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.

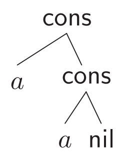
A ***ranked alphabet*** is a couple $(\mathcal{F}, Arity)$ where $\mathcal{F}$ is a finite set and $Arity$ is a mapping from $\mathcal{F}$ into N. The ***arity*** of a symbol $f \in \mathcal{F}$ is $Arity(f)$. The set of symbols of arity p is denoted by $\mathcal{F}_p$. Elements of arity $0, 1, \ldots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

(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 $1, \dots, (e, k)$ 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 $t \neq t'$.

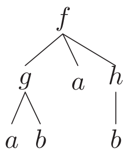
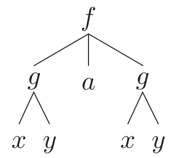
A set of terms F is said to be ***closed*** if it is closed under the subterm ordering, *i.e.*$\forall t \in F$ $(t \trianglerighteq t' \Rightarrow t' \in F)$.

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 $t \in \mathcal{F}_0$,
- $\mathcal{H}eight(t) = 1 + \max(\{\mathcal{H}eight(t_i) \mid i \in \{1, \dots, n\}\})$, $\|t\| = 1 + \sum_{i \in \{1, \dots, n\}} \|t_i\|$ 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 $\{11, 12, 31, 32\}$; $t|_3 = h(b)$; $t[a]_3 = f(g(a, b), a, a)$; $\mathcal{H}eight(t) = 3$; $\mathcal{H}eight(t') = 2$; $\|t\| = 7$; $\|t'\| = 4$.

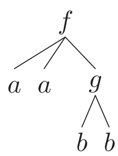
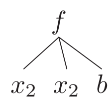
Substitutions

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 2

Regular Grammars and Regular Expressions

2.1 Tree Grammar

In the previous chapter, we have studied tree languages from the acceptor point of view, using tree automata and defining recognizable languages. In this chapter we study languages from the generation point of view, using regular tree grammars and defining regular tree languages. We shall see that the two notions are equivalent and that many properties and concepts on regular word languages smoothly generalize to regular tree languages, and that algebraic characterization of regular languages do exist for tree languages. Actually, this is not surprising since tree languages can be seen as word languages on an infinite alphabet of contexts. We shall show also that the set of derivation trees of a context-free language is a regular tree language.

2.1.1 Definitions

When we write programs, we often have to know how to produce the elements of the data structures that we use. For instance, a definition of the lists of integers in a functional language like ML is similar to the following definition:

$$\begin{array}{l} Nat = 0 \mid s(Nat) \\ List = nil \mid cons(Nat, List) \end{array}$$

This definition is nothing but a tree grammar in disguise, more precisely the set of lists of integers is the tree language generated by the grammar with axiom $List$, non-terminal symbols $List, Nat$, terminal symbols $0, s, nil, cons$ and rules

$$\begin{array}{lcl} Nat & \to & 0 \\ Nat & \to & s(Nat) \\ List & \to & nil \\ List & \to & cons(Nat, List) \end{array}$$

Tree grammars are similar to word grammars except that basic objects are trees, therefore terminals and non-terminals may have an arity greater than 0. More precisely, a ***tree grammar*** $G = (S, N, \mathcal{F}, R)$ is composed of an ***axiom***

S, a set N of ***non-terminal*** symbols with $S \in N$, a set $\mathcal{F}$ of ***terminal*** symbols, a set R of ***production rules*** of the form $\alpha \to \beta$ where α, β are trees of $T(\mathcal{F} \cup N \cup \mathcal{X})$ where $\mathcal{X}$ is a set of dummy variables and α contains at least one non-terminal. Moreover we require that $\mathcal{F} \cap N = \emptyset$, that each element of $N \cup \mathcal{F}$ has a fixed arity and that the arity of the axiom S is 0. In this chapter, we shall concentrate on ***regular tree grammars*** where a regular tree grammar $G = (S, N, \mathcal{F}, R)$ is a tree grammar such that all non-terminal symbols have arity 0 and production rules have the form $A \to \beta$, with A a non-terminal of N and β a tree of $T(\mathcal{F} \cup N)$.

Example 17. The grammar G with axiom $List$, non-terminals $List, Nat$ terminals $0, nil, s(), cons(,)$, rules

$$\begin{array}{l} List \to nil \\ List \to cons(Nat, List) \\ Nat \to 0 \\ Nat \to s(Nat) \end{array}$$

is a regular tree grammar.

A tree grammar is used to build terms from the axiom, using the corresponding ***derivation relation***. Basically the idea is to replace a non-terminal A by the right-hand side α of a rule $A \to \alpha$. More precisely, given a regular tree grammar $G = (S, N, \mathcal{F}, R)$, the derivation relation $\to_G$ associated to G is a relation on pairs of terms of $T(\mathcal{F} \cup N)$ such that $s \to_G t$ if and only if there are a rule $A \to \alpha \in R$ and a context C such that $s = C[A]$ and $t = C[\alpha]$. The ***language generated*** by G, denoted by $L(G)$, is the set of terms of $T(\mathcal{F})$ which can be reached by successive derivations starting from the axiom, *i.e.*$L(G) = \{s \in T_{\mathcal{F}} \mid S \xrightarrow{+}_G s\}$ with $\xrightarrow{+}$ the transitive closure of $\to_G$. We write $\to$ instead of $\to_G$ when the grammar G is clear from the context. A ***regular tree language*** is a language generated by a regular tree grammar.

Example 18. Let G be the grammar of the previous example, then a derivation of $cons(s(0), nil)$ from $List$ is

$$\begin{array}{ll} List \to_G cons(Nat, List) \to_G cons(s(Nat), List) & \to_G cons(s(Nat), nil) \\ & \to_G cons(s(0), nil) \end{array}$$

and the language generated by G is the set of lists of non-negative integers.

From the example, we can see that trees are generated top-down by replacing a leaf by some other term. When A is a non-terminal of a regular tree grammar G, we denote by $L_G(A)$ the language generated by the grammar G' identical to G but with A as axiom. When there is no ambiguity on the grammar referred to, we drop the subscript G. We say that two grammars G and G' are ***equivalent*** when they generate the same language. Grammars can contain useless rules or non-terminals and we want to get rid of these while preserving the generated language. A non-terminal is ***reachable*** if there is a derivation from the axiom containing this non-terminal. A non-terminal A is ***productive*** if $L_G(A)$ is non-empty. A regular tree grammar is ***reduced*** if and only if all its non-terminals are reachable and productive. We have the following result:

Proposition 2. *A regular tree grammar is equivalent to a reduced regular tree grammar.*

Proof. Given a grammar $G = (S, N, \mathcal{F}, R)$, we can compute the set of reachable non-terminals and the set of productive non-terminals using the sequences $(Reach)_n$ and $(Prod)_n$ which are defined in the following way.

$$\begin{array}{ll} Prod_0 = \emptyset & \\ Prod_n = & Prod_{n-1} \\ & \cup \\ & \{A \in N \mid \exists (A \to \alpha) \in R \ s.t. \text{each non-terminal of } \alpha \text{ is in } Prod_{n-1}\} \\ Reach_0 = \{S\} & \\ Reach_n = & Reach_{n-1} \\ & \cup \\ & \{A \in N \mid \exists (A' \to \alpha) \in R \ s.t. A' \in Reach_{n-1} \text{ and } A \text{ occurs in } \alpha\} \end{array}$$

For each sequence, there is an index such that all elements of the sequence with greater index are identical and this element is the set of productive (resp. reachable) non-terminals of G. Each regular tree grammar is equivalent to a reduced tree grammar which is computed by the following cleaning algorithm.

Computation of an equivalent reduced grammar
input: a regular tree grammar $G = (S, N, \mathcal{F}, R)$.

1. Compute the set of productive non-terminals $N_{Prod} = \bigcup_{n \geq 0} Prod_n$ for G and let $G' = (S, N_{Prod}, \mathcal{F}, R')$ where R' is the subset of R involving rules containing only productive non-terminals.
2. Compute the set of reachable non-terminals $N_{Reach} = \bigcup_{n \geq 0} Reach_n$ for G' (not G) and let $G'' = (S, N_{Reach}, \mathcal{F}, R'')$ where R'' is the subset of R' involving rules containing only reachable non-terminals.

output: G''

The equivalence of G, G' and G'' is left to the reader. Moreover each non-terminal A of G'' must appear in a derivation $S \xrightarrow{*}_{G''} C[A] \xrightarrow{*}_{G''} C[s]$ which proves that G'' is reduced. The reader should notice that exchanging the two steps of the computation may result in a grammar which is not reduced (see Exercise 22).

□

Actually, we shall use even simpler grammars, *i.e.****normalized*** regular tree grammar, where the production rules have the form $A \to f(A_1, \ldots, A_n)$ or $A \to a$ where f, a are symbols of $\mathcal{F}$ and $A, A_1, \ldots, A_n$ are non-terminals. The following result shows that this is not a restriction.

Proposition 3. *A regular tree grammar is equivalent to a normalized regular tree grammar.*

Proof. Replace a rule $A \to f(s_1, \dots, s_n)$ by $A \to f(A_1, \dots, A_n)$ with $A_i = s_i$ if $s_i \in N$ otherwise A_i is a new non-terminal. In the last case add the rule $A_i \to s_i$. Iterate this process until one gets a (necessarily equivalent) grammar with rules of the form $A \to f(A_1, \dots, A_n)$ or $A \to a$ or $A_1 \to A_2$. The last rules are replaced by the rules $A_1 \to \alpha$ for all $\alpha \notin N$ such that $A_1 \xrightarrow{+} A_i$ and $A_i \to \alpha \in R$ (these $A_i's$ are easily computed using a transitive closure algorithm). □

From now on, we assume that all grammars are normalized, unless this is stated otherwise explicitly.

2.1.2 Regularity and Recognizabilty

Given some normalized regular tree grammar $G = (S, N, \mathcal{F}, R_G)$, we show how to build a top-down tree automaton which recognizes $L(G)$. We define $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$ by

- $Q = \{q_A \mid A \in N\}$
- $I = \{q_S\}$
- $q_A(f(x_1, \dots, x_n)) \to f(q_{A_1}(x_1), \dots, q_{A_n}(x_n)) \in \Delta$ if and only if $A \to f(A_1, \dots, A_n) \in R_G$.

A standard proof by induction on derivation length yields $L(G) = L(\mathcal{A})$. Therefore we have proved that the languages generated by regular tree grammar are recognizable languages.

The next question to ask is whether recognizable tree languages can be generated by regular tree grammars. If L is a regular tree language, there exists a top-down tree automata $\mathcal{A} = (Q, \mathcal{F}, I, \Delta)$ such that $L = L(\mathcal{A})$. We define $G = (S, N, \mathcal{F}, R_G)$ with S a new symbol, $N = \{A_q \mid q \in Q\}$, $R_G = \{A_q \to f(A_{q_1}, \dots, A_{q_n}) \mid q(f(x_1, \dots, x_n)) \to f(q_1(x_1), \dots, q_n(x_n)) \in R\} \cup \{S \to A_I \mid A_I \in I\}$. A standard proof by induction on derivation length yields $L(G) = L(\mathcal{A})$.

Combining these two properties, we get the equivalence between recognizability and regularity.

Theorem 18. *A tree language is recognizable if and only if it is a regular tree language.*

2.2 Regular Expressions. Kleene's Theorem for Tree Languages

Going back to our example of lists of non-negative integers, we can write the sets defined by the non-terminals Nat and $List$ as follows.

$$Nat = \{0, s(0), s(s(0)), \dots\}$$
$$List = \{nil, cons(\_, nil), cons(\_, cons(\_, nil), \dots\}$$

where _ stands for any element of Nat. There is some regularity in each set which reminds of the regularity obtained with regular word expressions constructed with the union, concatenation and iteration operators. Therefore we can try to use the same idea to denote the sets Nat and $List$. However, since we are dealing with trees and not words, we must put some information to indicate where concatenation and iteration must take place. This is done by using a new symbol which behaves as a constant. Moreover, since we have two independent iterations, the first one for Nat and the second one for $List$, we shall use two different new symbols $\square_1$ and $\square_2$ and a natural extension of regular word expression leads us to denote the sets Nat and $List$ as follows.

$$\begin{aligned}&Nat = s(\square_1)^{*,\square_1} \ ._{\square_1} \ 0\\ &List = nil + cons(\ (s(\square_1)^{*,\square_1} \ ._{\square_1} \ 0)\ , \square_2)^{*,\square_2} \ ._{\square_2} \ nil\end{aligned}$$

Actually the first term nil in the second equality is redundant and a shorter (but slightly less natural) expression yields the same language.

We are going to show that this is a general phenomenon and that we can define a notion of regular expressions for trees and that Kleene's theorem for words can be generalized to trees. Like in the example, we must introduce a particular set of constants $\mathcal{K}$ which are used to indicate the positions where concatenation and iteration take place in trees. This explains why the syntax of regular tree expressions is more cumbersome than the syntax of word regular expressions. These new constants are usually denoted by $\square_1, \square_2, \ldots$. Therefore, in this section, we consider trees constructed on $\mathcal{F} \cup \mathcal{K}$ where $\mathcal{K}$ is a distinguished finite set of symbols of arity 0 disjoint from $\mathcal{F}$.

2.2.1 Substitution and Iteration

First, we have to generalize the notion of substitution to languages, replacing some $\square_i$ by a tree of some language L_i. The main difference with term substitution is that different occurrences of the same constant $\square_i$ can be replaced by different terms of L_i. Given a tree t of $T(\mathcal{F} \cup \mathcal{K})$, $\square_1, \ldots, \square_n$ symbols of $\mathcal{K}$ and $L_1, \ldots, L_n$ languages of $T(\mathcal{F} \cup \mathcal{K})$, the ***tree substitution*** (substitution for short) of $\square_1, \ldots, \square_n$ by $L_1, \ldots, L_n$ in t, denoted by $t\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\}$, is the tree language defined by the following identities.

- $\square_i\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\} = L_i$ for $i = 1, \ldots, n$,
- $a\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\} = \{a\}$ for all $a \in \mathcal{F} \cup \mathcal{K}$ such that arity of a is 0 and $a \neq \square_1, \ldots, a \neq \square_n$,
- $f(s_1, \ldots, s_n)\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\} = \{f(t_1, \ldots, t_n) \mid t_i \in s_i\{\begin{array}{l}\square_1 \leftarrow L_1\\ \ldots\\ \square_n \leftarrow L_n\}\}\end{array}$

Example 19. Let $\mathcal{F} = \{0, nil, s(), cons(,)\}$ and $\mathcal{K} = \{\square_1, \square_2\}$, let

$$t = cons(\square_1, cons(\square_1, \square_2))$$

and let

$$L_1 = \{0, s(0)\}$$

then

$$\begin{array}{rcl} t\{\Box_1 \leftarrow L\} & = & \{cons(0, cons(0, \Box_2)), \\ & & cons(0, cons(s(0), \Box_2)), \\ & & cons(s(0), cons(0, \Box_2)), \\ & & cons(s(0), cons(s(0), \Box_2))\} \end{array}$$

Symbols of $\mathcal{K}$ are mainly used to distinguish places where the substitution must take place, and they are usually not relevant. For instance, if t is a tree on the alphabet $\mathcal{F} \cup \{\Box\}$ and L be a language of trees on the alphabet $\mathcal{F}$, then the trees of $t\{\Box \leftarrow L\}$ don't contain the symbol $\Box$.

The substitution operation generalizes to languages in a straightforward way. When $L, L_1, \ldots, L_n$ are languages of $T(\mathcal{F} \cup \mathcal{K})$ and $\Box_1, \ldots, \Box_n$ are elements of $\mathcal{K}$, we define $L\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}$ to be the set $\bigcup_{t \in L}\{\, t\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}\}$.

Now, we can define the concatenation operation for tree languages. Given L and M two languages of $T_{\mathcal{F} \cup \mathcal{K}}$, and $\Box$ be a element of $\mathcal{K}$, the ***concatenation*** of M to L through $\Box$, denoted by $L \, ._\Box \, M$, is the set of trees obtained by substituting the occurrence of $\Box$ in trees of L by trees of M, *i.e.*$L \, ._\Box \, M = \bigcup_{t \in L}\{t\{\Box \leftarrow M\}\}$.

To define the closure of a language, we must define the sequence of successive iterations. Given L a language of $T(\mathcal{F} \cup \mathcal{K})$ and $\Box$ an element of $\mathcal{K}$, the sequence $L^{n,\Box}$ is defined by the equalities.

- $L^{0,\ \Box} = \{\Box\}$
- $L^{n+1,\ \Box} = L^{n,\ \Box} \cup L \, ._\Box \, L^{n,\ \Box}$

The ***closure*** $L^{*,\Box}$ of L is the union of all $L^{n,\ \Box}$ for non-negative n, *i.e.*, $L^{*,\Box} = \cup_{n \geq 0} L^{n,\Box}$. From the definition, one gets that $\{\Box\} \subseteq L^{*,\Box}$ for any L.

Example 20. Let $\mathcal{F} = \{0, nil, s(), cons(,)\}$, let $L = \{0, cons(0, \Box)\}$ and $M = \{nil, cons(s(0), \Box)\}$, then

$$\begin{array}{rcl} L \, ._\Box \, M & = & \{0, cons(0, nil), cons(0, cons(s(0), \Box))\} \\ L^{*,\Box} & = & \{\Box\} \cup \\ & & \{0, cons(0, \Box)\} \cup \\ & & \{0, cons(0, \Box), cons(0, cons(0, \Box))\} \cup \ldots \end{array}$$

We prove now that the substitution and concatenation operations yield regular languages when they are applied to regular languages.

Proposition 4. *Let L be a regular tree language on $\mathcal{F} \cup \mathcal{K}$, let $L_1, \ldots, L_n$ be regular tree languages on $\mathcal{F} \cup \mathcal{K}$, let $\Box_1, \ldots, \Box_n \in \mathcal{K}$, then $L\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}$ is a regular tree language.*

Proof. Since L is regular, there exists some normalized regular tree grammar $G = (S, N, \mathcal{F} \cup \mathcal{K}, R)$ such that $L = L(G)$, and for each $i = 1, \ldots, n$ there exists a normalized grammar $G_i = (S_i, N_i, \mathcal{F} \cup \mathcal{K}, R_i)$ such that $L_i = L(G_i)$. We can assume that the sets of non-terminals are pairwise disjoint. The idea of the proof is to construct a grammar G' which starts by generating trees like

G but replaces the generation of a symbol $\square_i$ by the generation of a tree of L_i via a branching towards the axiom of G_i. More precisely, we show that $L\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\} = L(G')$ where $G' = (S, N', \mathcal{F} \cup \mathcal{K}, R')$ such that

- $N' = N \cup N_1 \cup \ldots \cup N_n$,
- R' contains the rules of R_i and the rules of R but the rules $A \rightarrow \square_i$ which are replaced by the rules $A \rightarrow S_i$, where S_i is the axiom of L_i.

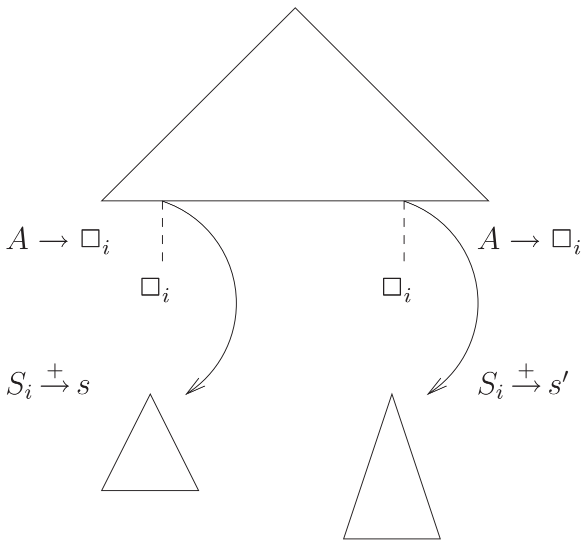


Figure 2.1: Replacement of rules $A \rightarrow \square_i$

A straightforward induction on the height of trees proves that G' generates each tree of $L\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\}$.

The converse is to prove that $L(G') \subseteq L\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\}$. This is achieved by proving the following property by induction on the derivation length.

$$\begin{array}{c} A \xrightarrow{+} s' \text{ where } s' \in T(\mathcal{F} \cup \mathcal{K}) \text{ using the rules of } G' \\ \text{if and only if} \\ \text{there is some } s \text{ such that } A \xrightarrow{+} s \text{ using the rules of } G \text{ and} \\ s' \in s\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\}. \end{array}$$

- base case: $A \rightarrow s$ in one step. Therefore this derivation is a derivation of the grammar G and no $\square_i$ occurs in s, yielding $s \in L\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\}$
- induction step: we assume that the property is true for any terminal and derivation of length less than n. Let A be such that $A \rightarrow s'$ in n steps. This derivation can be decomposed as $A \rightarrow s_1 \xrightarrow{+} s'$. We distinguish several cases depending on the rule used in the derivation $A \rightarrow s_1$.
  - the rule is $A \rightarrow f(A_1, \ldots, A_m)$, therefore $s' = f(t_1, \ldots, t_m)$ and $t_i \in L(A_i)\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\}$, therefore $s' \in L(A)\{\square_1 \leftarrow L_1, \ldots, \square_n \leftarrow L_n\}$,

– the rule is $A \to S_i$, therefore $A \to \Box_i \in R$ and $s' \in L_i$ and $s' \in L(A)\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}$.

– the rule $A \to a$ with $a \in \mathcal{F}$, a of arity 0, $a \neq \Box_1, \ldots, a \neq \Box_n$ are not considered since no further derivation can be done.

$\Box$

The following proposition states that regular languages are stable also under closure.

Proposition 5. *Let L be a regular tree language of $T(\mathcal{F} \cup \mathcal{K})$, let $\Box \in \mathcal{K}$, then $L^{*,\Box}$ is a regular tree language of $T(\mathcal{F} \cup \mathcal{K})$.*

Proof. There exists a normalized regular grammar $G = (S, N, \mathcal{F} \cup \mathcal{K}, R)$ such that $L = L(G)$ and we obtain from G a grammar $G' = (S', N \cup \{S'\}, \mathcal{F} \cup \mathcal{K}, R')$ for $L^{*,\Box}$ by replacing rules leading to $\Box$ such as $A \to \Box$ by rules $A \to S'$ leading to the (new) axiom. Moreover we add the rule $S' \to \Box$ to generate $\{\Box\} = L^{0,\Box}$ and the rule $S' \to S$ to generate $L^{i,\Box}$ for $i > 0$. By construction G' generates the elements of $L^{*,\Box}$.

Conversely a proof by induction on the length on the derivation proves that $L(G') \subseteq L^{*,\Box}$. $\Box$

2.2.2 Regular Expressions and Regular Tree Languages

Now, we can define regular tree expression in the flavor of regular word expression using the $+, ._\Box, ^{*,\Box}$ operators.

Definition 2. *The set Regexp$(\mathcal{F}, \mathcal{K})$ of* ***regular tree expressions*** *on $\mathcal{F}$ and $\mathcal{K}$ is the smallest set such that:*

- *the empty set $\emptyset$ is in Regexp$(\mathcal{F}, \mathcal{K})$*
- *if $a \in \mathcal{F}_0 \cup \mathcal{K}$ is a constant, then $a \in$ Regexp$(\mathcal{F}, \mathcal{K})$,*
- *if $f \in \mathcal{F}_n$ has arity $n > 0$ and $E_1, \ldots, E_n$ are regular expressions of Regexp$(\mathcal{F}, \mathcal{K})$ then $f(E_1, \ldots, E_n)$ is a regular expression of Regexp$(\mathcal{F}, \mathcal{K})$,*
- *if E_1, E_2 are regular expressions of Regexp$(\mathcal{F}, \mathcal{K})$ then $(E_1 + E_2)$ is a regular expression of Regexp$(\mathcal{F}, \mathcal{K})$,*
- *if E_1, E_2 are regular expressions of Regexp$(\mathcal{F}, \mathcal{K})$ and $\Box$ is an element of $\mathcal{K}$ then $E_1 ._\Box E_2$ is a regular expression of Regexp$(\mathcal{F}, \mathcal{K})$,*
- *if E is a regular expression of Regexp$(\mathcal{F}, \mathcal{K})$ and $\Box$ is an element of $\mathcal{K}$ then $E^{*,\Box}$ is a regular expression of Regexp$(\mathcal{F}, \mathcal{K})$.*

Each regular expression E represents a set of terms of $T(\mathcal{F} \cup \mathcal{K})$ which we denote $[\![E]\!]$ and which is formally defined by the following equalities.

- $[\![\emptyset]\!] = \emptyset$,
- $[\![a]\!] = \{a\}$ for $a \in \mathcal{F}_0 \cup \mathcal{K}$,
- $[\![f(E_1, \ldots, E_n)]\!] = \{f(s_1, \ldots, s_n) \mid s_1 \in [\![E_1]\!], \ldots, s_n \in [\![E_n]\!]\}$,

- $[\![E_1 + E_2]\!] = [\![E_1]\!] \cup [\![E_2]\!]$,
- $[\![E_1 ._\square E_2]\!] = [\![E_1]\!]\{\square \leftarrow [\![E_2]\!]\}$,
- $[\![E^{*,\square}]\!] = [\![E]\!]^{*,\square}$

Example 21. Let $\mathcal{F} = \{0, nil, s(), cons(,)\}$ and $\square \in \mathcal{K}$ then

$$(cons(0, \square)^{*,\square}) ._\square nil$$

is a regular expression of $Regexp(\mathcal{F}, \mathcal{K})$ which denotes the set of lists of zeros:

$$\{nil, cons(0, nil), cons(0, cons(0, nil)), \ldots\}$$

In the remaining of this section, we compare the relative expressive power of regular expressions and regular languages. It is easy to prove that for each regular expression E, the set $[\![E]\!]$ is a regular tree language. The proof is done by structural induction on E. The first three cases are obvious and the two last cases are consequences of Propositions 5 and 4. The converse, *i.e.*a regular tree language can be denoted by a regular expression, is more involved and the proof is similar to the proof of Kleene's theorem for word language. Let us state the result first.

Proposition 6. *Let $\mathcal{A} = (Q, \mathcal{F}, Q_F, \Delta)$ be a bottom-up tree automaton, then there exists a regular expression E of $Regexp(\mathcal{F}, Q)$ such that $L(\mathcal{A}) = [\![E]\!]$.*

The occurrence of symbols of Q in the regular expression denoting $L(\mathcal{A})$ doesn't cause any trouble since a regular expression of $Regexp(\mathcal{F}, Q)$ can denote a language of $T_{\mathcal{F}}$.

Proof. The proof is similar to the proof for word languages and word automata. For each $1 \leq i, j, \leq |Q|, K \subseteq Q$, we define the set $T(i, j, K)$ as the set of trees t of $T(\mathcal{F} \cup K)$ such that there is a run r of $\mathcal{A}$ on t satisfying the following properties:

- $r(\epsilon) = q_i$,
- $r(p) \in \{q_1, \ldots, q_j\}$ for all $p \neq \epsilon$ labelled by a function symbol.

Roughly speaking, a term is in $T(i, j, K)$ if we can reach q_i at the root by using only states in $\{q_1, \ldots, q_j\}$ when we assume that the leaves are states of K. By definition, $L(\mathcal{A})$ the language accepted by $\mathcal{A}$ is the union of the $T(i, |Q|, \emptyset)$'s for i such that q_i is a final state: these terms are the terms of $T(\mathcal{F})$ such that there is a successful run using any possible state of Q. Now, we prove by induction on j that $T(i, j, K)$ can be denoted by a regular expression of $Regexp(\mathcal{F}, Q)$.

- Base case $j = 0$. The set $T(i, 0, K)$ is the set of trees t where the root is labelled by q_i, the leaves are in $\mathcal{F} \cup K$ and no internal node is labelled by some q. Therefore there exist $a_1, \ldots, a_n, a \in \mathcal{F} \cup K$ such that $t = f(a_1, \ldots, a_n)$ or $t = a$, hence $T(i, 0, K)$ is finite and can be denoted by a regular expression of $Regexp(\mathcal{F} \cup Q)$.

- Induction case. Let us assume that for any $i', K' \subseteq Q$ and $0 \leq j' < j$, the set $T(i', j', K')$ can be denoted by a regular expression. We can write the following equality:

$$\begin{aligned} T(i,j,K) = \; & T(i,j-1,K) \\ & \cup \\ & T(i,j-1,K \cup \{q_j\}) \;.q_j\; T(j,j-1,K \cup \{q_j\})^{\;*,q_j} \;.q_j\; T(j,j-1,K) \end{aligned}$$

The inclusion of $T(i,j,K)$ in the right-hand side of the equality can be easily seen from Figure 2.2.2.

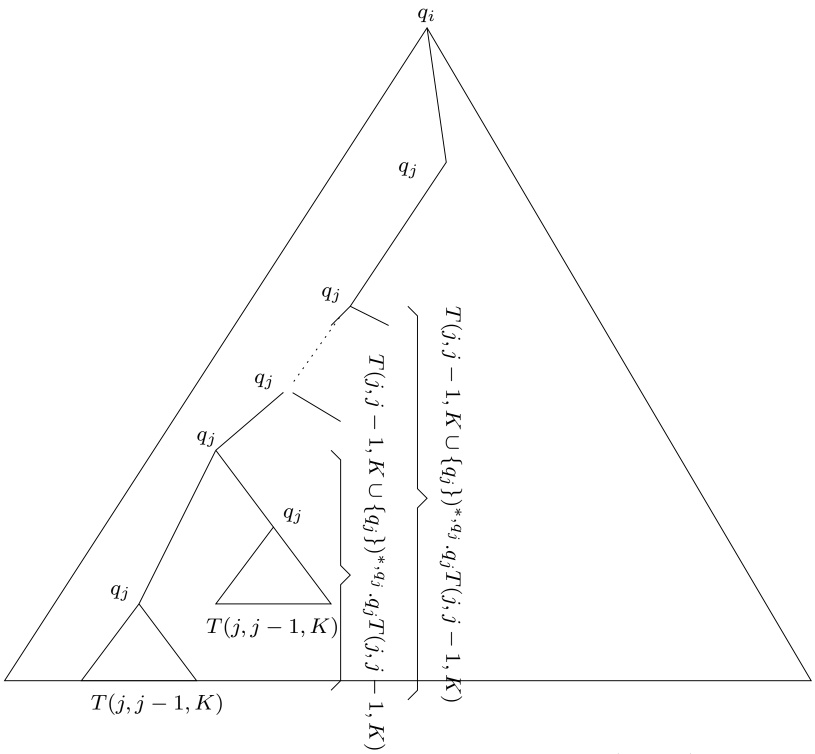


Figure 2.2: Decomposition of a term of $T(i,j,K)$

The converse inclusion is also not difficult. By definition:
$T(i,j-1,K) \subseteq T(i,j,K)$
and an easy proof by induction on the number of occurrences of q_j yields:
$T(i,j-1,K \cup \{q_j\}) \;.q_j\; T(j,j-1,K \cup \{q_j\})^{\;*,q_j} \;.q_j\; T(j,j-1,K) \subseteq T(i,j,K)$

By induction hypothesis, each set of the right-hand side of the equality defining $T(i,j,K)$ can be denoted by a regular expression of $Regex(\mathcal{F} \cup Q)$. This yields the desired result because the union of these sets is represented by the sum of the corresponding expressions.

□

Since we have already seen that regular expressions denote recognizable tree languages and that recognizable languages are regular, we can state Kleene's theorem for tree languages.

Theorem 19. *A tree language is recognizable if and only if it can be denoted by a regular tree expression.*

2.3 Regular Equations

Looking at our example of the set of lists of non-negative integers, we can realize that these lists can be defined by equations instead of grammar rules. For instance, denoting set union by $+$, we could replace the grammar given in Section 2.1.1 by the following equations.

$$\begin{array}{l} Nat = 0 \ + \ s(Nat) \\ List = nil \ + \ cons(Nat, List) \end{array}$$

where the variables are $List$ and Nat. To get the usual lists of non-negative numbers, we must restrict ourselves to the least fixed-point solution of this set of equations. Systems of language equations do not always have solution nor does a least solution always exists. Therefore we shall study ***regular equation systems*** defined as follows.

Definition 3. *Let $X_1, \ldots, X_n$ be variables denoting sets of trees, for $1 \leq j \leq p$, $1 \leq i \leq m_j$,let s_i^j's be terms over $\mathcal{F} \cup \{X_1, \ldots, X_n\}$, then a regular equation system S is a set of equations of the form:*

$$\begin{array}{rcl} X_1 & = & s_1^1 + \ldots + s_{m_1}^1 \\ & & \ldots \\ X_p & = & s_1^p + \ldots + s_{m_p}^p \end{array}$$

A solution of S is any n-tuple $(L_1, \ldots, L_n)$ of languages of $T(\mathcal{F})$ such that

$$\begin{array}{rcl} L_1 & = & s_1^1\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\} \cup \ldots \cup s_{m_1}^1\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\} \\ & & \ldots \\ L_p & = & s_1^p\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\} \cup \ldots \cup s_{m_p}^p\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\} \end{array}$$

Since equations with the same left-hand side can be merged into one equation, and since we can add equations $X_k = X_k$ without changing the set of solutions of a system, we assume in the following that $p = n$.

The ordering $\subseteq$ is defined on $T(\mathcal{F})^n$ by

$$(L_1, \ldots, L_n) \subseteq (L'_1, \ldots, L'_n) \text{ iff } L_i \subseteq L'_i \text{ for all i} = 1, \ldots, \text{n}$$

By definition $(\emptyset, \ldots, \emptyset)$ is the smallest element of $\subseteq$ and each increasing sequence has an upper bound. To a system of equations, we associate the fixed-point operator $\mathcal{TS} : T(\mathcal{F})^n \rightarrow T(\mathcal{F})^n$ defined by:

$$\begin{array}{l} \mathcal{TS}(L_1, \ldots, L_n) = (L'_1, \ldots, L'_n) \\ \textit{where} \\ \begin{array}{rcl} L'_1 & = & L_1 \cup s_1^1\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\} \cup \ldots \cup s_{m_1}^1\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\} \\ & & \ldots \\ L'_n & = & L_n \cup s_1^n\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\} \cup \ldots \cup s_{m_n}^n\{X_1 \leftarrow L_1, \ldots, X_n \leftarrow L_n\} \end{array} \end{array}$$

Example 22. Let S be

$$\begin{array}{rcl} Nat & = & 0 + s(Nat) \\ List & = & nil + cons(Nat, List) \end{array}$$

then

$$\begin{array}{l} \mathcal{TS}(\emptyset, \emptyset) = (\{0\}, \{nil\}) \\ \mathcal{TS}^2(\emptyset, \emptyset) = (\{0, s(0)\}, \{nil, cons(0, nil)\}) \end{array}$$

Using a classical approach we use the fixed-point operator to compute the least fixed-point solution of a system of equations.

Proposition 7. *The fixed-point operator $\mathcal{TS}$ is continuous and its least fixed-point $\mathcal{TS}^\omega(\emptyset, \dots, \emptyset)$ is the least solution of S.*

Proof. We show that $\mathcal{TS}$ is continuous in order to use Knaster-Tarski's theorem on continuous operators. By construction, $\mathcal{TS}$ is monotonous, and the last point is to prove that if $S_1 \subseteq S_2 \subseteq \dots$ is an increasing sequence of n-tuples of languages, the equality $\mathcal{TS}(\bigcup_{i \geq 1} S_i) = \bigcup_{i \geq 1} \mathcal{TS}(S_i))$ holds. By definition, each S_i can be written as $(S_1^i, \dots, S_n^i)$.

- We have that $\bigcup_{i=1,\dots} \mathcal{TS}(S_i) \subseteq \mathcal{TS}(\bigcup_{i=1,\dots}(S_i)$ holds since the sequence $S_1 \subseteq S_2 \subseteq \dots$ is increasing and the operator $\mathcal{TS}$ is monotonous.
- Conversely we must prove $\mathcal{TS}(\bigcup_{i=1,\dots} S_i) \subseteq \bigcup_{i=1,\dots} \mathcal{TS}(S_i))$.

  Let $v = (v^1, \dots, v^n) \in \mathcal{TS}(\bigcup_{i=1,\dots} S_i)$. Then for each $k = 1, \dots, n$ either $v^k \in \bigcup_{i=1,\dots} S_i$ hence $v^k \in S_{l_k}$ for some l_k, or there is some $u = (u^1, \dots, u^n) \in \bigcup_{i \geq 1} S_i$ such that $v^k = s_{j_k}^k\{X_1 \leftarrow u^1, \dots, X_n \leftarrow u^n\}$. Since the sequence $(S_{i,\ i \geq 1})$ is increasing we have that $u \in S_{l_k}$ for some l_k. Therefore $v^k \in \mathcal{TS}(S_L) \subseteq \mathcal{TS}(\bigcup_{i=1,\dots} S_i)$ for $L = max\{l_k \mid k = 1, \dots, n\}$.

□

We have introduced systems of regular equations to get an algebraic characterization of regular tree languages stated in the following theorem.

Theorem 20. *The least fixed-point solution of a system of regular equations is a tuple of regular tree languages. Conversely each regular tree language is a component of the least solution of a system of regular equations.*

Proof. Let S be a system of regular equations, and let $G_i = (X_i, \{X_1, \dots, X_n\}, \mathcal{F}, R)$ where $R = \cup_{k=1,\dots,n}\{X_k \rightarrow s_k^1, \dots, X_k \rightarrow s_k^{j_k}\}$ if the k^{th} equation of S is $X_k = s_k^1 + \dots + s_k^{j_k}$. We show that $L(G_i)$ is the i^{th} component of $(L_1, \dots, L_n)$ the least fixed-point solution of S.

- We prove that $\mathcal{TS}^p(\emptyset, \dots, \emptyset) \subseteq (L(G_1), \dots, L(G_n))$ by induction on p.

  Let us assume that this property holds for all $p' \leq p$. Let $u = (u_1, \dots, u_n)$ be an element of $TS^{p+1}(\emptyset, \dots, \emptyset) = \mathcal{TS}(\mathcal{TS}^p(\emptyset, \dots, \emptyset))$. For each i in

$1, \ldots, n$, either $u^i \in \mathcal{TS}^p(\emptyset, \ldots, \emptyset)$ and $u_i \in L(G_i)$ by induction hypothesis, or there exist $v^i = (v_1^i, \ldots, v_n^i) \in TS^p(\emptyset, \ldots, \emptyset)$ and s_i^j such that $u_i = s_i^j\{X_1 \to v_1^i, \ldots, X_n \to v_n^i\}$. By induction hypothesis $v_j^i \in L(G_j)$ for $j = 1, \ldots, n$ therefore $u_i \in L(G_i)$.

- We prove now that $(L(X_1), \ldots, L(X_n)) \subseteq \mathcal{TS}^\omega(\emptyset, \ldots, \emptyset)$ by induction on derivation length.

  Let us assume that for each $i = 1, \ldots, n$, for each $p' \leq p$, if $X_i \to^{p'} u_i$ then $u_i \in \mathcal{TS}^{p'}(\emptyset, \ldots, \emptyset)$. Let $X_i \to^{p+1} u_i$, then $X_i \to s_i^j(X_1, \ldots, X_n) \to^p v_i$ with $u_i = s_i^j(v_1, \ldots, v_n)$ and $X_j \to^{p'} v_j$ for some $p' \leq p$. By induction hypothesis $v_j \in \mathcal{TS}^{p'}(\emptyset, \ldots, \emptyset)$ which yields that $u_i \in \mathcal{TS}^{p+1}(\emptyset, \ldots, \emptyset)$.

Conversely, given a regular grammar $G = (S, \{A_1, \ldots, A_n\}, \mathcal{F}, R)$, with $R = \{A_1 \to s_1^1, \ldots, A_1 \to s_{p_1}^1, \ldots, A_n \to s_1^n, \ldots, A_n \to s_{p_n}^n\}$, a similar proof yields that the least solution of the system

$$\begin{array}{rcl} A_1 & = & s_1^1 + \ldots + s_{p_1}^1 \\ & & \ldots \\ A_n & = & s_1^n + \ldots + s_{p_n}^n \end{array}$$

is $(L(A_1), \ldots, L(A_n))$. □

Example 23. The grammar with axiom *List*, non-terminals *List*, *Nat* terminals $0, s(), nil, cons(,)$ and rules

$$\begin{array}{lcl} List & \to & nil \\ List & \to & cons(Nat, List) \\ Nat & \to & 0 \\ Nat & \to & s(Nat) \end{array}$$

generates the second component of the least solution of the system given in Example 22.

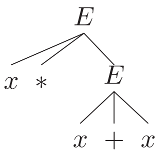
2.4 Context-free Word Languages and Regular Tree Languages

Context-free word languages and regular tree languages are strongly related. This is not surprising since derivation trees of context-free languages and derivations of tree grammars look alike. For instance let us consider the context-free language of arithmetic expressions on $+$,$*$ and a variable x. A context-free word grammar generating this set is $E \to x \mid E + E \mid E * E$ where E is the axiom. The generation of a word from the axiom can be described by a derivation tree which has the axiom at the root and where the generated word can be read by picking up the leaves of the tree from the left to the right (computing what we call the yield of the tree). The rules for constructing derivation trees show some regularity, which suggests that this set of trees is regular. The aim of this section is to show that this is true indeed. However, there are some traps which must be avoided when linking tree and word languages. First, we describe how to relate word and trees. The symbols of $\mathcal{F}$ are used to build trees but also words (by taking a symbol of $\mathcal{F}$ as a letter). The ***Yield*** operator computes a word from a tree by concatenating the leaves of the tree from the left to the right. More precisely, it is defined as follows.

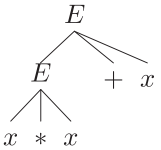
$Yield(a) = a$ if $a\imath\mathcal{F}_0$,

$Yield(f(s_1, \ldots, s_n)) = Yield(s_1) \ldots Yield(s_n)$ if $f \in \mathcal{F}_n$, $s_i \in T(\mathcal{F})$.

Example 24. Let $\mathcal{F} = \{x, +, *, E(,,)\}$ and let

$s =$ 

then $Yield(s) = x * x + x$ which is a word on $\{x, *, +\}$. Note that $*$ and $+$ are not the usual binary operator but syntactical symbols of arity 0. If

$t =$ 

then $Yield(t) = x * x + x$.

We recall that a ***context-free word grammar*** G is a tuple (S, N, T, R) where S is the axiom, N the set of non-terminals letters, T the set of terminal letters, R the set of production rules of the form $A \rightarrow \alpha$ with $A \in N, \alpha \in (T \cup N)^*$. The usual definition of derivation trees of context free word languages allow nodes labelled by a non-terminal A to have a variable number of sons, which is equal to the length of the right-hand side α of the rule $A \rightarrow \alpha$ used to build the derivation tree at this node.

Since tree languages are defined for signatures where each symbol has a fixed arity, we introduce a new symbol (A, m) for each $A \in N$ such that there is a rule $A \rightarrow \alpha$ with α of length m. Let $\mathcal{G}$ be the set composed of these new symbols and of the symbols of T. The set of derivation trees issued from $a \in \mathcal{G}$, denoted by $D(G, a)$ is the smallest set such that:

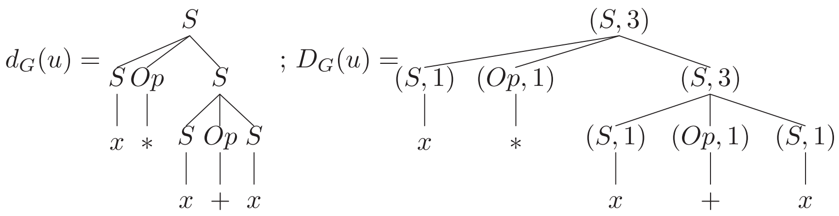
- $D(G, a) = \{a\}$ if $a \in T$,
- $(a, 0)(\epsilon) \in D(G, a)$ if $a \rightarrow \epsilon \in R$ where ϵ is the empty word,
- $(a, p)(t_1, \ldots, t_p) \in D(G, (a, p))$ if $t_1 \in D(G, a_1), \ldots, t_p \in D(G, a_p)$ and $(a \rightarrow a_1 \ldots a_p) \in R$ where $a_i \in \mathcal{G}$.

The set of ***derivation trees*** of G is $D(G) = \cup_{(S,i) \in \mathcal{G}} D(G, (S, i))$.

Example 25. Let $T = \{x, +, *\}$ and let G be the context free word grammar with axiom S, non terminal Op, and rules

$$
\begin{array}{l}
S \to S\ Op\ S \\
S \to x \\
Op \to + \\
Op \to *
\end{array}
$$

Let the word $u = x * x + x$, a derivation tree for u with G is $d_G(u)$, and the same derivation tree with our notations is $D_G(u) \in D(G, S)$



By definition, the language generated by a context-free word grammar G is the set of words computed by applying the *Yield* operator to derivation trees of G. The next theorem states how context-free word languages and regular tree languages are related.

Theorem 21. *The following statements hold.*

1. *Let G be a context-free word grammar, then the set of derivation trees of $L(G)$ is a regular tree language.*
2. *Let L be a regular tree language then $Yield(L)$ is a context-free word language.*
3. *There exists a regular tree language which is not the set of derivation trees of a context-free language.*

Proof. We give the proofs of the three statements.

1. Let $G = (S, N, T, R)$ be a context-free word language. We consider the tree grammar $G' = (S, N, \mathcal{F}, R'))$ such that
   - the axiom and the set of non-terminal symbols of G and G' are the same,
   - $\mathcal{F} = T \cup \{\epsilon\} \cup \{(A, n) \mid A \in N,\ \exists A \to \alpha \in R \textit{ with } \alpha \textit{ of length } n\}$,
   - if $A \to \epsilon$ then $A \to (A, 0)(\epsilon) \in R'$
   - if $(A \to a_1 \dots a_p) \in R$ then $(A \to (A, p)(a_1, \dots, a_p)) \in R'$

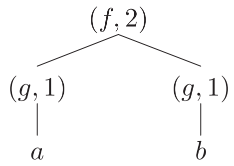
   Then $L(G) = \{\, Yield(s) \mid s \in L(G')\}$. The proof is a standard induction on derivation length. It is interesting to remark that there may and usually does exist several tree languages (not necessarily regular) such that the corresponding word language obtained via the *Yield* operator is a given context-free word language.

2. Let G be a normalized tree grammar (S, X, N, R). We build the word context-free grammar $G' = (S, X, N, R')$ such that a rule $X \to X_1 \dots X_n$ (resp. $X \to a$) is in R' if and only if the rule $X \to f(X_1, \dots, X_n)$ (resp. $X \to a$) is in R for some f. It is straightforward to prove by induction on the length of derivation that $L(G') = Yield(L(G))$.

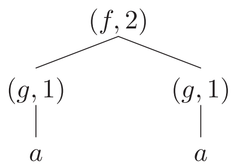
3. Let G be the regular tree grammar with axiom X, non-terminals X, Y, Z, terminals a, b, g and rules

$$\begin{array}{lcl} X & \rightarrow & f(Y, Z) \\ Y & \rightarrow & g(a) \\ Z & \rightarrow & g(b) \end{array}$$

   The language $L(G)$ consists of the single tree (arity have been indicated explicitly to make the link with derivation trees):



   Assume that $L(G)$ is the set of derivation trees of some context-free word grammar. To generate the first node of the tree, one must have a rule $F \rightarrow G\ G$ where F is the axiom and rules $G \rightarrow a$, $G \rightarrow b$ (to get the inner nodes). Therefore the following tree:



   should be in $L(G)$ which is not the case.

□

2.5 Beyond Regular Tree Languages: Context-free Tree Languages

For word language, the story doesn't end with regular languages but there is a strict hierarchy.

$$\text{regular} \subset \text{context-free} \subset \text{recursively enumerable}$$

Recursively enumerable tree languages are languages generated by tree grammar as defined in the beginning of the chapter, and this class is far too general for having good properties. Actually, any Turing machine can be simulated by a one rule rewrite system which shows how powerful tree grammars are (any grammar rule can be seen as a rewrite rule by considering both terminals and non-terminals as syntactical symbols). Therefore, most of the research has been done on context-free tree languages which we describe now.

2.5.1 Context-free Tree Languages

A ***context-free tree grammar*** is a tree grammar $G = (S, N, \mathcal{F}, R)$ where the rules have the form $X(x_1, \ldots, x_n) \rightarrow t$ with t a tree of $T(\mathcal{F} \cup N \cup \{x_1, \ldots, x_n\})$, $x_1, \ldots, x_n \in \mathcal{X}$ where $\mathcal{X}$ is a set of reserved variables with $\mathcal{X} \cap (\mathcal{F} \cup N) = \emptyset$, X a non-terminal of arity n. The definition of the derivation relation is slightly more complicated than for regular tree grammar: a term t derives a term t' if no variable of $\mathcal{X}$ occurs in t or t', there is a rule $l \rightarrow r$ of the grammar, a substitution σ such that the domain of σ is included in $\mathcal{X}$ and a context C such that $t = C[l\sigma]$ and $t' = C[r\sigma]$. The ***context-free tree language*** $L(G)$ is the set of trees which can be derived from the axiom of the context-free tree grammar G.

Example 26. The grammar of axiom $Prog$, set of non-terminals $\{Prog, Nat, Fact()\}$, set of terminals $\{0, s, if(,), eq(,), not(), times(,), dec()\}$ and rules

$$\begin{array}{lcl}
Prog & \rightarrow & Fact(Nat) \\
Nat & \rightarrow & 0 \\
Nat & \rightarrow & s(Nat) \\
Fact(x) & \rightarrow & if(eq(x,0), s(0)) \\
Fact(x) & \rightarrow & if(not(eq(x,0)), times(x, Fact(dec(x))))
\end{array}$$

where $\mathcal{X} = \{x\}$ is a context-free tree grammar. The reader can easily see that the last rule is the classical definition of the factorial function.

The derivation relation associated to a context-free tree grammar G is a generalization of the derivation relation for regular tree grammar. The derivation relation $\rightarrow$ is a relation on pairs of terms of $T(\mathcal{F} \cup N)$ such that $s \rightarrow t$ iff there is a rule $X(x_1, \ldots, x_n) \rightarrow \alpha \in R$, a context C such that $s = C[X(t_1, \ldots, t_n)]$ and $t = C[\alpha\{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}]$. For instance, the previous grammar can yield the sequence of derivations

$$Prog \rightarrow Fact(Nat) \rightarrow Fact(0) \rightarrow if(eq(0,0), s(0))$$

The language generated by G, denoted by $L(G)$ is the set of terms of $T(\mathcal{F})$ which can be reached by successive derivations starting from the axiom. Such languages are called ***c***ontext-free tree languages. Context-free tree languages are closed under union, concatenation and closure. Like in the word case, one can define pushdown tree automata which recognize exactly the set of context-free tree languages. We discuss only IO and OI grammars and we refer the reader to the bibliographic notes for more informations.

2.5.2 IO and OI Tree Grammars

Context-free tree grammars have been extensively studied in connection with the theory of recursive program scheme. A non-terminal F can be seen as a function name and production rules $F(x_1, \ldots, x_n) \rightarrow t$ define the function. Recursive definitions are allowed since t may contain occurrences of F. Since we know that such recursive definitions may not give the same results depending on the evaluation strategy, IO and OI tree grammars have been introduced to account for such differences.

A context-free grammar is ***IO*** (for innermost-outermost) if we restrict legal derivations to derivations where the innermost terminals are derived first. This control corresponds to call by value evaluation. A context-free grammar is ***OI*** (for outermost-innermost) if we restrict legal derivations to derivations where the outermost terminals are derived first. This corresponds to call by name evaluation. Therefore, given one context-free grammar G, we can define $IO\text{-}G$ and $OI\text{-}G$ and the next example shows that the languages generated by these grammars may be different.

Example 27. Let G be the context-free grammar with axiom Exp, non-terminals $\{Exp, Nat, Dup\}$, terminals $\{double, s, 0\})$ and rules
$Exp \to Dup(Nat)$
$Nat \to s(Nat)$
$Nat \to 0$
$Dup(x) \to double(x, x)$
Then outermost-innermost derivations have the form

$$Exp \to Dup(Nat) \to double(Nat, Nat) \xrightarrow{*} double(s^n(0), s^m(0))$$

while innermost-outermost derivations have the form

$$Exp \to Dup(Nat) \xrightarrow{*} Dup(s^n(0)) \to double(s^n(0), s^n(0))$$

Therefore $L(OI\text{-}G) = \{double(s^n(0), s^m(0)) \mid n, m \in \mathbb{N}\}$ and
$L(IO\text{-}G) = \{double(s^n(0), s^n(0)) \mid n \in \mathbb{N}\}$.

A tree language L is IO if there is some context-free grammar G such that $L = L(IO\text{-}G)$. The next theorem shows the relation between $L(IO\text{-}G)$, $L(OI\text{-}G)$ and $L(G)$.

Theorem 22. *The following inclusion holds:* $L(IO\text{-}G) \subseteq L(OI\text{-}G) = L(G)$

Example 27 shows that the inclusion can be strict. IO-languages are closed under intersection with regular languages and union, but the closure under concatenation requires another definition of concatenation: all occurrences of a constant generated by a non right-linear rule are replaced by the *same* term, as shown by the next example.

Example 28. Let G be the context-free grammar with axiom Exp, non-terminals $\{Exp, Nat, Fct\}$, terminals $\{\square, f(\_,\_,\_)\}$ and rules
$Exp \to Fct(Nat, Nat)$
$Nat \to \square$
$Fct(x, y) \to f(x, x, y)$
and let $L = IO\text{-}G$ and $M = \{0, 1\}$, then $L._\square M$ contains $f(0, 0, 0)$,$f(0, 0, 1)$, $f(1, 1, 0)$, $f(1, 1, 1)$ but not $f(1, 0, 1)$ nor $f(0, 1, 1)$.

There is a lot of work on the extension of results on context-free word grammars and languages to context-free tree grammars and languages. Unfortunately, many constructions and theorem can't be lifted to the tree case. Usually the failure is due to non-linearity which expresses that the same subtrees must occur at different positions in the tree. A similar phenomenon occurred when we stated results on recognizable languages and tree homomorphisms: the inverse image of a recognizable tree language by a tree homorphism is recognizable, but the assumption that the homomorphism is linear is needed to show that the direct image is recognizable.

2.6 Exercises

Exercise 20. Let $\mathcal{F} = \{f(,), g(), a\}$. Consider the automaton $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ defined by: $Q = \{q, q_g, q_f\}$, $Q_f = \{q_f\}$, and $\Delta =$

$$\begin{array}{lrcllcl} \{ & a & \rightarrow & q(a) & g(q(x)) & \rightarrow & q(g(x)) \\ & g(q(x)) & \rightarrow & q_g(g(x)) & g(q_g(x)) & \rightarrow & q_f(g(x)) \\ & f(q(x), q(y)) & \rightarrow & q(f(x,y)) & \}. & & \end{array}$$

Define a regular tree grammar generating $L(\mathcal{A})$.

Exercise 21.

1. Prove the equivalence of a regular tree grammar and of the reduced regular tree grammar computed by algorithm of proposition 2.
2. Let $\mathcal{F} = \{f(,), g(), a\}$. Let G be the regular tree grammar with axiom X, non-terminal A, and rules

   $$\begin{array}{l} X \rightarrow f(g(A), A) \\ A \rightarrow g(g(A)) \end{array}$$

   Define a top-down NFTA, a NFTA and a DFTA for $L(G)$. Is it possible to define a top-down DFTA for this language?

Exercise 22. Let $\mathcal{F} = \{f(,), a\}$. Let G be the regular tree grammar with axiom X, non-terminals A, B, C and rules

$$\begin{array}{l} X \rightarrow C \\ X \rightarrow a \\ X \rightarrow A \\ A \rightarrow f(A, B) \\ B \rightarrow a \end{array}$$

Compute the reduced regular tree grammar equivalent to G applying the algorithm defined in the proof of Proposition 2. Now, consider the same algorithm, but first apply step 2 and then step 1. Is the output of this algorithm reduced? equivalent to G?

Exercise 23.

1. Prove Theorem 6 using regular tree grammars.
2. Prove Theorem 7 using regular tree grammars.

Exercise 24. (Local languages) Let $\mathcal{F}$ be a signature, let t be a term of $T(\mathcal{F})$, then we define $fork(t)$ as follows:

- $fork(a) = \emptyset$, for each constant symbol a;

- $fork(f(t_1, \ldots, t_n)) = \{f(\mathcal{H}ead(t_1), \ldots, \mathcal{H}ead(t_n))\} \cup \bigcup_{i=1}^{i=n} fork(t_i)$

A tree language L is ***local*** if and only if there exist a set $\mathcal{F}' \subseteq \mathcal{F}$ and a set $G \subseteq fork(T(\mathcal{F}))$ such that $t \in L$ iff $root(t) \in \mathcal{F}'$ and $fork(t) \subseteq G$. Prove that every local tree language is a regular tree language. Prove that a language is local iff it is the set of derivation trees of a context-free word language.

Exercise 25. The pumping lemma for context-free word languages states:

> for each context-free language L, there is some constant $k \geq 1$ such that each $z \in L$ of length greater than or equal to k can be written $z = uvwxy$ such that vx is not the empty word, vwx has length less than or equal to k, and for each $n \geq 0$, the word uv^nwx^ny is in L.

Prove this result using the pumping lemma for tree languages and the results of this chapter.

Exercise 26. Another possible definition for the iteration of a language is:

- $L^{0,\ \Box} = \{\Box\}$
- $L^{n+1,\ \Box} = L^{n,\ \Box} \cup L^{n,\ \Box} \ ._\Box\ L$

(Unfortunately that definition was given in the previous version of TATA)

1. Show that this definition may generate non-regular tree languages. Hint: one binary symbol $f(\ ,\ )$ and $\Box$ are enough.
2. Are the two definitions equivalent (*i.e.*generate the same languages) if Σ consists of unary symbols and constants only?

Exercise 27. Let $\mathcal{F}$ be a ranked alphabet, let t be a term of $T(\mathcal{F})$, then we define the word language $Branch(t)$ as follows:

- $Branch(a) = a$, for each constant symbol a;
- $Branch(f(t_1, \ldots, t_n)) = \bigcup_{i=1}^{i=n} \{fu \mid u \in Branch(t_i)\}$

Let L be a regular tree language, prove that $Branch(L) = \bigcup_{t \in L} Branch(t)$ is a regular word language. What about the converse?

Exercise 28.

1. Let $\mathcal{F}$ be a ranked alphabet such that $\mathcal{F}_0 = \{a, b\}$. Find a regular tree language L such that $Yield(L) = \{a^nb^n \mid n \geq 0\}$. Find a non regular tree language L such that $Yield(L) = \{a^nb^n \mid n \geq 0\}$.
2. Same questions with $Yield(L) = \{u \in \mathcal{F}_0^* \mid |u|_a = |u|_b\}$ where $|u|_a$ (respectively $|u|_b$) denotes the number of a (respectively the number of b) in u.
3. Let $\mathcal{F}$ be a ranked alphabet such that $\mathcal{F}_0 = \{a, b, c\}$, let $A_1 = \{a^nb^nc^p \mid n, p \geq 0\}$, and let $A_2 = \{a^nb^pc^p \mid n, p \geq 0\}$. Find regular tree languages such that $Yield(L_1) = A_1$ and $Yield(L_2) = A_2$. Does there exist a regular tree language such that $Yield(L) = A_1 \cap A_2$.

Exercise 29.

1. Let G be the context free word grammar with axiom X, terminals a, b, and rules

$$X \to XX$$
$$X \to aXb$$
$$X \to \epsilon$$

   where ϵ stands for the empty word. What is the word language $L(G)$? Give a derivation tree for $u = aabbab$.

2. Let G' be the context free word grammar in Greibach normal form with axiom X, non terminals X', Y', Z' terminals a, b, and rules l

$$\begin{aligned}
&X' \rightarrow aX'Y' \\
&X' \rightarrow aY' \\
&X' \rightarrow aX'Z' \\
&X' \rightarrow aZ' \\
&Y' \rightarrow bX' \\
&Z' \rightarrow b
\end{aligned}$$

prove that $L(G') = L(G)$. Give a derivation tree for $u = aabbab$.

3. Find a context free word grammar G'' such that $L(G'') = A_1 \cup A_2$ (A_1 and A_2 are defined in Exercise 28). Give two derivation trees for $u = abc$.

Exercise 30. Let $\mathcal{F}$ be a ranked alphabet.

1. Let L and L' be two regular tree languages. Compare the sets $\mathit{Yield}(L \cap L')$ and $\mathit{Yield}(L) \cap \mathit{Yield}(L')$.
2. Let A be a subset of $\mathcal{F}_0$. Prove that $T(\mathcal{F}, A) = T(\mathcal{F} \cap A)$ is a regular tree language. Let L be a regular tree language over $\mathcal{F}$, compare the sets $\mathit{Yield}(L \cap T(\mathcal{F}, A))$ and $\mathit{Yield}(L) \cap \mathit{Yield}(T(\mathcal{F}, A))$.
3. Let R be a regular word language over $\mathcal{F}_0$. Let $T(\mathcal{F}, R) = \{t \in T(\mathcal{F}) \mid \mathit{Yield}(t) \in R\}$. Prove that $T(\mathcal{F}, R)$ is a regular tree language. Let L be a regular tree language over $\mathcal{F}$, compare the sets $\mathit{Yield}(L \cap T(\mathcal{F}, R))$ and $\mathit{Yield}(L) \cap \mathit{Yield}(T(\mathcal{F}, R))$. As a consequence of the results obtained in the present exercise, what could be said about the intersection of a context free word language and of a regular tree language?

2.7 Bibliographic notes

This chapter only scratches the topic of tree grammars and related topics. A useful reference on algebraic aspects of regular tree language is [GS84] which contains a lot of classical results on these features. There is a huge litterature on tree grammars and related topics, which is also relevant for the chapter on tree transducers, see the references given in this chapter. Systems of equations can be generalized to formal tree series with similar results [BR82, Boz99, Boz01, Kui99, Kui01]. The notion of pushdown tree automaton has been introduced by Guessarian [Gue83] and generalized to formal tree series by Kuich [Kui01] The reader may consult [Eng82, ES78] for IO and OI grammars. The connection between recursive program scheme and formalisms for regular tree languages is also well-known, see [Cou86] for instance. We should mention that some open problems like equivalence of deterministic tree grammars are now solved using the result of Senizergues on the equivalence of deterministic pushdown word automata [Sén97].

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[BT92] B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In A. Finkel and M. Jantzen, editors, *9th 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*$^{\text{th}}$ *Annual IEEE Symposium on Logic in Computer Science* [IEE97].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[GTT93] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints with negated subset relationships. In *Proceedings of the* 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 6*th *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 6*th *International Joint Conference on Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 42–61, 1995.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[Uri92] T. E. Uribe. Sorted Unification Using Set Constraints. In D. Kapur, editor, *Proceedings of the 11^{th} 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