TREE AUTOMATA AND TREE GRAMMARS

by
Joost Engelfriet

Institute of Mathematics, University of Aarhus
DEPARTMENT OF COMPUTER SCIENCE
Ny Munkegade, 8000 Aarhus C, Denmark

Preface

I wrote these lecture notes during my stay in Aarhus in the academic year 1974/75. As a young researcher I had a wonderful time at DAIMI, and I have always been happy to have had that early experience.

I wish to thank Heiko Vogler for his noble plan to move these notes into the digital world, and I am grateful to Florian Starke and Markus Napierkowski (and Heiko) for the excellent transformation of my hand-written manuscript into LaTeX.

Apart from the reparation of errors and some cosmetical changes, the text of the lecture notes has not been changed. Of course, many things have happened in tree language theory since 1975. In particular, most of the problems mentioned in these notes have been solved. The developments until 1984 are described in the book "Tree Automata" by Ferenc Gécseg and Magnus Steinby, and for recent developments I recommend the Appendix of the reissue of that book at arXiv.org/abs/1509.06233.

Joost Engelfriet, October 2015

LIACS, Leiden University,
The Netherlands

Tree automata and tree grammars

To appreciate the theory of tree automata and tree grammars one should already be motivated by the goals and results of formal language theory. In particular one should be interested in "derivation trees". A derivation tree models the grammatical structure of a sentence in a (context-free) language. By considering only the bottom of the tree the sentence may be recovered from the tree.

The first idea in tree language theory is to generalize the notion of a finite automaton working on strings to that of a finite automaton operating on trees. It turns out that a large part of the theory of regular languages can rather easily be generalized to a theory of regular tree languages. Moreover, since a regular tree language is (almost) the same as the set of derivation trees of some context-free language, one obtains results about context-free languages by "taking the bottom" of results about regular tree languages.

The second idea in tree language theory is to generalize the notion of a generalized sequential machine (that is, finite automaton with output) to that of a finite state tree transducer. Tree transducers are more complicated than string transducers since they are equipped with the basic capabilities of copying, deleting and reordering (of subtrees). The part of (tree) language theory that is concerned with translation of languages is mainly motivated by compiler writing (and, to a lesser extent, by natural linguistics). When considering bottoms of trees, finite state transducers are essentially the same as syntax-directed translation schemes. Results in this part of tree language theory treat the composition and decomposition of tree transformations, and the properties of those tree languages that can be obtained by finite state transformation of regular tree languages (or, taking bottoms, those languages that can be obtained by syntax-directed translation of context-free languages).

Thirdly there are, of course, many other ideas in tree language theory. In the literature one can find, for instance, context-free tree grammars, recognition of subsets of arbitrary algebras, tree walking automata, hierarchies of tree languages (obtained by iterating old ideas), decomposition of tree automata, Lindenmayer tree grammars, etc.

These lectures will be divided in the following five parts: (1) and (2) contain preliminaries, (3), (4) and (5) are the main parts.

(1) Introduction. (p. 1)

(2) Some basic definitions. (p. 2)

(3) Recognizable (= regular) tree languages. (p. 10)

(4) Finite state tree transformations. (p. 32)

(5) Whatever there is more to consider.

Part (5) is not contained in these notes; instead, some Notes on the literature are given on p. 69.
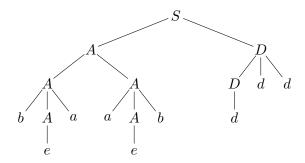
Contents

1 Introduction

Our basic data type is the kind of tree used to express the grammatical structure of strings in a context-free language.

Example 1.1. Consider the context-free grammar $G = (N, \Sigma, R, S)$ with nonterminals $N = \{S, A, D\}$, terminals $\Sigma = \{a, b, d\}$, initial nonterminal S and the set of rules R, consisting of the rules $S \to AD$, $A \to aAb$, $A \to bAa$, $A \to AA$, $A \to \lambda$, $D \to Ddd$ and $D \to d$ (we use λ to denote the empty string).

The string $baabddd \in \Sigma^*$ can be generated by G and has the following derivation tree (see [Sal, II.6], [A&U, 0.5 and 2.4.1]):

- Note that we use e as a symbol standing for the empty string λ.
- The string $baabddd$ is called the "yield" or "result" of the derivation tree. □

Thus, in graph terminology, our trees are finite (finite number of nodes and branches), directed (the branches are "growing downwards"), rooted (there is a node, the root, with no branches entering it), ordered (the branches leaving a node are ordered from left to right) and labeled (the nodes are labeled with symbols from some alphabet).

The following intuitive terminology will be used:

- the <u>rank</u> (or out-degree) of a node is the number of branches leaving it (note that the in-degree of a node is always 1, except for the root which has in-degree 0)
- a <u>leaf</u> is a node with rank 0
- the <u>top</u> of a tree is its root
- the <u>bottom</u> (or frontier) of a tree is the set (or sequence) of its leaves
- the <u>yield</u> (or result, or frontier) of a tree is the string obtained by writing the labels of its leaves (except the label e) from left to right
- a <u>path</u> through a tree is a sequence of nodes connected by branches ("leading downwards"); the <u>length</u> of the path is the number of its nodes minus one (that is, the number of its branches)
- the <u>height</u> (or depth) of a tree is the length of the longest path from the top to the bottom

- if there is a path of length ≥ 1 (of length $= 1$) from a node a to a node b then b is a <u>descendant</u> (<u>direct descendant</u>) of a and a is an <u>ancestor</u> (<u>direct ancestor</u>) of b
- a <u>subtree</u> of a tree is a tree determined by a node together with all its descendants; a <u>direct subtree</u> is a subtree determined by a direct descendant of the root of the tree; note that each tree is uniquely determined by the label of its root and the (possibly empty) sequence of its direct subtrees
- the phrases "bottom-up", "bottom-to-top" and "frontier-to-root" are used to indicate this $\uparrow$ direction, while the phrases "top-down", "top-to-bottom" and "root-to-frontier" are used to indicate that $\downarrow$ direction.

In derivation trees of context-free grammars each symbol may only label nodes of certain ranks. For instance, in the above example, a, b, d and e may only label leaves (nodes of rank 0), A labels nodes with ranks 1, 2 and 3, S labels nodes with rank 2, and D nodes of rank 1 and 3 (these numbers being the lengths of the right hand sides of rules).

Therefore, given some alphabet, we require the specification of a finite number of ranks for each symbol in the alphabet, and we restrict attention to those trees in which nodes of rank k are labeled by symbols of rank k.

2 Some basic definitions

The mathematical definition of a tree may be given in several, equivalent, ways. We will define a <u>tree as a special kind of string</u> (others call this string a representation of the tree, see [A&U, 0.5.7]). Before doing so, let us define ranked alphabets.

Definition 2.1. An alphabet Σ is said to be <u>ranked</u> if for each nonnegative integer k a subset Σ_k of Σ is specified, such that Σ_k is nonempty for a finite number of k's only, and such that $\Sigma = \bigcup_{k \geq 0} \Sigma_k$. [†]

If $a \in \Sigma_k$, then we say that a has rank k (note that a may have more than one rank).

Usually we define a specific ranked alphabet Σ by specifying those Σ_k that are nonempty. $\qquad \square$

Example 2.2. The alphabet $\Sigma = \{a, b, +, -\}$ is made into a ranked alphabet by specifying $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{-\}$ and $\Sigma_2 = \{+, -\}$. (Think of negation and subtraction). $\qquad \square$

Remark 2.3. Throughout our discussions we shall use the symbol e as a special symbol, intuitively representing λ. Whenever e belongs to a ranked alphabet, it is of rank 0. $\qquad \square$

Operations on ranked alphabets should be defined as for instance in the following definition.

[†]To be more precise one should define a ranked alphabet as a pair (Σ, f), where Σ is an alphabet and f is a mapping from $\mathbb{N}$ into $\mathcal{P}(\Sigma)$ such that $\exists n \; \forall k \geq n : f(k) = \emptyset$, and then denote $f(k)$ by Σ_k and (Σ, f) by Σ. Note that $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers and that $\mathcal{P}(\Sigma)$ is the set of subsets of Σ.
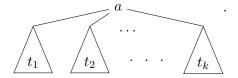
Definition 2.4. Let Σ and Δ be ranked alphabets. The <u>union</u> of Σ and Δ, denoted by $\Sigma \cup \Delta$, is defined by $(\Sigma \cup \Delta)_k = \Sigma_k \cup \Delta_k$, for all $k \geq 0$. We say that Σ and Δ are <u>equal</u>, denoted by $\Sigma = \Delta$, if, for all $k \geq 0$, $\Sigma_k = \Delta_k$. $\qquad\square$

We now define the notion of tree. Let "[" and "]" be two symbols which are never elements of a ranked alphabet.

Definition 2.5. Given a ranked alphabet Σ, the set of <u>trees over Σ</u>, denoted by T_Σ, is the language over the alphabet $\Sigma \cup \{[,]\}$ defined inductively as follows.

(i) If $a \in \Sigma_0$, then $a \in T_\Sigma$.

(ii) For $k \geq 1$, if $a \in \Sigma_k$ and $t_1, t_2, \ldots, t_k \in T_\Sigma$, then $a[t_1 t_2 \cdots t_k] \in T_\Sigma$. $\qquad\square$

Intuitively, a is a tree with one node labeled "a", and $a[t_1 t_2 \cdots t_k]$ is the tree

Example 2.6. Consider the ranked alphabet of Example 2.2. Then $+[-[a-[b]]a]$ is a tree over this alphabet, intuitively "representing" the tree

which on its turn "represents" the expression $(a - (-b)) + a$ (note that the "official" tree is the prefix notation of this expression). $\qquad\square$

Example 2.7. Consider the ranked alphabet Δ, where $\Delta_0 = \{a, b, d, e\}$, $\Delta_1 = \Delta_3 = \{A, D\}$ and $\Delta_2 = \{A, S\}$. A picture of the tree

$$S[A[A[bA[e]a]A[aA[e]b]]D[D[d]dd]]$$

in T_Δ is given in Example 1.1. $\qquad\square$

Exercise 2.8. Take some ranked alphabet Σ and show that T_Σ is a context-free language over $\Sigma \cup \{[,]\}$. $\qquad\square$

Our main aim will be to study several ways of constructively representing sets of trees and relations between trees. The basic terminology is the following.

Definition 2.9. Let Σ be a ranked alphabet. A <u>tree language</u> over Σ is any subset of T_Σ. $\qquad\square$

Definition 2.10. Let Σ and Δ be ranked alphabets. A <u>tree transformation</u> from T_Σ into T_Δ is any subset of $T_\Sigma \times T_\Delta$. $\qquad\square$

Exercise 2.11. Show that the context-free grammar $G = (N, \Sigma, R, S)$ with $N = \{S\}$, $\Sigma = \{a, b, [,]\}$ and $R = \{S \to b[aS], S \to a\}$ generates a tree language over Δ, where $\Delta_0 = \{a\}$ and $\Delta_2 = \{b\}$. $\qquad\square$

The above definition of "tree" (Definition 2.5) gives rise to the following principles of proof by induction and definition by induction for trees. (Note that each tree is, uniquely, either in Σ_0 or of the form $a[t_1 \cdots t_k]$).

Principle 2.12. Principle of proof by induction (or recursion) on trees. Let P be a property of trees (over Σ).

If (i) all elements of Σ_0 have property P, and

 (ii) for each $k \geq 1$ and each $a \in \Sigma_k$, if $t_1, \ldots, t_k$ have property P, then $a[t_1 \cdots t_k]$ has property P,

then all trees in T_Σ have property P. $\qquad\square$

Principle 2.13. Principle of definition by induction (or recursion) on trees. Suppose we want to associate a value $h(t)$ with each tree t in T_Σ. Then it suffices to define $h(a)$ for all $a \in \Sigma_0$, and to show how to compute the value $h(a[t_1 \cdots t_k])$ from the values $h(t_1), \ldots, h(t_k)$. More formally expressed,

given a set O of objects, and

 (i) for each $a \in \Sigma_0$, an object $o_a \in O$, and

 (ii) for each $k \geq 1$ and each $a \in \Sigma_k$, a mapping $f_a^k : O^k \to O$,

there is exactly one mapping $h : T_\Sigma \to O$ such that

 (i) $h(a) = o_a$ for all $a \in \Sigma_0$, and

 (ii) $h(a[t_1 \cdots t_k]) = f_a^k(h(t_1), \ldots, h(t_k))$ for all $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$. $\qquad\square$

Example 2.14. Let $\Sigma_0 = \{e\}$ and $\Sigma_1 = \{/\}$. The trees in T_Σ are in an obvious one-to-one correspondence with the natural numbers. The above principles are the usual induction principles for these numbers. $\qquad\square$

To illustrate the use of the induction principles we give the following useful definitions.

Definition 2.15. The mapping <u>yield</u> from T_Σ into Σ_0^* is defined inductively as follows.

 (i) For $a \in \Sigma_0$, $\mathrm{yield}(a) = \begin{cases} a & \text{if } a \neq e \\ \lambda & \text{if } a = e. \end{cases}$

 (ii) For $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
 $\mathrm{yield}(a[t_1 \cdots t_k]) = \mathrm{yield}(t_1) \cdot \mathrm{yield}(t_2) \cdots \mathrm{yield}(t_k).$ [†]

[†]That is, the concatenation of $\mathrm{yield}(t_1), \ldots, \mathrm{yield}(t_k)$.

Moreover, for a tree language $L \subseteq T_\Sigma$, we define

$$\text{yield}(L) = \{\text{yield}(t) \mid t \in L\}.$$

We shall sometimes abbreviate "yield" by "y". □

Definition 2.16. The mapping <u>height</u> from T_Σ into $\mathbb{N}$ is defined recursively as follows.
   (i) For $a \in \Sigma_0$, height$(a) = 0$.
   (ii) For $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
   height$(a[t_1 \cdots t_k]) = \max\limits_{1 \leq i \leq k} (\text{height}(t_i)) + 1$. □

Example 2.17. As an example of a proof by induction on trees we show that, if $e \notin \Sigma_0$ and $\Sigma_1 = \emptyset$, then, for all $t \in T_\Sigma$, height$(t) < |\text{yield}(t)|$.

Proof. For $a \in \Sigma_0$, height$(a) = 0$ and $|\text{yield}(a)| = |a| = 1$ (since $a \neq e$). Now let $a \in \Sigma_k$ ($k \geq 2$) and assume (induction hypothesis) that height$(t_i) < |\text{yield}(t_i)|$ for $1 \leq i \leq k$.

$$
\begin{aligned}
\text{Then} \quad |\text{yield}(a[t_1 \cdots t_k])| &= \sum_{i=1}^{k} |\text{yield}(t_i)| && \text{(Def. 2.15(ii))} \\
&\geq \left(\sum_{i=1}^{k} \text{height}(t_i)\right) + k && \text{(ind. hypothesis)} \\
&\geq \left(\max_{1 \leq i \leq k} \text{height}(t_i)\right) + 2 && (k \geq 2 \text{ and height}(t_i) \geq 0) \\
&> \text{height}(a[t_1 \cdots t_k]) && \text{(Def. 2.16(ii))}.
\end{aligned}
$$

□

Exercise 2.18. Let Σ be a ranked alphabet such that $\Sigma_0 \cap \Sigma_k = \emptyset$ for all $k \geq 1$. Define a (string) homomorphism h from $(\Sigma \cup \{[,]\})^*$ into Σ_0^* such that, for all $t \in T_\Sigma$, $h(t) = \text{yield}(t)$. □

Exercise 2.19. Give a recursive definition of the notion of "subtree", for instance as a mapping sub : $T_\Sigma \to \mathcal{P}(T_\Sigma)$ such that sub(t) is the set of all subtrees of t. Give also an alternative definition of "subtree" in a more string-like fashion. □

Exercise 2.20. Let path(t) denote the set of all paths from the top of t to its bottom. Think of a formal definition for "path". □

The generalization of formal language theory to a formal tree language theory will come about by viewing a <u>string as a special kind of tree</u> and taking the obvious generalizations. To be able to view strings as trees we "turn them 90 degrees" to a vertical position, as follows.

Definition 2.21. A ranked alphabet Σ is <u>monadic</u> if
   (i) $\Sigma_0 = \{e\}$, and
   (ii) for $k \geq 2$, $\Sigma_k = \emptyset$.
The elements of T_Σ are called monadic trees. □

Thus a monadic ranked alphabet Σ is fully determined by the alphabet Σ_1. Monadic trees obviously can be made to correspond to the strings in Σ_1^*. There are two ways to do this, depending on whether we read top-down or bottom-up:

$f_{\mathrm{td}} : T_\Sigma \to \Sigma_1^*$ is defined by

      (i) $f_{\mathrm{td}}(e) = \lambda$

      (ii) $f_{\mathrm{td}}(a[t]) = a \cdot f_{\mathrm{td}}(t)$ for $a \in \Sigma_1$ and $t \in T_\Sigma$

and $f_{\mathrm{bu}} : T_\Sigma \to \Sigma_1^*$ is defined by

      (i) $f_{\mathrm{bu}}(e) = \lambda$

      (ii) $f_{\mathrm{bu}}(a[t]) = f_{\mathrm{bu}}(t) \cdot a$ for $a \in \Sigma_1$ and $t \in T_\Sigma$.

(Obviously both f_{td} and f_{bu} are bijections).

Accordingly, when generalizing a string-concept to trees, we often have the choice between a top-down and a bottom-up generalization.

Example 2.22. The string alphabet $\Delta = \{a, b, c\}$ corresponds to the monadic alphabet Σ with $\Sigma_0 = \{e\}$ and $\Sigma_1 = \Delta$. The tree

$$
\begin{array}{c}
a \\
| \\
b \\
| \\
c \\
| \\
b \\
| \\
e
\end{array}
$$

in T_Σ corresponds either to the string $abcb$ in Δ^* (top-down), or to the string $bcba$ in Δ^* (bottom-up). Note that, due to our "prefix definition" of trees (Definition 2.5), the above tree looks "top-down like" in its official form $a[b[c[b[e]]]]$. Obviously this is not essential. $\qquad\square$

Let us consider some basic operations on trees. A basic operation on strings is right-concatenation with one symbol (that is, for each symbol a in the alphabet there is an operation rc_a such that, for each string w, $\mathrm{rc}_a(w) = wa$). Every string can uniquely be built up from the empty string by these basic operations (consider the way you write and read!). Generalizing bottom-up, the corresponding basic operations on trees, here called "top concatenation", are the following.

Definition 2.23. For each $a \in \Sigma_k$ ($k \geq 1$) we define the (k-ary) operation of <u>top concatenation</u> with a, denoted by tc_a^k, to be the mapping from T_Σ^k into T_Σ such that, for all $t_1, \ldots, t_k \in T_\Sigma$,

$$\mathrm{tc}_a^k(t_1, \ldots, t_k) = a[t_1 \cdots t_k].$$

Moreover, for tree languages $L_1, \ldots, L_k$, we define

$$\mathrm{tc}_a^k(L_1, \ldots, L_k) = \{a[t_1 \cdots t_k] \mid t_i \in L_i \text{ for all } 1 \leq i \leq k\}.$$

$\square$

Note that every tree can uniquely be built up from the elements of Σ_0 by repeated top concatenation.

The next basic operation on strings is concatenation. When viewed monadically, concatenation corresponds to substituting one vertical string into the e of the other vertical string. In the general case, we may take one tree and substitute a tree into each leaf of the original tree, such that different trees may be substituted into leaves with different labels. Thus we obtain the following basic operation on trees.

Definition 2.24. Let $n \geq 1$, $a_1, \ldots, a_n \in \Sigma_0$ all different, and $s_1, \ldots, s_n \in T_\Sigma$. For $t \in T_\Sigma$, the <u>tree concatenation</u> of t with $s_1, \ldots, s_n$ at $a_1, \ldots, a_n$, denoted by $t\langle a_1 \leftarrow s_1, \ldots, a_n \leftarrow s_n\rangle$, is defined recursively as follows.

(i) for $a \in \Sigma_0$,
$$a\langle a_1 \leftarrow s_1, \ldots, a_n \leftarrow s_n\rangle = \begin{cases} s_i & \text{if } a = a_i \\ a & \text{otherwise} \end{cases}$$

(ii) for $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
$$a[t_1 \cdots t_k]\langle \ldots \rangle = a[t_1\langle \ldots \rangle \cdots t_k\langle \ldots \rangle],$$
where $\langle \ldots \rangle$ abbreviates $\langle a_1 \leftarrow s_1, \ldots, a_n \leftarrow s_n\rangle$.

If, in particular, $n = 1$, then, for each $a \in \Sigma_0$ and $t, s \in T_\Sigma$, the tree $t\langle a \leftarrow s\rangle$ is also denoted by $t \cdot_a s$. $\square$

Example 2.25. Let $\Delta_0 = \{x, y, c\}$, $\Delta_2 = \{b\}$ and $\Delta_3 = \{a\}$. If $t = a[b[xy]xc]$, then $t\langle x \leftarrow b[cx], y \leftarrow c\rangle = a[b[b[cx]c]b[cx]c]$. $\square$

Exercise 2.26. Check that in the monadic case tree concatenation corresponds to string concatenation. $\square$

For tree languages tree concatenation is defined analogously.

Definition 2.27. Let $n \geq 1$, $a_1, \ldots, a_n \in \Sigma_0$ all different, and $L_1, \ldots, L_n \subseteq T_\Sigma$. For $L \subseteq T_\Sigma$ we define the <u>tree concatenation</u> of L with $L_1, \ldots, L_n$ at $a_1, \ldots, a_n$, denoted by $L\langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n\rangle$, as follows. [†]

(i) for $a \in \Sigma_0$,
$$a\langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n\rangle = \begin{cases} L_i & \text{if } a = a_i \\ a & \text{otherwise} \end{cases}$$

(ii) for $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
$$a[t_1 \cdots t_k]\langle \ldots \rangle = a[t_1\langle \ldots \rangle \cdots t_k\langle \ldots \rangle] \quad \text{[††]}$$

(iii) for $L \subseteq T_\Sigma$,
$$L\langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n\rangle = \bigcup_{t \in L} t\langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n\rangle.$$

[†]As usual, given a string w, we use w also to denote the language $\{w\}$.

[††]For tree languages $M_1, \ldots, M_k$ we also write $a[M_1 \cdots M_k]$ to denote $\mathrm{tc}_a^k(M_1, \ldots, M_k)$. This notation is fully justified since $a[M_1 \cdots M_k]$ is the (string) concatenation of the languages a, $[$, $M_1, \ldots, M_k$ and $]$!

If, in particular, $n = 1$, then, for each $a \in \Sigma_0$ and each $L_1, L_2 \subseteq T_\Sigma$, we denote $L_1 \langle a \leftarrow L_2 \rangle$ also by $L_1 \cdot_a L_2$. $\qquad \square$

Remarks 2.28.

(1) Obviously, if $L, L_1, \ldots, L_n$ are singletons, then Definition 2.27 is the same as Definition 2.24.

(2) Note that tree concatenation, as defined above, is "nondeterministic" in the sense that, for instance, to obtain $t \langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n \rangle$ different elements of L_1 may be substituted at different occurrences of a_1 in t. "Deterministic" tree concatenation of t with $L_1, \ldots, L_n$ at $a_1, \ldots, a_n$ could be defined as $\{ t \langle a_1 \leftarrow s_1, \ldots, a_n \leftarrow s_n \rangle \mid s_i \in L_i \text{ for all } 1 \leq i \leq n \}$. In this case different occurrences of a_1 in t should be replaced by the same element of L_1. It is clear that, in the case that $L_1, \ldots, L_n$ are singletons, this distinction cannot be made. $\qquad \square$

Intuitively, since trees are strings, tree concatenation is nothing else but ordinary string substitution, familiar from formal language theory (see, for instance, [Sal, I.3]).

For completeness we give the definition of substitution of string languages.

Definition 2.29. Let Δ be an alphabet. Let $n \geq 1$, $a_1, \ldots, a_n \in \Delta$ all different and let $L_1, \ldots, L_n$ be languages over Δ. For any $L \subseteq \Delta^*$, the <u>substitution</u> of $L_1, \ldots, L_n$ for $a_1, \ldots, a_n$ in L, denoted by $L \langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n \rangle$, is the language over Δ defined as follows:

(i) $\lambda \langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n \rangle = \lambda$

(ii) for $a \in \Delta$,
$$a \langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n \rangle = \begin{cases} L_i & \text{if } a = a_i \\ a & \text{otherwise} \end{cases}$$

(iii) for $w \in \Delta^*$ and $a \in \Delta$,
$$wa \langle \ldots \rangle = w \langle \ldots \rangle \cdot a \langle \ldots \rangle$$

(iv) for $L \subseteq \Delta^*$,
$$L \langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n \rangle = \bigcup_{w \in L} w \langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n \rangle.$$

If $n = 1$, $L_1 \langle a \leftarrow L_2 \rangle$ will also be denoted as $L_1 \cdot_a L_2$. If $L_1, \ldots, L_n$ are singletons, then the substitution is called a <u>homomorphism</u>. $\qquad \square$

Exercise 2.30. Let $n \geq 1$, $a_1, \ldots, a_n \in \Sigma_0$ all different, $a_i \neq e$ for all $1 \leq i \leq n$, and $L, L_1, \ldots L_n \subseteq T_\Sigma$. Prove that

$$\text{yield}(L \langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n \rangle) = \text{yield}(L) \langle a_1 \leftarrow \text{yield}(L_1), \ldots, a_n \leftarrow \text{yield}(L_n) \rangle.$$

(Thus: "yield of tree concatenation is string substitution of yields"). $\qquad \square$

Exercise 2.31. Prove that Definitions 2.27 and 2.29 give exactly the same result for $L \langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n \rangle$ where $a_1, \ldots, a_n \in \Sigma_0$ and $L, L_1, \ldots L_n$ are tree languages over Σ (and thus, string languages over $\Sigma \cup \{ [,] \}$). $\qquad \square$

Exercise 2.32. Define the notion of associativity for tree concatenation, and show that tree concatenation is associative. Show that, in general, "deterministic tree concatenation" is not associative (cf. Remark 2.28(2)). □

We shall need the following special case of tree concatenation.

Definition 2.33. Let Σ be a ranked alphabet and let S be a set of symbols or a tree language. Then the set of <u>trees indexed by S</u>, denoted by $T_\Sigma(S)$, is defined inductively as follows.

(i) $S \cup \Sigma_0 \subseteq T_\Sigma(S)$

(ii) If $k \geq 1$, $a \in \Sigma_k$ and $t_1, \dots, t_k \in T_\Sigma(S)$, then $a[t_1 \cdots t_k] \in T_\Sigma(S)$.
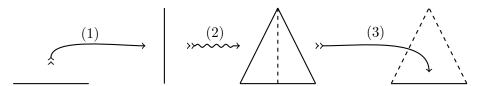
Note that $T_\Sigma(\emptyset) = T_\Sigma$. □

Thus, if S is a set of symbols, then $T_\Sigma(S) = T_{\Sigma \cup S}$, where the elements of S are assumed to have rank 0. If S is a tree language over a ranked alphabet Δ, then $T_\Sigma(S)$ is a tree language over the ranked alphabet $\Sigma \cup \Delta$.

Exercise 2.34. Show that, for any $a \in \Sigma_0$, $T_\Sigma(S) = T_\Sigma \cdot_a (S \cup \{a\})$. □

We close this section with two general remarks.

Remark 2.35. Definition 2.5 of a tree is of course rather arbitrary. Other, equally useful, ways of defining trees as a special kind of strings are obtained by replacing $a[t_1 \cdots t_k]$ in Definition 2.5 by $[t_1 \cdots t_k]a$ or $at_1 \cdots t_k$ or $[at_1 \cdots t_k]$ or $at_1 \cdots t_k$ (only in the case that each symbol has exactly one rank) or $\underset{a}{[} t_1 \cdots t_k \underset{a}{]}$ (where $[$ is a new symbol for each a) or $a[t_1, t_2, \dots, t_k]$ (where "," is a new symbol). □

Remark 2.36. Remark on the general philosophy in tree language theory. The general philosophy looks like this:

(1) Take vertical string language theory (cf. Definition 2.21),

(2) generalize it to tree language theory, and

(3) map this into horizontal string language theory via the yield operation (Definition 2.15).

The fourth part of the philosophy is

(4) Tree language theory is a specific part of string language theory, illustrated as follows:

Example:

(1). (vertical) string concatenation

(2). tree concatenation

(3). (horizontal) string substitution   (see Exercise 2.30)

(4). (2) is a special case of (3)   (see Exercise 2.31)

$\square$

3 Recognizable tree languages

3.1 Finite tree automata and regular tree grammars

Let us first consider the usual finite automaton on strings. A deterministic finite automaton is a structure $M = (Q, \Sigma, \delta, q_0, F)$, where Q is the set of states, Σ is the input alphabet, q_0 is the initial state, F is the set of final states and δ is a family $\{\delta_a\}_{a \in \Sigma}$, where $\delta_a : Q \to Q$ is the transition function for the input a. There are several ways to describe the functioning of M and the language it recognizes. One of them (see for instance [Sal, I.4]), is to describe explicitly the sequence of steps taken by the automaton while processing some input string. This point of view will be considered in Part (4). Another way is to give a recursive definition of the effect of an input string on the state of M. Since a recursive definition is in particular suitable for generalization to trees, let us consider one in detail. We define a function $\widehat{\delta} : \Sigma^* \to Q$ such that, for $w \in \Sigma^*$, $\widehat{\delta}(w)$ is intuitively the state M reaches after processing w, starting from the initial state q_0:

(i) $\widehat{\delta}(\lambda) = q_0$

(ii) for $w \in \Sigma^*$ and $a \in \Sigma$, $\widehat{\delta}(wa) = \delta_a(\widehat{\delta}(w))$.

The language recognized by M is $L(M) = \{w \in \Sigma^* \mid \widehat{\delta}(w) \in F\}$. When considering this definition of $\widehat{\delta}$ for "bottom-up" monadic trees (see Definition 2.21), one easily arrives at the following generalization to the tree case:

There should be a start state for each element of Σ_0. The finite tree automaton starts at all leaves ("at the same time", "in parallel") and processes the tree in a bottom-up fashion. The automaton arrives at each node of rank k with a sequence of k states (one state for each direct subtree of the node), and the transition function δ_a of the label a of that node is a mapping $\delta_a : Q^k \to Q$, which, from that sequence of k states, determines

the state at that node. A tree is recognized iff the tree automaton is in a final state at the root of the tree. Formally:

Definition 3.1. A <u>deterministic bottom-up finite tree automaton</u> is a structure $M = (Q, \Sigma, \delta, s, F)$,

where $\quad Q \quad$ is a finite set (of <u>states</u>),

$\quad\quad \Sigma \quad$ is a ranked alphabet (of <u>input symbols</u>),

$\quad\quad \delta \quad$ is a family $\{\delta_a^k\}_{k \geq 1, a \in \Sigma_k}$ of mappings $\delta_a^k : Q^k \to Q$ (the <u>transition function</u> for $a \in \Sigma_k$),

$\quad\quad s \quad$ is a family $\{s_a\}_{a \in \Sigma_0}$ of states $s_a \in Q$ (the <u>initial state</u> for $a \in \Sigma_0$), and

$\quad\quad F \quad$ is a subset of Q (the set of <u>final states</u>).

The mapping $\widehat{\delta} : T_\Sigma \to Q$ is defined recursively as follows:

(i) for $a \in \Sigma_0$, $\widehat{\delta}(a) = s_a$,

(ii) for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
$$\widehat{\delta}(a[t_1 \cdots t_k]) = \delta_a^k(\widehat{\delta}(t_1), \ldots, \widehat{\delta}(t_k)).$$

The <u>tree language recognized by M</u> is defined to be $L(M) = \{t \in T_\Sigma \mid \widehat{\delta}(t) \in F\}$. $\quad\square$

Intuitively, $\widehat{\delta}(t)$ is the state reached by M after bottom-up processing of t.
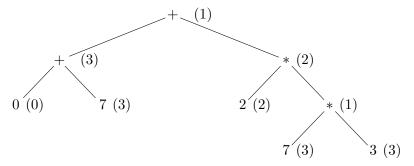
For convenience, when k is understood, we shall write δ_a rather than δ_a^k. Note therefore that each symbol $a \in \Sigma$ may have several transition functions δ_a (one for each of its ranks).

We shall abbreviate "finite tree automaton" by "fta", and "deterministic" by "det.".

Definition 3.2. A tree language L is called <u>recognizable</u> (or regular) if $L = L(M)$ for some det. bottom-up fta M.

The class of recognizable tree languages will be denoted by *RECOG*. $\quad\square$

Example 3.3. Let us consider the det. bottom-up fta $M = (Q, \Sigma, \delta, s, F)$, where $Q = \{0, 1, 2, 3\}$, $\Sigma_0 = \{0, 1, 2, \ldots, 9\}$, $\Sigma_2 = \{+, *\}$, $s_a \equiv a \pmod 4$, $F = \{1\}$, and δ_+ and δ_* (both mappings $Q^2 \to Q$) are addition modulo 4 and multiplication modulo 4 respectively. Then M recognizes the set of all "expressions" whose value modulo 4 is 1. Consider for instance the expression $+[+[07]*[2*[73]]]$, the prefix form of $(0+7)+(2*(7*3))$. In the following picture,

the state of M at each node of the tree is indicated between parentheses. $\quad\square$

11

Example 3.4. Let $\Sigma_0 = \{a\}$ and $\Sigma_2 = \{b\}$. Consider the language of all trees in T_Σ which have a "right comb-like" structure like for instance the tree $b[ab[ab[ab[aa]]]]$. This tree language is recognized by the det. bottom-up fta $M = (Q, \Sigma, \delta, s, F)$, where $Q = \{A, C, W\}$, $s_a = A$, $F = \{C\}$ and δ_b is defined by $\delta_b(A, A) = \delta_b(A, C) = C$ and $\delta_b(q_1, q_2) = W$ for all other pairs of states (q_1, q_2). $\qquad\square$

Exercise 3.5. Let $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{p\}$ and $\Sigma_2 = \{p, q\}$. Construct det. bottom-up finite tree automata recognizing the following tree languages:

  (i) the language of all trees t, such that if a node of t is labeled q, then its descendants are labeled q or a;

 (ii) the set of all trees t such that $\text{yield}(t) \in a^+ b^+$;

(iii) the set of all trees t such that the total number of p's occurring in t is odd. $\qquad\square$

A (theoretically) convenient extension of the deterministic finite automaton is to make it nondeterministic. A nondeterministic finite automaton (on strings) is a structure $M = (Q, \Sigma, \delta, S, F)$, where Q, Σ and F are the same as in the deterministic case, S is a set of initial states, and, for each $a \in \Sigma$, δ_a is a mapping $Q \to \mathcal{P}(Q)$ (intuitively, $\delta_a(q)$ is the set of states which M can possibly, nondeterministically, enter when reading a in state q). Again a mapping $\widehat{\delta}$, now from Σ^* into $\mathcal{P}(Q)$, can be defined, such that for every $w \in \Sigma^*$, $\widehat{\delta}(w)$ is the set of states M can possibly reach after processing w, having started from one of the initial states in S:

  (i) $\widehat{\delta}(\lambda) = S$,

 (ii) for $w \in \Sigma^*$ and $a \in \Sigma$, $\widehat{\delta}(wa) = \bigcup \{\delta_a(q) \mid q \in \widehat{\delta}(w)\}$.

The language recognized by M is $L(M) = \{w \in \Sigma^* \mid \widehat{\delta}(w) \cap F \neq \emptyset\}$.

Generalizing to trees we obtain the following definition.

Definition 3.6. A <u>nondeterministic bottom-up finite tree automaton</u> is a 5-tuple $M = (Q, \Sigma, \delta, S, F)$, where Q, Σ and F are as in the deterministic case, S is a family $\{S_a\}_{a \in \Sigma_0}$ such that $S_a \subseteq Q$ for each $a \in \Sigma_0$, and δ is a family $\{\delta_a^k\}_{k \geq 1, a \in \Sigma_k}$ of mappings $\delta_a^k : Q^k \to \mathcal{P}(Q)$.

The mapping $\widehat{\delta} : T_\Sigma \to \mathcal{P}(Q)$ is defined recursively by

  (i) for $a \in \Sigma_0$, $\widehat{\delta}(a) = S_a$,

 (ii) for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
    $\widehat{\delta}(a[t_1 \cdots t_k]) = \bigcup \{\delta_a(q_1, \ldots, q_k) \mid q_i \in \widehat{\delta}(t_i) \text{ for } 1 \leq i \leq k\}$.

The <u>tree language recognized by M</u> is $L(M) = \{t \in T_\Sigma \mid \widehat{\delta}(t) \cap F \neq \emptyset\}$. $\qquad\square$

Note that, for $\bar{q} \in Q^k$, $\delta_a^k(\bar{q})$ may be empty.

Example 3.7. Let $\Sigma_0 = \{p\}$ and $\Sigma_2 = \{a, b\}$. Consider the following tree language over Σ:
$$
\begin{aligned}
L \;=\; & \{u_1 a[a[s_1 s_2] a[t_1 t_2]] u_2 \in T_\Sigma \mid \cdots\} \cup \\
& \{u_1 b[b[s_1 s_2] b[t_1 t_2]] u_2 \in T_\Sigma \mid \cdots\}
\end{aligned}
$$

where "$\cdots$" stands for $u_1, u_2 \in (\Sigma \cup \{[,]\})^*, s_1, s_2, t_1, t_2 \in T_\Sigma$. In other words, L is the set of all trees containing a configuration

$$
\begin{array}{c}
a \\
/ \; \backslash \\
a \quad a \\
/\backslash \; /\backslash
\end{array}
\quad \text{or a configuration} \quad
\begin{array}{c}
b \\
/ \; \backslash \\
b \quad b \\
/\backslash \; /\backslash
\end{array}
$$

(or both). L is recognized by the nondet. bottom-up fta $M = (Q, \Sigma, \delta, S, F)$, where $Q = \{q_s, q_a, q_b, r\}$, $S_p = \{q_s\}$, $F = \{r\}$ and $\delta_a(q_s, q_s) = \{q_s, q_a\}$, $\delta_b(q_s, q_s) = \{q_s, q_b\}$, $\delta_a(q_a, q_a) = \delta_b(q_b, q_b) = \{r\}$, for all $q \in Q : \delta_a(q, r) = \delta_a(r, q) = \delta_b(q, r) = \delta_b(r, q) = \{r\}$, and $\delta_x(q_1, q_2) = \emptyset$ for all other possibilities. □

It is rather obvious in the last example that we can find a deterministic bottom-up fta recognizing the same language (find it!). We now show that this is possible in general (as in the case of strings).

Theorem 3.8. For each nondeterministic bottom-up fta we can find a deterministic one recognizing the same language.

Proof. The proof uses the "subset-construction", well known from the string-case. Let $M = (Q, \Sigma, \delta, S, F)$ be a nondeterministic bottom-up fta. Construct the deterministic bottom-up fta $M_1 = (\mathcal{P}(Q), \Sigma, \delta_1, s_1, F_1)$ such that $(s_1)_a = S_a$ for all $a \in \Sigma_0$, $F_1 = \{Q_1 \in \mathcal{P}(Q) \mid Q_1 \cap F \neq \emptyset\}$, and, for $a \in \Sigma_k$ and $Q_1, \ldots, Q_k \subseteq Q$,

$$
(\delta_1)_a(Q_1, \ldots, Q_k) = \bigcup \{\delta_a(q_1, \ldots, q_k) \mid q_i \in Q_i \text{ for all } 1 \leq i \leq k\}.
$$

It is straightforward to show, using Definitions 3.1 and 3.6, that $\widehat{\delta_1}(t) = \widehat{\delta}(t)$ for all $t \in T_\Sigma$ (proof by induction on t). From this it follows that $L(M_1) = \{t \mid \widehat{\delta_1}(t) \in F_1\} = \{t \mid \widehat{\delta}(t) \cap F \neq \emptyset\} = L(M)$. □

Exercise 3.9. Check the proof of Theorem 3.8. Construct the det. bottom-up fta corresponding to the fta M of Example 3.7 according to that proof, and compare this det. fta with the one you found before. □

Let us now consider the top-down generalization of the finite automaton. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a det. finite automaton. Another way to define $L(M)$ is by giving a recursive definition of a mapping $\widetilde{\delta} : \Sigma^* \to \mathcal{P}(Q)$ such that intuitively, for each $w \in \Sigma^*$, $\widetilde{\delta}(w)$ is the set of states q such that the machine M, when started in state q, enters a final state after processing w. The definition of $\widetilde{\delta}$ is as follows:

(i) $\widetilde{\delta}(\lambda) = F$

(ii) for $w \in \Sigma^*$ and $a \in \Sigma$,
$\widetilde{\delta}(aw) = \{q \mid \delta_a(q) \in \widetilde{\delta}(w)\}$

(the last line may be read as: to check whether, starting in q, M recognizes aw, compute $q_1 = \delta_a(q)$ and check whether M recognizes w starting in q_1). The language recognized by M is $L(M) = \{w \in \Sigma^* \mid q_0 \in \widetilde{\delta}(w)\}$. This definition, applied to "top-down" monadic trees, leads to the following generalization to arbitrary trees. The finite tree automaton starts at the root of the tree in the initial state, and processes the tree in a top-down

fashion. The automaton arrives at each node in one state, and the transition function δ_a of the label a of that node is a mapping $\delta_a : Q \to Q^k$ (where k is the rank of the node), which, from that state, determines the state in which to continue for each direct descendant of the node (the automaton "splits up" into k independent copies, one for each direct subtree of the node). Finally the automaton arrives at all leaves of the tree. There should be a set of final states for each element of Σ_0. The tree is recognized if the fta arrives at each leaf in a state which is final for the label of that leaf. Formally:

Definition 3.10. A <u>deterministic top-down finite tree automaton</u> is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$,

where Q is a finite set (of <u>states</u>),

         Σ is a ranked alphabet (of <u>input symbols</u>),

         δ is a family $\{\delta_a^k\}_{k \geq 1, a \in \Sigma_k}$ of mappings $\delta_a^k : Q \to Q^k$ (the <u>transition function</u> for $a \in \Sigma_k$),

         q_0 is in Q (the <u>initial state</u>), and

         F is a family $\{F_a\}_{a \in \Sigma_0}$ of sets $F_a \subseteq Q$ (the set of <u>final states</u> for $a \in \Sigma_0$).

The mapping $\widetilde{\delta} : T_\Sigma \to \mathcal{P}(Q)$ is defined recursively by

  (i) for $a \in \Sigma_0$, $\widetilde{\delta}(a) = F_a$

  (ii) for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
$$\widetilde{\delta}(a[t_1 \cdots t_k]) = \{q \mid \delta_a(q) \in \widetilde{\delta}(t_1) \times \cdots \times \widetilde{\delta}(t_k)\}.$$

The <u>tree language recognized by M</u> is defined to be $L(M) = \{t \in T_\Sigma \mid q_0 \in \widetilde{\delta}(t)\}$. $\square$

Intuitively, $\widetilde{\delta}(t)$ is the set of states q such that M, when starting at the root of t in state q, arrives at the leaves of t in final states.

Example 3.11. Consider the tree language of Exercise 3.5(i). A det. top-down fta recognizing this language is $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{A, R, W\}$, $q_0 = A$, $F_a = \{A, R\}$, $F_b = \{A\}$ and

$$
\begin{aligned}
\delta_p^1(A) &= A, & \delta_p^1(R) &= \delta_p^1(W) = W, \\
\delta_p^2(A) &= (A, A), & \delta_p^2(R) &= \delta_p^2(W) = (W, W), \\
\delta_q(A) &= (R, R), & \delta_q(R) &= (R, R), \ \ \delta_q(W) = (W, W).
\end{aligned}
$$
   $\square$

Exercise 3.12. Let Σ be a ranked alphabet, and $p \in \Sigma_2$. Let L be the tree language defined recursively by

  (i) for all $t_1, t_2 \in T_\Sigma$, $p[t_1 t_2] \in L$

  (ii) for all $a \in \Sigma_k$, if $t_1, \ldots, t_k \in L$, then $a[t_1 \cdots t_k] \in L$ ($k \geq 1$).

Construct a deterministic top-down fta recognizing L. Give a nonrecursive description of L. $\square$

Exercise 3.13. Construct a det. top-down fta M such that $\mathrm{yield}(L(M)) = a^+ b^+$. $\square$

We now show that the det. top-down fta recognizes less languages than its bottom-up counterpart.

Theorem 3.14. There are recognizable tree languages which cannot be recognized by a deterministic top-down fta.

Proof. Let $\Sigma_0 = \{a, b\}$ and $\Sigma_2 = \{S\}$. Consider the (finite!) tree language $L = \{S[ab], S[ba]\}$. Suppose that the det. top-down fta $M = (Q, \Sigma, \delta, q_0, F)$ recognizes L. Let $\delta_S(q_0) = (q_1, q_2)$. Since $S[ab] \in L(M)$, $q_1 \in F_a$ and $q_2 \in F_b$. But, since $S[ba] \in L(M)$, $q_1 \in F_b$ and $q_2 \in F_a$. Hence both $S[aa]$ and $S[bb]$ are in $L(M)$. Contradiction. $\square$

Exercise 3.15. Show that the tree languages of Exercise 3.5(ii,iii) are not recognizable by a det. top-down fta. $\square$

It will be clear that the nondeterministic top-down fta is able to recognize all recognizable languages. We give the definition without comment.

Definition 3.16. A <u>nondeterministic top-down finite tree automaton</u> is a structure $M = (Q, \Sigma, \delta, S, F)$, where Q, Σ and F are as in the deterministic case, S is a subset of Q and δ is a family $\{\delta_a^k\}_{k \geq 1, a \in \Sigma_k}$ of mappings $\delta_a^k : Q \to \mathcal{P}(Q^k)$.
The mapping $\widetilde{\delta} : T_\Sigma \to \mathcal{P}(Q)$ is defined recursively as follows

  (i) for $a \in \Sigma_0$, $\widetilde{\delta}(a) = F_a$,

  (ii) for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
     $\widetilde{\delta}(a[t_1 \cdots t_k]) = \{q \mid \exists (q_1, \ldots, q_k) \in \delta_a(q) : q_i \in \widetilde{\delta}(t_i) \text{ for all } 1 \leq i \leq k\}$.

The <u>tree language recognized by M</u> is $L(M) = \{t \in T_\Sigma \mid \widetilde{\delta}(t) \cap S \neq \emptyset\}$. $\square$

We now show that, nondeterministically, there is no difference between bottom-up or top-down recognition.

Theorem 3.17. A tree language is recognizable by a nondet. bottom-up fta iff it is recognizable by a nondet. top-down fta.

Proof. Let us say that a nondet. bottom-up fta $M = (Q, \Sigma, \delta, S, F)$ and a nondet. top-down fta $N = (P, \Delta, \mu, R, G)$ are "associated" if the following requirements are satisfied:

  (i) $Q = P$, $\Sigma = \Delta$, $F = R$ and, for all $a \in \Sigma_0$, $S_a = G_a$;

  (ii) for all $k \geq 1$, $a \in \Sigma_k$ and $q_1, \ldots, q_k, q \in Q$,
     $q \in \delta_a(q_1, \ldots, q_k)$ iff $(q_1, \ldots, q_k) \in \mu_a(q)$.

In that case, one can easily prove by induction that $\widehat{\delta} = \widetilde{\mu}$, and so $L(M) = L(N)$. Since obviously for each nondet. bottom-up fta there is an associated nondet. top-down fta, and vice versa, the theorem holds. $\square$

Thus the classes of tree languages recognized by the nondet. bottom-up, det. bottom-up and nondet. top-down fta are all equal (and are called *RECOG*), whereas the class of tree languages recognized by the det. top-down fta is a proper subclass of *RECOG*.

The next victim of generalization is the regular grammar (right-linear, type-3 grammar). In this case it seems appropriate to take the top-down point of view only. Consider an

ordinary regular grammar $G = (N, \Sigma, R, S)$. All rules have either the form $A \to wB$ or the form $A \to w$, where $A, B \in N$ and $w \in \Sigma^*$. Monadically, the string wB may be considered as the result of treeconcatenating the tree we with B at e, where B is of rank 0. Thus we can take the generalization of strings of the form wB or w to be trees in $T_\Delta(N)$, where Δ is a ranked alphabet (for the definition of $T_\Delta(N)$, see Definition 2.33). Thus, let us consider a "tree grammar" with rules of the form $A \to t$, where $A \in N$ and $t \in T_\Delta(N)$. Obviously, the application of a rule $A \to t$ to a tree $s \in T_\Delta(N)$ should intuitively consist of replacing one occurrence of A in s by the tree t. Starting with the initial nonterminal, nonterminals at the frontier of the tree are then repeatedly replaced by right hand sides of rules, until the tree does not contain nonterminals any more. Now, since trees are defined as strings, it turns out that this process is precisely the way a context-free grammar works. Thus we arrive at the following formal definition.

Definition 3.18. A <u>regular tree grammar</u> is a tuple $G = (N, \Sigma, R, S)$ where N is a finite set (of <u>nonterminals</u>), Σ is a ranked alphabet (of <u>terminals</u>), such that $\Sigma \cap N = \emptyset$, $S \in N$ is the <u>initial nonterminal</u>, and R is a finite set of <u>rules</u> of the form $A \to t$ with $A \in N$ and $t \in T_\Sigma(N)$. The <u>tree language generated by G</u>, denoted by $L(G)$, is defined to be $L(H)$, where H is the context-free grammar $(N, \Sigma \cup \{[,]\}, R, S)$. We shall use $\underset{G}{\Rightarrow}$ and $\underset{G}{\overset{*}{\Rightarrow}}$ (or $\Rightarrow$ and $\overset{*}{\Rightarrow}$ when G is understood) to denote the restrictions of $\underset{H}{\Rightarrow}$ and $\underset{H}{\overset{*}{\Rightarrow}}$ to $T_\Sigma(N)$. $\quad\square$

Example 3.19. Let $\Sigma_0 = \{a, b, c, d, e\}$, $\Sigma_2 = \{p\}$ and $\Sigma_3 = \{p, q\}$. Consider the regular tree grammar $G = (N, \Sigma, R, S)$, where $N = \{S, T\}$ and R consists of the rules $S \to p[aTa]$, $T \to q[cp[dT]b]$ and $T \to e$. Then G generates the tree $p[aq[cp[de]b]a]$ as follows:

$$S \Rightarrow p[aTa] \Rightarrow p[aq[cp[dT]b]a] \Rightarrow p[aq[cp[de]b]a]$$

or, pictorially,

The tree language generated by G is $\{p[a(q[cp[d])^n e(]b])^n a] \mid n \geq 0\}$. $\quad\square$

Exercise 3.20. Write regular tree grammars generating the tree languages of Exercise 3.5. $\quad\square$

As in the case of strings, each regular tree grammar is equivalent to one that has the property that at each step in the derivation exactly one terminal symbol is produced.

Definition 3.21. A regular tree grammar $G = (N, \Sigma, R, S)$ is <u>in normal form</u>, if each of its rules is either of the form $A \to a[B_1 \cdots B_k]$ or of the form $A \to b$, where $k \geq 1$, $a \in \Sigma_k$, $A, B_1, \ldots, B_k \in N$ and $b \in \Sigma_0$. $\quad\square$

Theorem 3.22. Each regular tree grammar has an equivalent regular tree grammar in normal form.

Proof. Consider an arbitrary regular tree grammar $G = (N, \Sigma, R, S)$. Let $G_1 = (N, \Sigma, R_1, S)$ be the regular tree grammar such that $(A \to t) \in R_1$ if and only if $t \notin N$ and there is a B in N such that $A \overset{*}{\underset{G}{\Rightarrow}} B$ and $(B \to t) \in R_1$. Then $L(G_1) = L(G)$, and R_1 does not contain rules of the form $A \to B$ with $A, B \in N$. (This is the well-known procedure of removing rules $A \to B$ from a context-free grammar). Suppose that G_1 is not yet in normal form. Than there is a rule of the form $A \to a[t_1 \cdots t_i \cdots t_k]$ such that $t_i \notin N$. Construct a new regular tree grammar G_2 by adding a new nonterminal B to N and replacing the rule $A \to a[t_1 \cdots t_i \cdots t_k]$ by the two rules $A \to a[t_1 \cdots B \cdots t_k]$ and $B \to t_i$ in R_1. It should be clear that $L(G_2) = L(G_1)$, and that, by repeating the latter process a finite number of times, one ends up with an equivalent grammar in normal form. $\square$

Exercise 3.23. Put the regular tree grammar of Example 3.19 into normal form. $\square$

Exercise 3.24. What does Theorem 3.22 actually say in the case of strings (the monadic case)? $\square$

In the next theorem we show that the regular tree grammars generate exactly the class of recognizable tree languages.

Theorem 3.25. A tree language can be generated by a regular tree grammar iff it is an element of $RECOG$.

Proof. Exercise. $\square$

Note therefore that each recognizable tree language is a special kind of context-free language.

Exercise 3.26. Show that all finite tree languages are in $RECOG$. $\square$

Exercise 3.27. Show that each recognizable tree language can be generated by a "backwards deterministic" regular tree grammar. A regular tree grammar is called "backwards deterministic" if (1) it may have more than one initial nonterminal, (2) it is in normal form, and (3) rules with the same right hand side are equal. $\square$

It is now easy to show the connection between recognizable tree languages and context-free languages. Let CFL denote the class of context-free languages.

Theorem 3.28. yield($RECOG$) = CFL (in words, the yield of each recognizable tree language is context-free, and each context-free language is the yield of some recognizable tree language).

Proof. Let $G = (N, \Sigma, R, S)$ be a regular tree grammar. Consider the context-free grammar $\overline{G} = (N, \Sigma_0, \overline{R}, S)$, where $\overline{R} = \{A \rightarrow \text{yield}(t) \mid A \rightarrow t \in R\}$. Then $L(\overline{G}) = \text{yield}(L(G))$.

Now let $G = (N, \Sigma, R, S)$ be a context-free grammar. Let $*$ be a new symbol, and let $\Delta = \Sigma \cup \{e, *\}$ be the ranked alphabet such that $\Delta_0 = \Sigma \cup \{e\}$, and, for $k \geq 1$, $\Delta_k = \{*\}$ if and only if there is a rule in R with a right hand side of length k. Consider the regular tree grammar $\overline{G} = (N, \Delta, \overline{R}, S)$ such that

(i) if $A \rightarrow w$ is in R, $w \neq \lambda$, then $A \rightarrow *[w]$ is in $\overline{R}$,

(ii) if $A \rightarrow \lambda$ is in R, then $A \rightarrow e$ is in $\overline{R}$.

Then $\text{yield}(L(\overline{G})) = L(G)$. □

In the next section we shall give the connection between regular tree languages and derivation trees of context-free languages.

Exercise 3.29. A context-free grammar is "invertible" if rules with the same right hand side are equal. Show that each context-free language can be generated by an invertible context-free grammar. □

For regular string languages a useful stronger version of Theorem 3.28 can be proved.

Theorem 3.30. Let Σ be a ranked alphabet. If R is a regular string language over Σ_0, then the tree language $\{t \in T_\Sigma \mid \text{yield}(t) \in R\}$ is recognizable.

Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automaton recognizing R. We construct a nondeterministic bottom-up fta $N = (Q \times Q, \Sigma, \mu, S, G)$, which, for each tree t, checks whether a successful computation of M on $\text{yield}(t)$ is possible. The states of N are pairs of states of M. Intuitively we want that $(q_1, q_2) \in \widehat{\mu}(t)$ if and only if M arrives in state q_2 after processing $\text{yield}(t)$, starting from state q_1. Thus we define

(i) for all $a \in \Sigma_0$, $S_a = \{(q_1, q_2) \mid \delta_a(q_1) = q_2\}$,

(ii) for all $k \geq 1$, $a \in \Sigma_k$ and states $q_1, q_2, \ldots, q_{2k} \in Q$,

$$\mu_a((q_1, q_2), (q_3, q_4), \ldots, (q_{2k-1}, q_{2k})) = \begin{cases} \{(q_1, q_{2k})\} & \text{if } q_{2i} = q_{2i+1} \text{ for} \\ & \quad \text{all } 1 \leq i \leq k-1 \\ \emptyset & \text{otherwise .} \end{cases}$$

Then $L(N) = \{t \in T_\Sigma \mid \text{yield}(t) \in R\}$. □

Exercise 3.31. Show that, if $\Sigma_2 \neq \emptyset$, then Theorem 3.30 holds conversely: if L is a string language such that $\{t \in T_\Sigma \mid \text{yield}(t) \in L\}$ is recognizable, then L is regular. What can you say in case $\Sigma_2 = \emptyset$? □

3.2 Closure properties of recognizable tree languages

We first consider set-theoretic operations.

Theorem 3.32. *RECOG* is closed under union, intersection and complementation.

Proof. To show closure under complementation, consider a deterministic bottom-up fta $M = (Q, \Sigma, \delta, s, F)$. Let N be the det. bottom-up fta $(Q, \Sigma, \delta, s, Q - F)$. Then, obviously, $L(N) = T_\Sigma - L(M)$.

To show closure under union, consider two regular tree grammars $G_i = (N_i, \Sigma_i, R_i, S_i)$, $i = 1, 2$ (with $N_1 \cap N_2 = \emptyset$). Then $G = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}, S)$ is a regular tree grammar such that $L(G) = L(G_1) \cup L(G_2)$. $\square$

As a corollary we obtain the following closure property of context-free languages.

Corollary 3.33. *CFL* is closed under intersection with regular languages.

Proof. Let L and R be a context-free and regular language respectively. According to Theorem 3.28, there is a recognizable tree language U such that $\text{yield}(U) = L$. Consequently, by Theorems 3.30 and 3.32, the tree language $V = U \cap \{t \mid \text{yield}(t) \in R\}$ is recognizable. Obviously $L \cap R = \text{yield}(V)$ and so, again by Theorem 3.28, $L \cap R$ is context-free. $\square$

We now turn to the closure of *RECOG* under concatenation operations (see Definitions 2.23 and 2.27).

Theorem 3.34. For every $k \geq 1$ and $a \in \Sigma_k$, *RECOG* is closed under tc_a^k.

Proof. Exercise. $\square$

Theorem 3.35. *RECOG* is closed under tree concatenation.

Proof. The proof is obtained by generalizing that for regular string languages. Let $n \geq 1$, $a_1, \ldots, a_n \in \Sigma_0$ all different and $L_0, L_1, \ldots, L_n$ recognizable tree languages (we may assume that all languages are over the same ranked alphabet Σ). Let $G_i = (N_i, \Sigma, R_i, S_i)$ be a regular tree grammar in normal form for L_i ($i = 0, 1, \ldots, n$). A regular tree grammar generating $L_0 \langle a_1 \leftarrow L_1, \ldots, a_n \leftarrow L_n \rangle$ is $G = (\bigcup_{i=0}^{n} N_i, \Sigma, R, S_0)$, where $R = \overline{R}_0 \cup \bigcup_{i=1}^{n} R_i$, and $\overline{R}_0$ is R_0 with each rule of the form $A \to a_i$ replaced by the rule $A \to S_i$ ($1 \leq i \leq n$). $\square$

Corollary 3.36. *CFL* is closed under substitution.

Proof. Use Theorem 3.28 and Exercise 2.30. $\square$

Note also that Theorem 3.35 is essentially a special case of Corollary 3.36.

Next we generalize the notion of (concatenation) closure of string languages to trees, and show that *RECOG* is closed under this closure operation. We shall, for convenience, restrict ourselves to the case that tree concatenation happens at one element of Σ_0.

Definition 3.37. Let $a \in \Sigma_0$ and let L be a tree language over Σ. Then the <u>tree concatenation closure</u> of L at a, denoted by L^{*a}, is defined to be $\bigcup_{n=0}^{\infty} X_n$, where $X_0 = \{a\}$ and, for $n \geq 0$, $X_{n+1} = X_n \cdot_a (L \cup \{a\})$. [†] □

Example 3.38. Let $G = (N, \Sigma, R, S)$ be the regular tree grammar with $N = \{S\}$, $\Sigma_0 = \{a\}$, $\Sigma_2 = \{b\}$ and $R = \{S \to b[aS], \ S \to a\}$. Then $L(G) = \{b[aS]\}^{*S} \cdot_S a$. □

The "corresponding" operation on strings has several names in the literature. Let us call it "substitution closure".

Definition 3.39. Let Δ be an alphabet and $a \in \Delta$. For a language L over Δ, the <u>substitution closure</u> of L at a, denoted by L^{*a}, is defined to be $\bigcup_{n=0}^{\infty} X_n$, where $X_0 = \{a\}$ and, for $n \geq 0$, $X_{n+1} = X_n \cdot_a (L \cup \{a\})$. □

Exercise 3.40. Let $a \in \Sigma_0$, $a \neq e$, and let $L \subseteq T_\Sigma$. Prove that $\text{yield}(L^{*a}) = (\text{yield}(L))^{*a}$. □

Theorem 3.41. *RECOG* is closed under tree concatenation closure.

Proof. Again the proof is a straightforward generalization of the string case. Let $G = (N, \Sigma, R, S)$ be a regular tree grammar in normal form, and let $a \in \Sigma_0$. Construct the regular tree grammar $\overline{G} = (N \cup \{S_0\}, \Sigma, \overline{R}, S_0)$, where $\overline{R} = R \cup \{A \to S \mid A \to a \text{ is in } R\} \cup \{S_0 \to S, S_0 \to a\}$. Then $L(\overline{G}) = (L(G))^{*a}$. □

Corollary 3.42. *CFL* is closed under substitution closure.

Proof. Use Theorem 3.28 and Exercise 3.40. □

It is well known that the class of regular string languages is the smallest class containing the finite languages and closed under union, concatenation and closure. A similar result holds for recognizable tree languages.

Theorem 3.43. *RECOG* is the smallest class of tree languages containing the finite tree languages and closed under union, tree concatenation and tree concatenation closure.

Proof. We have shown that *RECOG* satisfies the above conditions in Exercise 3.26 and Theorems 3.32, 3.35 and 3.41. It remains to show that every recognizable tree language can be built up from the finite tree languages using the operations $\cup$, $\cdot_a$ and ^{*a}. Let $G = (N, \Sigma, R, S)$ be a regular tree grammar (it is easy to think of it as being in normal form). We shall use the elements of N to do tree concatenation at. For $A \in N$ and $P, Q \subseteq N$ with $P \cap Q = \emptyset$, let us denote by $L_{A,P}^{Q}$ the set of all trees $t \in T_\Sigma(P)$ for which there is a derivation $A \Rightarrow t_1 \Rightarrow t_2 \Rightarrow \cdots \Rightarrow t_n \Rightarrow t_{n+1} = t$ $(n \geq 0)$ such that, for $1 \leq i \leq n$, $t_i \in T_\Sigma(Q \cup P)$ and a rule with left hand side in Q is applied to t_i to obtain

---

[†]Recall the notation $L_1 \cdot_a L_2$ from Definition 2.27.

t_{i+1}. We shall show, by induction on the cardinality of Q, that all sets $L^Q_{A,P}$ can be built up from the finite tree languages by the operations $\cup$, $\cdot_B$ and ^{*B} (for all $B \in N$). For $Q = \emptyset$, $L^\emptyset_{A,P}$ is the set of all those right hand sides of rules with left hand side A, that are in $T_\Sigma(P)$. Thus $L^\emptyset_{A,P}$ is a finite tree language for all A and P. Assuming now that, for $Q \subseteq N$, all sets $L^Q_{A,P}$ can be built up from the finite tree languages, the same holds for all sets $L^{Q \cup \{B\}}_{A,P}$, where $B \in N - Q$, since

$$L^{Q \cup \{B\}}_{A,P} = L^Q_{A,P \cup \{B\}} \cdot_B (L^Q_{B,P \cup \{B\}})^{*B} \cdot_B L^Q_{B,P}$$

(a formal proof of this equation is left to the reader). Thus, since $L(G) = L^N_{S,\emptyset}$, the theorem is proved. $\qquad\square$

In other words, each recognizable tree language can be denoted by a "regular expression" with trees as constants and $\cup$, $\cdot_A$ and ^{*A} as operators.

Exercise 3.44. Try to find a regular expression for the language generated by the regular tree grammar $G = (N, \Sigma, R, S)$ with $N = \{S, T\}$, $\Sigma_0 = \{a\}$, $\Sigma_2 = \{p\}$ and $R = \{S \to p[TS], S \to a, T \to p[TT], T \to a\}$. Use the algorithm in the proof of Theorem 3.43. $\qquad\square$

As a corollary we obtain the result that all context-free languages can be denoted by "context-free expressions".

Corollary 3.45. *CFL* is the smallest class of languages containing the finite languages and closed under union, substitution and substitution closure.

Proof. Exercise. $\qquad\square$

Exercise 3.46. Define the operation of "iterated concatenation at a" (for tree languages) and "iterated substitution at a" (for string languages) by $\mathrm{it}_a(L) = L^{*a} \cdot_a \emptyset$. Prove (using Theorem 3.43) that *RECOG* is the smallest class of tree languages containing the finite tree languages and closed under the operations of union, top concatenation and iterated concatenation. Show that this implies that *CFL* is the smallest class of languages containing the finite languages and closed under the operations of union, concatenation and iterated substitution (cf. [Sal, VI.11]). $\qquad\square$

Let us now turn to another operation on trees: that of relabeling the nodes of a tree.

Definition 3.47. Let Σ and Δ be ranked alphabets. A <u>relabeling</u> r is a family $\{r_k\}_{k \geq 0}$ of mappings $r_k : \Sigma_k \to \mathcal{P}(\Delta_k)$. A relabeling determines a mapping $r : T_\Sigma \to \mathcal{P}(T_\Delta)$ by the requirements

  (i) for $a \in \Sigma_0$, $r(a) = r_0(a)$,

  (ii) for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
    $r(a[t_1 \cdots t_k]) = \{b[s_1 \cdots s_k] \mid b \in r_k(a) \text{ and } s_i \in r(t_i)\}$.

If, for each $k \geq 0$ and each $a \in \Sigma_k$, $r_k(a)$ consists of one element only, then r is called a projection. $\qquad\square$

Obviously, $RECOG$ is closed under relabelings.

Theorem 3.48. $RECOG$ is closed under relabelings.

Proof. Let r be a relabeling, and consider some regular tree grammar G. By replacing each rule $A \to t$ of G by all rules $A \to s$, $s \in r(t)$, one obtains a regular tree grammar for $r(L(G))$. (In order that "$r(t)$" makes sense, we define $r(B) = \{B\}$ for each nonterminal B of G). $\qquad\square$

We are now in a position to study the connection between recognizable tree languages and sets of derivation trees of context-free grammars. We shall consider two kinds of derivation trees. First we define the "ordinary" kind of derivation tree (cf. Example 1.1).

Definition 3.49. Let $G = (N, \Sigma, R, S)$ be a context-free grammar. Let Δ be the ranked alphabet such that $\Delta_0 = \Sigma \cup \{e\}$ and, for $k \geq 1$, Δ_k is the set of nonterminals $A \in N$ for which there is a rule $A \to w$ with $|w| = k$ (in case $k = 1 : |w| = 1$ or $|w| = 0$). For each $\alpha \in N \cup \Sigma$, the set of derivation trees with top α, denoted by D_G^α, is the tree language over Δ defined recursively as follows

(i) for each a in Σ, $a \in D_G^a$;

(ii) for each rule $A \to \alpha_1 \cdots \alpha_n$ in R ($n \geq 1$, $A \in N$, $\alpha_i \in \Sigma \cup N$),
   if $t_i \in D_G^{\alpha_i}$ for $1 \leq i \leq n$, then $A[t_1 \cdots t_n] \in D_G^A$;

(iii) for each rule $A \to \lambda$ in R, $A[e] \in D_G^A$. $\qquad\square$

Definition 3.50. A tree language L is said to be local if, for some context-free grammar $G = (N, \Sigma, R, S)$ and some set of symbols $V \subseteq N \cup \Sigma$, $L = \bigcup_{\alpha \in V} D_G^\alpha$. $\qquad\square$

Exercise 3.51. Show that each local tree language is recognizable. $\qquad\square$

Note that a local tree language is the set of all derivation trees of a context-free grammar which has a set of initial symbols (instead of one initial nonterminal).

The reason for the name "local" is that such a tree language L is determined by (1) a finite set of trees of height one, (2) a finite set of "initial symbols", (3) a finite set of "final symbols", and the requirement that L consists of all trees t such that each node of t together with its direct descendants belongs to (1), the top label of t belongs to (2), and the leaf labels of t to (3).

We now show that the class of local tree languages is properly included in $RECOG$.

Theorem 3.52. There are recognizable tree languages which are not local.

Proof. Let $\Sigma_0 = \{a, b\}$ and $\Sigma_2 = \{S\}$. Consider the tree language $L = \{S[S[ba]S[ab]]\}$. Obviously L is recognizable. Suppose that L is local. Then there is a context-free

grammar G such that $D_G^S = L$. Thus $S \to SS$, $S \to ba$ and $S \to ab$ are rules of G. But then $S[S[ab]S[ba]] \in L$. Contradiction. [†] $\qquad \square$

Note that the recognizable tree language L in the above proof can be recognized by a deterministic top-down fta. Note also that the tree language given in the proof of Theorem 3.14 is local. Hence the local tree languages and the tree languages recognized by det. top-down fta are incomparable.

Exercise 3.53. Find a recognizable tree language which is neither local nor recognizable by a det. top-down fta. $\qquad \square$

It is clear that, if $\Sigma_0 = \{a, b\}$ and $\Sigma_2 = \{S_1, S_2, S_3\}$, then $L' = \{S_1[S_2[ba]S_3[ab]]\}$ is a local language. Hence the language L in Theorem 3.52 is the projection of the local language L' (project S_1, S_2 and S_3 on S). We will show that this is true in general: each recognizable tree language is the projection of a local tree language. In fact we shall show a slightly stronger fact. To do this we define the second type of derivation tree of a context-free grammar, called "rule tree".

Definition 3.54. Let $G = (N, \Sigma, R, S)$ be a context-free grammar. Let $\overline{R}$ be any set of symbols in one-to-one correspondence with R, $\overline{R} = \{\overline{r} \mid r \in R\}$. Each element of $\overline{R}$ is given a rank such that, if r in R is of the form $A \to w_0 A_1 w_1 A_2 w_2 \cdots A_k w_k$ (for some $k \geq 0$, $A_1, \ldots, A_k \in N$ and $w_0, w_1, \ldots, w_k \in \Sigma^*$), then $\overline{r} \in \overline{R}_k$. The set of rule trees of G, denoted by $RT(G)$, is defined to be the tree language generated by the regular tree grammar $\overline{G} = (N, \overline{R}, P, S)$, where P is defined by
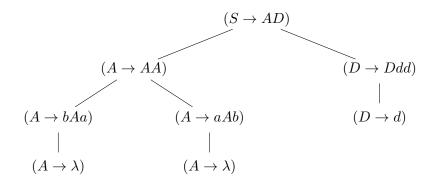
(i) if $r = (A \to w_0 A_1 \cdots w_{k-1} A_k w_k)$, $k \geq 1$, is in R,
   then $A \to \overline{r}[A_1 \cdots A_k]$ is in P;

(ii) if $r = (A \to w_0)$ is in R,
   then $A \to \overline{r}$ is in P. $\qquad \square$

Definition 3.55. We shall say that a tree language L is a rule tree language if $L = RT(G)$ for some context-free grammar G. $\qquad \square$

Thus, a rule tree is a derivation tree in which the nodes are labeled by the rules applied during the derivation. It should be obvious, that for each context-free grammar $G = (N, \Sigma, R, S)$ there is a one-to-one correspondence between the tree languages $RT(G)$ and D_G^S.

Example 3.56. Consider Example 1.1. For each rule r in that example, let (r) stand for a new symbol. The rule tree "corresponding" to the derivation tree displayed in Example 1.1 is

---

[†] Other examples are for instance $\{S[T[a]T[b]]\}$ and $\{S[S[a]]\}$.

$$
\begin{array}{c}
(S \to AD) \\
\end{array}
$$

The tree structure:

- $(S \to AD)$
  - $(A \to AA)$
    - $(A \to bAa)$
      - $(A \to \lambda)$
    - $(A \to aAb)$
      - $(A \to \lambda)$
  - $(D \to Ddd)$
    - $(D \to d)$

Note that this tree is obtained from the other one by viewing the building blocks (trees of height one) of the local tree as the nodes of the rule tree. $\square$

The following theorem shows the relationship of the rule languages to those defined before.

Theorem 3.57. The class of rule tree languages is properly included in the intersection of the class of local tree languages and the class of tree languages recognizable by a det. top-down fta.

Proof. We first show inclusion in the class of local tree languages. Let $G = (N, \Sigma, R, S)$ be a context-free grammar, and $\overline{R} = \{\overline{r} \mid r \in R\}$. Consider the context-free grammar $G_1 = (\overline{R} - \overline{R}_0, \overline{R}_0, P, -)$ where P is defined as follows: if $r = (A \to w_0 A_1 w_1 \cdots A_k w_k)$, $k \geq 1$, is in R, then $\overline{r} \to \overline{r}_1 \cdots \overline{r}_k$ is in P for all rules $r_1, \ldots, r_k \in R$ such that the left hand side of r_i is A_i $(1 \leq i \leq k)$. Let $\overline{V} = \{\overline{r} \mid r \in R \text{ has left hand side } S\}$. Then $RT(G) = \bigcup_{\alpha \in \overline{V}} D_{G_1}^{\alpha}$, and hence $RT(G)$ is local.

To show that $RT(G)$ can be recognized by a deterministic top-down fta, consider $M = (Q, \overline{R}, \delta, q_0, F)$, where $Q = N \cup \{W\}$, $q_0 = S$, for $\overline{r} \in \overline{R}_0$, $F_{\overline{r}}$ consists of the left hand side of r only, and for $\overline{r} \in \overline{R}_k$, r of the form $A \to w_0 A_1 w_1 A_2 w_2 \cdots A_k w_k$, $\delta_{\overline{r}}(A) = (A_1, \ldots, A_k)$ and $\delta_{\overline{r}}(B) = (W, \ldots, W)$ for all other $B \in Q$. Then $L(M) = RT(G)$.

To show proper inclusion, let H be the context-free grammar with rules $S \to SS$, $S \to aS$, $S \to Sb$ and $S \to ab$. Then D_H^S is a local tree language. It is easy to see that D_H^S can be recognized by a det. top-down fta. Now suppose that $D_H^S = RT(G)$ for some context-free grammar G. Since S has rank 2 and since the configuration $\begin{smallmatrix} & S & \\ \swarrow & & \searrow \\ S & & S \end{smallmatrix}$ occurs in D_H^S, S is the name of a rule of G of the form $A \to w_0 A w_1 A w_2$. Now, since a and b are of rank 0 and since $\begin{smallmatrix} & S & \\ \swarrow & & \searrow \\ a & & b \end{smallmatrix}$ is in D_H^S, a and b are names of rules $A \to w_3$ and $A \to w_4$. Hence $S[ba]$ is a rule tree of G. Contradiction. $\square$

24

We now characterize the recognizable tree languages in terms of rule tree languages.

Theorem 3.58. Every recognizable tree language is the projection of a rule tree language.

Proof. Let $G = (N, \Sigma, R, S)$ be a regular tree grammar in normal form. We shall define a regular tree grammar $\overline{G}$ and a projection p such that $L(G) = p(L(\overline{G}))$ and $L(\overline{G})$ is a rule tree language. $\overline{G}$ will simulate G, but $\overline{G}$ will put all information about the rules, applied during the derivation of a tree t, into the tree itself. This is a useful technique.

Let $\overline{R}$ be a set of symbols in one-to-one correspondence with R, and let $\overline{G} = (N, \overline{R}, P, S)$. The ranking of $\overline{R}$, the set P of rules and the projection p are defined simultaneously as follows:

  (i) if $r \in R$ is the rule $A \to a[B_1 \cdots B_k]$, then $\overline{r}$ has rank k, $A \to \overline{r}[B_1 \cdots B_k]$ is in P and $p_k(\overline{r}) = a$;

  (ii) if $r \in R$ is the rule $A \to a$, then $\overline{r}$ has rank 0, $A \to \overline{r}$ is in P and $p_0(\overline{r}) = a$.

It is obvious that $p(L(\overline{G})) = L(G)$. Now note that G may be viewed as a context-free grammar (over $\Sigma \cup \{[,]\}$). In fact, $\overline{G}$ is the same as the one constructed in Definition 3.54! Thus $L(\overline{G})$ is a rule tree language. $\qquad\square$

Since $RECOG$ is closed under projections (Theorem 3.48), we now easily obtain the following corollary.

Corollary 3.59. For each tree language L the following four statements are equivalent:
  (i) L is recognizable
  (ii) L is the projection of a rule tree language
  (iii) L is the projection of a local tree language
  (iv) L is the projection of a tree language recognizable by a det. top-down fta. $\qquad\square$

Exercise 3.60. Show that, in the case of local tree languages, the projection involved in the above corollary (iii) can be taken as the identity on symbols of rank 0 (thus the yields are preserved). $\qquad\square$

As a final operation on trees we consider the notion of tree homomorphism. For strings, a homomorphism h associates a string $h(a)$ with each symbol a of the alphabet, and transforms a string $a_1 a_2 \cdots a_n$ into the string $h(a_1) \cdot h(a_2) \cdots h(a_n)$. Generalizing this to trees, a tree homomorphism h associates a tree $h(a)$ with each symbol a of the ranked alphabet (actually, one tree for each rank). The application of h to a tree t consists in replacing each symbol a of t by the tree $h(a)$, and tree concatenating all the resulting trees. Note that, if a is of rank k, then $h(a)$ should be tree concatenated with k other trees; therefore, since tree concatenation happens <u>at</u> symbols of rank 0, the tree $h(a)$ should contain at least k different symbols of rank 0. Since, in general, the number of symbols of rank 0 in some alphabet may be less than the rank of some other symbol, we allow for the use of an arbitrary number of auxiliary symbols of rank 0, called "variables" (recall the use of nonterminals as auxiliary symbols of rank 0 in Theorem 3.43).

Definition 3.61. Let $x_1, x_2, x_3, \ldots$ be an infinite sequence of different symbols, called variables. Let $X = \{x_1, x_2, x_3, \ldots\}$, for $k \geq 1$, $X_k = \{x_1, x_2, \ldots, x_k\}$, and $X_0 = \emptyset$. Elements of X will also be denoted by x, y and z. $\qquad\square$

Definition 3.62. Let Σ and Δ be ranked alphabets. A tree homomorphism h is a family $\{h_k\}_{k \geq 0}$ of mappings $h_k : \Sigma_k \to T_\Delta(X_k)$. A tree homomorphism determines a mapping $h : T_\Sigma \to T_\Delta$ as follows:

  (i) for $a \in \Sigma_0$, $h(a) = h_0(a)$;

 (ii) for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,

$$h(a[t_1 \cdots t_k]) = h_k(a)\langle x_1 \leftarrow h(t_1), \ldots, x_k \leftarrow h(t_k)\rangle.$$

In the particular case that, for each $a \in \Sigma_k$, $h_k(a)$ does not contain two occurrences of the same x_i ($i = 1, 2, 3, \ldots$), h is called a linear tree homomorphism. $\qquad\square$

A general tree homomorphism h has the abilities of deleting ($h_k(a)$ does not contain x_i), copying ($h_k(a)$ contains ≥ 2 occurrences of x_i) and permuting (if $i < j$, then x_j may occur before x_i in $h_k(a)$) subtrees. Moreover, at each node, it can add pieces of tree (the frontier of $h_k(a)$ need not to be an element of X^*). A linear homomorphism cannot copy. Note that, to obtain the proper generalization of the monadic case, one should also forbid deletion, and require that $h_0(a) = a$ for all $a \in \Sigma_0$ (moreover, no pieces of tree should be added).

Exercise 3.63. Let $\Sigma_0 = \{a, b\}$ and $\Sigma_2 = \{p\}$. Consider the tree homomorphism h such that $h_0(a) = a$, $h_0(b) = b$ and $h_2(p) = p[x_2 x_1]$. Show that, for every t in T_Σ, $\mathrm{yield}(h(t))$ is the mirror image of $\mathrm{yield}(t)$. $\qquad\square$

It is easy to see that recognizable tree languages are not closed under arbitrary tree homomorphisms; they are closed under linear tree homomorphisms. This is shown in the following two theorems.

Theorem 3.64. *RECOG* is not closed under arbitrary tree homomorphisms.

Proof. Let $\Sigma_0 = \{a\}$ and $\Sigma_1 = \{b\}$. Let h be the tree homomorphism defined by $h_0(a) = a$ and $h_1(b) = b[x_1 x_1]$. Consider the recognizable tree language T_Σ. It is easy to prove that $\mathrm{yield}(h(T_\Sigma)) = \{a^{2^n} \mid n \geq 0\}$. Since $\{a^{2^n} \mid n \geq 0\}$ is not a context-free language, Theorem 3.28 implies that $h(T_\Sigma)$ is not recognizable. $\qquad\square$

Theorem 3.65. *RECOG* is closed under linear tree homomorphisms.

Proof. The idea of the proof is obvious. Given some regular tree grammar generating a recognizable tree language, we change the right hand sides of all rules into their homomorphic images. The resulting grammar generates homomorphic images of "sentential forms" of the original grammar (note that this wouldn't work in the nonlinear case). The only thing we should worry about is that the homomorphism may be deleting. In that case superfluous rules in the original grammar might be transformed into useful rules

in the new grammar. This is solved by requiring that the original grammar does not contain any superfluous rule. The formal construction is as follows.

Let $G = (N, \Sigma, R, S)$ be a regular tree grammar in normal form, such that for each nonterminal A there is at least one $t \in T_\Sigma$ such that $A \overset{*}{\Rightarrow} t$ (since G is a context-free grammar, it is well known that each regular tree grammar has an equivalent one satisfying this condition). Let h be a homomorphism from T_Σ into T_Δ, for some ranked alphabet Δ. Extend h to trees in $T_\Sigma(N)$ by defining $h_0(A) = A$ for all A in N. Thus h is now a homomorphism from $T_\Sigma(N)$ into $T_\Delta(N)$. Construct the regular tree grammar $H = (N, \Delta, \overline{R}, S)$, where $\overline{R} = \{A \to h(t) \mid A \to t \text{ is in } R\}$.

To show that $L(H) = h(L(G))$ we shall prove that

(1) if $A \overset{*}{\underset{G}{\Rightarrow}} t$, then $A \overset{*}{\underset{H}{\Rightarrow}} h(t)$ $\hspace{2cm}$ $(t \in T_\Sigma)$; and

(2) if $A \overset{*}{\underset{H}{\Rightarrow}} s$, then there exists t such that

$$h(t) = s \text{ and } A \overset{*}{\underset{G}{\Rightarrow}} t \hspace{2cm} (s \in T_\Delta, t \in T_\Sigma).$$

Let us give the straightforward proofs as detailed as possible.

(1) The proof is by induction on the number of steps in the derivation $A \overset{*}{\underset{G}{\Rightarrow}} t$. If, in one step, $A \underset{G}{\Rightarrow} t$, then $t \in \Sigma_0$ and $A \to t$ is in R. Hence $A \to h(t)$ is in $\overline{R}$, and so $A \overset{*}{\underset{H}{\Rightarrow}} h(t)$. Now suppose that the first step in $A \overset{*}{\underset{G}{\Rightarrow}} t$ results from the application of a rule of the form $A \to a[B_1 \cdots B_k]$. Then $A \overset{*}{\underset{G}{\Rightarrow}} t$ is of the form $A \underset{G}{\Rightarrow} a[B_1 \cdots B_k] \overset{*}{\underset{G}{\Rightarrow}} t$. It follows that t is of the form $a[t_1 \cdots t_k]$ such that $B_i \overset{*}{\underset{G}{\Rightarrow}} t_i$ for all $1 \le i \le k$. Hence, by induction, $B_i \overset{*}{\underset{H}{\Rightarrow}} h(t_i)$. Now, since the rule $A \to h_k(a)\langle x_1 \leftarrow B_1, \ldots, x_k \leftarrow B_k \rangle$ is in $\overline{R}$ by definition, we have (prove this!)

$$A \underset{H}{\Rightarrow} h_k(a)\langle x_1 \leftarrow B_1, \ldots, x_k \leftarrow B_k \rangle \overset{*}{\underset{H}{\Rightarrow}} h_k(a)\langle x_1 \leftarrow h(t_1), \ldots, x_k \leftarrow h(t_k) \rangle$$

$$= h(a[t_1 \cdots t_k]) = h(t).$$

(2) The proof is by induction on the number of steps in $A \overset{*}{\underset{H}{\Rightarrow}} s$. For zero steps the statement is trivially true. Suppose that the first step in $A \overset{*}{\underset{H}{\Rightarrow}} s$ results from the application of a rule $A \to h_0(a)$ for some a in Σ_0. Then $h(a) = s$ and $A \overset{*}{\underset{G}{\Rightarrow}} a$. Now suppose that the first step results from the application of a rule $A \to h_k(a)\langle x_1 \leftarrow B_1, \ldots, x_k \leftarrow B_k \rangle$, where $A \to a[B_1 \cdots B_k]$ is a rule of G. Then the derivation is $A \underset{H}{\Rightarrow} h_k(a)\langle x_1 \leftarrow B_1, \ldots, x_k \leftarrow B_k \rangle \overset{*}{\underset{H}{\Rightarrow}} s$. At this point we need both linearity of h (to be sure that each B_i in $h_k(a)\langle x_1 \leftarrow B_1, \ldots, x_k \leftarrow B_k \rangle$ produces at most one subtree of s) and the condition on G (to deal with deletion: since $h_k(a)\langle x_1 \leftarrow B_1, \ldots, x_k \leftarrow B_k \rangle$ need not contain an occurrence of B_i, we need an arbitrary tree generated, in G, by B_i to be able to construct the tree t such that $h(t) = s$). There exist trees $s_1, \ldots, s_k$ in T_Δ such that $s = h_k(a)\langle x_1 \leftarrow s_1, \ldots, x_k \leftarrow s_k \rangle$ and

(i) if x_i occurs in $h_k(a)$, then $B_i \stackrel{*}{\underset{H}{\Rightarrow}} s_i$;

(ii) if x_i does not occur in $h_k(a)$, then $s_i = h(t_i)$ for some arbitrary t_i such that $B_i \stackrel{*}{\underset{G}{\Rightarrow}} t_i$.

Hence, by induction and (ii), there are trees $t_1, \ldots, t_k$ such that $h(t_i) = s_i$ and $B_i \stackrel{*}{\underset{G}{\Rightarrow}} t_i$ for all $1 \le i \le k$. Consequently, if $t = a[t_1 \cdots t_k]$, then $A \underset{G}{\Rightarrow} a[B_1 \cdots B_k] \stackrel{*}{\underset{G}{\Rightarrow}} a[t_1 \cdots t_k] = t$, and $h(t) = h_k(a)\langle x_1 \leftarrow h(t_1), \ldots, x_k \leftarrow h(t_k)\rangle = s$.

This proves the theorem. $\square$

Exercise 3.66. In the string case one can also prove that the regular languages are closed under homomorphisms by using the Kleene characterization theorem. Give an alternative proof of Theorem 3.65 by using Theorem 3.43 (use the fact, which is implicit in the proof of that theorem, that each regular tree language over the ranked alphabet Σ can be built up from finite tree languages using operations $\cup$, $\cdot_A$ and $*^A$, where $A \notin \Sigma$). $\square$

As an indication how one could use theorems like Theorem 3.65, we prove the following theorem, which is (slightly!) stronger than Theorem 3.28.

Theorem 3.67. Each context-free language over Δ is the yield of a recognizable tree language over Σ, where $\Sigma_0 = \Delta \cup \{e\}$ and $\Sigma_2 = \{*\}$.

Proof. Let L be a context-free language over Δ. By Theorem 3.28, there is a recognizable tree language U over some ranked alphabet Ω with $\Omega_0 = \Delta \cup \{e\}$, such that yield$(U) = L$. Let h be the linear tree homomorphism from T_Ω into T_Σ such that

$h_0(a) = a$ for all a in $\Delta \cup \{e\}$,

$h_1(a) = x_1$ for all a in Ω_1, and

$h_k(a) = *[x_1 * [x_2 * [\cdots * [x_{k-1} x_k] \cdots]]]$ for all a in Ω_k, $k \ge 2$.

By Theorem 3.65, $h(U)$ is a recognizable tree language over Σ. It is easy to show that, for each t in T_Ω, yield$(h(t)) = $ yield(t). Hence yield$(h(U)) = $ yield$(U) = L$. $\square$

Note that Theorem 3.67 is "equivalent" to the fact that each context-free language can be generated by a context-free grammar in Chomsky normal form.

Exercise 3.68. Try to show that $RECOG$ is closed under inverse (not necessarily linear) homomorphisms; that is, if $L \in RECOG$ and h is a tree homomorphism, then $h^{-1}(L) = \{t \mid h(t) \in L\}$ is recognizable. (Represent L by a det. bottom-up fta). $\square$

We have now discussed all *AFL* operations (see [Sal, IV]) generalized to trees: union, tree concatenation, tree concatenation closure, (linear) tree homomorphism, inverse tree homomorphism and intersection with a recognizable tree language. Thus, according to previous results, $RECOG$ is a "tree *AFL*".

Exercise 3.69. Generalize the operation of string substitution (see Definition 2.29) to trees, and show that $RECOG$ is closed under "linear tree substitution". $\square$

Exercise 3.70. Suppose you don't know about context-free grammars. Consider the notion of regular tree grammar. Give a recursive definition of the relation $\Rightarrow$ for such a grammar. Show that, if $a[t_1 \cdots t_k] \overset{*}{\Rightarrow} s$, then there are $s_1, \ldots, s_k$ such that $s = a[s_1 \cdots s_k]$ and $t_i \overset{*}{\Rightarrow} s_i$ for all $1 \leq i \leq k$. Which of the two definitions of regular tree grammar do you prefer? □

3.3 Decidability

Obviously the membership problem for recognizable tree languages is solvable: given a tree t and an fta M, just feed t into M and see whether t is recognized or not.

We now want to prove that the emptiness and finiteness problems for recognizable tree languages are solvable. To do this we generalize the pumping lemma for regular string languages to recognizable tree languages: for each regular string language L there is an integer p such that for all strings z in L, if $|z| \geq p$, then there are strings u, v and w such that $z = uvw$, $|vw| \leq p$, $|v| \geq 1$ and, for all $n \geq 0$, $uv^n w \in L$.

Theorem 3.71. Let Σ be a ranked alphabet, and x a symbol not in Σ. For each recognizable tree language L over Σ we can find an integer p such that for all trees t in L, if height$(t) \geq p$, then there are trees $u, v, w \in T_\Sigma(\{x\})$ such that

  (i) u and v contain exactly one occurrence of x, and $w \in T_\Sigma$;

 (ii) $t = u \cdot_x v \cdot_x w$;

(iii) height$(v \cdot_x w) \leq p$;

(iv) height$(v) \geq 1$, and

 (v) for all $n \geq 0$, $u \cdot_x v^{nx} \cdot_x w \in L$, where $v^{nx} = v \cdot_x v \cdot_x \cdots \cdot_x v$ (n times).

Proof. Let $M = (Q, \Sigma, \delta, s, F)$ be a deterministic bottom-up fta recognizing L. Let p be the number of states of M. Consider a tree $t \in L(M)$ such that height$(t) \geq p$. Considering some path of maximal length through t, it is clear that there are trees $t_1, t_2, \ldots, t_n \in T_\Sigma(\{x\})$ such that $n \geq p + 1$, $t = t_n \cdot_x t_{n-1} \cdot_x \cdots \cdot_x t_2 \cdot_x t_1$, the trees $t_2, \ldots, t_n$ contain exactly one occurrence of x and have height ≥ 1, and $t_1 \in T_\Sigma$ (this is a "linearization" of t according to some path). Now consider the states $q_i = \widehat{\delta}(t_i \cdot_x \cdots \cdot_x t_1)$ for $1 \leq i \leq n$. Then, among $q_1, \ldots, q_{p+1}$ there are two equal states: there are i, j such that $q_i = q_j$ and $1 \leq i < j \leq p + 1$. Let $u = t_n \cdot_x \cdots \cdot_x t_{j+1}$, $v = t_j \cdot_x \cdots \cdot_x t_{i+1}$ and $w = t_i \cdot_x \cdots \cdot_x t_1$. Then requirements (i)-(iv) in the statement of the theorem are obviously satisfied. Furthermore, in general, if $\widehat{\delta}(s_1) = \widehat{\delta}(s_2)$, then $\widehat{\delta}(s \cdot_x s_1) = \widehat{\delta}(s \cdot_x s_2)$. Hence, since $\widehat{\delta}(v \cdot_x w) = \widehat{\delta}(w)$, requirement (v) is also satisfied. □

As a corollary to Theorem 3.71 we obtain the pumping lemma for context-free languages.

Corollary 3.72. For each context-free language L over Δ we can find an integer q such that for all strings $z \in L$, if $|z| \geq q$, then there are strings u_1, v_1, w_0, v_2 and u_2 in Δ^*, such that $z = u_1 v_1 w_0 v_2 u_2$, $|v_1 w_0 v_2| \leq q$, $|v_1 v_2| > 0$, and, for all $n \geq 0$, $u_1 v_1^n w_0 v_2^n u_2 \in L$.

Proof. By Theorem 3.67, and the fact that each context-free language can be generated by a λ-free context-free grammar, there is a recognizable tree language U over Σ such that $\text{yield}(U) = L$, where $\Sigma_0 = \Delta$ and $\Sigma_2 = \{*\}$. Let p be the integer corresponding to U according to Theorem 3.71, and put $q = 2^p$. Obviously, if $z \in L$ and $|z| \geq q$, then there is a t in U such that $\text{yield}(t) = z$ and $\text{height}(t) \geq p$. Then, by Theorem 3.71 there are trees u, v and w such that (i)-(v) in that theorem hold. Thus $t = u \cdot_x v \cdot_x w$. Let $\text{yield}(u) = u_1 x u_2$, $\text{yield}(v) = v_1 x v_2$ and $\text{yield}(w) = w_0$ (see (i)). Then $z = \text{yield}(t) = \text{yield}(u \cdot_x v \cdot_x w) = \text{yield}(u) \cdot_x \text{yield}(v) \cdot_x \text{yield}(w) = u_1 v_1 w_0 v_2 u_2$. It is easy to see that all other requirements stated in the corollary are also satisfied. $\qquad\square$

Exercise 3.73. Let Σ be a ranked alphabet such that $\Sigma_0 = \{a, b\}$. Show that the tree language $\{t \in T_\Sigma \mid \text{yield}(t) \text{ has an equal number of } a\text{'s and } b\text{'s}\}$ is not recognizable. $\quad\square$

From the pumping lemma the decidability of both emptiness and finiteness problem for $RECOG$ follows.

Theorem 3.74. The emptiness problem for recognizable tree languages is decidable.

Proof. Let L be a recognizable tree language, and let p be the integer of Theorem 3.71. Obviously (using $n = 0$ in point (v)), L is nonempty if and only if L contains a tree of height $< p$. $\qquad\square$

Theorem 3.75. The finiteness problem for recognizable tree languages is decidable.

Proof. Let L be a recognizable tree language, and let p be the integer of Theorem 3.71. Obviously, L is finite if and only if all trees in L are of height $< p$. Thus, L is finite iff $L \cap \{t \mid \text{height}(t) \geq p\} = \emptyset$. Since $L \cap \{t \mid \text{height}(t) \geq p\}$ is recognizable (Exercise 3.26 and Theorem 3.32), this is decidable by the previous theorem. $\qquad\square$

Note that the decidability of emptiness and finiteness problem for context-free languages follows from these two theorems together with the "yield-theorem" (with $e \notin \Sigma_0$, $\Sigma_1 = \emptyset$).

As in the string case we now obtain the decidability of inclusion of recognizable tree languages (and hence of equality).

Theorem 3.76. It is decidable, for arbitrary recognizable tree languages U and V, whether $U \subseteq V$ (and also, whether $U = V$).

Proof. Since U is included in V iff the intersection of U with the complement of V is empty, the theorem follows from Theorems 3.32 and 3.74. $\qquad\square$

Note again that each regular tree language is a special kind of context-free language. Note also that inclusion of context-free languages is not decidable (neither equality). Therefore it is nice that we have found a subclass of CFL for which inclusion and equality are decidable. Note also that CFL is not closed under intersection, but $RECOG$ is. We shall now relate these facts to some results in the literature concerning "parenthesis languages" and "structural equivalence" of context-free grammars (see [Sal, VIII.3]).

Definition 3.77. A <u>parenthesis grammar</u> is a context-free grammar $G = (N, \Sigma \cup \{[,]\}, R, S)$ such that each rule in R is of the form $A \to [w]$ with $A \in N$ and $w \in (\Sigma \cup N)^*$. The language generated by G is called a <u>parenthesis language</u>. $\square$

To relate parenthesis languages to recognizable tree languages, let us restrict attention to ranked alphabets Δ such that, for $k \geq 1$, if $\Delta_k \neq \emptyset$, then $\Delta_k = \{*\}$, where $*$ is a fixed symbol. Suppose that in our recursive definition of "tree" we change $a[t_1 \cdots t_k]$ into $[t_1 \cdots t_k]$ (see Definition 2.5 and Remark 2.35). Then, obviously, all our results about the class $RECOG$ are still valid. Furthermore, since $*$ is the only symbol of rank ≥ 1, we may as well replace $\underset{*}{[}$ by $[$. In this way, each parenthesis language is in $RECOG$ (in fact, each parenthesis grammar is a regular tree grammar). It is also easy to see that, if L is a recognizable tree language (over a restricted ranked alphabet Δ), then $L - \Delta_0$ is a parenthesis language. From these considerations we obtain the following theorem.

Theorem 3.78. The class of parenthesis languages is closed under union, intersection and subtraction. The inclusion problem for parenthesis languages is decidable.

Proof. The first statement follows directly from Theorem 3.32 and the last remark above. The second statement follows directly from Theorem 3.76. $\square$

A paraphrase of this theorem is obtained as follows.

Definition 3.79. For any ranked alphabet, let p be the projection such that $p(a) = a$ for all symbols of rank 0, and $p(a) = *$ for all symbols of rank ≥ 1. Let G be a context-free grammar. The <u>bare tree language</u> of G, denoted by $BT(G)$, is $p(D_G^S)$, where S is the initial nonterminal of G. We say that two context-free grammars G_1 and G_2 are <u>structurally equivalent</u> iff they generate the same bare tree language (i.e., $BT(G_1) = BT(G_2)$). $\square$

Thus, G_1 and G_2 are structurally equivalent if their sets of derivation trees are the same after "erasing" all nonterminals.

Theorem 3.80. It is decidable for arbitrary context-free grammars whether they are structurally equivalent.

Proof. For any context-free grammar $G = (N, \Sigma, R, S)$, let $[G]$ be the parenthesis grammar $(N, \Sigma \cup \{[,]\}, \overline{R}, S)$, where $\overline{R} = \{A \to [w] \mid A \to w \text{ is in } R\}$. Obviously, $L([G]) = BT(G)$. Hence, by Theorem 3.78, the theorem holds. $\square$

Exercise 3.81. Show that, for any two context-free grammars G_1 and G_2 there exists a context-free grammar G_3 such that $BT(G_3) = BT(G_1) \cap BT(G_2)$. $\square$

Exercise 3.82. Show that each context-free grammar has a structurally equivalent context-free grammar that is invertible (cf. Exercise 3.29). $\square$

Exercise 3.83. Consider the "bracketed context-free languages" of Ginsburg and Harrison, and show that some of their results follow easily from results about $RECOG$ (show first that each recognizable tree language is a deterministic context-free language). $\square$

Exercise 3.84. Investigate whether it is decidable for an arbitrary recognizable tree language R

(i) whether R is local;

(ii) whether R is a rule tree language;

(iii) whether R is recognizable by a det. top-down fta. ☐

4 Finite state tree transformations

4.1 Introduction: Tree transducers and semantics

In this part we will be concerned with the notion of a tree transducer: a machine that takes a tree as input and produces another tree as output. In all generality we may view a tree transducer as a device that gives meaning to structured objects (i.e., a semantics defining device). Let us try to indicate this aspect of tree transducers.

Consider a ranked alphabet Σ. The elements of Σ may be viewed as "operators", i.e., symbols denoting operations (functions of several arguments). The rank of an operator stands for the number of arguments of the operation (note therefore that one operator may denote several operations). The operators of rank 0 have no arguments: they are (denote) constants. As an example, the ranked alphabet Σ with $\Sigma_0 = \{e, a, b\}$ and $\Sigma_2 = \{f\}$ may be viewed as consisting of three constants e, a and b and one binary operator f. From operators we may form "terms" or "expressions", like for instance $f(a, f(e, b))$, or perhaps, denoting f by $*$, $(a * (e * b))$. Obviously the terms are in one-to-one correspondence with the set T_Σ of trees over Σ. Thus the notions tree and term may be identified. Intuitively, terms denote structured objects, obtained by applying the operations to the constants.

Formally, meaning is given to operators and terms by way of an "interpretation". An interpretation of Σ consists of a "domain" B, for each element $a \in \Sigma_0$ an element $h_0(a)$ of B, and for each $k \geq 1$ and operator $a \in \Sigma_k$ an operation $h_k(a) : B^k \to B$. An interpretation of Σ is also called a "Σ-algebra" or "algebra of type Σ". An interpretation $(B, \{h_0(a)\}_{a \in \Sigma_0}, \{h_k(a)\}_{a \in \Sigma_k})$ determines a mapping $h : T_\Sigma \to B$ (giving an interpretation to each term as an element of B) as follows:

(i) for $a \in \Sigma_0$, $h(a) = h_0(a)$;

(ii) for $k \geq 1$ and $a \in \Sigma_k$,
$h(a[t_1 \cdots t_k]) = h_k(a)(h(t_1), \ldots, h(t_k))$.

(Such a mapping is also called a "homomorphism" from T_Σ into B). Thus the meaning of a tree is uniquely determined by the meaning of its subtrees and the interpretation of the operator applied to these subtrees. In general we can say that the meaning of a structured object is a function of the meanings of its substructures, the function being determined by the way the object is constructed from its substructures.

As an example, an interpretation of the above-mentioned ranked alphabet $\Sigma = \{e, a, b, f\}$ might for instance consist of a group B with unity $h_0(e)$, multiplication $h_2(f)$ and two specific elements $h_0(a)$ and $h_0(b)$. Or it might consist of $B = \{a, b\}^*$,

$h_0(e) = \lambda$, $h_0(a) = a$, $h_0(b) = b$ and $h_2(f)$ is concatenation. Note that in this case the mapping $h : T_\Sigma \to B$ is the yield!
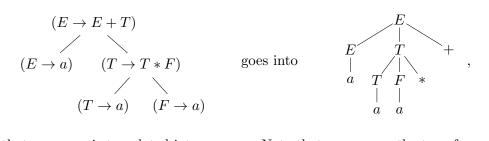
It is now easy to see that a deterministic bottom-up fta with input alphabet Σ is nothing else but a Σ-algebra with a finite domain (its set of states). Such an automaton may therefore be used as a semantic defining device in case there are only a finite number of possible semantical values. Obviously, in general, one needs an infinite number of semantical values. However, it is not attractive to consider arbitrary infinite domains B since this provides us with no knowledge about the structure of the elements of B. We therefore assume that the elements of B are structured objects: trees (or interpretations of them). Thus we consider Σ-algebras with domain T_Δ for some ranked alphabet Δ. Our complete semantics of T_Σ may then consist of two parts: an interpretation of T_Σ into T_Δ and an interpretation of T_Δ in some Δ-algebra. The interpretation of T_Σ into T_Δ may be realized by a tree transducer.

An example of an interpretation of T_Σ into T_Δ is the tree homomorphism of Definition 3.62. In fact each tree $s \in T_\Delta(X_k)$ may be viewed as an operation $\widetilde{s} : T_\Delta^k \to T_\Delta$, defined by

$$\widetilde{s}(s_1, \ldots, s_k) = s\langle x_1 \leftarrow s_1, \ldots, x_k \leftarrow s_k\rangle.$$

A tree homomorphism is then the same thing as an interpretation of Σ with domain T_Δ, where the allowable interpretations of the elements of Σ are the mappings $\widetilde{s}$ above. Note that these interpretations are very natural, since the interpretation of a tree is obtained by "applying a finite number of Δ-operators to the interpretations of its subtrees". To show the relevance of tree homomorphisms (and therefore tree transducers in general) to the semantics of context-free languages we consider the following very simple example.

Example 4.1. Consider a context-free grammar generating expressions by the rules $E \to E + T$, $T \to T * F$, $E \to a$, $T \to a$, $F \to a$ and $F \to (E)$. Suppose we want to translate each expression into the equivalent post-fix expression. To do this we consider the rule tree language corresponding to this grammar and apply to the rule trees in this language the tree homomorphism h defined by $h_2(E \to E + T) = E[x_1 x_2 +]$, $h_2(T \to T * F) = T[x_1 x_2 *]$, $h_0(E \to a) = E[a]$, $h_0(T \to a) = T[a]$, $h_0(F \to a) = F[a]$ and $h_1(F \to (E)) = F[x_1]$. Then the rule tree corresponding to an expression is translated into a tree whose yield is the corresponding post-fix expression. For instance,

so that $a + a * a$ is translated into $aaa * +$. Note that, moreover, the transformed tree is the derivation tree of the post-fix expression in the context-free grammar with the rules $E \to ET+$, $T \to TF*$, $E \to a$, $T \to a$, $F \to a$ and $F \to E$. Instead of interpreting this

derivation tree as its yield (the post-fix expression), one might also interpret it as, for instance, a sequence of machine code instructions, like "load a; load a; load a; multiply; add". $\qquad\square$

It is not difficult to see that the syntax-directed translation schemes of [A&U, I.3] correspond in some way to linear, nondeleting homomorphisms working on rule tree languages.

To arrive at our general notion of tree transducer, we combine the finite tree automaton and the tree homomorphism into a "tree homomorphism with states" or a "finite tree automaton with output". This tree transducer will not any more be an interpretation of T_Σ into T_Δ, but involves a generalization of this concept (although, by replacing T_Δ by some other set, it can again be formulated as an interpretation of T_Σ). Two ideas occur in this generalization.

Idea 4.2. The translation (meaning) of a tree may depend not only on the translation of its subtrees but also on certain properties of these subtrees. Assuming that these properties are recognizable (that is, the set of all trees having the property is in $RECOG$), they may be represented as states of a (deterministic) bottom-up fta. Thus we can combine the deterministic bottom-up fta and the tree homomorphism by associating to each symbol $a \in \Sigma_k$ a mapping $f_a : Q^k \to Q \times T_\Delta(X_k)$. This f_a may be split up into two mappings $\delta_a : Q^k \to Q$ and $h_a : Q^k \to T_\Delta(X_k)$. The δ-functions determine a mapping $\widehat{\delta} : T_\Sigma \to Q$, as for the bottom-up fta, and the h-functions determine an output-mapping $\widehat{h} : T_\Sigma \to T_\Delta$ by the formula (cf. the corresponding tree homomorphism formula):

$$\widehat{h}(a[t_1 \cdots t_k]) = h_a(\widehat{\delta}(t_1), \ldots, \widehat{\delta}(t_k))\langle x_1 \leftarrow \widehat{h}(t_1), \ldots, x_k \leftarrow \widehat{h}(t_k)\rangle.$$

Thus our tree transducer works through the tree in a bottom-up fashion just like the bottom-up fta, but at each step, it produces output by combining the output trees, already obtained from the subtrees, into one new output tree. Note that, if we allow our bottom-up tree transducer to be nondeterministic, then the above formula for $\widehat{h}$ is intuitively wrong (we need "deterministic substitution"). $\qquad\square$

Idea 4.3. To obtain the translation of the input tree one may need several different translations of each subtree. Suppose that one needs m different kinds of translation of each tree (where one of them is the "main meaning" and the others are "auxiliary meanings"), then these may be realized by m states of the transducer, say $q_1, \ldots, q_m$. The ith translation may then be specified by associating to each $a \in \Sigma_k$ a tree $h_{q_i}(a) \in T_\Delta(Y_{m,k})$, where $Y_{m,k} = \{y_{i,j} \mid 1 \le i \le m, 1 \le j \le k\}$. The ith translation of a tree $a[t_1 \cdots t_k]$ may then be defined by the formula

$$\widehat{h}_{q_i}(a[t_1 \cdots t_k]) = h_{q_i}(a)\langle y_{r,s} \leftarrow \widehat{h}_{q_r}(t_s)\rangle_{1 \le r \le m, 1 \le s \le k}.$$

Thus the ith translation of a tree is expressed in terms of all possible translations of its subtrees. Realizing such a translation in a bottom-up fashion would mean that we should compute all m possible translations of each tree in parallel, whereas working in a top-down way we know exactly from $h_{q_i}(a)$ which translations of which subtrees are

needed (note that, in general, not all elements of $Y_{m,k}$ appear in $h_{q_i}(a)$). Therefore, such a translation seems to be realized best by a top-down tree transducer. We note that the generalized syntax-directed translation scheme of [A&U, II.9.3] corresponds to such a top-down tree transducer working on a rule tree language. □

As already indicated in Example 4.1, tree transducers are of interest to the translation of context-free languages (in particular the context-free part of a programming language). For this reason we often restrict the tree transducer to a rule tree language, a local tree language or a recognizable tree language (the difference being slight: a projection). This restriction is also of interest from a linguistical point of view: a natural language may be described by a context-free set of kernel sentences to which transformations may be applied, working on the derivation trees (as for instance the transformation active → passive). The language then consists of all transformations of kernel sentences. We note that if derivation tree d_1 of sentence s_1 is transformed into tree d_2 with yield s_2, then the sentence s_2 is said to have "deep structure" d_1 and "surface structure" d_2.

4.2 Top-down and bottom-up finite tree transducers

Since tree transducers define tree transformations (recall Definition 2.10), we start by recalling some terminology concerning relations. We note first that, for ranked alphabets Σ and Δ, we shall identify any mapping $f : T_\Sigma \to T_\Delta$ with the tree transformation $\{(s,t) \mid f(s) = t\}$, and we shall identify any mapping $f : T_\Sigma \to \mathcal{P}(T_\Delta)$ with the tree transformation $\{(s,t) \mid t \in f(s)\}$.

Definition 4.4. Let Σ, Δ and Ω be ranked alphabets. If $M_1 \subseteq T_\Sigma \times T_\Delta$ and $M_2 \subseteq T_\Delta \times T_\Omega$, then the composition of M_1 and M_2, denoted by $M_1 \circ M_2$, is the tree transformation $\{(s,t) \in T_\Sigma \times T_\Omega \mid (s,u) \in M_1 \text{ and } (u,t) \in M_2 \text{ for some } u \in T_\Delta\}$. If F and G are classes of tree transformations, then $F \circ G$ denotes the class $\{M_1 \circ M_2 \mid M_1 \in F \text{ and } M_2 \in G\}$. □

Definition 4.5. Let M be a tree transformation from T_Σ into T_Δ. The inverse of M, denoted by M^{-1}, is the tree transformation $\{(t,s) \in T_\Delta \times T_\Sigma \mid (s,t) \in M\}$. □

Definition 4.6. Let M be a tree transformation and L a tree language. The image of L under M, denoted by $M(L)$, is the tree language $M(L) = \{t \mid (s,t) \in M \text{ for some } s \text{ in } L\}$. If M is a tree transformation from T_Σ into T_Δ, then the domain of M, denoted by $\text{dom}(M)$, is $M^{-1}(T_\Delta)$, and the range of M, denoted by $\text{range}(M)$, is $M(T_\Sigma)$. □

In Part (3) we already considered certain simple tree transformations: relabelings and tree homomorphisms.

Notation 4.7. We shall use *REL* to denote the class of all relabelings, *HOM* to denote the class of all tree homomorphisms, and *LHOM* to denote the class of linear tree homomorphisms. □

Moreover we want to view each finite tree automaton as a simple "checking" tree transducer, which, given some input tree, produces the same tree as output if it belongs to the tree language recognized by the fta, and produces no output if not.

Definition 4.8. Let Σ be a ranked alphabet. A tree transformation $R \subseteq T_\Sigma \times T_\Sigma$ is called a <u>finite tree automaton restriction</u> if there is a recognizable tree language L such that $R = \{(t, t) \mid t \in L\}$. If M is an fta, then we shall denote the finite tree automaton restriction $\{(t, t) \mid t \in L(M)\}$ by $T(M)$. We shall use FTA to denote the class of all finite tree automaton restrictions. $\qquad\square$

Exercise 4.9. Prove that the classes of tree transformations REL, HOM and FTA are each closed under composition. Show that REL and FTA are also closed under inverse. $\quad\square$

Before defining tree transducers we first discuss a very general notion of tree rewriting system that can be used to define both tree transducers and tree grammars. The reason to introduce these tree rewriting systems is that recursive definitions like those for the finite tree automata and tree homomorphisms tend to become very cumbersome when used for tree transducers, whereas rewriting systems are more "machine-like" and therefore easier to visualize. To arrive at the notion of tree rewriting system we first generalize the notion of string rewriting system to allow for the use of rule "schemes". Recall the set X of variables from Definition 3.61.

Definition 4.10. A <u>rewriting system with variables</u> is a pair $G = (\Delta, R)$, where Δ is an alphabet and R a finite set of "rule schemes". A <u>rule scheme</u> is a triple (v, w, D) such that, for some $k \geq 0$, v and w are strings over $\Delta \cup \overline{X_k}$ and D is a mapping from X_k into $\mathcal{P}(\Delta^*)$. Whenever D is understood, (v, w, D) is denoted by $v \to w$. For $1 \leq i \leq k$, the language $D(x_i)$ is called the <u>range</u> (or domain) of the variable x_i.

A relation $\underset{G}{\Longrightarrow}$ on Δ^* is defined as follows. For strings $s, t \in \Delta^*$, $s \underset{G}{\Longrightarrow} t$ if and only if there exists a rule scheme (v, w, D) in R, strings $\phi_1, \ldots, \phi_k$ in $D(x_1), \ldots, D(x_k)$ respectively (where X_k is the domain of D), and strings α and β in Δ^* such that

$$s = \alpha \cdot v\langle x_1 \leftarrow \phi_1, \ldots, x_k \leftarrow \phi_k\rangle \cdot \beta \qquad \text{and}$$
$$t = \alpha \cdot w\langle x_1 \leftarrow \phi_1, \ldots, x_k \leftarrow \phi_k\rangle \cdot \beta \quad.$$

As usual $\underset{G}{\overset{*}{\Longrightarrow}}$ denotes the transitive-reflexive closure of $\underset{G}{\Longrightarrow}$. $\qquad\square$

For convenience we shall, in what follows, use the word "rule" rather than "rule scheme".

Of course, in a rewriting system with variables, the ranges of the variables should be specified in some effective way (note that we would like the relation $\Rightarrow$ to be decidable). In what follows we shall only use the case that the variables range over recognizable tree languages.

Examples 4.11.

(1) Consider the rewriting system with variables $G = (\Delta, R)$, where $\Delta = \{a, b, c\}$ and R consists of the one rule $ax_1c \to aax_1bcc$, where $D(x_1) = b^*$. Then, for instance, $aabbcc \Rightarrow aaabbbccc$ (by application of the ordinary rewriting rule $abbc \to aabbbcc$ obtained by substituting bb for x_1 in the rule above). It is easy to see that $\{w \in \Delta^* \mid abc \overset{*}{\Rightarrow} w\} = \{a^n b^n c^n \mid n \geq 1\}$.

(2) Consider the rewriting system with variables $G = (\Delta, R)$, where $\Delta = \{[,], *, 1\}$ and R consists of the rules $[x_1 * x_2 1] \to [x_1 * x_2]x_1$ and $[x_1 * 1] \to x_1$, where in both rules $D(x_1) = D(x_2) = 1^*$. It is easy to see that, for arbitrary $u, v, w \in 1^*$, $[u * v] \overset{*}{\Rightarrow} w$ iff w is the product of u and v (in unary notation).

(3) The two-level grammar used to describe Algol 68 may be viewed as a rewriting system with variables. The variables (= meta notions) range over context-free languages, specified by the meta grammar. $\qquad\square$

By specializing to trees we obtain the notion of tree rewriting system.

Definition 4.12. A rewriting system with variables $G = (\Delta, R)$ is called a <u>tree rewriting system</u> if

(i) $\Delta = \Sigma \cup \{[,]\}$ for some ranked alphabet Σ;

(ii) for each rule (v, w, D) in R, v and w are trees in $T_\Sigma(X_k)$ and, for $1 \leq i \leq k$, $D(x_i) \subseteq T_\Sigma$ (where X_k is the domain of D). $\qquad\square$

It should be clear that, for a tree rewriting system $G = (\Sigma \cup \{[,]\}, R)$, if $s \in T_\Sigma$ and $s \underset{G}{\Rightarrow} t$, then $t \in T_\Sigma$. In fact, the application of a rule to a tree consists of replacing some piece in the middle of the tree by some other piece, where the variables indicate how the subtrees of the old piece should be connected to the new one. As an example, if we have a rule $a[b[x_1 x_2]b[x_3 d]] \to b[x_2 a[x_1 d x_2]]$, then the application of this rule to a tree t (if possible) consists of replacing a subtree of t of the form
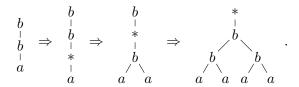

where t_1, t_2 and t_3 are in the ranges of x_1, x_2 and x_3. Thus t is of the form $\alpha a[b[t_1 t_2]b[t_3 d]]\beta$ and is transformed into $\alpha b[t_2 a[t_1 d t_2]]\beta$.

Example 4.13. Let $\Sigma_0 = \{a\}$, $\Sigma_1 = \{b\}$, $\Delta_0 = \{a\}$, $\Delta_2 = \{b\}$, $\Omega_0 = \{a\}$, $\Omega_1 = \{*, b\}$ and $\Omega_2 = \{b\}$.

(i) Consider the tree rewriting system $G = (\Omega \cup \{[,]\}, R)$, where R consists of the rules
$$a \to *[a],$$
$$b[*[x_1]] \to *[b[x_1 x_1]],$$
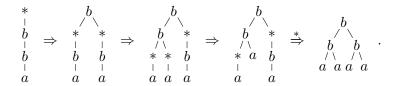and $D(x_1) = T_\Delta$. Then, for instance,

$$
\begin{array}{c}
b \\ | \\ b \\ | \\ a
\end{array}
\;\Rightarrow\;
\begin{array}{c}
b \\ | \\ b \\ | \\ * \\ | \\ a
\end{array}
\;\Rightarrow\;
\begin{array}{c}
b \\ | \\ * \\ | \\ b \\ \diagup\diagdown \\ a \quad a
\end{array}
\;\Rightarrow\;
\begin{array}{c}
* \\ | \\ b \\ \diagup \quad \diagdown \\ b \qquad b \\ \diagup\diagdown \; \diagup\diagdown \\ a \; a \;\; a \; a
\end{array}
\;.
$$

It is easy to see that, if h is the tree homomorphism from T_Σ into T_Δ defined by $h_0(a) = a$ and $h_1(b) = b[x_1 x_1]$, then, for $s \in T_\Sigma$ and $t \in T_\Delta$, $h(s) = t$ iff $s \overset{*}{\Rightarrow} *[t]$.

(ii) Consider the tree rewriting system $G' = (\Omega \cup \{[,]\}, R')$, where R' consists of the rules

$$*[b[x_1]] \to b[*[x_1] * [x_1]],$$

$$*[a] \to a,$$

and $D(x_1) = T_\Sigma$. Then, for instance,

$$
\begin{array}{c}
* \\ | \\ b \\ | \\ b \\ | \\ a
\end{array}
\;\Rightarrow\;
\begin{array}{c}
b \\ \diagup\diagdown \\ * \;\; * \\ | \;\;\; | \\ b \;\; b \\ | \;\;\; | \\ a \;\; a
\end{array}
\;\Rightarrow\;
\begin{array}{c}
b \\ \diagup\diagdown \\ b \;\; * \\ \diagup\diagdown \; | \\ * \; * \; b \\ | \; | \; | \\ a \, a \, a
\end{array}
\;\Rightarrow\;
\begin{array}{c}
b \\ \diagup\diagdown \\ b \;\; * \\ \diagup\diagdown \; | \\ * \; a \; b \\ | \;\;\;\; | \\ a \;\;\;\; a
\end{array}
\;\overset{*}{\Rightarrow}\;
\begin{array}{c}
b \\ \diagup\diagdown \\ b \;\; b \\ \diagup\diagdown \; \diagup\diagdown \\ a \, a \; a \, a
\end{array}
\;.
$$

It is easy to see that, if h is the homomorphism defined above, then, for $s \in T_\Sigma$ and $t \in T_\Delta$, $h(s) = t$ iff $*[s] \overset{*}{\Rightarrow} t$. $\qquad\square$

The tree transducers to be defined will be a generalization of the generalized sequential machine working on strings, which is essentially a finite automaton with output.

A (nondeterministic) generalized sequential machine is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, S, F)$, where Q is the set of states, Σ is the input alphabet, Δ the output alphabet, δ is a mapping $Q \times \Sigma \to \mathcal{P}(Q \times \Delta^*)$, S is a set of initial states and F a set of final states. Intuitively, if $\delta(q, a)$ contains (q', w) then, in state q and scanning input symbol a, the machine M may go into state q' and add w to the output. Formally we may define the functioning of M in several ways. As already said, the recursive definition (as for the fta) is too cumbersome, although it is the most exact one (and should be used in very formal proofs). The other way is to describe the sequence of configurations the machine goes through during the translation of the input string. A configuration is usually a triple (v, q, s), where v is the output generated so far, q is the state and s is the rest of the input. If $s = as_1$, then the next configuration might be (vw, q', s_1). A useful variation of this is to replace (v, q, s) by the string $vqs \in \Delta^* Q \Sigma^*$. The next configuration can now be obtained by applying the string rewriting rule $qa \to wq'$, thus $vqas_1 \Rightarrow vwq's_1$. Replacing δ by a corresponding set of rewriting rules, the string translation realized by M can be defined as $\{(v_1, v_2) \mid q_0 v_1 \overset{*}{\Rightarrow} v_2 q_f$ for some $q_0 \in S$ and $q_f \in F\}$.

Let us first consider the bottom-up generalization of this machine to trees, which is conceptually easier than the top-down version, although perhaps less interesting. The bottom-up finite tree transducer goes through the input tree in the same way as the bottom-up fta, at each step producing a piece of output to which the already generated

output is concatenated. The transducer arrives at a node of rank k with a sequence of k states and a sequence of k output trees (one state and one output tree for each direct subtree of the node). The sequence of states and the label at the node determine (nondeterministically) a new state and a piece of output containing the variables $x_1, \ldots, x_k$. The transducer processes the node by going into the new state and computing a new output tree by substituting the k output trees for $x_1, \ldots, x_k$ in the piece of output. There should be start states and output for each node of rank 0. If the transducer arrives at the top of the tree in a final state, then the computed output tree is the transformation of the input tree. (Cf. the story in Idea 4.2).

To be able to put the states of the transducer as labels on trees, we make them into symbols of rank 1. The configurations of the bottom-up tree transducer will be elements of $T_\Sigma(Q[T_\Delta])$, [†] and the steps of the tree transducer (including the start steps) are modelled by the application of tree rewriting rules to these configurations.

We now give the formal definition.

Definition 4.14. A <u>bottom-up (finite) tree transducer</u> is a structure $M = (Q, \Sigma, \Delta, R, Q_d)$, where

Q is a ranked alphabet (of <u>states</u>), such that all elements of Q have rank 1 and no other ranks;

Σ is a ranked alphabet (of <u>input symbols</u>);

Δ is a ranked alphabet (of <u>output symbols</u>), $Q \cap (\Sigma \cup \Delta) = \emptyset$;

Q_d is a subset of Q (the set of <u>final states</u>); and

R is a finite set of <u>rules</u> of one of the forms (i) or (ii):

   (i) $a \to q[t]$, where $a \in \Sigma_0$, $q \in Q$ and $t \in T_\Delta$;

   (ii) $a[q_1[x_1] \cdots q_k[x_k]] \to q[t]$, where $k \geq 1$, $a \in \Sigma_k$, $q_1, \ldots, q_k, q \in Q$ and $t \in T_\Delta(X_k)$.

M is viewed as a tree rewriting system over the ranked alphabet $Q \cup \Sigma \cup \Delta$ with R as the set of rules, such that the range of each variable occurring in R is T_Δ. Therefore the relations $\underset{M}{\Longrightarrow}$ and $\underset{M}{\overset{*}{\Longrightarrow}}$ are well defined according to Definition 4.10.

The <u>tree transformation realized by M</u>, denoted by $T(M)$ or simply M, is

$$\{(s, t) \in T_\Sigma \times T_\Delta \mid s \underset{M}{\overset{*}{\Longrightarrow}} q[t] \text{ for some } q \text{ in } Q_d\}. \qquad \square$$

We shall abbreviate "(finite) tree transducer" by "ftt".

Remark 4.15. Note that $T(M)$ is also denoted by M. In general, we shall often make no distinction between a tree transducer and the tree transformation it realizes. Hopefully this will not lead to confusion. $\qquad \square$

Definition 4.16. The class of tree transformations realized by bottom-up ftt will be denoted by B. An element of B will be called a <u>bottom-up tree transformation</u>. $\qquad \square$

[†]Note that $Q[T_\Delta] = \{q[t] \mid q \in Q \text{ and } t \in T_\Delta\}$.

Example 4.17. An example of a bottom-up ftt realizing a homomorphism was given in Example 4.13(i) (it had one state $*$). $\square$

Example 4.18. Consider the bottom-up ftt $M = (Q, \Sigma, \Delta, R, Q_d)$, where $Q = \{q_0, q_1\}$, $\Sigma_0 = \{a, b\}$, $\Sigma_2 = \{f, g\}$, $\Delta_0 = \{a, b\}$, $\Delta_2 = \{m, n\}$, $Q_d = \{q_0\}$ and the rules are

$$a \rightarrow q_0[a], \qquad\qquad b \rightarrow q_0[b],$$
$$f[q_i[x_1]q_j[x_2]] \rightarrow q_{1-i}[m[x_1 x_1]], \qquad g[q_i[x_1]q_j[x_2]] \rightarrow q_{1-i}[n[x_1 x_1]],$$
$$f[q_i[x_1]q_j[x_2]] \rightarrow q_{1-j}[m[x_2 x_2]], \qquad g[q_i[x_1]q_j[x_2]] \rightarrow q_{1-j}[n[x_2 x_2]],$$

for all $i, j \in \{0, 1\}$. The transformation realized by M may be described by saying that, given some input tree t, M selects some path of even length through t, relabels f by m and g by n and then doubles every subtree. For example $f[a[g[ab]]]$ may be transformed into the tree $m[n[bb]n[bb]]$ corresponding to the path fgb. The tree $g[ab]$ is not in the domain of M. $\square$

Exercise 4.19. Construct bottom-up tree transducers M_1 and M_2 such that

(i) $\{(\text{yield}(s), \text{yield}(t)) \mid (s, t) \in T(M_1)\} = \{(a(cd)^n fe^n b, ac^n fd^{2n} b) \mid n \geq 0\}$;

(ii) M_2 deletes, given an input tree t, all subtrees t' of t such that $\text{yield}(t') \in a^+ b^+$. $\square$

Exercise 4.20. Give a recursive definition of the transformation realized by a bottom-up tree transducer (without using the notion of tree rewriting system). $\square$

Exercise 4.21. Given a bottom-up ftt M with input alphabet Σ, find a suitable Σ-algebra such that M may be viewed as an interpretation of Σ into this Σ-algebra (cf. Section 4.1). $\square$

We now define some subclasses of the class of bottom-up tree transformations.

Definition 4.22. Let Σ be a ranked alphabet and $k \geq 0$. A tree t in $T_\Sigma(X_k)$ is <u>linear</u> if each element of X_k occurs at most once in t. The tree t is called <u>nondeleting with respect to X_k</u> if each element of X_k occurs at least once in t. $\square$

Definition 4.23. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up ftt.
M is called <u>linear</u> if the right hand side of each rule in R is linear.
M is called <u>nondeleting</u> if the right hand side of each rule in R is nondeleting with respect to X_k, where k is the rank of the input symbol in the left hand side.
M is called <u>one-state</u> (or <u>pure</u>) if Q is a singleton.
M is called (partial) <u>deterministic</u> if

(i) for each $a \in \Sigma_0$ there is at most one rule in R with left hand side a, and

(ii) for each $k \geq 1$, $a \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$ there is at most one rule in R with left hand side $a[q_1[x_1] \cdots q_k[x_k]]$.

M is called <u>total deterministic</u> if (i) and (ii) hold with "at most one" replaced by "exactly one" and $Q_d = Q$. $\square$

Notation 4.24. The same terminology will be applied to the transformations realized by such transducers. Thus, for instance, a linear deterministic bottom-up tree transformation is one that can be realized by a linear deterministic bottom-up ftt.

The classes of tree transformations obtained by putting one or more of the above restrictions on the bottom-up tree transducers will be denoted by adding the symbols L, N, P, D and D_t (standing for linear, nondeleting, pure, deterministic, and total deterministic respectively) to the letter B. Thus the class of linear deterministic bottom-up tree transformations is denoted by LDB. $\square$

Example 4.25. Let $\Sigma_0 = \{e\}$, $\Sigma_1 = \{a, f\}$, $\Delta_0 = \{e\}$, $\Delta_1 = \{a, b\}$ and $\Delta_2 = \{f\}$. Consider the bottom-up ftt $M = (Q, \Sigma, \Delta, R, Q_d)$, where $Q = Q_d = \{*\}$ and R consists of the rules

$$e \to *[e],$$
$$a[*[x_1]] \to *[a[x_1]], \qquad\qquad a[*[x_1]] \to *[b[x_1]],$$
$$f[*[x_1]] \to *[f[x_1 x_1]].$$

Then $M \in PNB$. $\square$

Let us make the following remarks about the concepts defined in Definition 4.23.

Remarks 4.26.

(1) Deletion is different from erasing. A rule may be called erasing if its right hand side belongs to $Q[X]$. Thus, symbols of rank 0 cannot be erased. Symbols of rank 1 can be erased without any deletion, but symbols of rank $k \geq 2$ can only be erased by deleting also $k - 1$ subtrees. Thus a nondeleting tree transducer is still able to erase symbols of rank 1.

(2) The one-state bottom-up tree transformations correspond intuitively to the finite substitutions in the string case.

(3) The total deterministic bottom-up ftt realize tree transformations which are total functions. $\square$

Exercise 4.27. Show that, in the definition of "(partial) deterministic", we may replace the phrase "at most one" by "exactly one" without changing the corresponding class DB of deterministic bottom-up tree transformations. $\square$

In the next theorem we show that all relabelings, finite tree automaton restrictions and tree homomorphisms are realizable by bottom-up ftt.

Theorem 4.28.

(1) $REL \subseteq PNLB$,

(2) $FTA \subseteq NLDB$,

(3) $HOM = PD_tB$ and $LHOM = PLD_tB$.

Proof. (1) Let r be a relabeling from T_Σ into T_Δ. Thus r is determined by a family of mappings $r_k : \Sigma_k \to \mathcal{P}(\Delta_k)$. Obviously the following bottom-up ftt realizes r: $M = (\{*\}, \Sigma, \Delta, R, \{*\})$, where R is constructed as follows:

(i) for $a \in \Sigma_0$, if $b \in r_0(a)$, then $a \to *[b]$ is in R;

(ii) for $k \geq 1$ and $a \in \Sigma_k$, if $b \in r_k(a)$, then
$a[*[x_1] \cdots *[x_k]] \to *[b[x_1 \cdots x_k]]$ is in R.

Clearly $M \in PNLB$.

(2) From the definition of *FTA* and from Part (3) it follows that we need only consider a deterministic bottom-up fta $M = (Q, \Sigma, \delta, s, F)$ and show that $T(M) = \{(t, t) \mid t \in L(M)\}$ is realized by a bottom-up ftt. Consider the bottom-up ftt $\widetilde{M} = (Q, \Sigma, \Sigma, R, F)$, where R is constructed as follows:

(i) for $a \in \Sigma_0$, $a \to q[a]$ is in R, where $q = s_a$;

(ii) for $k \geq 1$ and $a \in \Sigma_k$, if $\delta_a^k(q_1, \ldots, q_k) = q$,
then $a[q_1[x_1] \cdots q_k[x_k]] \to q[a[x_1 \cdots x_k]]$ is in R.

Clearly $\widetilde{M}$ realizes $T(M)$ and $\widetilde{M} \in NLDB$ (the determinism of $\widetilde{M}$ follows from that of M).

(3) We first show that $HOM \subseteq PD_tB$ (and $LHOM \subseteq PLD_tB$). An example of this was already given in Example 4.13(i). Let h be a tree homomorphism from T_Σ into T_Δ determined by the mappings $h_k : \Sigma_k \to T_\Delta(X_k)$. Consider the bottom-up ftt $M = (\{*\}, \Sigma, \Delta, R, \{*\})$, where R contains the following rules:

(i) for $a \in \Sigma_0$, $a \to *[h_0(a)]$ is in R;

(ii) for $k \geq 1$ and $a \in \Sigma_k$, the rule $a[*[x_1] \cdots *[x_k]] \to *[h_k(a)]$ is in R.

Obviously M is in PD_tB (and linear, if h is linear). Let us prove that M realizes h. Thus we have to show that, for $s \in T_\Sigma$ and $t \in T_\Delta$, $h(s) = t$ iff $s \overset{*}{\Rightarrow} *[t]$.

The proof is by induction on s. The case $s \in \Sigma_0$ is clear. Now let $s = a[s_1 \cdots s_k]$. Suppose that $h(s) = t$. Then, by definition of h, $t = h_k(a)\langle x_1 \leftarrow h(s_1), \ldots, x_k \leftarrow h(s_k)\rangle$. By induction, $s_i \overset{*}{\Rightarrow} *[h(s_i)]$ for all i, $1 \leq i \leq k$. Hence (but note that formally this needs a proof) $a[s_1 \cdots s_k] \overset{*}{\Rightarrow} a[*[h(s_1)] \cdots *[h(s_k)]]$. But, by rule (ii) above, $a[*[h(s_1)] \cdots *[h(s_k)]] \Rightarrow *[h_k(a)\langle x_1 \leftarrow h(s_1), \ldots, x_k \leftarrow h(s_k)\rangle]$. Consequently $s \overset{*}{\Rightarrow} *[t]$.

Now suppose that $s = a[s_1 \cdots s_k] \overset{*}{\Rightarrow} *[t]$. Then (and again this needs a formal proof) there are trees $t_1, \ldots, t_k$ such that $s_i \overset{*}{\Rightarrow} *[t_i]$ for $1 \leq i \leq k$ and $a[s_1 \cdots s_k] \overset{*}{\Rightarrow} a[*[t_1] \cdots *[t_k]] \Rightarrow *[h_k(a)\langle x_1 \leftarrow t_1, \ldots, x_k \leftarrow t_k\rangle] = *[t]$. By induction, $t_i = h(s_i)$ for all i, $1 \leq i \leq k$. Hence $t = h_k(a)\langle x_1 \leftarrow h(s_1), \ldots, x_k \leftarrow h(s_k)\rangle = h(s)$.

This proves that $(L)HOM \subseteq P(L)D_tB$. To show the converse, consider a one-state total deterministic bottom-up tree transducer $M = (\{*\}, \Sigma, \Delta, R, \{*\})$. Define the tree homomorphism h from T_Σ into T_Δ as follows:

(i) for $a \in \Sigma_0$, $h_0(a)$ is the tree t occurring in the (unique) rule $a \to *[t]$ in R;

(ii) for $k \geq 1$ and $a \in \Sigma_k$, $h_k(a)$ is the tree t occurring in the (unique) rule $a[*[x_1] \cdots *[x_k]] \to *[t]$ in R.

Then, obviously, by the same proof as above, $h = T(M)$. □

Exercise 4.29. Prove that the domain of a bottom-up tree transformation is a recognizable tree language, and vice-versa. □

Let us now consider the top-down generalization of the generalized sequential machine.

The top-down finite tree transducer goes through the input tree in the same way as the top-down fta, at each step producing a piece of output to which the (unprocessed) rest of the input is concatenated. Note therefore that the transducer not really "goes through" the input tree in the same way as the bottom-up ftt does, since in the top-down case the rest of the input may be modified (deleted, permuted, copied) during translation, whereas in the bottom-up case the rest of the input is unmodified during translation. The top-down transducer arrives at a node of rank k in a certain state; on that moment the configuration is an element of $T_\Delta(Q[T_\Sigma])$, where Σ and Δ are the input and output alphabet, and Q the set of states. The state and the label at the node determine (nondeterministically) a piece of output containing the variables $x_1, \ldots, x_k$, and states with which to continue the translation of the subtrees. These states are also specified in the piece of output, which is in fact a tree in $T_\Delta(Q[X_k])$, where an occurrence of $q[x_i]$ means that the processing of the ith subtree should, at this point, be continued in state q. The transducer processes the node by replacing it and its direct subtrees by the piece of output, in which the k subtrees are substituted for the variables $x_1, \ldots, x_k$. The processing of (all copies of) the subtrees is continued as indicated above.

The transducer starts at the root of the input tree in some initial state. There should be final states and output for each node of rank 0. If the transducer arrives in a final state at each leaf, then it replaces each leaf by the final output, and the resulting tree is the transformation of the input tree. (Cf. the story in Idea 4.3). The steps of the transducer, including the final steps, are modelled by the application of rewriting rules to the elements of $T_\Delta(Q[T_\Sigma])$.

We now give the formal definition.

Definition 4.30. A <u>top-down (finite) tree transducer</u> is a structure
$M = (Q, \Sigma, \Delta, R, Q_d)$, where Q, Σ and Δ are as for the bottom-up ftt,
$\quad Q_d \quad$ is a subset of Q (the set of <u>initial states</u>), and
$\quad R \quad$ is a finite set of <u>rules</u> of one of the forms (i) or (ii):
$\qquad$ (i) $q[a[x_1 \cdots x_k]] \to t$, where $k \geq 1$, $a \in \Sigma_k$, $q \in Q$ and $t \in T_\Delta(Q[X_k])$;
$\qquad$ (ii) $q[a] \to t$, where $q \in Q$, $a \in \Sigma_0$ and $t \in T_\Delta$.

M is viewed as a tree rewriting system over the ranked alphabet $Q \cup \Sigma \cup \Delta$ with R as the set of rules, such that the range of each variable in X is T_Σ.

The <u>tree transformation realized by M</u>, denoted by $T(M)$ or simply M, is
$\{(s, t) \in T_\Sigma \times T_\Delta \mid q[s] \overset{*}{\underset{M}{\Longrightarrow}} t \text{ for some } q \text{ in } Q_d\}$. □

Definition 4.31. The class of tree transformations realized by top-down ftt will be denoted by T. An element of T will be called a <u>top-down tree transformation</u>. □

Example 4.32. An example of a top-down ftt realizing a homomorphism was given in Example 4.13(ii) (it had one state $*$). $\square$

The next example is a top-down ftt computing the formal derivative of an arithmetic expression.

Example 4.33. Consider the top-down ftt $M = (Q, \Sigma, \Delta, R, Q_d)$, where $\Sigma_0 = \{a, b\}$, $\Delta_0 = \{a, b, 0, 1\}$, $\Sigma_1 = \Delta_1 = \{-, \underline{\sin}, \underline{\cos}\}$, $\Sigma_2 = \Delta_2 = \{+, *\}$, $Q = \{q, i\}$, $Q_d = \{q\}$,

the rules for q are

$$q[+[x_1 x_2]] \rightarrow +[q[x_1]q[x_2]],$$
$$q[*[x_1 x_2]] \rightarrow +[*[q[x_1]i[x_2]] * [i[x_1]q[x_2]]],$$
$$q[-[x_1]] \rightarrow -[q[x_1]],$$
$$q[\underline{\sin}[x_1]] \rightarrow *[\underline{\cos}[i[x_1]]q[x_1]],$$
$$q[\underline{\cos}[x_1]] \rightarrow *[-[\underline{\sin}[i[x_1]]]q[x_1]],$$
$$q[a] \rightarrow 1, \ q[b] \rightarrow 0,$$

and the rules for i are

$$i[+[x_1 x_2]] \rightarrow +[i[x_1]i[x_2]],$$
$$i[*[x_1 x_2]] \rightarrow *[i[x_1]i[x_2]],$$
$$i[-[x_1]] \rightarrow -[i[x_1]],$$
$$i[\underline{\sin}[x_1]] \rightarrow \underline{\sin}[i[x_1]],$$
$$i[\underline{\cos}[x_1]] \rightarrow \underline{\cos}[i[x_1]],$$
$$i[a] \rightarrow a \text{ and } i[b] \rightarrow b.$$

Then $(t, s) \in T(M)$ iff s is the formal derivative of t with respect to a. For instance, $q[*[+[ab] - [a]]] \overset{*}{\Rightarrow} +[*[+[10] - [a]] * [+[ab] - [1]]]$. Note that $i[t_1] \overset{*}{\Rightarrow} t_2$ iff $t_1 = t_2$ $(t_1, t_2 \in T_\Sigma)$. $\square$

Exercise 4.34. Let $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{\sim\}$ and $\Sigma_2 = \{\wedge, \vee\}$. T_Σ may be viewed as the set of all boolean expressions over the boolean variables a and b, using negation, conjunction and disjunction. Write a top-down tree transducer which transforms every boolean expression into an equivalent one in which a and b are the only subexpressions which may be negated. $\square$

Exercise 4.35. (i) Give a recursive definition of the transformation realized by a top-down ftt. (ii) Find a suitable Σ-algebra such that the top-down ftt may be viewed as an interpretation of Σ into this Σ-algebra. $\square$

As in the bottom-up case we define some subclasses of T.

Definition 4.36. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a top-down tree transducer. The definitions of linear, nondeleting and one-state are identical to the bottom-up ones

in Definition 4.23.

M is called (partial) <u>deterministic</u> if

(i) Q_d is a singleton;

(ii) for each $q \in Q$, $k \geq 1$, and $a \in \Sigma_k$, there is at most one rule in R with left hand side $q[a[x_1 \cdots x_k]]$;

(iii) for each $q \in Q$ and $a \in \Sigma_0$ there is at most one rule in R with left hand side $q[a]$.

M is called <u>total deterministic</u> if (i), (ii) and (iii) hold with "at most one" replaced by "exactly one". $\qquad\square$

Notation 4.24 also applies to the top-down case. Thus, PLT is the class of one-state linear top-down tree transformations.

Example 4.37. Let $\Sigma_0 = \{e\}$, $\Sigma_1 = \{a, f\}$, $\Delta_0 = \{e\}$, $\Delta_1 = \{a, b\}$ and $\Delta_2 = \{f\}$. Consider the top-down tree transducer $M = (Q, \Sigma, \Delta, R, Q_d)$ with $Q = Q_d = \{*\}$ and R consists of the rules

$$* [f[x_1]] \to f[*[x_1] * [x_1]],$$
$$* [a[x_1]] \to a[*[x_1]], \quad *[a[x_1]] \to b[*[x_1]],$$
$$* [e] \to e.$$

Then $M \in PNT$. $\qquad\square$

Remarks 4.26 also apply to the top-down case.

Exercise 4.38. Show that, in the definition of "(partial) deterministic" top-down ftt, we may replace in (ii) the phrase "at most one" by "exactly one" without changing DT. $\quad\square$

The next theorem shows that all relabelings, finite tree automaton restrictions and tree homomorphisms are realizable by top-down tree transducers (cf. Theorem 4.28). Note therefore that these tree transformations are not specifically bottom-up or top-down.

Theorem 4.39.

(1) $REL \subseteq PNLT$,

(2) $FTA \subseteq NLT$,

(3) $HOM = PD_t T$ and $LHOM = PLD_t T$.

Proof. Exercise. $\qquad\square$

In what follows we shall need one other type of tree transformation which corresponds to the ordinary sequential machine in the string case (which translates each input symbol into one output symbol). It is a combination of an fta and a relabeling.

Definition 4.40. A <u>top-down</u> (resp. <u>bottom-up</u>) <u>finite state relabeling</u> is a (tree transformation realized by a) top-down (resp. bottom-up) tree transducer $M = (Q, \Sigma, \Delta, R, Q_d)$

in which all rules are of the form $q[a[x_1 \cdots x_k]] \to b[q_1[x_1] \cdots q_k[x_k]]$ with $q, q_1, \ldots, q_k \in Q$, $a \in \Sigma_k$ and $b \in \Delta_k$, or of the form $q[a] \to b$ with $q \in Q$, $a \in \Sigma_0$ and $b \in \Delta_0$ (resp. of the form $a[q_1[x_1] \cdots q_k[x_k]] \to q[b[x_1 \cdots x_k]]$ with $q, q_1, \ldots, q_k \in Q$, $a \in \Sigma_k$ and $b \in \Delta_k$, or of the form $a \to q[b]$ with $q \in Q$, $a \in \Sigma_0$ and $b \in \Delta_0$). $\qquad\square$

It is clear that the classes of top-down and bottom-up finite state relabelings coincide. This class will be denoted by $QREL$. The classes of deterministic top-down and deterministic bottom-up finite state relabelings obviously do not coincide. They will be denoted by $DTQREL$ and $DBQREL$ respectively. Note that $FTA \cup REL \subseteq QREL \subseteq NLB \cap NLT$.

Apart from the tree transformation realized by a tree transducer we will also be interested in the image of a recognizable tree language under a tree transformation and the yield of that image.

Definition 4.41. Let F be a class of tree transformations. An F-surface tree language is a language $M(L)$ with $M \in F$ and $L \in RECOG$. An F-target language is the yield of an F-surface language. An F-translation is a string relation $\{(\mathrm{yield}(s), \mathrm{yield}(t)) \mid (s,t) \in M$ and $s \in L\}$ for some $M \in F$ and $L \in RECOG$.

The classes of F-surface and F-target languages will be denoted by F-Surface and F-Target respectively. $\qquad\square$

It is clear that, for all classes F discussed so far, since the identity transformation is in F, $RECOG \subseteq F$-Surface, and so $CFL \subseteq F$-Target. Moreover it is clear from the proof of Theorem 3.64 that if $HOM \subseteq F$, then the above inclusions are proper.

4.3 Comparison of B and T, the nondeterministic case

The main differences between the bottom-up and the top-down tree transducer are the following.

Property (B). Nondeterminism followed by copying.
A bottom-up ftt has the ability of first processing an input subtree nondeterministically and then copying the resulting output tree.

Property (T). Copying followed by different processing (by nondeterminism or by different states).
A top-down ftt has the ability of first copying an input subtree and then treating the resulting copies differently.

Property (B$'$). Checking followed by deletion.
A bottom-up ftt has the ability of first processing an input subtree and then deleting the resulting output subtree. In other words, depending on a (recognizable) property of the input subtree, it can decide whether to delete the output subtree or do something else with it.

It should be intuitively clear that top-down ftt do not possess properties (B) and (B$'$), whereas bottom-up ftt do not have property (T). We now show that these differences

also result in differences in the corresponding classes of tree transformations.

Notation 4.42. For any alphabet Σ, not containing the brackets [and], we define a function $m : \Sigma^+ \to (\Sigma \cup \{[,]\})^+$ as follows: for $a \in \Sigma$ and $w \in \Sigma^+$, $m(a) = a$ and $m(aw) = a[m(w)]$. For instance, $m(aab) = a[a[b]]$.

Note that m is a kind of converse to the mapping f_{td} discussed after Definition 2.21. A tree of the form $m(w)$ will also be called a monadic tree. $\quad\square$

Theorem 4.43. The classes of bottom-up and top-down tree transformations are incomparable. In particular, there are tree transformations in $PNB - T$ and $PNT - B$.

Proof.
(1) Consider the bottom-up ftt M of Example 4.25. M is in PNB and is a typical example of an ftt having property (B). It is intuitively clear that M is not realizable by a top-down ftt. In fact, consider for each $n \geq 1$ the specific input tree $f[m(a^n e)]$. This tree is nondeterministically transformed by M into all trees of the form $f[m(we)m(we)]$, where w is a string over $\{a,b\}$ of length n. Suppose that the top-down ftt $N = (Q', \Sigma', \Delta', R', Q'_d)$ could do the same transformation of these input trees. Then, roughly speaking, N would first have to make a copy of $m(a^n e)$ and would then have to relabel the two copies in an arbitrary but identical way, which is clearly impossible. A formal proof goes as follows. If N realizes the same transformation, then, for each $n \geq 1$ and each $w \in \{a,b\}^*$ of length n, there is a derivation $q_0[f[m(a^n e)]] \overset{*}{\underset{N}{\Rightarrow}} f[m(we)m(we)]$ for some q_0 in Q'_d.

Let us consider a fixed n. Consider, in each of these 2^n derivations, the first string of the form $f[t_1 t_2]$; that is, consider the moment that f is produced as output. Note that this is not necessarily the second string of the derivation, since the transducer may first erase the input symbol f and some of the a's before producing any output (thus the derivation may look like $q_0[f[m(a^n e)]] \overset{*}{\Rightarrow} q[a[m(a^k e)]] \Rightarrow f[t_1 t_2] \overset{*}{\Rightarrow} f[m(we)m(we)]$ for some $q \in Q'$ and some k, $0 \leq k < n$, or even like $q_0[f[m(a^n e)]] \overset{*}{\Rightarrow} q[e] \Rightarrow f[t_1 t_2] = f[m(we)m(we)]$ for some $q \in Q$). Obviously, for different derivations these strings have to be different: if, for $w \neq w'$, both $t_1 \overset{*}{\Rightarrow} m(we)$, $t_2 \overset{*}{\Rightarrow} m(we)$ and $t_1 \overset{*}{\Rightarrow} m(w'e)$, $t_2 \overset{*}{\Rightarrow} m(w'e)$, then also $f[t_1 t_2] \overset{*}{\Rightarrow} f[m(we)m(w'e)]$, which is an invalid output. Therefore there are 2^n of such strings $f[t_1 t_2]$. However it is clear that $f[t_1 t_2]$ is of the form $f[\bar{t}_1 \bar{t}_2]\langle x_1 \leftarrow m(a^k e)\rangle$, where $0 \leq k \leq n$ and $f[\bar{t}_1 \bar{t}_2]$ is the right hand side of a rule in R'. Therefore the number of possible $f[t_1 t_2]$'s is less than $(n+1)r$, where $r = \#(R')$. For n sufficiently large this is a contradiction.

(2) Consider now the top-down ftt M of Example 4.37. M is in PNT and is a typical example of an ftt having property (T). Suppose that M can be realized by a bottom-up ftt $N = (Q', \Sigma', \Delta', R', Q'_d)$. Consider again for each $n \geq 1$ the specific input tree $f[m(a^n e)]$. This tree should be transformed by N into all trees of the form $f[m(w_1 e)m(w_2 e)]$ for $w_1, w_2 \in \{a,b\}^*$ of length n. Let us consider, in each of the derivations realizing this transformation, the first string which contains the output tree. Note that this is not necessarily the last string since N may end its computation by erasing a number of a's and the input f. Obviously, this string is obtained from the previous one by application of a rule with right hand side of the form $q[f[\bar{t}_1 \bar{t}_2]]$, where $q \in Q'$, $\bar{t}_1, \bar{t}_2 \in T_\Delta(\{x_1\})$

and there are s_1 and s_2 such that $\bar{t}_1\langle x_1 \leftarrow s_1\rangle = m(w_1 e)$ and $\bar{t}_2\langle x_1 \leftarrow s_2\rangle = m(w_2 e)$. Obviously, if $f[\bar{t}_1\bar{t}_2]$ contains no x_1 or only one x_1, then the rule can only be used for exactly one input tree $f[m(a^n e)]$. Thus we may choose n such that in all derivations starting with $f[m(a^n e)]$ the right hand side $q[f[\bar{t}_1\bar{t}_2]]$ contains two x_1's (it cannot contain more). Thus $q[f[\bar{t}_1\bar{t}_2]]$ is of the form $q[f[m(v_1 x_1)m(v_2 x_1)]]$ for certain $v_1, v_2 \in \{a, b\}^*$. By choosing n larger than the length of all such v_1's and v_2's occurring in right hand sides of rules in R', we see that the output tree always has two equal subtrees $\neq e$: it has to be of the form $f[m(v_1 we)m(v_2 we)]$ for some $w \in \{a, b\}^+$. Thus, for such an n, not all possible outputs are produced. This is a contradiction. $\qquad\square$

An important property of a class F of tree transformations is whether it is closed under composition. If so, then we know that each sequence of transformations from F can be realized by one tree transducer (corresponding to the class F). We then also know that the class of F-surface tree languages is closed under the transformations of F. The next theorem shows that unfortunately the classes of top-down and bottom-up tree transformations are not closed under composition. This nonclosure is caused by the failure of property (B) for top-down transformations (property (T) for bottom-up transformations).

Theorem 4.44. T and B are not closed under composition. In particular, there are tree transformations in $(REL \circ HOM) - T$ and in $(HOM \circ REL) - B$.

Proof.
(1)   The bottom-up ftt M of Example 4.25 can be realized by the composition of a relabeling and a homomorphism. Let $\Omega_0 = \{e\}$ and $\Omega_1 = \{a, b, f\}$. Let r be the relabeling from Σ into Ω defined by $r_0(e) = \{e\}$, $r_1(a) = \{a, b\}$ and $r_1(f) = \{f\}$. Let h be the tree homomorphism from Ω into Δ defined by $h_0(e) = e$, $h_1(a) = a[x_1]$, $h_1(b) = b[x_1]$ and $h_1(f) = f[x_1 x_1]$. Then, for all $s \in T_\Sigma$ and $t \in T_\Delta$, $(s, t) \in M$ iff there exists u in T_Ω such that $u \in r(s)$ and $h(u) = t$.
    Thus, by the first part of the proof of Theorem 4.43, M is in $(REL \circ HOM) - T$.
    (2)   The top-down ftt M of Example 4.37 can be realized by the composition of a homomorphism and a relabeling. Let Π be the ranked alphabet with $\Pi_0 = \{e\}$, $\Pi_1 = \{a\}$ and $\Pi_2 = \{f\}$. Let h be the tree homomorphism from Σ into Π defined by $h_0(e) = e$, $h_1(a) = a[x_1]$ and $h_1(f) = f[x_1 x_1]$. Let r be the relabeling from Π into Δ defined by $r_0(e) = \{e\}$, $r_1(a) = \{a, b\}$ and $r_2(f) = \{f\}$. Then, for all $s \in T_\Sigma$ and $t \in T_\Delta$, $(s, t) \in M$ iff there exists u in T_Π such that $h(s) = u$ and $t \in r(u)$.
    Thus, by the second part of the proof of Theorem 4.43, M is in $(HOM \circ REL) - B$. $\square$

Exercise 4.45. Prove the statements in the above proof. $\square$

One might get the impression that each bottom-up (resp. top-down) tree transformation can be realized by two top-down (resp. bottom-up) tree transducers (i.e., $B \subseteq T \circ T$, resp. $T \subseteq B \circ B$). We shall show later that this is true.

Let us now consider the linear case. Since properties (B) and (T) are now eliminated, the only remaining difference between linear top-down and bottom-up tree transducers is caused by property (B').

Lemma 4.46. There is a tree transformation M that belongs to LDB, but not to T. M can be realized by the composition of a deterministic top-down fta with a linear homomorphism.

Proof. Let $\Sigma_0 = \{c\}$, $\Sigma_1 = \{b\}$, $\Sigma_2 = \{a\}$, $\Delta_0 = \{c\}$ and $\Delta_1 = \{a, b\}$. Consider the tree transformation $M = \{(a[tc], a[t]) \mid t = m(b^n c) \text{ for some } n \geq 0\}$. We shall show that $M \notin T$. The rest of the proof is left as an exercise. Suppose that there is a top-down ftt $N = (Q, \Sigma', \Delta', R, Q_d)$ such that $T(N) = M$. Each successful derivation of N has to start with the application of a rule $q_0[a[x_1 x_2]] \to s$, where $q_0 \in Q_d$ and $s \in T_{\Delta'}(X_2)$. Now, if s contains no x_1, then we could change the input $a[tc]$ into $a[t'c]$ without changing the output. If s contains no x_2, then we could change $a[tc]$ into $a[tb[c]]$ and still obtain (the same) output. But if s contains both x_1 and x_2 then it has to contain a symbol of rank 2 and so $a[t]$ cannot be derived. $\qquad\square$

Since both deterministic top-down fta and linear homomorphisms belong to LDT we can state the following corollary.

Corollary 4.47. Composition of linear deterministic top-down tree transformations leads out of the class of top-down tree transformations; in a formula: $(LDT \circ LDT) - T \neq \emptyset$. $\qquad\square$

We now show that, in some sense, property (B$'$) is the only cause of difference between linear bottom-up and linear top-down tree transformations. Firstly, all linear top-down tree transformations can be realized linear bottom-up. Secondly, in the nondeleting linear case, all differences between top-down and bottom-up are gone (this can be considered as a generalization of Theorem 3.17).

Theorem 4.48.

(1) $LT \subsetneq LB$,

(2) $NLT = NLB$.

Proof. We first show part (2). Let us say that a nondeleting linear bottom-up ftt $M = (Q, \Sigma, \Delta, R, Q_d)$ and a nondeleting linear top-down ftt $N = (Q', \Sigma', \Delta', R', Q'_d)$ are "associated" if $Q = Q'$, $\Sigma = \Sigma'$, $\Delta = \Delta'$, $Q_d = Q'_d$ and

(i) for each $a \in \Sigma_0$, $q \in Q$ and $t \in T_\Delta$,
$a \to q[t]$ is in R iff $q[a] \to t$ is in R';

(ii) for each $k \geq 1$, $a \in \Sigma_k$, $q_1, \ldots, q_k, q \in Q$ and $t \in T_\Delta(X_k)$ linear and nondeleting w.r.t. X_k,
$a[q_1[x_1] \cdots q_k[x_k]] \to q[t]$ is in R iff
$q[a[x_1 \cdots x_k]] \to t\langle x_1 \leftarrow q_1[x_1], \ldots, x_k \leftarrow q_k[x_k]\rangle$ is in R'.

Note that each tree $r \in T_\Delta(Q[X_k])$, which is linear and nondeleting w.r.t. X_k, is of the form $t\langle x_1 \leftarrow q_1[x_1], \ldots, x_k \leftarrow q_k[x_k]\rangle$, where $t \in T_\Delta(X_k)$ is linear and nondeleting w.r.t. X_k (in fact, t is the result of replacing $q_i[x_i]$ by x_i in r). Therefore it is clear that for each $M \in NLB$ there exists an associated $N \in NLT$ and vice versa. Hence it suffices to prove that associated ftt realize the same tree transformation. Let M and N be associated as above. We shall prove, by induction on s, that for every $q \in Q$, $s \in T_\Sigma$ and $u \in T_\Delta$,

$$s \overset{*}{\underset{M}{\Rightarrow}} q[u] \quad \text{iff} \quad q[s] \overset{*}{\underset{N}{\Rightarrow}} u. \tag{$*$}$$

For $s \in \Sigma_0$, $(*)$ is obvious. Suppose now that $s = a[s_1 \cdots s_k]$ for some $k \geq 1$, $a \in \Sigma_k$ and $s_1, \ldots, s_k \in T_\Sigma$. The only-if part of $(*)$ is left to the reader (it is similar to the proof of Theorem 4.28(3)). The if-part of $(*)$ is proved as follows (it is similar to the proof of Theorem 3.65). Let the first rule applied in the derivation $q[a[s_1 \cdots s_k]] \overset{*}{\underset{N}{\Rightarrow}} u$ be $q[a[x_1 \cdots x_k]] \to r$, and let $r = t\langle x_1 \leftarrow q_1[x_1], \ldots, x_k \leftarrow q_k[x_k]\rangle$ for certain $t \in T_\Delta(X_k)$ and $q_1, \ldots, q_k \in Q$. Thus $q[a[s_1 \cdots s_k]] \underset{N}{\Rightarrow} t\langle x_1 \leftarrow q_1[s_1], \ldots, x_k \leftarrow q_k[s_k]\rangle \overset{*}{\underset{N}{\Rightarrow}} u$. Since t is linear and nondeleting, there exist $u_1, \ldots, u_k \in T_\Delta$ such that $u = t\langle x_1 \leftarrow u_1, \ldots, x_k \leftarrow u_k\rangle$ and $q_i[s_i] \overset{*}{\underset{N}{\Rightarrow}} u_i$ for all i, $1 \leq i \leq k$. Hence, by induction, $s_i \overset{*}{\underset{M}{\Rightarrow}} q_i[u_i]$ for all i, $1 \leq i \leq k$. Also, by associatedness, the rule $a[q_1[x_1] \cdots q_k[x_k]] \to q[t]$ is in R. Consequently, $a[s_1 \cdots s_k] \overset{*}{\underset{M}{\Rightarrow}} a[q_1[u_1] \cdots q_k[u_k]] \underset{M}{\Rightarrow} q[t\langle x_1 \leftarrow u_1, \ldots, x_k \leftarrow u_k\rangle] = q[u]$.

We now show part (1). By Lemma 4.46, it suffices to show that $LT \subseteq LB$. In principle we can use the construction used above to show $NLT \subseteq NLB$. The only problem is that the top-down transducer N may delete subtrees, whereas a bottom-up transducer is forced to process a subtree before deleting it. The solution is to add an "identity state" d to the set of states of M which allows M to process any subtree which has to be deleted (d is such that for all $t \in T_\Sigma$, $t \overset{*}{\underset{M}{\Rightarrow}} d[t]$). The formal construction is as follows. Let $N = (Q, \Sigma, \Delta, R, Q_d)$ be a linear top-down ftt. Construct the linear bottom-up ftt $M = (Q \cup \{d\}, \Sigma, \Delta \cup \Sigma, R_M, Q_d)$, where R_M is obtained as follows.

(i) For each $a \in \Sigma_0$ the rule $a \to d[a]$ is in R_M, and for each $k \geq 1$ and $a \in \Sigma_k$ the rule $a[d[x_1] \cdots d[x_k]] \to d[a[x_1 \cdots x_k]]$ is in R_M.

(ii) For $q \in Q$, $a \in \Sigma_0$ and $t \in T_\Delta$, if $q[a] \to t$ is in R, then $a \to q[t]$ is in R_M.

(iii) Let $q[a[x_1 \cdots x_k]] \to t$ be in R, where $q \in Q$, $k \geq 1$, $a \in \Sigma_k$ and t is a linear tree in $T_\Delta(Q[X_k])$. Determine the (unique) states $q_1, \ldots, q_k \in Q \cup \{d\}$ such that, for $1 \leq i \leq k$, either $q_i[x_i]$ occurs in t or (x_i does not occur in t and) $q_i = d$. Determine $t' \in T_\Delta(X_k)$ such that $t'\langle x_1 \leftarrow q_1[x_1], \ldots, x_k \leftarrow q_k[x_k]\rangle = t$. Then the rule $a[q_1[x_1] \cdots q_k[x_k]] \to q[t']$ is in R_M.

Again $(*)$ can be proved, and since the proof only slightly differs from the previous one, it is left to the reader. $\qquad\square$

Exercise 4.49. Find an example of a tree transformation in $LD_tB - T$. $\qquad\square$

Exercise 4.50. Compare the classes PLT and PLB. $\qquad\square$

Exercise 4.51. Let a deterministic top-down ftt be called "simple" if it is not allowed to make different translations of the same input subtree (if $q[a[x_1 \cdots x_k]] \to t$ is a rule and $q_1[x_i]$, $q_2[x_i]$ occur in t, then $q_1 = q_2$). Prove that the class of simple deterministic top-down tree transformations is included in B. (This result should be expected from the fact that property (T) is eliminated. Similarly, one can prove that $NDB \subseteq T$, because properties (B) and (B$'$) are eliminated.)   □

4.4 Decomposition and composition of bottom-up tree transformations

Since bottom-up tree transformations are theoretically easier to handle than top-down tree transformations, we start investigating the former.

We have seen that a bottom-up ftt can copy after nondeterministic processing (property (B)). The next theorem shows that these two things can in fact be taken apart into different phases of the transformation: each bottom-up ftt can be decomposed into two transducers, the first doing the nondeterminism (linearly) and the second doing the copying (deterministically).

Theorem 4.52. Each bottom-up tree transformation can be realized by a finite state relabeling followed by a homomorphism. In formula:

$$B \subseteq QREL \circ HOM.$$

Moreover $LB \subseteq QREL \circ LHOM$ and

$$DB \subseteq DBQREL \circ HOM.$$

Proof. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up ftt. To simulate M in two phases we apply a technique similar to the one use in the proof of Theorem 3.58: a finite state relabeling is used to put information on each node indicating by which piece of the tree the node should be replaced; then a homomorphism is used to actually replace each node by that piece of tree. The formal construction is as follows.

We simultaneously construct a ranked alphabet Ω, the set of rules R_N of a bottom-up ftt $N = (Q, \Sigma, \Omega, R_N, Q_d)$ and a homomorphism $h : T_\Omega \to T_\Delta$ as follows.

(i) If $a \to q[t]$ is a rule in R, then d_t is a (new) symbol in Ω_0, $a \to q[d_t]$ is in R_N and $h_0(d_t) = t$.

(ii) If $a[q_1[x_1] \cdots q_k[x_k]] \to q[t]$ is a rule in R, then d_t is a (new) symbol in Ω_k, $a[q_1[x_1] \cdots q_k[x_k]] \to q[d_t[x_1 \cdots x_k]]$ is in R_N and $h_k(d_t) = t$.

The only requirement on the symbols of Ω is that if $t_1 \neq t_2$ then $d_{t_1} \neq d_{t_2}$.

Obviously N is a (bottom-up) finite state relabeling. Also, if M is linear then h is linear, and if M is deterministic then so is N.

It can easily be shown (by induction on s) that, for $s \in T_\Sigma$, $q \in Q$ and $t \in T_\Delta$,

$$s \xRightarrow[M]{*} q[t] \quad \text{iff} \quad \exists u \in T_\Omega : s \xRightarrow[N]{*} q[u] \text{ and } h(u) = t.$$

From this it follows that $M = N \circ h$, which proves the theorem.   □

Example 4.53. Consider the bottom-up ftt M of Example 4.18. It can be decomposed as follows. Firstly, $\Omega_0 = \{a, b\}$ and $\Omega_2 = \{m_1, m_2, n_1, n_2\}$, where $d_a = a$, $d_b = b$, $d_{m[x_1 x_1]} = m_1$, $d_{m[x_2 x_2]} = m_2$, $d_{n[x_1 x_1]} = n_1$ and $d_{n[x_2 x_2]} = n_2$. Secondly, $N = (Q, \Sigma, \Omega, R_N, Q_d)$, where R_N consists of the rules

$$a \to q_0[a], \qquad\qquad\qquad b \to q_0[b]$$
$$f[q_i[x_1]q_j[x_2]] \to q_{1-i}[m_1[x_1 x_2]], \qquad g[q_i[x_1]q_j[x_2]] \to q_{1-i}[n_1[x_1 x_2]],$$
$$f[q_i[x_1]q_j[x_2]] \to q_{1-j}[m_2[x_1 x_2]], \qquad g[q_i[x_1]q_j[x_2]] \to q_{1-j}[n_2[x_1 x_2]]$$

for all $i, j \in \{0, 1\}$. Finally, h is defined by $h_0(a) = a$, $h_0(b) = b$, $h_2(m_1) = m[x_1 x_1]$, $h_2(m_2) = m[x_2 x_2]$, $h_2(n_1) = n[x_1 x_1]$ and $h_2(n_2) = n[x_2 x_2]$. For example, $f[a[g[ab]]] \stackrel{*}{\underset{N}{\Rightarrow}} q_0[m_2[a[n_2[ab]]]]$ and $h(m_2[a[n_2[ab]]]) = m[n[bb]n[bb]]$. $\qquad\square$

Note that Theorem 4.52 means (among other things) that each bottom-up tree transformation can be realized by the composition of two top-down tree transformations (cf. Theorem 4.43).

We now show that, in the nondeterministic case, the finite state relabeling can still be decomposed further into a relabeling followed by a finite tree automaton restriction.

Theorem 4.54.

$$B \subseteq REL \circ FTA \circ HOM \quad \text{and}$$
$$LB \subseteq REL \circ FTA \circ LHOM.$$

Proof. By the previous theorem and the fact that HOM and $LHOM$ are closed under composition (see Exercise 4.9) it clearly suffices to show that $QREL \subseteq REL \circ FTA \circ LHOM$. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up finite state relabeling. We shall actually show that M can be simulated by a relabeling, followed by an fta, followed by a projection (which is in $LHOM$). The relabeling guesses which rule is applied by M at each node (and puts that rule as a label on the node), the bottom-up fta checks whether this guess is in accordance with the possible state transitions of M, and finally the projection labels the node with the right label.

Formally we construct a ranked alphabet Ω, a relabeling r from T_Σ into T_Ω, a (nondeterministic) bottom-up fta $N = (Q, \Omega, \delta, S, Q_d)$ and a projection p from T_Ω into T_Δ as follows.

(i) If rule m in R is of the form $a \to q[b]$, then d_m is a (new) symbol in Ω_0, $d_m \in r_0(a)$, $q \in S_{d_m}$ and $p_0(d_m) = b$.

(ii) If rule m in R is of the form $a[q_1[x_1] \cdots q_k[x_k]] \to q[b[x_1 \cdots x_k]]$, then d_m is a (new) symbol in Ω_k, $d_m \in r_k(a)$, $q \in \delta_{d_m}^k(q_1, \ldots, q_k)$ and $p_k(d_m) = b$.

We require that if m and n are different rules, then $d_m \neq d_n$.

It is left to the reader to show that, for $s \in T_\Sigma$, $q \in Q$ and $t \in T_\Delta$,

$$s \stackrel{*}{\underset{M}{\Rightarrow}} q[t] \quad \text{iff} \quad \exists u \in T_\Omega : u \in r(s), u \in L(N) \text{ and } t = p(u).$$

This proves the theorem. $\qquad\square$

These decomposition results are often very helpful when proving something about bottom-up tree transformations: the proof can often be split up into proofs about REL, FTA and HOM only. As an example, we immediately have the following result from Theorem 4.54 and Theorems 3.32, 3.48 and 3.65 (note that a class of tree languages is closed under fta restrictions if and only if it is closed under intersection with recognizable tree languages!).

Corollary 4.55. $RECOG$ is closed under linear bottom-up tree transformations. $\quad\square$

(This expresses that the image of a recognizable tree language under a linear bottom-up tree transformation is again recognizable. In other words, LB-Surface $= RECOG$.)

Exercise 4.56. Prove, using Theorem 4.54, that B-Surface $= HOM$-Surface. Prove that, in fact, each B-Surface tree language is the homomorphic image of a rule tree language. $\quad\square$

We now prove that under certain circumstances the composition of two elements in B is again in B. Recall from Section 4.3 that the non-closure of B under composition was caused by the failure of property (T) for B: in general, in B, we can't compose a copying transducer with a nondeterministic one. We now show that if either the first transducer is noncopying or the second one is deterministic, then their composition is again in B. Thus, when eliminating (the failure of) property (T), closure results are obtained.

Theorem 4.57.

> (1) $\quad LB \circ B \subseteq B \quad$ and $\quad LB \circ LB \subseteq LB$.
>
> (2) $\quad B \circ DB \subseteq B \quad$ and $\quad DB \circ DB \subseteq DB$.

Proof. Because of our decomposition of B we only need to look at special cases. These are treated in three lemmas, concerning composition with homomorphisms, fta restrictions and relabelings respectively. Detailed induction proofs of these lemmas are left to the reader.

Lemma. $B \circ HOM \subseteq B$, $\quad LB \circ LHOM \subseteq LB$ and $DB \circ HOM \subseteq DB$.

Proof. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up ftt and h a tree homomorphism from T_Δ into T_Ω. We have to show that $M \circ h$ can be realized by a bottom-up ftt N. The idea is the same as that of Theorem 3.65: N simulates M but outputs at each step the homomorphic image of the output of M. Note that, contrary to the proof of Theorem 3.65 (which was concerned with regular tree grammars, a top-down device), we need not require linearity of h.

The construction is as follows. Extend h to trees in $T_\Delta(X)$ by defining $h_0(x_i) = x_i$ for all x_i in X. Thus h is now a homomorphism from $T_\Delta(X)$ into $T_\Omega(X)$. Define $N = (Q, \Sigma, \Omega, R_N, Q_d)$ such that

(i) if $a \to q[t]$ is in R, then $a \to q[h(t)]$ is in R_N;

(ii) if $a[q_1[x_1] \cdots q_k[x_k]] \to q[t]$ is in R, then $a[q_1[x_1] \cdots q_k[x_k]] \to q[h(t)]$ is in R_N.

53

Obviously, if M and h are linear then so is N (the linear homomorphism transforms a linear tree in $T_\Delta(X)$ into a linear tree in $T_\Omega(X)$), and if M is deterministic then so is N. $\square$

Lemma. $B \circ FTA \subseteq B$, $LB \circ FTA \subseteq LB$ and $DB \circ FTA \subseteq DB$.

Proof. The idea of the proof is similar to the one used to solve Exercise 3.68. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a bottom-up ftt and let $\widetilde{N} = (Q_N, \Delta, \Delta, R_N, Q_{dN})$ be a deterministic bottom-up ftt corresponding to a deterministic bottom-up fta as in the proof of Theorem 4.28(2). We have to show that $M \circ \widetilde{N}$ can be realized by a bottom-up ftt K. K will have $Q \times Q_N$ as its set of states and it will simultaneously simulate M and keep track of the state of $\widetilde{N}$ at the computed output of M.

Extend $\widetilde{N}$ by expanding its alphabet to $\Delta \cup X$ (or, better, to $\Delta \cup X_m$ where m is the highest subscript of a variable occurring in the rules of M) and by allowing the variables in its rules to range over $T_\Delta(X)$. Thus the computation of the finite tree automaton $\widetilde{N}$ may now be started with an element of $T_\Delta(Q_N[X])$, which means that, at certain places in the tree, $\widetilde{N}$ has to start in prescribed start states.

Construct $K = (Q \times Q_N, \Sigma, \Delta, R_K, Q_d \times Q_{dN})$ such that

   (i) if $a \to q[t]$ is in R and if $t \underset{\widetilde{N}}{\overset{*}{\Longrightarrow}} q'[t]$,

       then $a \to (q, q')[t]$ is in R_K;

   (ii) if $a[q_1[x_1] \cdots q_k[x_k]] \to q[t]$ is in R, and if $t \langle x_1 \leftarrow q_1'[x_1], \ldots, x_k \leftarrow q_k'[x_k] \rangle \underset{\widetilde{N}}{\overset{*}{\Longrightarrow}} q'[t]$,

       then the rule $a[(q_1, q_1')[x_1] \cdots (q_k, q_k')[x_k]] \to (q, q')[t]$ is in R_K.

Note that if M is linear, then so is K. Moreover, since $\widetilde{N}$ is deterministic, if M is deterministic then so is K. $\square$
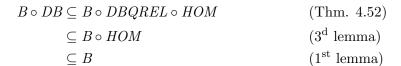
Lemma. $B \circ DBQREL \subseteq B$, $DB \circ DBQREL \subseteq DB$ and $LB \circ REL \subseteq LB$.

Proof. The proofs of the first two statements are easy generalizations of the proof of the previous lemma. The proof of the third statement is left as an easy exercise. $\square$

We now finish the proof of Theorem 4.57.
Firstly

$$
\begin{aligned}
LB \circ B &\subseteq LB \circ REL \circ FTA \circ HOM &&\text{(Thm. 4.54)} \\
&\subseteq LB \circ FTA \circ HOM &&\text{(3}^\text{d}\text{ lemma)} \\
&\subseteq LB \circ HOM &&\text{(2}^\text{d}\text{ lemma)} \\
&\subseteq B &&\text{(1}^\text{st}\text{ lemma)}
\end{aligned}
$$

and similarly for $LB \circ LB \subseteq LB$.

Secondly

$$
\begin{aligned}
B \circ DB &\subseteq B \circ DBQREL \circ HOM & \text{(Thm. 4.52)} \\
&\subseteq B \circ HOM & \text{(3^{d} lemma)} \\
&\subseteq B & \text{(1^{st} lemma)}
\end{aligned}
$$

and similarly for $DB \circ DB \subseteq DB$. $\qquad\square$

Note that the "right hand side" of Theorem 4.57 states that LB and DB are closed under composition.

Exercise 4.58. Is LDB closed under composition? $\qquad\square$

Exercise 4.59. Prove that B-Surface is closed under intersection with recognizable tree languages. $\qquad\square$

A consequence of Theorem 4.57 (or in fact its second lemma) is the following useful fact.

Corollary 4.60. $RECOG$ is closed under inverse bottom-up tree transformations (in particular under inverse tree homomorphisms, cf. Exercise 3.68).

Proof. Let $L \in RECOG$ and $M \in B$. We have to show that $M^{-1}(L)$ is in the class $RECOG$. Obviously, if R is the finite tree automaton restriction $\{(t, t) \mid t \in L\}$, then $M^{-1}(L) = \text{dom}(M \circ R)$. By Theorem 4.57, $M \circ R$ is in B and so, by Exercise 4.29, its domain is in $RECOG$. $\qquad\square$

Remark 4.61. Note that, since, for arbitrary tree transformations M_1 and M_2, $(M_1 \circ M_2)^{-1} = M_2^{-1} \circ M_1^{-1}$, Corollary 4.60 implies that if M is the composition of any finite number of bottom-up tree transformations (in particular, elements of REL, FTA and HOM), then $RECOG$ is closed under M^{-1}; moreover, since $\text{dom}(M) = M^{-1}(T_\Delta)$, the domain of any such tree transformation M is recognizable. $\qquad\square$

We finally note that, because of the composition results of Theorem 4.57, the inclusion signs in Theorems 4.52 and 4.54 may be replaced by equality signs. It is also easy to prove equations like $B = LB \circ HOM$, $B = LB \circ DB$ or $B = REL \circ DB$ (cf. property (B)). The first equation has some importance of its own, since it characterizes B without referring to the notion "bottom-up"; this is so because LB has such a characterization as shown next (we first need a definition).

Definition 4.62. Let F be a class of tree transformations containing all identity transformations. For each $n \geq 1$ we define F^n inductively by $F^1 = F$ and $F^{n+1} = F^n \circ F$. Moreover, the <u>closure of F under composition</u>, denoted by F^*, is defined to be $\bigcup_{n \geq 1} F^n$.

Thus F^* consists of all tree transformations of the form $M_1 \circ M_2 \circ \cdots \circ M_n$, where $n \geq 1$ and $M_i \in F$ for all i, $1 \leq i \leq n$. $\qquad\square$

Corollary 4.63.

(1) $LB = (REL \cup FTA \cup LHOM)^*$.

(2) $B = LB \circ HOM$.

Proof. The inclusions $\subseteq$ follow from Theorems 4.52 and 4.54; the inclusions $\supseteq$ follow from Theorem 4.57. $\qquad\Box$

4.5 Decomposition of top-down tree transformations

We now show that, analogously to the bottom-up case, we can decompose each top-down ftt into two transducers, the first doing the copying and the second doing the rest of the work (cf. property (T)).

Theorem 4.64. $T \subseteq HOM \circ LT$ and $DT \subseteq HOM \circ LDT$.

Proof. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a top-down ftt. While processing an input tree, M generally makes a lot of copies of input subtrees in order to get different translations of these subtrees. To simulate M we can first use a homomorphism which simply makes as many copies of subtrees as are needed by M, and then we can simulate M linearly (since all copies are already there). The formal construction is as follows.

We first determine the "degree of copying" of M, that is the maximal number of copies needed by M in any step of its computation. Thus, for $x \in X$ and $r \in R$, let r_x be the number of occurrences of x in the right hand side of r. Let $n = \max\{r_x \mid x \in X, r \in R\}$. We now let Ω be the ranked alphabet obtained from Σ by multiplying the ranks of all symbols by n (so that a node may be connected to n copies of each of its subtrees). Thus $\Omega_{kn} = \Sigma_k$ for all $k \geq 0$. The "copying" homomorphism h from T_Σ into T_Ω is now defined by

(i) for $a \in \Sigma_0$, $h_0(a) = a$

(ii) for $k \geq 1$ and $a \in \Sigma_k$, $h_k(a) = a[x_1^n x_2^n \cdots x_k^n]$.

(For example, if $k = 2$ and $n = 3$, then $h_2(a) = a[x_1 x_1 x_1 x_2 x_2 x_2]$.)
Finally the top-down ftt $N = (Q, \Omega, \Delta, R_N, Q_d)$ is defined as follows.

(i) If $q[a] \to t$ is a rule in R, then it is also in R_N.

(ii) Suppose that $q[a[x_1 \cdots x_k]] \to t$ is a rule in R. Let us denote the variables $x_1, x_2, \ldots, x_{kn}$ by $x_{1,1}, x_{1,2}, \ldots, x_{1,n}, x_{2,1}, \ldots, x_{2,n}, \ldots, x_{k,1}, \ldots, x_{k,n}$ respectively. Then the rule $q[a[x_{1,1} \cdots x_{1,n} \cdots x_{k,1} \cdots x_{k,n}]] \to t'$ is in R_N, where t' is taken such that it is linear and such that $t'\langle x_{i,j} \leftarrow x_i\rangle_{\substack{1 \leq i \leq k \\ 1 \leq j \leq n}} = t$

(t' can be obtained by putting different second subscripts on different occurrences of the same variable in t. For instance, if $q[a[x_1 x_2]] \to b[q_1[x_2]cd[q_2[x_1]q_3[x_2]]]$ is in R and $n = 3$, then we can put the rule $q[a[x_{1,1}x_{1,2}x_{1,3}x_{2,1}x_{2,2}x_{2,3}]] \to b[q_1[x_{2,1}]cd[q_2[x_{1,1}]q_3[x_{2,2}]]]$ in R_N.)

Obviously, if M is deterministic, then so is N. A formal proof of the fact that $M = h \circ N$ is left to the reader. $\qquad\Box$

Example 4.65. Consider the top-down ftt M of Example 4.33 and let us consider its decomposition according to the above proof. Clearly $n = 2$ and therefore the definition of h for, for example, $+$, $*$, $-$, a and b is $h_2(+) = +[x_1 x_1 x_2 x_2]$, $h_2(*) = *[x_1 x_1 x_2 x_2]$, $h_1(-) = -[x_1 x_1]$, $h_0(a) = a$ and $h_0(b) = b$. Thus, for example, $h(*[+[ab] - [a]]) = *[+[aabb] + [aabb] - [aa] - [aa]]$. For instance the first three rules of M turn into the following three rules for N:

$$q[+[x_{1,1} x_{1,2} x_{2,1} x_{2,2}]] \rightarrow +[q[x_{1,1}] q[x_{2,1}]],$$
$$q[*[x_{1,1} x_{1,2} x_{2,1} x_{2,2}]] \rightarrow +[*[q[x_{1,1}] i[x_{2,1}]] * [i[x_{1,2}] q[x_{2,2}]]],$$
$$q[-[x_{1,1} x_{1,2}]] \rightarrow -[q[x_{1,1}]].$$

It is left to the reader to see how N processes $h(*[+[ab] - [a]])$. $\qquad\square$

Since $LT \subseteq LB$ (Theorem 4.48), we know already how to decompose LT. This gives us the following result.

Corollary 4.66. $T \subseteq HOM \circ LB = HOM \circ REL \circ FTA \circ LHOM.$ $\qquad\square$

Notice that the inclusion is proper by Lemma 4.46. From this corollary we see that each top-down tree transformation can be realized by the composition of two bottom-up tree transformations (cf. Theorem 4.43). Another way of expressing our decomposition results concerning B and T is by the equation $B^* = T^* = (REL \cup FTA \cup HOM)^*$.

By the above corollary and Remark 4.61 we obtain that $RECOG$ is closed under inverse top-down tree transformations, and in particular

Corollary 4.67. The domain of a top-down tree transformation is recognizable. $\qquad\square$

The next theorem says, analogously to Theorem 4.52, that each deterministic element of LT can be decomposed into two simpler (deterministic) ones.

Theorem 4.68. $LDT \subseteq DTQREL \circ LHOM.$

Proof. See the proof of Theorem 4.52. $\qquad\square$

We now note that we cannot obtain very nice results about closure under composition analogously to those of Theorem 4.57, since for instance LT and DT are not closed under composition (see Corollary 4.47). The reason is essentially the failure of property (B′) for top-down ftt. One could get closure results by eliminating both properties (B) and (B′). For example, one can show that $D_t T \circ T \subseteq T$, $T \circ NLT \subseteq T$, etc. However, after having compared DB with DT in the next section, we prefer to extend the top-down ftt in such a way that it has the capability of checking before deletion. It will turn out that the so extended top-down transducer has all the nice properties "dual" to those of B.

The following is an easy exercise in composition.

Exercise 4.69. Prove that every T-surface tree language is in fact the image of a rule tree language under a top-down tree transformation. $\qquad\square$

4.6 Comparison of B and T, the deterministic case

Although, by determinism, some differences between B and T are eliminated, DB and DT are still incomparable for a number of reasons. We first discuss the question why DB contains elements not in DT, and then the reverse question.

Firstly, we have seen in Lemma 4.46 that DB contains elements not in T. This was caused by property (B′).

Secondly, we have seen in Theorem 3.14 (together with Theorem 3.8) that there are deterministic bottom-up recognizable tree languages which cannot be recognized deterministically top-down. It is easy to see that the corresponding fta restrictions cannot be realized by a deterministic top-down transducer. In fact the following can be shown.

Exercise 4.70. Prove that the domain of a deterministic top-down ftt can be recognized by a deterministic top-down fta. □

Thirdly, there is a trivial reason that DB is stronger than DT: a bottom-up ftt can, for example, recognize the "lowest" occurrence of some symbol in a tree (since it is the first occurrence), whereas a deterministic top-down ftt cannot (since for him it is the last occurrence).

Lemma 4.71. There is a tree transformation in $DBQREL$ which is not in DT.

Proof. Let $\Sigma_0 = \{b\}$, $\Sigma_1 = \{a, f\}$, $\Delta_0 = \{b\}$ and $\Delta_1 = \{a, \bar{a}, f\}$. Consider the bottom-up ftt $M = (Q, \Sigma, \Delta, R, Q_d)$ where $Q = Q_d = \{q_1, q_2\}$ and R consists of the rules

$$b \to q_1[b],$$
$$a[q_1[x]] \to q_1[\bar{a}[x]], \qquad\qquad f[q_1[x]] \to q_2[f[x]],$$
$$a[q_2[x]] \to q_2[a[x]], \qquad\qquad f[q_2[x]] \to q_2[f[x]],$$

where x denotes x_1.

Thus M is a deterministic bottom-up finite state relabeling which bars all a's below the lowest f. Obviously a deterministic top-down ftt cannot give the right translation for both $m(a^n b)$ and $m(a^n f b)$. □

Let us now consider DT.

Firstly, we note that property (T) has not been eliminated: a deterministic top-down ftt still has the ability to copy an input subtree and continue the translation of these copies in different states. Consider for example the tree transformation $M = \{(m(ab^n c), a[m(p^n c)m(q^n c)]) \mid n \geq 0\}$. Obviously M is in DT. It can be shown, similarly to the proof of Theorem 4.43(2), that M is not in B (see also Exercise 4.51).

Secondly, deterministic bottom-up ftt cannot distinguish between left and right (because they start at the bottom!), whereas deterministic top-down ftt can. Consider for example the tree transformation $M = \{(a[m(b^n c)m(b^k c)], a[m(b^n d)m(b^k e)]) \mid n, k \geq 0\}$. This is obviously an element of DT (even $DTQREL$) and can be shown not to be in DB (of course it is in B: a nondeterministic bottom-up ftt can guess whether it is left or right and check its guess when arriving at the top). Thus we have the following lemma.

Lemma 4.72. There is a tree transformation in $DTQREL$ which is not in DB. $\qquad\square$

Thirdly, there is a proof of this lemma which is analogous to the one of Lemma 4.71: there is a deterministic top-down finite state relabeling that bars all a's above the highest f, and this cannot be done by an element of DB.

4.7 Top-down finite tree transducers with regular look-ahead

One way to take away the advantages of DB over DT is to allow the top-down tree transducer to have a look-ahead: that is, the ability to inspect an input subtree and, depending on the result of that inspection, decide which rule to apply next. Moreover it seems to be sufficient (and natural) that this look-ahead ability should consist of inspecting whether the input subtree belongs to a certain recognizable tree language or not (in other words, checking whether it has a certain "recognizable property"). As a result of this capability the top-down tree transducer would first of all have property (B'): it can check a recognizable property of a subtree and decide whether to delete it or not. Secondly, the domain of a deterministic top-down tree transducer would be arbitrary recognizable (it just starts by checking whether the whole input tree belongs to the recognizable tree language). And, thirdly, a deterministic top-down tree transducer would for instance be able to see the "lowest" occurrence of some symbol in a tree (it just checks whether the subtree beneath the symbol contains another occurrence of the same symbol, and that is a recognizable property).

We now formally define the top-down transducer with regular (= recognizable) look-ahead. It turns out that the look-ahead feature can be expressed easily in a tree rewriting system (see Definition 4.12): for each rule we specify the ranges of the variables in the rule to be certain recognizable tree languages (such a rule is then applicable only if the corresponding input subtrees belong to these recognizable tree languages).

Definition 4.73. A <u>top-down (finite) tree transducer with regular look-ahead</u> is a structure $M = (Q, \Sigma, \Delta, R, Q_d)$, where Q, Σ, Δ and Q_d are as for the ordinary top-down ftt and R is a finite set of rules of the form $(t_1 \to t_2, D)$, where $t_1 \to t_2$ is an ordinary top-down ftt rule and D is a mapping from X_k into $\mathcal{P}(T_\Sigma)$ (where k is the number of variables in t_1) such that, for $1 \le i \le k$, $D(x_i) \in RECOG$. (Whenever D is understood or will be specified later we write $t_1 \to t_2$ rather than $(t_1 \to t_2, D)$. We call t_1 and t_2 the left hand side and right hand side of the rule respectively).

M is viewed as a tree rewriting system in the obvious way, $(t_1 \to t_2, D)$ being a "rule scheme" (t_1, t_2, D).

The <u>tree transformation realized by M</u>, denoted by $T(M)$ or M, is $\{(s, t) \in T_\Sigma \times T_\Delta \mid q[s] \underset{M}{\overset{*}{\Longrightarrow}} t \text{ for some } q \in Q_d\}$. $\qquad\square$

Thus a top-down ftt with regular look-ahead works in exactly the same way as an ordinary one, except that the application of each of its rules is restricted: the (input sub-)trees substituted in the rule should belong to prespecified recognizable tree languages. Note that for rules of the form $q[a] \to t$ the mapping D need not be specified.

Notation 4.74. The phrase "with regular look-ahead" will be indicated by a prime. Thus the class of top-down tree transformations with regular look-ahead will be denoted by T'. An element of T' is also called a top-down$'$ tree transformation. $\qquad\square$

Example 4.75. Consider the tree transformation M in the proof of Lemma 4.46. It can be realized by the top-down$'$ ftt $N = (Q, \Sigma, \Delta, R, Q_d)$ where $Q = \{q_0, q\}$, $Q_d = \{q_0\}$ and R consists of the following rules:

$$q_0[a[x_1 x_2]] \to a[q[x_1]] \text{ with ranges } D(x_1) = \{m(b^n c) \mid n \geq 0\} \text{ and } D(x_2) = \{c\},$$
$$q[b[x_1]] \to b[q[x_1]] \text{ with } D(x_1) = T_\Sigma, \text{ and}$$
$$q[c] \to c .$$

Note that, in the first rule, $D(x_1)$ could as well be T_Σ since it is checked later by N that the left subtree contains no a's. The essential use of regular look-ahead in this example is the restriction of the right subtree to $\{c\}$. $\qquad\square$

We now define some subclasses of T'.

Definition 4.76. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a top-down$'$ ftt.

The definitions of <u>linear</u>, <u>nondeleting</u> and <u>one-state</u> are identical to the bottom-up and top-down ones (see Definition 4.23).

M is called (partial) <u>deterministic</u> if the following holds.

(i) Q_d is a singleton.

(ii) If $(s \to t_1, D_1)$ and $(s \to t_2, D_2)$ are different rules in R (with the same left hand side), then $D_1(x_i) \cap D_2(x_i) = \emptyset$ for some i, $1 \leq i \leq k$ (where k is the number of variables in s). $\qquad\square$

Since the ranges of the variables are recognizable, it can effectively be determined whether a top-down$'$ ftt is deterministic (if, of course, these ranges are effectively specified, which we always assume).

Notation 4.24 also applies to T'. Thus LDT' is the class of linear deterministic top-down tree transformations with regular look-ahead.

Observe that, obviously, $T \subseteq T'$ since each top-down ftt can be transformed trivially into a top-down$'$ ftt by specifying all ranges of all variables in all rules to be the (recognizable) tree language T_Σ (where Σ is the input alphabet). Moreover, if Z is a modifier, then $ZT \subseteq ZT'$.

It can easily be seen that Theorems 4.43 and 4.44 still hold with T replaced by T'. In fact, the proofs of the theorems are true without further change.

In the next theorem we show that the regular look-ahead can be "taken out" of a top-down$'$ ftt.

Theorem 4.77. $T' \subseteq DBQREL \circ T$ and $ZT' \subseteq DBQREL \circ ZT$ for $Z \in \{L, D, LD\}$.

Proof. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be in T'. Consider all recognizable properties which M needs for its look-ahead (that is, all recognizable tree languages $D(x_i)$ occurring in the rules of M). We can use a total deterministic bottom-up finite state relabeling to check, for a given input tree t, whether the subtrees of t have these properties or not, and to put at each node a (finite) amount of information telling us whether the direct subtrees of that node have the properties or not. After this relabeling we can use an ordinary top-down ftt to simulate M, because the look-ahead information is now contained in the label of each node.

The formal construction might look as follows. Let $L_1, \ldots, L_n$ be all the recognizable tree languages occurring as ranges of variables in the rules of M. Let U denote the set $\{0, 1\}^n$, that is, the set of all sequences of 0's and 1's of length n. The jth element of $u \in U$ will be denoted by u^j. An element u of U will be used to indicate whether a tree belongs to $L_1, \ldots, L_n$ or not ($u^j = 1$ iff the tree is in L_j).

We now introduce a new ranked alphabet Ω such that $\Omega_0 = \Sigma_0$ and, for $k \geq 1$, $\Omega_k = \Sigma_k \times U^k$. Thus an element of Ω_k is of the form $(a, (u_1, \ldots, u_k))$ with $a \in \Sigma_k$ and $u_1, \ldots, u_k \in U$. If a node is labeled by such a symbol, it will mean that u_i contains all the information about the ith subtree of the node.

Next we define the mapping $f : T_\Sigma \to T_\Omega$ as follows:

(i) for $a \in \Sigma_0$, $f(a) = a$;

(ii) for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
$f(a[t_1 \cdots t_k]) = b[f(t_1) \cdots f(t_k)]$, where $b = (a, (u_1, \ldots, u_k))$ and, for $1 \leq i \leq k$ and $1 \leq j \leq n$, $u_i^j = 1$ iff $t_i \in L_j$.

It is left as an <u>exercise</u> to show that f can be realized by a (total) deterministic bottom-up finite state relabeling (given the deterministic bottom-up fta's recognizing $L_1, \ldots, L_n$).

We now define a top-down ftt $N = (Q, \Omega, \Delta, R_N, Q_d)$ such that

(i) if $q[a] \to t$ is in R, then it is in R_N;

(ii) if $(q[a[x_1 \cdots x_k]] \to t, D)$ is in R, then each rule of the form $q[(a, \overline{u})[x_1 \cdots x_k]] \to t$ is in R_N, where $\overline{u} = (u_1, \ldots, u_k) \in U^k$ and $\overline{u}$ satisfies the condition: if $D(x_i) = L_j$ then $u_i^j = 1$ (for all i and j, $1 \leq i \leq k$, $1 \leq j \leq n$).

This completes the construction. It is obvious from this construction that $M = f \circ N$. Moreover, if M is linear, then so is N. It is also clear that, in the construction above, the set U may be replaced by the set $\{u \in U \mid$ for all j_1 and j_2, if $L_{j_1} \cap L_{j_2} = \emptyset$, then $u^{j_1} \cdot u^{j_2} \neq 1\}$ (note that this influences R_N: rules containing elements not in this set are removed). One can now easily see that if M is deterministic, then so is N. $\qquad\square$

An immediate consequence of this theorem and previous decomposition results is that each element of T' is decomposable into elements of *REL*, *FTA* and *HOM*.

Corollary 4.78. The domain of a top-down ftt with regular look-ahead is recognizable.

Proof. For instance by Remark 4.61. $\qquad\square$

Another consequence of Theorem 4.77 is that the addition of regular look-ahead has no influence on the surface tree languages.

Corollary 4.79. T'-Surface $= T$-Surface and DT'-Surface $= DT$-Surface.

Proof. Let L be a $(D)T'$-surface tree language, so $L = M(L_1)$ for some $M \in (D)T'$ and $L_1 \in RECOG$. Now, by Theorem 4.77, $M = R \circ N$ for some $R \in DBQREL$ and $N \in (D)T$. Hence $L = N(R(L_1))$. Since $RECOG$ is closed under linear bottom-up tree transformations (Corollary 4.55), $N(R(L_1))$ is a $(D)T$-surface tree language. $\square$

We now show that, in the linear case, there is no difference between bottom-up and top-down' tree transformations (all properties (B), (T) and (B') are "eliminated"), cf. Theorem 4.48.

Theorem 4.80. $LT' = LB$.

Proof. First of all we have

$$\begin{aligned} LT' &\subseteq DBQREL \circ LT & \text{(Theorem 4.77)} \\ &\subseteq DBQREL \circ LB & \text{(Theorem 4.48)} \\ &\subseteq LB & \text{(Theorem 4.57)}. \end{aligned}$$

Let us now prove that $LB \subseteq LT'$. The construction is the same as in the proof of Theorem 4.48(2), but now we use look-ahead in case the bottom-up ftt is deleting. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a linear bottom-up ftt. Define, for each q in Q, M_q to be the bottom-up ftt $(Q, \Sigma, \Delta, R, \{q\})$. Construct the linear top-down' ftt $N = (Q, \Sigma, \Delta, R_N, Q_d)$ such that

(i) if $a \to q[t]$ is in R, then $q[a] \to t$ is in R_N;

(ii) if $a[q_1[x_1] \cdots q_k[x_k]] \to q[t]$ is in R, then the rule
$q[a[x_1 \cdots x_k]] \to t\langle x_1 \leftarrow q_1[x_1], \ldots, x_k \leftarrow q_k[x_k]\rangle$ is in R_N, where, for $1 \leq i \leq k$,
if x_i does not occur in t, then $D(x_i) = \mathrm{dom}(M_{q_i})$, and $D(x_i) = T_\Sigma$ otherwise.

Note that $\mathrm{dom}(M_{q_i})$ is recognizable by Exercise 4.29.
It should be clear that $T(N) = T(M)$. $\square$

From the proof of Theorem 4.80 it follows that each element of LT' can be realized by a linear top-down' ftt with the property that look-ahead is only used on subtrees which are deleted. This property corresponds precisely to property (B').

Let us now consider composition of top-down' ftt. Analogously to the bottom-up case, we can now expect the results in the next theorem from property (B).

Theorem 4.81.

(1) $T' \circ LT' \subseteq T'$.

(2) $DT' \circ T' \subseteq T'$ and $DT' \circ DT' \subseteq DT'$.

Proof. As in the bottom-up case, we only consider a number of special cases.

Lemma. $T' \circ LHOM \subseteq T'$ and $DT' \circ HOM \subseteq DT'$.

Proof. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be a top-down$'$ ftt and h a tree homomorphism from T_Δ into T_Ω. We construct a new top-down$'$ ftt using the old idea of applying the homomorphism to the right hand sides of the rules of M. Therefore we extend h to trees in $T_\Delta(Q[X])$ by defining $h_0(x) = x$ for x in X and $h_1(q) = q[x_1]$ for $q \in Q$. Let, for $q \in Q$, $M_q = (Q, \Sigma, \Delta, R, \{q\})$. Note that, by Corollary 4.78, $\text{dom}(M_q)$ is recognizable.

Construct now $N = (Q, \Sigma, \Omega, R_N, Q_d)$ such that

(i) if $q_0[a] \to t$ is in R, then $q_0[a] \to h(t)$ is in R_N;

(ii) if $(q_0[a[x_1 \cdots x_k]] \to t, D)$ is in R, then $(q_0[a[x_1 \cdots x_k]] \to h(t), \overline{D})$ is in R_N, where, for $1 \leq i \leq k$, $\overline{D}(x_i)$ is the intersection of $D(x_i)$ and all tree languages $\text{dom}(M_q)$ such that $q[x_i]$ occurs in t but not in $h(t)$.

Thus N simultaneously simulates M and applies h to the output of M. But, whenever M starts making a translation of a subtree t starting in a state q, and this translation is deleted by h, N checks that t is translatable by M_q.

If h is linear or M is deterministic, then $N = M \circ h$. Obviously, if M is deterministic, then so is N. $\qquad\square$

Lemma.

$$T' \circ QREL \subseteq T',$$
$$DT' \circ DTQREL \subseteq DT' \quad \text{and}$$
$$DT' \circ DBQREL \subseteq DT'.$$

Proof. It is not difficult to see that, given $M \in T'$ and a top-down finite state relabeling N, the composition of these two can be realized by one top-down$'$ ftt K, which at each step simultaneously simulates M and N by transforming the output of M according to N. The construction is basically the same as that in the second lemma of Theorem 4.57. It is also clear that if M and N are both deterministic, then so is K. This gives us the first two statements of the lemma. The third one is more difficult since the finite state relabeling is now bottom-up. However the same kind of construction can be applied again, using the look-ahead facility to make the resulting top-down$'$ ftt deterministic.

Let $M = (Q, \Sigma, \Delta, R, Q_d)$ with $Q_d = \{q_d\}$ be in DT' and let $N = (Q_N, \Delta, \Omega, R_N, Q_{dN})$ be in $DBQREL$. Without loss of generality we assume that q_d does not occur in the right hand sides of the rules in R. Let, as usual, for $q \in Q$, $M_q = (Q, \Sigma, \Delta, R, \{q\})$, and, for $p \in Q_N$, $N_p = (Q_N, \Delta, \Omega, R_N, \{p\})$. We shall realize $M \circ N$ by a deterministic top-down$'$ ftt $K = (Q, \Sigma, \Omega, R_K, Q_d)$, where the set R_K of rules is determined as follows.

Let $q \in Q_M$ and $p \in Q_N$, such that if $q = q_d$ then $p \in Q_{dN}$.

(i) If $q[a] \to t$ is in R and $t \overset{*}{\underset{N}{\Rightarrow}} p[t']$, then $q[a] \to t'$ is in R_K.

(ii) Let $(q[a[x_1 \cdots x_k]] \rightarrow t, D)$ be a rule in R. Then t can be written as $t = s\langle x_1 \leftarrow q_1[x_{i_1}], \ldots, x_m \leftarrow q_m[x_{i_m}]\rangle$ for certain $m \geq 0$, $s \in T_\Delta(X_m)$, $q_1, \ldots, q_m \in Q$ and $x_{i_1}, \ldots, x_{i_m} \in X_k$, such that s is nondeleting w.r.t. X_m.

Let $p_1, \ldots, p_m$ be a sequence of m states of N such that $s\langle x_1 \leftarrow p_1[x_1], \ldots, x_m \leftarrow p_m[x_m]\rangle \underset{N}{\overset{*}{\Rightarrow}} p[s']$, where $s' \in T_\Omega(X_m)$. (Of course N was first extended in the usual way).

Then the rule $q[a[x_1 \cdots x_k]] \rightarrow s'\langle x_1 \leftarrow q_1[x_{i_1}], \ldots, x_m \leftarrow q_m[x_{i_m}]\rangle$ is in R_K, where the ranges of the variables are specified by $\overline{D}$ as follows. For $1 \leq u \leq k$, $\overline{D}(x_u)$ is the intersection of $D(x_u)$ and all tree languages $\mathrm{dom}(M_{q_j} \circ N_{p_j})$ such that $x_{i_j} = x_u$.

This ends the construction. Intuitively, when K arrives at the root of an input subtree $a[t_1 \cdots t_k]$ (in the same state as M), it first uses its regular look-ahead to determine the rule applied by M and to determine, for every $1 \leq j \leq m$, the (unique) state p_j in which N will arrive after translation of the M_{q_j}-translation of t_{i_j}. It then runs N on the piece of output of M, starting in the states $p_1, \ldots, p_m$, and produces the output of N as its piece of output. It is straightforward to check formally that K is a deterministic top-down$'$ ftt (using the determinism of both M and N). $\qquad\square$

We now complete the proof of Theorem 4.81.
Firstly

$$
\begin{aligned}
T' \circ LT' = T' \circ LB & \qquad \text{(Theorem 4.80)} \\
\subseteq T' \circ QREL \circ LHOM & \qquad \text{(Theorem 4.52)} \\
\subseteq T' \circ LHOM & \qquad \text{(second lemma)} \\
\subseteq T' & \qquad \text{(first lemma)}.
\end{aligned}
$$

Secondly

$$
\begin{aligned}
DT' \circ (D)T' \subseteq DT' \circ DBQREL \circ (D)T & \qquad \text{(Theorem 4.77)} \\
\subseteq DT' \circ (D)T & \qquad \text{(second lemma)} \\
\subseteq DT' \circ HOM \circ L(D)T & \qquad \text{(Theorem 4.64)} \\
\subseteq DT' \circ L(D)T & \qquad \text{(first lemma)}.
\end{aligned}
$$

Now $DT' \circ LT \subseteq T'$ (by (1) of this theorem) and

$$
\begin{aligned}
DT' \circ LDT \subseteq DT' \circ DTQREL \circ LHOM & \qquad \text{(Theorem 4.68)} \\
\subseteq DT' \circ LHOM & \qquad \text{(second lemma)} \\
\subseteq DT' & \qquad \text{(first lemma)}.
\end{aligned}
$$

This proves Theorem 4.81. $\qquad\square$

Note that the "right hand side" of Theorem 4.81(2) states that DT' is closed under composition. Clearly we know already that LT' is closed under composition (since

$LT' = LB$). It can also easily be checked from the proof of Theorem 4.81 that LDT' is closed under composition.

We can now show that indeed regular look-ahead has made DT stronger than DB.

Corollary 4.82. $DB \subsetneq DT'$.

Proof. By Theorem 4.52, $DB \subseteq DBQREL \circ HOM$. Hence, since the identity tree transformation is in DT', we trivially have $DB \subseteq DT' \circ DBQREL \circ HOM$. But, by the second and the first lemma in the proof of Theorem 4.81, $DT' \circ DBQREL \circ HOM \subseteq DT'$. Hence $DB \subseteq DT'$. Proper inclusion follows from Lemma 4.72. $\qquad\square$

Exercise 4.83. Show that each T'-surface tree language is the range of some element of T'. $\qquad\square$

Exercise 4.84. Prove that T-Surface is closed under linear tree transformations (recall Corollary 4.79).

Prove that DT-Surface is closed under deterministic top-down and bottom-up tree transformations. $\qquad\square$

It follows from Theorem 4.81 that the inclusion signs in Theorem 4.77 may be replaced by equality signs. Hence, for example, $DT' = DBQREL \circ DT = DB \circ DT$. We finally show a result "dual" to Corollary 4.63(2) (recall that $LT' = LB$).

Theorem 4.85. $T' = HOM \circ LT'$.

Proof. The inclusion $HOM \circ LT' \subseteq T'$ is immediate from Theorem 4.81. The inclusion $T' \subseteq HOM \circ LT'$ can be shown in exactly the same way as in the proof of $T \subseteq HOM \circ LT$ (Theorem 4.64). The only problem is the regular look-ahead: the image of a recognizable tree language under the homomorphism h need not be recognizable (we use the notation of the proof of Theorem 4.64). The solution is to consider a homomorphism g from T_Ω into T_Σ such that, for all t in T_Σ, $g(h(t)) = t$; g is easy to find. Now, whenever, in a rule of M, we have a recognizable tree language L as look-ahead (for some variable), we can use $g^{-1}(L)$ as the look-ahead in the corresponding rule of N. The details are left to the reader. $\qquad\square$

4.8 Surface and target languages

In this section we shall consider a few properties of the tree (and string) languages which are obtained from the recognizable tree languages by application of a finite number of tree transducers. In other words we shall consider the classes

$(REL \cup FTA \cup HOM)^*$-Surface, briefly denoted by Surface, and

$(REL \cup FTA \cup HOM)^*$-Target, briefly denoted by Target.

Note that Target = yield(Surface). Note also that, by various composition results, $(REL \cup FTA \cup HOM)^* = T^* = (T')^* = B^* = (T' \cup B)^* = \dots$ etc.

Let us first consider some classes of tree languages obtained by restricting the number of transducers applied. In particular, let us consider, for each $k \geq 1$, the

classes T^k-Surface, $(T')^k$-Surface and B^k-Surface. Obviously, by the above remark, Surface $= \bigcup_{k \geq 1} T^k$-Surface $= \bigcup_{k \geq 1} (T')^k$-Surface $= \bigcup_{k \geq 1} B^k$-Surface.

As a corollary to previous results we can show that regular look-ahead has no influence on the class of surface languages (cf. Corollary 4.79).

Corollary 4.86. For all $k \geq 1$,

  (i) $(T')^k = DBQREL \circ T^k$,

  (ii) $(T')^k$-Surface $= T^k$-Surface, and

 (iii) T^k-Surface is closed under linear tree transformations.

Proof.

  (i) By Theorem 4.77, $T' \subseteq DBQREL \circ T$. Also, by Corollary 4.82 and Theorem 4.81(2), $DBQREL \circ T \subseteq T'$. Hence $T' = DBQREL \circ T$.

We now show that $T' \circ T' = T' \circ T$. Trivially, $T' \circ T \subseteq T' \circ T'$. Also $T' \circ T' = T' \circ DBQREL \circ T$ and, by Theorem 4.81(1), $T' \circ DBQREL \subseteq T'$. Hence $T' \circ T' \subseteq T' \circ T$.

From this it is straightforward to see that $(T')^{k+1} = T' \circ T^k = DBQREL \circ T \circ T^k = DBQREL \circ T^{k+1}$.

  (ii) This is an immediate consequence of (i) and the fact that $RECOG$ is closed under linear tree transformations.

 (iii) This follows easily from (ii) and the fact that $(T')^k$ is closed under composition with linear tree transformations (Theorem 4.81(1), recall also Theorem 4.80). $\qquad\square$

We mention here that it can also be shown that T^k-Surface is closed under union, tree concatenation and tree concatenation closure. From that a lot of closure properties of T^k-Target and Target follow.

The relation between the top-down and the bottom-up surface tree languages is easy.

Corollary 4.87. For all $k \geq 1$,

  (i) T^k-Surface $= (B^k \circ LB)$-Surface and

      B^{k+1}-Surface $= (T^k \circ HOM)$-Surface;

  (ii) B^k-Surface $\subseteq T^k$-Surface $\subseteq B^{k+1}$-Surface.

Proof. (i) follows from the fact that $B = LB \circ HOM$ (Corollary 4.63(2)) and that $T' = HOM \circ LB$ (Theorem 4.85).

(ii) is an obvious consequence of (i).

Note that B-Surface $= HOM$-Surface (Exercise 4.56). $\qquad\square$

It is not known, but conjectured, that for all k the inclusions in Corollary 4.87(ii) are proper.

Note that, by taking yields, Corollary 4.87 also holds for the corresponding target languages. Again it is not known whether the inclusions are proper.

In the rest of this section we show that the emptiness-, the membership- and the finiteness-problem are solvable for Surface and Target.

Theorem 4.88. The emptiness- and membership-problem are solvable for Surface.

Proof. Let $M \in (REL \cup FTA \cup HOM)^*$ and $L \in RECOG$. Consider the tree language $M(L) \in$ Surface. Obviously, $M(L) = \emptyset$ iff $L \cap \mathrm{dom}(M) = \emptyset$. But, by Remark 4.61, $\mathrm{dom}(M)$ is recognizable. Hence, by Theorem 3.32 and Theorem 3.74, it is decidable whether $L \cap \mathrm{dom}(M) = \emptyset$.

To show solvability of the membership-problem note first that Surface is closed under intersection with a recognizable tree language (if $R \in RECOG$ and $\widehat{R}$ is the fta restriction such that $\mathrm{dom}(\widehat{R}) = R$, then $M(L) \cap R = (M \circ \widehat{R})(L)$). Now, for any tree t, $t \in M(L)$ iff $M(L) \cap \{t\} \neq \emptyset$. Since $\{t\}$ is recognizable, $M(L) \cap \{t\} \in$ Surface, and we just showed that it is decidable whether a surface tree language is empty or not. $\square$

To show decidability of the finiteness-problem we shall use the following result.

Lemma 4.89. Each monadic tree language in Surface is recognizable (and hence regular as a set of strings).

Proof. Let L be a monadic tree language. Obviously it suffices to show that, for each $k \geq 1$, if $L \in (T')^k$-Surface, then $L \in (T')^{k-1}$-Surface (where, by definition, $(T')^0$-Surface $= RECOG$). Suppose therefore that $L \in (T')^k$-Surface, so $L = (M_1 \circ \cdots \circ M_{k-1} \circ M_k)(R)$ for certain $M_i \in T'$ and $R \in RECOG$. Consider all right hand sides of rules in M_k. Obviously, since L is monadic, these right hand sides do not contain elements of rank ≥ 2, that is, they are monadic in the sense of Notation 4.42 (rules which have nonmonadic right hand sides may be removed). But from this it follows that M_k is linear. It now follows from Theorem 4.81(1) (and Corollary 4.55 and Theorem 4.80 in the case $k = 1$), that $L \in (T')^{k-1}$-Surface. $\square$

Theorem 4.90. The finiteness-problem is solvable for Surface.

Proof. Intuitively, a tree language is finite if and only if the set of paths through this tree language is finite. For $L \subseteq T_\Sigma$ we define $\mathrm{path}(L) \subseteq \Sigma^*$ recursively as follows

  (i) for $a \in \Sigma_0$, $\mathrm{path}(a) = \{a\}$;

  (ii) for $k \geq 1$, $a \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$,
    $\mathrm{path}(a[t_1 \cdots t_k]) = \{a\} \cdot (\mathrm{path}(t_1) \cup \cdots \cup \mathrm{path}(t_k))$;

  (iii) for $L \subseteq T_\Sigma$, $\mathrm{path}(L) = \bigcup_{t \in L} \mathrm{path}(t)$.

Thus, $\mathrm{path}(t)$ consists of all strings which can be obtained by following a path through t (for instance, if $t = a[bb[cc]]$, then $\mathrm{path}(t) = \{ab, abc\}$). We remark here that any other similar definition of "path" would also satisfy our purposes. It is left to the reader to show that, for any tree language L, L is finite iff $\mathrm{path}(L)$ is finite.

We now show that, given $L \subseteq T_\Sigma$, the set $\mathrm{path}(L)$ can be obtained by feeding L into a top-down tree transducer M_p: $M_p(L)$ will be equal to $\mathrm{path}(L)$, modulo the correspondence

between strings and monadic trees (see Definition 2.21). In fact, $M_p = (Q, \Sigma, \Delta, R, Q_d)$, where $Q = Q_d = \{p\}$, $\Delta_0 = \{e\}$, $\Delta_1 = \Sigma$ and R consists of the following rules:

  (i) for each $a \in \Sigma_0$, $p[a] \to a[e]$ is in R;

  (ii) for every $k \geq 1$, $a \in \Sigma_k$ and $1 \leq i \leq k$, $p[a[x_1 \cdots x_k]] \to a[p[x_i]]$ is in R.

Consequently, L is finite iff $M_p(L)$ is finite. Now, let $L \in$ Surface. Then, obviously, $M_p(L) \in$ Surface. Moreover $M_p(L)$ is monadic. Hence, by Lemma 4.89, $M_p(L)$ is recognizable. Thus, by Theorem 3.75, it is decidable whether $M_p(L)$ is finite. $\qquad \square$

To show that the above mentioned problems are solvable for Target, we need the following lemma.

Lemma 4.91. Each language in Target is (effectively) of the form $\mathrm{yield}(L)$ or $\mathrm{yield}(L) \cup \{\lambda\}$, where $L \subseteq T_\Sigma$ for some Σ such that $\Sigma_1 = \emptyset$ and $e \notin \Sigma$, $L \in$ Surface.

Proof. It is left as an <u>exercise</u> to show that, for any $L' \subseteq T_{\Sigma'}$, there exists a bottom-up tree transducer M such that $M(L') \subseteq T_\Sigma$ for some Σ satisfying the requirements, and such that $\mathrm{yield}(M(L')) = \mathrm{yield}(L') - \{\lambda\}$.

It is also left as an <u>exercise</u> to show that it is decidable whether $\lambda \in \mathrm{yield}(L')$. From these two facts the lemma follows. $\qquad \square$

Theorem 4.92. The emptiness-, membership- and finiteness-problem are solvable for Target.

Proof. It is obvious from the previous lemma that we may restrict ourselves to target-languages $\mathrm{yield}(L)$, where $L \in$ Surface and $L \subseteq T_\Sigma$ for some Σ such that $\Sigma_1 = \emptyset$ and $e \notin \Sigma_0$.

Obviously, $\mathrm{yield}(L) = \emptyset$ iff $L = \emptyset$. Hence the emptiness-problem is solvable by Theorem 4.88.

Note that, by Example 2.17, for a given $w \in \Sigma_0^*$, there are only a finite number of trees such that $\mathrm{yield}(t) = w$. From this and Theorem 4.88 the decidability of the membership-problem follows. Moreover it follows that $\mathrm{yield}(L)$ is finite iff L is finite. Hence, by Theorem 4.90, the finiteness-problem is solvable. $\qquad \square$

We note that it can be shown that Target is (properly, by Theorem 4.92) contained in the class of context-sensitive languages. Thus, Target lies properly between the context-free and the context-sensitive languages.

We finally note that, in a certain sense, Surface $\subseteq$ Target (cf. the similar situation for *RECOG* and *CFL*). In fact, let $L \subseteq T_\Sigma$ be in Surface. Let $[\![$ and $]\!]$ be two new symbols ("standing for $[$ and $]$"). Let Δ be the ranked alphabet such that $\Delta_0 = \Sigma \cup \{[\![,]\!]\}$, $\Delta_1 = \Delta_2 = \Delta_3 = \emptyset$ and $\Delta_{k+3} = \Sigma_k$ for $k \geq 1$. Let $M = (Q, \Sigma, \Delta, R, Q_d)$ be the (deterministic) top-down tree transducer such that $Q = Q_d = \{f\}$ and the rules are the following

  (i) for $k \geq 1$ and $a \in \Sigma_k$, $f[a[x_1 \cdots x_k]] \to a[a[\![f[x_1] \cdots f[x_k]]\!]]$ is in R,

(ii) for $a \in \Sigma_0$, $f[a] \to a$ is in R.

(Note that M is in fact a homomorphism). It is left to the reader to show that yield$(M(L)) = L\langle [\leftarrow [\![,] \leftarrow]\!] \rangle$ (as string languages).

5 Notes on the literature

In the text there are some references to

[A&U] A.V. Aho and J.D. Ullman, The theory of parsing, translation and compiling, I and II, Prentice-Hall, 1972.

[Sal] A. Salomaa, Formal languages, Academic Press, 1973.

An informal survey of the theory of tree automata and tree transducers (uptil 1970) is given by [Thatcher, 1973].

On Section 3

Bottom-up finite tree automata were invented around 1965 independently by [Doner, 1965, 1970] and [Thatcher & Wright, 1968] (and somewhat later by [Pair & Quere, 1968]). The original aim of the theory of tree automata was to apply it to the decision problems of second-order logical theories concerning strings. The connection with context-free languages was established in [Mezei & Wright, 1967] and [Thatcher, 1967], and the idea to give "tree-oriented" proofs for results on strings is expressed in [Thatcher, 1973] and [Rounds, 1970a]. Independently, results concerning parenthesis languages and structural equivalence were obtained by [McNaugthon, 1967], [Knuth, 1967], [Ginsburg & Harrison, 1967] and [Paull & Unger, 1968]. Top-down finite tree automata were introduced by [Rabin, 1969] and [Magidor & Moran, 1969], and regular tree grammars by [Brainerd, 1969]. The notion of rule tree languages occurs in one form or another in several places in the literature.

Most of the results of Section 3 can be found in the above mentioned papers.
Other work on finite tree automata and recognizable tree languages is written down in the following papers:

[Arbib & Give'on, 1968], automata on acyclic graphs, category theory;

[Brainerd, 1968], state minimalization of finite tree automata;

[Costich, 1972], closure properties of $RECOG$;

[Eilenberg & Wright, 1967], category theoretic formulation of fta;

[Ito & Ando, 1974], axioms for regular tree expressions;

[Maibaum, 1972, 1974], tree automata on "many-sorted" trees;

[Ricci, 1973], decomposition of fta;

[Takahashi, 1972, 1973], several results;

[Yeh, 1971], generalization of "semigroup of fa" to fta.

Remark: The notation of substitution (tree concatenation) can be formalized algebraically as in [Eilenberg & Wright, 1967], [Goguen & Thatcher, 1974], [Thatcher, 1970] and [Yeh, 1971].

Generalizations of finite automata are the following:

- finite automata on derivation graphs of type 0 grammars,
  [Benson, 1970], [Buttelmann, 1971], [Hart, 1974];

- probabilistic tree automata,
  [Ellis, 1970], [Magidor & Moran, 1970];

- automata and grammars for infinite trees,
  [Rabin, 1969], [Goguen & Thatcher, 1974], [Engelfriet, 1972];

- recognition of subsets of an arbitrary algebra (rather than the algebra of trees),
  [Mezei & Wright, 1967], [Shephard, 1969].

There are no real open problems in finite tree automata theory. It is only a question of (i) how far one wants to go in generalizing finite automata theory to trees (for instance, decomposition theory, theory of incomplete sequential machines, noncounting regular languages, etc), and (ii) which results on context-free languages one can prove via trees (for instance, Greibach normal form, Parikh's theorem, etc.).

On Section 4

For the literature on syntax-directed translation, see [A&U]. The notion of "generalized syntax-directed translation" is defined in [Aho & Ullman, 1971].

The top-down tree transducer was introduced in [Rounds, 1968], [Thatcher, 1970] and [Rounds, 1970b] as a model for syntax-directed translation and transformational grammars. The bottom-up tree transducer was introduced in [Thatcher, 1973]. The notion of a tree rewriting system occurs in one form or another in [Brainerd, 1969], [Rosen, 1971, 1973], [Engelfriet, 1971] and [Maibaum, 1974].

Most results of Section 4 can be found in [Rounds, 1970b], [Thatcher, 1970], [Rosen, 1971], [Engelfriet, 1971], [Baker, 1973] and [Ogden & Rounds, 1972]. Other papers on tree transformations are [Alagić, 1973], [Benson, 1971], [Bertsch, 1973], [Kosaraju, 1973], [Levy & Joshi, 1973], [Martin & Vere, 1970] and [Rounds, 1973].

We mention the following problems concerning tree transformations.

– "Statements such as "similar models have been studied by [x,y,z]" are symptomatic of the disarray in the development of the theory of translation and semantics" (free after [Thatcher, 1973]).

Establish the precise relationships between various models of syntax-directed translation, semantics of context-free languages and tree transducers (see [Goguen & Thatcher, 1974]):

  (i) Compare existing models of syntax-directed translation with top-down tree transducers, in particular with respect to the classes of translations they define (see Definition 4.41). See [A&U], [Aho & Ullman, 1971], [Thatcher, 1973] and [Martin & Vere, 1970].

  (ii) Define a tree transducer corresponding to the semantics definition method of [Knuth, 1968].

– Develop a general theory of operations on tree languages (tree AFL theory). Relate these operations to the yield languages, as illustrated in Section 3. This work was started by [Baker, 1973].

It would perhaps be convenient to consider tree transducers $(Q, \Sigma, \Delta, R, Q_d)$ such that R consists of rules $t_1 \to t_2$ with $t_1 \in Q[T_\Sigma(X)]$, $t_2 \in T_\Delta[Q(X)]$ in the top-down case, or $t_1 \in T_\Sigma(Q[X])$, $t_2 \in Q[T_\Delta(X)]$ in the bottom-up case.

– Consider surface tree languages and target languages more carefully. Prove that the class T-Target is incomparable with the class of indexed languages. Prove that the classes T^k-Surface form a proper hierarchy (see [Ogden & Rounds, 1972] and [Baker, 1973]). Prove that the classes T^k-Target form a proper hierarchy (then you also proved the previous one). Prove that DT-Target $\subsetneq$ T-Target. Is it possible to obtain each target language by a sequence of nondeleting (and nonerasing) tree transducers? etc.

– Consider the complexity of target languages and translations (see [Aho & Ullman, 1971], [Baker, 1973] and [Rounds, 1973]).

– What is the practical use of tree transducers? (see [A&U], [de Remer, 1974]).

Other subjects

We mention finally the following subjects in tree theory.

– Context-free tree grammars, [Fischer, 1968], [Rounds, 1969, 1970a, 1970b], [Downey, 1974], [Maibaum, 1974].

Let us explain briefly the notion of a context-free tree grammar. Consider a system $G = (N, \Sigma, R, S)$, where N is a ranked alphabet of nonterminals, Σ is a ranked alphabet of terminals, $V := N \cup \Sigma$, $S \in N_0$ is the initial nonterminal, and R is a finite set of rules of one of the forms

$A \to t$ with $A \in N_0$ and $t \in T_V$, or

71

$A[x_1 \cdots x_k] \to t$ with $A \in N_k$ and $t \in T_V(X_k)$.

G is considered as a tree rewriting system on T_V. If we let all variables in the rules range over T_V then G is called a context-free tree grammar. If we let all variables range over T_Σ, then G is called a bottom-up (or inside-out) context-free tree grammar. The language generated by G is $L(G) = \{t \in T_\Sigma \mid S \overset{*}{\Rightarrow} t\}$. In general the languages generated by G under the above two interpretations differ (consider for instance the grammar $S \to F[A]$, $F[x_1] \to f[x_1 x_1]$, $A \to a$, $A \to b$). Thus restriction to bottom-up ($\approx$ right-most in the string case) generation gives another language. On the other hand, restriction to top-down ($\approx$ left-most) generation can be done without changing the language. The yield of the class of context-free tree languages is equal to the class of indexed languages. The yield of the class of bottom-up context-free tree languages is called IO. These two classes are incomparable ([Fischer, 1968]).

How do these classes compare with the target languages? Is it possible to iterate the $CFL \to RECOG$ procedure and obtain results about (bottom-up) context-free tree languages from regular tree languages (of a "higher order") (see [Maibaum, 1974]). Is there any sense in considering pushdown tree automata?

– General computability on trees: [Rus, 1967], [Mahn, 1969], the Vienna method.

– Tree walking automata (at each moment of time the finite automaton is at one node of the tree; depending on its state and the label of the node it goes to the father node or to one of the son nodes): [Aho & Ullman, 1971], [Martin & Vere, 1970].

– Tree adjunct grammars (another, linguistically motivated, way of generating tree languages): [Joshi & Levy & Takahashi, 1973].

– Lindenmayer tree systems (parallel rewriting):
[Čulik, 1974], [Čulik & Maibaum, 1974], [Engelfriet, 1974], [Szilard, 1974].

References

A.V. Aho and J.D. Ullman, 1971. Translations on a context-free grammar, Inf. & Control 19, 439-475.

S. Alagić, 1973. Natural state transformations, Techn. Report 73B-2, Univ. of Massachusetts at Amherst.

M.A. Arbib and Y. Give'on, 1968. Algebra automata, I & II, Inf. & Control 12, 331-370.

B.S. Baker, 1973. Tree transductions and families of tree languages, Report TR-9-73, Harvard University (Abstract in: 5th Theory of Computing, 200-206).

D.B. Benson, 1970. Syntax and semantics: a categorical view, Inf. & Control 17, 145-160.

D.B. Benson, 1971. Semantic preserving translations, Working paper, Washington State University, Washington.

E. Bertsch, 1973. Some considerations about classes of mappings between context-free derivation systems, Lecture Note in Computer Science 2, 278-283.

D. Bjørner, 1972. Finite state tree computations, IBM Report RJ 1053.

W.S. Brainerd, 1968. The minimalization of tree automata, Inf. & Control 13, 484-491.

W.S. Brainerd, 1969. Tree generating regular systems, Inf. & Control 14, 217-231.

W.S. Brainerd, 1969a. Semi-Thue systems and representation of trees, 10th SWAT, 240-244.

H.W. Buttelmann, 1971. On generalized finite automata and unrestricted generative grammars, 3[d] Theory of Computing, 63-77.

O.L. Costich, 1972. A Medvedev characterization of sets recognized by generalized finite automata, Math. Syst. Th. 6, 263-267.

S.C. Crespi Reghizzi and P. Della Vigna, 1973. Approximation of phrase markers by regular sets, Automata, Languages and Programming (ed. M. Nivat), North-Holland Publ. Co., 367-376.

K. Čulik II, 1974. Structured OL-Systems, L-Systems (eds. Rozenberg & Salomaa), Lecture Notes in Computer Science 15, 216-229.

K. Čulik II and T.S.E. Maibaum, 1974. Parallel rewriting systems on terms, Automata, Languages and Programming (ed. Loeckx), Lecture Notes in Computer Science 14, 495-511.

J. Doner, 1970. Tree acceptors and some of their applications, J. Comp. Syst. Sci. 4, 406-451 (announced in Notices Am. Math. Soc. 12(1965), 819 as Decidability of the weak second-order theory of two successors).

P.J. Downey, 1974. Formal languages and recursion schemes, Report TR 16-74, Harvard University.

S. Eilenberg and J.B. Wright, 1967. Automata in general algebras, Inf. & Control 11, 452-470.

C.A. Ellis, 1970. Probabilistic tree automata, 2nd Theory of Computing, 198-205.

J. Engelfriet, 1971. Bottom-up and top-down tree transformations – a comparison, Memorandum 19, T.H. Twente, Holland (to be publ. in Math. Syst. Th.).

J. Engelfriet, 1972. A note on infinite trees, Inf. Proc. Letters 1, 229-232.

J. Engelfriet, 1974. Surface tree languages and parallel derivation trees, Daimi Report PB-44, Aarhus University, Denmark.

M.J. Fischer, 1968. Grammars with macro-like productions, 9th SWAT, 131-142 (Doctoral dissertation, Harvard University).

S. Ginsburg and M.A. Harrison, 1967. Bracketed context-free languages, J. Comp. Syst. Sci. 1, 1-23.

J.A. Goguen and J.W. Thatcher, 1974. Initial algebra semantics, 15th SWAT.

J.M. Hart, 1974. Acceptors for the derivation languages of phrase-structure grammars, Inf. & Control 25, 75-92.

T. Ito and S. Ando, 1974. A complete axiom system of super-regular expressions, Proc. IFIP Congress 74, 661-665.

A.K. Joshi, L.S. Levy and M. Takahashi, 1973. A tree generating system, Automata, Languages and Programming (ed. Nivat), North-Holland Publ. Co., 453-465.

D.E. Knuth, 1967. A characterization of parenthesis languages, Inf. & Control 11, 269-289.

D.E. Knuth, 1968. Semantics of context-free languages, Math. Syst. Th. 2, 127-145 (see also: correction in Math. Syst. Th. 5(1971), 95-96, and "Examples of formal semantics" in Lecture Notes in Mathematics 188 (ed. Engeler)).

S. Kosaraju, 1973. Context-sensitiveness of translational languages, 7th Princeton Conf. on Inf. Sci. and Syst.

L.S. Levy and A.K. Joshi, 1973. Some results in tree automata, Math. Syst. Th. 6, 334-342.

M. Magidor and G. Moran, 1969. Finite automata over finite trees, Techn. Report No. 30, Hebrew University, Jerusalem.

M. Magidor and G. Moran, 1970. Probabilistic tree automata and context-free languages, Israel J. Math. 8, 340-348.

F.K. Mahn, 1969. Primitiv-rekursive Funktionen auf Termmengen, Archiv f. Math. Logik und Grundlagenforschung 12, 54-65.

T.S.E. Maibaum, 1972. The characterization of the derivation trees of context-free sets of terms as regular sets, 13th SWAT, 224-230.

T.S.E. Maibaum, 1974. A generalized approach to formal languages, J. Comp. Syst. Sci. 8, 409-439.

D.F. Martin and S.A. Vere, 1970. On syntax-directed transduction and tree-transducers, 2nd Theory of Computing, 129-135.

R. McNaugthon, 1967. Parenthesis grammars, Journal of the ACM 14, 490-500.

J. Mezei and J.B. Wright, 1967. Algebraic automata and context-free sets, Inf. & Control 11, 3-29.

D.E. Muller, 1968. Use of multiple index matrices in generalized automata theory, 9th SWAT, 395-404.

W.F. Ogden and W.C. Rounds, 1972. Composition of n tree transducers, 4th Theory of Computing.

C. Pair and A. Quere, 1968. Définition et etude des bilangages réguliers, Inf. & Control 13, 565-593.

M.C. Paull and S.H. Unger, 1968. Structural equivalence of context-free grammars, J. Comp. Syst. Sci. 2, 427-463.

M.O. Rabin, 1969. Decidability of second-order theories and automata on infinite trees, Transactions of the Am. Math. Soc. 141, 1-35.

F.L. de Remer, 1974. Transformational grammars for languages and compilers, Lecture Notes in Computer Science 21.

G. Ricci, 1973. Cascades of tree-automata and computations in universal algebras, Math. Syst. Th. 7, 201-218.

B.K. Rosen, 1971. Subtree replacement systems, Ph. D. Thesis, Harvard University.

B.K. Rosen, 1973. Tree-manipulating systems and Church-Rosser theorems, Journal of the ACM 20, 160-188.

W.C. Rounds, 1968. Trees, transducers and transformations, Ph. D. Dissertation, Stanford University.

W.C. Rounds, 1969. Context-free grammars on trees, 1st Theory of Computing, 143-148.

W.C. Rounds, 1970a. Tree-oriented proofs of some theorems on context-free and indexed languages, 2nd Theory of Computing, 109-116.

W.C. Rounds, 1970b. Mappings and grammars on trees, Math. Syst. Th. 4, 257-287.

W.C. Rounds, 1973. Complexity of recognition in intermediate-level languages, 14th SWAT, 145-158.

T. Rus, 1967. Some observations concerning the application of the electronic computers in order to solve nonarithmetical problems, Mathematica 9, 343-360.

C.D. Shephard, 1969. Languages in general algebras, 1st Theory of Computing, 155-163.

A.L. Szilard, 1974. Ω-OL Systems, L-Systems (eds. Rozenberg and Salomaa), Lecture Notes in Computer Science 15, 258-291.

M. Takahashi, 1972. Regular sets of strings, trees and W-structures, Dissertation, University of Pennsylvania.

M. Takahashi, 1973. Primitive transformations of regular sets and recognizable sets, Automata, Languages and Programming (ed. Nivat), North-Holland Publ. Co., 475-480.

J.W. Thatcher, 1967. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, J. Comp. Syst. Sci. 1, 317-322.

J.W. Thatcher, 1970. Generalized^2 sequential machine maps, J. Comp. Syst. Sci. 4, 339-367 (also IBM Report RC 2466, also published in 1st Theory of Computing as "Transformations and translations from the point of view of generalized finite automata theory").

J.W. Thatcher, 1973. Tree automata: an informal survey, Currents in the Theory of Computing (ed. Aho), Prentice-Hall, 143-172 (also published in 4th Princeton Conf. on Inf. Sci. and Systems, 263-276, as "There's a lot more to finite automata theory than you would have thought").

J.W. Thatcher and J.B. Wright, 1968. Generalized finite automata theory with an application to a decision problem of second-order logic, Math. Syst. Th. 2, 57-81.

R. Turner, 1973. An infinite hierarchy of term languages – an approach to mathematical complexity, Automata, Languages and Programming (ed. Nivat), North-Holland Publ. Co., 593-608.

R.T. Yeh, 1971. Some structural properties of generalized automata and algebras, Math. Syst. Th. 5, 306-318.