Tree Automata Techniques and Applications

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

Contents

Acknowledgments

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

Introduction

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

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

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

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

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

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

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

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

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

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

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

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

Preliminaries

Terms

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

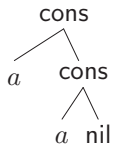
A ***ranked alphabet*** is a couple $(\mathcal{F}, Arity)$ where $\mathcal{F}$ is a finite set and $Arity$ is a mapping from $\mathcal{F}$ into N. The ***arity*** of a symbol $f \in \mathcal{F}$ is $Arity(f)$. The set of symbols of arity p is denoted by $\mathcal{F}_p$. Elements of arity $0, 1, \ldots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

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

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

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

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



Terms and Trees

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

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

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

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

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

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

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

SubTerms

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

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

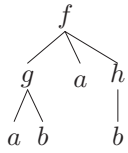
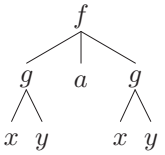
We denote by $\trianglerighteq$ the ***subterm ordering***, *i.e.*we write $t \trianglerighteq t'$ if t' is a subterm of t. We denote $t \triangleright t'$ if $t \trianglerighteq t'$ and $t \neq t'$.

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

Functions on Terms

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

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

$t =$  ; $t' =$ 

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

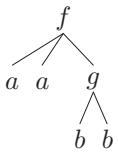
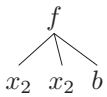
Substitutions

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

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

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

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

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

Contexts

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

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

Chapter 6

Tree Transducers

6.1 Introduction

Finite state transformations of words, also called a-transducers or rational transducers in the literature, model many kinds of processes, such as coffee machines or lexical translators. But these transformations are not powerful enough to model syntax directed transformations, and compiler theory is an important motivation to the study of finite state transformations of trees. Indeed, translation of natural or computing languages is directed by syntactical trees, and a translator from LaTeXinto HTML is a tree transducer. Unfortunately, from a theoretical point of view, tree transducers do not inherit nice properties of word transducers, and the classification is very intricate. So, in the present chapter we focus on some aspects. In Sections 6.2 and 6.3, toy examples introduce in an intuitive way different kinds of transducers. In Section 6.2, we summarize main results in the word case. Indeed, this book is mainly concerned with trees, but the word case is useful to understand the tree case and its difficulties. The bimorphism characterization is the ideal illustration of the link between the "machine" point of view and the "homomorphic" one. In Section 6.3, we motivate and illustrate bottom-up and top-down tree transducers, using compilation as leitmotiv. We precisely define and present the main classes of tree transducers and their properties in Section 6.4, where we observe that general classes are not closed under composition, mainly because of alternation of copying and nondeterministic processing. Nevertheless most useful classes, as those used in Section 6.3, have closure properties. In Section 6.5 we present the homomorphic point of view.

Most of the proofs are tedious and are omitted. This chapter is a very incomplete introduction to tree transducers. Tree transducers are extensively studied for themselves and for various applications. But as they are somewhat complicated objects, we focus here on the definitions and main general properties. It is usefull for every theoretical computer scientist to know main notions about tree transducers, because they are the main model of syntax directed manipulations, and that the heart of sofware manipulations and interfaces are syntax directed. Tree transducers are an essential frame to develop practical modular syntax directed algorithms, thought an effort of algorithmic engineering remains to do. Tree transducers theory can be fertilized by other area or can be usefull for

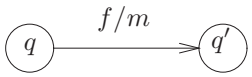
other areas (example: Ground tree transducers for decidability of the first order theory of ground rewriting). We will be happy if after reading this chapter, the reader wants for further lectures, as monograph of Z. Fülöp and H. Vögler (december 1998 [FV98]).

6.2 The Word Case

6.2.1 Introduction to Rational Transducers

We assume that the reader roughly knows popular notions of language theory: homomorphisms on words, finite automata, rational expressions, regular grammars. See for example the recent survey of A. Mateescu and A. Salomaa [MS96]. A rational transducer is a finite word automaton W with output. In a word automaton, a transition rule $f(q) \to q'(f)$ means "if W is in some state q, if it reads the input symbol f, then it enters state q' and moves its head one symbol to the right". For defining a rational transducer, it suffices to add an output, and a transition rule $f(q) \to q'(m)$ means "if the transducer is in some state q, if it reads the input symbol f, then it enters state q', writes the word m on the output tape, and moves its head one symbol to the right". Remark that with these notations, we identify a finite automaton with a rational transducer which writes what it reads. Note that m is not necessarily a symbol but can be a word, including the empty word. Furthermore, we assume that it is not necessary to read an input symbol, *i.e.*we accept transition rules of the form $\varepsilon(q) \to q'(m)$ (ε denotes the empty word).

Graph presentations of finite automata are popular and convenient. So it is for rational transducers. The rule $f(q) \to q'(m)$ will be drawn



Example 54. (**Language** L_1) Let $\mathcal{F} = \{\langle, \rangle, ;, 0, 1, A, ..., Z\}$. In the following, we will consider the language L_1 defined on $\mathcal{F}$ by the regular grammar (the axiom is **program**):

program $\to \langle$ **instruct**
instruct $\to$ `LOAD` **register** | `STORE` **register** | `MULT` **register**
$\to$ | `ADD` **register**
register $\to 1$**tailregister**
tailregister $\to 0$**tailregister** | 1**tailregister** | $;$ **instruct** | $\rangle$

($a \to b|c$ is an abbreviation for the set of rules $\{a \to b, a \to c\}$)

L_1 is recognized by deterministic automaton A_1 of Figure 6.1. Semantic of L_1 is well known: `LOAD i` loads the content of register `i` in the accumulator; `STORE i` stores the content of the accumulator in register `i`; `ADD i` adds in the accumulator the content of the accumulator and the content of register `i`; `MULT i` multiplies in the accumulator the content of the accumulator and the content of register `i`.

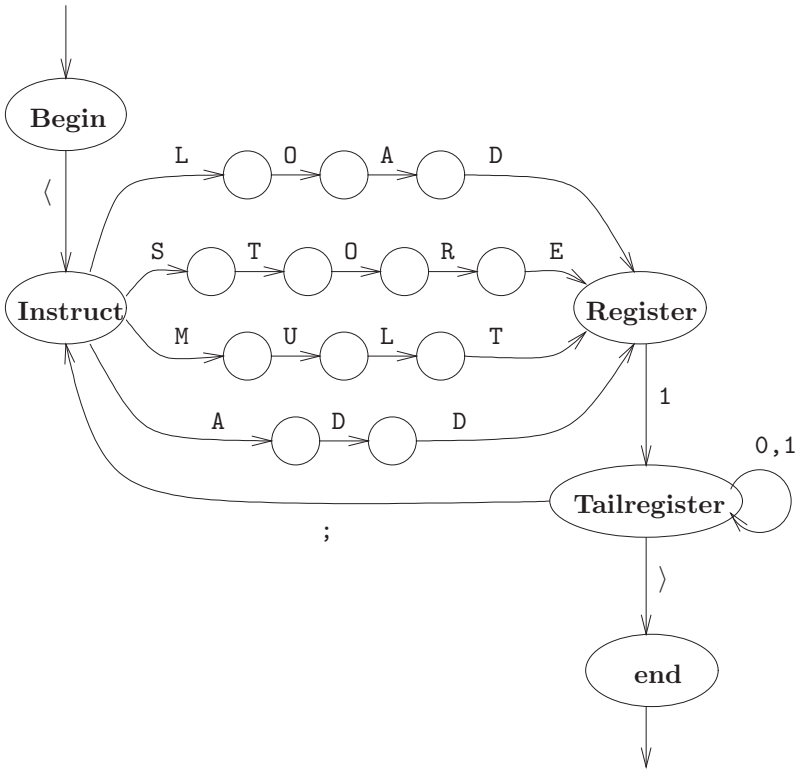


Figure 6.1: A recognizer of L_1

A ***rational transducer*** is a tuple $R = (Q, \mathcal{F}, \mathcal{F}', Q_i, Q_f, \Delta)$ where Q is a set of states, $\mathcal{F}$ and $\mathcal{F}'$ are finite nonempty sets of input letters and output letters, $Q_i, Q_f \subseteq Q$ are sets of initial and final states and Δ is a set of transduction rules of the following type:

$$f(q) \to q'(m),$$

where $f \in \mathcal{F} \cup \{\varepsilon\}$, $m \in \mathcal{F}'^*$, $q, q' \in Q$.

R is ***ε-free*** if there is no rule $f(q) \to q'(m)$ with $f = \varepsilon$ in Δ.

The ***move relation*** $\to_R$ is defined by: let $t, t' \in \mathcal{F}^*$, $u \in \mathcal{F}'^*$, $q, q' \in Q$, $f \in \mathcal{F}$, $m \in \mathcal{F}'^*$,

$$(tqft', u) \underset{R}{\to} (tfq't, um) \Leftrightarrow f(q) \to q'(m) \in \Delta,$$

and $\to_R^*$ is the reflexive and transitive closure of $\to_R$. A (partial) transduction of R on $tt't''$ is a sequence of move steps of the form $(tqt't'', u) \to_R^* (tt'q't'', uu')$. A transduction of R from $t \in \mathcal{F}^*$ into $u \in \mathcal{F}'^*$ is a transduction of the form $(qt, \varepsilon) \to_R^* (tq', u)$ with $q \in Q_i$ and $q' \in Q_f$.

The relation T_R induced by R can now be formally defined by:

$$T_R = \{(t, u) \mid (qt, \varepsilon) \underset{R}{\overset{*}{\to}} (tq', u) \text{ with } t \in \mathcal{F}^*, u \in \mathcal{F}'^*, q \in Q_i, q' \in Q_f\}.$$

A relation in $\mathcal{F}^* \times \mathcal{F}'^*$ is a rational transduction if and only if it is induced by some rational transducer. We also need the following definitions: let $t \in \mathcal{F}^*$, $T_R(t) = \{u \mid (t, u) \in T_R\}$. The translated of a language L is obviously the language defined by $T_R(L) = \{u \mid \exists t \in L, u \in T_R(t)\}$.

Example 55.

Ex. 55.1 Let us name French-L_1 the translation of L_1 in French (`LOAD` is translated into `CHARGER` and `STORE` into `STOCKER`). Transducer of Figure 6.2 realizes this translation. This example illustrates the use of rational transducers as lexical transducers.

Ex. 55.2 Let us consider the rational transducer *Diff* defined by $Q = \{q_i, q_s, q_l, q_d\}$, $\mathcal{F} = \mathcal{F}' = \{a, b\}$, $Q_i = \{q_i\}$, $Q_f = \{q_s, q_l, q_d\}$, and Δ is the set of rules:

type i $a(q_i) \to q_i(a)$, $b(q_i) \to q_i(b)$

type s $\varepsilon(q_i) \to q_s(a)$, $\varepsilon(q_i) \to q_s(b)$, $\varepsilon(q_s) \to q_s(a)$, $\varepsilon(q_s) \to q_s(b)$

type l $a(q_i) \to q_l(\varepsilon)$, $b(q_i) \to q_l(\varepsilon)$, $a(q_l) \to q_l(\varepsilon)$, $b(q_l) \to q_l(\varepsilon)$

type d $a(q_i) \to q_d(b)$, $b(q_i) \to q_d(a)$, $a(q_d) \to q_d(\varepsilon)$, $b(q_d) \to q_d(\varepsilon)$, $\varepsilon(q_d) \to q_d(a)$, $\varepsilon(q_d) \to q_d(b)$.

It is easy to prove that $T_{Diff} = \{(m, m') \mid m \neq m', m, m' \in \{a, b\}^*\}$.

We give without proofs some properties of rational transducers. For more details, see [Sal73] or [MS96] and Exercises 65, 66, 68 for 1, 4 and 5. The homomorphic approach presented in the next section can be used as an elegant way to prove 2 and 3 (Exercise 70).

Proposition 46 (Main properties of rational transducers).

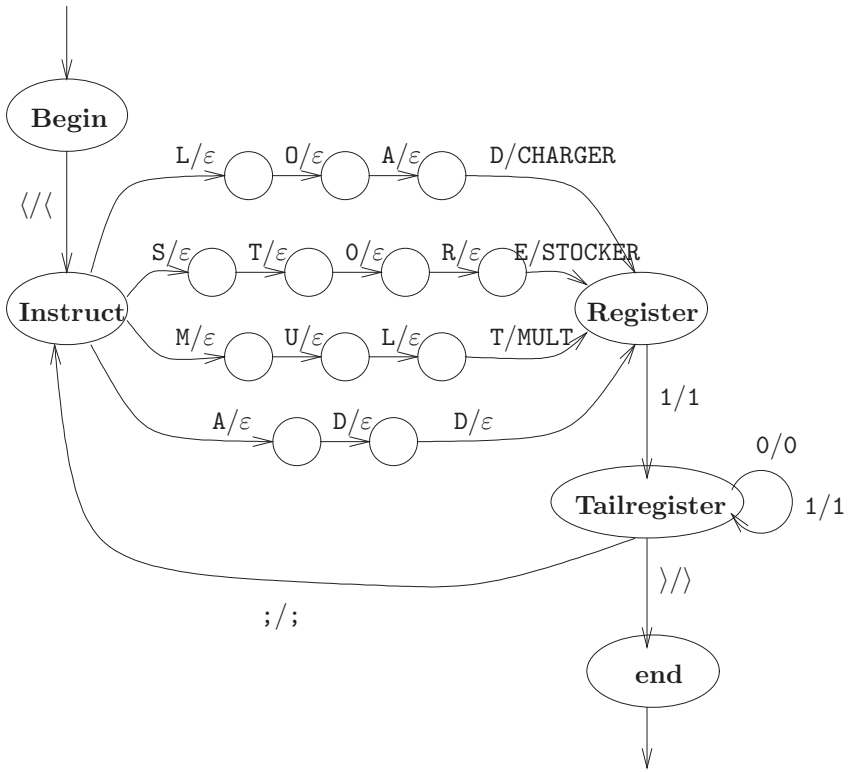


Figure 6.2: A rational transducer from L_1 into French-L_1.

1. *The class of rational transductions is closed under union but not closed under intersection.*

2. *The class of rational transductions is closed under composition.*

3. *Regular languages and context-free languages are closed under rational transduction.*

4. *Equivalence of rational transductions is undecidable.*

5. *Equivalence of deterministic rational transductions is decidable.*

6.2.2 The Homomorphic Approach

A ***bimorphism*** is defined as a triple $B = (\Phi, L, \Psi)$ where L is a recognizable language and Φ and Ψ are homomorphisms. The relation induced by B (also denoted by B) is defined by $B = \{(\Phi(t), \Psi(t)) \mid t \in L\}$. Bimorphism (Φ, L, Ψ) is ε-free if Φ is ε-free (an homomorphism is ε-free if the image of a letter is never reduced to ε). Two bimorphisms are equivalent if they induce the same relation.

We can state the following theorem, generally known as Nivat Theorem [Niv68] (see Exercises 69 and 70 for a sketch of proof).

Theorem 45 (Bimorphism theorem). *Given a rational transducer, an equivalent bimorphism can be constructed. Conversely, any bimorphism defines a rational transduction. Construction preserve ε-freeness.*

Example 56.

Ex. 56.1 The relation $\{(a(ba)^n, a^n) \mid n \in \mathbb{N}\} \cup \{((ab)^n, b^{3n}) \mid n \in \mathbb{N}\}$ is processed by transducer R and bimorphism B of Figure 6.3

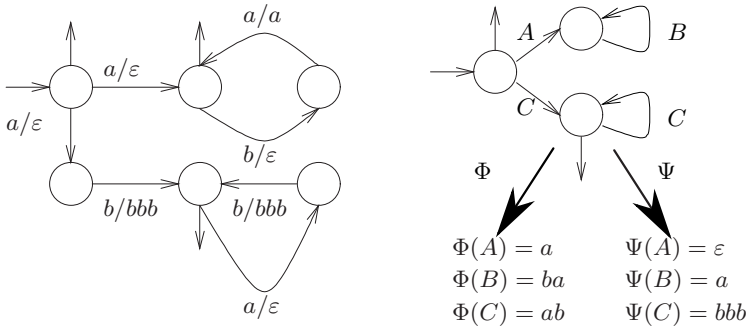


Figure 6.3: Transducer R and an equivalent bimorphism $B = \{(\Phi(t), \Psi(t)) \mid t \in AB^* + CC^*\}$.

Ex. 56.2 Automaton L of Figure 6.4 and morphisms Φ and Ψ bellow define a bimorphism equivalent to transducer of Figure 6.2

$\Phi(\beta) = \langle$ $\Phi(\lambda) =$ LOAD $\Phi(\sigma) =$ STORE $\Phi(\mu) =$ MULT
$\Phi(\alpha) =$ ADD $\Phi(\rho) =;$ $\Phi(\omega) = 1$ $\Phi(\zeta) = 0$
$\Phi(\theta) = \rangle$
$\Psi(\beta) = \langle$ $\Psi(\lambda) =$ CHARGER $\Psi(\sigma) =$ STOCKER $\Psi(\mu) =$ MULT
$\Psi(\alpha) =$ ADD $\Psi(\rho) =;$ $\Psi(\omega) = 1$ $\Psi(\zeta) = 0$
$\Psi(\theta) = \rangle$

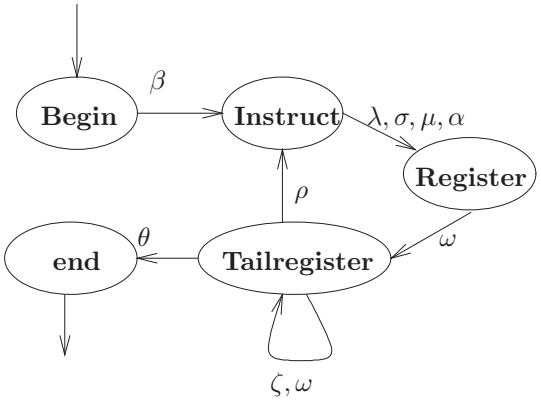


Figure 6.4: The control automaton L.

Nivat characterization of rational transducers makes intuitive sense. Automaton L can be seen as a control of the actions, morphism Ψ can be seen as output function and Φ^{-1} as an input function. Φ^{-1} analyses the input — it is a kind of part of lexical analyzer — and it generates symbolic names; regular grammatical structure on theses symbolic names is controlled by L. Examples 56.1 and 56.2 are an obvious illustration. L is the common structure to English and French versions, Φ generates the English version and Ψ generates the French one. This idea is the major idea of compilation, but compilation of computing languages or translation of natural languages are directed by syntax, that is to say by syntactical trees. This is the motivation of the rest of the chapter. But unfortunately, from a formal point of view, we will lose most of the best results of the word case. Power of non-linear tree transducers will explain in part this complication, but even in the linear case, there is a new phenomena in trees, the understanding of which can be introduced by the "problem of homomorphism inversion" that we describe in Exercise 71.

6.3 Introduction to Tree Transducers

Tree transducers and their generalizations model many syntax directed transformations (see exercises). We use here a toy example of compiler to illustrate how usual tree transducers can be considered as modules of compilers.

We consider a simple class of arithmetic expressions (with usual syntax) as source language. We assume that this language is analyzed by a LL1 parser. We consider two target languages: L_1 defined in Example 54 and an other language L_2. A transducer A translates syntactical trees in abstract trees (Figure 6.5). A second tree transducer R illustrates how tree transducers can be seen as

part of compilers which compute attributes over abstract trees. It decorates abstract trees with numbers of registers (Figure 6.7). Thus R translates abstract trees into attributed abstract trees. After that, tree transducers T_1 and T_2 generate target programs in L_1 and L_2, respectively, starting from attributed abstract trees (Figures 6.7 and 6.8). This is an example of nonlinear transducer. Target programs are yields of generated trees. So composition of transducers model succession of compilation passes, and when a class of transducers is closed by composition (see section 6.4), we get universal constructions to reduce the number of compiler passes and to meta-optimize compilers.

We now define **the source language**. Let us consider the terminal alphabet $\{(,),+,\times,a,b,\ldots,z\}$. First, the context-free word grammar G_1 is defined by rules (E is the axiom):

$$\begin{array}{rcl} E & \rightarrow & M \mid M + E \\ M & \rightarrow & F \mid F \times M \\ F & \rightarrow & I \mid (E) \\ I & \rightarrow & a \mid b \mid \cdots \mid z \end{array}$$

Another context-free word grammar G_2 is defined by (E is the axiom):

$$\begin{array}{rcl} E & \rightarrow & ME' \\ E' & \rightarrow & +E \mid \varepsilon \\ M & \rightarrow & FM' \\ M' & \rightarrow & \times M \mid \varepsilon \\ F & \rightarrow & I \mid (E) \\ I & \rightarrow & a \mid b \mid \cdots \mid z \end{array}$$

Let E be the axiom of G_1 and G_2. The semantic of these two grammars is obvious. It is easy to prove that they are equivalent, *i.e.*they define the same source language. On the one hand, G_1 is more natural, on the other hand G_2 could be preferred for syntactical analysis reason, because G_2 is LL1 and G_1 is not LL. We consider **syntactical trees** as derivation trees for the tree grammar G_2. Let us consider word $u = (a + b) \times c$. u of the source language. We define the abstract tree associated with u as the tree $\times(+(a,b),c)$ defined over $\mathcal{F} = \{+(,),\times(,),a,b,c\}$. **Abstract trees** are ground terms over $\mathcal{F}$. Evaluate expressions or compute attributes over abstract trees than over syntactical trees. The following transformation associates with a syntactical tree t its corresponding abstract tree $A(t)$.

$$\begin{array}{rcl} I(x) \rightarrow x & & F(x) \rightarrow x \\ M(x, M'(\varepsilon)) \rightarrow x & & E(x, E'(\varepsilon)) \rightarrow x \\ M(x, M'(\times, y)) \rightarrow \times(x, y) & & E(x, E'(+, y)) \rightarrow +(x, y) \\ F((, x,)) \rightarrow x & & \end{array}$$

We have not precisely defined the use of the arrow $\rightarrow$, but it is intuitive. Likewise we introduce examples before definitions of different kinds of tree transducers (section 6.4 supplies a formal frame).

To illustrate nondeterminism, let us introduce two new transducers A and A'. Some brackets are optional in the source language, hence A' is nondeterministic. Note that A works from frontier to root and A' works fromm root to frontier.

A: an Example of Bottom-up Tree Transducer

The following linear deterministic bottom-up tree transducer A carries out transformation of derivation trees for G_2 into the corresponding abstract trees. Empty word ε is identified as a constant symbol in syntactical trees. States of A are q, q_ε, q_I, q_F, $q_{M'\varepsilon}$, $q_{E'\varepsilon}$, q_E, $q_\times$, $q_{M'\times}$, q_+, $q_{E'+}$, $q_($, and $q_)$. Final state is q_E. The set of transduction rules is:

$$
\begin{array}{rlrl}
a & \to q(a) & b & \to q(b) \\
c & \to q(c) & \varepsilon & \to q_\varepsilon(\varepsilon) \\
) & \to q_)()) & (& \to q_((() \\
+ & \to q_+(+) & \times & \to q_\times(\times) \\
I(q(x)) & \to q_I(x) & F(q_I(x)) & \to q_F(x) \\
M'(q_\varepsilon(x)) & \to q_{M'\varepsilon}(x) & E'(q_\varepsilon(x)) & \to q_{E'\varepsilon}(x) \\
M(q_F(x), q_{M'\varepsilon(y)}) & \to q_M(x) & E(q_M(x), q_{E'\varepsilon}(y)) & \to q_E(x) \\
M'(q_\times(x), q_M(y)) & \to q_{M'\times}(y) & M(q_F(x), q_{M'\times}(y)) & \to q_M(\times(x,y)) \\
E'(q_+(x), q_E(y)) & \to q_{E'+}(y) & E(q_M(x), q_{E'+}(y)) & \to q_E(+(x,y)) \\
F(q_((x), q_E(y), q_)(z)) & \to q_F(y) & &
\end{array}
$$

The notion of (successful) run is an intuitive generalization of the notion of run for finite tree automata. The reader should note that FTAs can be considered as a special case of bottom-up tree transducers whose output is equal to the input. We give in Figure 6.5 an example of run of A which translates derivation tree t which yields $(a + b) \times c$ for context-free grammar G_2 into the corresponding abstract tree $\times(+(a, b), c)$.

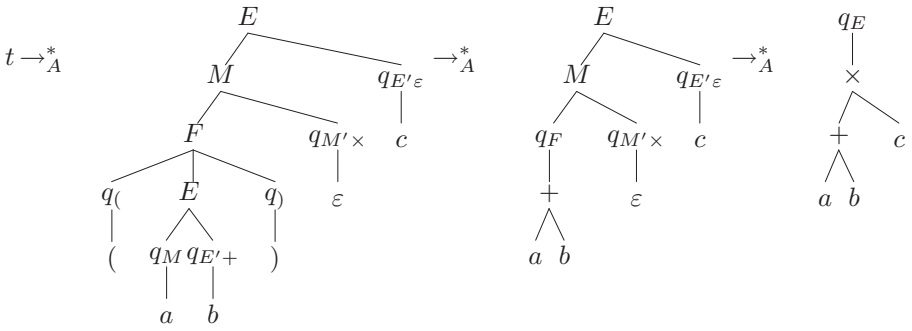


Figure 6.5: Example of run of A

A': an Example of Top-down Tree Transducer

The inverse transformation A^{-1}, which computes the set of derivation trees of G_2 associated with an abstract tree, is computed by a nondeterministic top-down tree transducer A'. The states of A' are q_E, q_F, q_M. The initial state is q_E. The set of transduction rules is:

$$
\begin{array}{ll}
q_E(x) \to E(q_M(x), E'(\varepsilon)) & q_E(+(x,y)) \to E(q_M(x), E'(+, q_E(y))) \\
q_M(x) \to M(q_F(x), M'(\varepsilon)) & q_M(\times(x,y)) \to M(q_F(x), M'(\times, q_M(y))) \\
q_F(x) \to F((, q_E(x),)) & q_F(a) \to F(I(a)) \\
q_F(b) \to F(I(b)) & q_F(c) \to F(I(c))
\end{array}
$$

Transducer A' is nondeterministic because there are ε-rules like $q_E(x) \to E(q_M(x), E'(\varepsilon))$. We give in Figure 6.6 an example of run of A' which transforms abstract tree $+(a, \times(b, c))$ into a syntactical tree t' of the word $a + b \times c$.

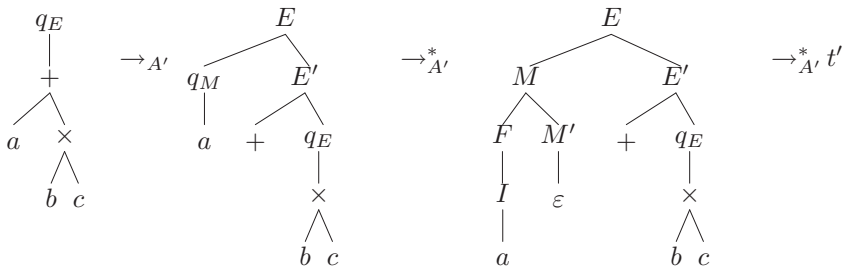


Figure 6.6: Example of run of A'

Compilation

The compiler now transforms abstract trees into programs for some target languages. We consider two target languages. The first one is L_1 of Example 54. To simplify, we omit ";", because they are not necessary — we introduced semicolons in Section 6.2 to avoid ε-rules, but this is a technical detail, because word (and tree) automata with ε-rules are equivalent to usual ones. The second target language is an other very simple language L_2, namely sequences of two instructions $+(i, j, k)$ (put the sum of contents of registers i and j in the register k) and $\times(i, j, k)$. In a first pass, we attribute to each node of the abstract tree the minimal number of registers necessary to compute the corresponding subexpression in the target language. The second pass generates target programs.

First pass: computation of register numbers by **a deterministic linear bottom-up transducer** R.

States of a tree automaton can be considered as values of (finitely valued) attributes, but formalism of tree automata does not allow decorating nodes of trees with the corresponding values. On the other hand, this decoration is easy with a transducer. Computation of finitely valued inherited (respectively synthesized) attributes is modeled by top-down (respectively bottom-up) tree transducers. Here, we use a bottom-up tree transducer R. States of R are $q_0, \ldots, q_n$. All states are final states. The set of rules

is:

$$\begin{array}{rr} a \to q_0(a) & b \to q_0(b) \\ c \to q_0(c) & \\ +(q_i(x), q_i(y)) \to q_{i+1}({}^{+}_{i+1}(x,y)) & \times(q_i(x), q_i(y)) \to q_{i+1}({}^{+}_{i}1 \times (x,y)) \\ \text{if } i > j & \\ +(q_i(x), q_j(y)) \to q_i({}^{+}_{i}(x,y) & \times(q_i(x), q_j(y)) \to q_i({}^{\times}_{i}(x,y)) \\ \text{if } i < j\text{, we permute the order of subtrees} & \\ +(q_i(x), q_j(y)) \to q_j({}^{+}_{j}(y,x)) & \times(q_i(x), q_j(y)) \to q_j({}^{\times}_{j}(y,x)) \end{array}$$

A run $t \to^*_R q_i(u)$ means that i registers are necessary to evaluate t. Root of t is then relabelled in u by symbol ${}^{+}_{i}$ or ${}^{\times}_{i}$.

Second pass: generation of target programs in L_1 or L_2, by **top-down deterministic transducers T_1 and T_2.** T_1 contains only one state q. Set of rules of T_1 is:

$$\begin{array}{r} q({}^{+}_{i}(x,y)) \to \diamond(q(x), \texttt{STORE}i, q(y), \texttt{ADD}i, \texttt{STORE}i) \\ q({}^{\times}_{i}(x,y)) \to \diamond(q(x), \texttt{STORE}i, q(y), \texttt{MULT}i, \texttt{STORE}i) \\ q(a) \to \diamond(\texttt{LOAD}, a) \\ q(b) \to \diamond(\texttt{LOAD}, b) \\ q(c) \to \diamond(\texttt{LOAD}, c) \end{array}$$

where $\diamond(,,,,)$ and $\diamond(,)$ are new symbols.

State set of T_2 is $\{q, q'\}$ where q' is the initial state. Set of rules of T_2 is:

$$\begin{array}{rr} q({}^{+}_{i}(x,y)) \to \#(q(x), q(y), +, (, q'(x), q'(y), i,)) & q'({}^{+}_{i}(x,y)) \to i \\ q({}^{\times}_{i}(x,y)) \to \#(q(x), q(y), \times, (, q'(x), q'(y), i,)) & q'({}^{\times}_{i}(x,y)) \to i \\ q(a) \to \varepsilon & q'(a) \to a \\ q(b) \to \varepsilon & q'(b) \to b \\ q(c) \to \varepsilon & q'(c) \to c \end{array}$$

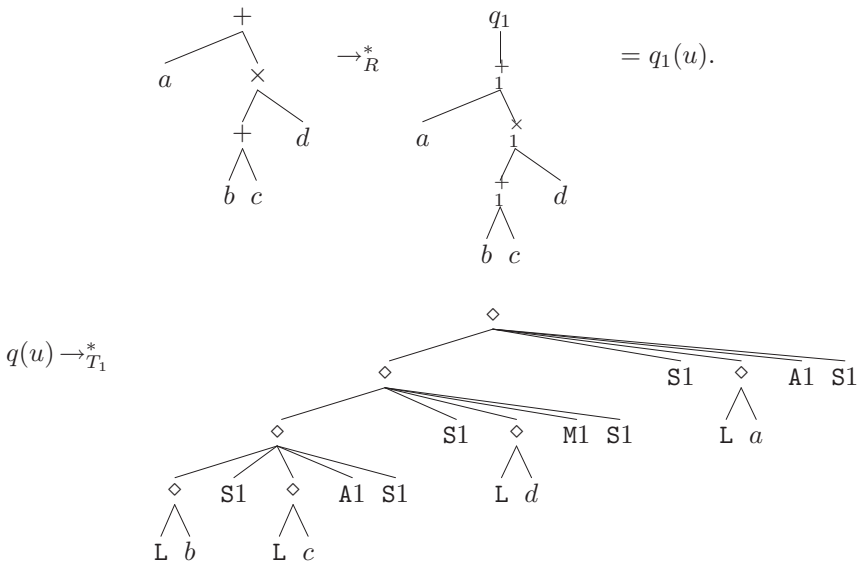
where # is a new symbol of arity 8.

The reader should note that target programs are words formed with leaves of trees, *i.e.*yields of trees. Examples of transductions computed by T_1 and T_2 are given in Figures 6.7 and 6.8. The reader should also note that T_1 is an homomorphism. Indeed, an homomorphism can be considered as a particular case of deterministic transducer, namely a transducer with only one state (we can consider it as bottom-up as well as top-down). The reader should also note that T_2 is deterministic but not linear.

6.4 Properties of Tree Transducers

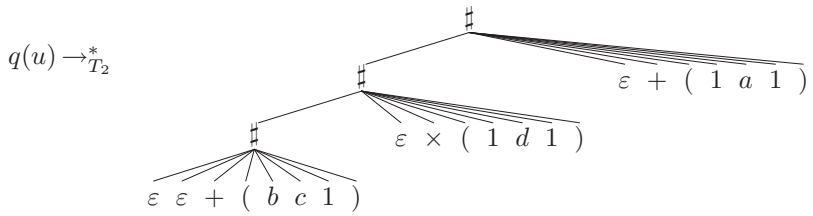
6.4.1 Bottom-up Tree Transducers

We now give formal definitions. In this section, we consider academic examples, without intuitive semantic, to illustrate phenomena and properties. Tree transducers are both generalization of word transducers and tree automata. We first



where `L` stands for `LOAD`, `S` stands for `STORE`, `A` stands for `ADD`, `M` stands for `MULT`. The corresponding program is the yield of this tree:
`LOAD`b `STORE1` `LOAD`c `ADD1` `STORE1` `STORE1` `LOAD`d `MULT1` `STORE1` `STORE1` `LOAD`a `ADD1` `STORE1`

Figure 6.7: Decoration with synthesized attributes of an abstract tree, and translation into a target program of L_1.



The corresponding program is the yield of this tree: $+(bc1) \times (1d1) + (1a1)$

Figure 6.8: Translation of an abstract tree into a target program of L_2

consider bottom-up tree transducers. A transition rule of a NFTA is of the type $f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n))$. Here we extend the definition (as we did in the word case), accepting to change symbol f into any term.

A ***bottom-up Tree Transducer*** (NUTT) is a tuple $U = (Q, \mathcal{F}, \mathcal{F}', Q_f, \Delta)$ where Q is a set of (unary) states, $\mathcal{F}$ and $\mathcal{F}'$ are finite nonempty sets of input symbols and output symbols, $Q_f \subseteq Q$ is a set of final states and Δ is a set of transduction rules of the following two types:

$$f(q_1(x_1), \ldots, q_n(x_n)) \to q(u) \ ,$$

where $f \in \mathcal{F}_n$, $u \in T(\mathcal{F}', \mathcal{X}_n)$, $q, q_1, \ldots, q_n \in Q$, or

$$q(x_1) \to q'(u) \quad (\varepsilon\text{-rule}),$$

where $u \in T(\mathcal{F}', \mathcal{X}_1)$, $q, q' \in Q$.

As for NFTA, there is no initial state, because when a symbol is a leave a (*i.e.*a constant symbol), transduction rules are of the form $a \to q(u)$, where u is a ground term. These rules can be considered as "initial rules". Let $t, t' \in T(\mathcal{F} \cup \mathcal{F}' \cup Q)$. The move relation $\to_U$ is defined by:

$$t \underset{U}{\to} t' \Leftrightarrow \left\{ \begin{array}{l} \exists f(q_1(x_1), \ldots, q_n(x_n)) \to q(u) \in \Delta \\ \exists C \in \mathcal{C}(\mathcal{F} \cup \mathcal{F}' \cup Q) \\ \exists u_1, \ldots, u_n \in T(\mathcal{F}') \\ t = C[f(q_1(u_1), \ldots, q_n(u_n))] \\ t' = C[q(u\{x_1 \leftarrow u_1, \ldots, x_n \leftarrow u_n\})] \end{array} \right.$$

This definition includes the case of ε-rule as a particular case. The reflexive and transitive closure of $\to_U$ is $\to_U^*$. A transduction of U from a ground term $t \in T(\mathcal{F})$ to a ground term $t' \in T(\mathcal{F}')$ is a sequence of move steps of the form $t \to_U^* q(t')$, such that q is a final state. The relation induced by U is the relation (also denoted by U) defined by:

$$U = \{(t, t') \mid t \underset{U}{\overset{*}{\to}} q(t'), t \in T(\mathcal{F}), t' \in T(\mathcal{F}'), q \in Q_f\}.$$

The domain of U is the set $\{t \in T(\mathcal{F}) \mid (t, t') \in U\}$. The image by U of a set of ground terms L is the set $U(L) = \{t' \in T(\mathcal{F}') \mid \exists t \in L, (t, t') \in U\}$.

A transducer is ***ε-free*** if it contains no ε-rule. It is ***linear*** if all transition rules are linear (no variable occurs twice in the right-hand side). It is ***non-erasing*** if, for every rule, at least one symbol of $\mathcal{F}'$ occurs in the right-hand side. It is said to be ***complete*** (or non-deleting) if, for every rule $f(q_1(x_1), \ldots, q_n(x_n)) \to q(u)$, for every $x_i (1 \leq i \leq n)$, x_i occurs at least once in u. It is ***deterministic*** (DUTT) if it is ε-free and there is no two rules with the same left-hand side.

Example 57.

Ex. 57.1 Tree transducer A defined in Section 6.3 is a linear DUTT. Tree transducer R in Section 6.3 is a linear and complete DUTT.

Ex. 57.2 States of U_1 are q, q'; $\mathcal{F} = \{f(), a\}$; $\mathcal{F}' = \{g(,), f(), f'(), a\}$; q' is the final state; the set of transduction rules is:

$$\begin{array}{c} a \to q(a) \\ f(q(x)) \to q(f(x)) \mid q(f'(x)) \mid q'(g(x, x)) \end{array}$$

U_1 is a complete, non linear NUTT. We now give the transductions of the ground term $f(f(f(a)))$. For the sake of simplicity, $fffa$ stands for $f(f(f(a)))$. We have:

$$U_1(\{fffa\}) = \{g(ffa, ffa), g(ff'a, ff'a), g(f'fa, f'fa), g(f'f'a, f'f'a)\}.$$

U_1 illustrates an ability of NUTT, that we describe following Gécseg and Steinby.

B1- "Nprocess and copy" A NUTT can first process an input subtree nondeterministically and then make copies of the resulting output tree.

Ex. 57.3 States of U_2 are q, q'; $\mathcal{F} = \mathcal{F}' = \{f(), f'(), a\}$; q is the final state; the set of transduction rules is defined by:

$$\begin{aligned} a &\to q(a) \\ f(q(x)) &\to q'(a) \\ f'(q'(x)) &\to q(a) \end{aligned}$$

U_2 is a non complete DUTT. The tree transformation induced by U_2 is

$$\left\{ (t, a) \mid \begin{array}{l} t \text{ is accepted by the DFTA of final state } q \text{ and rules} \\ a \to q(a), f(q(x)) \to q'(f(x)), f'(q'(x)) \to q(f'(x)) \end{array} \right\}.$$

B2- "check and delete" A NUTT can first check regular constraints on input subterms and delete these subterms afterwards.

Bottom-up tree transducers translate the input trees from leaves to root, so bottom-up tree transducers are also called frontier-to-root transducers. Top-down tree transducers work in opposite direction.

6.4.2 Top-down Tree Transducers

A ***top-down Tree Transducer*** (NDTT) is a tuple $D = (Q, \mathcal{F}, \mathcal{F}', Q_i, \Delta)$ where Q is a set of (unary) states, $\mathcal{F}$ and $\mathcal{F}'$ are finite nonempty sets of input symbols and output symbols, $Q_i \subseteq Q$ is a set of initial states and Δ is a set of transduction rules of the following two types:

$$q(f(x_1, \ldots, x_n)) \to u[q_1(x_{i_1}), \ldots, q_p(x_{i_p})] \ ,$$

where $f \in \mathcal{F}_n$, $u \in \mathcal{C}^p(\mathcal{F}')$, $q, q_1, \ldots, q_p \in Q$, , $x_{i_1}, \ldots, x_{i_p} \in \mathcal{X}_n$, or

$$q(x) \to u[q_1(x), \ldots, q_p(x)] \quad (\varepsilon\text{-rule}),$$

where $u \in \mathcal{C}^p(\mathcal{F}')$, $q, q_1, \ldots, q_p \in Q$, $x \in \mathcal{X}$.

As for top-down NFTA, there is no final state, because when a symbol is a leave a (*i.e.*a constant symbol), transduction rules are of the form $q(a) \to u$, where u is a ground term. These rules can be considered as "final rules". Let $t, t' \in T(\mathcal{F} \cup \mathcal{F}' \cup Q)$. The move relation $\to_D$ is defined by:

$$t \underset{D}{\rightarrow} t' \Leftrightarrow \begin{cases} \exists q(f(x_1,\ldots,x_n)) \rightarrow u[q_1(x_{i_1}),\ldots,q_p(x_{i_p})] \in \Delta \\ \exists C \in \mathcal{C}(\mathcal{F} \cup \mathcal{F}' \cup Q) \\ \exists u_1,\ldots,u_n \in T(\mathcal{F}) \\ t = C[q(f(u_1,\ldots,u_n))] \\ t' = C[u[q_1(v_1),\ldots,q_p(v_p)]] \text{ where } v_j = u_k \text{ if } x_{i_j} = x_k \end{cases}$$

This definition includes the case of ε-rule as a particular case. $\rightarrow_D^*$ is the reflexive and transitive closure of $\rightarrow_D$. A transduction of D from a ground term $t \in T(\mathcal{F})$ to a ground term $t' \in T(\mathcal{F}')$ is a sequence of move steps of the form $q(t) \rightarrow_D^* t'$, where q is an initial state. The transformation induced by D is the relation (also denoted by D) defined by:

$$D = \{(t,t') \mid q(t) \underset{D}{\overset{*}{\rightarrow}} t', t \in T(\mathcal{F}), t' \in T(\mathcal{F}'), q \in Q_i\}.$$

The domain of D is the set $\{t \in T(\mathcal{F}) \mid (t,t') \in D\}$. The image of a set of ground terms L by D is the set $D(L) = \{t' \in T(\mathcal{F}') \mid \exists t \in L, (t,t') \in D\}$. ε-free, linear, non-erasing, complete (or non-deleting), deterministic top-down tree transducers are defined as in the bottom-up case.

Example 58.

Ex. 58.1 Tree transducers A', T_1, T_2 defined in Section 6.3 are examples of NDTT.

Ex. 58.2 Let us now define a non-deterministic and non linear NDTT D_1. States of D_1 are q, q'. The set of input symbols is $\mathcal{F} = \{f(), a\}$. The set of output symbols is $\mathcal{F}' = \{g(,), f(), f'(), a\}$. The initial state is q. The set of transduction rules is:

$$\begin{array}{rl} q(f(x)) \rightarrow g(q'(x), q'(x)) & \text{(copying rule)} \\ q'(f(x)) \rightarrow f(q'(x)) \mid f'(q'(x)) & \text{(non deterministic relabeling)} \\ q'(a) \rightarrow a & \end{array}$$

D_1 transduces $f(f(f(a)))$ (or briefly $fffa$) into the set of 16 trees:

$\{g(ffa,ffa), g(ffa,ff'a), g(ffa,f'fa), \ldots, g(f'f'a,f'fa), g(f'f'a,f'f'a)\}$.

D_1 illustrates a new property.

> *D- "copy and Nprocess" A NDTT can first make copies of an input subtree and then process different copies independently and nondeterministically .*

6.4.3 Structural Properties

In this section, we use tree transducers U_1, U_2 and D_1 of the previous section in order to point out differences between top-down and bottom-up tree transducers.

Theorem 46 (Comparison Theorem).

1. *There is no top-down tree transducer equivalent to U_1 or to U_2.*
2. *There is no bottom-up tree transducer equivalent to D_1.*
3. *Any linear top-down tree transducer is equivalent to a linear bottom-up tree transducer. In the linear complete case, classes of bottom-up and top-down tree transducers are equal.*

It is not hard to verify that neither NUTT nor NDTT are closed under composition. Therefore, comparison of D-property "copy and Nprocess" and U-property "Nprocess and copy" suggests an important question:

> *does alternation of copying and non-determinism induces an infinite hierarchy of transformations?*

The answer is affirmative [Eng78, Eng82], but it was a relatively long-standing open problem. The fact that top-down transducers copy before non-deterministic processes, and bottom-up transducers copy after non-deterministic processes (see Exercise 75) suggests too that we get by composition two intricate infinite hierarchies of transformation. The following theorem summarizes results.

Theorem 47 (Hierarchy theorem). *By composition of NUTT, we get an infinite hierarchy of transformations. Any composition of n NUTT can be processed by composition of n+1 NDTT, and conversely (i.e.any composition of n NDTT can be processed by composition of $n+1$ NUTT).*

Transducer A' of Section 6.3 shows that it can be useful to consider ε-rules, but usual definitions of tree transducers in literature exclude this case of non determinism. This does not matter, because it is easy to check that all important results of closure or non-closure hold simultaneously for general classes and ε-free classes. Deleting is also a minor phenomenon. Indeed, it gives rise to the "check and delete" property, which is specific to bottom-up transducers, but it does not matter for hierarchy theorem, which remains true if we consider complete transducers.

Section 6.3 suggests that for practical use, non-determinism and non-linearity are rare. Therefore, it is important to note than if we assume linearity or determinism, hierarchy of Theorem 48 collapses. Following results supply algorithms to compose or simplify transducers.

Theorem 48 (Composition Theorem).

1. *The class of linear bottom-up transductions is closed under composition.*
2. *The class of deterministic bottom-up transductions is closed under composition.*
3. *The class of linear top-down transductions is included in the class of linear bottom-up transductions. These classes are equivalent in the complete case.*

4. *Any composition of deterministic top-down transductions is equivalent to a deterministic complete top-down transduction composed with a linear homomorphism.*

The reader should note that bottom-up determinism and top-down determinism are incomparable (see Exercise 72).

Recognizable tree languages play a crucial role because derivation trees of context-free word grammars are recognizable. Fortunately, we get:

Theorem 49 (Recognizability Theorem). *The domain of a tree transducer is a recognizable tree language. The image of a recognizable tree language by a linear tree transducer is recognizable.*

6.4.4 Complexity Properties

We present now some decidability and complexity results. As for structural properties, the situation is more complicated than in the word case, especially for top-down tree transducers. Most of problems are untractable in the worst case, but empirically "not so much complex" in real cases, though there is a lake of "algorithmic engineering" to get performant algorithms. As in the word case, emptiness is decidable, and equivalence in undecidable in the general case but is decidable in the k-valued case (a transducer is k-valued if there is no tree which is transduced in more than k different terms; so a deterministic transducer is a particular case of 1-valued transducer).

Theorem 50 (Recidability and complexity). *Emptiness of tree transductions is decidable. Equivalence of k-valued tree transducers is decidable.*

Emptiness for bottom-up transducers is essentially the same as emptiness for tree automata and therefore PTIME complete. Emptiness for top-down automata, however, is essentially the same as emptiness for alternating topdown tree automata, giving DEXPTIME completeness for emptiness. The complexity PTIME for testing single-valuedness in the bottom-up case is contained in Seidl [Sei92]. Ramsey theory gives combinatorial properties onto which equivalence tests for k-valued tree transducers [Sei94a].

Theorem 51 (Equivalence Theorem). *Equivalence of deterministic tree transducers is decidable.*

6.5 Homomorphisms and Tree Transducers

Exercise 74 illustrates how decomposition of transducers using homomorphisms can help to get composition results, but we are far from the nice bimorphism theorem of the word case, and in the tree case, there is no illuminating theorem, but many complicated partial statements. Seminal paper of Engelfriet [Eng75] contains a lot of decomposition and composition theorems. Here, we only present the most significant results.

A delabeling is a linear, complete, and symbol-to-symbol tree homomorphism (see Section 1.4). This very special kind of homomorphism changes only the label of the input letter and possibly order of subtrees. Definition of tree

bimorphisms is not necessary, it is the same as in the word case. We get the following characterization theorem. We say that a bimorphism is linear, (respectively complete, etc) if the two morphisms are linear, (respectively complete, etc).

Theorem 52. *The class of bottom-up tree transductions is equivalent to the class of bimorphisms* (Φ, L, Ψ) *where* Φ *is a delabeling.*

Relation defined by (Φ, L, Ψ) *is computed by a transduction which is linear (respectively complete,* ε*-free) if* Ψ *is linear (respectively complete,* ε*-free).*

Remark that Nivat Theorem illuminates the symmetry of word transductions: the inverse relation of a rational transduction is a rational transduction. In the tree case, non-linearity obviously breaks this symmetry, because a tree transducer can copy an input tree and process several copies, but it can never check equality of subtrees of an input tree. If we want to consider symmetric relations, we have two main situations. In the non-linear case, it is easy to prove that composition of two bimorphisms simulates a Turing machine. In the linear and the linear complete cases, we get the following results.

Theorem 53 (Tree Bimorphisms). .

1. *The class* LCFB *of linear complete* ε*-free tree bimorphisms satisfies* $\mathsf{LCFB} \subset \mathsf{LCFB}^2 = \mathsf{LCFB}^3$.

2. *The class* LB *of linear tree bimorphisms satisfies* $\mathsf{LB} \subset \mathsf{LB}^2 \subset \mathsf{LB}^3 \subset \mathsf{LB}^4 = \mathsf{LB}^5$.

Proof of $\mathsf{LCFB}^2 = \mathsf{LCFB}^3$ requires many refinements and we omit it.

To prove $\mathsf{LCFB} \subset \mathsf{LCFB}^2$ we use twice the same homomorphism $\Phi(a) = a, \Phi(f(x)) = f(x), \Phi(g(x,y)) = g(x,y), \Phi(h(x,y,z)) = g(x, g(y,z))$.

For any subterms $(t_1, \ldots, t_{2p+2})$, let

$$t = h(t_1, t_2, h(t_3, t_4, h(t_{2i+1}, t_{2i+2}, \ldots, h(t_{2p-1}, t_{2p}, g(t_{2p+1}, t_{2p+2}) \ldots)))$$

and

$$t' = g(t_1, h(t_2, t_3, h(t_4, \ldots, h(t_{2i}, t_{2i+1}, h(t_{2i+2}, t_{2i+3}, \ldots, h(t_{2p}, t_{2p+1}, t_{2p+2}) \ldots)))).$$

We get $t' \in (\Phi \circ \Phi^{-1})(t)$. Assume that $\Phi \circ \Phi^{-1}$ can be processed by some $\Psi^{-1} \circ \Psi'$. Consider for simplicity subterms t_i of kind $f^{ni}(a)$. Roughly, if lengths of t_i are different enough, Ψ and Ψ' must be supposed linear complete. Suppose that for some u we have $\Psi(u) = t$ and $\Psi'(u) = t'$, then for any context u' of u, $\Psi(u')$ is a context of t with an odd number of variables, and $\Psi'(u')$ is a context of t' with an even number of variables. That is impossible because homomorphisms are linear complete.

Point 2 is a refinement of point 1 (see Exercise 79).

This example shows a stronger fact: the relation cannot be processed by any bimorphism, even non-linear, nor by any bottom-up transducer A direct characterization of these transformations is given in [AD82] by a special class of top-down tree transducers, which are not linear but are "globally" linear, and which are used to prove $\mathsf{LCFB}^2 = \mathsf{LCFB}^3$.

6.6 Exercises

Exercises 65 to 71 are devoted to the word case, which is out of scoop of this book. For this reason, we give precise hints for them.

Exercise 65. *The class of rational transductions is closed under rational operations.* Hint: for closure under union, connect a new initial state to initial state with $(\varepsilon, \varepsilon)$-rules (parallel composition). For concatenation, connect by the same way final states of the first transducer to initial states of the second (serial composition). For iteration, connect final states to initial states (loop operation).

Exercise 66. *The class of rational transductions is not closed under intersection.* Hint: consider rational transductions $\{(a^n b^p, a^n) \mid n, p \in \mathbb{N}\}$ and $\{(a^n b^p, a^p) \mid n, p \in \mathbb{N}\}$.

Exercise 67. *Equivalence of rational transductions is undecidable.* Hint: Associate the transduction $T_P = \{(f(u), g(u)) \mid u \in \Sigma^+$ with each instance $P = (f, g)$ of the Post correspondance Problem such that T_P defines $\{(\Phi(m), \Psi(m)) \mid m \in \Sigma^*\}$. Consider *Diff* of example 55.2. *Diff* $\neq$ *Diff* $\cup\, T_P$ if and only if P satisfies Post property.

Exercise 68. *Equivalence of deterministic rational transductions is decidable.* Hint: design a pumping lemma to reduce the problem to a bounded one by suppression of loops (if difference of lengths between two transduced subwords is not bounded, two transducers cannot be equivalent).

Exercise 69. *Build a rational transducer equivalent to a bimorphism.* Hint: let $f(q) \to q'(f)$ a transition rule of L. If $\Phi(f) = \varepsilon$, introduce transduction rule $\varepsilon(q) \to q'(\Psi(f))$. If $\Phi(f) = a_0 \dots a_n$, introduce new states $q_1, \dots, q_n$ and transduction rules $a_0(q) \to q_1(\varepsilon), \dots a_i(q_i) \to q_{i+1}(\varepsilon), \dots a_n(q_n) \to q'(\Psi(f))$.

Exercise 70. *Build a bimorphism equivalent to a rational transducer.* Hint: consider the set Δ of transition rules as a new alphabet. We may speak of the first state q and the second state q' in a letter "$f(q) \to q'(m)$". The control language L is the set of words over this alphabet, such that (i) the first state of the first letter is initial (ii) the second state of the last letter is final (iii) in every two consecutive letters of a word, the first state of the second equals the second state of the first. We define Φ and Ψ by $\Phi(f(q) -> q'(m)) = f$ and $\Psi(f(q) -> q'(m)) = m$.

Exercise 71. *Homomorphism inversion and applications.* An homomorphism Φ is non-increasing if for every symbol a, $\Phi(a)$ is the empty word or a symbol.

1. For any morphism Φ, find a bimorphism (Φ', L, Ψ) equivalent to Φ^{-1}, with Φ' non-increasing, and such that furthermore Φ' is ε-free if Φ is ε-free. Hint: Φ^{-1} is equivalent to a transducer R (Exercise 69), and the output homomorphism Φ' associated to R as in Exercise 70 is non-increasing. Furthermore, if Φ is ε-free, R and Φ' are ε-free.
2. Let Φ and Ψ two homomorphism. If Φ is non-increasing, build a transducer equivalent to $\Psi \circ \Phi^{-1}$ (recall that this notation means that we apply Ψ before Φ^{-1}). Hint and remark: as Φ is non-increasing, Φ^{-1} satisfies the inverse homomorphism property $\Phi^{-1}(MM') = \Phi^{-1}(M)\Phi^{-1}(M')$ (for any pair of words or languages M and M'). This property can be used to do constructions "symbol by symbol". Here, it suffices that the transducer associates $\Phi^{-1}(\Psi(a))$ with a, for every symbol a of the domain of Ψ.
3. Application: prove that classes of regular and context-free languages are closed under bimorphisms (we admit that intersection of a regular language with a regular or context-free language, is respectively regular or context-free).

4. Other application: prove that bimorphisms are closed under composition. Hint: remark that for any application f and set E, $\{(x, f(x)) \mid f(x) \in E\} = \{(x, f(x)) \mid x \in f^{-1}(E)\}$.

Exercise 72. We identify words with trees over symbols of arity 1 or 0. Let relations $U = \{(f^n a, f^n a) \mid n \in \mathbb{N}\} \cup \{(f^n b, g^n b) \mid n \in \mathbb{N}\}$ and $D = \{(ff^n a, ff^n a) \mid n \in \mathbb{N}\} \cup \{(gf^n a, gf^n b) \mid n \in \mathbb{N}\}$. Prove that U is a deterministic linear complete bottom-up transduction but not a deterministic top-down transduction. Prove that D is a deterministic linear complete top-down transduction but not a deterministic bottom-up transduction.

Exercise 73. Prove point 3 of Comparison Theorem. Hint. Use rule-by-rule techniques as in Exercise 74.

Exercise 74. Prove Composition Theorem. Hints: Prove 1 and 2 using composition "rule-by-rule", illustrated as following. States of $A \circ B$ are products of states of A and states of B. Let $f(q(x)) \to_A q'(g(x, g(x, a)))$ and $g(q_1(x), g(q_2(y), a) \to_B q_4(u)$. Subterms substituted to x and y in the composition must be equal, and determinism implies $q_1 = q_2$. Then we build new rule $f((q, q1)(x)) \to_{A \circ B} (q', q_4)(u)$. To prove 3 for example, associate $q(g(x, y)) \to u(q'(x), q''(y))$ with $g(q'(x), q''(y)) \to q(u)$, and conversely. For 4, Using ad hoc kinds of "rule-by-rule" constructions, prove $\mathsf{DDTT} \subset \mathsf{DCDTT} \circ \mathsf{LHOM}$ and $\mathsf{LHOM} \circ \mathsf{DCDTT} \subset \mathsf{DCDTT} \circ \mathsf{LHOM}$ (L means linear, C complete, D deterministic - and suffix DTT means top-down tree transducer as usually).

Exercise 75. Prove $\mathsf{NDTT} = \mathsf{HOM} \circ \mathsf{NLDTT}$ and $\mathsf{NUTT} = \mathsf{HOM} \circ \mathsf{NLBTT}$. Hint: to prove $\mathsf{NDTT} \subset \mathsf{HOM} \circ \mathsf{NLDTT}$ use a homomorphism H to produce in advance as may copies of subtrees of the input tree as the NDTT may need, ant then simulate it by a linear NDTT.

Exercise 76. Use constructions of composition theorem to reduce the number of passes in process of Section 6.3.

Exercise 77. Prove recognizability theorem. Hint: as in exercise 74, "naive" constructions work.

Exercise 78. Prove Theorem 52. Hint: "naive" constructions work.

Exercise 79. Prove point 2 of Theorem 53. Hint: E denote the class of homomorphisms which are linear and symbol-to-symbol. L, LC, LCF denotes linear, linear complete, linear complete ε-free homomorphisms, respectively. Prove $\mathsf{LCS} = \mathsf{L} \circ \mathsf{E} = \mathsf{E} \circ \mathsf{L}$ and $\mathsf{E}^{-1} \circ \mathsf{L} \subset \mathsf{L} \circ \mathsf{E}^{-1}$. Deduce from these properties and from point 1 of Theorem 53 that $\mathsf{LB}^4 = \mathsf{E} \circ \mathsf{LCFB}^2 \circ \mathsf{E}^{-1}$. To prove that $\mathsf{LB}^3 \neq \mathsf{LB}^4$, consider $\Psi_1 \circ \Psi_2^{-1} \circ \Phi \circ \Phi^{-1} \circ \Psi_2 \circ \Psi_1^{-1}$, where Φ is the homomorphism used in point 1 of Theorem 53; Ψ_1 identity on a, $f(x)$, $g(x, y)$, $h(x, y, z)$, $\Psi_1(e(x)) = x$; Ψ_2 identity on a, $f(x)$, $g(x, y)$ and $\Psi_2(c(x, y, z) = b(b(x, y), z)$.

Exercise 80. Sketch of proof of $\mathsf{LCFB}^2 = \mathsf{LCFB}^3$ (difficult). Distance $D(x, y, u)$ of two nodes x and y in a tree u is the sum of the lengths of two branches which join x and y to their younger common ancestor in u. $D(x, u)$ denotes the distance of x to the root of u.

Let H the class of deterministic top-down transducers T defined as follows: $q_0, \ldots, q_n$ are states of the transducer, q_0 is the initial state. For every context, consider the result u_i of the run starting from $q_i(u)$. $\exists k, \forall$ context u such that for every variable x of u, $D(x, u) > k$:

- u_0 contains at least an occurrence of each variable of u,

- for any i, u_i contains at least a non variable symbol,
- if two occurrences x' and x'' of a same variable x occur in u_i, $D(x', x", u_i) < k$.

Remark that LCF is included in H and that there is no right hand side of rule with two occurrences of the same variable associated with the same state. Prove that

1. $\mathsf{LCF}^{-1} \subseteq \mathsf{Delabeling}^{-1} \circ H$
2. $H \circ \mathsf{Delabeling}^{-1} \subseteq \mathsf{Delabeling}^{-1} \circ H$
3. $H \subseteq \mathsf{LCFB}^2$
4. Conclude. Compare with Exercise 71

Exercise 81. Prove that the image of a recognizable tree language by a linear tree transducer is recognizable.

6.7 Bibliographic notes

First of all, let us precise that several surveys have been devoted (at least in part) to tree transducers for 25 years. J.W. Thatcher [Tha73], one of the main pioneer, did the first one in 1973, and F. Gécseg and M. Steinby the last one in 1996 [GS96]. Transducers are formally studied too in the book of F. Gécseg and M. Steinby [GS84] and in the survey of J.-C. Raoult [Rao92]. Survey of M. Dauchet and S. Tison [DT92] develops links with homomorphisms.

In section 6.2, some examples are inspired by the old survey of Thatcher, because seminal motivation remain, namely modelization of compilers or, more generally, of syntax directed transformations as interfacing softwares, which are always up to date. Among main precursors, we can distinguish Thatcher [Tha73], W.S. Brainerd [Bra69], A. Aho, J.D. Ullman [AU71], M. A. Arbib, E. G. Manes [AM78]. First approaches where very linked to practice of compilation, and in some way, present tree transducers are evolutions of generalized syntax directed translations (B.S. Backer [Bak78] for example), which translate trees into strings. But crucial role of tree structure have increased later.

Many generalizations have been introduced, for example generalized finite state transformations which generalize both the top-down and the bottom-up tree transducers (J. Engelfriet [Eng77]); modular tree transducers (H. Vogler [EV91]); synchronized tree automata (K. Salomaa [Sal94]); alternating tree automata (G.Slutzki [Slu85]); deterministic top-down tree transducers with iterated look-ahead (G. Slutzki, S. Vàgvölgyi [SV95]). Ground tree transducers GTT are studied in Chapter 3 of this book. The first and the most natural generalization was introduction of top-down tree transducers with look-ahead. We have seen that "check and delete" property is specific to bottom-up tree transducers, and that missing of this property in the non-complete top-down case induces non closure under composition, even in the linear case (see Composition Theorem). Top-down transducers with regular look-ahead are able to recognize before the application of a rule at a node of an input tree whether the subtree at a son of this node belongs to a given recognizable tree language. This definition remains simple and gives to top-down transducers a property equivalent to "check and delete".

Contribution of Engelfriet to the theory of tree transducers is important, especially for composition, decomposition and hierarchy main results ([Eng75, Eng78, Eng82]).

We did not many discuss complexity and decidability in this chapter, because the situation is classical. Since many problems are undecidable in the word case, they are obviously undecidable in the tree case. Equivalence decidability holds as in the word case for deterministic or finite-valued tree transducers (Z. Zachar [Zac79], Z. Esik [Esi83], H. Seidl [Sei92, Sei94a]).

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Index