Tree Automata Techniques and Applications

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

Contents

Acknowledgments

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

Introduction

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

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

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

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

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

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

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

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

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

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

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

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

Preliminaries

Terms

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

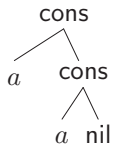
A ***ranked alphabet*** is a couple $(\mathcal{F}, Arity)$ where $\mathcal{F}$ is a finite set and $Arity$ is a mapping from $\mathcal{F}$ into N. The ***arity*** of a symbol $f \in \mathcal{F}$ is $Arity(f)$. The set of symbols of arity p is denoted by $\mathcal{F}_p$. Elements of arity $0, 1, \ldots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

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

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

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

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



Terms and Trees

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

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

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

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

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

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

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

SubTerms

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

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

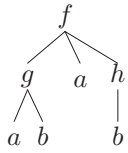
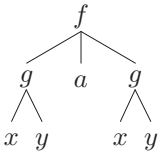
We denote by $\trianglerighteq$ the ***subterm ordering***, *i.e.*we write $t \trianglerighteq t'$ if t' is a subterm of t. We denote $t \triangleright t'$ if $t \trianglerighteq t'$ and $t \neq t'$.

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

Functions on Terms

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

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

$t =$  ; $t' =$ 

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

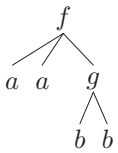
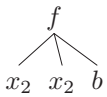
Substitutions

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

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

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

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

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

Contexts

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

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

Chapter 3

Logic, Automata and Relations

3.1 Introduction

As early as in the 50s, automata, and in particular tree automata, played an important role in the development of verification . Several well-known logicians, such as A. Church, J.R. Büchi, Elgott, MacNaughton, M. Rabin and others contributed to what is called "the trinity" by Trakhtenbrot: Logic, Automata and Verification (of Boolean circuits).

The idea is simple: given a formula ϕ with free variables $x_1, ..., x_n$ and a domain of interpretation D, ϕ defines the subset of D^n containing all assignments of the free variables $x_1, \ldots, x_n$ that satisfy ϕ. Hence formulas in this case are just a way of defining subsets of D^n (also called n-ary relations on D). In case $n = 1$ (and, as we will see, also for $n > 1$), finite automata provide another way of defining subsets of D^n. In 1960, Büchi realized that these two ways of defining relations over the free monoid $\{0, \ldots, n\}^*$ coincide when the logic is the *sequential calculus*, also called *weak second-order monadic logic with one successor*, WS1S. This result was extended to tree automata: Doner, Thatcher and Wright showed that the definability in the weak second-order monadic logic with k successors, WSkS coincide with the recognizability by a finite tree automaton. These results imply in particular the decidability of WSkS, following the decision results on tree automata (see chapter 1).

These ideas are the basis of several decision techniques for various logics some of which will be listed in Section 3.4. In order to illustrate this correspondence, consider Presburger's arithmetic: the atomic formulas are equalities and inequalities $s = t$ or $s \geq t$ where s, t are sums of variables and constants. For instance $x{+}y{+}y = z{+}z{+}z{+}1{+}1$, also written $x{+}2y = 3z{+}2$, is an atomic formula. In other words, atomic formulas are linear Diophantine (in)equations. Then atomic formulas can be combined using any logical connectives among $\wedge, \vee, \neg$ and quantifications $\forall, \exists$. For instance $\forall x.(\forall y.\neg(x = 2y)) \Rightarrow (\exists y.x = 2y{+}1))$ is a (true) formula of Presburger's arithmetic. Formulas are interpreted in the natural numbers (non-negative integers), each symbol having its expected meaning. A *solution* of a formula $\phi(x)$ whose only free variable is x, is an assignment of x to a natural number n such that $\phi(n)$ holds true in the interpretation. For

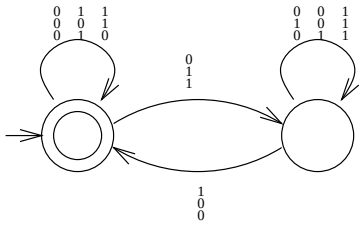


Figure 3.1: The automaton with accepts the solutions of $x = y + z$

instance, if $\phi(x)$ is the formula $\exists y.x = 2y$, its solutions are the even numbers.

Writing integers in base 2, they can be viewed as elements of the free monoid $\{0,1\}^*$, *i.e.*words of 0s and 1s. The representation of a natural number is not unique as $01 = 1$, for instance. Tuples of natural numbers are displayed by stacking their representations in base 2 and aligning on the right, then completing with some 0s on the left in order to get a rectangle of bits. For instance the pair (13,6) is represented as $\begin{smallmatrix}1&1&0&1\\0&1&1&0\end{smallmatrix}$ (or $\begin{smallmatrix}0&1&1&0&1\\0&0&1&1&0\end{smallmatrix}$ as well). Hence, we can see the solutions of a formula as a subset of $(\{0,1\}^n)^*$ where n is the number of free variables of the formula.

It is not difficult to see that the set of solutions of any atomic formula is recognized by a finite word automaton working on the alphabet $\{0,1\}^n$. For instance, the solutions of $x = y + z$ are recognized by the automaton of Figure 3.1.

Then, and that is probably one of the key ideas, each logical connective corresponds to a basic operation on automata (here word automata): $\vee$ is a union, $\wedge$ and intersection, $\neg$ a complement, $\exists x$ a *projection* (an operation which will be defined in Section 3.2.4). It follows that the set of solutions of any Presburger formula is recognized by a finite automaton.

In particular, a closed formula (without free variable), holds true in the interpretation if the initial state of the automaton is also final. It holds false otherwise. Therefore, this gives both a decision technique for Presburger formulas by computing automata and an effective representation of the set of solutions for open formulas.

The example of Presburger's arithmetic we just sketched is not isolated. That is one of the purposes of this chapter to show how to relate finite tree automata and formulas.

In general, the problem with these techniques is to design an appropriate notion of automaton, which is able to recognize the solutions of atomic formulas and which has the desired closure and decision properties. We have to cite here the famous *Rabin automata* which work on infinite trees and which have indeed the closure and decidability properties, allowing to decide the full second-order monadic logic with k successors (a result due to M. Rabin, 1969). It is however out of the scope of this book to survey automata techniques in logic and computer science. We restrict our attention to finite automata on finite trees and refer to the excellent surveys [Rab77, Tho90] for more details on other applications of automata to logic.

We start this chapter by reviewing some possible definitions of automata on pairs (or, more generally, tuples) of finite trees in Section 3.2. We define in this way several notions of recognizability for relations, which are not necessary unary, extending the frame of chapter 1. This extension is necessary since, automata recognizing the solutions of formulas actually recognize n-tuples of solutions, if there are n free variables in the formula.

The most natural way of defining a notion of recognizability on tuples is to consider products of recognizable sets. Though this happens to be sometimes sufficient, this notion is often too weak. For instance the example of Figure 3.1 could not be defined as a product of recognizable sets. Rather, we stacked the words and recognized these codings. Such a construction can be generalized to trees (we have to overlap instead of stacking) and gives rise to a second notion of recognizability. We will also introduce a third class called "Ground Tree Transducers" which is weaker than the second class above but enjoys stronger closure properties, for instance by iteration. Its usefulness will become evident in Section 3.4.

Next, in Section 3.3, we introduce the weak second-order monadic logic with k successor and show Thatcher and Wright's theorem which relates this logic with finite tree automata. This is a modest insight into the relations between logic and automata.

Finally in Section 3.4 we survey a number of applications, mostly issued from Term Rewriting or Constraint Solving. We do not detail this part (we give references instead). The goal is to show how the simple techniques developed before can be applied to various questions, with a special emphasis on decision problems. We consider the theories of *sort constraints* in Section 3.4.1, the theory of *linear encompassment* in Section 3.4.2, the theory of ground term rewriting in Section 3.4.3 and reduction strategies in orthogonal term rewriting in Section 3.4.4. Other examples are given as exercises in Section 3.5 or considered in chapters 4 and 5.

3.2 Automata on Tuples of Finite Trees

3.2.1 Three Notions of Recognizability

Let $Rec_\times$ be the subset of n-ary relations on $T(\mathcal{F})$ which are finite unions of products $S_1 \times \ldots \times S_n$ where $S_1, \ldots, S_n$ are recognizable subsets of $T(\mathcal{F})$. This notion of recognizability of pairs is the simplest one can imagine. Automata for such relations consist of pairs of tree automata which work independently. This notion is however quite weak, as e.g. the diagonal

$$\Delta = \{(t, t) \mid t \in T(\mathcal{F})\}$$

does not belong to $Rec_\times$. Actually a relation $R \in Rec_\times$ does not really relate its components!

The second notion of recognizability is used in the correspondence with WSkS and is strictly stronger than the above one. Roughly, it consists in overlapping the components of a n-tuple, yielding a term on a product alphabet. Then define Rec as the set of sets of pairs of terms whose overlapping coding is recognized by a tree automaton on the product alphabet.

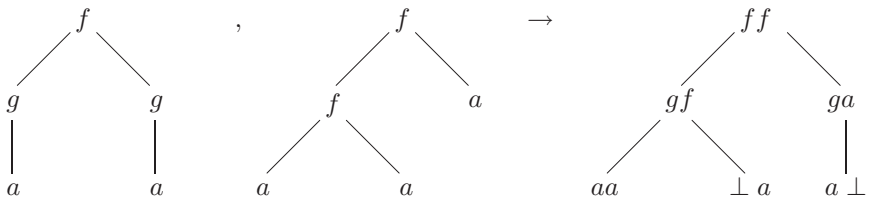


Figure 3.2: The overlap of two terms

Let us first define more precisely the notion of "coding". (This is illustrated by an example on Figure 3.2). We let $\mathcal{F}' = (\mathcal{F} \cup \{\bot\})^n$, where $\bot$ is a new symbol. This is the idea of "stacking" the symbols, as in the introductory example of Presburger's arithmetic. Let k be the maximal arity of a function symbol in $\mathcal{F}$. Assuming $\bot$ has arity 0, the arities of function symbols in $\mathcal{F}'$ are defined by $a(f_1 \dots f_n) = \max(a(f_1), \dots, a(f_n))$.

The coding of two terms $t_1, t_2 \in T(\mathcal{F})$ is defined by induction:

$$[f(t_1, \dots, t_n), g(u_1, \dots, u_m)] \stackrel{\text{def}}{=} fg([t_1, u_1], \dots [t_m, u_m], [t_{m+1}, \bot], \dots, [t_n, \bot])$$

if $n \geq m$ and

$$[f(t_1, \dots, t_n), g(u_1, \dots, u_m)] \stackrel{\text{def}}{=} fg([t_1, u_1], \dots [t_n, u_n], [\bot, u_{n+1}], \dots, [\bot, u_m])$$

if $m \geq n$.

More generally, the coding of n terms $f_1(t_1^1, \dots, t_1^{k_1}), \dots, f_n(t_1^n, \dots, t_n^{k_n})$ is defined as

$$f_1 \dots f_n([t_1^1, \dots, t_n^1], \dots, [t_1^m, \dots, t_n^m])$$

where m is the maximal arity of $f_1, \dots, f_n \in \mathcal{F}$ and t_i^j is, by convention, $\bot$ when $j > k_i$.

Definition 4. *Rec is the set of relations $R \subseteq T(\mathcal{F})^n$ such that*

$$\{[t_1, \dots, t_n] \mid (t_1, \dots, t_n) \in R\}$$

is recognized by a finite tree automaton on the alphabet $\mathcal{F}' = (\mathcal{F} \cup \{\bot\})^n$.

For example, consider the diagonal Δ, it is in *Rec* since its coding is recognized by the bottom-up tree automaton whose only state is q (also a final state) and transitions are the rules $ff(q, \dots, q) \to q$ for all symbols $f \in \mathcal{F}$.

One drawback of this second notion of recognizability is that it is not closed under iteration. More precisely, there is a binary relation R which belongs to *Rec* and whose transitive closure is not in *Rec* (see Section 3.5). For this reason, a third class of recognizable sets of pairs of trees was introduced: the *Ground Tree Transducers* (GTT for short) .

Definition 5. *A GTT is a pair of bottom-up tree automata $(\mathcal{A}_1, \mathcal{A}_2)$ working on the same alphabet. Their sets of states may however share some symbols (the* synchronization states*).*

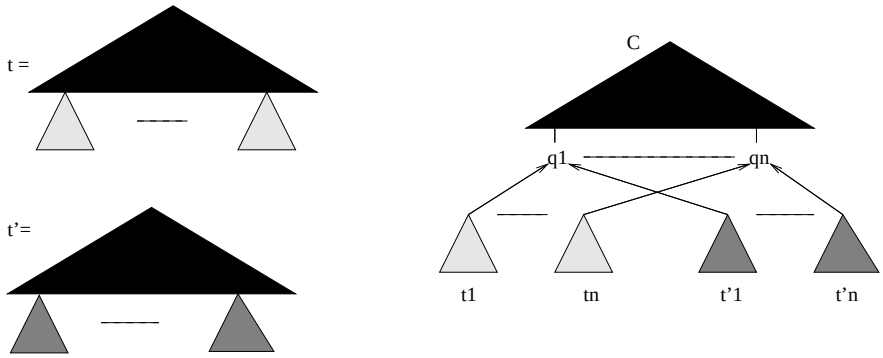


Figure 3.3: GTT acceptance

A pair (t, t') is recognized by a GTT $(\mathcal{A}_1, \mathcal{A}_2)$ if there is a context $C \in \mathcal{C}^n(\mathcal{F})$ such that $t = C[t_1, \ldots, t_n]$, $t' = C[t'_1, \ldots, t'_n]$ and there are states $q_1, \ldots, q_n$ of both automata such that, for all i, $t_i \xrightarrow[\mathcal{A}_1]{*} q_i$ and $t'_i \xrightarrow[\mathcal{A}_2]{*} q_i$. We write $L(\mathcal{A}_1, \mathcal{A}_2)$ the language accepted by the GTT $(\mathcal{A}_1, \mathcal{A}_2)$, i.e.the set of pairs of terms which are recognized.

The recognizability by a GTT is depicted on Figure 3.3. For instance, Δ is accepted by a GTT. Another typical example is the binary relation "one step parallel rewriting" for term rewriting system whose left members are linear and whose right hand sides are ground (see Section 3.4.3).

3.2.2 Examples of The Three Notions of Recognizability

The first example illustrates $Rec_\times$. It will be developed in a more general framework in Section 3.4.2.

Example 29. Consider the alphabet $\mathcal{F} = \{f, g, a\}$ where f is binary, g is unary and a is a constant. Let P be the predicate which is true on t if there are terms t_1, t_2 such that $f(g(t_1), t_2)$ is a subterm of t. Then the solutions of $P(x) \wedge P(y)$ define a relation in $Rec_\times$, using twice the following automaton:

$$
\begin{array}{lcl}
Q & = & \{q_f, q_g, q_\top\} \\
Q_f & = & \{q_f\} \\
T & = & \{
\begin{array}{rclrcl}
a & \to & q_\top & f(q_\top, q_\top) & \to & q_\top \\
g(q_\top) & \to & q_\top & f(q_f, q_\top) & \to & q_f \\
g(q_f) & \to & q_f & f(q_g, q_\top) & \to & q_f \\
g(q_\top) & \to & q_g & f(q_\top, q_f) & \to & q_f\}
\end{array}
\end{array}
$$

For instance the pair $(g(f(g(a), g(a))), f(g(g(a)), a))$ is accepted by the pair of automata.

The second example illustrates *Rec*. Again, it is a first account of the developments of Section 3.4.4

Example 30. Let $\mathcal{F} = \{f, g, a, \Omega\}$ where f is binary, g is unary, a and Ω are constants. Let R be the set of terms (t, u) such that u can be obtained from t by replacing each occurrence of Ω by some term in $T(\mathcal{F})$ (each occurrence of Ω needs not to be replaced with the same term). Using the notations of Chapter 2

$$R(t,u) \Longleftrightarrow u \in t._\Omega T(\mathcal{F})$$

R is recognized by the following automaton (on codings of pairs):

$$\begin{array}{lcl}
Q & = & \{q, q'\} \\
Q_f & = & \{q'\} \\
T & = & \{ \quad \begin{array}{rcl@{\qquad}rcl}
\bot\, a & \to & q & \bot\, f(q,q) & \to & q \\
\bot\, g(q) & \to & q & \Omega f(q,q) & \to & q' \\
\bot\, \Omega & \to & q & f f(q',q') & \to & q' \\
aa & \to & q' & gg(q') & \to & q' \\
\Omega\Omega & \to & q' & \Omega g(q) & \to & q' \\
\Omega a & \to & q' \}
\end{array}
\end{array}$$

For instance, the pair $(f(g(\Omega), g(\Omega)), f(g(g(a)), g(\Omega)))$ is accepted by the automaton: the overlap of the two terms yields

$$[tu] = f f(gg(\Omega g(\bot\, a)), gg(\Omega\Omega))$$

And the reduction:

$$\begin{array}{rcl}
[tu] & \xrightarrow{*} & f f(gg(\Omega g(q)), gg(q')) \\
& \xrightarrow{*} & f f(gg(q'), q') \\
& \to & f f(q', q') \\
& \to & q'
\end{array}$$

The last example illustrates the recognition by a GTT. It comes from the theory of rewriting; further developments and explanations on this theory are given in Section 3.4.3.

Example 31. Let $\mathcal{F} = \{\times, +, 0, 1\}$. Let $\mathcal{R}$ be the rewrite system $0 \times x \to 0$. The many-steps reduction relation defined by $\mathcal{R}$: $\xrightarrow[\mathcal{R}]{*}$ is recognized by the GTT$(\mathcal{A}_1, \mathcal{A}_2)$ defined as follows ($+$ and $\times$ are used in infix notation to meet their usual reading):

$$\begin{array}{lcl}
T_1 & = & \{ \quad \begin{array}{rcl@{\qquad}rcl}
0 & \to & q_\top & q_\top + q_\top & \to & q_\top \\
1 & \to & q_\top & q_\top \times q_\top & \to & q_\top \\
0 & \to & q_0 & q_0 \times q_\top & \to & q_0 \}
\end{array} \\
T_2 & = & \{ \quad 0 \to q_0 \}
\end{array}$$

Then, for instance, the pair $(1 + ((0 \times 1) \times 1), 1 + 0)$ is accepted by the GTT since

$$1 + ((0 \times 1) \times 1) \xrightarrow[\mathcal{A}_1]{*} 1 + (q_0 \times q_\top) \times q_\top \xrightarrow[\mathcal{A}_1]{} 1 + (q_0 \times q_\top) \xrightarrow[\mathcal{A}_1]{} 1 + q_0$$

one hand and $1 + 0 \xrightarrow[\mathcal{A}_2]{} 1 + q_0$ on the other hand.

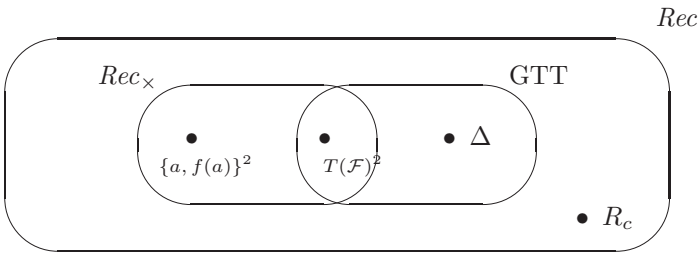


Figure 3.4: The relations between the three classes

3.2.3 Comparisons Between the Three Classes

We study here the inclusion relations between the three classes: $Rec_\times, Rec, GTT$.

Proposition 8. *$Rec_\times \subset Rec$ and the inclusion is strict.*

Proof. To show that any relation in $Rec_\times$ is also in Rec, we have to construct from two automata $\mathcal{A}_1 = (Q_1, \mathcal{F}, Q_1^f, R_1), \mathcal{A}_2 = (\mathcal{F}, Q_2, Q_2^f, R_2)$ an automaton which recognizes the overlaps of the terms in the languages. We define such an automaton $\mathcal{A} = (Q, (\mathcal{F} \cup \{\bot\})^2, Q^f, R)$ by: $Q = (Q_1 \cup \{q_\bot\}) \times (Q_2 \cup \{q_\bot\})$, $Q^f = Q_1^f \times Q_2^f$ and R is the set of rules:

- $f \bot ((q_1, q_\bot), \ldots, (q_n, q_\bot)) \rightarrow (q, q_\bot)$ if $f(q_1, \ldots, q_n) \rightarrow q \in R_1$
- $\bot f((q_\bot, q_1), \ldots, (q_\bot, q_n)) \rightarrow (q_\bot, q)$ if $f(q_1, \ldots, q_n) \rightarrow q \in R_2$
- $fg((q_1, q_1'), \ldots, (q_m, q_m'), (q_{m+1}, q_\bot), \ldots, (q_n, q_\bot)) \rightarrow (q, q')$ if $f(q_1, \ldots, q_n) \rightarrow q \in R_1$ and $g(q_1', \ldots, q_m') \rightarrow q' \in R_2$ and $n \geq m$
- $fg((q_1, q_1'), \ldots, (q_n, q_n'), (q_\bot, q_{n+1}), \ldots, (q_\bot, q_m)) \rightarrow (q, q')$ if $f(q_1, \ldots, q_n) \rightarrow q \in R_1$ and $g(q_1', \ldots, q_m') \rightarrow q' \in R_2$ and $m \geq n$

The proof that $\mathcal{A}$ indeed accepts $L(\mathcal{A}_1) \times L(\mathcal{A}_2)$ is left to the reader.

Now, the inclusion is strict since e.g. $\Delta \in Rec \setminus Rec_\times$. □

Proposition 9. *$GTT \subset Rec$ and the inclusion is strict.*

Proof. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be a GTT accepting R. We have to construct an automaton $\mathcal{A}$ which accepts the codings of pairs in R.

Let $\mathcal{A}_0 = (Q_0, \mathcal{F}, Q_0^f, T_0)$ be the automaton constructed in the proof of Proposition 8. $[t, u] \xrightarrow[\mathcal{A}_0]{*} (q_1, q_2)$ if and only if $t \xrightarrow[\mathcal{A}_1]{*} q_1$ and $u \xrightarrow[\mathcal{A}_2]{*} q_2$. Now we let $\mathcal{A} = (Q_0 \cup \{q_f\}, \mathcal{F}, Q_f = \{q_f\}, T)$. T consists of T_0 plus the following rules:

$$(q, q) \rightarrow q_f \qquad ff(q_f, \ldots, q_f) \rightarrow q_f$$

For every symbol $f \in F$ and every state $q \in Q_0$.

If (t, u) is accepted by the GTT, then

$$t \xrightarrow[\mathcal{A}_1]{*} C[q_1, \ldots, q_n]_{p_1, \ldots, p_n} \xleftarrow[\mathcal{A}_2]{*} u.$$

Then

$$[t,u] \xrightarrow[\mathcal{A}_0]{*} [C,C][(q_1,q_1),\ldots,(q_n,q_n)]_{p_1,\ldots,p_n} \xrightarrow[\mathcal{A}]{*} [C,C][q_f,\ldots,q_f]_{p_1,\ldots,p_n} \xrightarrow[\mathcal{A}]{*} q_f$$

Conversely, if $[t,u]$ is accepted by $\mathcal{A}$ then $[t,u] \xrightarrow[\mathcal{A}]{*} q_f$. By definition of $\mathcal{A}$, there should be a sequence:

$$[t,u] \xrightarrow[\mathcal{A}]{*} C[(q_1,q_1),\ldots,(q_n,q_n)]_{p_1,\ldots,p_n} \xrightarrow[\mathcal{A}]{*} C[q_f,\ldots,q_f]_{p_1,\ldots,p_n} \xrightarrow[\mathcal{A}]{*} q_f$$

Indeed, we let p_i be the positions at which one of the ϵ-transitions steps $(q,q) \to q_f$ is applied. $(n \geq 0)$. Now, $C[q_f,\ldots,q_f]_{p_1,\ldots,p_m} q_f$ if and only if C can be written $[C_1,C_1]$ (the proof is left to the reader).

Concerning the strictness of the inclusion, it will be a consequence of Propositions 8 and 10. □

Proposition 10. *$GTT \not\subseteq Rec_\times$ and $Rec_\times \not\subseteq GTT$.*

Proof. Δ is accepted by a GTT (with no state and no transition) but it does not belong to $Rec_\times$. On the other hand, if $\mathcal{F} = \{f,a\}$, then $\{a, f(a)\}^2$ is in $Rec_\times$ (it is the product of two finite languages) but it is not accepted by any GTT since any GTT accepts at least Δ. □

Finally, there is an example of a relation R_c which is in *Rec* and not in the union $Rec_\times \cup \mathrm{GTT}$; consider for instance the alphabet $\{a(),b(),0\}$ and the one step reduction relation associated with the rewrite system $a(x) \to x$. In other words,

$$(u,v) \in R_c \Longleftrightarrow \exists C \in \mathcal{C}(\mathcal{F}), \exists t \in T(\mathcal{F}), u = C[a(t)] \wedge v = C[t]$$

It is left as an exercise to prove that $R_c \in Rec \setminus (Rec_\times \cup \mathrm{GTT})$.

3.2.4 Closure Properties for $Rec_\times$ and *Rec*; Cylindrification and Projection

Let us start with the classical closure properties.

Proposition 11. *$Rec_\times$ and Rec are closed under Boolean operations.*

The proof of this proposition is straightforward and left as an exercise.

These relations are also closed under *cylindrification* and *projection*. Let us first define these operations which are specific to automata on tuples:

Definition 6. *If $R \subseteq T(\mathcal{F})^n$ $(n \geq 1)$ and $1 \leq i \leq n$ then the* ith projection *of R is the relation $R_i \subseteq T(\mathcal{F})^{n-1}$ defined by*

$$R_i(t_1,\ldots,t_{n-1}) \Leftrightarrow \exists t \in T(\mathcal{F}) \;\; R(t_1,\ldots,t_{i-1},t,t_i,\ldots,t_{n-1})$$

When $n=1$, $T(\mathcal{F})^{n-1}$ is by convention a singleton set $\{\top\}$ (so as to keep the property that $T(\mathcal{F})^{n+1} = T(\mathcal{F}) \times T(\mathcal{F})^n$). $\{\top\}$ is assumed to be a neutral element *w.r.t.*Cartesian product. In such a situation, a relation $R \subseteq T(\mathcal{F})^0$ is either $\emptyset$ or $\{\top\}$ (it is a propositional variable).

Definition 7. *If $R \subseteq T(\mathcal{F})^n$ $(n \geq 0)$ and $1 \leq i \leq n+1$, then the* ith cylindrification *of R is the relation $R^i \subseteq T(\mathcal{F})^{n+1}$ defined by*

$$R^i(t_1, \ldots, t_{i-1}, t, t_i, \ldots, t_n) \Leftrightarrow R(t_1, \ldots, t_{i-1}, t_i, \ldots, t_n)$$

Proposition 12. *$Rec_\times$ and Rec are effectively closed under projection and cylindrification. Actually, ith projection can be computed in linear time and the ith cylindrification of $\mathcal{A}$ can be computed in linear time (assuming that the size of the alphabet is constant).*

Proof. For $Rec_\times$, this property is easy: projection on the ith component simply amounts to remove the ith automaton. Cylindrification on the ith component simply amounts to insert as a ith automaton, an automaton accepting all terms.

Assume that $R \in Rec$. The ith projection of R is simply its image by the following linear tree homomorphism:

$$h_i([f_1, \ldots, f_n](t_1, \ldots, t_k)) \stackrel{\text{def}}{=} [f_1 \ldots f_{i-1} f_{i+1} \ldots f_n](h_i(t_1), \ldots, h_i(t_m))$$

in which m is the arity of $[f_1 \ldots f_{i-1} f_{i+1} \ldots f_n]$ (which is smaller or equal to k). Hence, by Theorem 6, the ith projection of R is recognizable (and we can extract from the proof a linear construction of the automaton).

Similarly, the ith cylindrification is obtained as an inverse homomorphic image, hence is recognizable thanks to Theorem 7.

□

Note that using the above construction, the projection of a deterministic automaton may be non-deterministic (see exercises)

Example 32. Let $\mathcal{F} = \{f, g, a\}$ where f is binary, g is unary and a is a constant. Consider the following automaton $\mathcal{A}$ on $\mathcal{F}' = (\mathcal{F} \cup \{\perp\})^2$: The set of states is $\{q_1, q_2, q_3, q_4, q_5\}$ and the set of final states is $\{q_3\}$[1]

$$\begin{array}{rcl} a\,\perp & \to & q_1 \\ g\,\perp(q_1) & \to & q_1 \\ ga(q_1) & \to & q_2 \\ g\,\perp(q_1) & \to & q_4 \\ gg(q_3) & \to & q_3 \\ aa & \to & q_5 \\ gg(q_5) & \to & q_5 \\ ff(q_5, q_5) & \to & q_5 \end{array} \qquad \begin{array}{rcl} f\,\perp(q_1, q_1) & \to & q_1 \\ fg(q_2, q_1) & \to & q_3 \\ f\,\perp(q_4, q_1) & \to & q_4 \\ fa(q_4, q_1) & \to & q_2 \\ ff(q_3, q_3) & \to & q_3 \\ ff(q_3, q_5) & \to & q_3 \\ ff(q_5, q_3) & \to & q_3 \end{array}$$

The first projection of this automaton gives:

$$\begin{array}{rcl} a & \to & q_2 \\ a & \to & q_5 \\ g(q_2) & \to & q_3 \\ f(q_3, q_5) & \to & q_3 \\ f(q_5, q_3) & \to & q_3 \end{array} \qquad \begin{array}{rcl} g(q_3) & \to & q_3 \\ g(q_5) & \to & q_5 \\ f(q_3, q_3) & \to & q_3 \\ f(q_5, q_5) & \to & q_5 \end{array}$$

[1] This automaton accepts the set of pairs of terms (u, v) such that u can be rewritten in one or more steps to v by the rewrite system $f(g(x), y) \to g(a)$.

Which accepts the terms containing $g(a)$ as a subterm[2].

3.2.5 Closure of GTT by Composition and Iteration

Theorem 23. *If $R \subseteq T(\mathcal{F})^2$ is recognized by a GTT, then its transitive closure R^* is also recognized by a GTT.*

The detailed proof is technical, so let us show it on a picture and explain it informally.

We consider two terms (t, v) and (v, u) which are both accepted by the GTT and we wish that (t, u) is also accepted. For simplicity, consider only one state q such that $t \xrightarrow[\mathcal{A}_1]{*} C[q] \xleftarrow[\mathcal{A}_2]{*} v$ and $v \xrightarrow[\mathcal{A}_1]{*} C'[q_1, \ldots, q_n] \xleftarrow[\mathcal{A}_2]{*} u$. There are actually two cases: C can be "bigger" than C' or "smaller". Assume it is smaller. Then q is reached at a position inside C': $C' = C[C'']_p$. The situation is depicted on Figure 3.5. Along the reduction of v to q by $\mathcal{A}_2$, we enter a configuration $C''[q'_1, \ldots, q'_n]$. The idea now is to add to $\mathcal{A}_2$ ϵ-transitions from q_i to q'_i. In this way, as can easily be seen on Figure 3.5, we get a reduction from u to $C[q]$, hence the pair (t, u) is accepted.

Proof. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be the pair of automata defining the GTT which accepts R. We compute by induction the automata $\mathcal{A}_1^n, \mathcal{A}_2^n$. $\mathcal{A}_i^0 = \mathcal{A}_i$ and $\mathcal{A}_i^{n+1}$ is obtained by adding new transitions to $\mathcal{A}_i^n$: Let Q_i be the set of states of $\mathcal{A}_i$ (and also the set of states of $\mathcal{A}_i^n$).

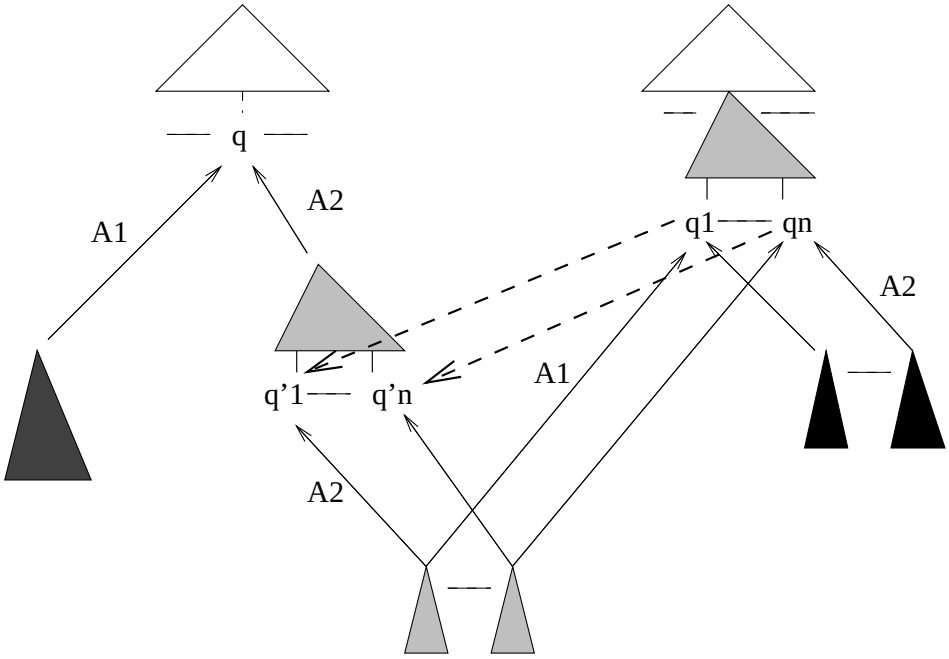
- If $L_{\mathcal{A}_2^n}(q) \cap L_{\mathcal{A}_1^n}(q') \neq \emptyset$, $q \in Q_1 \cap Q_2$ and $q \not\xrightarrow[\mathcal{A}_1^n]{*} q'$, then $\mathcal{A}_1^{n+1}$ is obtained from $\mathcal{A}_1^n$ by adding the ϵ-transition $q \to q'$ and $\mathcal{A}_2^{n+1} = \mathcal{A}_2^n$.
- If $L_{\mathcal{A}_1^n}(q) \cap L_{\mathcal{A}_2^n}(q') \neq \emptyset$, $q \in Q_1 \cap Q_2$ and $q \not\xrightarrow[\mathcal{A}_2^n]{*} q'$, then $\mathcal{A}_2^{n+1}$ is obtained from $\mathcal{A}_2^n$ by adding the ϵ-transition $q \to q'$ and $\mathcal{A}_1^{n+1} = \mathcal{A}_1^n$.

If there are several ways of obtaining $\mathcal{A}_i^{n+1}$ from $\mathcal{A}_i^n$ using these rules, we don't care which of these ways is used.

First, these completion rules are decidable by the decision properties of chapter 1. Their application also terminates as at each application strictly decreases $k_1(n) + k_2(n)$ where $k_i(n)$ is the number of pairs of states $(q, q') \in (Q_1 \cup Q_2) \times (Q_1 \cup Q_2)$ such that there is no ϵ-transition in $\mathcal{A}_i^n$ from q to q'. We let $\mathcal{A}_i^*$ be a fixed point of this computation. We show that $(\mathcal{A}_1^*, \mathcal{A}_2^*)$ defines a GTT accepting R^*.

- Each pair of terms accepted by the GTT $(\mathcal{A}_1^*, \mathcal{A}_2^*)$ is in R^*: we show by induction on n that each pair of terms accepted by the GTT $(\mathcal{A}_1^n, \mathcal{A}_2^n)$ is in R^*. For $n = 0$, this follows from the hypothesis. Let us now assume that $\mathcal{A}_1^{n+1}$ is obtained by adding $q \to q'$ to the transitions of $\mathcal{A}_1^n$ (The other case is symmetric). Let (t, u) be accepted by the GTT $(\mathcal{A}_1^{n+1}, \mathcal{A}_2^{n+1})$. By definition, there is a context C and positions $p_1, \ldots, p_k$

[2] *i.e.*the terms that are obtained by applying at least one rewriting step using $f(g(x), y) \to g(a)$



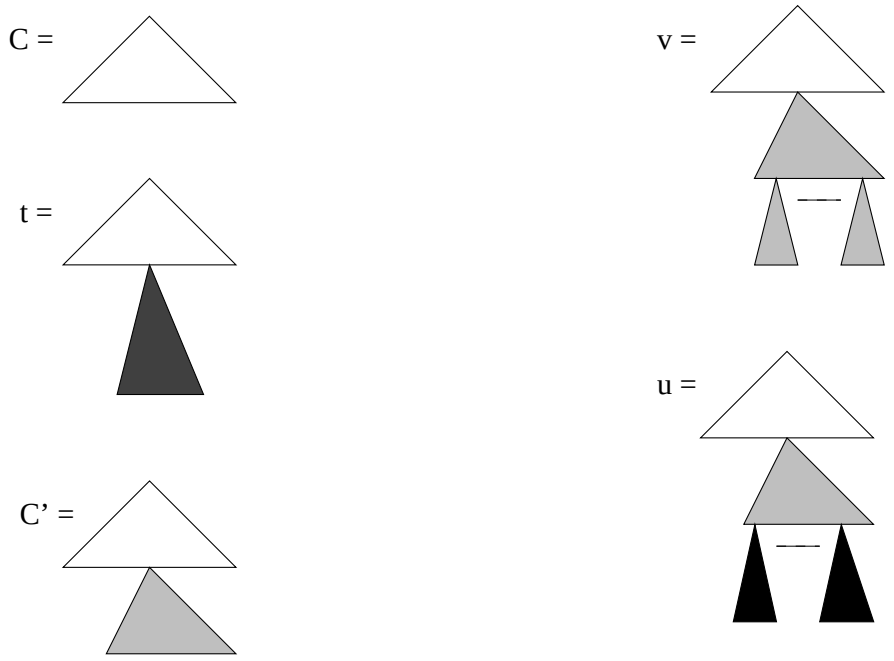


Figure 3.5: The proof of Theorem 23

such that $t = C[t_1, \ldots, t_k]_{p_1,\ldots,p_k}$, $u = C[u_1, \ldots, u_k]_{p_1,\ldots,p_k}$ and there are states $q_1, \ldots, q_k \in Q_1 \cap Q_2$ such that, for all i, $t_i \xrightarrow[\mathcal{A}_1^{n+1}]{*} q_i$ and $u_i \xrightarrow[\mathcal{A}_2^{n}]{*} q_i$.

We prove the result by induction on the number m of times $q \to q'$ is applied in the reductions $t_i \xrightarrow[\mathcal{A}_1^{n+1}]{*} q_i$. If $m = 0$. Then this is the first induction hypothesis: (t, u) is accepted by $(\mathcal{A}_1^n, \mathcal{A}_2^n)$, hence $(t, u) \in R^*$. Now, assume that, for some i,

$$t_i \xrightarrow[\mathcal{A}_1^{n+1}]{*} t_i'[q]_p \xrightarrow[q\to q']{*} t_i'[q']_p \xrightarrow[\mathcal{A}_1^{n}]{*} q_i$$

By definition, there is a term v such that $v \xrightarrow[\mathcal{A}_2^{n}]{*} q$ and $v \xrightarrow[\mathcal{A}_1^{n}]{*} q'$. Hence

$$t_i[v]_p \xrightarrow[\mathcal{A}_1^{n+1}]{*} q_i$$

And the number of reduction steps using $q \to q'$ is strictly smaller here than in the reduction from t_i to q_i. Hence, by induction hypothesis, $(t[v]_{p_i p}, u) \in R^*$. On the other hand, $(t, t[v]_{p_i p})$ is accepted by the GTT $(\mathcal{A}_1^{n+1}, \mathcal{A}_2^n)$ since $t|_{p_i p} \xrightarrow[\mathcal{A}_1^{n+1}]{*} q$ and $v \xrightarrow[\mathcal{A}_2^{n}]{*} q$. Moreover, by construction, the first sequence of reductions uses strictly less than m times the transition $q \to q'$. Then, by induction hypothesis, $(t, t[v]_{p_i p}) \in R^*$. Now from $(t, t[v]_{p_i p}) \in R^*$ and $(t[v]_{p_i p}, u) \in R^*$, we conclude $(t, u) \in R^*$.

- If $(t, u) \in R^*$, then (t, u) is accepted by the GTT $(\mathcal{A}_1^*, \mathcal{A}_2^*)$. Let us prove the following intermediate result:

  Lemma 1.

  $$\textit{If} \left\{ \begin{array}{l} t \xrightarrow[\mathcal{A}_1^*]{*} q \\ v \xrightarrow[\mathcal{A}_2^*]{*} q \\ v \xrightarrow[\mathcal{A}_1^*]{*} C[q_1, \ldots, q_k]_{p_1,\ldots,p_k} \\ u \xrightarrow[\mathcal{A}_2^*]{*} C[q_1, \ldots, q_k]_{p_1,\ldots,p_k} \end{array} \right\} \textit{ then } u \xrightarrow[\mathcal{A}_2^*]{*} q$$

  and hence (t, u) is accepted by the GTT.

  Let $v \xrightarrow[\mathcal{A}_2^*]{*} C[q_1', \ldots, q_k']_{p_1,\ldots,p_k} \xrightarrow[\mathcal{A}_2^*]{*} q$. For each i, $v|_{p_i} \in L_{\mathcal{A}_2^*}(q_i') \cap L_{\mathcal{A}_1^*}(q_i)$ and $q_i \in Q_1 \cap Q_2$. Hence, by construction, $q_i \xrightarrow[\mathcal{A}_2^*]{} q_i'$. It follows that

  $$u \xrightarrow[\mathcal{A}_2^*]{*} C[q_1, \ldots q_k]_{p_1,\ldots,p_k} \xrightarrow[\mathcal{A}_2^*]{*} C[q_1', \ldots, q_k']_{p_1,\ldots,p_k} \xrightarrow[\mathcal{A}_2^*]{*} q$$

  Which proves our lemma.

  Symmetrically, if $t \xrightarrow[\mathcal{A}_1^*]{*} C[q_1, \ldots, q_k]_{p_1,\ldots,p_k}$, $v \xrightarrow[\mathcal{A}_2^*]{*} C[q_1, \ldots, q_k]_{p_1,\ldots,p_k}$, $v \xrightarrow[\mathcal{A}_1^*]{*} q$ and $u \xrightarrow[\mathcal{A}_2^*]{*} q$, then $t \xrightarrow[\mathcal{A}_1^*]{*} q$

Now, let $(t,u) \in R^n$: we prove that (t,u) is accepted by the GTT $(\mathcal{A}_1^*, \mathcal{A}_2^*)$ by induction on n. If $n = 1$, then the result follows from the inclusion of $L(\mathcal{A}_1, \mathcal{A}_2)$ in $L(\mathcal{A}_1^*, \mathcal{A}_2^*)$. Now, let v be such that $(t,v) \in R$ and $(v,u) \in R^{n-1}$. By induction hypothesis, both (t,v) and (v,u) are accepted by the GTT $(\mathcal{A}_1^*, \mathcal{A}_2^*)$: there are context C and C' and positions $p_1, \ldots, p_k, p'_1, \ldots, p'_m$ such that

$$t = C[t_1, \ldots, t_k]_{p_1,\ldots,p_k},\ v = C[v_1, \ldots, v_k]_{p_1,\ldots,p_k}$$

$$v = C'[v'_1, \ldots, v'_m]_{p'_1,\ldots,p'_m},\ u = C'[u_1, \ldots, u_m]$$

and states $q_1, \ldots, q_k, q'_1, \ldots, q'_m \in Q_1 \cap Q_2$ such that for all i, j, $t_i \xrightarrow[\mathcal{A}_1^*]{*} q_i$, $v_i \xrightarrow[\mathcal{A}_2^*]{*} q_i$, $v'_j \xrightarrow[\mathcal{A}_1^*]{*} q'_j$, $u_j \xrightarrow[\mathcal{A}_2^*]{*} q'_j$. Let C'' be the largest context more general than C and C'; the positions of C'' are the positions of both $C[q_1, \ldots, q_n]_{p_1,\ldots,p_n}$ and $C'[q'_1, \ldots, q'_m]_{p'_1,\ldots,p'_m}$. C'', $p''_1, \ldots, p''_l$ are such that:

- For each $1 \leq i \leq l$, there is a j such that either $p_j = p''_i$ or $p'_j = p''_i$
- For all $1 \leq i \leq n$ there is a j such that $p_i \geq p''_j$
- For all $1 \leq i \leq m$ there is a j such that $p'_i \geq p''_j$
- the positions p''_j are pairwise incomparable *w.r.t.*the prefix ordering.

Let us fix a $j \in [1..l]$. Assume that $p''_j = p_i$ (the other case is symmetric). We can apply our lemma to $t_j = t|_{p''_j}$ (in place of t), $v_j = v|_{p''_j}$ (in place of v) and $u|_{p''_j}$ (in place of u), showing that $u|_{p''_j} \xrightarrow[\mathcal{A}_2^*]{*} q_i$. If we let now $q''_j = q_i$ when $p''_j = p_i$ and $q''_j = q'_i$ when $p''_j = p'_i$, we get

$$t \xrightarrow[\mathcal{A}_1^*]{*} C''[q''_1, \ldots, q''_l]_{p''_1,\ldots,p''_l} \xleftarrow[\mathcal{A}_2^*]{*} u$$

which completes the proof.

□

Proposition 13. *If R and R' are in GTT then their composition $R \circ R'$ is also in GTT.*

Proof. Let $(\mathcal{A}_1, \mathcal{A}_2)$ and $(\mathcal{A}'_1, \mathcal{A}'_2)$ be the two pairs of automata which recognize R and R' respectively. We assume without loss of generality that the set of states are disjoint:

$$(Q_1 \cup Q_2) \cap (Q'_1 \cup Q'_2) = \emptyset$$

We define the automaton $\mathcal{A}_1^*$ as follows: the set of states is $Q_1 \cup Q'_1$ and the transitions are the union of:

- the transitions of $\mathcal{A}_1$
- the transitions of $\mathcal{A}'_1$
- the ϵ-transitions $q \to q'$ if $q \in Q_1 \cap Q_2$, $q' \in Q'_1$ and $L_{\mathcal{A}_2}(q) \cap L_{\mathcal{A}'_1}(q') \neq \emptyset$

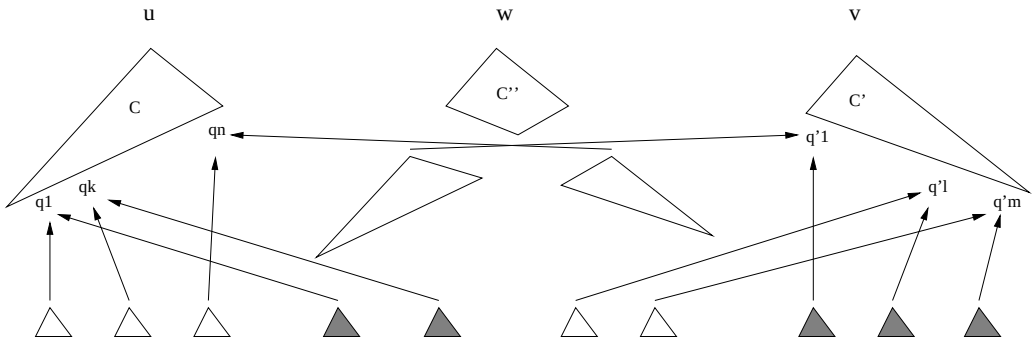


Figure 3.6: The proof of Proposition 13

Symmetrically, the automaton $\mathcal{A}_2^*$ is defined by: its states are $Q_2 \cup Q_2'$ and the transitions are:

- the transitions of $\mathcal{A}_2$
- the transitions of $\mathcal{A}_2'$
- the ϵ-transitions $q' \rightarrow q$ if $q' \in Q_1' \cap Q_2'$, $q \in Q_2$ and $L_{\mathcal{A}_1'}(q') \cap L_{\mathcal{A}_2}(q) \neq \emptyset$

We prove below that $(\mathcal{A}_1^*, \mathcal{A}_2^*)$ is a GTT recognizing $R \circ R'$. See also the figure 3.6.

- Assume first that $(u, v) \in R \circ R'$. Then there is a term w such that $(u, w) \in R$ and $(w, v) \in R'$:

$$u = C[u_1, \ldots, u_k]_{p_1,\ldots,p_k},\ w = C[w_1, \ldots, w_k]_{p_1,\ldots,p_k}$$

$$w = C'[w_1', \ldots, w_m']_{p_1',\ldots,p_m'},\ v = C'[v_1, \ldots, v_m]_{p_1',\ldots,p_m'}$$

and, for every $i \in \{1, \ldots, k\}$, $u_i \xrightarrow[\mathcal{A}_1]{*} q_i$, $w_i \xrightarrow[\mathcal{A}_2]{*} q_i$, for every $i \in \{1, ..., m\}$, $w_i' \xrightarrow[\mathcal{A}_1']{*} q_i'$, $v_i \xrightarrow[\mathcal{A}_2']{*} q_i'$. Let $p_1'', \ldots, p_l''$ be the minimal elements (*w.r.t.*the prefix ordering) of the set $\{p_1, \ldots, p_k\} \cup \{p_1', \ldots, p_m'\}$. Each p_i'' is either some p_j or some p_j'. Assume first $p_i'' = p_j$. Then p_j is a position in C' and

$$C'[q_1', \ldots, q_m']_{p_1',\ldots,p_m'}|_{p_j} = C_j[q_{m_j}', \ldots, q_{m_j+k_j}']_{p_{m_j}',\ldots,p_{m_j+k_j}'}$$

Now, $w_j \xrightarrow[\mathcal{A}_2]{*} q_j$ and

$$w_j = C_j[w_{m_j}', \ldots, w_{m_j+k_j}']_{p_{m_j}',\ldots,p_{m_j+k_j}'}$$

with $w_{m_j+i}' \xrightarrow[\mathcal{A}_1']{*} q_{m_j+i}'$ for every $i \in \{1, \ldots, k_j\}$. For $i \in \{1, \ldots, k_j\}$, let $q_{j,i}$ be such that:

$$\begin{cases} w_{m_j+i}' = w_j|_{p_{m_j+i}'} \xrightarrow[\mathcal{A}_2]{*} q_{j,i} \\ C_j[q_{j,1}, \ldots, q_{j,k_j}]_{p_{m_j}',\ldots,p_{m_j+k_j}'} \xrightarrow[\mathcal{A}_2]{*} q_j \end{cases}$$

For every i, $w'_{m_j+i} \in L_{\mathcal{A}_2}(q_{j,i}) \cap L_{\mathcal{A}'_1}(q'_{m_j+i})$ and $q'_{m_j+i} \in Q'_1 \cap Q'_2$. Then, by definition, there is a transition $q'_{m_j+i} \xrightarrow[\mathcal{A}_2^*]{} q_{j,i}$. Therefore, $C_j[q'_{m_j}, \ldots, q'_{m_j+k_j}] \xrightarrow[\mathcal{A}_2^*]{*} q_j$ and then $v|_{p_j} \xrightarrow[\mathcal{A}_2^*]{*} q_j$.

Now, if $p''_i = p'_j$, we get, in a similar way, $u|_{p'_j} \xrightarrow[\mathcal{A}_1^*]{*} q'_j$. Altogether:

$$u \xrightarrow[\mathcal{A}_1^*]{*} C''[q''_1, \ldots, q''_l]_{p''_1,\ldots,p''_l} \xleftarrow[\mathcal{A}_2^*]{*} v$$

where $q''_i = q_j$ if $p''_i = p_j$ and $q''_j = q'_j$ if $p''_i = p'_j$.

- Conversely, assume that (u, v) is accepted by $(\mathcal{A}_1^*, \mathcal{A}_2^*)$. Then

$$u \xrightarrow[\mathcal{A}_1^*]{*} C[q''_1, \ldots, q''_l]_{p''_1,\ldots,p''_l} \xleftarrow[\mathcal{A}_2^*]{*} v$$

and, for every i, either $q''_i \in Q_1 \cap Q_2$ or $q''_i \in Q'_1 \cap Q'_2$ (by the disjointness hypothesis). Assume for instance that $q''_i \in Q'_1 \cap Q'_2$ and consider the computation of $\mathcal{A}_1^*$: $u|_{p''_i} \xrightarrow[\mathcal{A}_1^*]{*} q''_i$. By definition, $u|_{p''_i} = C_i[u_1, \ldots, u_{k_i}]$ with

$$u_j \xrightarrow[\mathcal{A}_1]{*} q_j \xrightarrow[\mathcal{A}_1^*]{} q'_j$$

for every $j = 1, \ldots, k_i$ and $C_i[q'_1, \ldots, q'_{k_i}] \xrightarrow[\mathcal{A}'_1]{*} q''_i$. By construction, $q_j \in Q_1 \cap Q_2$ and $L_{\mathcal{A}_2}(q_j) \cap L_{\mathcal{A}'_1}(q'_j) \neq \emptyset$. Let $w_{i,j}$ be a term in this intersection and $w_i = C_i[w_{i,1}, \ldots, w_{i,k_i}]$. Then

$$\begin{cases} w_i \xrightarrow[\mathcal{A}_2]{*} C_i[q_1, \ldots, q_{k_i}] \\ u|_{p''_i} \xrightarrow[\mathcal{A}_1]{*} C_i[q_1, \ldots, q_{k_i}] \\ w_i \xrightarrow[\mathcal{A}'_1]{*} q''_i \\ v|_{p''_i} \xrightarrow[\mathcal{A}'_2]{*} q''_i \end{cases}$$

The last property comes from the fact that $v|_{p''_i} \xrightarrow[\mathcal{A}_2^*]{*} q''_i$ and, since $q''_i \in Q'_2$, there can be only transition steps from $\mathcal{A}'_2$ in this reduction.

Symmetrically, if $q''_i \in Q_1 \cap Q_2$, then we define w_i and the contexts C_i such that

$$\begin{cases} w_i \xrightarrow[\mathcal{A}_2]{*} q''_i \\ u|_{p''_i} \xrightarrow[\mathcal{A}_1]{*} q''_i \\ w_i \xrightarrow[\mathcal{A}'_1]{*} C_i[q'_1, \ldots, q'_{k_i}] \\ v|_{p''_i} \xrightarrow[\mathcal{A}'_2]{*} C_i[q'_1, \ldots, q'_{k_i}] \end{cases}$$

Finally, letting $w = C[w_1, \ldots, w_l]$, we have $(u, w) \in R$ and $(w, v) \in R'$.

□

GTTs do not have many other good closure properties (see the exercises).

3.3 The Logic WSkS

3.3.1 Syntax

Terms of WSkS are formed out of the constant ϵ, first-order variable symbols (typically written with lower-case letters $x, y, z, x', x_1, ...$) and unary symbols $1, \ldots, n$ written in postfix notation. For instance $x1123, \epsilon 2111$ are terms. The latter will be often written omitting ϵ (e.g. 2111 instead of $\epsilon 2111$).

Atomic formulas are either equalities $s = t$ between terms, inequalities $s \leq t$ or $s \geq t$ between terms, or membership constraints $t \in X$ where t is a term and X is a *second-order variable symbol*. Second-order variables will be typically denoted using upper-case letters.

Formulas are built from the atomic formulas using the logical connectives $\wedge, \vee, \neg, \Rightarrow, \Leftarrow, \Leftrightarrow$ and the quantifiers $\exists x, \forall x$(quantification on individuals)$\exists X, \forall X$ (quantification on sets); we may quantify both first-order and second-order variables.

As usual, we do not need all this artillery: we may stick to a subset of logical connectives (and even a subset of atomic formulas as will be discussed in Section 3.3.4). For instance $\phi \Leftrightarrow \psi$ is an abbreviation for $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$, $\phi \Rightarrow \psi$ is another way of writing $\psi \Leftarrow \phi$, $\phi \Rightarrow \psi$ is an abbreviation for $(\neg\phi) \vee \psi$, $\forall x.\phi$ stands for $\neg\exists x.\neg\phi$ etc ... We will use the extended syntax for convenience, but we will restrict ourselves to the atomic formulas $s = t, s \leq t, t \in X$ and the logical connectives $\vee, \neg, \exists x, \exists X$ in the proofs.

The set of *free variables* of a formula ϕ is defined as usual.

3.3.2 Semantics

We consider the particular interpretation where terms are strings belonging to $\{1, \ldots, k\}^*$, $=$ is the equality of strings, and $\leq$ is interpreted as the prefix ordering. Second order variables are interpreted as *finite* subsets of $\{1, \ldots, k\}^*$, so $\in$ is then the membership predicate.

Let $t_1, \ldots, t_n \in \{1, \ldots, k\}^*$ and $S_1, \ldots, S_n$ be finite subsets of $\{1, \ldots, k\}^*$. Given a formula

$$\phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$$

with free variables $x_1, \ldots, x_n, X_1, \ldots, X_m$, the assignment $\{x_1 \mapsto t_1, \ldots x_n \mapsto t_n, X_1 \mapsto S_1, \ldots X_m \mapsto S_m\}$ *satisfies* ϕ, which is written $\sigma \models \phi$ (or also $t_1, \ldots, t_n, S_1, \ldots, S_m \models \phi$) if replacing the variables with their corresponding value, the formula holds in the above model.

Remark: the logic SkS is defined as above, except that set variables may be interpreted as infinite sets.

3.3.3 Examples

We list below a number of formulas defining predicates on sets and singletons. After these examples, we may use the below-defined abbreviations as if there were primitives of the logic.

X is a subset of Y:

$$X \subseteq Y \stackrel{\text{def}}{=} \forall x.(x \in X \Rightarrow x \in Y)$$

Finite union:

$$X = \bigcup_{i=1}^{n} X_i \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} X_i \subseteq X \wedge \forall x.(x \in X \Rightarrow \bigvee_{i=1}^{n} x \in X_i)$$

Intersection:

$$X \cap Y = Z \stackrel{\text{def}}{=} \forall x.x \in Z \Leftrightarrow (x \in X \wedge x \in Y)$$

Partition:

$$\text{Partition}(X, X_1, \ldots, X_n) \stackrel{\text{def}}{=} X = \bigcup_{i=1}^{n} X_i \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} X_i \cap X_j = \emptyset$$

X **is closed under prefix:**

$$\text{PrefixClosed}(X) \stackrel{\text{def}}{=} \forall z.\forall y.(z \in X \wedge y \leq z) \Rightarrow y \in X$$

Set equality:

$$Y = X \stackrel{\text{def}}{=} Y \subseteq X \wedge X \subseteq Y$$

Emptiness:

$$X = \emptyset \stackrel{\text{def}}{=} \forall Y.(Y \subseteq X \Rightarrow Y = X)$$

X **is a Singleton:**

$$\text{Sing}(X) \stackrel{\text{def}}{=} X \neq \emptyset \wedge \forall Y(Y \subseteq X \Rightarrow (Y = X \vee Y = \emptyset)$$

The prefix ordering:

$$x \leq y \stackrel{\text{def}}{=} \forall X.(y \in X \wedge (\forall z.(\bigvee_{i=1}^{k} zi \in X) \Rightarrow z \in X)) \Rightarrow x \in X$$

"every set containing y and closed by predecessor contains x"

This shows that $\leq$ can be removed from the syntax of WSkS formulas without decreasing the expressive power of the logic.

Coding of trees: assume that k is the maximal arity of a function symbol in $\mathcal{F}$. If $t \in T(\mathcal{F})$ $C(t)$ is the tuples of sets $(S, S_{f_1}, \ldots, S_{f_n})$ if $\mathcal{F} = \{f_1, \ldots, f_n\}$, $S = \bigcup_{i=1}^{n} S_{f_i}$ and S_{f_i} is the set of positions in t which are labeled with f_i.

For instance $C(f(g(a), f(a, b)))$ is the tuple $S = \{\varepsilon, 1, 11, 2, 21, 22\}$, $S_f = \{\varepsilon, 2\}$, $S_g = \{1\}$, $S_a = \{11, 21\}$, $S_b = \{22\}$.

$(S, S_{f_1}, \ldots, S_{f_n})$ is the coding of some $t \in T(\mathcal{F})$ is defined by:

$$\begin{array}{rll} \text{Term}(X, X_1, \ldots, X_n) \stackrel{\text{def}}{=} & X \neq \emptyset & \\ \wedge & \text{Partition}(X, X_1, \ldots, X_n) & \wedge \text{PrefixClosed}(X) \\ \wedge & \bigwedge_{i=1}^{k} \bigwedge_{a(f_j)=i} (\bigwedge_{l=1}^{i} \forall x.(x \in X_{f_j} \Rightarrow xl \in X) & \\ & \qquad \wedge \bigwedge_{l=i+1}^{k} \forall y.(y \in X_{f_j} \Rightarrow yl \notin X)) & \end{array}$$

3.3.4 Restricting the Syntax

If we consider that a first-order variable is a singleton set, it is possible to transform any formula into an equivalent one which does not contain any first-order variable.

More precisely, we consider now that formulas are built upon the atomic formulas:

$$X \subseteq Y, \mathrm{Sing}(X), X = Yi, X = \epsilon$$

using the logical connectives and second-order quantification only. Let us call this new syntax the *restricted syntax*.

These formulas are interpreted as expected. In particular $\mathrm{Sing}(X)$ holds true when X is a singleton set and $X = Yi$ holds true when X and Y are singleton sets $\{s\}$ and $\{t\}$ respectively and $s = ti$. Let us write $\models_2$ the satisfaction relation for this new logic.

Proposition 14. *There is a translation T from WSkS formulas to the restricted syntax such that*

$$s_1, \ldots, s_n, S_1, \ldots, S_m \models \phi(x_1, \ldots, x_n, X_1, \ldots, X_m)$$

if and only if

$$\{s_1\}, \ldots, \{s_n\}, S_1, \ldots, S_m \models_2 T(\phi)(X_{x_1}, \ldots, X_{x_n}, X_1, \ldots, X_m)$$

Conversely, there is a translation T' from the restricted syntax to WSkS such that

$$S_1, \ldots, S_m \models T'(\phi)(X_1, \ldots, X_m)$$

if and only if

$$S_1, \ldots, S_m \models_2 \phi(X_1, \ldots, X_m))$$

Proof. First, according to the previous section, we can restrict our attention to formulas built upon the only atomic formulas $t \in X$ and $s = t$. Then, each atomic formula is flattened according to the rules:

$$\begin{array}{rcl} ti \in X & \rightarrow & \exists y.y = ti \wedge y \in X \\ xi = yj & \rightarrow & \exists z.z = xi \wedge z = yj \\ ti = s & \rightarrow & \exists z.z = t \wedge zi = s \end{array}$$

The last rule assumes that t is not a variable

Next, we associate a second-order variable X_y to each first-order variable y and transform the flat atomic formulas:

$$\begin{array}{rcl} T(y \in X) & \stackrel{\mathrm{def}}{=} & X_y \subseteq X \\ T(y = xi) & \stackrel{\mathrm{def}}{=} & X_y = X_x i \\ T(x = \epsilon) & \stackrel{\mathrm{def}}{=} & X_x = \epsilon \\ T(x = y) & \stackrel{\mathrm{def}}{=} & X_x = X_y \end{array}$$

The translation of other flat atomic formulas can be derived from these ones, in particular when exchanging the arguments of $=$.

Now, $T(\phi \vee \psi) \stackrel{\text{def}}{=} T(\phi) \vee T(\psi)$, $T(\neg(\phi)) \stackrel{\text{def}}{=} \neg T(\phi)$, $T(\exists X.\phi) \stackrel{\text{def}}{=} \exists X.T(\phi)$, $T(\exists y.\phi) \stackrel{\text{def}}{=} \exists X_y.\text{Sing}(X_y) \wedge T(\phi)$. Finally, we add $\text{Sing}(X_x)$ for each free variable x.

For the converse, the translation T' has been given in the previous section, except for the atomic formulas $X = Yi$ (which becomes $\text{Sing}(X) \wedge \text{Sing}(Y) \wedge \exists x \exists y.x \in X \wedge y \in Y \wedge x = yi$) and $X = \epsilon$ (which becomes $\text{Sing}(X) \wedge \forall x.x \in X \Rightarrow x = \epsilon$).

□

3.3.5 Definable Sets are Recognizable Sets

Definition 8. *A set L of tuples of finite sets of words is* definable in WSkS *if there is a formula ϕ of WSkS with free variables $X_1, \ldots, X_n$ such that*

$$(S_1, \ldots, S_n) \in L \text{ if and only if } S_1, \ldots, S_n \models \phi.$$

Each tuple of finite sets of words $S_1, \ldots, S_n \subseteq \{1, \ldots, k\}^*$ is identified to a finite tree $(S_1, \ldots, S_n)^\sim$ over the alphabet $\{0, 1, \perp\}^n$ where any string containing a 0 or a 1 is k-ary and $\perp^n$ is a constant symbol, in the following way[3]:

$$\mathcal{P}os((S_1, \ldots, S_n)^\sim) \stackrel{\text{def}}{=} \{\epsilon\} \cup \{pi \mid \exists p' \in \bigcup_{i=1}^{n} S_i, p \leq p', i \in \{1, \ldots, k\}\}$$

is the set of prefixes of words in some S_i. The symbol at position p:

$$(S_1, \ldots, S_n)^\sim(p) = \alpha_1 \ldots \alpha_n$$

is defined as follows:

- $\alpha_i = 1$ if and only if $p \in S_i$
- $\alpha_i = 0$ if and only if $p \notin S_i$ and $\exists p' \in S_i$ and $\exists p''.p \cdot p'' = p'$
- $\alpha_i = \perp$ otherwise.

Example 33. Consider for instance $S_1 = \{\epsilon, 11\}$, $S_2 = \emptyset$, $S_3 = \{11, 22\}$ three subsets of $\{1, 2\}^*$. Then the coding $(S_1, S_2, S_3)^\sim$ is depicted on Figure 3.7.

Lemma 2. *If a set L of tuples of finite subsets of $\{1, \ldots, k\}^*$ is definable in WSkS, then $\widetilde{L} \stackrel{def}{=} \{(S_1, \ldots, S_n)^\sim \mid (S_1, \ldots, S_n) \in L\}$ is in Rec.*

Proof. By Proposition 14, if L is definable in WSkS, it is also definable with the restricted syntax. We are going now to prove the lemma by induction on the structure of the formula ϕ which defines L. We assume that all variables in ϕ are bound at most once in the formula and we also assume a fixed total ordering $\leq$ on the variables. If ψ is a subformula of ϕ with free variables $Y_1 < \ldots < Y_n$, we construct an automaton $\mathcal{A}_\psi$ working on the alphabet $\{0, 1, \perp\}^n$ such that $(S_1, \ldots, S_n) \models_2 \psi$ if and only if $(S_1, \ldots, S_n)^\sim \in L(\mathcal{A}_\psi)$

[3]This is very similar to the coding of Section 3.2.1

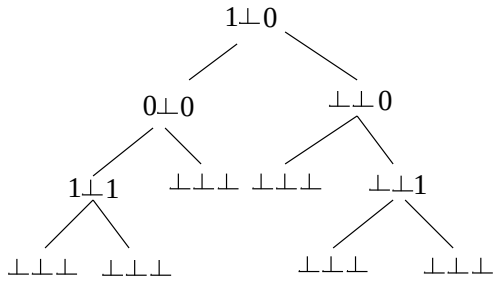


Figure 3.7: An example of a tree coding a triple of finite sets of strings

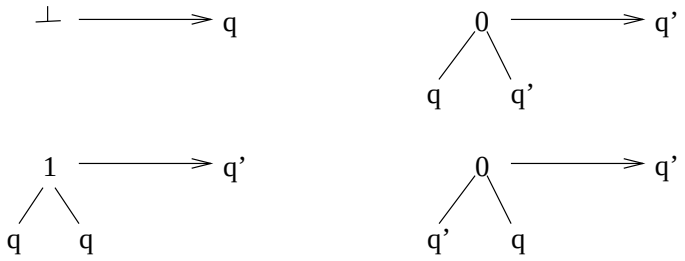


Figure 3.8: The automaton for Sing(X)

The base case consists in constructing an automaton for each atomic formula. (We assume here that $k = 2$ for simplicity, but this works of course for arbitrary k).

The automaton $\mathcal{A}_{\mathrm{Sing}(X)}$ is depicted on Figure 3.8. The only final state is q'.

The automaton $\mathcal{A}_{X\subseteq Y}$ (with $X < Y$) is depicted on Figure 3.9. The only state (which is also final) is q.

The automaton $\mathcal{A}_{X=Y1}$ is depicted on Figure 3.10. The only final state is q''. An automaton for $X = Y2$ is obtained in a similar way.

The automaton for $X = \epsilon$ is depicted on Figure 3.11 (the final state is q').

Now, for the induction step, we have several cases to investigate:

- If ϕ is a disjunction $\phi_1 \vee \phi_2$, where $\vec{X}_i$ are the set of free variables of ϕ_i respectively. Then we first cylindrify the automata for ϕ_1 and ϕ_2 respectively in such a way that they recognize the solutions of ϕ_1 and ϕ_2, with free variables $\vec{X_1} \cup \vec{X_2}$. (See Proposition 12).More precisely, let $\vec{X_1} \cup \vec{X_2} = \{Y_1, \ldots, Y_n\}$ with $Y_1 < \ldots < Y_n$. Then we successively apply the ith cylindrification to the automaton of ϕ_1 (resp. ϕ_2) for the variables Y_i which are not free in ϕ_1 (resp. ϕ_2). Then the automaton $\mathcal{A}_\phi$ is obtained as the union of these automata. (*Rec* is closed under union by Proposition 11).

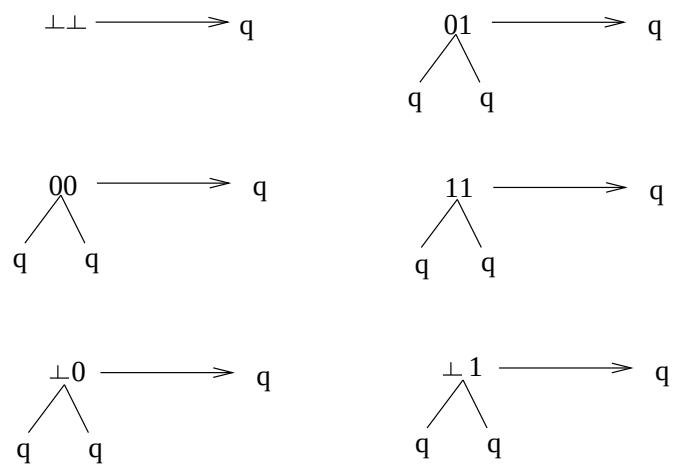


Figure 3.9: The automaton for $X \subseteq Y$

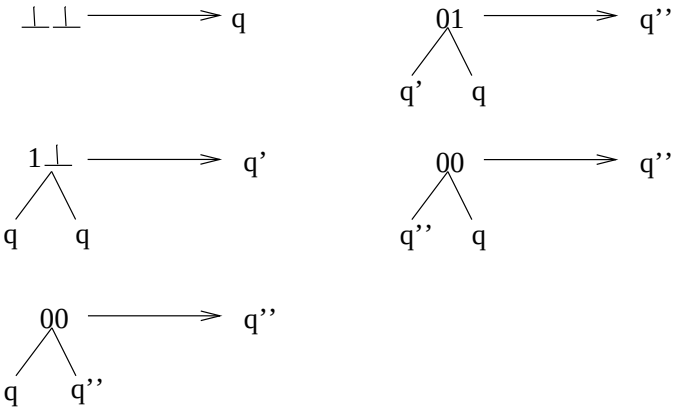


Figure 3.10: The automaton for $X = Y1$

Figure 3.11: The automaton for $X = \epsilon$

- If ϕ is a formula $\neg\phi_1$ then $\mathcal{A}_\phi$ is the automaton accepting the complement of $\mathcal{A}_{\phi_1}$. (See Theorem 5)
- If ϕ is a formula $\exists X.\phi_1$. Assume that X correspond to the ith component. Then $\mathcal{A}_\phi$ is the ith projection of $\mathcal{A}_{\phi_1}$ (see Proposition 12).

□

Example 34. Consider the following formula, with free variables X, Y:

$$\forall x, y.(x \in X \wedge y \in Y) \Rightarrow \neg(x \geq y)$$

We want to compute an automaton which accepts the assignments to X, Y satisfying the formula. First, write the formula as

$$\neg\exists X_1, Y_1.X_1 \subseteq X \wedge Y_1 \subseteq Y \wedge G(X_1, Y_1)$$

where $G(X_1, Y_1)$ expresses that X_1 is a singleton x, Y_1 is a singleton y and $x \geq y$. We can use the definition of $\geq$ as a WS2S formula, or compute directly the automaton, yielding

$$\begin{array}{rcl@{\qquad}rcl}
\bot\bot & \to & q & 11(q,q) & \to & q_2 \\
1\,\bot\,(q,q) & \to & q_1 & 0\,\bot\,(q,q_1) & \to & q_1 \\
0\,\bot\,(q_1,q) & \to & q_1 & 01(q_1,q) & \to & q_2 \\
01(q,q_2) & \to & q_2 & 00(q_2,q) & \to & q_2 \\
00(q,q_2) & \to & q_2 & & &
\end{array}$$

where q_2 is the only final state. Now, using cylindrification, intersection, projection and negation we get the following automaton (intermediate steps yield large automata which would require a full page to be displayed):

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
\bot\bot & \to & q_0 & \bot\,1(q_0,q_0) & \to & q_1 & 1\,\bot\,(q_0,q_0) & \to & q_2 \\
\bot\,0(q_0,q_1) & \to & q_1 & \bot\,0(q_1,q_0) & \to & q_1 & \bot\,0(q_1,q_1) & \to & q_1 \\
0\,\bot\,(q_0,q_2) & \to & q_2 & 0\,\bot\,(q_2,q_0) & \to & q_2 & 0\,\bot\,(q_2,q_2) & \to & q_2 \\
\bot\,1(q_0,q_1) & \to & q_1 & \bot\,1(q_1,q_0) & \to & q_1 & \bot\,1(q_1,q_1) & \to & q_1 \\
1\,\bot\,(q_0,q_2) & \to & q_2 & 1\,\bot\,(q_2,q_0) & \to & q_2 & 1\,\bot\,(q_2,q_2) & \to & q_2 \\
10(q_1,q_0) & \to & q_3 & 10(q_0,q_1) & \to & q_3 & 10(q_1,q_1) & \to & q_3 \\
10(q_1,q_2) & \to & q_3 & 10(q_2,q_1) & \to & q_3 & 10(q_i,q_3) & \to & q_3 \\
10(q_3,q_i) & \to & q_3 & 00(q_i,q_3) & \to & q_3 & 00(q_3,q_i) & \to & q_3
\end{array}$$

where i ranges over $\{0,1,2,3\}$ and q_3 is the only final state.

3.3.6 Recognizable Sets are Definable

We have seen in section 3.3.3 how to represent a term using a tuple of set variables. Now, we use this formula Term on the coding of tuples of terms; if $(t_1, \ldots, t_n) \in T(\mathcal{F})^n$, we write $\overline{[t_1, \ldots, t_n]}$ the $(|\mathcal{F}|+1)^n + 1$-tuple of finite sets which represents it: one set for the positions of $[t_1, \ldots, t_n]$ and one set for each element of the alphabet $(\mathcal{F} \cup \{\bot\})^n$. As it has been seen in section 3.3.3, there is a WSkS formula $\mathrm{Term}([t_1, \ldots, t_n])$ which expresses the image of the coding.

Lemma 3. *Every relation in Rec is definable. More precisely, if $R \in$ Rec there is a formula ϕ such that, for every terms $t_1, \ldots, t_n$, if $(S_1, \ldots, S_m) = \overline{[t_1, \ldots, t_n]}$, then*

$$(S_1, \ldots, S_m) \models_2 \phi \text{ if and only if } (t_1, \ldots, t_n) \in R$$

Proof. Let $\mathcal{A}$ be the automaton which accepts the set of terms $[t_1, \ldots, t_n]$ for $(t_1, \ldots, t_n) \in R$. The terminal alphabet of $\mathcal{A}$ is $\mathcal{F}' = (\mathcal{F} \cup \{\bot\})^n$, the set of states Q, the final states Q_f and the set of transition rules T. Let $\mathcal{F}' = \{f_1, \ldots, f_m\}$ and $Q = \{q_1, \ldots, q_l\}$. The following formula $\phi_{\mathcal{A}}$ (with $m+1$ free variables) defines the set $\{\overline{[t_1, \ldots, t_n]} \mid (t_1, \ldots, t_n) \in R\}$.

$$\begin{array}{ll} \exists Y_{q_1}, \ldots, \exists Y_{q_l}. & \\ & \mathrm{Term}(X, X_{f_1}, \ldots, X_{f_m}) \\ \wedge & \mathrm{Partition}(X, Y_{q_1}, \ldots, Y_{q_l}) \\ \wedge & \bigvee_{q \in Q_f} \epsilon \in Y_q \\ \wedge & \forall x. \bigwedge_{f \in \mathcal{F}'} \bigwedge_{q \in Q} ((x \in X_f \wedge x \in Y_q) \Rightarrow (\bigvee_{f(q_1,\ldots,q_s) \to q \in T} \bigwedge_{i=1}^{s} xi \in Y_{q_i})) \end{array}$$

This formula basically expresses that there is a successful run of the automaton on $[t_1, \ldots, t_n]$: the variables Y_{q_i} correspond to sets of positions which are labeled with q_i by the run. They should be a partition of the set of positions. The root has to be labeled with a final state (the run is successful). Finally, the last line expresses local properties that have to be satisfied by the run: if the sons xi of a position x are labeled with $q_1, ..., q_n$ respectively and x is labeled with symbol f and state q, then there should be a transition $f(q_1, \ldots, q_n) \to q$ in the set of transitions.

We have to prove two inclusions. First assume that $(S, S_1, \ldots, S_m) \models_2 \phi$. Then $(S, S_1, \ldots, S_m) \models \mathrm{Term}(X, X_{f_1}, \ldots, X_{f_m})$, hence there is a term $u \in T(\mathcal{F})'$ whose set of position is S and such that for all i, S_i is the set of positions labeled with f_i. Now, there is a partition $E_{q_1}, \ldots, E_{q_l}$ of S which satisfies

$$S, S_1, \ldots, S_m, E_{q_1}, \ldots, E_{q_l} \models$$
$$\forall x. \bigwedge_{f \in \mathcal{F}'} \bigwedge_{q \in Q} ((x \in X_f \wedge x \in Y_q) \Rightarrow (\bigvee_{f(q_1,\ldots,q_s) \to q \in T} \bigwedge_{i=1}^{s} xi \in Y_{q_i}))$$

This implies that the labeling $E_{q_1}, \ldots, E_{q_l}$ is compatible with the transition rules: it defines a run of the automaton. Finally, the condition that the root ε belongs to E_{q_f} for some final state q_f implies that the run is successful, hence that u is accepted by the automaton.

Conversely, if u is accepted by the automaton, then there is a successful run of $\mathcal{A}$ on u and we can label its positions with states in such a way that this labeling is compatible with the transition rules in $\mathcal{A}$. □

Putting together Lemmas 2 and 3, we can state the following slogan (which is not very precise; the precise statements are given by the lemmas):

Theorem 24. *L is definable if and only if L is in Rec.*

And, as a consequence:

Theorem 25 ([TW68]). *WSkS is decidable.*

Proof. Given a formula ϕ of WSkS, by Lemma 2, we can compute a finite tree automaton which has the same solutions as ϕ. Now, assume that ϕ has no free variable. Then the alphabet of the automaton is empty (or, more precisely, it contains the only constant $\top$ according to what we explained in Section 3.2.4). Finally, the formula is valid iff the constant $\top$ is in the language, *i.e.*iff there is a rule $\top \rightarrow q_f$ for some $q_f \in Q_f$. □

3.3.7 Complexity Issues

We have seen in chapter 1 that, for finite tree automata, emptiness can be decided in linear time (and is PTIME-complete) and that inclusion is EXPTIME-complete. Considering WSkS formulas with a fixed number of quantifiers alternations N, the decision method sketched in the previous section will work in time which is a tower of exponentials, the height of the tower being $O(N)$. This is so because each time we encounter a sequence $\forall X, \exists Y$, the existential quantification corresponds to a projection, which may yield a non-deterministic automaton, even if the input automaton was deterministic. Then the elimination of $\forall X$ requires a determinization (because we have to compute a complement automaton) which requires in general exponential time and exponential space.

Actually, it is not really possible to do much better since, even when $k = 1$, deciding a formula of WSkS requires non-elementary time, as shown in [SM73].

3.3.8 Extensions

There are several extensions of the logic, which we already mentioned: though quantification is restricted to finite sets, we may consider infinite sets as models (this is what is often called *weak second-order monadic logic with k successors* and also written WSkS), or consider quantifications on arbitrary sets (this is the full SkS).

These logics require more sophisticated automata which recognize sets of *infinite* terms. The proof of Theorem 25 carries over these extensions, with the provision that the devices enjoy the required closure and decidability properties. But this becomes much more intricate in the case of infinite terms. Indeed, for infinite terms, it is not possible to design bottom-up tree automata. We have to use a top-down device. But then, as mentioned in chapter 1, we cannot expect to reduce the non-determinism. Now, the closure by complement becomes problematic because the usual way of computing the complement uses reduction of non-determinism as a first step.

It is out of the scope of this book to define and study automata on infinite objects (see [Tho90] instead). Let us simply mention that the closure under complement for *Rabin automata* which work on infinite trees (this result is known as *Rabin's Theorem*) is one of the most difficult results in the field

3.4 Examples of Applications

3.4.1 Terms and Sorts

The most basic example is what is known in the algebraic specification community as *order-sorted signatures* . These signatures are exactly what we called bottom-up tree automata. There are only differences in the syntax. For instance, the following signature:

SORTS:Nat, int
SUBSORTS : Nat $\leq$ int
FUNCTION DECLARATIONS:

$$\begin{array}{lrcl} 0: & & \to & \mathsf{Nat} \\ +: & \mathsf{Nat} \times \mathsf{Nat} & \to & \mathsf{Nat} \\ s: & \mathsf{Nat} & \to & \mathsf{Nat} \\ p: & \mathsf{Nat} & \to & \mathsf{int} \\ +: & \mathsf{int} \times \mathsf{int} & \to & \mathsf{int} \\ abs: & \mathsf{int} & \to & \mathsf{Nat} \\ fact: & \mathsf{Nat} & \to & \mathsf{Nat} \\ \ldots & & & \end{array}$$

is an automaton whose states are Nat, int with an ϵ-transition from Nat to int and each function declaration corresponds to a transition of the automaton. For example $+(\mathsf{Nat}, \mathsf{Nat}) \to \mathsf{Nat}$. The set of *well-formed terms* (as in the algebraic specification terminology) is the set of terms recognized by the automaton in any state.

More general typing systems also correspond to more general automata (as will be seen e.g. in the next chapter).

This correspondence is not surprising; types and sorts are introduced in order to prevent run-time errors by some "abstract interpretation" of the inputs. Tree automata and tree grammars also provide such a symbolic evaluation mechanism. For other applications of tree automata in this direction, see e.g. chapter 5.

From what we have seen in this chapter, we can go beyond simply recognizing the set of well-formed terms. Consider the following *sort constraints* (the alphabet $\mathcal{F}$ of function symbols is given):

The set of *sort expressions* $\mathcal{SE}$ is the least set such that

- $\mathcal{SE}$ contains a finite set of *sort symbols* S, including the two particular symbols $\top_S$ and $\bot_S$.
- If $s_1, s_2 \in \mathcal{SE}$, then $s_1 \vee s_2$, $s_1 \wedge s_2$, $\neg s_1$ are in $\mathcal{SE}$
- If $s_1, \ldots, s_n$ are in $\mathcal{SE}$ and f is a function symbol of arity n, then $f(s_1, \ldots, s_n) \in \mathcal{SE}$.

The atomic formulas are expressions $t \in s$ where $t \in T(\mathcal{F}, \mathcal{X})$ and $s \in \mathcal{SE}$. The formulas are arbitrary first-order formulas built on these atomic formulas.

These formulas are interpreted as follows: we are giving an order-sorted signature (or a tree automaton) whose set of sorts is S. We define the interpretation $[\![\cdot]\!]_S$ of sort expressions as follows:

- if $s \in S$, $[\![s]\!]_S$ is the set of terms in $T(\mathcal{F})$ that are accepted in state s.

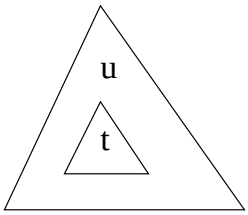


Figure 3.12: u encompasses t

- $[\![\top_S]\!]_S = T(\mathcal{F})$ and $[\![\bot_S]\!]_S = \emptyset$
- $[\![s_1 \vee s_2]\!]_S = [\![s_1]\!]_S \cup [\![s_2]\!]_S$, $[\![s_1 \wedge s_2]\!]_S = [\![s_1]\!]_S \cap [\![s_2]\!]_S$, $[\![\neg s]\!]_S = T(\mathcal{F}) \setminus [\![s]\!]_S$
- $[\![f(s_1, \ldots, s_n)]\!]_S = \{f(t_1, \ldots, t_n) \mid t_1 \in [\![s_1]\!]_S, \ldots t_n \in [\![s_n]\!]_S\}$

An assignment σ, mapping variables to terms in $T(\mathcal{F})$, *satisfies* $t \in s$ (we also say that σ *is a solution* of $t \in s$) if $t\sigma \in [\![s]\!]_S$. Solutions of arbitrary formulas are defined as expected. Then

Theorem 26. *Sort constraints are decidable.*

The decision technique is based on automata computations, following the closure properties of $Rec_\times$ and a decomposition lemma for constraints of the form $f(t_1, \ldots, t_n) \in s$.

More results and applications of sort constraints are discussed in the bibliographic notes.

3.4.2 The Encompassment Theory for Linear Terms

Definition 9. *If $t \in T(\mathcal{F}, \mathcal{X})$ and $u \in T(\mathcal{F})$,* u encompasses t *if there is a substitution σ such that $t\sigma$ is a subterm of u. (See Figure 3.12.) This binary relation is denoted $t \trianglelefteq u$ or, seen as a unary relation on ground terms parametrized by t: $\trianglelefteq_t(u)$.*

Encompassment plays an important role in rewriting: a term t is reducible by a term rewriting system R if and only if t encompasses at least one left hand side of a rule.

The relationship with tree automata is given by the proposition:

Proposition 15. *If t is linear, then the set of terms that encompass t is recognized by a NFTA of size $O(|t|)$.*

Proof. To each non-variable subterm v of t we associate a state q_v. In addition we have a state $q_\top$. The only final state is q_t. The transition rules are:

- $f(q_\top, \ldots, q_\top) \to q_\top$ for all function symbols.
- $f(q_{t_1}, \ldots, q_{t_n}) \to q_{f(t_1,\ldots,t_n)}$ if $f(t_1, \ldots, t_n)$ is a subterm of t and q_{t_i} is actually $q_\top$ is t_i is a variable.

- $f(q_\top \ldots, q_\top, q_t, q_\top, \ldots, q_\top) \to q_t$ for all function symbols f whose arity is at least 1.

The proof that this automaton indeed recognizes the set of terms that encompass t is left to the reader. □

Note that the automaton may be non deterministic. With a slight modification, if u is a linear term, we can construct in linear time an automaton which accepts the set of instances of u (this is also left as an exercise in chapter 1, exercise 8).

Corollary 4. *If $\mathcal{R}$ is a term rewriting system whose all left members are linear, then the set of reducible terms in $T(\mathcal{F})$, as well as the set NF of irreducible terms in $T(\mathcal{F})$ are recognized by a finite tree automaton.*

Proof. This is a consequence of Theorem 5. □

The *theory of reducibility* associated with a set of term $S \subseteq T(\mathcal{F}, \mathcal{X})$ is the set of first-order formulas built on the unary predicate symbols E_t, $t \in S$ and interpreted as the set of terms encompassing t.

Theorem 27. *The reducibility theory associated with a set of linear terms is decidable.*

Proof. By proposition 15, the set of solutions of an atomic formula is recognizable, hence definable in WSkS by Lemma 3. Hence, any first-order formula built on these atomic predicate symbols can be translated into a (second-order) formula of WSkS which has the same models (up to the coding of terms into tuples of sets). Then, by Theorem 25, the reducibility theory associated with a set of linear terms is decidable. □

Note however that we do not use here the full power of WSkS. Actually, the solutions of a Boolean combination of atomic formulas are in $Rec_\times$. So, we cannot apply the complexity results for WSkS here. (In fact, the complexity of the reducibility theory is unknown so far).

Let us simply show an example of an interesting property of rewrite systems which can be expressed in this theory.

Definition 10. *Given a term rewriting system R, a term t is* ground reducible *if, for every ground substitution σ, $t\sigma$ is reducible by R.*

Note that a term might be irreducible and still ground reducible. For instance consider the alphabet $\mathcal{F} = \{0, s\}$ and the rewrite system $R = \{s(s(0)) \to 0\}$. Then the term $s(s(x))$ is irreducible by R, but all its ground instances are reducible.

It turns out that ground reducibility of t is expressible in the encompassment theory by the formula:

$$\forall x.(\ \trianglelefteq_t(x) \Rightarrow \bigvee_{i=1}^{n} \trianglelefteq_{l_i}(x))$$

Where $l_1, \ldots, l_n$ are the left hand sides of the rewrite system. By Theorem 27, if $t, l_1, \ldots, l_n$ are linear, then ground reducibility is decidable. Actually, it has been shown that this problem is EXPTIME-complete, but is beyond the scope of this book to give the proof.

3.4.3 The First-order Theory of a Reduction Relation: the Case Where no Variables are Shared

We consider again an application of tree automata to decision problem in logic and term rewriting.

Consider the following logical theory. Let $\mathcal{L}$ be the set of all first-order formulas using no function symbols and a single binary predicate symbol $\rightarrow$.

Given a rewrite system $\mathcal{R}$, interpreting $\rightarrow$ as $\underset{\mathcal{R}}{\longrightarrow}$, yields the *theory of one step rewriting*; interpreting $\rightarrow$ as $\overset{*}{\underset{\mathcal{R}}{\longrightarrow}}$ yields the *theory of rewriting.*

Both theories are undecidable for arbitrary $\mathcal{R}$. They become however decidable if we restrict our attention to term rewriting systems in which each variable occurs at most once. Basically, the reason is given by the following:

Proposition 16. *If $\mathcal{R}$ is a linear rewrite system such that left and right members of the rules do not share variables, then $\overset{*}{\underset{\mathcal{R}}{\longrightarrow}}$ is recognized by a GTT.*

Proof. As in the proof of Proposition 15, we can construct in linear time a (non-deterministic) automaton which accepts the set of instances of a linear term. For each rule $l_i \rightarrow r_i$ we can construct a pair $(\mathcal{A}_i, \mathcal{A}'_i)$ of automata which respectively recognize the set of instances of l_i and the set of instances of r_i. Assume that the sets of states of the $\mathcal{A}_i$s are pairwise disjoint and that each $\mathcal{A}_i$ has a single final state q_f^i. We may assume a similar property for the $\mathcal{A}'_i$s: they do not share states and for each i, the only common state between $\mathcal{A}_i$ and $\mathcal{A}'_i$ is q_f^i (the final state for both of them). Then $\mathcal{A}$ (resp. $\mathcal{A}'$) is the union of the $\mathcal{A}_i$s: the states are the union of all sets of states of the $\mathcal{A}_i$s (resp. $\mathcal{A}'_i$s), transitions and final states are also unions of the transitions and final states of each individual automaton.

We claim that $(\mathcal{A}, \mathcal{A}')$ defines a GTT whose closure by iteration $(\mathcal{A}_*, \mathcal{A}'_*)$ (which is again a GTT according to Theorem 23) accepting $\overset{*}{\underset{\mathcal{R}}{\longrightarrow}}$. For, assume first that $u \xrightarrow[l_i \rightarrow r_i]{p} v$. Then $u|_p$ is an instance $l_i\sigma$ of l_i, hence is accepted in state q_f^i. $v|_p$ is an instance $r_i\theta$ of r_i, hence accepted in state q_f^i. Now, $v = u[r_i\theta]_p$, hence (u, v) is accepted by the GTT $(\mathcal{A}, \mathcal{A}')$. It follows that if $u \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} v$, (u, v) is accepted by $(\mathcal{A}_*, \mathcal{A}'_*)$.

Conversely, assume that (u, v) is accepted by $(\mathcal{A}, \mathcal{A}')$, then

$$u \overset{*}{\underset{\mathcal{A}}{\longrightarrow}} C[q_1, \ldots, q_n]_{p_1,\ldots,p_n} \overset{*}{\underset{\mathcal{A}'}{\longleftarrow}} v$$

Moreover, each q_i is some state q_f^j, which, by definition, implies that $u|_{p_i}$ is an instance of l_j and $v|_{p_i}$ is an instance of r_j. Now, *since l_j and r_j do not share variables*, for each i, $u|_{p_i} \underset{\mathcal{R}}{\longrightarrow} v|_{p_i}$. Which implies that $u \overset{*}{\underset{\mathcal{R}}{\longrightarrow}} v$. Now, if (u, v) is accepted by $(\mathcal{A}_*, \mathcal{A}'_*)$, u can be rewritten in v by the transitive closure of $\overset{*}{\underset{\mathcal{R}}{\longrightarrow}}$, which is $\overset{*}{\underset{\mathcal{R}}{\longrightarrow}}$ itself.

□

Theorem 28. *If $\mathcal{R}$ is a linear term rewriting system such that left and right members of the rules do not share variables, then the first-order theory of rewriting is decidable.*

Proof. By Proposition 16, $\xrightarrow[\mathcal{R}]{*}$ is recognized by a GTT. From Proposition 9, $\xrightarrow[\mathcal{R}]{*}$ is in *Rec*. By Lemma 3, there is a WSkS formula whose solutions are exactly the pairs (s,t) such that $s \xrightarrow[\mathcal{R}]{*} t$. Finally, by Theorem 25, the first-order theory of $\xrightarrow[\mathcal{R}]{*}$ is decidable. □

3.4.4 Reduction Strategies

So far, we gave examples of first-order theories (or *constraint systems*) which can be decided using tree automata techniques. Other examples will be given in the next two chapters. We give here another example of application in a different spirit: we are going to show how to decide the existence (and compute) "optimal reduction strategies" in term rewriting systems. Informally, a reduction sequence is optimal when every redex which is contracted along this sequence has to be contracted in any reduction sequence yielding a normal form. For example, if we consider the rewrite system $\{x \vee \top \to \top; \top \vee x \to \top\}$, there is no optimal sequential reduction strategy in the above sense since, given an expression $e_1 \vee e_2$, where e_1 and e_2 are unevaluated, the strategy should specify which of e_1 or e_2 has to be evaluated first. However, if we start with e_1, then maybe e_2 will reduce to $\top$ and the evaluation step on e_1 was unnecessary. Symmetrically, evaluating e_2 first may lead to unnecessary computations. An interesting question is to give sufficient criteria for a rewrite system to admit optimal strategies and, in case there is such a strategy, give it explicitly.

The formalization of these notions was given by Huet and Lévy in [HL91] who introduce the notion of *sequentiality*. We give briefly a summary of (part of) their definitions.

$\mathcal{F}$ is a fixed alphabet of function symbols and $\mathcal{F}_\Omega = \mathcal{F} \cup \{\Omega\}$ is the alphabet $\mathcal{F}$ enriched with a new constant Ω (whose intended meaning is "unevaluated term").

We define on $T(\mathcal{F}_\Omega)$ the relation "less evaluated than" as:

> $u \sqsubseteq v$ if and only if either $u = \Omega$ or else $u = f(u_1, \ldots, u_n)$, $v = f(v_1, \ldots, v_n)$ and for all i, $u_i \sqsubseteq v_i$

Definition 11. *A predicate P on $T(\mathcal{F}_\Omega)$ is* monotonic *if $u \in P$ and $u \sqsubseteq v$ implies $v \in P$.*

For example, a monotonic predicate of interest for rewriting is the predicate $N_\mathcal{R}$: $t \in N_\mathcal{R}$ if and only if there is a term $u \in T(\mathcal{F})$ such that u is irreducible by $\mathcal{R}$ and $t \xrightarrow[\mathcal{R}]{*} u$.

Definition 12. *Let P be a monotonic predicate on $T(\mathcal{F}_\Omega)$. If $\mathcal{R}$ is a term rewriting system and $t \in T(\mathcal{F}_\Omega)$, a position p of Ω in t is an* index *for P if for all terms $u \in T(\mathcal{F}_\Omega)$ such that $t \sqsubseteq u$ and $u \in P$, then $u|_p \neq \Omega$*

In other words: it is necessary to evaluate t at position p in order to have the predicate P satisfied.

Example 35. Let $\mathcal{R} = \{f(g(x), y) \to g(f(x, y));\ f(a, x) \to a; b \to g(b)\}$. Then 1 is an index of $f(\Omega, \Omega)$ for $N_\mathcal{R}$: any reduction yielding a normal form

without Ω will have to evaluate the term at position 1. More formally, every term $f(t_1, t_2)$ which can be reduced to a term in $T(\mathcal{F})$ in normal form satisfies $t_1 \neq \Omega$. On the contrary, 2 is not an index of $f(\Omega, \Omega)$ since $f(a, \Omega) \xrightarrow[R]{*} a$.

Definition 13. *A monotonic predicate P is* sequential *if every term t such that:*

- $t \notin P$
- *there is $u \in T(\mathcal{F})$, $t \sqsubseteq u$ and $u \in P$*

has an index for P.

If $N_{\mathcal{R}}$ is sequential, the reduction strategy consisting of reducing an index is optimal for non-overlapping and left linear rewrite systems.

Now, the relationship with tree automata is given by the following result:

Theorem 29. *If P is definable in WSkS, then the sequentiality of P is expressible as a WSkS formula.*

The proof of this result is quite easy: it suffices to translate directly the definitions.

For instance, if $\mathcal{R}$ is a rewrite system whose left and right members do not share variables, then $N_{\mathcal{R}}$ is recognizable (by Propositions 16 and 9), hence definable in WSkS by Lemma 3 and the sequentiality of $N_{\mathcal{R}}$ is decidable by Theorem 29.

In general, the sequentiality of $N_{\mathcal{R}}$ is undecidable. However, one can notice that, if $\mathcal{R}$ and $\mathcal{R}'$ are two rewrite systems such that $\xrightarrow[\mathcal{R}]{} \subseteq \xrightarrow[\mathcal{R}']{}$, then a position p which is an index for $\mathcal{R}'$ is also an index for $\mathcal{R}$. (And thus, $\mathcal{R}$ is sequential whenever $\mathcal{R}'$ is sequential).

For instance, we may approximate the term rewriting system, replacing all right hand sides by a new variable which does not occur in the corresponding left member. Let $\mathcal{R}?$ be this approximation and $N?$ be the predicate $N_{\mathcal{R}?}$. (This is the approximation considered by Huet and Lévy).

Another, refined, approximation consists in renaming all variables of the right hand sides of the rules in such a way that all right hand sides become linear and do not share variables with their left hand sides. Let $\mathcal{R}'$ be such an approximation of $\mathcal{R}$. The predicate $N_{\mathcal{R}'}$ is written NV.

Proposition 17. *If $\mathcal{R}$ is left linear, then the predicates $N?$ and NV are definable in WSkS and their sequentiality is decidable.*

Proof. The approximations $\mathcal{R}?$ and $\mathcal{R}'$ satisfy the hypotheses of Proposition 16 and hence $\xrightarrow[\mathcal{R}?]{*}$ and $\xrightarrow[\mathcal{R}']{*}$ are recognized by GTTs. On the other hand, the set of terms in normal form for a left linear rewrite system is recognized by a finite tree automaton (see Corollary 4). By Proposition 9 and Lemma 3, all these predicates are definable in WSkS. Then $N?$ and NV are also definable in WSkS. For instance for NV:

$$NV(t) \stackrel{\text{def}}{=} \exists u.t \xrightarrow[\mathcal{R}']{*} u \wedge u \in NF$$

Then, by Theorem 29, the sequentiality of $N?$ and NV are definable in WSkS and by Theorem 25 they are decidable. □

3.4.5 Application to Rigid E-unification

Given a finite (universally quantified) set of equations E, the classical problem of *E-unification* is, given an equation $s = t$, find substitutions σ such that $E \models s\sigma = t\sigma$. The associated decision problem is to decide whether such a substitution exists. This problem is in general unsolvable: there are decision procedures for restricted classes of axioms E.

The *simultaneous rigid E-unification problem* is slightly different: we are still giving E and a finite set of equations $s_i = t_i$ and the question is to find a substitution σ such that

$$\models (\bigwedge_{e \in E} e\sigma) \Rightarrow (\bigwedge_{i=1}^{n} s_i\sigma = t_i\sigma)$$

The associated decision problem is to decide the existence of such a substitution.

The relevance of such questions to automated deduction is very briefly described in the bibliographic notes. We want here to show how tree automata help in this decision problem.

Simultaneous rigid E-unification is undecidable in general. However, for the one variable case, we have:

Theorem 30. *The simultaneous rigid E-unification problem with one variable is EXPTIME-complete.*

The EXPTIME membership is a direct consequence of the following lemma, together with closure and decision properties for recognizable tree languages. The EXPTIME-hardness is obtained by reduction the intersection non-emptiness problem, see Theorem 12).

Lemma 4. *The solutions of a rigid E-unification problem with one variable are recognizable by a finite tree automaton.*

Proof. (sketch) Assume that we have a single equation $s = t$. Let x be the only variable occurring in $E, s = t$ and $\hat{E}$ be the set E in which x is considered as a constant. Let R be a canonical ground rewrite system (see e.g. [DJ90]) associated with $\hat{E}$ (and for which x is minimal). We define v as x if s and t have the same normal form *w.r.t.R* and as the normal form of $x\sigma$ *w.r.t.R* otherwise.

Assume $E\sigma \models s\sigma = t\sigma$. If $v \not\equiv x$, we have $\hat{E} \cup \{x = v\} \models x = x\sigma$. Hence $\hat{E} \cup \{x = v\} \models s = t$ in any case. Conversely, assume that $\hat{E} \cup \{x = v\} \models s = t$. Then $\hat{E} \cup \{x = x\sigma\} \models s = t$, hence $E\sigma \models s\sigma = t\sigma$.

Now, assume $v \not\equiv x$. Then either there is a subterm u of an equation in $\hat{E}$ such that $\hat{E} \models u = v$ or else $R_1 = R \cup \{v \to x\}$ is canonical. In this case, from $\hat{E} \cup \{v = x\} \models s = t$, we deduce that either $\hat{E} \models s = t$ (and $v \equiv x$) or there is a subterm u of s, t such that $\hat{E} \models v = u$. we can conclude that, in all cases, there is a subterm u of $E \cup \{s = t\}$ such that $\hat{E} \models u = v$.

To summarize, σ is such that $E\sigma \models s\sigma = t\sigma$ iff there is a subterm u of $E \cup \{s = t\}$ such that $\hat{E} \models u = x\sigma$ and $\hat{E} \cup \{u = x\} \models s = t$.

If we let T be the set of subterms u of $E \cup \{s = t\}$ such that $\hat{E} \cup \{u = x\} \models s = t$, then T is finite (and computable in polynomial time). The set of solutions is then $\xrightarrow[R^{-1}]{*}(T)$, which is a recognizable set of trees, thanks to Proposition 16.

□

Further comments and references are given in the bibliographic notes.

3.4.6 Application to Higher-order Matching

We give here a last application (but the list is not closed!), in the typed lambda-calculus.

To be self-contained, let us first recall some basic definitions in typed lambda calculus.

The set of *types* of the simply typed lambda calculus is the least set containing the constant o (basic type) and such that $\tau \to \tau'$ is a type whenever τ and τ' are types.

Using the so-called Curryfication, any type $\tau \to (\tau' \to \tau'')$ is written $\tau, \tau' \to \tau''$. In this way all non-basic types are of the form $\tau_1, \ldots, \tau_n \to o$ with intuitive meaning that this is the type of functions taking n arguments of respective types $\tau_1, \ldots, \tau_n$ and whose result is a basic type o.

The ***order*** of a type τ is defined by:

- $O(o) = 1$
- $O(\tau_1, \ldots, \tau_n \to o) = 1 + \max\{O(\tau_1), \ldots, O(\tau_n)\}$

Given, for each type τ a set of variables $\mathcal{X}_\tau$ of type τ and a set C_τ of constants of type τ, the set of ***terms*** (of the simply typed lambda calculus) is the least set Λ such that:

- $x \in \mathcal{X}_\tau$ is a term of type τ
- $c \in C_\tau$ is a term of type τ
- If $x_1 \in \mathcal{X}_{\tau_1}, \ldots, x_n \in \mathcal{X}_{\tau_n}$ and t is a term of type τ, then $\lambda x_1, \ldots x_n : t$ is a term of type $\tau_1, \ldots, \tau_n \to \tau$
- If t is a term of type $\tau_1, \ldots, \tau_n \to \tau$ and $t_1, \ldots, t_n$ are terms of respective types $\tau_1, \ldots, \tau_n$, then $t(t_1, \ldots, t_n)$ is a term of type τ.

The ***order*** of a term t is the order of its type $\tau(t)$.

The set of ***free variables*** $\mathcal{V}ar(t)$ of a term t is defined by:

- $\mathcal{V}ar(x) = \{x\}$ if x is a variable
- $\mathcal{V}ar(c) = \emptyset$ if c is a constant
- $\mathcal{V}ar(\lambda x_1, \ldots, x_n : t) = \mathcal{V}ar(t) \setminus \{x_1, \ldots, x_n\}$
- $\mathcal{V}ar(t(u_1, \ldots, u_n)) = \mathcal{V}ar(t) \cup \mathcal{V}ar(u_1) \cup \ldots \cup \mathcal{V}ar(u_n)$

Terms are always assumed to be in ***η-long form***, *i.e.*they are assumed to be in normal form with respect to the rule:

$$(\eta) \quad t \to \lambda x_1, \ldots, x_n.t(x_1, \ldots, x_n) \quad \begin{array}{l}\text{if } \tau(t) = \tau_1, \ldots \tau_n \to \tau \\ \text{and } x_i \in \mathcal{X}_{\tau_i} \setminus \mathcal{V}ar(t) \text{ for all } i\end{array}$$

We define the ***α-equivalence*** $=_\alpha$ on Λ as the least congruence relation such that: $\lambda x_1, \ldots, x_n : t =_\alpha \lambda x'_1 \ldots, x'_n : t'$ when

- t' is the term obtained from t by substituting for every index i, every free occurrence of x_i with x'_i.

- There is no subterm of t in which, for some index i, both x_i and x'_i occur free.

In the following, we consider only lambda terms modulo α-equivalence. Then we may assume that, in any term, any variable is bounded at most once and free variables do not have bounded occurrences.

The ***β-reduction*** is defined on Λ as the least binary relation $\underset{\beta}{\longrightarrow}$ such that

- $\lambda x_1, \ldots, x_n : t(t_1, \ldots, t_n) \underset{\beta}{\longrightarrow} t\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$
- for every context C, $C[t] \underset{\beta}{\longrightarrow} C[u]$ whenever $t \underset{\beta}{\longrightarrow} u$

It is well-known that $\beta\eta$-reduction is terminating and confluent on Λ and, for every term $t \in \Lambda$, we let $t\downarrow$ be the unique normal form of t.

A ***matching problem*** is an equation $s = t$ where $s, t \in \Lambda$ and $\mathcal{V}ar(t) = \emptyset$. A ***solution*** of a matching problem is a substitution σ of the free variables of t such that $s\sigma\downarrow = t\downarrow$.

Whether or not the matching problem is decidable is an open question at the time we write this book. However, it can be decided when every free variable occurring in s is of order less or equal to 4. We sketch here how tree automata may help in this matter. We will consider only two special cases here, leaving the general case as well as details of the proofs as exercises (see also bibliographic notes).

First consider a problem

$$x(s_1, \ldots, s_n) = t \tag{1}$$

where x is a third order variable and $s_1, \ldots, s_n, t$ are terms without free variables.

The first result states that the set of solutions is recognizable by a $\Box$-automaton. $\Box$-automata are a slight extension of finite tree automata: we assume here that the alphabet contains a special symbol $\Box$. Then a term u is accepted by a $\Box$-automaton $\mathcal{A}$ if and only if there is a term v which is accepted (in the usual sense) by $\mathcal{A}$ and such that u is obtained from v by replacing each occurrence of the symbol $\Box$ with a term (of appropriate type). Note that two distinct occurrences of $\Box$ need not to be replaced with the same term.

We consider the automaton $\mathcal{A}_{s_1,\ldots,s_n,t}$ defined by: $\mathcal{F}$ consists of all symbols occurring in t plus the variable symbols $x_1, \ldots, x_n$ whose types are respectively the types of $s_1, \ldots, s_n$ and the constant $\Box$.

The set of states Q consists of all subterms of t, which we write q_u (instead of u) and a state $q_\Box$. In addition, we have the final state q_f.

The transition rules Δ consist in

- The rules
$$f(q_{t_1}, \ldots, q_{t_n}) \rightarrow q_{f(t_1,\ldots,t_n)}$$
each time $q_{f(t_1,\ldots,t_n)} \in Q$
- For $i = 1, \ldots, n$, the rules
$$x_i(q_{t_1}, \ldots, q_{t_n}) \rightarrow q_u$$
where u is a subterm of t such that $s_i(t_1, \ldots, t_n)\downarrow = u$ and $t_j = \Box$ whenever $s_i(t_1, \ldots, t_{j-1}, \Box, t_{j+1}, \ldots, t_n)\downarrow = u$.

- the rule $\lambda x_1, \ldots, \lambda x_n.q_t \to q_f$

Theorem 31. *The set of solutions of (1) (up to α-conversion) is accepted by the $\Box$-automaton $\mathcal{A}_{s_1,\ldots,s_n,t}$.*

More about this result, its proof and its extension to fourth order will be given in the exercises (see also bibliographic notes). Let us simply give an example.

Example 36. Let us consider the interpolation equation

$$x(\lambda y_1 \lambda y_2.y_1, \lambda y_3.f(y_3, y_3)) = f(a, a)$$

where y_1, y_2 are assumed to be of base type o. Then $\mathcal{F} = \{a, f, x_1, x_2, \Box_o\}$. $Q = \{q_a, q_{f(a,a)}, q_{\Box_o}, q_f\}$ and the rules of the automaton are:

$$\begin{array}{rcl@{\qquad}rcl} a & \to & q_a & f(q_a, q_a) & \to & q_{f(a,a)} \\ \Box_o & \to & q_{\Box_o} & x_1(q_a, q_{\Box_o}) & \to & q_a \\ x_1(q_{f(a,a)}, q_{\Box_o}) & \to & q_{f(a,a)} & x_2(q_a) & \to & q_{f(a,a)} \\ \lambda x_1 \lambda x_2.q_{f(a,a)} & \to & q_f & & & \end{array}$$

For instance the term $\lambda x_1 \lambda x_2.x_1(x_2(x_1(x_1(a, \Box_o), \Box_o)), \Box_o)$ is accepted by the automaton:

$$\begin{array}{rcl} \lambda x_1 \lambda x_2.x_1(x_2(x_1(x_1(a, \Box_o), \Box_o)), \Box_o) & \xrightarrow[\mathcal{A}]{*} & \lambda x_1 \lambda x_2.x_1(x_2(x_1(x_1(q_a, q_{\Box_o}), q_{\Box_o})), q_{\Box_o}) \\ & \xrightarrow[\mathcal{A}]{} & \lambda x_1 \lambda x_2.x_1(x_2(x_1(q_a, q_{\Box_o})), q_{\Box_o}) \\ & \xrightarrow[\mathcal{A}]{} & \lambda x_1 \lambda x_2.x_1(x_2(q_a), q_{\Box_o}) \\ & \xrightarrow[\mathcal{A}]{} & \lambda x_1 \lambda x_2.x_1(q_{f(a,a)}, q_{\Box_o}) \\ & \xrightarrow[\mathcal{A}]{} & \lambda x_1 \lambda x_2.q_{f(a,a)} \\ & \xrightarrow[\mathcal{A}]{} & q_f \end{array}$$

And indeed, for every terms t_1, t_2, t_3, $\lambda x_1 \lambda x_2.x_1(x_2(x_1(x_1(a, t_1), t_2)), t_3)$ is a solution of the interpolation problem.

3.5 Exercises

Exercise 31. Let $\mathcal{F}$ be the alphabet consisting of finitely many unary function symbols $a_1, \ldots, a_n$ and a constant 0.

1. Show that the set S of pairs (of words) $\{(a_1^n(a_1(a_2(a_2^p(0)))), a_1^n(a_2^p(0))) \mid n, p \in \mathbb{N}\}$ is in *Rec*. Show that S^* is not in *Rec*, hence that *Rec* is not closed under transitive closure.
2. More generally, show that, for any finite rewrite system $\mathcal{R}$ (on the alphabet $\mathcal{F}$!), the reduction relation $\xrightarrow[\mathcal{R}]{}$ is in *Rec*.
3. Is there any hope to design a class of tree languages which contains *Rec*, which is closed by all Boolean operations and by transitive closure and for which emptiness is decidable? Why?

Exercise 32. Show that the set of pairs $\{(t, f(t,t')) \mid t,t' \in T(\mathcal{F})\}$ is not in *Rec*.

Exercise 33. Show that if a binary relation is recognized by a GTT, then its inverse is also recognized by a GTT.

Exercise 34. Give an example of two relations that are recognized by GTTs and whose union is not recognized by any GTT.

Similarly, show that the class of relations recognized by a GTT is not closed by complement. Is the class closed by intersection?

Exercise 35. Give an example of a n-ary relation such that its ith projection followed by its ith cylindrification does not give back the original relation. On the contrary, show that ith cylindrification followed by ith projection gives back the original relation.

Exercise 36. About *Rec* and bounded delay relations. We assume that $\mathcal{F}$ only contains unary function symbols and a constant, *i.e.*we consider words rather than trees and we write $u = a_1 \dots a_n$ instead of $u = a_1(\dots(a_n(0))\dots)$. Similarly, $u \cdot v$ corresponds to the usual concatenation of words.

A binary relation R on $T(\mathcal{F})$ is called a *bounded delay relation* if and only if

$$\exists k / \forall (u,v) \in R, \ \big| \ |u| - |v| \ \big| \leq k$$

R preserves the length if and only if

$$\forall (u,v) \in R, \ \ |u| = |v|$$

If A, B are two binary relations, we write $A \cdot B$ (or simply AB) the relation

$$A \cdot B \stackrel{\text{def}}{=} \{(u,v) / \exists (u_1,v_1) \in A, (u_2,v_2) \in B u = u_1.u_2, v = v_1.v_2\}$$

Similarly, we write (in this exercise!)

$$A^* = \{(u,v) / \exists (u_1,v_1) \in A, \dots, (u_n,v_n) \in A, u = u_1 \dots u_n, v = v_1 \dots v_n\}$$

1. Given $A, B \in Rec$, is $A \cdot B$ necessary in *Rec*? is A^* necessary in *Rec*? Why?
2. Show that if $A \in Rec$ preserves the length, then $A^* \in Rec$.
3. Show that if $A, B \in Rec$ and A is of bounded delay, then $A \cdot B \in Rec$.
4. The family of *rational relations* is the smallest set of subsets of $T(\mathcal{F})^2$ which contains the finite subsets of $T(\mathcal{F})^2$ and which is closed under union, concatenation $(\cdot)$ and $*$.

   Show that, if A is a bounded delay rational relation, then $A \in Rec$. Is the converse true?

Exercise 37. Let R_0 be the rewrite system $\{x \times 0 \to 0; 0 \times x \to 0\}$ and $\mathcal{F} = \{0, 1, s, \times\}$

1. Construct explicitly the GTT accepting $\xrightarrow[R_0]{*}$.
2. Let $R_1 = R_0 \cup \{x \times 1 \to x\}$. Show that $\xrightarrow[R_1]{*}$ is is not recognized by a GTT.
3. Let $R_2 = R_1 \cup \{1 \times x \to x \times 1\}$. Using a construction similar to the transitive closure of GTTs, show that the set $\{t \in T(\mathcal{F}) \mid \exists u \in T(\mathcal{F}), t \xrightarrow[R_2]{*} u, u \in NF\}$ where NF is the set of terms in normal form for R_2 is recognized by a finite tree automaton.

Exercise 38. (*) More generally, prove that given any rewrite system $\{l_i \to r_i \mid 1 \leq i \leq n\}$ such that

1. for all i, l_i and r_i are linear
2. for all i, if $x \in \mathcal{V}ar(l_i) \cap \mathcal{V}ar(r_i)$, then x occurs at depth at most one in l_i.

the set $\{t \in T(\mathcal{F}) \mid \exists u \in NF, t \xrightarrow[R]{*} u\}$ is recognized by finite tree automaton.

What are the consequences of this result?

(See [Jac96] for details about this results and its applications. Also compare with Exercise 16, question 4.)

Exercise 39. Show that the set of pairs $\{(f(t,t'),t) \mid t,t' \in T(\mathcal{F})\}$ is not definable in WSkS. (See also Exercise 32)

Exercise 40. Show that the set of pairs of words $\{(w,w') \mid l(w) = l(w')\}$, where $l(x)$ is the length of x, is not definable in WSkS.

Exercise 41. Let $\mathcal{F} = \{a_1, \ldots, a_n, 0\}$ where each a_i is unary and 0 is a constant. Consider the following constraint system: terms are built on $\mathcal{F}$, the binary symbols $\cap, \cup$, the unary symbol $\neg$ and set variables. Formulas are conjunctions of inclusion constraints $t \subseteq t'$. The formulas are interpreted by assigning to variables finite subsets of $T(\mathcal{F})$, with the expected meaning for other symbols.

Show that the set of solutions of such constraints is in Rec_2. What can we conclude?

(*) What happens if we remove the condition on the a_i's to be unary?

Exercise 42. Complete the proof of Proposition 13.

Exercise 43. Show that the subterm relation is not definable in WSkS.

Given a term t Write a WSkS formula ϕ_t such that a term $u \models \phi_t$ if and only if t is a subterm of u.

Exercise 44. Define in SkS "X is finite". (Hint: express that every totally ordered subset of X has an upper bound and use König's lemma)

Exercise 45. A tuple $(t_1, \ldots, t_n) \in T(\mathcal{F})^n$ can be represented in several ways as a finite sequence of finite sets. The first one is the encoding given in Section 3.3.6, overlapping the terms and considering one set for each tuple of symbols. A second one consists in having a tuple of sets for each component: one for each function symbol.

Compare the number of free variables which result from both codings when defining an n-ary relation on terms in WSkS. Compare also the definitions of the diagonal Δ using both encodings. How is it possible to translate an encoding into the other one?

Exercise 46. (*) Let $\mathcal{R}$ be a finite rewrite system whose all left and right members are ground.

1. Let Termination(x) be the predicate on $T(\mathcal{F})$ which holds true on t when there is no infinite sequence of reductions starting from t. Show that adding this predicate as an atomic formula in the first-order theory of rewriting, this theory remains decidable for ground rewrite systems.
2. Generalize these results to the case where the left members of $\mathcal{R}$ are linear and the right members are ground.

Exercise 47. The complexity of automata recognizing the set of irreducible ground terms.

1. For each $n \in \mathbb{N}$, give a linear rewrite system $\mathcal{R}_n$ whose size is $O(n)$ and such that the minimal automaton accepting the set of irreducible ground terms has a size $O(2^n)$.

2. Assume that for any two strict subterms s, t of left hand side(s) of $\mathcal{R}$, if s and t are unifiable, then s is an instance of t or t is an instance of s. Show that there is a NFTA $\mathcal{A}$ whose size is linear in $\mathcal{R}$ and which accepts the set of irreducible ground terms.

Exercise 48. Prove Theorem 29.

Exercise 49. The Propositional Linear-time Temporal Logic. The logic **PTL** is defined as follows:

Syntax P is a finite set of *propositional variables*. Each symbol of P is a formula (an atomic formula). If ϕ and ψ are formulas, then the following are formulas:

$$\phi \wedge \psi,\ \phi \vee \psi,\ \phi \rightarrow \psi,\ \neg\phi,\ \phi \mathbf{U} \psi,\ \mathbf{N}\phi,\ \mathbf{L}\phi$$

Semantics Let P^* be the set of words over the alphabet P. A word $w \in P^*$ is identified with the sequence of letters $w(0)w(1)\dots w(|w|-1)$. $w(i..j)$ is the word $w(i)\dots w(j)$. The satisfaction relation is defined by:

- if $p \in P$, $w \models p$ if and only if $w(0) = p$
- The interpretation of logical connectives is the usual one
- $w \models \mathbf{N}\phi$ if and only if $|w| \geq 2$ and $w(1..|w|-1) \models \phi$
- $w \models \mathbf{L}\phi$ if and only if $|w| = 1$
- $w \models \phi \mathbf{U} \psi$ if and only if there is an index $i \in [0..|w|]$ such that for all $j \in [0..i]$, $w(j..|w|-1) \models \phi$ and $w(i..|w|-1) \models \psi$.

Let us recall that the language defined by a formula ϕ is the set of words w such that $w \models \phi$.

1. What it is the language defined by $\mathbf{N}(p_1 \mathbf{U} p_2)$ (with $p_1, p_2 \in P$)?
2. Give **PTL** formulas defining respectively $P^* p_1 P^*$, p_1^*, $(p_1 p_2)^*$.
3. Give a first-order WS1S formula (*i.e.*without second-order quantification and containing only one free second-order variable) which defines the same language as $\mathbf{N}(p_1 \mathbf{U} p_2)$
4. For any **PTL** formula, give a first-order WS1S formula which defines the same language.
5. Conversely, show that any language defined by a first-order WS1S formula is definable by a **PTL** formula.

Exercise 50. About 3rd-order interpolation problems

1. Prove Theorem 31.
2. Show that the size of the automaton $\mathcal{A}_{s_1,\dots,s_n,t}$ is $O(n \times |t|)$
3. Deduce from Exercise 19 that the existence of a solution to a system of interpolation equations of the form $x(s_1, \dots, s_n) = t$ (where x is a third order variable in each equation) is in NP.

Exercise 51. About general third order matching.

1. How is it possible to modify the construction of $\mathcal{A}_{s_1,\dots,s_n,t}$ so as to forbid some symbols of t to occur in the solutions?

2. Consider a third order matching problem $u = t$ where t is in normal form and does not contain any free variable. Let $x_1, \ldots, x_n$ be the free variables of u and $x_i(s_1, \ldots, s_m)$ be the subterm of u at position p. Show that, for every solution σ, either $u[\Box]_p\sigma \downarrow{=_\alpha} t$ or else that $x_i\sigma(s_1\sigma, \ldots, s_m\sigma) \downarrow$ is in the set S_p defined as follows: $v \in S_p$ if and only if there is a subterm t' of t and there are positions $p_1, \ldots, p_k$ of t' and variables $z_1, \ldots, z_k$ which are bound above p in u such that $v = t'[z_1, \ldots, z_k]_{p_1,\ldots,p_k}$.
3. By guessing the results of $x_i\sigma(s_1\sigma, \ldots, s_m\sigma)$ and using the previous exercise, show that general third order matching is in NP.

3.6 Bibliographic Notes

The following bibliographic notes only concern the applications of the usual finite tree automata on finite trees (as defined at this stage of the book). We are pretty sure that there are many missing references and we are pleased to receive more pointers to the litterature.

3.6.1 GTT

GTT were introduced in [DTHL87] where they were used for the decidability of confluence of ground rewrite systems.

3.6.2 Automata and Logic

The development of automata in relation with logic and verification (in the sixties) is reported in [Tra95]. This research program was explained by A. Church himself in 1962 [Chu62].

Milestones of the decidability of monadic second-order logic are the papers [Büc60] [Rab69]. Theorem 25 is proved in [TW68].

3.6.3 Surveys

There are numerous surveys on automata and logic. Let us mention some of them: M.O. Rabin [Rab77] surveys the decidable theories; W. Thomas [Tho90, Tho97] provides an excellent survey of relationships between automata and logic.

3.6.4 Applications of tree automata to constraint solving

Concerning applications of tree automata, the reader is also referred to [Dau94] which reviews a number of applications of Tree automata to rewriting and constraints.

The relation between sorts and tree automata is pointed out in [Com89]. The decidability of arbitrary first-order sort constraints (and actually the first order theory of finite trees with equality and sort constraints) is proved in [CD94].

More general sort constraints involving some second-order terms are studied in [Com98b] with applications to a sort constrained equational logic [Com98a].

Sort constraints are also applied to specifications and automated inductive proofs in [BJ97] where tree automata are used to represent some normal forms sets. They are used in logic programming and automated reasoning [FSVY91,

GMW97], in order to get more efficient procedures for fragments which fall into the scope of tree automata techniques. They are also used in automated deduction in order to increase the expressivity of (counter-)model constructions [Pel97].

Concerning encompassment, M. Dauchet et al gave a more general result (dropping the linearity requirement) in [DCC95]. We will come back to this result in the next chapter.

3.6.5 Application of tree automata to semantic unification

Rigid unification was originally considered by J. Gallier et al. [GRS87] who showed that this is a key notion in extending the matings method to a logic with equality. Several authors worked on this problem and it is out of the scope of this book to give a list of references. Let us simply mention that the result of Section 3.4.5 can be found in [Vea97b]. Further results on application of tree automata to rigid unification can be found in [DGN$^+$98], [GJV98].

Tree automata are also used in solving classical semantic unification problems. See e.g. [LM93] [KFK97] [Uri92]. For instance, in [KFK97], the idea is to capture some loops in the narrowing procedure using tree automata.

3.6.6 Application of tree automata to decision problems in term rewriting

Some of the applications of tree automata to term rewriting follow from the results on encompassment theory. Early works in this area are also mentioned in the bibliographic notes of Chapter 1. The reader is also referred to the survey [GT95].

The first-order theory of the binary (many-steps) reduction relation *w.r.t.*a ground rewrite system has been shown decidable by. M. Dauchet and S. Tison [DT90]. Extensions of the theory, including some function symbols, or other predicate symbols like the parallel rewriting or the termination predicate (Terminate(t) holds if there is no infinite reduction sequence starting from t), or fair termination etc... remain decidable [Tis89]. See also the exercises.

Both the theory of one step and the theory of many steps rewriting are undecidable for arbitrary $\mathcal{R}$ [Tre96].

Reduction strategies for term rewriting have been first studied by Huet and Lévy in 1978 [HL91]. They show here the decidability of *strong sequentiality* for orthogonal rewrite systems. This is based on an approximation of the rewrite system which, roughly, only considers the left sides of the rules. Better approximation, yielding refined criteria were further proposed in [Oya93], [Com95], [Jac96]. The orthogonality requirement has also been replaced with the weaker condition of left linearity. The first relation between tree automata, WSkS and reduction strategies is pointed out in [Com95]. Further studies of call-by-need strategies, which are still based on tree automata, but do not use a detour through monadic second-order logic can be found in [DM97]. For all these works, a key property is the preservation of regularity by (many-steps) rewriting, which was shown for ground systems in [Bra69], for linear systems which do not share variables in [DT90], for shallow systems in [Com95], for right linear monadic rewrite systems [Sal88], for linear semi-monadic rewrite systems

[CG90], also called (with slight differences) growing systems in [Jac96]. Growing systems are the currently most general class for which the preservation of recognizability is known.

As already pointed out, the decidability of the encompassment theory implies the decidability of ground reducibility. There are several papers written along these lines which will be explained in the next chapter.

Finally, approximations of the reachable terms are computed in [Gen97] using tree automata techniques, which implies the decision of some safety properties.

3.6.7 Other applications

The relationship between finite tree automata and higher-order matching is studied in [CJ97b].

Finite tree automata are also used in logic programming [FSVY91], type reconstruction [Tiu92] and automated deduction [GMW97].

For further applications of tree automata in the direction of program verification, see e.g. chapter 5 of this book or e.g. [Jon87].

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Index