For each $i \leq N$, we may define unambiguously a tree ρ_i by: $\rho_i = \rho[\rho|_{p_i}]_{[\![p_{N+1}]\!]}$.

First, note that, by determinacy, for each position $p \in [\![p_{N+1}]\!]$, $\tau\rho(p) = \tau\rho(p_{N+1}) = \tau\rho(p_i)$. To show that there is a ρ_i which is a run, it remains to find a ρ_i the constraints of which are satisfied. Equality constraints of any ρ_i are satisfied, from the construction of the equivalence classes (details are left to the reader).

Concerning disequality constraints, we choose i in such a way that all subterms at brother positions of p_{N+1} are distinct from $t|_{p_i}$ (this choice is possible since N is the maximal arity of a function symbol and there are N distinct candidates). We get a replacement as depicted on Figure 4.3.

Let $p_{N+1} = \pi \cdot k$ where $k \in \mathbb{N}$ (π is the position immediately above p_{N+1}). Every disequality in $cons(\pi)$ is satisfied by choice of i. Moreover, if $p' \in [\![p_{N+1}]\!]$ and $p' = \pi' \cdot k'$ with $k' \in \mathbb{N}$, then every disequality in $mathitcons(\pi')$ is satisfied since $v_i|_\pi = v_i|_{\pi'}$.

Hence we constructed a term which is smaller than t and which is accepted by the automaton. This yields the lemma. □

Theorem 33. *Emptiness can be decided in polynomial time for deterministic automata in AWCBB.*

Proof. The basic idea is that, if we have enough distinct terms in states $q_1, \ldots, q_n$, then the transition $f(q_1, \ldots, q_n) \xrightarrow{c} q$ is possible. Use a marking algorithm (as

in Theorem 3) and keep for each state the terms that are known to be accepted in this state. It is sufficient to keep at most N terms in each state (N is the maximal arity of a function symbol) according to Lemma 5 and the determinacy hypothesis (terms in different states are distinct). More precisely, we use the following algorithm:

input: AWCBB $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$
begin
– *Marked* is a mapping which associates each state with a set of terms accepted in that state.
Set *Marked* to the function which maps each state to the $\emptyset$
repeat
Set $Marked(q)$ to $Marked(q) \cup \{t\}$
where
$f \in \mathcal{F}_n$, $t_1 \in Marked(q_1), \ldots, t_n \in Marked(q_n)$,
$f(q_1, \ldots, q_n) \xrightarrow{c} q \in \Delta$,
$t = f(t_1, \ldots, t_n)$ and $t \models c$,
$|Marked(q)| \leq N - 1$,
until no term can be added to any $Marked(q)$
output: true if, for every state $q_f \in Q_f$, $Marked(q_f) = \emptyset$.
end

□

Complexity. For non-deterministic automata, an exponential time algorithm is derived from Proposition 20 and Theorem 33. Actually, in the non-deterministic case, the problem is EXPTIME-complete.

We may indeed reduce the following problem which is known to be EXPTIME-complete to non-emptiness decision for nondeterministic AWCBB.
Instance n tree automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ over $\mathcal{F}$.
Answer "yes" iff the intersection the respective languages recognized by $\mathcal{A}_1, \ldots, \mathcal{A}_n$ is not empty.

We may assume without loss of generality that the states sets of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ (called respectively $Q_1, \ldots, Q_n$) are pairwise disjoint, and that every $\mathcal{A}_i$ has a single final state called q_i^f. We also assume that $n = 2^k$ for some integer k. If this is not the case, let k be the smallest integer i such that $n < 2^i$ and let $n' = 2^k$. We consider a second instance of the above problem: $\mathcal{A'}_1, \ldots, \mathcal{A'}_{n'}$ where

$\mathcal{A'}_i = \mathcal{A}_i$ for each $i \leq n$.

$\mathcal{A'}_i = (\{q\}, \mathcal{F}, \{q\}, \{f(q, \ldots, q) \to q | f \in \mathcal{F}\})$ for each $n < i \leq n'$.

Note that the tree automaton in the second case is universal, *i.e.*it accepts every term of $T(\mathcal{F})$. Hence, the answer is "yes" for $\mathcal{A'}_1, \ldots, \mathcal{A'}_{n'}$ iff it is "yes" for $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

Now, we add a single new binary symbol g to $\mathcal{F}$, getting $\mathcal{F}'$, and consider the following AWCBB $\mathcal{A}$:

$$\mathcal{A} = (\bigcup_{i=1}^{n} Q_i \uplus \{q_1, \ldots, q_{n-1}\}, \mathcal{F}', \{q_1\}, \Delta)$$

where q_1,...,q_{2n-1} are new states, and the transition of Δ are:

every transition rule of $\mathcal{A}_1$,...,$\mathcal{A}_n$ is a transition rule of $\mathcal{A}$,

for each $i < \frac{n}{2}$, $g(q_{2i}, q_{2i+1}) \xrightarrow{1=2} q_i$ is a transition rule of $\mathcal{A}$,

for each i, $\frac{n}{2} \leq i < n-1$, $g(q^f_{2i}, q^f_{2i+1}) \xrightarrow{1=2} q_i$ is a transition rule of $\mathcal{A}$.

Note that $\mathcal{A}$ is non-deterministic, even if every $\mathcal{A}_i$ is deterministic.

We can show by induction on k ($n = 2^k$) that the answer to the above problem is "yes" iff the language recognized by $\mathcal{A}$ is not empty. Moreover, the size of $\mathcal{A}$ si linear in the size of the initial problem and $\mathcal{A}$ is constructed in a time which is linear in his size. This proves the EXPTIME-hardness of emptiness decision for AWCBB.

4.3.3 Applications

The main difference between AWCBB and NFTA is the non-closure of AWCBB under projection and cylindrification. Actually, the shift from automata on trees to automata on tuples of trees cannot be extended to the class AWCBB.

As long as we are interested in automata recognizing sets of trees, all results on NFTA (and all applications) can be extended to the class AWCBB (with an bigger complexity). For instance, Theorem 26 (sort constraints) can be extended to interpretations of sorts as languages accepted by AWCBB. Proposition 15 (encompassment) can be easily generalized to the case of non-linear terms in which non-linearities only occur between brother positions, provided that we replace NFTA with AWCBB. Theorem 27 can also be generalized to the reducibility theory with predicates $\trianglelefteq_t$ where t is non-linear terms, provided that non-linearities in t only occur between brother positions.

However, we can no longer invoke an embedding into WSkS. The important point is that this theory only requires the weak notion of recognizability on tuples ($Rec_\times$). Hence we do not need automata on tuples, but only tuples of automata. As an example of application, we get a decision algorithm for ground reducibility of a term t *w.r.t.*left hand sides $l_1, \ldots, l_n$, provided that all non-linearities in $t, l_1, \ldots, l_n$ occur at brother positions: simply compute the automata $\mathcal{A}_i$ accepting the terms that encompass l_i and check that $L(\mathcal{A}) \subseteq L(\mathcal{A}_1) \cup \ldots \cup L(\mathcal{A}_n)$.

Finally, the application on reduction strategies does not carry over the case of non-linear terms because there really need automata on tuples.

4.4 Reduction Automata

As we have seen above, the first-order theory of finitely many unary encompassment predicates $\trianglelefteq_{t_1}, \ldots, \trianglelefteq_{t_n}$ (reducibility theory) is decidable when non-linearities in the terms t_i are restricted to brother positions. What happens

when we drop the restrictions and consider arbitrary terms $t_1, \ldots, t_n, t$? It turns out that the theory remains decidable, as we will see. Intuitively, we make impossible counter examples like the one in the proof of Theorem 32 (stating undecidability of the emptiness problem for AWEDC) with an additional condition that using the automaton which accepts the set of terms encompassing t, we may only check for a bounded number of equalities along each branch. That is the idea of the next definitions of reduction automata.

4.4.1 Definition and Closure Properties

A ***reduction automaton*** $\mathcal{A}$ is a member of AWEDC such that there is an ordering on the states of $\mathcal{A}$ such that,

> for each rule $f(q_1, \ldots, q_n) \xrightarrow{\pi_1=\pi_2 \wedge c} q$, q is strictly smaller than each q_i.

In case of an automaton with ϵ-transitions $q \to q'$ we also require q' to be not larger than q.

Example 43. Consider the set of terms on the alphabet $\mathcal{F} = \{a, g\}$ encompassing $g(g(x, y), x)$. It is accepted by the following reduction automaton, the final state of which is q_f and q_f is minimal in the ordering on states.

$$
\begin{array}{rcl@{\qquad}rcl}
a & \to & q_\top & g(q_\top, q_\top) & \to & q_{g(x,y)} \\
g(q_\top, q_{g(x,y)}) & \to & q_{g(x,y)} & & & \\
g(q_{g(x,y)}, q_\top) & \xrightarrow{11=2} & q_f & g(q_{g(x,y)}, q_\top) & \xrightarrow{11\neq 2} & q_{g(x,y)} \\
g(q_{g(x,y)}, q_{g(x,y)}) & \xrightarrow{11=2} & q_f & g(q_{g(x,y)}, q_{g(x,y)}) & \xrightarrow{11\neq 2} & q_{g(x,y)} \\
g(q, q_f) & \to & q_f & g(q_f, q) & \to & q_f
\end{array}
$$

where $q \in \{q_\top, q_{g(x,y)}, q_f\}$

This construction can be generalized, along the lines of the proof of Proposition 15 (page 96):

Proposition 23. *The set of terms encompassing a term t is accepted by a deterministic and complete reduction automaton. The size of this automaton is polynomial in $\|t\|$ as well as the time complexity for its construction.*

As usual, we are now interested in closure properties:

Proposition 24. *The class of reduction automata is closed under union and intersection. It is not closed under complementation.*

Proof. The constructions for union and intersection are the same as in the proof of Proposition 21, and therefore, the respective time complexity and sizes are the same. The proof that the class of reduction automata is closed under these constructions is left as an exercise. Consider the set L of ground terms on the alphabet $\{a, f\}$ defined by $a \in L$ and for every $t \in L$ which is not a, t has a subterm of the form $f(s, s')$ where $s \neq s'$. The set L is accepted by a (non-deterministic, non-complete) reduction automaton, but its complement is the set of balanced binary trees and it cannot be accepted by a reduction automaton (see Exercise 56). □

The weak point is of course the non-closure under complement. Consequently, this is not possible to reduce the non-determinism.

However, we have a weak version of stability:

Proposition 25.
- *With each reduction automaton, we can associate a complete reduction automaton which accepts the same language. Moreover, this construction preserves the determinism.*
- *The class of complete deterministic reduction automata is closed under complement.*

4.4.2 Emptiness Decision

Theorem 34. *Emptiness is decidable for the class of reduction automata.*

The proof of this result is quite complicated and gives quite high upper bounds on the complexity (a tower of several exponentials). Hence, we are not going to reproduce it here. Let us only sketch how it works, in the case of deterministic reduction automata.

As in Section 4.3.2, we have both to preserve equality and disequality constraints.

Concerning equality constraints, we also define an equivalence relation between positions (of equal subtrees). We cannot claim any longer that two equivalent positions do have the same length. However, some of the properties of the equivalence classes are preserved: first, they are all finite and their cardinal can be bounded by a number which only depends on the automaton, because of the condition with the ordering on states (this is actually not true for the class AWCBB). Then, we can compute a bound b_2 (which only depends on the automaton) such that the difference of the lengths of two equivalent positions is smaller than b_2. Nevertheless, as in Section 4.3.2, equalities are not a real problem, as soon as the automaton is deterministic. Indeed, pumping can then be defined on equivalence classes of positions. If the automaton is not deterministic, the problem is more difficult since we cannot guarantee that we reach the same state at two equivalent positions, hence we have to restrict our attention to some particular runs of the automaton.

Handling disequalities requires more care; the number of distinct subterms of a minimal accepted term cannot be bounded as for AWCBB by $|Q|\times N$, where N is the maximal arity of a function symbol. The problem is the possible "overlap" of disequalities checked by the automaton. As in Example 41, a pumping may yield a term which is no longer accepted, since a disequality checked somewhere in the term is no longer satisfied. In such a case, we say that the pumping *creates an equality*. Then, we distinguish two kinds of equalities created by a pumping: the ***close equalities*** and the ***remote equalities***. Roughly, an equality created by a pumping $(t[v(u)]_p, t[u]_p)$ is a pair of positions $(\pi\cdot\pi_1, \pi\cdot\pi_2)$ of $t[v(u)]_p$ which was checked for disequality by the run ρ at position π on $t[v(u)]_p$ and such that $t[u]_p|_{\pi\cdot\pi_1} = t[u]_p|_{\pi\cdot\pi_2}$ (π is the longest common prefix to both members of the pair). This equality $(\pi\cdot\pi_1, \pi\cdot\pi_2)$ is a close equality if $\pi \leq p < \pi\cdot\pi_1$ or $\pi \leq p < \pi\cdot\pi_2$. Otherwise ($p \geq \pi\cdot\pi_1$ or $p \geq \pi\cdot\pi_2$), it is a remote equality. The different situations are depicted on Figures 4.4 and 4.5.

One possible proof sketch is

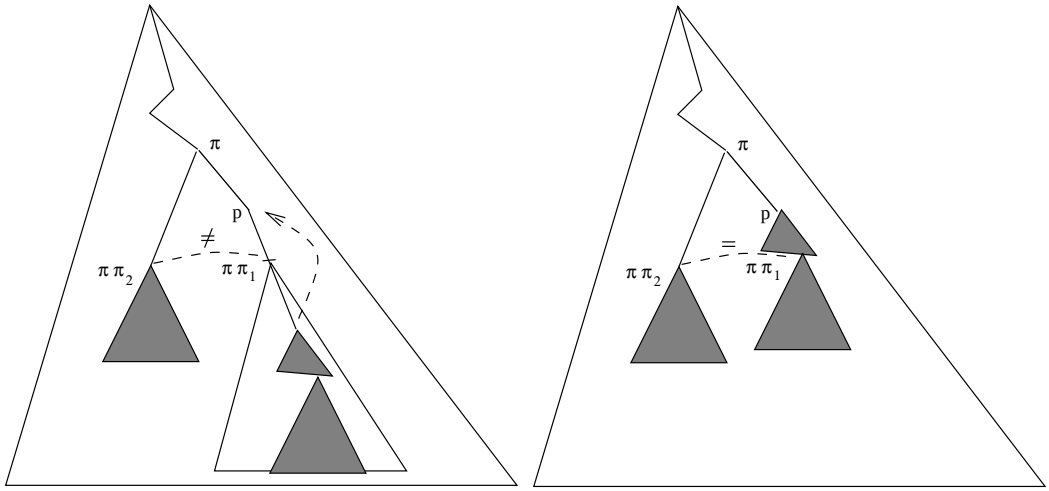


Figure 4.4: A close equality is created

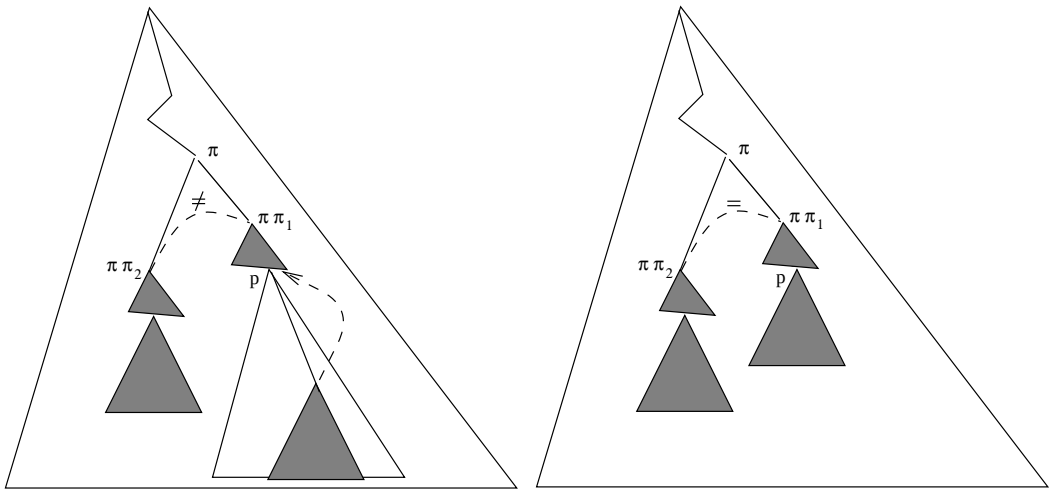


Figure 4.5: A remote equality is created

- First show that it is sufficient to consider equalities that are created at positions around which the states are incomparable *w.r.t.*>
- Next, show that, for a deep enough path, there is at least one pumping which does not yield a close equality (this makes use of a combinatorial argument; the bound is an exponential in the maximal size of a constraint).
- For remote equalities, pumping is not sufficient. However, if some pumping creates a remote equality anyway, this means that there are "big"equal terms in t. Then we switch to another branch of the tree, combining pumping in both subtrees to find one (again using a combinatorial argument) such that no equality is created.

Of course, this is a very sketchy proof. The reader is referred to the bibliography for more information about the proof.

4.4.3 Finiteness Decision

The following result is quite difficult to establish. We only mention them for sake of completeness.

Theorem 35. *Finiteness of the language is decidable for the class of reduction automata.*

4.4.4 Term Rewriting Systems

There is a strong relationship between reduction automata and term rewriting. We mention them readers interested in that topic.

Proposition 26. *Given a term rewriting system $\mathcal{R}$, the set of ground $\mathcal{R}$-normal forms is recognizable by a reduction automaton, the size of which is exponential in the size of $\mathcal{R}$. The time complexity of the construction is exponential.*

Proof. The set of $\mathcal{R}$-reducible ground terms can be defined as the union of sets of ground terms encompassing the left members of rules of $\mathcal{R}$. Thus, by Propositions 23 and 24 the set of $\mathcal{R}$-reducible ground terms is accepted by a deterministic and complete reduction automaton. For the union, we use the product construction, preserving determinism (see the proof of Theorem 5, Chapter 1) with the price of an exponential blowup. The set of ground $\mathcal{R}$-normal forms is the complement of the set of ground $\mathcal{R}$-reducible terms, and it is therefore accepted by a reduction automaton, according to Proposition 25. □

Thus, we have the following consequence of Theorems 35 and 34.

Corollary 5. *Emptiness and finiteness of the language of ground $\mathcal{R}$-normal forms is decidable for every term rewriting system $\mathcal{R}$.*

Let us cite another important result concerning recognizability of sets normal forms.

Theorem 36. *The membership of the language of ground normal forms to the class of recognizable tree languages is decidable.*

4.4.5 Application to the Reducibility Theory

Consider the reducibility theory of Section 3.4.2: there are unary predicate symbols $\trianglelefteq_t$ which are interpreted as the set of terms which encompass t. However, we accept now non linear terms t as indices.

Propositions 23, and 24, and 25 yield the following result:

Theorem 37. *The reducibility theory associated with any sets of terms is decidable.*

And, as in the previous chapter, we have, as an immediate corollary:

Corollary 6. *Ground reducibility is decidable.*

4.5 Other Decidable Subclasses

Complexity issues and restricted classes. There are two classes of automata with equality and disequality constraints for which tighter complexity results are known:

- For the class of automata containing only disequality constraints, emptiness can be decided in deterministic exponential time. For any term rewriting system $\mathcal{R}$, the set of ground $\mathcal{R}$-normal forms is still recognizable by an automaton of this subclass of reduction automata.

- For the class of deterministic reduction automata for which the constraints "cannot overlap", emptiness can be decided in polynomial time.

Combination of AWCBB and reduction automata. If you relax the condition on equality constraints in the transition rules of reduction automata so as to allow constraints between brothers, you obtain the biggest known subclass of AWEDC with a decidable emptiness problem.

Formally, these automata, called ***generalized reduction automata***, are members of AWEDC such that there is an ordering on the states set such that,

> for each rule $f(q_1, \ldots, q_n) \xrightarrow{\pi_1=\pi_2 \wedge c} q$, q is a lower bound of $\{q_1, \ldots, q_n\}$ and moreover, if $|\pi_1| > 1$ or $|\pi_2| > 1$, then q is strictly smaller than each q_i.

The closure and decidability results for reduction automata may be transposed to generalized reduction automata, with though a longer proof for the emptiness decision. Generalized reduction automata can thus be used for the decision of reducibility theory extended by some restricted sort declarations. In this extension, additionally to encompassment predicates $\trianglelefteq_t$, we allow a family of unary sort predicates $. \in S$, where S is a sort symbol. But, sort declarations are limited to atoms of the form $t \in S$ where where non linear variables in t only occur at brother positions. This fragment is decidable by an analog of Theorem 37 for generalized reduction automata.

4.6 Tree Automata with Arithmetic Constraints

Tree automata deal with finite trees which have a width bounded by the maximal arity of the signature but there is no limitation on the depth of the trees. A natural idea is to relax the restriction on the width of terms by allowing function of variadic arity. This has been considered by several authors for applications to graph theory, typing in object-oriented languages, temporal logic and automated deduction. In these applications, variadic functions are set or multiset constructors in some sense, therefore they enjoy additional properties like associativity and/or commutativity and several types of tree automata have been designed for handling these properties. We describe here a class of tree automata which recognize terms build with usual function symbols and multiset constructors. Therefore, we deal not only with terms, but with so-called flat terms. Equality on these terms is no longer the syntactical identity, but it is extended by the equality of multisets under permutation of their elements. To recognize sets of flat terms with automata, we shall use constrained rules where the constraints are Presburger's arithmetic formulas which set conditions on the multiplicities of terms in multisets. These automata enjoy similar properties to NFTA and are used to test completeness of function definitions and inductive reducibility when associative-commutative functions are involved, provided that some syntactical restrictions hold.

4.6.1 Flat Trees

The set of function symbols $\mathcal{G}$ is composed of $\mathcal{F}$, the set of function symbols and of $\mathcal{M}$, the set of function symbols for building multisets. For simplicity we shall assume that there is only one symbol of the latter form, denoted by $\sqcup$ and written as an infix operator. The set of variables is denoted by $\mathcal{X}$. ***Flat terms*** are terms generated by the non-terminal T of the following grammar.

$$\begin{array}{rcll} N & ::= & 1 \,|\, 2 \,|\, 3 \ldots & \\ T & ::= & S \,|\, U & \text{(flat terms)} \\ S & ::= & x \,|\, f(T_1, \ldots, T_n) & \text{(flat terms of sort } \mathcal{F}) \\ U & ::= & N_1.S_1 \sqcup \ldots \sqcup N_p.S_p & \text{(flat terms of sort } \sqcup) \end{array}$$

where $x \in \mathcal{X}$, $n \geq 0$ is the arity of f, $p \geq 1$ and $\sum_{i=1}^{i=p} N_i \geq 2$. Moreover the inequality $S_i \neq_P S_j$ holds for $i \neq j$, $1 \leq i, j < n$, where $=_P$ is defined as the smallest congruence such that:

- $x =_P x$,
- $f(s_1, \ldots, s_n) =_P f(t_1, \ldots, t_n)$ if $f \in \mathcal{F}$ and $s_i =_P t_i$ for $i = 1, \ldots, n$,
- $n_1.s_1 \sqcup \ldots \sqcup n_p.s_p =_P m_1.t_1 \sqcup \ldots \sqcup m_q.t_q$ if $p = q$ and there is some permutation σ on $\{1, \ldots, p\}$ such that $s_i =_P t_{\sigma(i)}$ and $n_i = m_{\sigma(i)}$ for $i = 1, \ldots, p$.

Example 44. $3.a$ and $3.a \sqcup 2.f(x, b)$ are flat terms, but $2.a \sqcup 1.a \sqcup f(x, b)$ is not since $2.a$ and $1.a$ must be grouped together to make $3.a$.

The usual notions on terms can be generalized easily for flat terms. We recall only what is needed in the following. A flat term is ***ground*** if it contains no variables. The ***root*** of a flat term is defined by

- for the flat terms of sort $\mathcal{F}$, $root(x) = x$, $root(f(t_1, \ldots, t_n)) = f$,
- for the flat terms of sort $\sqcup$, $root(s_1 \sqcup \ldots \sqcup s_n) = \sqcup$.

Our notion of subterm is slightly different from the usual one. We say that s is a subterm of t if and only if

- either s and t are identical,
- or $t = f(s_1, \ldots, s_n)$ and s is a subterm of some s_i,
- or $t = n_1.t_1 \sqcup \ldots \sqcup n_p.t_p$ and s is a subterm of some t_i.

For simplicity, we extend $\sqcup$ to an operation between flat terms s, t denoting (any) flat term obtained by regrouping elements of sort $\mathcal{F}$ in s and t which are equivalent modulo $=_P$, leaving the other elements unchanged. For instance $s = 2.a \sqcup 1.f(a, a)$ and $t = 3.b \sqcup 2.f(a, a)$ yields $s \sqcup t = 2.a \sqcup 3.b \sqcup 3.f(a, a)$.

4.6.2 Automata with Arithmetic Constraints

There is some regularity in flat terms that is likely to be captured by some class of automata-like recognizers. For instance, the set of flat terms such that all integer coefficients occurring in the terms are even, seems to be easily recognizable, since the predicate $even(n)$ can be easily decided. The class of automata that we describe now has been designed for accepting such sets of ground flat terms. A ***flat tree automaton with arithmetic constraints*** (NFTAC) over $\mathcal{G}$ is a tuple $(Q_{\mathcal{F}}, Q_{\sqcup}, \mathcal{G}, Q_f, \Delta)$ where

- $Q_{\mathcal{F}} \cup Q_{\sqcup}$ is a finite set of states, such that
  - $Q_{\mathcal{F}}$ is the set of states of sort $\mathcal{F}$,
  - $Q_{\sqcup}$ is the set of states of sort $\sqcup$,
  - $Q_{\mathcal{F}} \cap Q_{\sqcup} = \emptyset$,
- $Q_f \subseteq Q_{\mathcal{F}} \sqcup Q_{\sqcup}$ is the set of final states,
- Δ is a set of rules of the form:
  - $f(q_1, \ldots, q_n) \to q$, for $n \geq 0, f \in \mathcal{F}_n$, $q_1, \ldots, q_n \in Q_{\mathcal{F}} \cup Q_{\sqcup}$, and $q \in Q_{\mathcal{F}}$,
  - $N.q \xrightarrow{c(N)} q'$, where $q \in Q_{\mathcal{F}}$, $q' \in Q_{\sqcup}$, and c is a Presburger's arithmetic[2] formula with the unique free variable N,
  - $q_1 \sqcup q_2 \to q_3$ where $q_1, q_2, q_3 \in Q_{\sqcup}$.

  Moreover we require that

[2]Presburger's arithmetic is first order arithmetic with addition and constants 0 and 1. This fragment of arithmetic is known to be decidable.

- $q_1 \sqcup q_2 \to q_3$ is a rule of Δ implies that $q_2 \sqcup q_1 \to q_3$ is also a rule of Δ,
- $q_1 \sqcup (q_2 \sqcup q_3) \to q_4$ is a rule of Δ implies that $(q_1 \sqcup q_2) \sqcup q_3 \to q_4$ is also a rule of Δ where $q_2 \sqcup q_3$ (resp. $q_1 \sqcup q_2$) denotes any state q' such that there is a rule $q_2 \sqcup q_3 \to q'$ (resp. $q_1 \sqcup q_2 \to q'$).

These two conditions on Δ will ensure that two flat terms equivalent modulo $=_P$ reach the same states.

Example 45. Let $\mathcal{F} = \{a, f\}$ and let $\mathcal{A} = (Q_{\mathcal{F}}, Q_\sqcup, \mathcal{G}, Q_f, \Delta)$ where

$Q_{\mathcal{F}} = \{q\}, Q_\sqcup = \{q_\sqcup\}$,

$Q_f = \{q_u\}$,

$$\Delta = \left\{ \begin{array}{rcllcl} a & \longrightarrow & q & N.q & \stackrel{\exists n:N=2n}{\longrightarrow} & q_\sqcup \\ f(\_,\_) & \longrightarrow & q & q_\sqcup \sqcup q_\sqcup & \longrightarrow & q_\sqcup \end{array} \right\}$$

where $\_$ stands for q or $q_\sqcup$.

Let $\mathcal{A} = (Q_{\mathcal{F}}, Q_\sqcup, \mathcal{G}, Q_f, \Delta)$ be a flat tree automaton. The move relation $\to_{\mathcal{A}}$ is defined by: let $t, t' \in \mathcal{T}(\mathcal{F} \cup Q, X)$, then $t \to_{\mathcal{A}} t'$ if and only if there is a context $C \in \mathcal{C}(\mathcal{G} \cup Q)$ such that $t = C[s]$ and $t' =_P C[s']$ where

- either there is some $f(q_1, \ldots, q_n) \to q' \in \Delta$ and $s = f(q_1, \ldots, q_n)$, $s' = q'$,
- or there is some $N.q \stackrel{c(N)}{\longrightarrow} q' \in \Delta$ and $s = n.q$ with $\models c(n)$, $s' = q'$,
- or there is some $q_1 \sqcup q_2 \to q_3 \in \Delta$ and $s = q_1 \sqcup q_2$, $s' = q_3$.

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

Example 46. Using the automaton of the previous example, one has

$$\begin{array}{lcl} 2.a \sqcup 6.f(a,a) \sqcup 2.f(a,2.a) & \to^*_{\mathcal{A}} & 2.q \sqcup 6.f(q,q) \sqcup 2.f(q,2.q) \\ & \to^*_{\mathcal{A}} & 2.q \sqcup 6.q \sqcup 2.f(q,q_\sqcup) \\ & \to^*_{\mathcal{A}} & 2.q \sqcup 6.q \sqcup 2.q \\ & \to^*_{\mathcal{A}} & q_\sqcup \sqcup q_\sqcup \sqcup q_\sqcup \to^*_{\mathcal{A}} q_\sqcup \sqcup q_\sqcup \to^*_{\mathcal{A}} q_\sqcup \end{array}$$

We define now ***semilinear flat languages***. Let $\mathcal{A} = (Q_{\mathcal{F}}, Q_\sqcup, \mathcal{G}, Q_f, \Delta)$ be a flat tree automaton. A ground flat term t is ***accepted*** by $\mathcal{A}$, if there is some $q \in Q_f$ such that $t \to^*_{\mathcal{A}} q$. The flat tree language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of all ground flat terms accepted by $\mathcal{A}$. A set of flat terms is ***semilinear*** if there $L = L(\mathcal{A})$ for some NFTAC $\mathcal{A}$. Two flat tree automata are ***equivalent*** if they recognize the same language.

Example 47. The language of terms accepted by the automaton of Example 45 is the set of ground flat terms with root $\sqcup$ such that for each subterm $n_1.s_1 \sqcup \ldots \sqcup n_p.s_p$ we have that n_i is an even number.

Flat tree automata are designed to take into account the $=_P$ relation, which is stated by the next proposition.

Proposition 27. *Let s, t, be two flat terms such that $s =_P t$, let $\mathcal{A}$ be a flat tree automaton, then $s \to^*_{\mathcal{A}} q$ implies $t \to^*_{\mathcal{A}} q$.*

Proof. The proof is by structural induction on s. □

Proposition 28. *Given a flat term t and a flat tree automaton $\mathcal{A}$, it is decidable whether t is accepted by $\mathcal{A}$.*

Proof. The decision algorithm for membership for flat tree automata is nearly the same as the one for tree automata presented in Chapter 1, using an oracle for the decision of Presburger's arithmetic formulas. □

Our definition of flat tree automata corresponds to nondeterministic flat tree automata. We now define deterministic flat tree automata (DFTAC).

Let $\mathcal{A} = (Q_{\mathcal{F}}, Q_{\sqcup}, \mathcal{G}, Q_f, \Delta)$ be a NFTAC over $\mathcal{G}$.

- The automaton $\mathcal{A}$ is ***deterministic*** if for each ground flat term t, there is at most one state q such that $t \to^*_{\mathcal{A}} q$.
- The automaton $\mathcal{A}$ is ***complete*** if for each ground flat term t, there a state such that $t \to^*_{\mathcal{A}} q$.
- A state q is ***accessible*** if there is one ground flat term t such that $t \to^*_{\mathcal{A}} q$. The automaton is ***reduced*** if all states are accessible.

4.6.3 Reducing Non-determinism

As for usual tree automata, there is an algorithm for computing an equivalent DFTAC from any NFTAC which proves that a language recognized by a NFTAC is also recognized by a DFTAC. The algorithm is similar to the determinization algorithm of the class AWEDC: the ambiguity arising from overlapping constraints is lifted by considering mutually exclusive constraints which cover the original constraints, and using sets of states allows to get rid of the nondeterminism of rules having the same left-hand side. Here, we simply have to distinguish between states of $Q_{\mathcal{F}}$ and states of $Q_{\sqcup}$.

Determinization algorithm

input $\mathcal{A} = (Q_{\mathcal{F}}, Q_{\sqcup}, \mathcal{G}, Q_f, \Delta)$ a NFTAC.

begin

A state $[q]$ of the equivalent DFTAC is in $2^{Q_{\mathcal{F}}} \cup 2^{Q_{\sqcup}}$.

Set $Q^d_{\mathcal{F}} = \emptyset$, $Q^d_{\sqcup} = \emptyset$, $\Delta_d = \emptyset$.

repeat

for each f of arity n, $[q]_1, \ldots, [q]_n \in Q^d_{\mathcal{F}} \cup Q^d_{\sqcup}$ **do**

let $[q] = \{q \mid \exists f(q_1, \ldots, q_n) \to q \in \Delta$ with $q_i \in [q]_i$ for $i = 1, \ldots, n\}$

in Set $Q^d_{\mathcal{F}}$ to $Q^d_{\mathcal{F}} \cup \{[q]\}$
Set Δ_d to $\Delta_d \cup \{f([q]_1, \ldots, [q]_n) \to [q]\}$

endfor

for each $[q] \in Q_{\mathcal{F}}$ **do**

for each $[q'] \subseteq \{q'' \mid \exists N.q \xrightarrow{c(N)} q'' \in \Delta \text{ with } q \in [q]\}$ **do**

let C be $\Big(\bigwedge_{q\in[q]} \bigvee_{\substack{N.q \xrightarrow{c_i(N)} q' \in \Delta \\ q' \in [q']}} c_i(N)\Big) \wedge \Big(\bigwedge_{q\in[q]} \bigwedge_{\substack{N.q \xrightarrow{c_i(N)} q' \in \Delta \\ q' \notin [q']}} \neg c_i(N)\Big)$

in Set $Q^d_{\sqcup}$ to $Q^d_{\mathcal{F}} \cup \{[q']\}$
Set Δ_d to $\Delta_d \cup \{N.[q] \xrightarrow{C(N)} [q']\}$

endfor

endfor

for each $[q]_1, [q]_2 \in Q^d_{\sqcup}$ **do**

let $[q] = \{q \mid \exists q_1 \in [q]_1,\ q_2 \in [q]_2,\ q_1 \sqcup q_2 \to q \in \Delta\}$
in Set $Q^d_{\sqcup}$ to $Q^d_{\mathcal{F}} \cup \{[q]\}$
Set Δ_d to $\Delta_d \cup \{[q]_1 \sqcup [q]_2 \to [q]\}$

endfor

until no rule can be added to Δ_d
Set $Q^d_f = \{[q] \in Q^d_{\mathcal{F}} \cup Q^d_{\sqcup} \mid [q] \cap Q_f \neq \emptyset\}$,

end

output: $\mathcal{A}_d = (Q^d_F, Q^d_{\sqcup}, \mathcal{F}, Q^d_f, \Delta_d)$

Proposition 29. *The previous algorithm terminates and computes a deterministic flat tree automaton equivalent to the initial one.*

Proof. The termination is obvious. The proof of the correctness relies on the following lemma:

Lemma 6. $t \to^*_{\mathcal{A}_d} [q]$ *if and only if* $t \to^*_{\mathcal{A}} q$ *for all* $q \in [q]$.

The proof is by structural induction on t and follows the same pattern as the proof for the class AWEDC. □

Therefore we have proved the following theorem stating the equivalence between DFTAC and NFTAC.

Theorem 38. *Let L be a semilinear set of flat terms, then there exists a DFTAC that accepts L.*

4.6.4 Closure Properties of Semilinear Flat Languages

Given an automaton $\mathcal{A} = (Q, \mathcal{G}, Q_f, \Delta)$, it is easy to construct an equivalent complete automaton. If $\mathcal{A}$ is not complete then

- add two new trash states q_t of sort $\mathcal{F}$ and $q_t^{\sqcup}$ of sort $\sqcup$,
- for each $f \in \mathcal{F}$, $q_1, \ldots, q_n \in Q \cup \{q_t, q_t^{\sqcup}\}$, such that there is no rule having $f(q_1, \ldots, q_n)$ as left-hand side, then add $f(q_1, \ldots, q_n) \to q_t$,
- for each q of sort $\mathcal{F}$, let $c_1(N), \ldots, c_m(N)$ be the conditions of the rules $N.q \xrightarrow{c_i(N)} q'$,
  - if the formula $\exists N\ (c_1(N) \vee \ldots \vee c_m(N))$ is not equivalent to *true*, then add the rule $N.q \xrightarrow{\neg(c_1(N) \vee \ldots \vee c_m(N))(N)} q_t^{\sqcup}$,
  - if there are some q, q' of sort $\sqcup$ such that there is no rule $q \sqcup q' \to q$", then add the rules $q \sqcup q' \to q_t^{\sqcup}$ and $q' \sqcup q \to q_t^{\sqcup}$.
  - if there is some rule $(q_1 \sqcup q_2) \sqcup q_3 \to q_t^{\sqcup}$ (resp. $q_1 \sqcup (q_2 \sqcup q_3) \to q_t^{\sqcup}$, add the rule $q_1 \sqcup (q_2 \sqcup q_3) \to q_t^{\sqcup}$ (resp. $(q_1 \sqcup q_2) \sqcup q_3 \to q_t^{\sqcup}$) if it is missing.

This last step ensures that we build a flat tree automaton, and it is straightforward to see that this automaton is equivalent to the initial one (same proof as for DFTA). This is stated by the following proposition.

Theorem 39. *For each flat tree automaton $\mathcal{A}$, there exists a complete equivalent automaton $\mathcal{B}$.*

Example 48. The automaton of Example 45 is not complete. It can be completed by adding the states $q_t, q_t^{\sqcup}$, and the rules $N.q_t \xrightarrow{N \geq 0} q_t^{\sqcup}$, $N.q \xrightarrow{\exists n\ N=2n+1} q_t^{\sqcup}$, $f(\_,\_) \longrightarrow q_t$ where $(\_,\_)$ stands for a pair of $q, q_{\sqcup}, q_t, q_t^{\sqcup}$ such that if a rule the left hand side of which is $f(\_,\_)$ is not already in Δ.

Theorem 40. *The class of semilinear flat languages is closed under union.*

Proof. Let L (resp. M) be a semilinear flat language recognized by $\mathcal{A} = (Q_{\mathcal{F}}, Q_{\sqcup}, \mathcal{G}, Q_f, \Delta)$ (resp. $\mathcal{B} = (Q'_{\mathcal{F}}, Q'_{\sqcup}, \mathcal{G}, Q'_f, \Delta')$), then $L \cup M$ is recognized by $\mathcal{C} = (Q_{\mathcal{F}} \cup Q'_{\mathcal{F}}, Q_{\sqcup} \cup Q'_{\sqcup}, \mathcal{G}, Q_f \cup Q'_f, \Delta \cup \Delta')$. □

Theorem 41. *The class of semilinear flat languages is closed under complementation.*

Proof. Let $\mathcal{A}$ be an automaton recognizing L. Compute a complete automaton $\mathcal{B}$ equivalent to $\mathcal{A}$. Compute a deterministic automaton $\mathcal{C}$ equivalent to $\mathcal{B}$ using the determinization algorithm. The automaton $\mathcal{C}$ is still complete, and we get an automaton recognizing the complement of L by exchanging final and non-final states in $\mathcal{C}$. □

From the closure under union and complement, we get the closure under intersection (a direct construction of an automaton recognizing the intersection also exists).

Theorem 42. *The class of semilinear flat languages is closed under intersection.*

4.6.5 Emptiness Decision

The last important property to state is that the emptiness of the language recognized by a flat tree automaton is decidable. The decision procedure relies on an algorithm similar to the decision procedure for tree automata combined to a decision procedure for Presburger's arithmetic. However a straightforward modification of the algorithm in Chapter 1 doesn't work. Assume that the automaton contains the rule $q_1^{\sqcup} \sqcup q_1^{\sqcup} \to q_2^{\sqcup}$ and assume that there is some flat term t such that $1.t \to_{\mathcal{A}}^{*} q_1^{\sqcup}$. These two hypothesis don't imply that $1.t \sqcup 1.t \to_{\mathcal{A}}^{*} q_2^{\sqcup}$ since $1.t \sqcup 1.t$ is not a flat term, contrary to $2.t$. Therefore the decision procedure involves some combinatorics in order to ensure that we always deal with correct flat terms.

From now on, let $A = (Q_{\mathcal{F}}, Q_{\sqcup}, \mathcal{G}, Q_f, \Delta)$ be some given *deterministic* flat tree automaton and let M be the number of states of sort $\sqcup$. First, we need to control the possible infinite number of solutions of Presburger's conditions.

Proposition 30. *There is some computable B such that for each condition $c(N)$ of the rules of $\mathcal{A}$, either each integer n validating c is smaller than B or there are at least $M+1$ integers smaller than B validating c.*

Proof. First, for each constraint $c(N)$ of a rule of Δ, we check if $c(N)$ has a finite number of solutions by deciding if $\exists P : c(N) \Rightarrow N < P$ is true. If $c(N)$ has a finite number of solutions, it is easy to find a bound $B_1(c(N))$ on these solutions by testing $\exists n : n > k \wedge c(n)$ for $k = 1, 2, \ldots$ until it is false. If $c(N)$ has an infinite number of solutions, one computes the M^{th} solution obtained by checking $\models c(k)$ for $k = 1, 2, \ldots$. We call this M^{th} solution $B_2(c(N))$. The bound B is the maximum of all the $B_1(c(N))$'s and $B_2(c(N))$'s. □

Now we control the maximal width of terms needed to reach a state.

Proposition 31. *For all $t \to_{\mathcal{A}}^{*} q$, there is some $s \to_{\mathcal{A}}^{*} q$ such that for each subterm of s of the form $n_1.v_1 \sqcup \ldots \sqcup n_p.v_p$, we have $p \leq M$ and $n_i \leq B$.*

Proof. The bound on the coefficients n_i is a direct consequence of the previous proposition. The proof on p is by structural induction on t. The only non-trivial case is for $t = m_1.t_1 \sqcup \ldots \sqcup m_k.t_k$. Let us assume that t is the term with the smallest value of k among the terms $\{t' \mid t' \to_{\mathcal{A}}^{*} q\}$.

First we show that $k \leq M$. Let $q_i^{\sqcup}$ be the states such that $n_i.t_i \to_{\mathcal{A}} q_i^{\sqcup}$. We have thus $t \to_{\mathcal{A}}^{*} q_1^{\sqcup} \sqcup \ldots \sqcup q_k^{\sqcup} \to_{\mathcal{A}}^{*} q$. By definition of DFTAC, the reduction $q_1^{\sqcup} \sqcup \ldots \sqcup q_k^{\sqcup} \to_{\mathcal{A}}^{*} q$ has the form:

$$q_1^{\sqcup} \sqcup \ldots \sqcup q_k^{\sqcup} \underset{\mathcal{A}}{\overset{*}{\to}} q_{[12]}^{\sqcup} \sqcup q_3^{\sqcup} \sqcup \ldots \sqcup q_k^{\sqcup} \underset{\mathcal{A}}{\overset{*}{\to}} \ldots \underset{\mathcal{A}}{\overset{*}{\to}} q_{[1\ldots k]}^{\sqcup} = q$$

for some states $q_{[12]}^{\sqcup}, \ldots, q_{[1\ldots k]}^{\sqcup}$ of $Q_{\sqcup}$.

Assume that $k > M$. The pigeonhole principle yields that $q_{[1,\ldots,j_1]} = q_{[1,\ldots,j_2]}$ for some $1 \leq j_1 < j_2 \leq k$. Therefore the term

$$t = m_1.t_1 \sqcup \ldots \sqcup m_{j_1}.t_{j_1} \sqcup m_{j_2+1}.t_{j_2+1} \sqcup \ldots \sqcup m_k.t_k$$

also reaches the state q which contradicts our hypothesis that k is minimal.

Now, it remains only to use the induction hypothesis to replace each t_i by some s_i reaching the same state and satisfying the required conditions. □

A term s such that for all subterm $n_1.v_1 \sqcup \ldots n_p.v_p$ of s, we have $p \leq M$ and $n_i \leq B$ will be called *small*. We define some extension $\rightarrow^n_{\mathcal{A}}$ of the move relation by:

- $t \rightarrow^1_{\mathcal{A}} q$ if and only if $t \rightarrow_{\mathcal{A}} q$,
- $t \rightarrow^n_{\mathcal{A}} q$ if and only if $t \rightarrow^*_{\mathcal{A}} q$ and
  - either $t = f(t_1, \ldots, t_k)$ and for $i = 1, \ldots, k$ we have $t_i \rightarrow^{n-1}_{\mathcal{A}} q_i(t_i)$,
  - or $t = n_1.t_1 \sqcup \ldots \sqcup n_p.t_p$ and for $i = 1, \ldots, p$, we have $t_i \rightarrow^{n-1}_{\mathcal{A}} q_i(t_i)$.

Let $\mathcal{L}^n_q = \{t \rightarrow^p_{\mathcal{A}} q \mid p \leq n \text{ and } t \text{ is small}\}$ with the convention that $\mathcal{L}^0_q = \emptyset$ and $\mathcal{L}_q = \bigcup_{n=1}^{\infty} \mathcal{L}^n_q$. By Proposition 31, $t \rightarrow_{\mathcal{A}} q$ if and only if there is some $s \in \mathcal{L}_q$ such that $s \rightarrow_{\mathcal{A}} q$. The emptiness decision algorithm will compute a finite approximation $\mathcal{R}^n_q$ of these $\mathcal{L}^n_q$ such that $\mathcal{R}^n_q \neq \emptyset$ if and only if $\mathcal{L}^n_q \neq \emptyset$.

Some technical definition is needed first. Let L be a set of flat term, then we define $\| L \|_P$ as the number of distinct equivalence classes of terms for the $=_P$ relation such that one representant of the class occurs in L. The reader will check easily that the equivalence class of a flat term for the $=_P$ relation is finite.

The decision algorithm is the following one.

begin
  for each state q **do** set $\mathcal{R}^0_q$ to $\emptyset$.
  i=1.
  repeat

    for each state q **do** set $\mathcal{R}^i_q$ to $\mathcal{R}^{i-1}_q$
      if $\| \mathcal{R}^i_q \|_P \leq M$ **then**
        repeat
          add to $\mathcal{R}^i_q$ all flat terms $t = f(t_1, \ldots, t_n)$
          such that $t_j \in \mathcal{R}^{i-1}_{q_j}$, $j \leq n$ and $f(q_1, \ldots, q_n) \rightarrow q \in \Delta$
          add to $\mathcal{R}^i_q$ all flat terms $t = n_1.t_1 \sqcup \ldots \sqcup n_p.t_p$
          such that $p \leq M$,$n_j \leq B$, $t_j \in \mathcal{R}^{i-1}_{q_j}$ and $n_1.q_1 \sqcup \ldots \sqcup n_p.q_p \rightarrow^*_{\mathcal{A}} q$.
        until no new term can be added or $\| \mathcal{R}^i_q \|_P > M$
      endif
      i=i+1
    until $\exists q \in Q_f$ such that $\mathcal{R}^i_q \neq \emptyset$ or $\forall q, \mathcal{R}^i_q = \mathcal{R}^{i-1}_q$

  if $\exists q \in Q_F$ *s.t.*$\mathcal{R}^i_q \neq \emptyset$
  then return *not empty*
  else return *empty* **endif**

end

Proposition 32. *The algorithm terminates after n iterations for some n and $\mathcal{R}_q^n = \emptyset$ if and only if $\mathcal{L}_q = \emptyset$*

Proof. At every iteration, either one $\mathcal{R}_q^i$ increases or else all the $\mathcal{R}_q^i$'s are left untouched in the **repeat** ... **until** loop. Therefore the termination condition will be satisfied after a finite number of iterations, since equivalence classes for $=_P$ are finite.

By construction we have $\mathcal{R}_q^m \subseteq \mathcal{L}_q^m$, but we need the following additional property.

Lemma 7. *For all m,$\mathcal{R}_q^m = \mathcal{L}_q^m$ or $\mathcal{R}_q^m \subseteq \mathcal{L}_q^m$ and $\parallel \mathcal{R}_q^m \parallel_P > M$*

The proof is by induction on m.

Base case $m = 0$. Obvious from the definitions.

Induction step. We assume that the property is true for m and we prove that it holds for $m + 1$.
Either $\mathcal{L}_q^m = \emptyset$ therefore $\mathcal{R}_q^m = \emptyset$ and we are done, or $\mathcal{L}_q^m \neq \emptyset$, which we assume from now on.

- $q \in Q_{\mathcal{F}}$.
  - Either there is some rule $f(q_1, \ldots, q_n) \to q$ such that $\mathcal{R}_{q_i}^m \neq \emptyset$ for all $i = 1, \ldots, n$ and such that for some q' among $q_1, \ldots, q_n$, we have $\parallel \mathcal{R}_{q'}^m \parallel_P > M$. Then we can construct at least $M + 1$ terms $t = f(t_1, \ldots, t', \ldots, t_n)$ where $t' \in \mathcal{R}_{q'}^m$, such that $t \in \mathcal{R}_q^{m+1}$ by giving $M + 1$ non equivalent values to t' (corresponding values for t are also non equivalent). This yields that $\parallel \mathcal{R}_q^{m+1} \parallel_P > M$.
  - Or there is no rule as above, therefore $\mathcal{R}_q^{m+1} = \mathcal{L}_q^{m+1}$.
- $q \in Q_{\sqcup}$.

  For each small term $t = n_1.t_1 \sqcup \ldots \sqcup n_p.t_p$ such that $t \in \mathcal{L}_q^{m+1}$, there are some terms $s_1, \ldots, s_n$ in $\mathcal{R}_{q_i}^m$ such that $t_i \to_{\mathcal{A}}^* q_i$ implies that $s_i \to_{\mathcal{A}}^* q_i$. What we must prove is that $\parallel \mathcal{R}_{q_i}^m \parallel_P > M$ for some $i \leq p$ implies $\parallel \mathcal{R}_q^{m+1} \parallel_P > M$. Since $\mathcal{A}$ is deterministic, we have that $s \to_{\mathcal{A}}^* q$ and $t \to_{\mathcal{A}}^* q'$ with $q \neq q'$ implies that $s \neq_P t$. Let S be the set of states occurring in the sequence $q_1, \ldots, q_p$. We prove by induction on the cardinal of S that if there is some q_i such that $\parallel \mathcal{R}_{q_i}^m \parallel_P > M$, we can build at least $M + 1$ terms in $\mathcal{R}_q^{m+1}$ otherwise we build at least one term of $\mathcal{R}_q^{m+1}$.

  Base case $S = \{q'\}$, and therefore all the q_i are equal to q'. Either $\parallel \mathcal{R}_{q'}^m \parallel_P \leq M$ and we are done or $\parallel \mathcal{R}_{q'}^m \parallel_P > M$ and we know that there are $s_1, \ldots, s_{M+1}, \ldots$ pairwise non equivalent terms reaching q'. Therefore, there are at least $\binom{M+1}{M} \geq M + 1$ different non equivalent possible terms $n_{i_1}.s_{i_1} \sqcup \ldots \sqcup n_{i_p}.s_{i_p}$. Moreover each of these terms S satisfies $s \to_{\mathcal{A}}^{m+1} q$, which proves the result.

Induction step. Let $S = S' \cup \{q'\}$ where the property is true for S'. We can assume that $\| \mathcal{R}^m_{q'} \|_P \leq M$ (otherwise all $\| \mathcal{R}^m_{q_i} \|_P$ are less than or equal to M).

Let $i_1, \ldots, i_k$ be the positions of q' in $q_1, \ldots, q_p$, let $j_1, \ldots, j_l$ be the positions of the states different from q' in $q_1, \ldots, q_p$. By induction hypothesis, there are some flat terms s_j such that $n_{j_1}.s_{j_1} \sqcup \ldots \sqcup n_{j_l}.s_{j_l}$ is a valid flat term. Since $\mathcal{A}$ is deterministic and q' is different from all element of S', we know that $s_i \neq_P s_j$ for any $i \in \{i_1, \ldots, i_k\}$, $j \in \{j_1, \ldots, j_k\}$. Therefore, we use the same reasoning as in the previous case to build at least $C^k_{M+1} \geq M + 1$ pairwise non equivalent flat terms $s = n_1.s_1 \sqcup \ldots \sqcup n_p.s_p$ such that $s \rightarrow^{m+1}_{\mathcal{A}} q$.

The termination of the algorithm implies that for each $m \geq n$, $\mathcal{R}^m_q = \mathcal{L}^m_q$ or $\mathcal{R}^m_q \subseteq \mathcal{L}^m_q$ and $\| \mathcal{R}^m_q \|_P > M$. Therefore $\mathcal{L}_q = \emptyset$ if and only if $\mathcal{R}^n_q = \emptyset$.

□

The following theorem summarizes the previous results.

Theorem 43. *Let $\mathcal{A}$ be a DFTAC, then it is decidable whether the language accepted by $\mathcal{A}$ is empty or not.*

The reader should see that the property that $\mathcal{A}$ *deterministic* is crucial in proving the emptiness decision property. Therefore proving the emptiness of the language recognized by a NFTAC implies to compute an equivalent $DFTAC$ first.

Another point is that the previous algorithm can be easily modified to compute the set of accessible states of $\mathcal{A}$.

4.7 Exercises

Exercise 52.

1. Show that the automaton $\mathcal{A}_+$ of Example 38 accepts only terms of the form $f(t_1, s^n(0), s^m(0), s^{n+m}(0))$
2. Conversely, show that, for every pair of natural numbers (n, m), there exists a term t_1 such that $f(t_1, s^n(0), s^m(0), s^{n+m}(0))$ is accepted by $\mathcal{A}_+$.
3. Construct an automaton $\mathcal{A}_\times$ of the class AWEDC which has the same properties as above, replacing + with $\times$
4. Give a proof that emptiness is undecidable for the class AWEDC, reducing Hilbert's tenth problem.

Exercise 53. Give an automaton of the class AWCBB which accepts the set of terms t (over the alphabet $\{a(0), b(0), f(2)\}$) having a subterm of the form $f(u, u)$. (*i.e.*the set of terms that are reducible by a rule $f(x, x) \rightarrow v$).

Exercise 54. Show that the class AWCBB is not closed under linear tree homomorphisms. Is it closed under inverse image of such morphisms?

Exercise 55. Give an example of two automata in AWCBB such that the set of pairs of terms recognized respectively by the automata is not itself a member of AWCBB.

Exercise 56. (Proposition 24) Show that the class of (languages recognized by) reduction automata is closed under intersection and union. Show that the set of balanced term on alphabet $\{a, f\}$ is not recognizable by a reduction automaton, showing that the class of languages recognized by) reduction automata is not closed under complement.

Exercise 57. Show that the class of languages recognized by reduction automata is preserved under linear tree homomorphisms. Show however that this is no longer true for arbitrary tree homomorphisms.

Exercise 58. Let $\mathcal{A}$ be a reduction automaton. We define a ternary relation $q \xrightarrow{w} q'$ contained in $Q \times \mathbb{N}^* \times Q$ as follows:

- for $i \in \mathbb{N}$, $q \xrightarrow{i} q'$ if and only if there is a rule $f(q_1, \ldots, q_n) \xrightarrow[\mathcal{A}]{c} q'$ with $q_i = q$
- $q \xrightarrow{i \cdot w} q'$ if and only if there is a state q'' such that $q \xrightarrow{i} q''$ and $q'' \xrightarrow{w} q'$.

Moreover, we say that a state $q \in Q$ is a *constrained state* if there is a rule $f(q_1, \ldots, q_n) \xrightarrow[\mathcal{A}]{c} q$ in $\mathcal{A}$ such that c is not a valid constraint.

We say that the *the constraints of $\mathcal{A}$ cannot overlap* if, for each rule $f(q_1, \ldots, q_n) \xrightarrow{c} q$ and for each equality (resp. disequality) $\pi = \pi'$ of c, there is no strict prefix p of π and no constrained state q' such that $q' \xrightarrow{p} q$.

1. Consider the rewrite system on the alphabet $\{f(2), g(1), a(0)\}$ whose left members are $f(x, g(x)), g(g(x)), f(a, a)$. Compute a reduction automaton, whose constraints do not overlap and which accepts the set of irreducible ground terms.
2. Show that emptiness can be decided in polynomial time for reduction automata whose constraints do not overlap. (Hint: it is similar to the proof of Theorem 33.)
3. Show that any language recognized by a reduction automaton whose constraints do not overlap is an homomorphic image of a language in the class AWCBB. Give an example showing that the converse is false.

Exercise 59. Prove the Proposition **??** along the lines of Proposition 15.

Exercise 60. The purpose of this exercise is to give a construction of an automaton with disequality constraints (no equality constraints) whose emptiness is equivalent to the ground reducibility of a given term t with respect to a given term rewriting system $\mathcal{R}$.

1. Give a direct construction of an automaton with disequality constraints $\mathcal{A}_{\mathrm{NF}(\mathcal{R})}$ which accepts the set of irreducible ground terms
2. Show that the class of languages recognized by automata with disequality constraints is closed under intersection. Hence the set of irreducible ground instances of a linear term is recognized by an automaton with disequality constraints.
3. Let $\mathcal{A}_{\mathrm{NF}(\mathcal{R})} = (Q_{\mathrm{NF}}, \mathcal{F}, Q^f_{\mathrm{NF}}, \Delta_{\mathrm{NF}})$. We compute $\mathcal{A}_{\mathrm{NF},t} \stackrel{\mathrm{def}}{=} (Q_{\mathrm{NF},t}, \mathcal{F}, Q^f_{\mathrm{NF},t}, \Delta_{\mathrm{NF},t})$ as follows:
   - $Q_{\mathrm{NF},t} \stackrel{\mathrm{def}}{=} \{t\sigma|_p \mid p \in Pos(t)\} \times Q_{\mathrm{NF}}$ where σ ranges over substitutions from $NLV(t)$ (the set of variables occurring at least twice in t) into Q^f_{NF}.
   - For all $f(q_1, \ldots, q_n) \xrightarrow{c} q \in \Delta_{\mathrm{NF}}$, and all $u_1, \ldots, u_n \in \{t\sigma|_p \mid p \in Pos(t)\}$, $\Delta_{\mathrm{NF},t}$ contains the following rules:

- $f([q_{u_1}, q_1], \ldots, [q_{u_n}, q_n]) \xrightarrow{c \wedge c'} [q_{f(u_1,\ldots,u_n)}, q]$ if $f(u_1, \ldots, u_n) = t\sigma_0$ and c' is constructed as sketched below.
- $f([q_{u_1}, q_1], \ldots, [q_{u_n}, q_n]) \xrightarrow{c} [q_{f(u_1,\ldots,u_n)}, q]$ if $[q_{f(u_1,\ldots,u_n)}, q] \in Q_{\mathrm{NF},t}$ and we are not in the first case.
- $f([q_{u_1}, q_1], \ldots, [q_{u_n}, q_n]) \xrightarrow{c} [q_q, q]$ in all other cases

c' is constructed as follows. From $f(u_1, \ldots, u_n)$ we can retrieve the rules applied at position p in t. Assume that the rule at p checks $\pi_1 \neq \pi_2$. This amounts to check $p\pi_1 \neq p\pi_2$ at the root position of t. Let $\mathcal{D}$ be all disequalities $p\pi_1 \neq p\pi_2$ obtained in this way. The non linearity of t implies some equalities: let $\mathcal{E}$ be the set of equalities $p_1 = p_2$, for all positions p_1, p_2 such that $t|_{p_1} = t|_{p_2}$ is a variable. Now, c' is the set of disequalities $\pi \neq \pi'$ which are not in $\mathcal{D}$ and that can be inferred from $\mathcal{D}, \mathcal{E}$ using the rules

$$pp_1 \neq p_2,\ p = p' \ \vdash\ p'p_1 \neq p_2$$
$$p \neq p',\ pp_1 = p_2 \ \vdash\ p'p_1 \neq p_2$$

For instance, let $t = f(x, f(x, y))$ and assume that the automaton $\mathcal{A}_{\mathrm{NF}}$ contains a rule $f(q, q) \xrightarrow{1\neq 2} q$. Then the automaton $\mathcal{A}_{\mathrm{NF},t}$ will contain the rule $f([q_q, q], [q_{f(q,q)}, q]) \xrightarrow{1\neq 2 \wedge 1\neq 22} q$.

The final states are $[q_u, q_f]$ where $q_f \in Q^f_{\mathrm{NF}}$ and u is an instance of t.
Prove that $\mathcal{A}_{\mathrm{NF},t}$ accepts at least one term if and only if t is not ground reducible by $\mathcal{R}$.

Exercise 61. Prove Theorem 37 along the lines of the proof of Theorem 27.

Exercise 62. Show that the algorithm for deciding emptiness of deterministic complete flat tree automaton works for non-deterministic flat tree automata such that for each state q the number of non-equivalent terms reaching q is 0 or greater than or equal to 2.

Exercise 63. (Feature tree automata)
Let $\mathcal{F}$ be a finite set of feature symbols (or attributes) denoted by $f, g, \ldots$ and $\mathcal{S}$ be a set of constructor symbols (or sorts) denoted by $A, B, \ldots$. In this exercise and the next one, a tree is a rooted directed acyclic graph, a multitree is a tree such that the nodes are labeled over $\mathcal{S}$ and the edges over $\mathcal{F}$. A multitree is either $(A, \emptyset)$ or (A, E) where E is a finite multiset of pairs (f, t) with f a feature and t a multitree. A feature tree is a multitree such that the edges outgoing from the same node are labeled by different features. The $+$ operation takes a multitree $t = (A, E)$, a feature f and a multitree t' to build the multitree $(A, E \cup (f, t'))$ denoted by $t + ft'$.

1. Show that $t + f_1t_1 + f_2t_2 = t + f_2t_2 + f_1t_1$ (OI axiom: order independence axiom) and that the algebra of multitrees is isomorphic to the quotient of the free term algebra over $\{+\} \cup \mathcal{F} \cup \mathcal{S}$ by OI.
2. A deterministic $\mathcal{M}$-automaton is a triple $(\mathcal{A}, h, Q_f)$ where $\mathcal{A}$ is an finite $\{+\} \cup \mathcal{F} \cup \mathcal{S}$-algebra, $h : \mathcal{M} \to \mathcal{A}$ is a homomorphism, Q_f (the final states) is a subset of the set of the values of sort $\mathcal{M}$. A tree is accepted if and only if $h(t) \in Q_f$.
   (a) Show that a $\mathcal{M}$-automaton can be identified with a bottom-up tree automaton such that all trees equivalent under OI reach the same states.
   (b) A feature tree automaton is a $\mathcal{M}$-automaton such that for each sort s ($\mathcal{M}$ or $\mathcal{F}$), for each q the set of the c's of arity 0 interpreted as q in $\mathcal{A}$ is finite or co-finite. Give a feature tree to recognize the set of natural numbers where n is encoded as $(0, \{suc, (0, \{\ldots, (0, \emptyset)\})\})$ with n edges labeled by suc.

(c) Show that the class of languages accepted by feature tree automata is closed under boolean operations and that the emptiness of a language accepted by a feature automaton is decidable.

(d) A non-deterministic feature tree automaton is a tuple (Q, P, h, Q_f) such that Q is the set of states of sort $\mathcal{M}$, P the set of states of sort $\mathcal{F}$, h is composed of three functions $h_1 : \mathcal{S} \to 2^Q$, $h_2 : \mathcal{F} \to 2^P$ and the transition function $+ : Q \times P \times Q \to 2^Q$. Moreover $q + p_1q_1 + p_2q_2 = q + p_2q_2 + p_1q_1$ for each q, q_1, q_2, p_1, p_2, $\{s \in \mathcal{S} \mid p \in h_1(s)\}$ and $\{f \in \mathcal{F} \mid p \in h_2(f)\}$ are finite or co-finite for each p. Show that any non-deterministic feature tree automaton is equivalent to a deterministic feature tree automaton.

Exercise 64. (Characterization of recognizable flat feature languages)
A flat feature tree is a feature tree of depth 1 where depth is defined by $depth((A, \emptyset)) = 0$ and $depth((A, E)) = 1 + max\{depth(t) \mid (f, t) \in E\}$. Counting constraints are defined by:
$$\begin{array}{rcl} C(x) & ::= & card(\varphi \in F \mid \exists y.(x\varphi y) \wedge Ty\}) = n \bmod m \\ & & \mid Sx \\ & & \mid C(x) \vee C(x) \\ & & \mid C(x) \wedge C(x) \end{array}$$
where n, m are integers, S and T finite or co-finite subsets of $\mathcal{S}$, F a finite or co-finite subset of $\mathcal{F}$ and $n \bmod 0$ is defined as n. The semantics of the first type of constraint is: $C(x)$ holds if the number of edges of x going from the root to a node labeled by a symbol of T is equal to $n \bmod m$. The semantics of Sx is: Sx holds if the root of x is labeled by a symbol of S.

1. Show that the constraints are closed under negation. Show that the following constraints can be expressed in the constraint language (F is a finite subset of $\mathcal{F}$, $f \in F$, $A \in \mathcal{S}$): there is one edge labeled f from the root, a given finite subset of $\mathcal{F}$. There is no edge labeled f from the root, the root is labeled by A.
2. A set L of flat multitrees is counting definable if and only if there some counting constraint C such that $L = \{x \mid C(x) \; holds\}$. Show that a set of flat feature trees is counting definable if and only if it is recognizable by a feature tree automaton. hint: identify flat trees with multisets over $(\mathcal{F} \cup \{root\}) \times \mathcal{S}$ and $+$ with multiset union.

4.8 Bibliographic notes

RATEG appeared in Mongy's thesis [Mon81]. Unfortunately, as shown in [Mon81] the emptiness problem is undecidable for the class RATEG (and hence for AWEDC). The undecidability can be even shown for a more restricted class of *automata with equality tests between cousins* (see [Tom92]).
The remarkable subclass AWCBB is defined in [BT92]. This paper presents the results cited in Section 4.3, especially Theorem 33.
Concerning complexity, the result used in Section 4.3.2 (EXPTIME-completeness of the emptiness of the intersection of n recognizable tree languages) may be found in [FSVY91, Sei94b].
[DCC95] is concerned with reduction automata and their use as a tool for the decision of the encompassment theory in the general case.
The first decidability proof for ground reducibility is due to [Pla85]. In [CJ97a], ground reducibility decision is shown EXPTIME-complete. In this work, an EXPTIME algorithm for emptiness decision for AWEDC with only disequality constrained The result mentioned in Section 4.5.

The class of generalized reduction automata is introduced in [CCC+94]. In this paper, a efficient cleaning algorithm is given for emptiness decision.

There have been many work dealing with automata where the width of terms is not bounded. In [Cou89], Courcelle devises an algebraic notion of recognizability and studies the case of equational theories. Then he gives several equational theories corresponding to several notions of trees like ordered or unordered, ranked or unranked trees and provides the tree automata to accept these objects. Actually the axioms used for defining these notions are commutativity (for unordered) or associativity (for unranked) and what is needed is to build tree automata such that all element of the same equivalence class reach the same state. Trees can be also defined as finite, acyclic rooted ordered graphs of bounded degree. Courcelle [Cou92] has devised a notion of recognizable set of graphs and suggests to devise graph automata for accepting recognizable graphs of bounded tree width. He gives such automata for trees defined as unbounded, unordered, undirected, unrooted trees (therefore these are not what we call tree in this book). Actually, he shows that recognizable sets of graphs are (homomorphic image of) sets of equivalence class of terms, where the equivalence relation is the congruence induced by a set of equational axioms including associativity-commutativity axiom and identity element. He gives several equivalent notions for recognizability from which he gets the definitions of automata for accepting recognizable languages. Hedge automata [PQ68, Mur00, BKMW01] are automata that deal with unranked but ordered terms, and use constraint which are membership to some regular word expressions on an alphabet which is the set of states of the automaton. These automata are closed under the boolean operations and emptiness can be decided. Such automata are used for XML applications. Generalization of tree automata with Presburger's constraints can be found in [LD02].

Feature tree are a generalization of first-order trees introduced for modeling record structures. A feature tree is a finite tree whose nodes are labelled by constructor symbols and edges are labelled by feature symbols Niehren and Podelski [NP93] have studied the algebraic structures of feature trees and have devised feature tree automata for recognizing sets of feature trees. They have shown that this class of feature trees enjoys the same properties as regular tree language and they give a characterization of these sets by requiring that the number$n$ of occurrences of a feature f satisfies a Presburger formula $\psi_f(N)$. See Exercise 63 for more details. Equational tree automata, introduced by H.Ohsaki, allow equational axioms to take place during a run. For instance using AC axioms allows to recognize languages which are closed under associativity-commutativity which is not the case of ordinary regular languages. See [Ohs01] for details.

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