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.

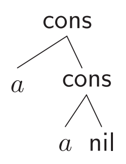
A ***ranked alphabet*** is a couple $(\mathcal{F}, Arity)$ where $\mathcal{F}$ is a finite set and $Arity$ is a mapping from $\mathcal{F}$ into N. The ***arity*** of a symbol $f \in \mathcal{F}$ is $Arity(f)$. The set of symbols of arity p is denoted by $\mathcal{F}_p$. Elements of arity $0, 1, \ldots p$ are respectively called constants, unary, ..., 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 $t \neq t'$.

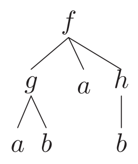
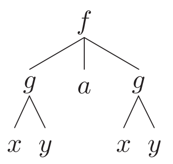
A set of terms F is said to be ***closed*** if it is closed under the subterm ordering, *i.e.*$\forall t \in F$ $(t \trianglerighteq t' \Rightarrow t' \in F)$.

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

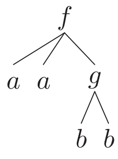
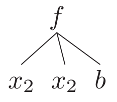
Substitutions

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 5

Tree Set Automata

This chapter introduces a class of automata for sets of terms called *Generalized Tree Set Automata*. Languages associated with such automata are sets of sets of terms. The class of languages recognized by *Generalized Tree Set Automata* fulfills properties that suffices to build automata-based procedures for solving problems involving sets of terms, for instance, for solving systems of set constraints.

5.1 Introduction

"The notion of type expresses the fact that one just cannot apply any operator to any value. Inferring and checking a program's type is then a proof of partial correction" quoting Marie-Claude Gaudel. *"The main problem in this field is to be flexible while remaining rigorous, that is to allow polymorphism (a value can have more than one type) in order to avoid repetitions and write very general programs while preserving decidability of their correction with respect to types."*

On that score, the set constraints formalism is a compromise between power of expression and decidability. This has been the object of active research for a few years.

Set constraints are relations between sets of terms. For instance, let us define the natural numbers with 0 and the successor relation denoted by s. Thus, the constraint

$$\mathsf{Nat} = 0 \cup s(\mathsf{Nat}) \tag{5.1}$$

corresponds to this definition. Let us consider the following system:

$$\begin{array}{rcl} \mathsf{Nat} & = & 0 \cup s(\mathsf{Nat}) \\ \mathsf{List} & = & cons(\mathsf{Nat}, \mathsf{List}) \cup \mathsf{nil} \\ \mathsf{List}_+ & \subseteq & \mathsf{List} \\ \mathsf{car}(\mathsf{List}_+) & \subseteq & s(\mathsf{Nat}) \end{array} \tag{5.2}$$

The first constraint defines natural numbers. The second constraint codes the set of LISP-like lists of natural numbers. The empty list is nil and other lists are obtained using the constructor symbol cons. The last two constraints represent the set of lists with a non zero first element. Symbol car has the usual interpretation: the head of a list. Here $\mathsf{car}(\mathsf{List}_+)$ can be interpreted as the set of all terms at first position in List_+, that is all terms t such that there exists u with $\mathsf{cons}(t, u) \in \mathsf{List}_+$. In the set constraint framework such an operator car is often written cons_1^{-1}.

Set constraints are the essence of Set Based Analysis. The basic idea is to reason about program variables as sets of possible values. Set Based Analysis involves first writing set constraints expressing relationships between sets of program values, and then solving the system of set constraints. A single approximation is: all dependencies between the values of program variables are ignored. Techniques developed for Set Based Analysis have been successfully applied in program analysis and type inference and the technique can be combined with others [HJ92].

Set constraints have also been used to define a constraint logic programming language over sets of ground terms that generalizes ordinary logic programming over an Herbrand domain [Koz98].

In a more general way, a *system of set constraints* is a conjunction of *positive* constraints of the form $exp \subseteq exp'$[1] and *negative* constraints of the form $exp \not\subseteq exp'$. Right hand side and left hand side of these inequalities are *set expressions*, which are built with

- function symbols: in our example 0, s, cons, nil are function symbols.
- operators: union $\cup$, intersection $\cap$, complement $\sim$
- projection symbols: for instance, in the last equation of system (5.2) car denotes the first component of cons. In the set constraints syntax, this is written $\mathsf{cons}_{(1)}^{-1}$.
- set variables like Nat or List.

An interpretation assigns to each set variable a set of terms only built with function symbols. A solution is an interpretation which satisfies the system. For example, $\{0, s(0), s(s(0)), \dots\}$ is a solution of Equation (5.1).

In the set constraint formalism, set inclusion and set union express in a natural way parametric polymorphism: $\mathsf{List} \subseteq \mathsf{nil} \cup \mathsf{cons}(X, \mathsf{List})$.

In logic or functional programming, one often use dynamic procedures to deal with type. In other words, a run-time procedure checks whether or not an expression is well-typed. This permits maximum programming flexibility at the potential cost of efficiency and security. Static analysis partially avoids these drawbacks with the help of type inference and type checking procedures. The information extracted at compile time is also used for optimization.

Basically, program sources are analyzed at compile time and an ad hoc formalism is used to represent the result of the analysis. For types considered as sets of values, the set constraints formalism is well suited to represent them and to express their relations. Numerous inference and type checking algorithms in logic, functional and imperative programming are based on a resolution procedure for set constraints.

[1] $exp = exp'$ for $exp \subseteq exp' \wedge exp' \subseteq exp$.

Most of the earliest algorithms consider systems of set constraints with weak power of expression. More often than not, these set constraints always have a least solution — *w.r.t.* inclusion — which corresponds to a (tuple of) regular set of terms. In this case, types are usual sorts. A sort signature defines a tree automaton (see Section 3.4.1 for the correspondence between automata and sorts). For instance, regular equations iontroduced in Section 2.3 such a subclass of set constraints. Therefore, these methods are closely related finite tree automata and use classical algorithms on these recognizers, like the ones presented in Chapter 1.

In order to obtain a more precise information with set constraints in static analysis, one way is to enrich the set constraints vocabulary. In one hand, with a large vocabulary an analysis can be accurate and relevant, but on the other hand, solutions are difficult to obtain.

Nonetheless, an essential property must be preserved: the decidability of satisfiability. There must exists a procedure which determines whether or not a system of set constraints has solutions. In other words, extracted information must be sufficient to say whether the objects of an analyzed program have a type. It is crucial, therefore, to know which classes of set constraints are decidable, and identifying the complexity of set constraints is of paramount importance.

A second important characteristic to preserve is to represent solutions in a convenient way. We want to obtain a kind of solved form from which one can decide whether a system has solutions and one can "compute" them.

In this chapter, we present an automata-based algorithm for solving systems of positive and negative set constraints where no projection symbols occurs. We define a new class of automata recognizing sets of (codes of) n-tuples of tree languages. Given a system of set constraints, there exists an automaton of this class which recognizes the set of solutions of the system. Therefore properties of our class of automata directly translate to set constraints.

In order to introduce our automata, we discuss the case of unary symbols, *i.e.*the case of strings over finite alphabet. For instance, let us consider the following constraints over the alphabet composed of two unary symbols a and b and a constant 0:

$$\begin{aligned} Xaa \cup Xbb &\subseteq X \\ Y &\subseteq X \end{aligned} \tag{5.3}$$

This system of set constraints can be encoded in a formula of the monadic second order theory of 2 successors named a and b:

$$\begin{aligned} &\forall u\ (u \in X \Rightarrow (uaa \in X \wedge ubb \in X))\wedge \\ &\forall u\ u \in Y \Rightarrow u \in X \end{aligned}$$

We have depicted in Fig 5.1 (a beginning of) an infinite tree which is a model of the formula. Each node corresponds to a string over a and b. The root is associated with the empty string; going down to the left concatenates a a; going down to the right concatenates a b. Each node of the tree is labelled with a couple of points. The two components correspond to sets X and Y. A black point in the first component means that the current node belongs to X. Conversely, a white point in the first component means that the current node does not belong to X. Here we have $X = \{\varepsilon, aa, bb, \dots\}$ and $Y = \{\varepsilon, bb, \dots\}$.

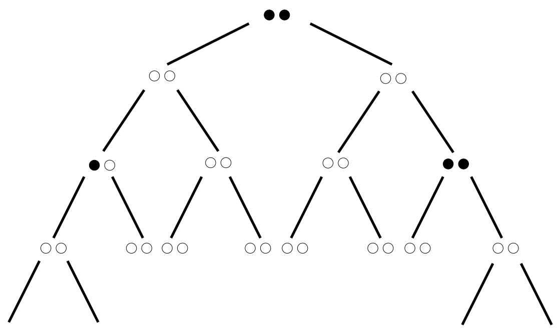


Figure 5.1: An infinite tree for the representation of a couple of word languages (X, Y). Each node is associated with a word. A black dot stands for belongs to. $X = \{\varepsilon, aa, bb, \dots\}$ and $Y = \{\varepsilon, bb, \dots\}$.

A tree language that encodes solutions of Eq. 5.3 is Rabin-recognizable by a tree automaton which must avoid the three forbidden patterns depicted in Figure 5.2.

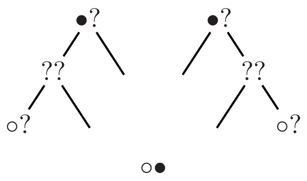


Figure 5.2: The set of three forbidden patterns. '?' stands for black or white dot. The tree depicted in Fig. 5.1 exclude these three patterns.

Given a ranked alphabet of unary symbols and one constant and a system of set constraints over $\{X_1, \dots, X_n\}$, one can encode a solution with a $\{0,1\}^n$-valued infinite tree and the set of solutions is recognized by an infinite tree automaton. Therefore, decidability of satisfiability of systems of set constraints can easily be derived from Rabin's Tree Theorem [Rab69] because infinite tree automata can be considered as an acceptor model for n-tuples of word languages over finite alphabet[2].

We extend this method to set constraints with symbols of arbitrary arity. Therefore, we define an acceptor model for mappings from $T(\mathcal{F})$, where $\mathcal{F}$ is a ranked alphabet, into a set $E = \{0,1\}^n$ of labels. Our automata can be viewed as an extension of infinite tree automata, but we will use weaker acceptance condition. The acceptance condition is: the range of a successful run is in a specified set of accepting set of states. We will prove that we can design an

[2]The entire class of Rabin's tree languages is not captured by solutions of set of words constraints. Set of words constraints define a class of languages which is strictly smaller than Büchi recognizable tree languages.

automaton which recognizes the set of solutions of a system of both positive and negative set constraints. For instance, let us consider the following system:

$$Y \not\subseteq \perp \tag{5.4}$$
$$X \subseteq f(Y, \sim X) \cup a \tag{5.5}$$

where $\perp$ stands for the empty set and $\sim$ stands for the complement symbol.

The underlying structure is different than in the previous example since it is now the whole set of terms on the alphabet composed of a binary symbol f and a constant a. Having a representation of this structure in mind is not trivial. One can imagine a directed graph whose vertices are terms and such that there exists an edge between each couple of terms in the direct subterm relation (see figure 5.3).

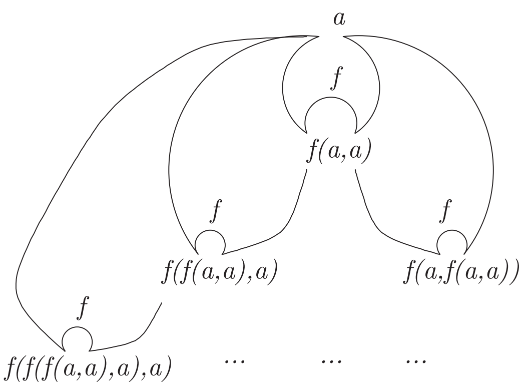


Figure 5.3: The (beginning of the) underlying structure for a two letter alphabet $\{f(,), a\}$.

An automaton have to associate a state with each node following a finite set of rules. In the case of the example above, states are also couples of $\bullet$ or $\circ$.

Each vertex is of infinite out-degree, nonetheless one can define as in the word case forbidden patterns for incoming vertices which such an automaton have to avoid in order to satisfy Eq. (5.5) (see Fig. 5.4, Pattern ? stands for $\circ$ or $\bullet$). The acceptance condition is illustrated using Eq. (5.4). Indeed, to describe a solution of the system of set constraints, the pattern ?$\bullet$ must occur somewhere in a successful "run" of the automaton.

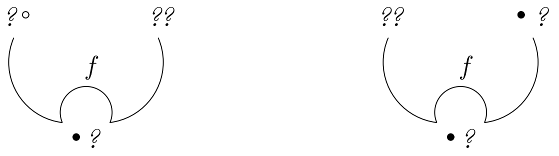


Figure 5.4: Forbidden patterns for (5.5).

Consequently, decidability of systems of set constraints is a consequence of decidability of emptiness in our class of automata. Emptiness decidability is easy for automata without acceptance conditions (it corresponds to the case of positive set constraints only). The proof is more difficult and technical in the general case and is not presented here. Moreover, and this is the main advantage of an automaton-based method, properties of recognizable sets directly translate to sets of solutions of systems of set constraints. Therefore, we are able to prove nice properties. For instance, we can prove that a non empty set of solutions always contain a regular solution. Moreover we can prove the decidability of existence of finite solutions.

5.2 Definitions and Examples

Infinite tree automata are an acceptor model for infinite trees, *i.e.*for mappings from A^* into E where A is a finite alphabet and E is a finite set of labels. We define and study $\mathcal{F}$-generalized tree set automata which are an acceptor model for mappings from $T(\mathcal{F})$ into E where $\mathcal{F}$ is a finite ranked alphabet and E is a finite set of labels.

5.2.1 Generalized Tree Sets

Let $\mathcal{F}$ be a ranked alphabet and E be a finite set. An ***E-valued $\mathcal{F}$-generalized tree set*** g is a mapping from $T(\mathcal{F})$ into E. We denote by $\mathcal{G}_E$ the set of E-valued $\mathcal{F}$-generalized tree sets.

For the sake of brevity, we do not mention the signature $\mathcal{F}$ which strictly speaking is in order in generalized tree sets. We also use the abbreviation GTS for generalized tree sets.

Throughout the chapter, if $c \in \{0,1\}^n$, then c_i denotes the i^{th} component of the tuple c. If we consider the set $E = \{0,1\}^n$ for some n, a generalized tree set g in $\mathcal{G}_{\{0,1\}^n}$ can be considered as a n-tuple $(L_1, \ldots, L_n)$ of tree languages over the ranked alphabet $\mathcal{F}$ where $L_i = \{t \in T(\mathcal{F}) \mid g(t)_i = 1\}$.

We will need in the chapter the following operations on generalized tree sets. Let g (resp. g') be a generalized tree set in $\mathcal{G}_E$ (resp. $\mathcal{G}_{E'}$). The generalized tree set $g \uparrow g' \in \mathcal{G}_{E\times E'}$ is defined by $g \uparrow g'(t) = (g(t), g'(t))$, for each term t in $T(\mathcal{F})$. Conversely let g be a generalized tree set in $\mathcal{G}_{E\times E'}$ and consider the projection π from $E \times E'$ into the E-component then $\pi(g)$ is the generalized tree set in $\mathcal{G}_E$ defined by $\pi(g)(t) = \pi(g(t))$. Let $G \subseteq \mathcal{G}_{E\times E'}$ and $G' \subseteq \mathcal{G}_E$, then $\pi(G) = \{\pi(g) \mid g \in G\}$ and $\pi^{-1}(G') = \{g \in \mathcal{G}_{E\times E'} \mid \pi(g) \in G'\}$.

5.2.2 Tree Set Automata

A ***generalized tree set automaton*** $\mathcal{A} = (Q, \Delta, \Omega)$ (GTSA) over a finite set E consist of a finite state set Q, a transition relation $\Delta \subseteq \bigcup_p Q^p \times \mathcal{F}_p \times E \times Q$ and a set $\Omega \subseteq 2^Q$ of accepting sets of states.

A ***run*** of $\mathcal{A}$ (or $\mathcal{A}$-run) on a generalized tree set $g \in \mathcal{G}_E$ is a mapping $r : T(\mathcal{F}) \to Q$ with:

$$(r(t_1), \ldots, r(t_p), f, g(f(t_1, \ldots, t_p)), r(f(t_1, \ldots, t_p))) \in \Delta$$

for $t_1, \ldots, t_p \in T(\mathcal{F})$ and $f \in \mathcal{F}_p$. The run r is ***successful*** if the range of r is in Ω *i.e.*$r(T(\mathcal{F})) \in \Omega$.

A generalized tree set $g \in \mathcal{G}_E$ is ***accepted*** by the automaton $\mathcal{A}$ if some run r of $\mathcal{A}$ on g is successful. We denote by $\mathcal{L}(\mathcal{A})$ the set of E-valued generalized tree sets accepted by a generalized tree set automaton $\mathcal{A}$ over E. A set $G \subseteq \mathcal{G}_E$ is recognizable if $G = \mathcal{L}(\mathcal{A})$ for some generalized tree set automaton $\mathcal{A}$.

In the following, a rule $(q_1, \dots, q_p, f, l, q)$ is also denoted by $f(q_1, \dots, q_p) \xrightarrow{l} q$. Consider a term $t = f(t_1, \dots, t_p)$ and a rule $f(q_1, \dots, q_p) \xrightarrow{l} q$, this rule can be applied in a run r on a generalized tree set g for the term t if $r(t_1) = q_1$,...,$r(t_p) = q_p$, t is labeled by l, *i.e.*$g(t) = l$. If the rule is applied, then $r(t) = q$.

A generalized tree set automaton $\mathcal{A} = (Q, \Delta, \Omega)$ over E is

- ***deterministic*** if for each tuple $(q_1, \dots, q_p, f, l) \in Q^p \times \mathcal{F}_p \times E$ there is at most one state $q \in Q$ such that $(q_1, \dots, q_p, f, l, q) \in \Delta$.
- ***strongly deterministic*** if for each tuple $(q_1, \dots, q_p, f) \in Q^p \times \mathcal{F}_p$ there is at most one pair $(l, q) \in E \times Q$ such that $(q_1, \dots, q_p, f, l, q) \in \Delta$.
- ***complete*** if for each tuple $(q_1, \dots, q_p, f, l) \in Q^p \times \mathcal{F}_p \times E$ there is at least one state $q \in Q$ such that $(q_1, \dots, q_p, f, l, q) \in \Delta$.
- ***simple*** if Ω is "subset-closed", that is $\omega \in \Omega \Rightarrow (\forall \omega' \subseteq \omega\ \omega' \in \Omega)$.

Successfulness for simple automata just implies some states are *not* assumed along a run. For instance, if the accepting set of a GTSA $\mathcal{A}$ is $\Omega = 2^Q$ then $\mathcal{A}$ is simple and any run is successful. But, if $\Omega = \{Q\}$, then $\mathcal{A}$ is not simple and each state must be assumed at least once in a successful run. The definition of simple automata will be clearer with the relationships with set constraints and the emptiness property (see Section 5.4). Briefly, positive set constraints are related to simple GTSA for which the proof of emptiness decision is straightforward. Another and equivalent definition for simple GTSA relies on the acceptance condition: a run r is successful if and only if $r(T(\mathcal{F})) \subseteq \omega \in \Omega$.

There is in general an *infinite* number of runs — and hence an *infinite* number of GTS recognized — even in the case of deterministic generalized tree set automata (see example 49.2). Nonetheless, given a GTS g, there is at most one run on g for a deterministic generalized tree set automata. But, in the case of *strongly* deterministic generalized tree set automata, there is at most one run (see example 49.1) and therefore there is at most one GTS recognized.

Example 49.

Ex. 49.1 Let $E = \{0, 1\}$, $\mathcal{F} = \{\mathsf{cons}(,), s(), \mathsf{nil}, 0\}$. Let $\mathcal{A} = (Q, \Delta, \Omega)$ be defined by $Q = \{\mathsf{Nat}, \mathsf{List}, \mathsf{Term}\}$, $\Omega = 2^Q$, and Δ is the following set of rules:

$$0 \xrightarrow{0} \mathsf{Nat}\ ;\ s(\mathsf{Nat}) \xrightarrow{0} \mathsf{Nat}\ ;\quad nil \xrightarrow{1} \mathsf{List}\quad ;$$
$$\mathsf{cons}(\mathsf{Nat}, \mathsf{List}) \xrightarrow{1} \mathsf{List}\ ;$$
$$\mathsf{cons}(q, q') \xrightarrow{0} \mathsf{Term}\quad \forall (q, q') \neq (\mathsf{Nat}, \mathsf{List})\ ;$$
$$s(q) \xrightarrow{0} \mathsf{Term}\quad \forall q \neq \mathsf{Nat}\ .$$

$\mathcal{A}$ is strongly deterministic, simple, and not complete. $\mathcal{L}(\mathcal{A})$ is a singleton set. Indeed, there is a unique run r on a unique generalized tree set $g \in \mathcal{G}_{\{0,1\}^n}$. The run r maps every natural number on state Nat, every list on state List and the other terms on state Term. Therefore g maps a natural number on 0, a list on 1 and the other terms on 0. Hence, we say that $\mathcal{L}(\mathcal{A})$ is the regular tree language L of Lisp-like lists of natural numbers.

Ex. 49.2 Let $E = \{0, 1\}$, $\mathcal{F} = \{\mathsf{cons}(,), s(), \mathsf{nil}, 0\}$, and let $\mathcal{A}' = (Q', \Delta', \Omega')$ be defined by $Q' = Q$, $\Omega' = \Omega$, and

$$\Delta' = \Delta \cup \{\mathsf{cons}(\mathsf{Nat}, \mathsf{List}) \xrightarrow{0} \mathsf{List}, \mathsf{nil} \xrightarrow{0} \mathsf{List}\}.$$

$\mathcal{A}'$ is deterministic (but not strongly), simple, and not complete, and $\mathcal{L}(\mathcal{A}')$ is the set of all subsets of the regular tree language L of Lisp-like lists of natural numbers. Indeed, successful runs can now be defined on generalized tree sets g such that a term in L is labeled by 0 or 1.

Ex. 49.3 Let $E = \{0, 1\}^2$, $\mathcal{F} = \{\mathsf{cons}(,), s(), \mathsf{nil}, 0\}$, and let $\mathcal{A} = (Q, \Delta, \Omega)$ be defined by $Q = \{\mathsf{Nat}, \mathsf{Nat}', \mathsf{List}, \mathsf{Term}\}$, $\Omega = 2^Q$, and Δ is the following set of rules:

$$\begin{array}{ccc}
0 \xrightarrow{(0,0)} \mathsf{Nat} & ; \quad 0 \xrightarrow{(1,0)} \mathsf{Nat}' & ; \; s(\mathsf{Nat}) \xrightarrow{(0,0)} \mathsf{Nat} \\
s(\mathsf{Nat}) \xrightarrow{(1,0)} \mathsf{Nat}' & ; \quad s(\mathsf{Nat}') \xrightarrow{(0,0)} \mathsf{Nat} & ; \; s(\mathsf{Nat}') \xrightarrow{(1,0)} \mathsf{Nat}' \\
\mathsf{nil} \xrightarrow{(0,1)} \mathsf{List} & ; \; \mathsf{cons}(\mathsf{Nat}', \mathsf{List}) \xrightarrow{(0,1)} \mathsf{List} & ; \\
\end{array}$$
$$s(q) \xrightarrow{(0,0)} \mathsf{Term} \quad \forall q \neq \mathsf{Nat}$$
$$\mathsf{cons}(q, q') \xrightarrow{(0,0)} \mathsf{Term} \quad \forall (q, q') \neq (\mathsf{Nat}', \mathsf{List})$$

$\mathcal{A}$ is deterministic, simple, and not complete, and $\mathcal{L}(\mathcal{A})$ is the set of 2-tuples of tree languages (N', L') where N' is a subset of the regular tree language of natural numbers and L' is the set of Lisp-like lists of natural numbers over N'.

Let us remark that the set N' may be non-regular. For instance, one can define a run on a characteristic generalized tree set g_p of Lisp-like lists of prime numbers. The generalized tree set g_p is such that $g_p(t) = (1, 0)$ when t is a (code of a) prime number.

In the previous examples, we only consider simple generalized tree set automata. Moreover all runs are successful runs. The following examples are non-simple generalized tree set automata in order to make clear the interest of acceptance conditions. For this, compare the sets of generalized tree sets obtained in examples 49.3 and 50 and note that with acceptance conditions, we can express that a set is non empty.

Example 50. *Example 49.3 continued*
Let $E = \{0, 1\}^2$, $\mathcal{F} = \{\mathsf{cons}(,), \mathsf{nil}, s(), 0\}$, and let $\mathcal{A}' = (Q', \Delta', \Omega')$ be defined by $Q' = Q$, $\Delta' = \Delta$, and $\Omega' = \{\omega \in 2^Q \mid \mathsf{Nat}' \in \omega\}$. $\mathcal{A}'$ is deterministic, not simple, and not complete, and $\mathcal{L}(\mathcal{A}')$ is the set of 2-tuples of tree languages (N', L') where N' is a subset of the regular tree language of natural numbers and L' is the set of Lisp-like lists of natural numbers over N', and $N' \neq \emptyset$. Indeed, for a successful r on g, there must be a term t such that $r(t) = \mathsf{Nat}'$ therefore, there must be a term t labelled by $(1, 0)$, henceforth $N' \neq \emptyset$.

5.2.3 Hierarchy of GTSA-recognizable Languages

Let us define:

- $\mathcal{R}_{\mathrm{GTS}}$, the class of languages recognizable by GTSA,
- $\mathcal{R}_{\mathrm{DGTS}}$, the class of languages recognizable by deterministic GTSA,
- $\mathcal{R}_{\mathrm{SGTS}}$, the class of languages recognizable by Simple GTSA.

The three classes defined above are proved to be different. They are also closely related to classes of languages defined from the set constraint theory point of view.

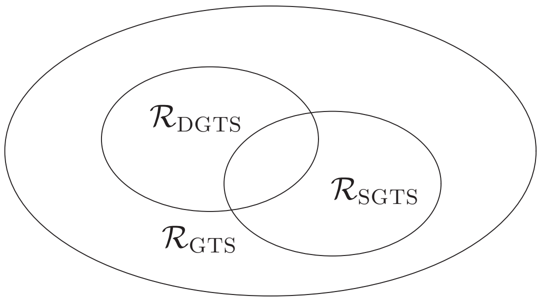

$\mathcal{R}_{\mathrm{DGTS}}$
$\mathcal{R}_{\mathrm{SGTS}}$
$\mathcal{R}_{\mathrm{GTS}}$

Figure 5.5: Classes of GTSA-recognizable languages

Classes of GTSA-recognizable languages have also different closure properties. We will prove in Section 5.3.1 that $\mathcal{R}_{\mathrm{SGTS}}$ and the entire class $\mathcal{R}_{\mathrm{GTS}}$ are closed under union, intersection, projection and cylindrification; $\mathcal{R}_{\mathrm{DGTS}}$ is closed under complementation and intersection.

We propose three examples that illustrate the differences between the three classes. First, $\mathcal{R}_{\mathrm{DGTS}}$ is not a subset of $\mathcal{R}_{\mathrm{SGTS}}$.

Example 51. Let $E = \{0, 1\}$, $\mathcal{F} = \{f, a\}$ where a is a constant and f is unary. Let us consider the deterministic but non-simple GTSA $\mathcal{A}_1 = (\{q_0, q_1\}, \Delta_1, \Omega_1)$ where Δ_1 is:

$$\begin{array}{ll} a \xrightarrow{0} q_0, & a \xrightarrow{1} q_1, \\ f(q_0) \xrightarrow{0} q_0, & f(q_1) \xrightarrow{0} q_0, \\ f(q_0) \xrightarrow{1} q_1, & f(q_1) \xrightarrow{1} q_0. \end{array}$$

and $\Omega_1 = \{\{q_0, q_1\}, \{q_1\}\}$. Let us prove that

$$\mathcal{L}(\mathcal{A}_1) = \{L \mid L \neq \emptyset\}$$

is not in $\mathcal{R}_{\mathrm{SGTS}}$.

Assume that there exists a simple GTSA $\mathcal{A}_s$ with n states such that $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_s)$. Hence, $\mathcal{A}_s$ recognizes also each one of the singleton sets $\{f^i(a)\}$ for $i > 0$. Let us consider some i greater than $n + 1$, we can deduce that a run r on the GTS g associated with $\{f^i(a)\}$ maps two terms $f^k(a)$ and $f^l(a)$, $k < l < i$ to the same state. We have $g(t) = 0$ for every term $t \trianglelefteq f^l(a)$ and r "loops" between $f^k(a)$ and $f^l(a)$. Therefore, one can build another run r_0 on a GTS g_0 such that $g_0(t) = 0$ for each $t \in T(\mathcal{F})$. Since $\mathcal{A}_s$ is simple, and since the range of r_0 is a subset of the range of r, g_0 is recognized, hence the empty set is recognized which contradicts the hypothesis.

Basically, using simple GTSA it is not possible to enforce a state to be assumed somewhere by every run. Consequently, it is not possible to express global properties of generalized tree languages such as non-emptiness.

Second, $\mathcal{R}_{\mathrm{SGTS}}$ is not a subset of $\mathcal{R}_{\mathrm{DGTS}}$.

Example 52. Let us consider the non-deterministic but simple GTSA $\mathcal{A}_2 = (\{q_f, q_h\}, \Delta_2, \Omega_2)$ where Δ_2 is:

$$\begin{array}{ll} a \xrightarrow{0} q_f \mid q_h, & a \xrightarrow{1} q_f \mid q_h, \\ f(q_f) \xrightarrow{1} q_f \mid q_h, & h(q_h) \xrightarrow{1} q_f \mid q_h, \\ f(q_h) \xrightarrow{0} q_f \mid q_h, & h(q_f) \xrightarrow{0} q_f \mid q_h, \end{array}$$

and $\Omega_2 = 2^{\{q_f, q_h\}}$. It is easy to prove that $\mathcal{L}(\mathcal{A}_2) = \{L \mid \forall t\ f(t) \in L \Leftrightarrow h(t) \notin L\}$. The proof that no deterministic GTSA recognizes $\mathcal{L}(\mathcal{A}_2)$ is left to the reader.

We terminate with an example of a non-deterministic and non-simple generalized tree set automaton. This example will be used in the proof of Proposition 36.

Example 53. Let $\mathcal{A} = (Q, \Delta, \Omega)$ be defined by $Q = \{q, q'\}$, $\Omega = \{Q\}$, and Δ is the following set of rules:

$$\begin{array}{llll} a \xrightarrow{1} q & ;\ a \xrightarrow{1} q' & ;\ a \xrightarrow{0} q' & ;\ f(q) \xrightarrow{1} q\ ; \\ f(q') \xrightarrow{0} q' & ;\ f(q') \xrightarrow{1} q' & ;\ f(q') \xrightarrow{1} q\ ; & \end{array}$$

The proof that $\mathcal{A}$ is not deterministic, not simple, and not complete, and $\mathcal{L}(\mathcal{A}) = \{L \subseteq T(\mathcal{F}) \mid \exists t \in T(\mathcal{F})\ ((t \in L) \wedge (\forall t' \in T(\mathcal{F})\ (t \trianglelefteq t') \Rightarrow (t' \in L)))\}$ is left as an exercise to the reader.

5.2.4 Regular Generalized Tree Sets, Regular Runs

As we mentioned it in Example 49.3, the set recognized by a GTSA may contain GTS corresponding to non-regular languages. But regularity is of major interest for practical reasons because it implies a GTS or a language to be finitely defined.

A generalized tree set $g \in \mathcal{G}_E$ is ***regular*** if there exist a finite set R, a mapping $\alpha : T(\mathcal{F}) \to R$, and a mapping $\beta : R \to E$ satisfying the following two properties.

1. $g = \alpha\beta$ $(i.e. g = \beta \circ \alpha)$,

2. α is closed under contexts, *i.e.*for all context c and terms t_1, t_2, we have

$$(\alpha(t_1) = \alpha(t_2)) \Rightarrow (\alpha(c[t_1]) = \alpha(c[t_2]))$$

In the case $E = \{0, 1\}^n$, regular generalized tree sets correspond to n-tuples of regular tree languages.

Although the definition of regularity could lead to the definition of regular run — because a run can be considered as a generalized tree set in $\mathcal{G}_Q$, we use stronger conditions for a run to be regular. Indeed, if we define regular runs as regular generalized tree sets in $\mathcal{G}_Q$, regularity of generalized tree sets and regularity of runs do not correspond in general. For instance, one could define regular runs on non-regular generalized tree sets in the case of non-strongly deterministic generalized tree set automata, and one could define non-regular runs on regular generalized tree sets in the case of non-deterministic generalized tree set automata. Therefore, we only consider regular runs on regular generalized tree sets:

> *A* ***run*** *r* ***on a generalized tree set*** *g* ***is regular*** *if $r \uparrow g \in \mathcal{G}_{E\times Q}$ is regular. Consequently, r and g are regular generalized tree sets.*

Proposition 33. *Let $\mathcal{A}$ be a generalized tree set automaton, if g is a regular generalized tree set in $\mathcal{L}(\mathcal{A})$ then there exists a regular $\mathcal{A}$-run on g.*

Proof. Consider a generalized tree set automaton $\mathcal{A} = (Q, \Delta, \Omega)$ over E and a regular generalized tree set g in $\mathcal{L}(\mathcal{A})$ and let r be a successful run on g. Let L be a finite tree language closed under the subterm relation and such that $\mathcal{F}_0 \subseteq L$ and $r(L) = r(T(\mathcal{F}))$. The generalized tree set g is regular, therefore there exist a finite set R, a mapping $\alpha : T(\mathcal{F}) \to R$ closed under context and a mapping $\beta : R \to E$ such that $g = \alpha\beta$. We now define a regular run r' on g.

Let $L_\star = L \cup \{\star\}$ where $\star$ is a new constant symbol and let ϕ be the mapping from $T(\mathcal{F})$ into $Q \times R \times L_\star$ defined by $\phi(t) = (r(t), \alpha(t), u)$ where $u = t$ if $t \in L$ and $u = \star$ otherwise. Hence $R' = \phi(T(\mathcal{F}))$ is a finite set because $R' \subseteq Q \times R \times L_\star$. For each ρ in R', let us fix $t_\rho \in T(\mathcal{F})$ such that $\phi(t_\rho) = \rho$.

The run r' is now (regularly) defined via two mappings α' and β'. Let β' be the projection from $Q \times R \times L_\star$ into Q and let $\alpha' : T(\mathcal{F}) \to R'$ be inductively defined by:

$$\forall a \in \mathcal{F}_0 \; \alpha'(a) = \phi(a);$$

and

$$\forall f \in \mathcal{F}_p \forall t_1, \dots, t_p \in T(\mathcal{F})$$

$$\alpha'(f(t_1, \dots, t_p)) = \phi(f(t_{\alpha'(t_1)}, \dots, t_{\alpha'(t_p)})).$$

Let $r' = \alpha'\beta'$. First we can easily prove by induction that $\forall t \in L \; \alpha'(t) = \phi(t)$ and deduce that $\forall t \in L \; r'(t) = r(t)$. Thus r' and r coincide on L. It remains to prove that (1) the mapping α' is closed under context, (2) r' is a run on g and (3) r' is a successful run.

(1) From the definition of α' we can easily derive that the mapping α' is closed under context.

(2) We prove that the mapping $r' = \alpha'\beta'$ is a run on g, that is if $t = f(t_1, \ldots, t_p)$ then $(r'(t_1), \ldots, r'(t_p), f, g(t), r'(t)) \in \Delta$.

Let us consider a term $t = f(t_1, \ldots, t_p)$. From the definitions of α', β', and r', we get $r'(t) = r(t')$ with $t' = f(t_{\alpha'(t_1)}, \ldots, t_{\alpha'(t_p)})$. The mapping r is a run on g, hence $(r(t_{\alpha'(t_1)}), \ldots, r(t_{\alpha'(t_p)}), f, g(t'), r(t')) \in \Delta$, and thus it suffices to prove that $g(t) = g(t')$ and, for all i, $r'(t_i) = r(t_{\alpha'(t_i)})$.

Let $i \in \{1, \ldots, p\}$, $r'(t_i) = \beta'(\alpha'(t_i))$ by definition of r'. By definition of $t_{\alpha'(t_i)}$, $\alpha'(t_i) = \phi(t_{\alpha'(t_i)})$, therefore $r'(t_i) = \beta'(\phi(t_{\alpha'(t_i)}))$. Now, using the definitions of ϕ and β', we get $r'(t_i) = r(t_{\alpha'(t_i)})$.

In order to prove that $g(t) = g(t')$, we prove that $\alpha(t) = \alpha(t')$. Let π be the projection from R' into R. We have $\alpha(t') = \pi(\phi(t'))$ by definition of ϕ and π. We have $\alpha(t') = \pi(\alpha'(t))$ using definitions of t' and α'. Now $\alpha(t') = \pi(\phi(t_{\alpha'(t)}))$ because $\phi(t_{\alpha'(t)}) = \alpha'(t)$ by definition of $t_{\alpha'(t)}$. And then $\alpha(t') = \alpha(t_{\alpha'(t)})$ by definition of π and ϕ. Therefore it remains to prove that $\alpha(t_{\alpha'(t)}) = \alpha(t)$. The proof is by induction on the structure of terms.

If $t \in \mathcal{F}_0$ then $t_{\alpha'(t)} = t$, so the property holds (note that this property holds for all $t \in L$). Let us suppose that $t = f(t_1, \ldots, t_p)$ and $\alpha(t_{\alpha'(t_i)}) = \alpha(t_i)\ \forall i \in \{1, \ldots, p\}$. First, using induction hypothesis and closure under context of α, we get

$$\alpha(f(t_1, \ldots, t_p)) \quad = \quad \alpha(f(t_{\alpha'(t_1)}, \ldots, t_{\alpha'(t_p)}))$$

Therefore,

$$\begin{aligned}
\alpha(f(t_1, \ldots, t_p)) &= \alpha(f(t_{\alpha'(t_1)}, \ldots, t_{\alpha'(t_p)})) \\
&= \pi(\phi(f(t_{\alpha'(t_1)}, \ldots, t_{\alpha'(t_p)}))) \text{ (def. of } \phi \text{ and } \pi) \\
&= \pi(\alpha'(f(t_1, \ldots, t_p))) \text{ (def. of } \alpha') \\
&= \pi(\phi(t_{\alpha'(f(t_1, \ldots, t_p))})) \text{ (def. of } t_{\alpha'(f(t_1, \ldots, t_p))}) \\
&= \alpha(t_{\alpha'(f(t_1, \ldots, t_p))}) \text{ (def. of } \phi \text{ and } \pi).
\end{aligned}$$

(3) We have $r'(T(\mathcal{F})) = r'(L) = r(L) = r(T(\mathcal{F}))$ using the definition of r', the definition of L, and the equality $r'(L) = r(L)$. The run r is a successful run. Consequently r' is a successful run.

□

Proposition 34. *A non-empty recognizable set of generalized tree sets contains a regular generalized tree set.*

Proof. Let us consider a generalized tree set automaton $\mathcal{A}$ and a successful run r on a generalized tree set g. There exists a tree language closed under the subterm relation F such that $r(F) = r(T(\mathcal{F}))$. We define a regular run rr on a regular generalized tree set gg in the following way.

The run rr coincides with r on F: $\forall t \in F$, $rr(t) = r(t)$ and $gg(t) = g(t)$. The runs rr and gg are inductively defined on $T(\mathcal{F}) \setminus F$: given $q_1, \ldots, q_p$ in $r(T(\mathcal{F}))$, let us fix a rule $f(q_1, \ldots, q_p) \xrightarrow{l} q$ such that $q \in r(T(\mathcal{F}))$. The rule exists since r is a run. Therefore, $\forall t = f(t_1, \ldots, t_p) \notin F$ such that $rr(t_i) = q_i$ for all $i \leq p$, we define $rr(t) = q$ and $gg(t) = l$, following the fixed rule $f(q_1, \ldots, q_p) \xrightarrow{l} q$.

□

From the preceding, we can also deduce that a finite and recognizable set of generalized tree sets only contains regular generalized tree sets.

5.3 Closure and Decision Properties

5.3.1 Closure properties

This section is dedicated to the study of classical closure properties on GTSA-recognizable languages. For all positive results — union, intersection, projection, cylindrification — the proofs are constructive. We show that the class of recognizable sets of generalized tree sets is not closed under complementation and that non-determinism cannot be reduced for generalized tree set automata.

Set operations on sets of GTS have to be distinguished from set operations on sets of terms. In particular, in the case where $E = \{0,1\}^n$, if G_1 and G_2 are sets of GTS in $\mathcal{G}_E$, then $G_1 \cup G_2$ contains all GTS in G_1 and G_2. This is clearly different from the set of all $(L_1^1 \cup L_1^2, \ldots, L_n^1 \cup L_n^2)$ where $(L_1^1, \ldots, L_n^1)$ belongs to G_1 and $(L_1^2, \ldots, L_n^2)$ belongs to G_2.

Proposition 35. *The class $\mathcal{R}_{\mathrm{GTS}}$ is closed under intersection and union, i.e.if G_1, $G_2 \subseteq \mathcal{G}_E$ are recognizable, then $G_1 \cup G_2$ and $G_1 \cap G_2$ are recognizable.*

This proof is an easy modification of the classical proof of closure properties for tree automata, see Chapter 1.

Proof. Let $\mathcal{A}_1 = (Q_1, \Delta_1, \Omega_1)$ and $\mathcal{A}_2 = (Q_2, \Delta_2, \Omega_2)$ be two generalized tree set automata over E. Without loss of generality we assume that $Q_1 \cap Q_2 = \emptyset$.

Let $\mathcal{A} = (Q, \Delta, \Omega)$ with $Q = Q_1 \cup Q_2$, $\Delta = \Delta_1 \cup \Delta_2$, and $\Omega = \Omega_1 \cup \Omega_2$. It is immediate that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

We denote by π_1 and π_2 the projections from $Q_1 \times Q_2$ into respectively Q_1 and Q_2. Let $\mathcal{A}' = (Q', \Delta', \Omega')$ with $Q' = Q_1 \times Q_2$, Δ' is defined by

$$(f(q_1, \ldots, q_p) \xrightarrow{l} q \in \Delta') \Leftrightarrow (\forall i \in \{1,2\}\ f(\pi_i(q_1), \ldots, \pi_i(q_p)) \xrightarrow{l} \pi_i(q) \in \Delta_i) \ ,$$

where $q_1, \ldots, q_p, q \in Q'$, $f \in \mathcal{F}_p$, $l \in E$, and Ω' is defined by

$$\Omega' = \{\omega \in 2^{Q'} \mid \pi_i(\omega) \in \Omega_i \ , \ i \in \{1,2\}\}.$$

One can easily verify that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$. □

Let us remark that the previous constructions also prove that the class $\mathcal{R}_{\mathrm{SGTS}}$ is closed under union and intersection.

The class languages recognizable by deterministic generalized tree set automata is closed under complementation. But, this property is false in the general case of GTSA-recognizable languages.

Proposition 36. *(a) Let $\mathcal{A}$ be a generalized tree set automaton, there exists a complete generalized tree set automaton $\mathcal{A}_c$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_c)$.*

(b) If $\mathcal{A}_{cd}$ is a deterministic and complete generalized tree set automaton, there exists a generalized tree set automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{G}_E - \mathcal{L}(\mathcal{A}_{cd})$.

(c) The class of GTSA-recognizable languages is not closed under complementation.

(d) Non-determinism can not be reduced for generalized tree set automata.

Proof. (a) Let $\mathcal{A} = (Q, \Delta, \Omega)$ be a generalized tree set automaton over E and let q' be a new state, *i.e.*$q' \notin Q$. Let $\mathcal{A}_c = (Q_c, \Delta_c, \Omega_c)$ be defined by $Q_c = Q \cup \{q'\}$, $\Omega_c = \Omega$, and

$$\begin{aligned}\Delta_c = \Delta \cup \{(q_1, \dots, q_p, f, l, q') \mid \quad & \{(q_1, \dots, q_p, f, l)\} \times Q \cap \Delta = \emptyset; \\ & q_1, \dots, q_p \in Q_c, f \in \mathcal{F}_p, l \in E\}.\end{aligned}$$

$\mathcal{A}_c$ is complete and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_c)$. Note that $\mathcal{A}_c$ is simple if $\mathcal{A}$ is simple.

(b) $\mathcal{A}_{cd} = (Q, \Delta, \Omega)$ be a deterministic and complete generalized tree set automaton over E. The automaton $\mathcal{A}' = (Q', \Delta', \Omega')$ with $Q' = Q$, $\Delta' = \Delta$, and $\Omega' = 2^Q - \Omega$ recognizes the set $\mathcal{G}_E - \mathcal{L}(\mathcal{A}_{cd})$.

(c) $E = \{0, 1\}$, $\mathcal{F} = \{c, a\}$ where a is a constant and c is of arity 1. Let $G = \{g \in \mathcal{G}_{\{0,1\}^n} \mid \exists t \in T(\mathcal{F})\ ((g(t) = 1) \wedge (\forall t' \in T(\mathcal{F})\ (t \trianglelefteq t') \Rightarrow (g(t') = 1)))\}$. Clearly, G is recognizable by a non deterministic GTSA (see Example 53). Let $\overline{G} = \mathcal{G}_{\{0,1\}^n} - G$, we have $\overline{G} = \{g \in \mathcal{G}_{\{0,1\}^n} \mid \forall t \in T(\mathcal{F})\ \exists t' \in T(\mathcal{F})\ (t \trianglelefteq t') \wedge (g(t') = 0)\}$ and $\overline{G}$ is not recognizable. Let us suppose that $\overline{G}$ is recognized by an automaton $\mathcal{A} = (Q, \Delta, \Omega)$ with $\mathsf{Card}(Q) = k - 2$ and let us consider the generalized tree set g defined by: $g(c^i(a)) = 0$ if $i = k \times z$ for some integer z, and $g(c^i(a)) = 1$ otherwise. The generalized tree set g is in $\overline{G}$ and we consider a successful run r on g. We have $r(T(\mathcal{F})) = \omega \in \Omega$ therefore there exists some integer n such that $r(\{g(c^i(a)) \mid i \leq n\}) = \omega$. Moreover we can suppose that n is a multiple of k. As $\mathsf{Card}(Q) = k - 2$ there are two terms u and v in the set $\{c^i(a) \mid n{+}1 \leq i \leq n{+}k{-}1\}$ such that $r(u) = r(v)$. Note that by hypothesis, for all i such that $n{+}1 \leq i \leq n{+}k{+}1$, $g(c^i(a)) = 1$. Consequently, a successful run g' could be defined from g on the generalized tree set g' defined by $g'(t) = g(t)$ if $t = c^i(a)$ when $i \leq n$, and $g'(t) = 1$ otherwise. This leads to a contradiction because $g' \notin \overline{G}$.

(d) This result is a consequence of (b) and (c).

□

We will now prove the closure under projection and cylindrification. We will first prove a stronger lemma.

Lemma 8. *Let $G \subseteq \mathcal{G}_{E_1}$ be a GTSA-recognizable language and let $R \subseteq E_1 \times E_2$. The set $R(G) = \{g' \in \mathcal{G}_{E_2} \mid \exists g \in G\ \forall t \in T(\mathcal{F})\ (g(t), g'(t)) \in R\}$ is recognizable.*

Proof. Let $\mathcal{A} = (Q, \Delta, \Omega)$ such that $\mathcal{L}(\mathcal{A}) = G$. Let $\mathcal{A}' = (Q', \Delta', \Omega')$ where $Q' = Q$, $\Delta' = \{f(q_1, \dots, q_p) \xrightarrow{l'} q \mid \exists l \in E_1\ f(q_1, \dots, q_p) \xrightarrow{l} q \in \Delta \text{ and } (l, l') \in R\}$ and $\Omega' = \Omega$. We prove that $R(G) = \mathcal{L}(\mathcal{A}')$.

$\supseteq$ Let $g' \in \mathcal{L}(\mathcal{A}')$ and let r' be a successful run on g'. We construct a generalized tree set g such that for all $t \in T(\mathcal{F})$, $(g(t), g'(t)) \in R$ and such that r' is also a successful $\mathcal{A}$-run on g.

Let a be a constant. According to the definition of Δ', $a \xrightarrow{g'(a)} r'(a) \in \Delta'$ implies that there exists l_a such that $(l_a, g'(a)) \in R$ and $a \xrightarrow{l_a} r'(a) \in \Delta$. So let $g(a) = l_a$.

Let $t = f(t_1, \dots, t_p)$ with $\forall i\, r'(t_i) = q_i$. There exists a rule $f(q_1, \dots, q_p) \xrightarrow{g'(t)} r'(t)$ in Δ' because r' is a run on g' and again, from the definition of Δ', there exists $l_t \in E_1$ such that $f(q_1, \dots, q_p) \xrightarrow{l_t} r'(t)$ in Δ with $(l_t(t), g'(t)) \in R$. So, we define $g(t) = l_t$. Clearly, g is a generalized tree set and r' is a successful run on g and for all $t \in T(\mathcal{F})$, $(g(t), g'(t)) \in R$ by construction.

$\subseteq$ Let $g' \in R(G)$ and let $g \in G$ such that $\forall t \in T(\mathcal{F})$ $(g(t), g'(t)) \in R$. One can easily prove that any successful $\mathcal{A}$-run on g is also a successful $\mathcal{A}'$-run on g'.

□

Let us recall that if g is a generalized tree set in $\mathcal{G}_{E_1 \times \cdots \times E_n}$, the ith projection of g (on the E_i-component, $1 \le i \le n$) is the GTS $\pi_i(g)$ defined by: let π from $E_1 \times \cdots \times E_n$ into E_i, such that $\pi(l_1, \dots, l_n) = l_i$ and let $\pi_i(g)(t) = \pi(g(t))$ for every term t. Conversely, the ith cylindrification of a GTS g denoted by $\pi_i^{-1}(g)$ is the set of GTS g' such that $\pi_i(g') = g$. Projection and cylindrification are usually extended to sets of GTS.

Corollary 7. *(a) The class of GTSA-recognizable languages is closed under projection and cylindrification.*

(b) Let $G \subseteq \mathcal{G}_E$ and $G' \subseteq \mathcal{G}_{E'}$ be two GTSA-recognizable languages. The set $G \uparrow G' = \{g \uparrow g' \mid g \in G,\ g' \in G'\}$ is a GTSA-recognizable language in $\mathcal{G}_{E \times E'}$.

Proof. (a) The case of projection is an immediate consequence of Lemma 8 using $E_1 = E \times E'$, $E_2 = E$, and $R = \pi$ where π is the projection from $E \times E'$ into E. The case of cylindrification is proved in a similar way.

(b) Consequence of (a) and of Proposition 35 because $G \uparrow G' = \pi_1^{-1}(G) \cap \pi_2^{-1}(G')$ where π_1^{-1} (respectively π_2^{-1}) is the inverse projection from E to $E \times E'$ (respectively from E' to $E \times E'$).

Let us remark that the construction preserves simplicity, so $\mathcal{R}_{\mathrm{SGTS}}$ is closed under projection and cylindrification.

□

We now consider the case $E = \{0,1\}^n$ and we give two propositions without proof. Proposition 37 can easily be deduced from Corollary 7. The proof of Proposition 38 is an extension of the constructions made in Examples 49.1 and 49.2.

Proposition 37. *Let $\mathcal{A}$ and $\mathcal{A}'$ be two generalized tree set automata over $\{0,1\}^n$.*

(a) $\{(L_1 \cup L'_1, \dots, L_n \cup L'_n) \mid (L_1, \dots, L_n) \in \mathcal{L}(\mathcal{A})$ and $(L'_1, \dots, L'_n) \in \mathcal{L}(\mathcal{A}')\}$ is recognizable.

(b) $\{(L_1 \cap L'_1, \dots, L_n \cap L'_n) \mid (L_1, \dots, L_n) \in \mathcal{L}(\mathcal{A})$ and $(L'_1, \dots, L'_n) \in \mathcal{L}(\mathcal{A}')\}$ is recognizable.

(c) $\{(\overline{L_1}, \ldots, \overline{L_n}) \mid (L_1, \ldots, L_n) \in \mathcal{L}(\mathcal{A})\}$ *is recognizable, where* $\overline{L_i} = T(\mathcal{F}) - L_i$, $\forall i$.

Proposition 38. *Let* $E = \{0,1\}^n$ *and let* $(F_1, \ldots, F_n)$ *be a* n*-tuple of regular tree languages. There exist deterministic simple generalized tree set automata* $\mathcal{A}$, $\mathcal{A}'$, *and* $\mathcal{A}''$ *such that*

- $\mathcal{L}(\mathcal{A}) = \{(F_1, \ldots, F_n)\}$;
- $\mathcal{L}(\mathcal{A}') = \{(L_1, \ldots, L_n) \mid L_1 \subseteq F_1, \ldots, L_n \subseteq F_n\}$;
- $\mathcal{L}(\mathcal{A}'') = \{(L_1, \ldots, L_n) \mid F_1 \subseteq L_1, \ldots, F_n \subseteq L_n\}$.

5.3.2 Emptiness Property

Theorem 44. *The emptiness property is decidable in the class of generalized tree set automata. Given a generalized tree set automaton* $\mathcal{A}$, *it is decidable whether* $\mathcal{L}(\mathcal{A}) = \emptyset$.

Labels of the generalized tree sets are meaningless for the emptiness decision thus we consider "label-free" generalized tree set automata. Briefly, the transition relation of a "label-free" generalized tree set automata is a relation $\Delta \subseteq \cup_p Q^p \times \mathcal{F}_p \times Q$.

The emptiness decision algorithm for simple generalized tree set automata is straightforward. Indeed, Let ω be a subset of Q and let $\mathsf{COND}(\omega)$ be the following condition:

$$\forall p \; \forall f \in \mathcal{F}_p \; \forall q_1, \ldots, q_p \in \omega \; \exists q \in \omega \quad (q_1, \ldots, q_p, f, q) \in \Delta$$

We easily prove that there exists a set ω satisfying $\mathsf{COND}(\omega)$ if and only if there exists an $\mathcal{A}$-run. Therefore, the emptiness problem for simple generalized tree set automata is decidable because 2^Q is finite and $\mathsf{COND}(\omega)$ is decidable. Decidability of the emptiness problem for simple generalized tree set automata is NP-complete (see Prop. 39).

The proof is more intricate in the general case, and it is not given in this book. Without the property of simple GTSA, we have to deal with a reachability problem of a set of states since we have to check that there exists $\omega \in \Omega$ and a run r such that r assumes exactly all the states in ω.

We conclude this section with a complexity result of the emptiness problem in the class of generalized tree set automata.

Let us remark that a finite initial fragment of a "label-free" generalized tree set corresponds to a finite set of terms that is closed under the subterm relation. The size or the number of nodes in such an initial fragment is the number of different terms in the subterm-closed set of terms (the cardinality of the set of terms). The size of a GTSA is given by:

$$\|\mathcal{A}\| = |Q| + \sum_{f(q_1,\ldots,q_p) \xrightarrow{l} q \in \Delta} (arity(f) + 3) + \sum_{\omega \in \Omega} |\omega|.$$

Let us consider a GTSA $\mathcal{A}$ with n states. The proof shows that one must consider at most all initial fragments of runs —hence corresponding to finite tree languages closed under the subterm relation— of size smaller than $B(\mathcal{A})$, a polynomial in n, in order to decide emptiness for $\mathcal{A}$. Let us remark that the polynomial bound $B(\mathcal{A})$ can be computed. The emptiness proofs relies on the following lemma:

Lemma 9. *There exists a polynomial function f of degree 4 such that:*

> *Let $\mathcal{A} = (Q, \Delta, \Omega)$ be a GTSA. There exists a successful run r_s such that $r_s(T(\mathcal{F})) = \omega \in \Omega$ if and only if there exists a run r_m and a closed tree language F such that:*
>
> - $r_m(T(\mathcal{F})) = r_m(F) = \omega$*;*
> - $\mathsf{Card}(F) \leq f(n)$ *where n is the number of states in ω.*

Proposition 39. *The emptiness problem in the class of (simple) generalized tree set automata is NP-complete.*

Proof. Let $\mathcal{A} = (Q, \Delta, \Omega)$ be a generalized tree set automaton over E. Let $n = \mathsf{Card}(Q)$.

We first give a non-deterministic and polynomial algorithm for deciding emptiness: (1) take a tree language F closed under the subterm relation such that the number of different terms in it is smaller than $B(\mathcal{A})$; (2) take a run r on F; (3) compute $r(F)$; (4) check whether $r(F) = \omega$ is a member of Ω; (5) check whether ω satisfies $\mathsf{COND}(\omega)$.

From Theorem 44, this algorithm is correct and complete. Moreover, this algorithm is polynomial in n since (1) the size of F is polynomial in n: step (2) consists in labeling the nodes of F with states following the rules of the automaton – so there is a polynomial number of states, step (3) consists in collecting the states; step (4) is polynomial and non-deterministic and finally, step (5) is polynomial.

We reduce the satisfiability problem of boolean expressions into the emptiness problem for generalized tree set automata. We first build a generalized tree set automaton $\mathcal{A}$ such that $L(\mathcal{A})$ is the set of (codes of) satisfiable boolean expressions over n variables $\{x_1, \ldots, x_n\}$.

Let $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2$ where $\mathcal{F}_0 = \{x_1, \ldots, x_n\}$, $\mathcal{F}_1 = \{\neg\}$, and $\mathcal{F}_2 = \{\wedge, \vee\}$. A boolean expression is a term of $T(\mathcal{F})$. Let $\mathsf{Bool} = \{0, 1\}$ be the set of boolean values. Let $\mathcal{A} = (Q, \Delta, \Omega)$, be a generalized tree set automaton such that $Q = \{q_0, q_1\}$, $\Omega = 2^Q$ and Δ is the following set of rules:

$$\begin{aligned}
x_j &\xrightarrow{i} q_i \text{ where } j \in \{1, \ldots, n\} \text{ and } i \in \mathsf{Bool} \\
\neg(q_i) &\xrightarrow{\neg i} q_{\neg i} \text{ where } i \in \mathsf{Bool} \\
\vee(q_{i_1}, q_{i_2}) &\xrightarrow{i_1 \vee i_2} q_{i_1 \vee i_2} \text{ where } i_1, i_2 \in \mathsf{Bool} \\
\wedge(q_{i_1}, q_{i_2}) &\xrightarrow{i_1 \wedge i_2} q_{i_1 \wedge i_2} \text{ where } i_1, i_2 \in \mathsf{Bool}
\end{aligned}$$

One can easily prove that $L(\mathcal{A}) = \{L_v \mid v \text{ \textit{is a valuation of} } \{x_1, \ldots, x_n\}\}$ where $L_v = \{t \mid t \text{ \textit{is a boolean expression which is true under} } v\}$. L_v corresponds to a run r_v on a GTS g_v and g_v labels each x_j either by 0 or 1. Hence, g_v can be considered as a valuation v of $x_1, \ldots, x_n$. This valuation is extended in g_v to every node, that is to say that every term (representing a boolean expression) is labeled either by 0 or 1 accordingly to the usual interpretation of $\neg$, $\wedge$, $\vee$. A given boolean expression is hence labeled by 1 if and only if it is true under the valuation v.

Now, we can derive an algorithm for the satisfiability of any boolean expression e: build $\mathcal{A}_e$ a generalized tree set automaton such that $\mathcal{L}(\mathcal{A})$ is the set of all tree languages containing e: $\{L \mid e \in L\}$; build $\mathcal{A}_e \cap \mathcal{A}$ and decide emptiness.

We get then the reduction because $\mathcal{A}_e \cap \mathcal{A}$ is empty if and only if e is not satisfiable.

Now, it remains to prove that the reduction is polynomial. The size of $\mathcal{A}$ is $2 * n + 10$. The size of $\mathcal{A}_e$ is the length of e plus a constant. So we get the result. □

5.3.3 Other Decision Results

Proposition 40. *The inclusion problem and the equivalence problem for deterministic generalized tree set automata are decidable.*

Proof. These results are a consequence of the closure properties under intersection and complementation (Propositions 35, 36), and the decidability of the emptiness property (Theorem 44). □

Proposition 41. *Let $\mathcal{A}$ be a generalized tree set automaton. It is decidable whether or not $\mathcal{L}(\mathcal{A})$ is a singleton set.*

Proof. Let $\mathcal{A}$ be a generalized tree set automaton. First it is decidable whether $\mathcal{L}(\mathcal{A})$ is empty or not (Theorem 44). Second if $\mathcal{L}(\mathcal{A})$ is non empty then a regular generalized tree set g in $\mathcal{L}(\mathcal{A})$ can be constructed (see the proof of Theorem 44). Construct the strongly deterministic generalized tree set automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}')$ is a singleton set reduced to the generalized tree set g. Finally, build $\mathcal{A} \cap \overline{\mathcal{A}}'$ to decide the equivalence of $\mathcal{A}$ and $\mathcal{A}'$. Note that we can build $\overline{\mathcal{A}}'$, since $\mathcal{A}'$ is deterministic (see Proposition 36). □

Proposition 42. *Let $L = (L_1, \dots, L_n)$ be a tuple of regular tree language and let $\mathcal{A}$ be a generalized tree set automaton over $\{0,1\}^n$. It is decidable whether $L \in \mathcal{L}(\mathcal{A})$.*

Proof. This result just follows from closure under intersection and emptiness decidability.

First construct a (strongly deterministic) generalized tree set automaton $\mathcal{A}_L$ such that $L(\mathcal{A})$ is reduced to the singleton set $\{L\}$. Second, construct $\mathcal{A} \cap \mathcal{A}_L$ and decide whether $L(\mathcal{A} \cap \mathcal{A}_L)$ is empty or not. □

Proposition 43. *Given a generalized tree set automaton over $E = \{0, 1, \}^n$ and $I \subseteq \{1, \dots, n\}$. The following two problems are decidable:*

1. *It is decidable whether or not there exists $(L_1, \dots, L_n)$ in $\mathcal{L}(\mathcal{A})$ such that all the L_i are finite for $i \in I$.*
2. *Let $x_1 \dots, x_n$ be natural numbers. It is decidable whether or not there exists $(L_1, \dots, L_n)$ in $\mathcal{L}(\mathcal{A})$ such that $\mathsf{Card}(L_i) = x_i$ for each $i \in I$.*

The proof is technical and not given in this book. It relies on Lemma 9 of the emptiness decision proof.

5.4 Applications to Set Constraints

In this section, we consider the satisfiability problem for systems of set constraints. We show a decision algorithm using generalized tree set automata.

5.4.1 Definitions

Let $\mathcal{F}$ be a finite and non-empty set of function symbols. Let $\mathcal{X}$ be a set of variables. We consider special symbols $\top$, $\bot$, $\sim$, $\cup$, $\cap$ of respective arities 0, 0, 1, 2, 2. A *set expression* is a term in $T_{\mathcal{F}'}(\mathcal{X})$ where $\mathcal{F}' = \mathcal{F} \cup \{\top, \bot, \sim, \cup, \cap\}$.

A set constraint is either a *positive* set constraint of the form $e \subseteq e'$ or a *negative* set constraint of the form $e \not\subseteq e'$ (or $\neg(e \subseteq e')$) where e and e' are set expressions, and a system of set constraints is defined by $\bigwedge_{i=1}^{k} SC_i$ where the SC_i are set constraints.

An interpretation $\mathcal{I}$ is a mapping from $\mathcal{X}$ into $2^{T(\mathcal{F})}$. It can immediately be extended to set expressions in the following way:

$$\begin{aligned}
\mathcal{I}(\top) &= T(\mathcal{F});\\
\mathcal{I}(\bot) &= \emptyset;\\
\mathcal{I}(f(e_1, \ldots, e_p)) &= f(\mathcal{I}(e_1), \ldots, \mathcal{I}(e_p));\\
\mathcal{I}(\sim e) &= T(\mathcal{F}) \setminus \mathcal{I}(e);\\
\mathcal{I}(e \cup e') &= \mathcal{I}(e) \cup \mathcal{I}(e');\\
\mathcal{I}(e \cap e') &= \mathcal{I}(e) \cap \mathcal{I}(e').
\end{aligned}$$

We deduce an interpretation of set constraints in $\mathsf{Bool} = \{0, 1\}$, the Boolean values. For a system of set constraints SC, all the interpretations $\mathcal{I}$ such that $\mathcal{I}(SC) = 1$ are called *solutions* of SC. In the remainder, we will consider systems of set constraints of n variables $X_1, \ldots, X_n$. We will make no distinction between a *solution* $\mathcal{I}$ of a system of set constraints and a *n-tuple of tree languages* $(\mathcal{I}(X_1), \ldots, \mathcal{I}(X_n))$. We denote by $\mathsf{SOL}(SC)$ the set of all solutions of a system of set constraints SC.

5.4.2 Set Constraints and Automata

Proposition 44. *Let SC be a system of set constraints (respectively of positive set constraints) of n variables $X_1, \ldots, X_n$. There exists a deterministic (respectively deterministic and simple) generalized tree set automaton $\mathcal{A}$ over $\{0,1\}^n$ such that $\mathcal{L}(\mathcal{A})$ is the set of characteristic generalized tree sets of the n-tuples $(L_1, \ldots, L_n)$ of solutions of SC.*

Proof. First we reduce the problem to a single set constraint. Let $SC = C_1 \wedge \ldots \wedge C_k$ be a system of set constraints. A solution of SC satisfies all the constraints C_i. Let us suppose that, for every i, there exists a deterministic generalized tree set automaton $\mathcal{A}_i$ such that $\mathsf{SOL}(C_i) = \mathcal{L}(\mathcal{A})$. As all variables in $\{X_1, \ldots, X_n\}$ do not necessarily occur in C_i, using Corollary 7, we can construct a deterministic generalized tree set automaton $\mathcal{A}_i^n$ over $\{0,1\}^n$ satisfying: $\mathcal{L}(\mathcal{A}_i^n)$ is the set of $(L_1, \ldots, L_n)$ which corresponds to solutions of C_i when restricted to the variables of C_i. Using closure under intersection (Proposition 35), we can construct a deterministic generalized tree set automaton $\mathcal{A}$ over $\{0,1\}^n$ such that $\mathsf{SOL}(SC) = \mathcal{L}(\mathcal{A})$.

Therefore we prove the result for a set constraint SC of n variables $X_1, \ldots, X_n$. Let $\mathcal{E}(exp)$ be the set of set variables and of set expression exp with a root symbol in $\mathcal{F}$ which occur in the set expression exp:

$$\mathcal{E}(exp) = \big\{exp' \in T_{\mathcal{F}'}(\mathcal{X}) \mid exp' \trianglelefteq exp \text{ and such that} \\ \text{either } \mathcal{H}ead(exp') \in \mathcal{F} \text{ or } exp' \in \mathcal{X}\big\}.$$

If $SC \equiv exp_1 \subseteq exp_2$ or $SC \equiv exp_1 \not\subseteq exp_2$ then $\mathcal{E}(SC) = \mathcal{E}(exp_1) \cup \mathcal{E}(exp_2)$.

Let us consider a set constraint SC and let φ be a mapping φ from $\mathcal{E}(SC)$ into Bool. Such a mapping is easily extended first to any set expression occurring in SC and second to the set constraint SC. The symbols $\cup$, $\cap$, $\sim$, $\subseteq$ and $\not\subseteq$ are respectively interpreted as $\vee$, $\wedge$, $\neg$, $\Rightarrow$ and $\neg\Rightarrow$.

We now define the generalized tree set automaton $\mathcal{A} = (Q, \Delta, \Omega)$ over $E = \{0,1\}^n$.

- The set of states is Q is the set $\{\varphi \mid \varphi : \mathcal{E}(SC) \to \mathsf{Bool}\}$.
- The transition relation is defined as follows: $f(\varphi_1, \ldots, \varphi_p) \xrightarrow{l} \varphi \in \Delta$ where $\varphi_1, \ldots, \varphi_p \in Q$, $f \in \mathcal{F}_p$, $l = (l_1, \ldots, l_n) \in \{0,1\}^n$, and $\varphi \in Q$ satisfies:

$$\forall i \in \{1, \ldots, n\} \; \varphi(X_i) = l_i \tag{5.6}$$

$$\forall e \in \mathcal{E}(SC) \setminus \mathcal{X} \;\; (\varphi(e) = 1) \Leftrightarrow \left(\begin{array}{l} e = f(e_1, \ldots, e_p) \\ \forall i \quad 1 \le i \le p \quad \varphi_i(e_i) = 1 \end{array} \right) \tag{5.7}$$

- The set of accepting sets of states Ω is defined depending on the case of a positive or a negative set constraint.
  - If SC is positive, $\Omega = \{\omega \in 2^Q \mid \forall \varphi \in \omega \; \varphi(SC) = 1\}$;
  - If SC is negative, $\Omega = \{\omega \in 2^Q \mid \exists \varphi \in \omega \; \varphi(SC) = 1\}$.

In the case of a positive set constraint, we can choose the state set $Q = \{\varphi \mid \varphi(SC) = 1\}$ and $\Omega = 2^Q$. Consequently, $\mathcal{A}$ is deterministic and simple.

The correctness of this construction is easy to prove and is left to the reader. $\square$

5.4.3 Decidability Results for Set Constraints

We now summarize results on set constraints. These results are immediate consequences of the results of Section 5.4.2. We use Proposition 44 to encode sets of solutions of systems of set constraints with generalized tree set automata and then, each point is deduced from Theorem 44, or Propositions 38, 43, 40, 41.

Properties on sets of solutions

Satisfiability The satisfiability problem for systems of set constraints is decidable.

Regular solution There exists a regular solution, that is a tuple of regular tree languages, in any non-empty set of solutions.

Inclusion, Equivalence Given two systems of set constraints SC and SC', it is decidable whether or not $\mathsf{SOL}(SC) \subseteq \mathsf{SOL}(SC')$.

Unicity Given a system SC of set constraints, it is decidable whether or not there is a unique solution in $\mathsf{SOL}(SC)$.

Properties on solutions

fixed cardinalities, singletons Given a system SC of set constraints over $(X_1, \ldots, X_n)$, $I \subseteq \{1, \ldots, n\}$, and $x_1 \ldots, x_n$ natural numbers;

- it is decidable whether or not there is a solution $(L_1, \ldots, L_n) \in \mathsf{SOL}(SC)$ such that $\mathsf{Card}(L_i) = x_i$ for each $i \in I$.
- it is decidable whether or not all the L_i are finite for $i \in I$.

In both cases, proofs are constructive and exhibits a solution.

Membership Given SC a system of set constraints over $(X_1, \ldots, X_n)$ and a n-tuple $(L_1, \ldots, L_n)$ of regular tree languages, it is decidable whether or not $(L_1, \ldots, L_n) \in \mathsf{SOL}(SC)$.

Proposition 45. *Let SC be a system of positive set constraints, it is decidable whether or not there is a least solution in $\mathsf{SOL}(SC)$.*

Proof. Let SC be a system of positive set constraints. Let $\mathcal{A}$ be the deterministic, simple generalized tree set automaton over $\{0,1\}^n$ such that $\mathcal{L}(\mathcal{A}) = \mathsf{SOL}(SC)$ (see Proposition 44). We define a partial ordering $\preceq$ on $\mathcal{G}_{\{0,1\}^n}$ by:

$$\begin{array}{ll} \forall l, l' \in \{0,1\}^n & l \preceq l' \Leftrightarrow (\forall i \; l(i) \leq l'(i)) \\ \forall g, g' \in \mathcal{G}_{\{0,1\}^n} & g \preceq g' \Leftrightarrow (\forall t \in T(\mathcal{F}) \; g(t) \preceq g'(t)) \end{array}$$

The problem we want to deal with is to decide whether or not there exists a least generalized tree set *w.r.t.*$\preceq$ in $\mathcal{L}(\mathcal{A})$. To this aim, we first build a minimal solution if it exists, and second, we verify that this solution is unique.

Let ω be a subset of states such that $\mathsf{COND}(\omega)$ (see the sketch of proof page 160). Let $\mathcal{A}_\omega = (\omega, \Delta_\omega, 2^\omega)$ be the generalized tree set automaton $\mathcal{A}$ restricted to state set ω.

Now let ${\Delta_\omega}^{min}$ defined by: for each $(q_1, \ldots, q_p, f) \in \omega^p \times \mathcal{F}_p$, choose in the set Δ_ω one rule $(q_1, \ldots, q_p, f, l, q)$ such that l is minimal *w.r.t.*$\preceq$. Let ${\mathcal{A}_\omega}^{min} = (\omega, {\Delta_\omega}^{min}, 2^\omega)$. Consequently,

1. There exists only one run r_ω on a unique generalized tree set g_ω in ${\mathcal{A}_\omega}^{min}$ because for all $q_1, \ldots, q_p \in \omega$ and $f \in \mathcal{F}_p$ there is only one rule $(q_1, \ldots, q_p, f, l, q)$ in ${\Delta_\omega}^{min}$;
2. the run r_ω on g_ω is regular;

3. the generalized tree set g_ω is minimal *w.r.t.*$\preceq$ in $\mathcal{L}(\mathcal{A}_\omega)$.

Points 1 and 2 are straightforward. The third point follows from the fact that $\mathcal{A}$ is deterministic. Indeed, let us suppose that there exists a run r' on a generalized tree set g' such that $g' \prec g_\omega$. Therefore, $\forall t\ g'(t) \preceq g_\omega(t)$, and there exists (w.l.o.g.) a minimal term $u = f(u_1, \dots, u_p)$ *w.r.t.*the subterm ordering such that $g'(u) \prec g_\omega(u)$. Since $\mathcal{A}$ is deterministic and $\forall v \lhd u\ g_\omega(v) = g'(v)$, we have $r_\omega(u_i) = r'(u_i)$. Hence, the rule $(r_\omega(u_1), \dots, r_\omega(u_p), f, g_\omega(u), r_\omega(u))$ is not such that $g_\omega(u)$ is minimal in Δ_ω, which contradicts the hypothesis.

Consider the generalized tree sets g_ω for all subsets of states ω satisfying $\mathsf{COND}(\omega)$. If there is no such g_ω, then there is no least generalized tree set g in $\mathcal{L}(\mathcal{A})$. Otherwise, each generalized tree set defines a n-tuple of regular tree languages and inclusion is decidable for regular tree languages. Hence we can identify a minimal generalized tree set g among all g_ω. This GTS g defines a n-tuple $(F_1, \dots, F_n)$ of regular tree languages. Let us remark this construction does not ensure that $(F_1, \dots, F_n)$ is minimal in $\mathcal{L}(\mathcal{A})$.

There is a deterministic, simple generalized tree set automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}')$ is the set of characteristic generalized tree sets of all $(L_1, \dots, L_n)$ satisfying $F_1 \subseteq L_1, \dots, F_n \subseteq L_n$ (see Proposition 38). Let $\mathcal{A}''$ be the deterministic generalized tree set automaton such that $\mathcal{L}(\mathcal{A}'') = \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}')$ (see Proposition 35). There exists a least generalized tree set *w.r.t.*$\preceq$ in $\mathcal{L}(\mathcal{A})$ if and only if the generalized tree set automata $\mathcal{A}$ and $\mathcal{A}''$ are equivalent. Since equivalence of generalized tree set automata is decidable (see Proposition 40) we get the result. □

5.5 Bibliographical Notes

We now survey decidability results for satisfiability of set constraints and some complexity issues.

Decision procedures for solving set constraints arise with [Rey69], and Mishra [Mis84]. The aim of these works was to obtain new tools for type inference and type checking [AM91, Hei92, HJ90b, JM79, Mis84, Rey69].

First consider systems of set constraints of the form:

$$X_1 = exp_1, \dots, X_n = exp_n \tag{5.8}$$

where the X_i are distinct variables and the exp_i are disjunctions of set expressions of the form $f(X_{i_1}, \dots, X_{i_p})$ with $f \in \mathcal{F}_p$. These systems of set constraints are essentially tree automata, therefore they have a unique solution and each X_i is interpreted as a regular tree language.

Suppose now that the exp_i are set expressions without complement symbols. Such systems are always satisfiable and have a least solution which is regular. For example, the system

$$\begin{aligned}&\mathsf{Nat} = s(\mathsf{Nat}) \cup 0\\ &X = X \cap \mathsf{Nat}\\ &\mathsf{List} = \mathsf{cons}(X, \mathsf{List}) \cup \mathsf{nil}\end{aligned}$$

has a least solution

$$\mathsf{Nat} = \{s^i(0) \mid i \geq 0\}, X = \emptyset, \mathsf{List} = \{\mathsf{nil}\}.$$

[HJ90a] investigate the class of definite set constraints which are of the form $exp \subseteq exp'$, where no complement symbol occurs and exp' contains no set operation. Definite set constraints have a least solution whenever they have a solution. The algorithm presented in [HJ90a] provides a specific set of transformation rules and, when there exists a solution, the result is a regular presentation of the least solution, in other words a system of the form (5.8).

Solving definite set constraints is EXPTIME-complete [CP97]. Many developments or improvements of Heinzte and Jaffar's method have been proposed and some are based on tree automata [DTT97].

The class of positive set constraints is the class of systems of set constraints of the form $exp \subseteq exp'$, where no projection symbol occur. In this case, when a solution exists, set constraints do not necessarily have a least solution. Several algorithms for solving systems in this class were proposed, [AW92] generalize the method of [HJ90a], [GTT93, GTT99] give an automata-based algorithm, and [BGW93] use the decision procedure for the first order theory of monadic predicates. Results on the computational complexity of solving systems of set constraints are presented in a paper of [AKVW93]. The systems form a natural complexity hierarchy depending on the number of elements of $\mathcal{F}$ of each arity. The problem of existence of a solution of a system of positive set constraints is NEXPTIME-complete.

The class of positive and negative set constraints is the class of systems of set constraints of the form $exp \subseteq exp'$ or $exp \not\subseteq exp'$, where no projection symbol occur. In this case, when a solution exists, set constraints do not necessarily have, neither a minimal solution, nor a maximal solution. Let $\mathcal{F} = \{a, b()\}$. Consider the system $(b(X) \subseteq X) \wedge (X \not\subseteq \perp)$, this system has no minimal solution. Consider the system $(X \subseteq b(X) \cup a) \wedge (\top \not\subseteq X)$, this system has no maximal solution. The satisfiability problem in this class turned out to be much more difficult than the positive case. [AKW95] give a proof based on a reachability problem involving Diophantine inequalities. NEXPTIME-completeness was proved by [Ste94]. [CP94a] gives a proof based on the ideas of [BGW93].

The class of positive set constraints with projections is the class of systems of set constraints of the form $exp \subseteq exp'$ with projection symbols. Set constraints of the form $f_i^{-1}(X) \subseteq Y$ can easily be solved, but the case of set constraints of the form $X \subseteq f_i^{-1}(Y)$ is more intricate. The problem was proved decidable by [CP94b].

The expressive power of these classes of set constraints have been studied and have been proved to be different [Sey94]. In [CK96, Koz93], an axiomatization is proposed which enlightens the reader on relationships between many approaches on set constraints.

Furthermore, set constraints have been studied in a logical and topological point of view [Koz95, MGKW96]. This last paper combine set constraints with Tarskian set constraints, a more general framework for which many complexity results are proved or recalled. Tarskian set constraints involve variables, relation and function symbols interpreted relative to a first order structure.

Topological characterizations of classes of GTSA recognizable sets, have also been studied in [Tom94, Sey94]. Every set in $\mathcal{R}_{\text{SGTS}}$ is a compact set and every set in $\mathcal{R}_{\mathrm{GTS}}$ is the intersection between a compact set and an open set. These remarks give also characterizations for the different classes of set constraints.

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, 12th 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, 12th 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*$^{\text{th}}$ *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, *$9^t h$ 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 34th 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 17th 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, 12th 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, *$10^t h$ 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, 11^{th} 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 1^{st} 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