JOURNAL OF COMPUTER AND SYSTEM SCIENCES 20, 150–202 (1980)

Tree Transducers, L Systems, and Two-Way Machines

JOOST ENGELFRIET

Twente University of Technology, Enschede, Netherlands

GRZEGORZ ROZENBERG

*University of Antwerp, U.I.A., Wilrijk, Belgium**

AND

GIORA SLUTZKI

Twente University of Technology, Enschede, Netherlands†

A relationship between parallel rewriting systems and two-way machines is investigated. Restrictions on the "copying power" of these devices endow them with rich structuring and give insight into the issues of determinism, parallelism, and copying. Among the parallel rewriting systems considered are the top-down tree transducer, the generalized syntax-directed translation scheme and the ETOL system, and among the two-way machines are the tree-walking automaton, the two-way finite-state transducer, and (generalizations of) the one-way checking stack automaton. The relationship of these devices to macro grammars is also considered. An effort is made to provide a systematic survey of a number of existing results.

1. INTRODUCTION

In this paper we make an effort to provide a systematic survey of the relationships between top-down tree transducers, generalized syntax-directed translation, L systems (in particular ETOL systems), two-way transducers, and checking machines (and, additionally, macro grammars). We investigate in particular the effect of restricting the "copying power" of these devices. The need for such a survey was prompted by the presence (in the literature) of quite a number of partial connections, together with the similarity between certain results in seemingly unconnected areas.

As a systematic approach to language definition we will use the concept of output-language of a transducer. In fact, a transducer may be viewed both as a generating device

* Present address : Rijksuniversiteit Leiden, Institute of Applied Mathematics and Computer Science, Leiden, Netherlands.

† Present address: Clarkson College of Technology, Dept. of Mathematics and Computer Science, Potsdam, New York 13676.

0022-0000/80/020150-53$02.00/0

(grammar) and as a recognizing device (acceptor) of its output language. The transducer is considered to be a grammar by viewing each of its computations as a derivation, the output of the computation as the generated string and its input as a control string of the derivation. The transducer is considered to be an acceptor by viewing its output as an input to the acceptor and its input as a preset memory of the acceptor (for instance, if the transducer is two way then the preset memory is a checking stack). Analogous remarks hold for trees instead of strings.

The main two models that we will use in this way to present this survey are the following.

(1.) The *top-down tree transducer* [48, 56, 9, 18], which serves as a model of the generalized syntax-directed translation (GSDT) of [4], and simultaneously, by restricting the input trees to be monadic (i.e., "vertical strings") and taking yields of the output trees, as a model of the (controlled) ETOL systems of [49], see [6, 20]. By viewing the top-down tree transducer as a grammar as explained above we obtain, therefore, a generalization of the ETOL system which may still be called a parallel rewriting system. The parallelism of the system is twofold: first, an independent parallelism (due to processing different input subtrees) and second, a dependent or synchronized parallelism (due to different processing of a single input subtree); only the second kind of parallelism is present in ETOL systems. Bounding the copying power of top-down tree transducers (i.e., the number of translations that can be made of each input subtree; see [4]) corresponds then to bounding the index (i.e., the number of nonterminals in each sentential form) of ETOL systems [50, 62].

(2.) A "new" tree-to-string transducer called *checking tree pushdown transducer* (or shortly ct–pd transducer). Such a pushdown transducer has an input tree, a pushdown memory, an output string, and a finite control with three pointers: one to a node of the input tree, one to the top of the pushdown tape, and one to the end of the output tape. The two elementary moves of the transducer are (depending, of course, on the label of the input node and the topsymbol of the pushdown) to move up to the father of the node and simultaneously pop the pushdown, or to move down to a (specific) son of the node and simultaneously push a symbol on the pushdown (and, in both cases, produce some output string). Thus the movements down and up the tree are synchronized with the pushes and pops on the pushdown, respectively.

By taking monadic input trees (i.e., strings) and viewing the transducer as an acceptor, as described above, we obtain the checking stack – pushdown (cs–pd) automaton of [61, 26], equivalent to the ETOL system [61], and by dropping the pushdown memory we obtain the tree-walking automaton of [4], equivalent to the top-down tree transducer with bounded copying power [4]. Taking both restrictions simultaneously (i.e., monadic input trees and no pushdown) results in the usual two-way finite-state transducer (or two-way gsm) of [3] and (viewed as an acceptor) the usual (one-way) checking stack automaton of [31].

The above two models are introduced in Sections 3 and 4 of this paper. Section 3 contains the definition of the top-down tree transducer together with several restrictions on its copying power, and its restriction to ETOL systems. The results in this section

provide an insight into the nature of the resulting restrictions on the generating power of the top-down tree transducer. In Section 4 the ct–pd transducer is defined and it is shown that it has the same generating power as the top-down tree transducer. This is the main result linking the parallel rewriting systems (of model (1)) to the two-way machines (of model (2)). It is then shown which restrictions on the ct–pd transducer correspond to the above restrictions on the top-down tree transducer. At the end of the section a comparison is made with several classes of stack automata and macro grammars considered in [26].

In Section 5 of this paper we consider the closure properties of the classes of languages defined by the above devices. Apart from the usual AFL and hyper-AFL operations we investigate closure under two-way deterministic finite-state transducers (and show, for instance, that the class of ranges of GSDT mappings is closed under such transducers).

Section 6 contains a discussion of macro grammars which are related to ET0L systems by their fixed-point characterization [13]. It turns out that the restrictions on the copying power of ET0L systems is related to both the copying power of the corresponding macro grammars and the number of arguments of their nonterminals.

Generalized macro grammars may be viewed as a particular kind of bottom-up tree transducer, equivalent to the top-down tree transducer.

2. Preliminaries

We assume the reader to be familiar with formal language theory [36, 52] and some tree language theory [57, 58, 59, 21]. In this section we fix some notation and recall some facts.

For a finite set A, $\#(A)$ denotes its cardinality.

For a string w, $|w|$ denotes its length and ***alph***(w) the set of symbols occurring in w (i.e., alph(w) is the smallest alphabet Σ such that $w \in \Sigma^*$). The empty string is denoted by λ, thus $|\lambda| = 0$ and alph$(\lambda) = \varnothing$. A language is λ-free if it does not contain λ.

$X = \{x_1, x_2, x_3, \ldots\}$ is a denumerably infinite set of variables, $X_0 = \varnothing$ and, for $n \geqslant 1$, $X_n = \{x_1, x_2, \ldots, x_n\}$. In examples we will use $x, y, z, \ldots$ rather than $x_1, x_2, x_3, \ldots$. For an alphabet Σ and strings $w_0 \in (\Sigma \cup X_n)^*$ and $w_1, \ldots, w_n \in \Sigma^*$ $(n \geqslant 0)$, $w_0[w_1, \ldots, w_n]$ denotes the result of substituting w_i for x_i in w_0 $(1 \leqslant i \leqslant n)$.

A *gsm mapping* is a mapping from languages into languages realized by a (nondeterministic) generalized sequential machine (i.e., a finite-state automaton which outputs a string for each input symbol, as defined in [36]). Similarly, a ***sequential machine mapping*** is a mapping from languages to languages realized by a (nondeterministic) sequential machine, i.e., a gsm which outputs one symbol for each input symbol. The classes of regular, linear, and context-free languages will be denoted by REG, LIN, and CF, respectively.

In the rest of this section we recall some tree terminology. An alphabet Σ is ranked if $\Sigma = \bigcup_{n=0}^{\infty} \Sigma_n$, where the Σ_n are (not necessarily disjoint) subsets of Σ such that only finitely many of them are nonempty. If $\sigma \in \Sigma_n$, then we say that σ has rank n. A tree over Σ is either a symbol of rank 0 or a string of the form $\sigma(t_1 \cdots t_n)$, where σ has rank n and t_i

is a tree over Σ ($1 \leqslant i \leqslant n$). The set of all trees over Σ is denoted T_Σ, thus $T_\Sigma \subseteq (\Sigma \cup \{(\,,\,)\})^*$. A tree language over Σ is a subset of T_Σ. If Y is a set of strings, then $T_\Sigma[Y]$ is the smallest set of strings such that $\Sigma_0 \cup Y \subseteq T_\Sigma[Y]$ and if $\sigma \in \Sigma_n$ and $t_i \in T_\Sigma[Y]$ for $1 \leqslant i \leqslant n$, then $\sigma(t_1 \cdots t_n) \in T_\Sigma[Y]$. Thus $T_\Sigma = T_\Sigma[\varnothing]$.

We shall also employ the usual more intuitive terminology concerning the above-defined finite labeled ordered rooted trees. Figure 1 displays the tree $t = b(aa(fg)\,c(d))$ over the ranked alphabet Σ with $\Sigma_0 = \{a, d, f, g\}$, $\Sigma_1 = \{c\}$, $\Sigma_2 = \{a\}$ and $\Sigma_3 = \{b\}$ (and $\Sigma_n = \varnothing$ otherwise). The root of t is the node labeled b. The nodes labeled f and g are the first and second son of the rightmost node labeled a, and the latter is their father. The father is connected to its sons by arcs. A path is a sequence of connected arcs. The tree $a(fg)$ is a subtree of t with (the rightmost) a as root. The nodes labeled a, f, g, and d are the leaves of t, and its yield is $afgd$. The height of t is 3.

Formally, for a ranked alphabet Σ, t_1 is a subtree of t_2 is $t_2 = ut_1v$ for some $u, v \in (\Sigma \cup \{(\,,\,)\})^*$. The yield and height are defined formally as follows:

(i) for $\sigma \in \Sigma_0$, $\text{yield}(\sigma) = \sigma$ and $\text{height}(\sigma) = 1$;

(ii) for $\sigma \in \Sigma_n\,(n \geqslant 1)$, $\text{yield}(\sigma(t_1 \cdots t_n)) = \text{yield}(t_1) \cdots \text{yield}(t_n)$ and $\text{height}(\sigma(t_1 \cdots t_n)) = 1 + \max\{\text{height}(t_i) \mid 1 \leqslant i \leqslant n\}$.

Thus $\text{yield}(t) \in \Sigma_0^+$. We will often abbreviate "yield" by "y".

The reader is assumed to be familiar with the notion of a (nondeterministic) finite tree automaton [57]. A tree language is *recognizable* if it can be accepted by a finite tree automaton. The class of recognizable tree languages will be denoted by REC. For properties of REC see [57–59, 21]. We only recall that $y(\text{REC}) = \text{CF}$. A *finite-state relabeling* is a mapping realized by a (top-down or bottom-up) finite tree automaton which changes the symbols on the tree; it is the tree analog of the sequential machine mapping. The notions of deterministic top-down and deterministic bottom-up finite-state relabeling should be clear; more formal definitions will be given in Definition 3.1.7 of Section 3 (see also [18, 19]).

A ranked alphabet Σ is *monadic* if $\Sigma = \Sigma_0 = \Sigma_1$ and $\Sigma_n = \varnothing$ for $n \geqslant 2$, i.e., each symbol in Σ has ranks 0 and 1 and no other ranks. Trees and tree languages over such a Σ are also called monadic. We will identify an unranked alphabet Σ with the corresponding monadic ranked alphabet, the monadic tree $\sigma_1(\sigma_2(\cdots \sigma_{n-1}(\sigma_n) \cdots))$ with the string

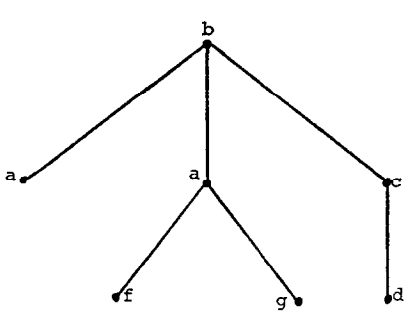

FIG. 1. A labeled tree.

$\sigma_1\sigma_2 \cdots \sigma_{n-1}\sigma_n$, and hence T_Σ with Σ^+ and tree languages over Σ with (λ-free) string languages over Σ. Whenever it does not give rise to confusion, we will identify $\mathfrak{L}$ with $\mathfrak{L}' = \{L - \{\lambda\} \mid L \in \mathfrak{L}\}$ for each class of languages $\mathfrak{L}$. It is easy to see that a class of string languages is closed under finite-state relabelings (when viewed as a class of monadic tree languages) iff it is closed under sequential machine mappings. Similarly, a λ-free string language is recognizable (as a monadic tree language) iff it is regular; thus we identify REG with the monadic recognizable tree languages (modulo λ).

We close this section with the definition of a particular operation on tree languages called *insertion of regular languages*. Intuitively it generalizes to trees a kind of regular substitution: above each node of the tree a regular (monadic) tree language is substituted. The formal definition is as follows. Let Δ be an alphabet and Σ a ranked alphabet. Let $\Sigma \cup \Delta$ be the ranked alphabet such that $(\Sigma \cup \Delta)_1 = \Sigma_1 \cup \Delta$ and $(\Sigma \cup \Delta)_n = \Sigma_n$ for $n \neq 1$. For $w = \delta_1\delta_2 \cdots \delta_n \in \Delta^*$ and $t \in T_\Sigma$, we denote by $w(t)$ the tree $\delta_1(\delta_2(\cdots \delta_n(t) \cdots))$ over $\Sigma \cup \Delta$. Let for each $\sigma \in \Sigma_n$ ($n \geqslant 0$) a regular language $f_\sigma \subseteq \Delta^*$ be specified. The mapping f from tree languages over Σ to tree languages over $\Sigma \cup \Delta$ is defined by

(i) for $\sigma \in \Sigma_0$, $f(\sigma) = \{w(\sigma) \mid w \in f_\sigma\}$;

(ii) for $\sigma \in \Sigma_n$ ($n \geqslant 1$), $f(\sigma(t_1 \cdots t_n)) = \{w(\sigma(t_1' \cdots t_n')) \mid w \in f_\sigma \text{ and } t_i' \in f(t_i)\}$;

(iii) for $L \subseteq T_\Sigma$, $f(L) = \{f(t) \mid t \in L\}$.

The mapping f is called the insertion of the regular languages f_σ. If $\mathfrak{L}$ is a class of tree languages such that if $L \in \mathfrak{L}$ then $f(L) \in \mathfrak{L}$ for each such mapping f, then $\mathfrak{L}$ is said to be closed under insertion of regular languages. It is easy to prove that REC is closed under insertion of regular languages. For monadic tree languages, insertion of regular languages is a special case of λ-free regular substitution. Thus we can say that REG and CF are closed under insertion of regular languages.

3. ETOL Systems and Top-Down Tree Transducers

In this section we study parallel rewriting systems, in particular the ETOL systems of [49] and the top-down tree transducers of [48, 56, 9, 18]. Of particular interest to us are the ETOL languages and the top-down tree transformation languages defined, respectively, by these two devices. It is well known that the latter class, when restricted to monadic input tree languages, is equal to the class of ETOL languages [6, 20]. In the same way deterministic ETOL systems (or EDTOL systems) correspond to deterministic top-down tree transducers.

We will be concerned with putting bounds on the derivations of ETOL systems and top-down tree transducers. Intuitively, these bounds will restrict the copying facility (i.e., the number of translations that is made of a subtree) of tree transducers, and correspondingly the amount of parallelism (i.e., the number of "simultaneously active" symbols during derivations) of ETOL systems. In particular, ETOL systems of finite index (introduced in [50, 62]) correspond to top-down tree transducers with bounded copying (discussed in [4]), and metalinear ETOL systems (introduced in [62, 51]) correspond to

"metalinear" top-down transducers, i.e., transducers that are allowed to copy only in the first step of their computation. It should be clear that the number of states of a top-down tree transducer (the number of "active" symbols of an ET0L system) is directly related to its "copying power." We will show that restricting the number of states gives rise to a proper hierarchy of tree transformation languages, with very simple counter-examples (viz., metalinear ET0L languages). This proves in one stroke that all the bounded classes which we consider are proper hierarchies with respect to their bounds. We then show that the properties of determinism, bounded copying, and metalinearity are increasingly restrictive for tree transformation languages and (independently) ET0L languages. In particular we state rather general results showing that deterministic tree transformation languages are not closed under inverse homomorphism, bounded-copying tree transformation languages have semilinear Parikh sets, and metalinear tree transformation languages are not closed under Kleene star. As a byproduct we obtain the fact that $\mathfrak{L}$-controlled EDT0L systems (for suitable $\mathfrak{L}$, not containing CF) cannot generate all context-free languages, cf. [14].

This section is divided into three parts. The first part introduces the main notions to be used in the rest of this section as well as in the rest of this paper. Several already existing concepts are reformulated here in an attempt to bring forth the similarities between them. The second part contains the main results of this section whereas the last part is intended as a bridge between the first two sections.

3.1. *Terminology and Definitions*

A (nondeterministic) generalized sequential machine can be described by a rewriting system (à la [42]) with rules of the form $q(\sigma x) \to wq'(x)$, where q and q' are states, σ is an input symbol, and w is an output string. Sentential forms of a derivation are of the form $wq(v)$, where w is the output, q the state, and v the rest of the input at this moment of time. An application of the above rule consists of replacing $q(\sigma u)$ by $wq'(u)$ (in a sentential form to which this rule is applied). The translation realized by the gsm consists of all pairs $\langle v, w\rangle$ such that $q_0(v) \overset{*}{\Rightarrow} wq_f(\lambda)$, for an initial state q_0 and some final state q_f. Another way of looking at the rules is by interpreting them as recursive equations; thus the rule $q(\sigma x) \to wq'(x)$ says that the q-translation of a string with first symbol σ is equal to w followed by the q'-translation of the rest of the string. From both points of view a very natural generalization of the gsm is obtained by allowing any number of translations (i.e., elements of the form $q'(x)$) to appear in the right-hand side of rules, mixed with output symbols. For example, a translation of the string $\delta\sigma^n\tau$ into $a^nb^nfc^nd^n$ may be described by the following rules:

$$q_0(\delta x) \to q_1(x)\, f q_2(x),$$
$$q_1(\sigma x) \to a q_1(x) b,\ q_1(\tau x) \to \lambda,$$
$$q_2(\sigma x) \to c q_2(x) d,\ q_2(\tau x) \to \lambda.$$

Such a device, called an ET0L system, is defined formally as follows.

(3.1.1) DEFINITION. An *ET0L system* is a construct $M = (Q, \Sigma, \Delta, q_0, R)$, where Q

is a finite set of states, Σ is the input alphabet, Δ is the output alphabet, $q_0 \in Q$ is the initial state, and R is a finite set of rules of the form

$$q(\sigma x) \to w_1 q_1(x)\, w_2 q_2(x) \cdots w_n q_n(x) w_{n+1}$$

with $n \geqslant 0$; $q, q_1, \ldots, q_n \in Q$; $\sigma \in \Sigma$ and $w_1, \ldots, w_{n+1} \in \Delta^*$. M is *deterministic* (an EDT0L system) if different rules in R have different left-hand sides.

A *sentential form* of M is a string in $(\Delta \cup Q(\Sigma^*))^*$, i.e., a string of the form $u_1 p_1(v_1)\, u_2 p_2(v_2) \cdots u_m p_m(v_m) u_{m+1}$ with $m \geqslant 0$, $p_i \in Q$, $u_i \in \Delta^*$ and $v_i \in \Sigma^*$. An application of the above rule to a sentential form s_1 consists of replacing an occurrence of a substring $q(\sigma v)$ in s_1 (for some $v \in \Sigma^*$) by $w_1 q_1(v) w_2 \cdots w_n q_n(v) w_{n+1}$, resulting in a new sentential form s_2; this direct derivation step between s_1 and s_2 is denoted by $s_1 \Rightarrow s_2$. A sequence of direct derivation steps $s_1 \Rightarrow s_2 \Rightarrow \cdots \Rightarrow s_k$ is called a *derivation* and is denoted by $s_1 \overset{*}{\Rightarrow} s_k$.

If in a sentential form $s_1 = u_1 p_1(v_1) u_2 \cdots u_m p_m(v_m) u_{m+1}$ all $p_i(v_i)$ are rewritten simultaneously, by application of a rule to each of them, resulting in a new sentential form s_2, then we have a *parallel derivation step*, denoted by $s_1 \Rightarrow_\pi s_2$. Parallel derivation is denoted by $\overset{*}{\Rightarrow}_\pi$.

The *translation* realized by M is $\{\langle v, w\rangle \in \Sigma^* \times \Delta^* \mid q_0(v) \overset{*}{\Rightarrow} w\}$ and the *language* generated by M, denoted by $L(M)$, is the range of this translation, i.e., $L(M) = \{w \in \Delta^* \mid q_0(v) \overset{*}{\Rightarrow} w \text{ for some } v \in \Sigma^*\}$. ∎

We denote by ET0L and EDT0L the classes of languages generated by ET0L and EDT0L systems, respectively.

One should observe that parallel derivation is equivalent to the ordinary one, i.e., $q_0(v) \overset{*}{\Rightarrow}_\pi w$ iff $q_0(v) \overset{*}{\Rightarrow} w$. Parallel derivation has the advantage that each sentential form obtained by it has the form $w_1 p_1(v)\, w_2 p_2(v) \cdots w_m p_m(v) w_{m+1}$, indicating precisely the output corresponding to an initial piece of input processed so far and the states with which the rest of the input, v, is to be translated. Thus in a parallel derivation one can keep track of the rest of the input separately. This also shows the equivalence of the above definition of ET0L system with the original one [49, 50]: the elements of Q are usually called active symbols (or nonterminals), elements of Σ are the tables, those of Δ are the nonactive symbols (or terminals), q_0 is the axiom (or initial nonterminal), and the elements of R determine the rules in the tables of the ET0L system; to the rule displayed in Definition 3.1.1 corresponds the production $q \to w_1 q_1 w_2 q_2 \cdots w_n q_n w_{n+1}$ in table σ (and moreover, each table contains rules $a \to a$ for all $a \in \Delta$). A more formal proof of the equivalence of the above definition to the usual one is left to the reader (cf. Theorem 1 of [50]).

In ET0L systems additional generative power can be gained by controlling the derivations of the system by a control language out of some class of languages, see [29, 7]. In our version of ET0L systems this amounts to considering the image of a class of languages under ET0L translations.

(3.1.2) DEFINITION. Let $\mathfrak{L}$ be a class of languages. An *$\mathfrak{L}$-controlled ET0L system* (or $\mathfrak{L}$-preset ET0L system) is a pair (M, L), where $M = (Q, \Sigma, \Delta, q_0, R)$ is an ET0L

system, $L \subseteq \Sigma^*$, and $L \in \mathfrak{L}$ (L is the *control language*). (M, L) is *deterministic* (an $\mathfrak{L}$-controlled EDT0L system) if M is deterministic. The language generated by (M, L) is $M(L) = \{w \in \Delta^* \mid q_0(v) \overset{*}{\Rightarrow} w \text{ for some } v \in L\}$. ▮

ET0L($\mathfrak{L}$) and EDT0L($\mathfrak{L}$) denote, respectively, the classes of languages generated by $\mathfrak{L}$-controlled ET0L and EDT0L systems.

Note that, due to the form of the rules of an ET0L system, the control string λ never produces a derivation. Hence $M(L) = M(L - \{\lambda\})$ and ET0L($\mathfrak{L}$) = ET0L($\mathfrak{L}'$), where $\mathfrak{L}' = \{L - \{\lambda\} \mid L \in \mathfrak{L}\}$.

We now define a number of restrictions on (controlled) ET0L systems, obtained by putting bounds on the derivations in these systems.

(3.1.3) DEFINITION. Let $L \in \mathfrak{L}$ be a (control) language, (M, L) an $\mathfrak{L}$-controlled ET0L system, $M = (Q, \Sigma, \Delta, q_0, R)$, and $k \geqslant 1$ an integer. Consider a parallel derivation

$$D_\pi\colon q_0(v) = w_1 \underset{\pi}{\Rightarrow} w_2 \underset{\pi}{\Rightarrow} w_3 \underset{\pi}{\Rightarrow} \cdots \underset{\pi}{\Rightarrow} w_n = w \in \Delta^*.$$

D_π has *state-bound* k if for every i $(1 \leqslant i \leqslant n)$ the number of different states that occur in w_i is at most k. Thus when computing the state-bound of a derivation we do not count multiple occurrences of the same state in a sentential form. Note that every derivation in M has state-bound $\#(Q)$ trivially. The parallel derivation D_π is of *index* k (or, has *copying-bound* k) if each w_i contains at most k occurrences of states. Thus in the case of copying-bound all occurrences of states in a sentential form are taken into account.

(M, L) has state-bound k {is of index k} if for every $w \in M(L)$ there exist $v \in L$ and a parallel derivation $q_0(v) \overset{*}{\Rightarrow}_\pi w$ with state-bound k {of index k}. (M, L) is *finite index* (or *finite copying*) if (M, L) is of index k for some k. The same terminology holds for M if it is true of (M, Σ^*); note that $M(\Sigma^*) = L(M)$.

M is *k-metalinear* if (1) q_0 does not appear in the right-hand sides of rules of M and (2) for each rule $q(\sigma x) \to w_1 q_1(x) w_2 \cdots w_n q_n(x) w_{n+1}$: if $q = q_0$ then $n \leqslant k$ and if $q \neq q_0$ then $n \leqslant 1$. (M, L) is k-metalinear if M is; M and (M, L) are *metalinear* if M is k-metalinear for some k. ▮

The classes of languages defined by the systems in the previous definition are given the same names. "Finite index" and "metalinear" are denoted by the subscripts "FIN" and "ml" respectively. Bounds are indicated by the subscript "(k)." Thus the classes of $\mathfrak{L}$-controlled state-bound k, index k, finite index, k-metalinear, and metalinear ET0L languages are denoted by $\text{ET0L}_{(k)}(\mathfrak{L})$, $\text{ET0L}_{\text{FIN}(k)}(\mathfrak{L})$, $\text{ET0L}_{\text{FIN}}(\mathfrak{L})$, $\text{ET0L}_{\text{ml}(k)}(\mathfrak{L})$, and $\text{ET0L}_{\text{ml}}(\mathfrak{L})$, respectively. The uncontrolled classes are obtained by dropping ($\mathfrak{L}$) and in the deterministic case "D" is inserted between "E" and "T" in the above names. Note that $\text{ET0L}_{\text{ml}(k)}(\mathfrak{L}) \subseteq \text{ET0L}_{\text{FIN}(k)}(\mathfrak{L}) \subseteq \text{ET0L}_{(k)}(\mathfrak{L})$.

The notions of index and metalinearity for ET0L systems were introduced in [50, 51, 62]. It can be shown easily that the absolutely parallel grammars of [45] are equivalent to ET0L systems of finite index, cf. [62]. The equal matrix grammars of [54] are a special

case of metalinear ET0L systems (see also [37]). The notion of state-bound is introduced in this paper. Its naturalness is apparent in particular for EDT0L systems: state-bound k means that of each suffix of the input string at most k different translations are made (which may each occur more than once in the output string).

We note that under very weak conditions on $\mathfrak{L}$, the $\mathfrak{L}$-controlled ET0L systems defined here are equivalent to those in the literature [29, 7]; it is sufficient to require closure under right-marking (i.e., if $L \in \mathfrak{L}$ and b is a new symbol, then $Lb \in \mathfrak{L}$) to account for a "final table" (see the proof of Theorem 1 in [50]) or preferably closure under (nondeterministic) sequential machine mappings, to change the last symbol into a new one (so that it can act as a "final table").

(3.1.4) EXAMPLES. (i) The rules mentioned just before Definition 3.1.1 define a 2-metalinear ET0L system.

(ii) Consider the ET0L system $M = (Q, \Sigma, \Delta, q_0, R)$ with $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{\sigma, \tau, \delta\}$, $\Delta = \{a, b, c, \#\}$ and R consisting of the rules

$$q_0(\sigma x) \to q_1(x)\, q_2(x), \qquad q_0(\delta x) \to \lambda,$$
$$q_1(\tau x) \to a q_1(x) b, \qquad q_1(\sigma x) \to \lambda, \qquad q_1(\delta x) \to \lambda,$$
$$q_2(\tau x) \to c q_2(x), \qquad q_2(\sigma x) \to \# q_1(x)\, q_2(x), \qquad q_2(\delta x) \to \#.$$

M translates $\sigma\tau^{n_1}\sigma\tau^{n_2} \cdots \sigma\tau^{n_k}\delta$ into $a^{n_1}b^{n_1}c^{n_1} \# a^{n_2}b^{n_2}c^{n_2} \# \cdots \# a^{n_k}b^{n_k}c^{n_k} \#$; for instance

$$\begin{aligned} q_0(\sigma\tau\tau\sigma\tau\delta) &\underset{\pi}{\Rightarrow} q_1(\tau\tau\sigma\tau\delta)\, q_2(\tau\tau\sigma\tau\delta) \\ &\underset{\pi}{\Rightarrow} a q_1(\tau\sigma\tau\delta)\, bc q_2(\tau\sigma\tau\delta) \\ &\underset{\pi}{\Rightarrow} a^2 q_1(\sigma\tau\delta)\, b^2c^2 q_2(\sigma\tau\delta) \\ &\underset{\pi}{\Rightarrow} a^2b^2c^2 \# q_1(\tau\delta)\, q_2(\tau\delta) \\ &\underset{\pi}{\Rightarrow} a^2b^2c^2 \# a q_1(\delta)\, bc q_2(\delta) \\ &\underset{\pi}{\Rightarrow} a^2b^2c^2 \# abc \#. \end{aligned}$$

It is easy to see that M is an $\text{EDT0L}_{\text{FIN}(2)}$ system generating the language $L(M) = \{a^n b^n c^n \# \mid n \geqslant 0\}^*$.

(iii) The language $\{w \in \{a, b\}^* \mid |w| = 2^n \text{ for some } n \geqslant 0\}$ is generated by the ET0L system $M = (\{q_0\}, \{\sigma, \tau\}, \{a, b\}, q_0, R)$ with rules $q_0(\sigma x) \to q_0(x)\, q_0(x)$, $q_0(\tau x) \to a$ and $q_0(\tau x) \to b$. M has state-bound 1 and is, therefore, and $\text{ET0L}_{(1)}$ system.

The language $\{w \in \{a, b\}^* \mid |w| = 2^{2n} \text{ for some } n \geqslant 0\} \cup \{w \in \{c, d\}^* \mid |w| = 2^{2n+1} \text{ for some } n \geqslant 0\}$ is generated by the $\text{ET0L}_{(1)}$ system with the following rules.

$$q_0(\sigma x) \to q_1(x)\, q_1(x), \qquad q_0(\tau x) \to a, \qquad q_0(\tau x) \to b,$$
$$q_1(\sigma x) \to q_0(x)\, q_0(x), \qquad q_1(\tau x) \to c, \qquad q_1(\tau x) \to d.$$

The language $\{w \in \{a, b\}^* \mid |w| = 2^{2^n} \text{ for some } n \geqslant 0\}$ is generated by the EDT0L-controlled ET0L system (M, L), where M was defined above and $L = \{\sigma^{2^n}\tau \mid n \geqslant 0\}$. ∎

We now consider tree transducers. The top-down tree transducer may be viewed as a generalization of the gsm to trees [56], and in fact, it can be defined even as a generalization of the intermediate concept of ET0L system, cf. [20, 18]. In this paper we will be interested mainly in transductions of trees into strings, thus making the similarity to ET0L systems even stronger.

(3.1.5) DEFINITION. A *top-down tree-to-string transducer* (abbreviated as "yT transducer") is a construct $M = (Q, \Sigma, \Delta, q_0, R)$, where Q is a finite set of states, Σ is the ranked input alphabet, Δ is the output alphabet, $q_0 \in Q$ is the initial state, and R is a finite set of rules of the form

$$q(\sigma(x_1 \cdots x_k)) \to w_1 q_1(x_{i_1})\, w_2 q_2(x_{i_2}) \cdots w_n q_n(x_{i_n})\, w_{n+1}$$

with $n, k \geqslant 0$; $q, q_1, \ldots, q_n \in Q$; $\sigma \in \Sigma_k$; $w_1, \ldots, w_{n+1} \in \Delta^*$, and $1 \leqslant i_m \leqslant k$ for $1 \leqslant m \leqslant n$ (if $k = 0$ then the left-hand side is $q(\sigma)$). M is *deterministic* if different rules in R have different left-hand sides. M is *linear* if, for each rule in R, no x_i occurs more than once in its right-hand side. M is *λ-free* if λ is not the right-hand side of any rule in R.

A sentential form of M is an element of $(Q(T_\Sigma) \cup \Delta)^*$, i.e., a string of the form $u_1 p_1(t_1)\, u_2 p_2(t_2) \cdots u_m p_m(t_m) u_{m+1}$ with $m \geqslant 0$, $p_i \in Q$, $u_i \in \Delta^*$ and $t_i \in T_\Sigma$. For sentential forms s_1 and s_2 we write $s_1 \Rightarrow s_2$ if s_2 is obtained from s_1 by replacing a substring $q(\sigma(t_1 \cdots t_k))$, for certain $t_1, \ldots, t_k \in T_\Sigma$, by $w_1 q_1(t_{i_1})\, w_2 q_2(t_{i_2}) \cdots w_n q_n(t_{i_n}) w_{n+1}$, using the rule above. As usual, $\overset{*}{\Rightarrow}$ is used to denote derivations. The (tree-to-string) *translation* realized by M, also denote by M, is defined by $M = \{\langle t, w\rangle \in T_\Sigma \times \Delta^* \mid q_0(t) \overset{*}{\Rightarrow} w\}$. ∎

In this paper we will be interested in the ranges of yT transducers and, more generally, the images of a class of tree languages under these transducers. The practical motivation of this approach is the syntax-directed translation of context-free languages (or other languages), see [48, 4].

(3.1.6) DEFINITION. Let $\mathfrak{L}$ be a class of tree languages. A *top-down $\mathfrak{L}$-tree transformation system* (or an $\mathfrak{L}$-preset yT transducer) is a pair (M, L), where $M = (Q, \Sigma, \Delta, q_0, R)$ is a yT transducer, $L \subseteq T_\Sigma$ and $L \in \mathfrak{L}$. (M, L) is *deterministic* {*linear*} if M is. The language generated by (M, L) is $M(L) = \{w \in \Delta^* \mid q_0(t) \overset{*}{\Rightarrow} w \text{ for some } t \in L\}$. $M(L)$ is called a *top-down $\mathfrak{L}$-tree transformation language*. If $\mathfrak{L}$ is clear from the context, then we drop $\mathfrak{L}$ in this terminology. In (M, L), L is called the *input language*. ∎

The class of {deterministic} top-down tree-to-string transducers, and the class of translations they realize will both be denoted by yT {yDT}. The class of {deterministic} top-down $\mathfrak{L}$-tree transformation languages will be denoted by $yT(\mathfrak{L})$ {$yDT(\mathfrak{L})$}. The reason for the "y" is that the $\mathfrak{L}$-tree transformation languages are usually defined by taking the yield of the tree languages which are images of $\mathfrak{L}$-tree languages under conventional top-down tree transducers [48].

(3.1.7) DEFINITION. A *top-down tree transducer* is a top-down yT transducer $M = (Q, \Sigma, \Delta \cup \{(\ ,\ )\}, q_0, R)$ such that Δ is a ranked alphabet and if $q(\sigma(x_1 \cdots x_k)) \to v$ is in R, then $v \in T_\Delta[Q(X_k)]$. M is a (top-down) *finite-state relabeling* if all rules in R are of the form $q(\sigma(x_1 \cdots x_k)) \to \tau(q_1(x_1) \cdots q_k(x_k))$ for $\sigma \in \Sigma_k$ and $\tau \in \Delta_k$. M is a (top-down) *finite tree automaton* if $\Sigma = \Delta$ and all rules in R have the above form with $\tau = \sigma$. The deterministic and linear cases are defined as in (3.1.5).

A finite-state relabeling is said to be *deterministic bottom-up* if, for given $q_1, \ldots, q_k \in Q$ and $\sigma \in \Sigma_k$, there is at most one rule of the above form in R (and moreover the transducer is allowed to have more than one initial state). ∎

Note that REC is the class of all domains of finite tree automata. It is known that REC is closed under linear tree transducers [56, 21].

By dropping "y" from the name of a class of top-down tree transformation languages we obtain the name of the corresponding class of tree languages. Thus $DT(\mathfrak{L})$ denotes the class of images of $\mathfrak{L}$-tree languages under deterministic top-down tree transducers. It should be clear that this is a class of tree languages. It is straightforward to show that for each λ-free $M \in yT$ there is an $M' \in T$ such that $M = \{\langle t_1, \text{yield}(t_2)\rangle \mid \langle t_1, t_2\rangle \in M'\}$, and vice versa. The same holds for yDT and DT. Moreover, if $\mathfrak{L}$ is closed under finite-state relabelings, then $yT(\mathfrak{L}) = \text{yield}(T(\mathfrak{L}))$ and $yDT(\mathfrak{L}) = \text{yield}(DT(\mathfrak{L}))$ [19 (Corollary 4.5)].

There is a close relationship between yDT(REC) and the generalized syntax-directed translations (GSDT) of [4, 5]. Suppose that the recognizable input tree language of a top-down tree transformation system consists of derivation trees of a context-free grammar, such that the nodes of a derivation tree are labeled with the rules of the grammar, see [4]. Let σ denote the context-free rule $A \to v_1 B_1 v_2 B_2 \cdots v_k B_k v_{k+1}$, where $A, B_1, \ldots, B_k$ are nonterminals and $v_1, \ldots, v_{k+1}$ terminal strings. A rule $q(\sigma(x_1 \cdots x_k)) \to w_1 q_1(x_{i_1}) \cdots w_n q_n(x_{i_n}) w_{n+1}$ of the yT transducer corresponds then to the semantic rule $\tau_q(A) = w_1 \tau_{q_1}(B_{i_1}) \cdots w_n \tau_{q_n}(B_{i_n}) w_{n+1}$ associated with the rule σ (for notation see [4]).

Using the fact that each recognizable tree language is a projection of the set of derivation trees of a context-free grammar [58] it is easy to show that the class of (string-to-string) GSDT is equal to the class of all translations $\{\langle \text{yield}(t_1), w_1\rangle \mid \langle t_1, w_1\rangle \in M \text{ and } t_1 \in L\}$, where $L \in$ REC and (M, L) is a deterministic top-down tree transformation system. Hence yDT(REC) is the class of ranges of GSDT. Also, yT may be viewed as the appropriate nondeterministic version of GSDT.

We now want to introduce bounds on the derivations of yT transducers, as was done in the case of ET0L systems. In order to do so we need the concept of *state-sequence of a derivation at a node of the input tree.* Intuitively, it is the sequence of states in which the transducer starts to translate (the left-to-right sequence of copies of) the subtree which has the given node as root. Another way of looking at this concept (before formally defining it) is the following. Consider a derivation $q_0(t) \overset{*}{\Rightarrow} w$ of $M \in yT$ and a node d in the input tree t. Take $t_1 \in T_\Sigma[\{x\}]$, with a single occurrence of x, and $t_2 \in T_\Sigma$ such that $t = t_1[t_2]$ and d is the root of t_2; see Fig. 2 (recall that $t_1[t_2]$ is the result of substituting t_2 for x in t_1). Now delete from the original derivation all the derivation steps which operate on t_2. The new derivation, keeping t_2 untouched leads to $q_0(t) \overset{*}{\Rightarrow} w_1 q_1(t_2) w_2 \cdots w_n q_n(t_2) w_{n+1}$,

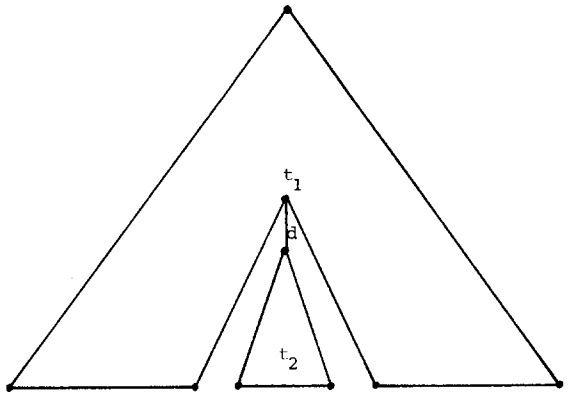

FIG. 2. Tree with subtree.

such that if $q_i(t_2) \overset{*}{\Rightarrow} v_i$ by those derivation steps which were deleted, we have $w = w_1v_1w_2v_2 \cdots w_nv_nw_{n+1}$. The state-sequence of the original derivation at the node d is then $\langle q_1, q_2, \ldots, q_n\rangle$. Thus $v_1, \ldots, v_n$ are the translations made by M of the given subtree, and the q_i indicates what kind of translation v_i is. We shall also need the *rule-sequence* of the derivation at d. It is the sequence $\langle r_1, \ldots, r_n\rangle$, where r_i is the first rule applied in the derivation $q_i(t_2) \overset{*}{\Rightarrow} v_i$. A more precise formal definition follows.

(3.1.8) DEFINITION. Let $M = (Q, \Sigma, \Delta, q_0, R)$ be in yT, let α: $q(t) \overset{*}{\Rightarrow} w$ be a derivation of M with $q \in Q$, $t \in T_\Sigma$, and $w \in \Delta^*$. Let d be a node of t. The *state-sequence of* α *at* d is a sequence $\langle q_1, \ldots, q_m\rangle$ of states of M ($m \geqslant 0$) defined recursively as follows.

(i) If $t = \sigma \in \Sigma_0$ and d is the unique node labeled by σ, then the state-sequence of α at d is $\langle q\rangle$.

(ii) Assume $t = \sigma(t_1 \cdots t_k)$, $\sigma \in \Sigma_k$, $k \geqslant 1$. If d is the root of t then the state-sequence of α at d is $\langle q\rangle$. Now let d be a node of t_i for some i, $1 \leqslant i \leqslant k$. Consider the first step of the derivation α: $q(\sigma(t_1 \cdots t_k)) \Rightarrow r[t_1, \ldots, t_k] \overset{*}{\Rightarrow} w$, where $q(\sigma(x_1 \cdots x_k)) \to r$ is in R. If x_i does not occur in r, then the state-sequence of α at d is empty. Assume now that x_i occurs in r and let $r = u_1q_1(x_i)\, u_2q_2(x_i) \cdots u_nq_n(x_i)u_{n+1}$ with $n \geqslant 1$ and $u_j \in (\Delta \cup Q(X_k - \{x_i\}))^*$. It should be clear that there are unique derivations α_j: $q_j(t_i) \overset{*}{\Rightarrow} v_j$ ($1 \leqslant j \leqslant n$) which are a "part" of the derivation $r[t_1, \ldots, t_k] \overset{*}{\Rightarrow} w$. Let s_j be the state-sequence of α_j at d. Then their "concatenation" $s_1s_2 \cdots s_n$ is the state-sequence of α at d.

If $\langle q_1, \ldots, q_m\rangle$ is the state-sequence of α at node d, then $\{q_1, \ldots, q_m\}$ is called the *state-set* of α at d.

The notion of the *rule-sequence* of the derivation α (above) at node d is defined similarly. In fact, in case (i) above, and in the "root-case" of (ii), the rule-sequence consists of the first rule applied in the derivation. If the state-sequence of α at d is empty then so is the rule-sequence. In the nontrivial case let r_j be the rule-sequence of α_j at d. Then, as in the case of state-sequence, the rule-sequence of α at d is $r_1r_2 \cdots r_n$. ∎

In this definition we have used the obvious fact that if $w_1q_1(t_1)w_2 \cdots q_n(t_n)w_{n+1} \overset{*}{\Rightarrow} w$ (for some w, $w_j \in \Delta^*$, $q_j \in Q$ and $t_j \in T_\Sigma$) then there are unique derivations $q_j(t_j) \overset{*}{\Rightarrow} v_j$

such that $w = w_1v_1w_2 \cdots v_nw_{n+1}$ and the latter derivations are "part" of the original one in the sense that some reordering of them is precisely the original derivation.

We note that if the rule-sequence of a derivation α at node d labeled σ is known to be $\langle r_1, r_2, \ldots, r_n \rangle$ with $r_j = q_j(\sigma(x_1 \cdots x_k)) \to u_j$, then the state-sequence of the ith son of d in α is $\langle p_1, \ldots, p_m \rangle$ if $p_1(x_i), p_2(x_i), \ldots, p_m(x_i)$ occur in this order in $u_1u_2 \cdots u_n$, and there are no other occurrences of x_i. The possible rule-sequences at the ith son are then all $\langle r'_1, r'_2, \ldots, r'_m \rangle$ such that r'_j is a rule for p_j. It should be realized that in special cases the rule-sequences (and state-sequences) can be computed by a top-down finite tree automaton.

We are now ready to define the restrictions on yT transducers.

(3.1.9) DEFINITION. Let $M = (Q, \Sigma, \Delta, q_0, R)$ be in yT, and let $k \geqslant 1$ be an integer. A derivation α: $q_0(t) \overset{*}{\Rightarrow} w$ has *state-bound* k if, for each node d of t, the cardinality of the state-set of α at d is at most k; α has *copying-bound* k if, for each node d of t, the length of its state-sequence at d is at most k.

Let L be a tree language. (M, L) has state-bound k {copying-bound k} if for each $w \in M(L)$ there exist $t \in L$ and a derivation $q_0(t) \overset{*}{\Rightarrow} w$ with state-bound k {copying-bound k}. Note that (M, L) has state-bound $\#(Q)$ trivially. (M, L) is *finite copying* if it has copying-bound k for some k. The same terminology holds for M if it is true of (M, T_Σ).

M is *k-metalinear* if (1) q_0 does not appear in the right-hand sides of rules of R and (2) for each rule $q(\sigma(x_1 \cdots x_k)) \to u$ in R, if $q = q_0$ then, for each i, the number of occurrences of x_i in u is at most k, and if $q \neq q_0$ then this number is 0 or 1. M is *metalinear* if it is k-metalinear for some k. (M, L) is (k-) metalinear if M is. ▮

Finite copying and metalinearity are denoted by subscripts "fc" and "ml" respectively. Bounds are indicated by subscripts "(k)." Thus the classes of state-bound k, copying-bound k, finite copying, k-metalinear, and metalinear top-down Ω-tree transformation languages are denoted by $yT_{(k)}(\Omega)$, $yT_{\mathrm{fc}(k)}(\Omega)$, $yT_{\mathrm{fc}}(\Omega)$, $yT_{\mathrm{ml}(k)}(\Omega)$, and $yT_{\mathrm{ml}}(\Omega)$, respectively. Observe that $yT_{\mathrm{ml}(k)}(\Omega) \subseteq yT_{\mathrm{fc}(k)}(\Omega) \subseteq yT_{(k)}(\Omega)$. Also note that $yT_{\mathrm{fc}(1)}(\Omega) = yT_{\mathrm{ml}(1)}(\Omega)$ is the set of images of Ω-tree languages under linear yT transducers. Finally it can easily be shown that if Ω is closed under finite-state relabeling then $yDT_{(1)}(\Omega) = y\mathrm{HOM}(\Omega)$, where HOM is the class of tree homomorphisms [18].

The notion of finite copying was first investigated in [4]; yDT_{fc} corresponds to those GSDT for which S_1 is in $\Gamma^{(0)}$, in the terminology of [4]. Metalinear top-down tree transducers are an obvious extension of linear top-down tree transducers; they are the simplest kind of copying device of this type (compare with the metalinear extension of linear context-free grammars [52]). The notion of state-bound is (as in the ET0L case) most natural for deterministic transducers: state-bound k means that at most k different translations are made of each subtree of the input tree. Note also, that in the deterministic case any node in the input tree has a unique state-set, state-sequence, and rule-sequence associated with it.

(3.1.10) EXAMPLES. (i) Let G be a context-free grammar and let $L = \{\$(t) \mid t \text{ is a derivation tree of } G\}$; then $L \in \mathrm{REC}$. Consider the top-down yT transducer $M = (\{q_0, q_1\}, \Sigma, \Delta, q_0, R)$, where Σ is the ranked alphabet from which the derivation trees of G are

constructed, together with the new symbol \$ of rank 1, Δ is the set of terminals of G together with the new symbol #, and R consists of the rules

$$\begin{aligned} &q_0(\$(x)) \to q_1(x) \# q_1(x), \\ &q_1(\sigma(x_1 \cdots x_n)) \to q_1(x_1) \cdots q_1(x_n) \qquad \text{for all } \sigma \neq \$, \\ &q_1(\sigma) \to \sigma \qquad \text{for all } \sigma \in \Sigma_0 . \end{aligned}$$

Then M is a deterministic, 2-metalinear yT transducer, and $M(L) = \{w \# w \mid w \in L(G)\}$. By identifying q_0 and q_1 we obtain $M' \in yDT_{(1)}$ with $M'(L) = M(L)$.

(ii) Consider the yT transducer $M = (\{q_0, q_1, q_2\}, \{\sigma, \tau, \delta\}, \{a, b, c, d\}, q_0, R)$ such that σ, τ and δ have ranks 2, 1, and 0, respectively; R consists of the rules

$$\begin{aligned} &q_0(\sigma(xy)) \to q_1(x)\, q_0(y)\, q_2(x), \qquad q_0(\delta) \to \lambda, \\ &q_1(\tau(x)) \to a q_1(x) b, \qquad q_1(\delta) \to \lambda, \\ &q_2(\tau(x)) \to c q_2(x) d, \qquad q_2(\delta) \to \lambda. \end{aligned}$$

M translates a tree of the form $\sigma(\tau^{n_1}(\delta)\, \sigma(\tau^{n_2}(\delta) \cdots \sigma(\tau^{n_k}(\delta)\, \delta) \cdots))$ into the string $a^{n_1}b^{n_1}a^{n_2}b^{n_2} \cdots a^{n_k}b^{n_k}c^{n_k}d^{n_k} \cdots c^{n_2}d^{n_2}c^{n_1}d^{n_1}$. The state-sequence at each node labeled σ and at the rightmost δ is $\langle q_0 \rangle$; at each node labeled τ and all other nodes labeled δ the state-sequence is $\langle q_1, q_2 \rangle$. Thus, if L consists of the above trees, then (M, L) has copying-bound 2 and $M(L) \in yDT_{\mathrm{tc}(2)}(\mathrm{REC})$.

3.2. *Results*

We show first that recognizable input tree languages are not needed, meaning that in the case $\mathfrak{L} = \mathrm{REC}$ we may consider ranges of transducers in yT, rather than $yT(\mathrm{REC})$. The reason that we keep considering recognizable input languages, apart from the motivation of syntax-directed translation, is that their presence facilitates proofs.

(3.2.1) THEOREM. *For each top-down tree transformation system (M, L) with $L \in$ REC, there exists a top-down yT transducer M' such that $M(L) = M'(T_\Sigma)$, where Σ is the input alphabet of M'. The construction involved preserves determinism, state-bound, copying-bound, metalinear bound, and monadicness of the input alphabet.*

Proof. We note first that it may be assumed that L is the domain of a deterministic top-down finite tree automaton. Indeed, by standard arguments it can be shown that each recognizable tree language is the projection of the domain of such an automaton; the projection can then be incorporated into the transducer M while preserving all the mentioned properties.

Let $M = (Q, \Sigma, \Delta, q_0, R)$, and let $A = (P, \Sigma, \Sigma, p_0, R_A)$ be a deterministic top-down finite tree automaton with domain L. We may assume that for each $p \in P$ there exists $t \in T_\Sigma$ such that $p(t) \overset{*}{\Rightarrow} t$ by A (if not, we just delete from R_A all rules which involve p). We now construct $M' = (Q \times P, \Sigma, \Delta, \langle q_0, p_0 \rangle, R')$ as follows:

(i) if $q(\sigma(x_1 \cdots x_k)) \to w_1 q_1(x_{i_1}) \cdots w_n q_n(x_{i_n}) w_{n+1}$ is in R and $p(\sigma(x_1 \cdots x_k)) \to \sigma(p_1(x_1) \cdots p_k(x_k))$ is in R_A, then $\langle q, p\rangle(\sigma(x_1 \cdots x_k)) \to w_1\langle q_1, p_{i_1}\rangle(x_{i_1}) \cdots w_n\langle q_n, p_{i_n}\rangle \times (x_{i_n})w_{n+1}$ is in R'.

(ii) if $q(\sigma) \to w$ is in R and $p(\sigma) \to \sigma$ is in R_A, then $\langle q, p\rangle(\sigma) \to w$ is in R'.

M' simulates M and simultaneously checks whether the input tree is in L. It may happen that M does not visit a subtree t of the input tree. In such a case the checking of t cannot be done by M'. However, we know that A arrives at the root of t in a certain state p. Replacing t by t' such that $p(t') \overset{*}{\Rightarrow} t'$ in A, gives an input tree in L with the same computation as that of M on the original input tree. It is left to the reader to prove formally that $M'(T_\Sigma) = M(L)$.

Because A is deterministic (nondeleting and noncopying), determinism and all the mentioned bounds are preserved. The latter follows from the observation that if $\langle q_1, \ldots, q_n\rangle$ is the state-sequence of some derivation of M at a certain node, and if A arrives at this node in state p, then the corresponding state-sequence of M' is $\langle\langle q_1, p\rangle, \ldots, \langle q_n, p\rangle\rangle$. ∎

We show now that $\mathfrak{L}$-controlled ETOL systems are the "monadic case" of the top-down $\mathfrak{L}$-tree transformation systems (as already shown in [6, 20] for the uncontrolled and unbounded classes).

(3.2.2) THEOREM. *If $\mathfrak{L}$ is a class of languages closed under sequential machine mappings, then $yT(\mathfrak{L}) = \text{ETOL}(\mathfrak{L})$. Also, ETOL = ETOL(REG) = yT(REG). The constructions preserve determinism, state-bound, copying-bound (= index), and metalinear-bound.*

Proof. To be precise, the theorem should say that $yT(\mathfrak{L}') = \text{ETOL}(\mathfrak{L}') = \text{ETOL}(\mathfrak{L})$, where $\mathfrak{L}' = \{L - \{\lambda\} \mid L \in \mathfrak{L}\}$. In what follows we assume that the languages in $\mathfrak{L}$ are λ-free. To show $yT(\mathfrak{L}) \subseteq \text{ETOL}(\mathfrak{L})$ let $L \in \mathfrak{L}$ be a language over the monadic alphabet Σ and let $M = (Q, \Sigma, \Delta, q_0, R)$ be in yT. Since $\mathfrak{L}$ is closed under sequential machine mappings we may assume that $\Sigma_0 \cap \Sigma_1 = \varnothing$. Construct an $\mathfrak{L}$-controlled ETOL system (M', L) with $M' = (Q, \Sigma, \Delta, q_0, R')$ with R' defined as follows. If $q(\sigma(x)) \to r$ or $q(\sigma) \to r$ are in R then $q(\sigma x) \to r$ is in R'. Then $M'(L) = M(L)$.

For the special case $\mathfrak{L} = \text{REG}$, it follows from Theorem 3.2.1 that we may assume that $L = T_\Sigma$ (recall that for monadic $L, L \in \text{REG}$ iff $L \in \text{REC}$). Hence $M(L) = M'(\Sigma_1^*\Sigma_0)$. It is easy to see that $M'(\Sigma_1^*\Sigma_0) = M'(\Sigma^*) = L(M')$, and hence $M(L) \in \text{ETOL}$. This shows that $yT(\text{REG}) \subseteq \text{ETOL}$.

To show $\text{ETOL}(\mathfrak{L}) \subseteq yT(\mathfrak{L})$ let (M, L) be an $\mathfrak{L}$-controlled ETOL system with $M = (Q, \Sigma, \Delta, q_0, R)$. Construct $M' = (Q, \Sigma, \Delta, q_0, R')$ in yT such that $\Sigma_0 = \Sigma_1 = \Sigma$; if $q(\sigma x) \to r$ is in R then $q(\sigma(x)) \to r$ is in R', and if $q(\sigma x) \to w$ is in R with $w \in \Delta^*$ then $q(\sigma) \to w$ is in R'. Then $M'(L) = M(L)$.

It should be clear that both constructions preserve all the properties mentioned in the theorem. In particular, the sequences of states in the sentential forms of a parallel derivation in an ETOL system are precisely the state-sequences at the corresponding nodes of the input. Therefore, state-bound and copying-bound (= index) are preserved. ∎

We note that Theorem 3.2.1 is a generalization of the well-known fact [7, 29] that

regular control does not increase the generative power of ET0L systems (in the monadic case the recognizable input languages correspond to regular control).

Another observation is that yDT(REG), i.e., EDT0L by Theorem 3.2.2, is equal to the class of ranges of those GSDT whose underlying CF grammar is linear. In fact, the derivation trees of linear CF grammars (in the way derivation trees are defined in [4]) are monadic. Thus ET0L may be viewed as the class of languages which may be obtained from linear CF grammars by nondeterministic GSDT.

The next theorem shows that the language generating power of finite copying and metalinear transducers is not affected by nondeterminism. This is a generalization of Lemma 2 of [50]. Informally, the idea is that the nondeterminism of the tree transducer can be transferred to the input tree language by means of a (top-down) finite-state relabeling; the reason that this can be done is (precisely) the property of finite copying.

(3.2.3) LEMMA. *Let $\mathfrak{L}$ be a class of tree languages closed under finite-state relabelings. Then for every $k \geqslant 1$ $yT_{\mathrm{fc}(k)}(\mathfrak{L}) = yDT_{\mathrm{fc}(k)}(\mathfrak{L})$ and $yT_{\mathrm{ml}(k)}(\mathfrak{L}) = yDT_{\mathrm{ml}(k)}(\mathfrak{L})$.*

Proof. Follows from Theorem 4.9 and (to avoid repetition) is not presented here. We only remark here that the proof for the monadic case (Lemma 2 of [50]) can easily be generalized. ▮

We note here that it may even be assumed that *all* derivations of a bounded (i.e., either finite copying or metalinear) deterministic yT transducer M are bounded. In fact, it is easy to modify M so that its state keeps track of M's state-sequence at the node of the input tree at which it arrives (see remarks following Definition 3.1.8) and rejects the input tree if there is a state-sequence longer than the given bound. In the monadic case this corresponds to Theorem 2 of [50].

From Lemma 3.2.3 it follows that under appropriate conditions on $\mathfrak{L}$, $yT_{\mathrm{ml}(k)}(\mathfrak{L}) \subseteq yT_{\mathrm{fc}(k)}(\mathfrak{L}) \subseteq yDT_{(k)}(\mathfrak{L}) \subseteq yT_{(k)}(\mathfrak{L})$. The diagram of Fig. 3 shows the classes of languages discussed so far (written without the bound) for $\mathfrak{L}$ = REC and $\mathfrak{L}$ = REG. The same diagram holds for all $k \geqslant 2$ (just add subscript (k)). It is left to the reader to imagine the

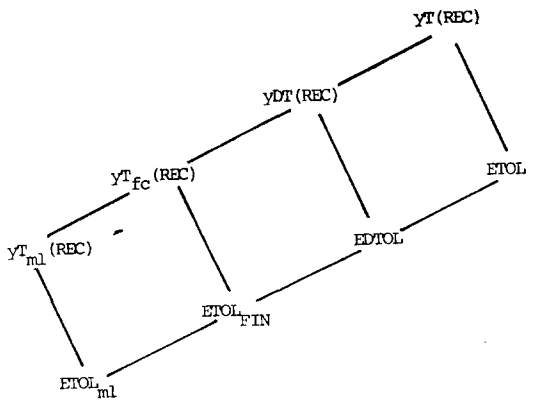

FIG. 3. Classes of tree transformation languages.

full "3-dimensional" diagram in which the k-bounded classes appear on the kth "level" with their "limit classes" above.

In [4] Aho and Ullman investigate a hierarchy of top-down tree transformation systems between $yT_{\text{fc}}(\text{REC})$ and $yDT(\text{REC})$. They consider the length of the state-sequence at a node d as a function of the distance, n, of d to the root of the input tree, and they show that this function is either a polynomial in n, i.e., a function of the form $k \cdot n^m$ (for some $k \geqslant 1$, $m \geqslant 0$), or an exponential, that is k^n (for some $k \geqslant 2$). They conclude that if, for a tree transformation system (M, L), there exists a linear relationship between the number of nodes of the input tree and the length of the output string, then there is an equivalent tree transformation system (M', L') such that M' is finite copying. Since their definitions and proofs preserve monadicness, the same remarks can be made concerning ET0L_{FIN} and EDT0L. In fact there is a close connection between the above hierarchy and the recently investigated ET0L systems "with rank" [16].

We shall now busy ourselves with proving the correctness of the 3-dimensional version of the diagram of Fig. 3.

We start by showing that all the discussed bounds give rise to proper hierarchies in all the classes appearing in the diagram. This will be made possible by an appropriate refinement of the intercalation lemma for $yDT(\text{REC})$ of [44], cf. also Lemmas 4.2 and 5.6 of [4].

(3.2.4) THEOREM. *Let $k \geqslant 1$ be an integer. For each $L \in yDT_{(k)}(\text{REC})$ there exists an integer p such that for all $z \in L$ with $|z| \geqslant p$ there are strings $z_1, \ldots, z_s$ and $x_1, \ldots, x_{s+1}$ $(s \geqslant 1)$ such that $z = x_1 z_1 x_2 z_2 \cdots x_s z_s x_{s+1}$ and the following conditions are satisfied.*

(i) $0 < |z_i| \leqslant p$ *for all* i, $1 \leqslant i \leqslant s$;

(ii) $\#(\{z_i \mid 1 \leqslant i \leqslant s\}) \leqslant k$;

(iii) *for every integer N there exist strings $v_1, v_2, \ldots, v_s$ such that*

for $v = x_1 v_1 x_2 v_2 \cdots x_s v_s x_{s+1}$,

(a) $v \in L$,

(b) $|v| \geqslant N$,

(c) $\text{alph}(v_i) = \text{alph}(z_i)$ *for* $1 \leqslant i \leqslant s$,

(d) *if* $z_i = z_j$ *then* $v_i = v_j$ *for* $1 \leqslant i, j \leqslant s$;

(iv) *if moreover $L \in yDT_{\text{fc}(k)}(\text{REC})$, then $s \leqslant k$.*

Proof. The proof is a slight modification of the proof of the intercalation lemma of [44]. The basic idea is the following. If the input tree is sufficiently large, then it has two nodes (on one path) with the same state-set and the same label. Consequently the part of the input tree between these two nodes, together with the associated computations of the transducer, can be repeated indefinitely. In order to be sure that the length of the output string will then grow, we need several assumptions about the transducer and its input language (see [65]).

Let $L = M(L')$ with $L' \in \text{REC}$ and $M \in yDT_{(k)}$. First we may assume (modulo λ) that M is λ-free. In fact, we can put the information whether $q(t) \overset{*}{\Rightarrow} \lambda$ by M, at the father

node of the root of each subtree t of the input tree (for each state q). Since for $N \in yT$ the tree language $N^{-1}(\lambda)$ is recognizable (cf. [43, 19]), this results in a new recognizable input language. Using the extra information we can remove $q(x_i)$ from the right-hand side of a rule of M whenever q translates the ith subtree into λ. Now (the new) M just skips those subtrees which would otherwise be translated into λ. The construction clearly preserves determinism, state-bound and copying-bound.

Second, we may assume that M visits each node of an input tree in L', i.e., that the state-set at each node is nonempty. In fact, applying a top-down finite-state relabeling, the state-set at each node may be added to the label of the node (again, see the remarks following Definition 3.1.8). Then a linear deterministic bottom-up tree transducer [18] can be used to delete all nodes with empty state-set from the trees of L' (and to put information at the father node which of its sons was deleted). It is easy to adapt M to this new (recognizable) input tree language. Clearly (the new) M visits all nodes of the (new) input trees. The construction preserves determinism, state-bound, copying-bound, and λ-freedom.

Third, we may assume that there are no "useless monadic nodes" in the input trees (a node d of a tree $t \in L'$, labeled by a symbol σ of rank 1, is called useless if for each state q in the state-set of d the applied rule is $q(\sigma(x)) \to q'(x)$ for some state q').

In the input tree we replace each sequence $d_1, d_2, \ldots, d_n, d_{n+1}$ of nodes in which $d_1, d_2, \ldots, d_n$ are useless monadic nodes but d_{n+1} is not (and d_{i+1} is the son of d_i), by the single node d_{n+1} of which the label τ is replaced by $\langle \tau, f \rangle$, where f is the state-transformation function obtained in an obvious way from the rules $q(\sigma(x)) \to q'(x)$ of the useless nodes $d_1, \ldots, d_n$. This change results in a new recognizable input tree language (as it can be realized by a linear top-down tree transducer). Now M is changed such that it has a rule $q(\langle \tau, f \rangle(\cdots)) \to r$ if it originally had a rule $f(q)(\tau(\cdots)) \to r$. Note that d_{n+1} has the same state-sequence as d_1 had. The construction preserves determinism, state-bound, copying-bound, λ-freedom, and the nonemptiness of state-sets.

Finally we may assume that *all* computations of M are k-bounded (cf. the remark following Lemma 3.2.3) and that $L' = T_\Sigma$, where Σ is the input alphabet of the (final) M, the latter following from Theorem 3.2.1 the proof of which clearly preserves all the mentioned properties.

Let $M = (Q, \Sigma, \Delta, q_0, R)$, let $h = \#(\Sigma) \cdot 2^{k^2}$ (or more precisely, because of multiple ranks, $h = \sum_{n=0}^{\infty} \#(\Sigma_n) \cdot 2^{k^2}$) and let p be an integer larger than the length of each $w \in \Delta^*$ such that $q(t) \overset{*}{\Rightarrow} w$ for some $q \in Q$ and $t \in T_\Sigma$ of height $\leqslant h$. We show now that the theorem holds with this p. Consider $z \in L$ with $|z| \geqslant p$. Then $q_0(t) \overset{*}{\Rightarrow} z$ for some t of height $> h$. Consider a subtree t_0 of t of height h and a longest path π from the root d_0 of t_0 to a leaf. Let the state-set at d_0 be $\{q_1, \ldots, q_m\}$, $m \leqslant k$. For each node d on the path π we can consider, first its label, and second the sequence $\langle Q_1, \ldots, Q_m \rangle$ of subsets of Q such that $\cup \{Q_i \mid 1 \leqslant i \leqslant m\}$ is the state-set at d, and Q_i is the state-set at d obtained from the derivation that starts with $q_i(t_0)$.

Since $h = \#(\Sigma)(2^k)^k$, there exist two nodes d_1 and d_2 on π with the same label (of the same rank) and the same sequence $\langle Q_1, \ldots, Q_m \rangle$. Let t_2 be the subtree of t with root d_2 and let $t_1 \in T_\Sigma[\{x\}]$ have exactly one occurrence of x such that $t_1[t_2]$ is the subtree of t rooted at d_1. The situation is displayed in Fig. 4.

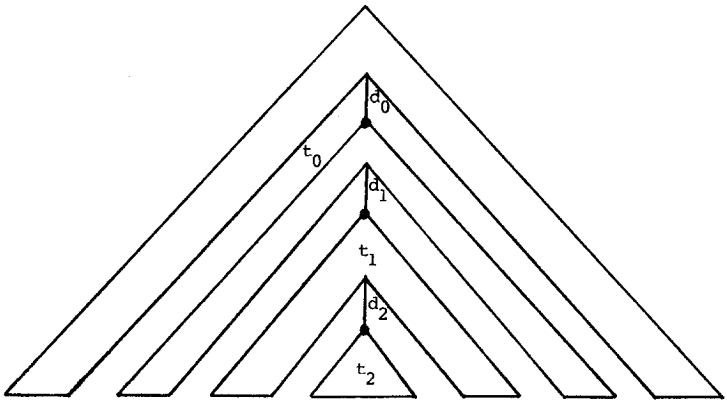

FIG. 4. Tree t with subtrees t_0, $t_1[t_2]$, and t_2.

It is now possible to repeat t_1 in the input tree an arbitrary number of times (i.e., $t_1[t_2]$ is replaced by $t_1[t_1[t_2]]$, $t_1[t_1[t_1[t_2]]]$, etc.). Let the state-sequence at d_0 be $\langle p_1, p_2, \ldots, p_s \rangle$. Then $z = x_1 z_1 x_2 z_2 \cdots z_s x_{s+1}$ with $p_i(t_0) \overset{*}{\Rightarrow} z_i$. Since t_0 has height h, $|z_i| \leqslant p$; since M is λ-free, $|z_i| > 0$; since M visits all nodes, $s \geqslant 1$; and since M is deterministic, $\#(\{z_i \mid 1 \leqslant i \leqslant s\}) = m \leqslant k$. If M has copying-bound k, then $s \leqslant k$. Repetition of the tree t_1 results in a derivation of M in which $z_1, \ldots, z_s$ are changed, but such that the alphs of the new pieces are precisely identical to the alphs of the corresponding z_i's. This is due to the fact that the state-sets at d_1 and d_2 are the same for each derivation $p_i(t_0) \overset{*}{\Rightarrow} z_i$. Also, if $z_i = z_j$ then the corresponding "pumped" pieces are equal. It remains to show that repetition of t_1 results in arbitrary long output strings. Let $Q_0 = \bigcup \{Q_i \mid 1 \leqslant i \leqslant m\}$ be the state-set at d_1 and d_2, and let $Q_0 = \{r_1, \ldots, r_n\}$. Consider all subderivations $r_i(t_1) \overset{*}{\Rightarrow} w_i$ of $q_0(t) \overset{*}{\Rightarrow} z$, with $w_i \in (\Delta \cup Q(x))^+$. If at least one w_i contains an element of Δ, then clearly each repetition of t_1 produces at least one more output symbol. Now suppose that all w_i are in $Q(x)^+$. It at least one w_i contains more than one element of $Q(x)$, then repetition of t_1 will lead to arbitrary long sequences in $Q(x)^+$ at the level of d_2 and hence to arbitrary long output strings. Now suppose that all w_i are in $Q(x)$. Then (since M visits all nodes of t) t_1 is a monadic tree. This, however, implies that d_1 is a useless monadic node, which contradicts our assumption on M. ∎

We now use this intercalation theorem to prove the properness of the hierarchies of Fig. 3.

(3.2.5) THEOREM. *All classes of the diagram of Fig.* 3 *are proper hierarchies with respect to state-bound or copying-bound (as appropriate). In particular, the language* $L_k = \{a_1^n a_2^n \cdots a_{2k}^n \mid n \geqslant 0\}$ *is in* $\mathrm{ETOL}_{\mathrm{ml}(k)}$ *but not in* $yT_{(k-1)}(\mathrm{REC})$, *for* $k \geqslant 2$.

Proof. To see that $L_k \in \mathrm{ETOL}_{\mathrm{ml}(k)}$ consider the EDTOL system with rules

$$q_0(ax) \to q_1(x)\, q_2(x) \cdots q_k(x),$$
$$q_i(bx) \to a_{2i-1} q_i(x) a_{2i} \qquad \text{for } 1 \leqslant i \leqslant k,$$
$$q_i(cx) \to \lambda \qquad \text{for } 1 \leqslant i \leqslant k.$$

Clearly the system is k-metalinear and it generates L_k. Assume now that $L_k \in yT_{(k-1)}(\mathrm{REC})$. In [24] it is shown that if $L \in yT(\mathrm{REC})$ and L has property (P2), defined there, then $L \in yDT(\mathrm{REC})$. It can easily be checked that the construction presented in [24] preserves the state-bound; and since L_k has property (P2), it follows that $L_k \in yDT_{(k-1)}(\mathrm{REC})$. Now Theorem 3.2.4 can be applied. Let $z = a_1{}^p a_2{}^p \cdots a_{2k}^p \in L_k$. Then $z = x_1 z_1 x_2 z_2 \cdots x_s z_s x_{s+1}$ with $|z_i| \leqslant p$. Since there are at most $k-1$ different z_i and $|z_i| \leqslant p$, it follows that at most $2k-2$ different a_j occur in $z_1, \ldots, z_s$. Hence there is some a_j that does not occur in any of the z_i. Thus, when "intercalating," the $a_j{}^p$ stays as it is and no new a_j is introduced (by the fact that the alphs remain the same). It follows that the number of a_j in the string remains p, which is a contradiction. ∎

Note that the theorem holds also for $\{a_1{}^n \cdots a_{2k-1}^n \mid n \geqslant 0\}$ instead of L_k. Note also that if we take ab^*c as the input language to the EDTOL system of the above proof, then q_0 can be identified with, say q_1. Consequently, L_k can be defined by a top-down tree transformation system with exactly k states. If we call (M, L) with $M = (Q, \Sigma, \Delta, q_0, R)$ and $\#(Q) \leqslant k$ an "$s(k)$ system," then Theorem 3.2.5 shows that $yT_{s(k)}(\mathrm{REC})$ and $yDT_{s(k)}(\mathrm{REC})$ are proper hierarchies (as are $\mathrm{ET0L}_{s(k)}$ and $\mathrm{EDT0L}_{s(k)}$ if we allow regular control; alternatively we could allow all $q_1(v)\, q_2(v) \cdots q_k(v)$ as initial sentential forms).

The hierarchy theorem for $\mathrm{ET0L}_{\mathrm{FIN}}$ was proved in [50] using the same counterexamples (see also [62]). It was also proved in [35] using a machine approach, see also the next section. For $\mathrm{ET0L}_{\mathrm{ml}}$ (characterized differently) it was shown in [30]; see Section 3.

We can use Theorem 3.2.4 to show that the language $\{(a^n b a^n c)^m \mid n, m \geqslant 1\}$ is not in $yT_{\mathrm{fc}}(\mathrm{REC})$; compare similar statements concerning $\mathrm{ET0L}_{\mathrm{FIN}}$ in [50, 35, 17]. In the next theorem we give another way of obtaining languages not in $yT_{\mathrm{fc}}(\mathrm{REC})$; for the monadic case of $\mathrm{ET0L}_{\mathrm{FIN}}$ see [50, 35, 45, 41, 30].

Let $\mathrm{Par}(L)$ denote the set of Parikh vectors of strings in L, and $\mathrm{Par}(\mathfrak{L}) = \{\mathrm{Par}(L) \mid L \in \mathfrak{L}\}$ for a class $\mathfrak{L}$ of languages.

(3.2.6) THEOREM. *Let $\mathfrak{L}$ be a class of tree languages. Then* $\mathrm{Par}(yT_{\mathrm{fc}}(\mathfrak{L})) = \mathrm{Par}(yT_{\mathrm{fc}(1)}(\mathfrak{L}))$.

Proof. Let (M, L) be a top-down $\mathfrak{L}$-tree transformation system with copying-bound $k > 1$ and let $M = (Q, \Sigma, \Delta, q_0, R)$. By rearranging the right-hand sides of rules of R we can put the (at most k) translations of each subtree together, and thus obtain a linear transducer which simulates M on L modulo a permutation of the output string.

Formally we construct $M' = (Q', \Sigma, \Delta, \langle q_0 \rangle, R')$ such that $Q' = \{\langle q_1, \ldots, q_n \rangle \mid 1 \leqslant n \leqslant k$ and $q_i \in Q\}$ and R' is obtained as follows. If, for each i, $1 \leqslant i \leqslant n$, $q_i(\sigma(x_1 \cdots x_m)) \to r_i$ is in R, then the rule $\langle q_1, \ldots, q_n \rangle(\sigma(x_1 \cdots x_m)) \to w s_1(x_1)\, s_2(x_2) \cdots s_m(x_m)$ is in R', where $w \in \Delta^*$ is the string obtained from $r_1 r_2 \cdots r_n$ by erasing all elements of $Q(X)$, and s_i is the sequence of all occurrences of states q in $r_1 r_2 \cdots r_n$ that occur in the context $q(x_i)$, it being understood that $s_i(x_i)$ is actually λ if this sequence is empty. It should be clear from the construction that the state of M' at a node of the input tree is the state-sequence of M at that node, and that the output string produced by M' is a permutation of that M. Hence $\mathrm{Par}(M'(L)) = \mathrm{Par}(M(L))$, and since M' is linear, $M'(L) \in yT_{\mathrm{fc}(1)}(\mathfrak{L})$. ∎

(3.2.7) COROLLARY. *If $\mathfrak{L}$ is a class of tree languages closed under linear top-down tree transducers, then* $\mathrm{Par}(yT_{\mathrm{fc}}(\mathfrak{L})) = \mathrm{Par}(y\mathfrak{L})$. *Thus in particular,* $\mathrm{Par}(yT_{\mathrm{fc}}(\mathrm{REC})) = \mathrm{Par}(\mathrm{REG})$.

Proof. Follows from Theorem 3.2.6, from the fact that REC is closed under linear tree transducers and from the known result $\mathrm{Par}(y\ \mathrm{REC}) = \mathrm{Par}(\mathrm{CF}) = \mathrm{Par}(\mathrm{REG})$. ∎

(3.2.8) COROLLARY. $\{a^{2^n} \mid n \geqslant 0\} \in \mathrm{EDT0L}_{(1)} - yT_{\mathrm{fc}}(\mathrm{REC})$.

It follows from Corollary 3.2.7 that $\mathrm{Par}(\mathrm{ET0L}_{\mathrm{FIN}}) = \mathrm{Par}(\mathrm{REG})$, cf. [50]. A similar result for $\mathfrak{L}$-controlled $\mathrm{ET0L}_{\mathrm{FIN}}$ systems follows from Theorem 3.2.6 (see [35]).

(3.2.9) COROLLARY. *If $\mathfrak{L}$ is a class of languages closed under gsm mappings, then* $\mathrm{Par}(\mathrm{ET0L}_{\mathrm{FIN}}(\mathfrak{L})) = \mathrm{Par}(\mathfrak{L})$.

Proof. It follows from Theorems 3.2.6 and 3.2.2 that $\mathrm{Par}(\mathrm{ET0L}_{\mathrm{FIN}}(\mathfrak{L})) = \mathrm{Par}(yT_{\mathrm{fc}}(\mathfrak{L})) = \mathrm{Par}(yT_{\mathrm{fc}(1)}(\mathfrak{L})) = \mathrm{Par}(\mathrm{ET0L}_{\mathrm{FIN}(1)}(\mathfrak{L}))$. The proof of Theorem 3.2.6 even shows that all rules of the $\mathrm{ET0L}_{\mathrm{FIN}(1)}$ system are of the form $q_1(\sigma x) \to wq_2(x)$ or $q_1(\sigma x) \to w$. Since this is clearly a gsm mapping and $\mathfrak{L}$ is closed under gsm mappings, $\mathrm{Par}(\mathrm{ET0L}_{\mathrm{FIN}(1)}(\mathfrak{L})) = \mathrm{Par}(\mathfrak{L})$. ∎

The next theorem provides a method to obtain languages not in $yT_{\mathrm{ml}}(\mathrm{REC})$. It shows that $yT_{\mathrm{ml}}(\mathrm{REC})$ behaves badly with respect to Kleene closure.

(3.2.10) THEOREM. *If $\mathfrak{L}$ is a class of tree languages such that $yT_{\mathrm{ml}(1)}(\mathfrak{L})$ is a full semi-*AFL, *and $(L\#)^* \in yT_{\mathrm{ml}}(\mathfrak{L})$, then $L \in yT_{\mathrm{ml}(1)}(\mathfrak{L})$. If $\mathfrak{L}$ is a class of monadic tree languages and $(L\#)^* \in yT_{\mathrm{ml}}(\mathfrak{L})$, then L is in the smallest full semi-*AFL *containing $\mathfrak{L}$.*

Proof. Let $(L\#)^* = M(L')$ for $M = (Q, \Sigma, \Delta, q_0, R)$ in $yT_{\mathrm{ml}(k)}$ and $L' \in \mathfrak{L}$. We distinguish two cases (of which the second actually never happens).

Case 1. For each $y \in L$ there exists a derivation of the form $q_0(\sigma(t_1 \cdots t_n)) \Rightarrow u_1q(t_i)u_2 \overset{*}{\Rightarrow} u_1'w_1 \# y \# w_2u_2' \in \Delta$ such $q(t_i) \overset{*}{\Rightarrow} w_1 \# y \# w_2$, and $u_1 \overset{*}{\Rightarrow} u_1'$, $u_2 \overset{*}{\Rightarrow} u_2'$, $\sigma(t_1 \cdots t_n) \in L'$, $q \in Q$, $1 \leqslant i \leqslant n$, u_1 and u_2 in $(\Delta \cup Q(T_\Sigma))^*$.

For each $\sigma \in \Sigma_n$ $(n \geqslant 1)$, $q \in Q$ and $x_i \in X_n$ such that $q(x_i)$ occurs in the right-hand side of a rule with left-hand side $q_0(\sigma(x_1 \cdots x_n))$, let $N(\sigma, q, x_i) = \{w \in \Delta^* \mid q(t_i) \overset{*}{\Rightarrow} w$ for some $\sigma(t_1 \cdots t_n) \in L'\}$. The language $N(\sigma, q, x_i)$ can be produced by the linear top-down $\mathfrak{L}$-tree transformation system (M', L'), where M' is obtained from M by replacing all rules of M with q_0 in their left-hand side by the single rule $q_0(\sigma(x_1 \cdots x_n)) \to q(x_i)$. Hence the language N which is the union of all $N(\sigma, q, x_i)$ is in $yT_{\mathrm{ml}(1)}(\mathfrak{L})$. Let A be the nondeterministic gsm mapping $\{\langle w, y\rangle \mid y \in (\Delta - \{\#\})^*$ and $w = w_1 \# y \# w_2$ for some $w_1, w_2 \in \Delta^*\}$. It follows from the assumption of this case and from $yT_{\mathrm{ml}(1)}(\mathfrak{L})$ being a full semi-AFL that $L = A(N) \in yT_{\mathrm{ml}(1)}(\mathfrak{L})$.

Assume now that $\mathfrak{L}$ is monadic. In this case we first modify M as follows, cf. Theorem 4 of [51]. For each state $q \in Q$ we introduce a new state q'. In the right-hand side of each rule for q_0 we replace every occurrence of $q(x_i)$ by $q(x_i)\, q'(x_i)$. For each $q_1 \neq q_0$ a rule $q_1(\sigma(x)) \to w_1q_2(x)w_2$ is replaced by the two rules $q_1(\sigma(x)) \to w_1q_2(x)$ and $q_1'(\sigma(x)) \to$

$q_2'(x)w_2$, and a rule $q_1(\sigma) \to w$ by the two rules $q_1(\sigma) \to w$ and $q_1'(\sigma) \to \lambda$ (and similarly for a rule $q_1(\sigma(x)) \to w$). The new M clearly is in $yT_{\text{ml}(2k)}$. Moreover, it should be clear that by applying the previous construction each M' is in fact a gsm. Hence each $N(\sigma, q, x_i)$ and also their union N is in the smallest full semi-AFL containing $\mathfrak{L}$. The same holds for $L = A(N)$.

Case 2. Suppose that the assumption of Case 1 is false. Take $y_0 \in L$ such that $y_0 \notin A(N)$, and consider the string $(y_0\#)^n$ in $M(L')$ for some large n. Let $q_0(t) \stackrel{*}{\Rightarrow} (y_0\#)^n$ and consider the rule applied initially. Due to the assumption of this case, the output produced by each $q(t_i)$ in the resulting sentential form contains at most one $\#$. Hence the number of $\#$'s in the output string of this derivation is bounded. This contradicts the choice of n. ∎

For the next corollary, cf. [34, Theorem 6.11; 35, Theorem 3.4].

(3.2.11) COROLLARY. *If $\mathfrak{L}$ is a full semi-*AFL *and* $(L\#)^* \in \text{ET0L}_{\text{ml}}(\mathfrak{L})$, *then* $L \in \mathfrak{L}$.

Proof. By Theorem 3.2.2. ∎

(3.2.12) COROLLARY. *If* $(L\#)^* \in yT_{\text{ml}}(\text{REC})$, *then* $L \in \text{CF}$. *If* $(L\#)^* \in \text{ET0L}_{\text{ml}}$ *then* $L \in \text{REG}$.

Proof. REC is closed under linear transducers and $y\text{REC} = \text{CF}$ is a full semi-AFL. Also REG is a full semi-AFL. ∎

It follows that $\{a^nb^n\# \mid n \geqslant 1\}^*$ is not in ET0L_{ml}, see also [51]. For $yT_{\text{ml}}(\text{REC})$ we obtain the following result.

(3.2.13) COROLLARY. $\{a^nb^nc^n\# \mid n \geqslant 0\}^* \in \text{ET0L}_{\text{FIN}(2)} - yT_{\text{ml}}(\text{REC})$.

Proof. Example 3.1.4 (ii) provides an $\text{ET0L}_{\text{FIN}(2)}$ system for this language, whereas Corollary 3.2.12 shows that it is not in $yT_{\text{ml}}(\text{REC})$. ∎

The next, very useful, theorem provides a method to obtain languages not in $yDT(\mathfrak{L})$. It shows that regular substitution (or inverse homomorphism) cannot be handled by a deterministic yT transducer, unless its copying power is not fully used (i.e., it is finite copying). Similar theorems in the literature, concerning $\mathfrak{L} = \text{REC}$ and $\mathfrak{L} = \text{REG}$ [15, 44], have used languages with strings of exponential length to force the transducer to use its full copying power. In [63] the result was obtained independently for $\mathfrak{L} = \text{REG}$.

(3.2.14) THEOREM. *Let L be a language over alphabet Ω and let $b \notin \Omega$. Let* rub(*bish*) *be the regular substitution defined by* $\text{rub}(a) = b^*ab^*$ *for all* $a \in \Omega$. *Let $\mathfrak{L}$ be a class of tree languages closed under finite-state relabelings. Then* $\text{rub}(L) \in yDT(\mathfrak{L})$ *implies* $L \in yT_{\text{fc}}(\mathfrak{L})$. *More precisely, for all* $k \geqslant 1$, $\text{rub}(L) \in yDT_{(k)}(\mathfrak{L})$ *implies* $L \in yT_{\text{fc}(k)}(\mathfrak{L})$.

Proof. The idea of the proof is borrowed from Fischer's proof [28] that the language $\text{rub}(\{a^{2^n} \mid n \geqslant 0\})$ is not an IO macro language. Let $\text{rub}(L) = M(K)$ with $K \in \mathfrak{L}$ and $M = (Q, \Sigma, \Delta, q_0, R)$ in yDT with $\Delta = \Omega \cup \{b\}$ and $Q = \{p_1, \ldots, p_m\}$. A string of the

form $a_1b^{n_1}a_2b^{n_2}\cdots a_rb^{n_r}a_{r+1} \in \mathrm{rub}(L)$ with $a_i \in \Omega$ and $a_1a_2\cdots a_{r+1} \in L$ will be called a δ-string if the n_i are all different. Consider a derivation

$$q_0(t) \overset{*}{\Rightarrow} w_1q_1(t_1)\, w_2q_2(t_1)\cdots w_sq_s(t_1)w_{s+1}$$
$$\overset{*}{\Rightarrow} w_1v_1w_2v_2\cdots w_sv_sw_{s+1} = w$$

of M with $t \in K$, $\langle q_1, \ldots, q_s\rangle$ the state-sequence at the root of t_1, $q_i(t_1) \overset{*}{\Rightarrow} v_i$ by M, and $w \in \Delta^*$ a δ-string. If $q_i = q_j$ then v_i (which, due to determinism, is equal to v_j) contains at most one occurrence of a symbol of Ω (because w is a δ-string). Thus if a state occurs more than once in a state-sequence at some node of t, then the corresponding translation does not contain two occurrences of symbols of Ω. We can get rid of the multiple occurrences of states in state-sequences by producing a single occurrence of an Ω-symbol (or λ) in a translation of a subtree directly, instead of doing the translation. By the previous remarks, we will still be able to recover L (by simply disregarding the b's).

The formal construction is as follows. Each node d labeled $\sigma \in \Sigma_n$ of an input tree $t \in K$ is relabeled by $\langle \sigma, D\rangle \in \Sigma'_n$, where $D = (d_{ij})$ is an $m \times n$ matrix defined as follows. Let the subtree of t with root d be $\sigma(t_1 \cdots t_n)$. Then, for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$,

$$\begin{aligned} d_{ij} &= \lambda && \text{if } p_i(t_j) \overset{*}{\Rightarrow} w \text{ for some } w \in b^*, \\ &= a && \text{if } p_i(t_j) \overset{*}{\Rightarrow} w \text{ for some } w \in b^*ab^*,\ a \in \Omega, \\ &= p_i(x_j) && \text{otherwise.} \end{aligned}$$

Thus the matrix D contains the information about the sons of d whether they are translated into 0, 1, or $\geqslant 2$ symbols of Ω (and in case it is 1, which one). Let t' be the resulting labeled tree and $K' = \{t' \mid t \in K\}$. It is left to the reader to show that the D-matrices can be put on the trees by a finite-state relabeling and hence $K' \in \mathfrak{L}$ (note that if $N \in yT$ and $L_0 \in \mathrm{REG}$, then $N^{-1}(L_0) \in \mathrm{REC}$; cf. [43, 19]).

We now construct $M' = (Q, \Sigma', \Delta, q_0, R')$, where Σ' is the ranked alphabet of all $\langle \sigma, D\rangle$ as described above, and if $q(\sigma(x_1 \cdots x_n)) \to r$ is a rule in R, then $q(\langle \sigma, D\rangle(x_1 \cdots x_n)) \to r'$ is in R', where r' is obtained from r by replacing each occurrence of $p_i(x_j)$ by d_{ij} and each occurrence of b by λ. Thus M' does not produce b, and produces directly those translations that contain 0 or 1 Ω-symbols. It is obvious that $M'(K') = L$. More precisely, it can be shown that if $q_0(t) \overset{*}{\Rightarrow} w$ in M, then $q_0(t') \overset{*}{\Rightarrow} v$ in M', where v is the result of erasing the b's of w, such that at each node of t' the state-sequence of M' is obtained from that of M (at the corresponding node of t) by erasing all states which will produce at most one Ω-symbol in $q_0(t) \overset{*}{\Rightarrow} w$. In particular, if w is a δ-string, then all multiple occurrences are erased from the state-sequences. Hence if (M, K) has state-bound k, then (M', K') has copying-bound k and so $L = M'(K') \in yT_{\mathrm{fc}(k)}(\mathfrak{L})$. ∎

Several corollaries can be obtained from this theorem.

(3.2.15) COROLLARY. *If $\mathfrak{L}$ is a class of languages closed under sequential machine mappings, then* $\mathrm{rub}(L) \in \mathrm{EDT0L}(\mathfrak{L})$ *implies* $L \in \mathrm{ET0L}_{\mathrm{FIN}}(\mathfrak{L})$. *In particular,* $\mathrm{rub}(L) \in \mathrm{EDT0L}$ *implies* $L \in \mathrm{ET0L}_{\mathrm{FIN}}$. (rub *is the regular substitution defined in Theorem* 3.2.14).

Proof. Immediate from Theorems 3.2.2 and 3.2.14. ▮

(3.2.16) COROLLARY. *The language* $\{w \in \{a, b\}^* \mid$ *the number of occurrences of* a *in* w *is* 2^n *for some* $n \geqslant 0\}$ *is in* $\mathrm{ET0L}_{(1)}$ *but not in* $yDT(\mathrm{REC})$.

Proof. The language is generated by the $\mathrm{ET0L}_{(1)}$ system with rules

$$q(\sigma x) \to q(x)\, q(x),$$
$$q(\tau x) \to bq(x), \qquad q(\tau x) \to q(x)b, \qquad q(\tau x) \to q(x),$$
$$q(\delta x) \to a.$$

If the language would be in $yDT(\mathrm{REC})$, then, by Theorem 3.2.14, the language $\{a^{2^n} \mid n \geqslant 0\}$ would be in $yT_{\mathrm{fc}}(\mathrm{REC})$. This contradicts Corollary 3.2.8. ▮

We now state a result which can be proved using Theorem 3.2.14 and a result of [35]. It says that, for any full semi-AFL $\mathfrak{L}$, $\mathfrak{L}$-controlled EDT0L systems cannot generate all context-free languages, unless of course $\mathrm{CF} \subseteq \mathfrak{L}$. It thus improves the result of [14] that $\mathrm{CF} \not\subseteq \mathrm{EDT0L}$. Our result even holds for an arbitrary full principal substitution-closed AFL instead of CF.

(3.2.17) THEOREM. *Let* $\mathfrak{L}$ *be a full semi-*AFL *and* $\mathfrak{L}_1$ *a full principal substitution-closed* AFL. *If* $\mathfrak{L}_1 \subseteq \mathrm{EDT0L}(\mathfrak{L})$, *then* $\mathfrak{L}_1 \subseteq \mathfrak{L}$.

Proof. Greibach [35] has shown that the theorem is true for $\mathrm{ET0L}_{\mathrm{FIN}}(\mathfrak{L})$ (which is denoted as FINITEVISIT($\mathfrak{L}$) in [35]; cf. Corollary 4.10) instead of $\mathrm{EDT0L}(\mathfrak{L})$. Assume now that $\mathfrak{L}_1 \subseteq \mathrm{EDT0L}(\mathfrak{L})$ and let $L \in \mathfrak{L}_1$. Then clearly $\mathrm{rub}(L) \in \mathfrak{L}_1$ and hence $\mathrm{rub}(L) \in \mathrm{EDT0L}(\mathfrak{L})$. It now follows from Corollary 3.2.15 that $L \in \mathrm{ET0L}_{\mathrm{FIN}}(\mathfrak{L})$. Consequently $\mathfrak{L}_1 \subseteq \mathrm{ET0L}_{\mathrm{FIN}}(\mathfrak{L})$ and hence $\mathfrak{L}_1 \subseteq \mathfrak{L}$ by Greibach's result. ▮

(3.2.18) COROLLARY. *Let* $\mathfrak{L}$ *be a full semi-AFL.*

(i) *If* $CF \subseteq \mathrm{EDT0L}(\mathfrak{L})$, *then* $\mathrm{CF} \subseteq \mathfrak{L}$.

(ii) *If* $\mathrm{ET0L} \subseteq \mathrm{EDT0L}(\mathfrak{L})$, *then* $\mathrm{ET0L} \subseteq \mathfrak{L}$.

Proof. Both CF and ET0L are full-principal substitution-closed AFL. ▮

See [63] for essentially the same proof of (i) for $\mathfrak{L} = \mathrm{REG}$.

Note that if $\mathrm{EDT0L}(\mathfrak{L})$ does not contain CF, then in particular it does not contain the Dyck set over two letters. In fact, $\mathrm{EDT0L}(\mathfrak{L})$ is closed under deterministic gsm mappings (cf. Section 5) and each context-free language is the image of the Dyck set over two letters by some deterministic gsm mapping (cf. [63]).

The proof of Theorem 3.2.17 is essentially different from that in [14]. In fact it employs the usual language-theoretic techniques to show that certain classes of languages are not closed under certain operations [31, 33, 28, 24, 62].

To complete the proof of the correctness of the diagram of Fig. 3 we show that

$yT_{\mathrm{ml}(2)}(\mathrm{REC}) - \mathrm{ETOL} \neq \varnothing$. In fact, let $L_0 = \{\sigma(t) \mid t \in L_1\}$ such that $\mathrm{yield}(L_1) \in \mathrm{CF} - \mathrm{EDTOL}$ and σ is a new symbol. Let M be the yT transducer with rules

$$\begin{aligned} &q_0(\sigma(x)) \to q_1(x) \# q_1(x), \\ &q_1(\tau(x_1 \cdots x_n)) \to q_1(x_1) \cdots q_1(x_n) \qquad \text{for} \quad \tau \neq \sigma, \\ &q_1(\tau) \to \tau \qquad\qquad\qquad\qquad\quad \text{for} \quad \tau \neq \sigma. \end{aligned}$$

Clearly $M \in yT_{\mathrm{ml}(2)}$ and $M(L_0) = \{w \# w \mid w \in \mathrm{yield}(L_1)\}$. But $M(L_0) \in \mathrm{ETOL}$ would imply that $M(L_0) \in \mathrm{EDTOL}$ and hence $\mathrm{yield}(L_1) \in \mathrm{EDTOL}$ (see [24]) contrary to the choice of L_1. Thus $M(L_0) \in yT_{\mathrm{ml}(2)}(\mathrm{REC}) - \mathrm{ETOL}$.

The correctness of the diagram of Fig. 3 is now proved for $k \geqslant 2$. For $k = 1$ we have that $yT_{\mathrm{fc}(1)}(\mathrm{REC}) = yT_{\mathrm{ml}(1)}(\mathrm{REC}) = \mathrm{CF}$ (because REC is closed under linear transducers), and $\mathrm{ETOL}_{\mathrm{FIN}(1)} = \mathrm{ETOL}_{\mathrm{ml}(1)} = \mathrm{LIN}$ as can easily be proved. We note that the above language $M(L_0)$ is also in $yDT_{(1)}(\mathrm{REC})$. Together with Corollary 3.2.8 this almost shows the correctness of the diagram of Fig. 3 with the $k = 1$ case added. The open question is whether $\mathrm{CF} \subseteq \mathrm{ETOL}_{(k)}$ for some k (of course $\mathrm{CF} \subseteq \mathrm{ETOL}$).

3.3. *Two Extensions*

We end this section by introducing two generalizations of the top-down tree-to-string transducer which will be useful in the next sections. The first generalization consists of allowing rules with right-hand sides which are regular languages (for the monadic case these are the *iteration grammars* discussed in [60, 53, 7]). The second generalization consists of allowing the transducer to have an infinite (but recognizable) look-ahead on its input subtrees (see [19]). For both generalizations we show that they do not extend the classes of tree transformation languages.

(3.3.1) DEFINITION. A *regularly extended top-down tree-to-string transducer* (notation yRT) is defined as in Definition 3.1.5, except that the set of rules R may be infinite. It is required, however, that for given $q \in Q$ and $\sigma \in \Sigma_n$ the set of all r such that $q(\sigma(x_1 \cdots x_n)) \to r$ is in R is a regular language over $\Delta \cup Q(X_n)$. The definition of derivation is as in Definition 3.1.5. ∎

Since the translations in yT clearly have the finite-image property (i.e., each input tree is translated into a finite number of output strings), yRT is a larger class of translations. It can be shown that yT is equal to the class of all finite-image yRT translations. We now show that the corresponding $\mathfrak{L}$-tree transformation classes are equal.

(3.3.2) LEMMA. *Let $\mathfrak{L}$ be a class of tree languages closed under insertion of regular languages. Then $yRT(\mathfrak{L}) = yT(\mathfrak{L})$ and, for each $k \geqslant 1$, $yRT_{\mathrm{fc}(k)}(\mathfrak{L}) = yT_{\mathrm{fc}(k)}(\mathfrak{L})$ and $yRT_{\mathrm{ml}(k)}(\mathfrak{L}) = yT_{\mathrm{ml}(k)}(\mathfrak{L})$.*

Proof. Let us first show that $yRT(\mathfrak{L}) \subseteq yT(\mathfrak{L})$. Let $M = (Q, \Sigma, \Delta, q_0, R)$ be in yRT and $L \in \mathrm{REC}$. For each $q \in Q$ and $\sigma \in \Sigma_n$ denote by $R(q, \sigma)$ the set of all r such that $q(\sigma(x_1 \cdots x_n)) \to r$ is in R. Let $\Sigma' = \{\sigma' \mid \sigma \in \Sigma\}$ be a set of new symbols of rank 1. The

tree language $L' \in \mathfrak{L}$ is obtained from L by inserting arbitrary long sequences of symbols σ' above each node labeled σ. We now construct $M' = (Q', \Sigma'', \Delta, q_0', R')$ in yT such that $M'(L') = M(L)$. Let $G(q, \sigma)$ be a right-linear grammar for $R(q, \sigma)$, with terminal alphabet $\Delta \cup Q(X)$, and let N be the set of all nonterminals so obtained. Let $Q' = Q \cup (Q \times N) \cup (Q \times \{1,..., m\})$, where m is the maximal rank in Σ. Let $\Sigma'' = \Sigma \cup \Sigma'$, let $q_0' = q_0$ and let the rules in R' be defined as follows. If S is the initial nonterminal of $G(q, \sigma)$, then $q(\sigma'(x)) \to \langle q, S\rangle(x)$ is in R'. If $A \to aB$ with $a \in \Delta$ is a rule in $G(q, \sigma)$, then $\langle q, A\rangle(\sigma'(x)) \to a\langle q, B\rangle(x)$ is in R'. If $A \to p(x_i)B$ is a rule of $G(q, \sigma)$, then $\langle q, A\rangle(\sigma'(x)) \to \langle p, i\rangle(x)$ $\langle q, B\rangle(x)$ is in R'. If $A \to \lambda$ is a rule of $G(q, \sigma)$, then $\langle q, A\rangle(\sigma'(x)) \to \lambda$ is in R'. Finally, all rules $\langle q, i\rangle(\sigma'(x)) \to \langle q, i\rangle(x)$ and $\langle q, i\rangle(\sigma(x_1 \cdots x_n)) \to q(x_i)$ are in R'. It is left to the reader to prove that $M'(L') = M(L)$.

Assume now that M is in $yRT_{\mathrm{fc}(k)}$. This implies that each $R(q, \sigma)$ is a finite union of languages of the form $R_1 q_1(x_{i_1})\, R_2 q_2(x_{i_2}) \cdots R_n q_n(x_{i_n}) R_{n+1}$, where the R_i are regular languages over Δ. Let δ be a new symbol of rank 1. We now construct $L' \in \mathfrak{L}$ by inserting arbitrary sequences of symbols σ' above each node labeled σ, as before, but also inserting arbitrary long sequences of symbols δ below σ (for each of its branches). M' is now constructed such that it first outputs a string of R_1 (using the sequence of symbols σ' to simulate a grammar for R_1, as before), then (arriving at the node labeled σ) applies a rule with right-hand side $q_1(x_{i_1})\, q_2(x_{i_2}) \cdots q_n(x_{i_n})$, and then uses the δ's to simulate (left-linear) grammars for $R_2, R_3,..., R_{n+1}$. A formal construction is left to the reader. It should be clear that the number of copies made of each new monadic node is equal to that of the first old node beneath it. Hence $M(L) = M'(L') \in yT_{\mathrm{fc}(k)}(\mathfrak{L})$.

The metalinear case can be proved by a slight variation of this method, and is left to the reader. ∎

Next we define regular look-ahead.

(3.3.3) DEFINITION. A *top-down tree-to-string transducer with regular look-ahead* (notation yT^R) is the same as in Definition 3.1.5 except that with each rule $q(\sigma(x_1 \cdots x_n)) \to r$ of R a mapping $D: X_n \to \mathrm{REC}$ is associated. The mapping D restricts the application of the rule by requiring that, in a sentential form, a substring $q(\sigma(t_1 \cdots t_n))$ can be replaced by $r[t_1,..., t_n]$ only if $t_i \in D(x_i)$ for all i, $1 \leqslant i \leqslant n$.

A transducer in yT^R is *deterministic* (notation yDT^R) if for any pair of different rules $q(\sigma(x_1 \cdots x_n)) \to r_1$ and $q(\sigma(x_1 \cdots x_n)) \to r_2$ (with the same left-hand side) there exists i, $1 \leqslant i \leqslant n$, such that $D_1(x_i) \cap D_2(x_i) = \varnothing$, where D_1 and D_2 are the mappings associated with these rules. ∎

All previous definitions in this section can be generalized in an obvious way to the case of regular look-ahead.

The next lemma shows that regular look-ahead does not extend the class of $\mathfrak{L}$-tree transformation languages.

(3.3.4) LEMMA. *Let $\mathfrak{L}$ be a class of tree languages closed under finite-state relabelings. Then $yT^R(\mathfrak{L}) = yT(\mathfrak{L})$ and $T^R(\mathfrak{L}) = T(\mathfrak{L})$, and the constructions involved preserve determinism, state-bound, copying-bound and metalinear-bound.*

Proof. The inclusions $yT(\mathfrak{L}) \subseteq yT^R(\mathfrak{L})$ and $T(\mathfrak{L}) \subseteq T^R(\mathfrak{L})$ are trivial by providing the rules with look-ahead T_Σ, where Σ is the input alphabet. The inclusion $T^R(\mathfrak{L}) \subseteq T(\mathfrak{L})$ is proved in Theorems 2.6 and 4.2 of [19]. It is easy to see that the same proof applies to yT transducers, giving $yT^R(\mathfrak{L}) \subseteq yT(\mathfrak{L})$, and that the construction preserves determinism and bounds. ∎

4. Two-Way Transducers and One-Way Checking Machines

The devices studied in the previous section can be viewed in two different ways. First, they can be considered as parallel rewriting systems that generate languages. This is true in particular of ET0L systems, but it should be clear that the top-down tree transformation system can also be formulated with parallel rewriting rather than unrestricted rewriting (see the discussion following Definition 3.1.1). In fact, subcases of the GSDT have actually been formulated in such a way [1].

Alternatively, as argued in Section 3, these devices can be viewed as transducers, the ranges of which are of special interest. These transducers are a generalization of the usual transducers in automata theory in that they operate in a highly parallel fashion.

In this section we investigate automata of the usual sequential type which correspond to the parallel devices studied in the previous section. The feature of parallelism is simulated by a two-way (sequential) motion on the tree or string (up and down, or left and right, respectively). Analogous to the above discussion, such an automaton can be viewed in two ways which we discuss now in the opposite order. First, it can be viewed as a tree-to-string (or string-to-string) transducer which moves two-way on the input tree (or string). As such it is a sequential implementation of the parallel transducer, and we will be interested mainly in its range. Second, it can be considered as an acceptor by viewing the output string of the transducer as (one-way) input string and the input tree (or string) of the transducer as part of its (two-way) memory. Since this memory can first be filled nondeterministically with any tree (or string), such an acceptor is a generalization of the checking stack automaton of [31]; we will therefore call it a one-way checking machine (its memory will also contain a kind of pushdown store, as discussed later). As such the automaton is an acceptor of the languages generated by parallel rewriting systems.

Several examples of this correspondence between parallel and sequential transducers are known from the literature. We mention some of them. In [4] it is shown that finite copying GSDT can be realized by a sequential deterministic tree-to-string transducer (with the derivation trees of a context-free grammar as input language); from this it follows that $yT_{fc}(\mathrm{REC})$ equals the class of images of REC under these transducers. The monadic case of this result is proved in [45]. In fact, it is shown there that absolutely parallel grammars (which are equivalent to $\mathrm{ET0L_{FIN}}$ systems) generate precisely all the ranges of two-way deterministic finite-state string transducers. This result is generalized to arbitrary input (control) languages in [35], where also bounds are taken into account. Earlier it was shown in [55] that the equal matrix grammars (which are special $\mathrm{ET0L_{ml}}$ systems) generate the class of languages accepted by finite-turn checking automata. In

[45] it was observed that the two-way (nondeterministic) finite-state string transducer is equivalent to the checking stack automaton of [31] and this fact was generalized to arbitrary classes of input languages in [41]. For the unbounded case, it was proved in [61] that the class of ET0L languages is accepted by the cs–pd (checking stack – pushdown) automaton, whose memory consists of both a checking stack and a pushdown store, synchronized in such a way that a move up (or down) the stack is always accompanied by a pop (or push, respectively) on the pushdown. From the transducer point of view this machine is therefore a two-way pushdown transducer in which the movements (left or right) on the input string are synchronized with the operations (pop or push, respectively) on the pushdown. It was shown in [26] that the class of ranges of the deterministic version of this transducer is EDT0L. In [26] the cs–pd machine was generalized to the s–pd machine (with a stack rather than a checking stack) in order to characterize certain classes of languages generated by macro grammars. In this section we generalize both the cs–pd automaton and the tree transducer of [4] to a two-way tree-to-string pushdown transducer of which the movements up and down the tree are synchronized with the pops and pushes on the pushdown (respectively). Viewed as an acceptor this machine has a checking tree rather than a checking stack in its memory. In order to keep the flavor of both views we will call this new device a ct–pd (checking tree – pushdown) transducer. We will show that the class of images of REC under ct–pd transducers is equal to $yT(\mathrm{REC})$, and similarly for arbitrary classes of input tree languages (satisfying some weak conditions). We then show that this result can be restricted to the deterministic, bounded, and monadic cases, thereby proving most of the results mentioned above. In particular a connection will be shown between finite copying and finite crossing, where "finite crossing" means that there is a bound on the number of times an arc of the input tree may be crossed (in either direction) by the ct–pd transducer. For the monadic case this issue was investigated in [46], and systematically in [35]. Similarly, the metalinear restriction corresponds to finite-pass transducers, where "finite-pass" means that the transducer can make only a bounded number of passes over the input tree. This was investigated in the monadic case in [55], and systematically in [35]. At the end of the section we exhibit an inclusion diagram for all the classes of machines considered (including those of [26]) and prove the correctness of this diagram.

The formal definition of the ct–pd transducer is as follows.

(4.1) DEFINITION. A *checking tree – pushdown transducer* (abbreviated by ct–pd transducer) is a construct $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$, where Q is a finite set of states, Σ is the ranked input alphabet, Γ is the pushdown alphabet, Δ is the output alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is a mapping from $Q \times \Sigma \times \Gamma$ into the finite subsets of $Q \times D \times \Delta^*$, where $D = \{\mathrm{up}\} \cup \{\mathrm{stay}(\gamma) \mid \gamma \in \Gamma\} \cup \{\mathrm{down}(i, \gamma) \mid \gamma \in \Gamma, i \geqslant 1\}$. A ct–pd transducer is *deterministic* (notation: dct–pd) if δ is a partial function from $Q \times \Sigma \times \Gamma$ into $Q \times D \times \Delta^*$. ▮

The words up, stay, and down are just identifiers used to facilitate the reading of the specifications of δ. We note that, to be precise, we should have defined δ to have the domain $Q \times (\Sigma \times \mathbb{N}) \times \Gamma$, where for $\sigma \in \Sigma$ and $i \in \mathbb{N}$, $\langle \sigma, i \rangle$ indicates a symbol of rank i

(that is, M "knows" the rank of the input symbol). To avoid complicated notation, the rank of σ should always be clear from the context.

Intuitively, a configuration of a ct–pd transducer M consists of (see Fig. 5) an input tree (or checking tree), a pushdown tape, an output tape, and a finite control with three pointers: one to a node of the checking tree (the input pointer), one to the end of the output tape and one to the top of the pushdown. Suppose that the node d of the input tree pointed at is labeled $\sigma \in \Sigma_n$, that γ is the symbol on the top of the pushdown, and that the transducer is in state q. If $\delta(q, \sigma, \gamma)$ contains (q', e, w), then the transducer can go into state q', add w to the output, and act as follows depending on the value of e: if $e =$ up, then M moves the input pointer to the father of d and pops the pushdown; if $e =$ stay(γ'), then M changes γ into γ' and does not move its input pointer; if $e =$ down(i, γ') and $1 \leqslant i \leqslant n$, then M moves its input pointer to the ith son of d and pushes γ' on the pushdown.

Given some input tree, M starts in the initial state, its input pointer at the root of the tree, the pushdown filled with one pushdown element and empty output. The computation of M ends successfully when M "falls off" the tree (by moving up from its root), empties its pushdown, and goes into a final state. It follows from this description that the number of symbols on the pushdown is always equal to the number of nodes on the path from the root to the node pointed at. In fact a good way to view the pushdown is to assume that each node of the input tree has an associated square on which a pushdown symbol can be printed, and to let the pushdown consist of the squares on the above-mentioned path (this is the reason we have drawn the pushdown upside-down in Fig. 5; but note that trees are in fact also drawn upside down!).

At the end of their paper [4], Aho and Ullman discuss pebble automata, walking on trees, which keep the pebbles between a node and the root of the tree. Such a pebble automaton is in fact a restricted ct–pd transducer: a k-pebble automaton is a ct–pd transducer $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$ such that $\Gamma = \{0, 1\}$ and during computation at most k squares of the pushdown may contain 1, i.e., a pebble.

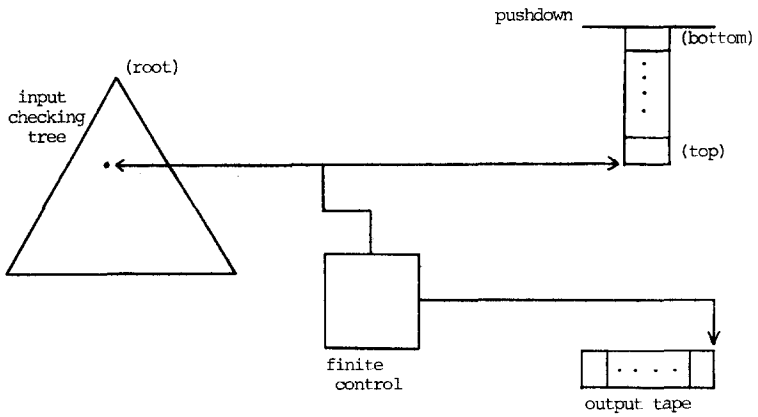

FIG. 5. The checking tree pushdown transducer.

(4.2) DEFINITION. Let $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$ be a ct–pd transducer. A *configuration* of M is a sequence $\langle q, \$(t), d, \psi, w\rangle$ with $q \in Q$, $t \in T_\Sigma$, d is a node of $\$(t)$, $\psi \in \Gamma^*$, and $w \in \Delta^*$ ($\$$ is a "new" symbol of rank 1). The "move-relation" $\vdash$ between configurations is defined as follows. Let $q, q' \in Q$; $t \in T_\Sigma$; d a node of $\$(t)$ labeled with $\sigma \in \Sigma_n$ $(n \geqslant 0)$; $\psi \in \Gamma^*$; $\gamma, \gamma' \in \Gamma$; and $v \in \Delta^*$. Let $C_1 = \langle q, \$(t), d, \psi\gamma, w\rangle$ and let $\delta(q, \sigma, \gamma)$ contain (q', e, v). Then $C_1 \vdash C_2$, such that

(i) if $e =$ up, then $C_2 = \langle q', \$(t), d', \psi, wv\rangle$, where d' is the father of d;

(ii) if $e =$ stay(γ'), then $C_2 = \langle q', \$(t), d, \psi\gamma', wv\rangle$; and

(iii) if $e =$ down(i, γ) and $1 \leqslant i \leqslant n$, then $C_2 = \langle q', \$(t), d', \psi\gamma\gamma', wv\rangle$, where d' is the ith son of d.

As usual, $\vdash^*$ is used to denote computations of M (i.e., sequences of $\vdash$ moves). The *translation* realized by M, denoted also by M, is $M = \{\langle t, w\rangle \in T_\Sigma \times \Delta^* \mid \langle q_0, \$(t), d_0, \gamma, \lambda\rangle \vdash^* \langle q, \$(t), d_\infty, \lambda, w\rangle$ for some $\gamma \in \Gamma$ and $q \in F\}$, where d_0 denotes the root of t and d_∞ the root of $\$(t)$. ∎

The class of translations realized by ct–pd transducers {dct–pd transducers} is denoted by CT–PD{DCT–PD}. The class of images of tree languages from a class $\mathfrak{L}$ under ct–pd transducers will be denoted by CT–PD($\mathfrak{L}$).

Let $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$ be a ct–pd transducer. M is called a *checking tree transducer* (ct transducer) if Γ is a singleton. In that case the pushdown is useless. For ct transducers we omit all reference to Γ, thus $M = (Q, \Sigma, \Delta, \delta, q_0, F)$ and δ is a multivalued function from $Q \times \Sigma$ into $Q \times D \times \Delta^*$, where $D = \{\text{up, stay}\} \cup \{\text{down}(i) \mid i \geqslant 1\}$. The deterministic checking tree transducer (dct transducer) is the tree automaton of [4]. M is a *checking stack – pushdown transducer* (cs–pd transducer) if Σ is monadic, and M is a *checking stack transducer* (cs transducer) if Σ is monadic and Γ is a singleton. In the monadic case we shall sometimes write "left" and "right(γ)" rather than "up" and "down(1, γ)." It is easy to show that (with respect to ranges) the cs–pd transducer is equivalent to the cs–pd machine of [61], and that the cs transducer is equivalent to both the checking stack automaton of [31] and the 2-way gsm of [3, 17, 41, 45].

The classes of translations corresponding to the above transducers will be indicated by capitals. Thus CT and DCS–PD denote the class of translations realized by ct and dcs–pd transducers, respectively.

Next we define restrictions on the ct–pd transducer corresponding to the finite copying and metalinear restrictions of the previous section (there seems to be no clear concept corresponding to state-bound).

(4.3) DEFINITION. Let $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$ be a ct–pd transducer. A move of of type (i) or (iii), as in Definition 4.2, is called a *crossing* of the arc between d and d' (upward or downward, respectively). For $k \geqslant 1$ a computation of M is *k-crossing* if each arc of t is crossed at most $2k$ times (in either direction) in that computation.

Let L be a tree language. For $k \geqslant 1$, the pair (M, L) is *k-crossing* if for each $w \in M(L)$ there exist $t \in L$ and a successful k-crossing computation of M on t with output w. (M, L) is *finite crossing* if it is k-crossing for some k.

A *pass* of M on a subtree t_i of an input tree $t = \sigma(t_1 \cdots t_n)$ is a computation of M on t which consists of a move from the root of t down to the root of t_i, followed by a 1-crossing computation on t_i, followed by a move from the root of t_i up to the root of t. A computation of M on $t = \sigma(t_1 \cdots t_n)$ is *k-pass* if it consists of consecutive passes on the t_i's (ending by a move up to d_∞), such that each t_i is passed at most k times. (M, L) is k-pass if there is a succesful k-pass computation for each $w \in M(L)$, and (M, L) is *finite pass* if it is k-pass for some $k \geqslant 1$. The above terminology also applies to M if it is true of (M, T_Σ). ▮

"Finite crossing" is denoted by a subscript "fc," finite pass" by a subscript "fp," and bounds, as usual, by subscripts "(k)." Thus CT–PD$_{\text{fc}(k)}$(REC) denotes the class of all languages $M(L)$ with $M \in$ CT–PD and $L \in$ REC, such that (M, L) is k-crossing.

Intuitively, a ct–pd transducer is k-crossing if it makes at most k translations of each subtree s of the input tree. In fact, each subcomputation which starts by crossing downward the arc to the root of s from its father and ends by crossing it upward may be viewed as one translation of s.

We first show that each dct transducer is finite crossing.

(4.4) LEMMA. *For every class $\mathfrak{L}$ of tree languages,* DCT($\mathfrak{L}$) = DCT$_{\text{fc}}$($\mathfrak{L}$) *and* DCS($\mathfrak{L}$) = DCS$_{\text{fc}}$($\mathfrak{L}$).

Proof. Consider a successful computation of a dct transducer $M = (Q, \Sigma, \Delta, \delta, q_0, F)$ on an input tree t. Suppose that this computation is not k-crossing, where $k = (\#(Q))^2$. Then there exists an arc of t which is twice crossed downward going from the same state q_1 into the same state q_2. This would mean that M is in an endless computation. ▮

We will show later that even CT–PD$_{\text{fc}}$($\mathfrak{L}$) = CT$_{\text{fc}}$($\mathfrak{L}$) = DCT($\mathfrak{L}$) and similarly for CS–PD (see the remarks following Theorem 4.9).

A deep investigation into the properties of the ranges of finite-crossing and finite-pass checking stack transducers (with arbitrary class of input languages) was made in [35]. It is shown there [35 (Theorem 2.2)] that finite-crossing cs transducers are equivalent (with respect to ranges) to "finite-visit" cs transducers (meaning that the transducer visits each "node" of the input string a bounded number of times), with the same bound. The same fact can easily be shown for ct–pd (and ct) transducers, however due to non-monadicness the bounds do not correspond. The finite-pass cs transducers are called "finite reversal" in [35], and the definition of finite reversal is somewhat less restrictive than that of finite pass. It can however easily be shown that both definitions are equivalent. The connection between the notation of [35] and ours is that FINITEVISIT($\mathfrak{L}$) = DCS($\mathfrak{L}$), FINITEREVERSAL($\mathfrak{L}$) = DCS$_{\text{fp}}$($\mathfrak{L}$), DCS$_{\text{fc}(k)}$($\mathfrak{L}$) = $2k$-VISIT($\mathfrak{L}$), and DCS$_{\text{fp}(k)}$($\mathfrak{L}$) = $2k$-REVERSAL($\mathfrak{L}$), where $\mathfrak{L}$ is a full semi-AFL (cf. Corollary 4.10).

In the first theorem of this section we show the precise relationship between ct–pd translations and top-down tree-to-string translations.

(4.5) THEOREM. CT–PD = yRT, *and the constructions involved preserve crossing (= copying) bound and pass (= metalinear) bound.*

Proof. To prove that CT–PD $\subseteq yRT$, let $M = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, F)$ be a ct–pd transducer and assume first that M uses no stay instructions. The proof will be similar to that of Lemma 7.5 in [4], where the DCT case is treated. Construct the top-down tree-to-string transducer with regular right-hand sides (see Definition 3.3.3) $M' = (Q \times \Gamma \times Q, \Sigma, \Delta, \{q_0\} \times \Gamma \times F, R)$. Note that M' has a set $\{q_0\} \times \Gamma \times F$ of initial states; this can easily be taken care of. We shall construct the set of rules R of M' in such a way that

$$\langle q_1, \gamma, q_2\rangle(t) \overset{*}{\Rightarrow} w \text{ in } M' \text{ if and only if } \langle q_1, \$(t), d_0, \gamma, \lambda\rangle \overset{*}{\vdash} \langle q_2, \$(t), d_\infty, \lambda, w\rangle \text{ in } M, \tag{*}$$

i.e., M walks on a subtree t starting in state q_1 at its root with pushdown symbol γ "at this root," and "falls off" the subtree in state q_2, producing output w. It is clear that this statement implies that $M' = M$. The rules of R now easily follow.

(1.) Let $\sigma \in \Sigma_0$. If $\delta(q_1, \sigma, \gamma)$ contains (q_2, up, w), then $\langle q_1, \gamma, q_2\rangle(\sigma) \to w$ is in R.

(2.) Let $\sigma \in \Sigma_n$ with $n \geqslant 1$. The rule $\langle q_1, \gamma, q_2\rangle(\sigma(x_1 \cdots x_n)) \to w_1\langle p_1, \gamma_1, p_1'\rangle \times (x_{i_1})\, w_2\langle p_2, \gamma_2, p_2'\rangle(x_{i_2}) \cdots w_k\langle p_k, \gamma_k, p_k'\rangle(x_{i_k}) w_{k+1}$ is in R if and only if $\delta(q_1, \sigma, \gamma)$ contains $(p_1, \text{down}(i_1, \gamma_1), w_1)$, $\delta(p_{m-1}', \sigma, \gamma)$ contains $(p_m, \text{down}(i_m, \gamma_m), w_m)$ for $2 \leqslant m \leqslant k$, and $\delta(p_k', \sigma, \gamma)$ contains (q_2, up, w_{k+1}).

This ends the construction of M'. For fixed q_1, γ, q_2, and σ, the set of possible right-hand sides is clearly regular (it is determined by the finite control δ of M). It is easy to prove the above statement (*) that links M and M', by induction. It can be proved simultaneously, using Definition 3.1.8, that the derivation of M' has copying-bound k if and only if the corresponding computation of M is k-crossing, for any k. After replacing the initial states of M' by one new initial state q_0', and adding the appropriate rules, it should be clear that M' is k-metalinear if M is k-pass.

It is left to the reader to prove the case that M uses stay instructions (it complicates the definition of R only slightly, because γ can be changed by the stay instructions).

To show that $yRT \subseteq$ CT–PD, let $M = (Q, \Sigma, \Delta, q_0, R)$ be in yRT. For $q \in Q$ and $\sigma \in \Sigma_n$ $(n \geqslant 0)$, let $G(q, \sigma)$ be a right-linear grammar generating the set of all right-hand sides of rules with left-hand side $q(\sigma(x_1 \cdots x_n))$. We assume that the corresponding sets $N(q, \sigma)$ of nonterminals of these grammars are mutually disjoint, and we assume also that the rules of $G(q, \sigma)$ are of the form $A \to wB$, $A \to w$, or $A \to p(x_i)B$ with $A, B \in N(q, \sigma)$, $w \in \Delta^*$, $p \in Q$, and $1 \leqslant i \leqslant n$. Let $N = \bigcup\{N(q, \sigma) \mid q \in Q, \sigma \in \Sigma\}$. Let $\not\in$ and $\#$ be new symbols. Construct the ct–pd transducer $M' = (Q', \Sigma, \Gamma, \Delta, \delta, q_0, F)$ such that $Q' = F = \{\#\} \cup Q \cup (Q \times \{1, \ldots, m\})$, where m is the maximal rank of a symbol in Σ, $\Gamma = \{\not\in\} \cup N$, and δ is defined as follows.

(i) $\delta(q, \sigma, \not\in) = (\#, \text{stay}(S), \lambda)$, where S is the initial nonterminal of $G(q, \sigma)$.

(ii) $\delta(\#, \sigma, A)$ contains $(\#, \text{stay}(B), w)$ if $A \to wB$ is a rule, $(\#, up, w)$ if $A \to w$ is a rule, and $(\langle q, i\rangle, \text{stay}(B), \lambda)$ if $A \to q(x_i)B$ is a rule.

(iii) $\delta(\langle q, i\rangle, \sigma, A) = (q, \text{down}(i, \not\in), \lambda)$.

This ends the construction of M'. Its δ is constructed in such a way that the nonterminal in the pushdown square "associated with" a node remembers which part of the right-hand side of the rule applied at this node has already been simulated and which part should still be treated. Formally it can be shown that $q(t) \overset{*}{\Rightarrow} w$ in M if and only if $\langle q, \$(t), d_0, \not\in, \lambda\rangle \overset{*}{\vdash} \langle \#, \$(t), d_\infty, \lambda, w\rangle$ in M' and that the derivation of M has copying-bound k if and only if the computation of M' is k-crossing, for any k. After extending the definition of δ such that $\delta(q_0, \sigma, S) = \delta(\#, \sigma, S)$, where S is the initial nonterminal of $G(q_0, \sigma)$, it is clear that M' is k-pass if M is k-metalinear. This proves the theorem. ∎

Note that it follows from Theorem 4.5 that the domain of a ct–pd transducer is recognizable. In fact, it is easy to associate with each yRT transducer a yT transducer with the same domain; yT transducers have recognizable domains [48].

Note also that yT equals the class of finite image ct–pd translations (by the remark following Definition 3.3.1).

(4.6) COROLLARY. *If $\mathfrak{L}$ is a class of tree languages closed under insertion of regular languages, then* CT–PD($\mathfrak{L}$) = $yT(\mathfrak{L})$. *If $\mathfrak{L}$ is a class of languages closed under λ-free regular substitution and sequential machine mappings, then* CS–PD($\mathfrak{L}$) = ET0L($\mathfrak{L}$). *In particular,* CT–PD(REC) = yT(REC) *and* CS–PD(REG) = ET0L; *moreover, in this case, we may assume that the input language is always T_Σ, where Σ is the input alphabet.*

Proof. The equalities follow from the previous theorem and Theorems 3.3.2 and 3.2.2. The rest of the statement follows by observing that Theorem 3.2.1 can easily be generalized to regular right-hand sides. Hence CT–PD(REC) = yRT(REC) = $yRT(\{T_\Sigma \mid \Sigma$ ranked alphabet$\})$ = CT–PD($\{T_\Sigma \mid \Sigma$ ranked alphabet$\}$) and similarly for CS–PD. ∎

The characterization of ET0L by cs–pd machines was shown in [61].

The next theorem provides a precise characterization of the deterministic ct–pd translations.

(4.7) THEOREM. DCT–PD = yDT^R, *and the construction involved to prove* DCT–PD $\subseteq yDT^R$ *preserves crossing* (= *copying*) *bound and pass* (= *metalinear*) *bound.*

Proof. To prove that DCT–PD $\subseteq yDT^R$ we turn the transducer M' in the first half of the proof of Theorem 4.5 into a deterministic one, using regular look-ahead (see Definition 3.3.3) and the fact that M is deterministic (for notation we refer to the proof of Theorem 4.5). It follows from the remark following Theorem 4.5 that the domain of a ct–pd transducer is recognizable; hence, for given $q_1, q_2 \in Q$ and $\gamma \in \Gamma$, the set of all $t \in T_\Sigma$ such that $\langle q_1, \$(t), d_0, \gamma, \lambda\rangle \overset{*}{\vdash} \langle q_2, \$(t), d_\infty, \lambda, w\rangle$ in M for some $w \in \Delta^*$ is recognizable; let us denote this set by $D(q_1, \gamma, q_2)$. Moreover, since M is deterministic, q_2 is determined uniquely by q_1 and γ. We now associate with the rule mentioned under (2) in the proof of Theorem 4.5 the regular look-ahead mapping D such that, for $1 \leqslant j \leqslant n$, $D(x_j)$ is the intersection of all $D(p_m, \gamma_m, p'_m)$ such that $x_{i_m} = x_j$. By the previous observations this change turns M' into a deterministic top-down tree-to-string transducer with regular look-ahead. The case that M uses stay rules is entirely similar. It should be clear that bounds are still preserved.

Let us now show that $yDT^R \subseteq$ DCT–PD. It is easy to see from the second half of the proof of Theorem 4.5 that if $M \in yDT$, then $M' \in$ DCT–PD (since the $G(q, \sigma)$ generate singletons, we may assume that each nonterminal of $G(q, \sigma)$ is the left-hand side of exactly one of its rules). It remains to prove that the dct–pd transducer can handle regular look-ahead. For each recognizable tree language L we can find a dct–pd transducer which has L as its domain. In fact, if A is a (nondeterministic) top-down finite tree automaton recognizing L, then a dct–pd transducer A' can simulate A by back-tracking. On the pd-square associated with a node, A' puts a possible state-transition of A and then simulates the behavior of A on the succesive subtrees of the node; if this does not lead to acceptance, A' puts the next possible state-transition of A on the pd-square, etc.

Now, if $M \in yDT^R$, then we can construct a dct–pd transducer M' which, when arriving at a node, first checks the regular look-ahead of the immediate subtrees of the node (using back-tracking as described above, and marking the pd-square of the node in order to find it back), and then picks the unique rule to be applied, continuing the simulation as in the second half of the proof of Theorem 4.5. ∎

It was shown in [19] that DT^R has nicer closure properties than DT. The above theorem is another reason to prefer DT^R to DT.

(4.8) COROLLARY. *If $\mathfrak{L}$ is a class of tree languages closed under finite-state relabelings, then* DCT–PD($\mathfrak{L}$) = $yDT(\mathfrak{L})$. *If $\mathfrak{L}$ is a class of languages closed under sequential machine mappings, then* DCS–PD($\mathfrak{L}$) = EDT0L($\mathfrak{L}$). *In particular,* DCT–PD(REC) = yDT(REC) and DCS–PD(REG) = EDT0L; *moreover, in this case, we may assume that the input language is always T_Σ, where Σ is the input alphabet.*

Proof. Similar to the proof of Corollary 4.6, using Lemma 3.3.4 and the generalization of Theorem 3.2.1 to regular look-ahead. ∎

The characterization of EDT0L by dcs–pd transducers was also shown in [26].

We now turn to the deterministic checking tree transducer (without pd-facility) and show that it is closely connected to the finite-copying top-down tree-to-string transducer [4].

(4.9) THEOREM. *Let $\mathfrak{L}$ be a class of tree languages closed under finite-state relabelings. For each $k \geqslant 1$,* $\mathrm{DCT}_{\mathrm{fc}(k)}(\mathfrak{L}) = yT_{\mathrm{fc}(k)}(\mathfrak{L})$, *and* $\mathrm{DCT}_{\mathrm{fp}(k)}(\mathfrak{L}) = yT_{\mathrm{ml}(k)}(\mathfrak{L})$. *Hence* $\mathrm{DCT}(\mathfrak{L}) = yT_{\mathrm{fc}}(\mathfrak{L})$ *and* $\mathrm{DCT}_{\mathrm{fp}}(\mathfrak{L}) = yT_{\mathrm{ml}}(\mathfrak{L})$.

Proof. It follows from Theorem 4.7 that $\mathrm{DCT}_{\mathrm{fc}(k)}(\mathfrak{L}) \subseteq \mathrm{DCT\text{–}PD}_{\mathrm{fc}(k)}(\mathfrak{L}) \subseteq yDT^R_{\mathrm{fc}(k)}(\mathfrak{L})$, and hence by Lemma 3.3.4 $\mathrm{DCT}_{\mathrm{fc}(k)}(\mathfrak{L}) \subseteq yDT_{\mathrm{fc}(k)}(\mathfrak{L})$; and similarly for fp and ml. Thus to prove the theorem (and Lemma 3.2.3!), it suffices to show that $yT_{\mathrm{fc}(k)}(\mathfrak{L}) \subseteq \mathrm{DCT}_{\mathrm{fc}(k)}(\mathfrak{L})$, and similarly for ml and fp (cf. Lemma 4.4).

The simultation is similar to the one given in the second half of the proof of Theorem 4.5. Due to the finite-copying property of the top-down transducer we do not need the pd-facility; instead, the information concerning the (bounded number of) rules applied at each node can be printed on that node in advance. The proof is entirely similar to that of Theorem 7.2 of [4].

Let $M = (Q, \Sigma, \Delta, q_0, R)$ be in $yT_{fc(k)}$ and $L \in \mathfrak{L}$. We first construct a new input language L' by relabeling each node of an input tree by $\langle r_1, \ldots, r_s, n\rangle$, $s \leqslant k$, where $\langle r_1, \ldots, r_s\rangle$ is the rule-sequence at the node corresponding to some derivation of M, and n indicates that this node is the nth son of its father. By the remarks following Definition 3.1.8 this relabeling can be realized by a nondeterministic top-down finite-state relabeling, and hence $L' \in \mathfrak{L}$. To simplify notation we assume that instead of rules only their right-hand sides are printed, in which, moreover, each occurrence of a $q(x_i)$ is replaced by x_i. Thus each r_i is actually an element of $(\Delta \cup X)^*$. We now construct a dct transducer M' which simulates a derivation of M on input tree t by traversing the relabeled t as indicated by the rule-sequences of the derivation; thus $M'(L') = M(L)$.

Let $M' = (Q', \Sigma', \Delta, \delta, q_0', F)$, where Σ' is the set of all $\langle r_1, \ldots, r_s, n\rangle$ such that $0 \leqslant s \leqslant k$, r_i is the right-hand side of a rule in R with states and parentheses deleted and $1 \leqslant n \leqslant \max$ (where max is the maximal rank of an element of Σ); $Q' = F = \{[\text{down}, i] \mid 1 \leqslant i \leqslant k\} \cup \{[\text{up}, i, j] \mid 1 \leqslant i \leqslant k,\ 1 \leqslant j \leqslant \max\}$, and $q_0' = [\text{down}, 1]$; δ will be constructed later.

The strings "down" and "up" are just identifiers to facilitate the specification of δ. Intuitively, if M' is in state $[\text{down}, i]$ at some node, it will start the simulation of the ith translation of the subtree at that node; if M' is in state $[\text{up}, i, j]$ at some node, it has just finished the simulation of the ith translation of the subtree at the jth son of that node. Due to the presence of the rule-sequences and the clear relationship between the rule-sequence of a father and those of its sons, M' can always see in which state it has to be. We now specify δ formally.

(1) Let $q = [\text{down}, i]$ and $\sigma = \langle r_1, \ldots, r_s, n\rangle$.

(1a) If $r_i = w_1 x_m w_2$ for some $w_1 \in \Delta^*$, $m \geqslant 1$, $w_2 \in (\Delta \cup X)^*$, and if this x_m is the jth occurrence of x_m in $r_1 r_2 \cdots r_s$ (i.e., x_m occurs $j - 1$ times in $r_1 \cdots r_{i-1}$), then $\delta(q, \sigma) = ([\text{down}, j], \text{down}(m), w_1)$.

(1b) If $r_i \in \Delta^*$, then $\delta(q, \sigma) = ([\text{up}, i, n], \text{up}, r_i)$.

(2) Let $q = [\text{up}, i, m]$ and $\sigma = \langle r_1, \ldots, r_s, n\rangle$. Let the ith occurrence of x_m in $r_1 \cdots r_s$ occur in r_u, i.e., $r_u = w_1 x_m w_2$ with $w_1, w_2 \in (\Delta \cup X)^*$ and x_m occurs $i - 1$ times in $r_1 r_2 \cdots r_{u-1} w_1$.

(2a) If $w_2 = v_2 x_p v_2'$ for some $v_2 \in \Delta^*$, $p \geqslant 1$ and $v_2' \in (\Delta \cup X)^*$, and if this x_p is the jth occurrence of x_p in $r_1 \cdots r_s$ (i.e., x_p occurs $j - 1$ times in $r_1 \cdots r_{u-1} w_1 x_m$), then $\delta(q, \sigma) = ([\text{down}, j], \text{down}(p), v_2)$.

(2b) If $w_2 \in \Delta^*$, then $\delta(q, \sigma) = ([\text{up}, u, n], \text{up}, w_2)$.

This ends the construction of M'. It should be clear that M' faithfully obeys the indications of the rule-sequences at the nodes of the input tree. Also, if x_m occurs n times in a sequence $r_1, \ldots, r_s$ at some node, then M' crosses n times the arc from that node to its mth son (and back). Hence M' is k-crossing. It should also be clear that M' is k-pass if M is k-metalinear. Formal proofs of these facts are left to the reader. ∎

As indicated in the proof of this theorem, we have simultaneously obtained a proof of Lemma 3.2.3.

We note that if $\mathfrak{L}$ is a class of tree languages closed under finite-state relabelings and insertion of regular languages, then $\text{CT–PD}_{\text{fc}(k)}(\mathfrak{L}) = \text{DCT}_{\text{fc}(k)}(\mathfrak{L})$ and hence $\text{CT–PD}_{\text{fc}}(\mathfrak{L}) = \text{DCT}(\mathfrak{L})$. In fact, by Theorem 4.5 and Theorem 3.3.2, $\text{CT–PD}_{\text{fc}(k)}(\mathfrak{L}) = yRT_{\text{fc}(k)}(\mathfrak{L}) = yT_{\text{fc}(k)}(\mathfrak{L})$ and the above equality follows from Theorem 4.9. Similarly, $\text{CT–PD}_{\text{fp}(k)}(\mathfrak{L}) = \text{DCT}_{\text{fp}(k)}(\mathfrak{L})$. These equalities are in particular true for $\mathfrak{L} = \text{REC}$ and $\mathfrak{L} = \text{REG}$ (and hence for cs–pd transducers). In [46, 35] it is shown that the same class is obtained even when arbitrary printing is allowed on the checking stack, but the machine is still restricted to be finite crossing (or finite pass).

By considering the monadic case of Theorem 4.9 we obtain the following corollary (see Theorem 3.2.2).

(4.10) COROLLARY. *Let $\mathfrak{L}$ be a class of languages closed under sequential machine mappings. Then* $\text{DCS}(\mathfrak{L}) = \text{ETOL}_{\text{FIN}}(\mathfrak{L})$ *and* $\text{DCS}_{\text{fp}}(\mathfrak{L}) = \text{ETOL}_{\text{ml}}(\mathfrak{L})$ *and similarly for the corresponding bounded classes.*

For the recognizable and regular languages the result looks as follows.

(4.11) COROLLARY. (i) $yT_{\text{fc}}(\text{REC}) = \text{DCT}(\text{REC})$ *and* $\text{ETOL}_{\text{FIN}} = \text{DCS}(\text{REG})$,

(ii) $yT_{\text{ml}}(\text{REC}) = \text{DCT}_{\text{fp}}(\text{REC})$ *and* $\text{ETOL}_{\text{ml}} = \text{DCS}_{\text{fp}}(\text{REG})$.

Similar equalities hold for the corresponding bounded classes.

It is not clear whether, in this corollary, the input language of the dct transducer may be restricted to T_Σ as in the previous theorems. For the monadic case this can easily be proved [35, Lemmas 2.1 and 2.2].

The first equality of Corollary 4.11(i) was shown in Section 6 of [4] and the second in [45], both without bounds. The first equality of Corollary 4.10 was shown (with bounds) in [35]. It was also shown there that $\text{DCS}_{\text{fp}}(\mathfrak{L})$ can be obtained by iteration of control on linear context-free grammars. As mentioned already in Section 3, several of the results proved for the ETOL classes in Section 3 (or in [50, 51, 62]) can also be found in the literature for the corresponding DCS classes (see [35, 41, 17, 34, 40]).

We have found sequential machines corresponding to all the classes of top-down tree-to-string transducers and ETOL systems in the diagram of Fig. 3. The corresponding machine diagram is given in Fig. 6, where the indications (REC) and (REG) and the bounds have been left out for reasons of readability.

To this diagram we add the classes CS(REG) and CT(REC) of checking stack and checking tree languages, respectively, for which no corresponding top-down tree-to-string transducers have been found. We also add the diagram of s–pd machines, stack machines and nonerasing stack machines considered in [26], which is contained in the class of macro grammars, recognized by the nested stack machine. This gives the diagram of Fig. 7 which represents the relationships between various classes of s–pd and ct–pd machines that recognize macro languages or tree transformation languages. (To improve readability lowercase letters are used rather than capitals. An ascending line denotes inclusion.)

An s–pd (stack–pushdown) machine is the same as a cs–pd machine, except that it has

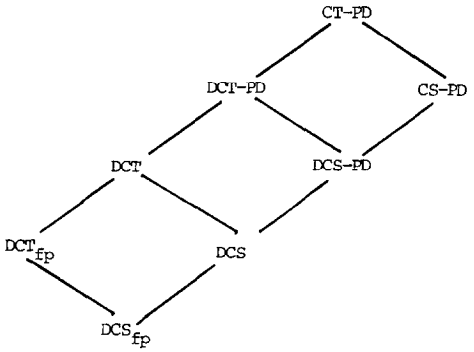

FIG. 6. Classes of output languages of checking transducers (without REC and REG).

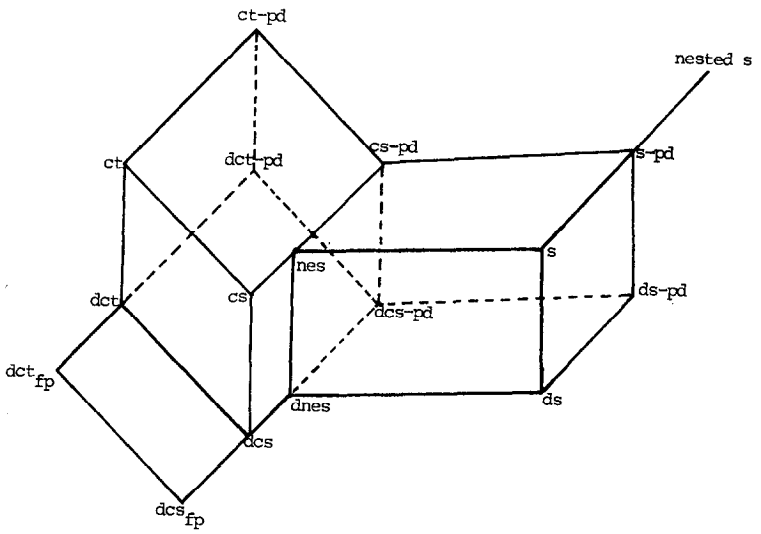

FIG. 7. Classes of tree transformation languages and macro languages.

a usual stack rather than a checking stack. Thus Fig. 7 shows that quite a number of well-known classes of tree transformation and macro (indexed) languages can be obtained by simple variations of one type of machine model.

To prove the correctness of this diagram it suffices (because the correctness of the diagram of Fig. 6 and that of the diagram of [26] are known) to prove the existence of a language L_1 in CS(REG) but not in DCT–PD(REC), and a language L_2 in DNES but not in CT(REC). An example of L_1 is the language $\{w \in \{a, b\}^* \mid$ the number of a's in w is not prime$\}$, which can easily be shown to be in CS(REG) (see [31]), but is not in DCT–PD(REC) $= yDT$(REC) by Theorem 3.2.14 and Corollary 3.2.7. (Actually, we found Theorem 3.2.14 when trying to find L_1). To obtain a language L_2 we generalize a result concerning checking stack languages [41, 35] to trees.

(4.12) LEMMA. *Let $\mathfrak{L}$ be a class of tree languages closed under finite-state relabelings and insertion of regular languages. Let L be a language which contains no infinite regular language. If $L \in \mathrm{CT}(\mathfrak{L})$, then $L \in \mathrm{DCT}(\mathfrak{L})$.*

Proof. Let $L = M(L')$ with $M \in \mathrm{CT}$ and $L' \in \mathfrak{L}$. Since L does not contain an infinite regular language, (M, L') must be finite crossing. In fact, if (M, L') is not finite crossing then a computation of M can be found that crosses twice the same arc of an input tree from L', in the same direction and in the same states. Consequently, the piece of the computation between these two crossings may be repeated an arbitrary number of times. By originally restricting our attention to shortest computations (of an output string) we may assume that the output produced between the two crossings is nonempty. Hence the above repetition gives rise to output strings of the form $w_1 w_2^n w_3$ with $w_2 \neq \lambda$ and $n \geqslant 1$: i.e., an infinite regular subset of L. Consequently $L \in \mathrm{CT}_{\mathrm{fc}}(\mathfrak{L})$. By the remark following Theorem 4.9, $\mathrm{CT}_{\mathrm{fc}}(\mathfrak{L}) = \mathrm{DCT}(\mathfrak{L})$ and the lemma is proved. ∎

Let $L_2 = \{a^{n^2} \mid n \geqslant 1\}$ and assume that this DNES language is in CT(REC). Since L_2 contains no infinite regular subset, Lemma 4.12 implies that $L_2 \in \mathrm{DCT}(\mathrm{REC})$. However, because $\mathrm{DCT}(\mathrm{REC}) = yT_{\mathrm{fc}}(\mathrm{REC})$ by Corollary 4.11 and the languages of $yT_{\mathrm{fc}}(\mathrm{REC})$ are Parikh by Corollary 3.2.7, this leads to a contradiction. This shows the correctness of the diagram of Fig. 7.

Recall the pebble automaton of [4] as discussed before Definition 4.2. It is noted in [4] (without proof) that on monadic trees (i.e., derivation trees of linear context-free grammars) the k-pebble automata are equivalent to EDT0L systems (i.e., GSDT) for which the length of the state-sequence at a node d is $c \cdot n^k$, where n is the distance of d to the root; cf. the remarks on such systems before Theorem 3.2.4. Whether this relation also holds in general is still open. In [4] it is also suggested that stack-languages could be characterized by pebble automata on trees. It follows from the diagram of Fig. 7 that this is not possible in general (for the pebble automata of [4]). It can, however, easily be shown that the nonerasing stack (nes) languages can be produced by 1-pebble cs–pd transducers (see [26]).

5. CLOSURE PROPERTIES

The classes of languages discussed in the previous two sections have nice closure properties. They are closed under most of the usual AFL operations on languages. In the monadic case closure properties have been studied extensively, and in various degrees of generality, in the literature on checking stack automata [31, 35, 55], two-way finite-state transducers [3, 11, 17, 41, 45], ET0L systems [49, 50, 51, 62] and AFL theory [30, 33]. Operations on tree languages and the closely related topic of composition of tree transducers were studied in [48, 56, 9, 18, 19].

In this section we first consider AFL operations. In particular we show that $yT_{\mathrm{fc}(k)}(\mathrm{REC})$ is a full substitution-closed AFL and that $yT(\mathrm{REC})$ is not a hyper-AFL. Then we focus attention on closure under dcs (and cs) translations, i.e., 2-way gsm mappings. It is well known [41, 35] that the application of DCS on a class of languages is idempotent (in fact

dcs translations are closed under composition [11]), implying that DCS(REG) is closed under dcs translations. We generalize this by showing that deterministic top-down Ω-tree transformation languages (in particular yDT(REC) and EDT0L) are closed under dcs translations. The result is obtained by a straightforward simulation of a 2-way string (dcs) transducer working on the yield of a tree, by a 2-way tree-to-string (dct) transducer walking on the tree itself, and using the fact that DT^R tree translations are closed under composition [19].

We shall consider the AFL closure properties of all the classes in the diagram of Fig. 3, including the bounded ones. It is left to the interested reader to see how the results can be generalized to an arbitrary class of input languages (cf. [41, 35, 9]). It is easy to see that all these classes are closed under union and homomorphisms. Also, they are all closed under intersection with a regular language; in fact, for a top-down tree transformation system (M, L) and any regular language R, $M(L) \cap R = M(L \cap M^{-1}(R))$ and, since $M^{-1}(R)$ is recognizable [43], $L \cap M^{-1}(R)$ is a new recognizable input language to the same transducer M. The nonmonadic classes are easily seen to be closed under concatenation and, except for the metalinear ones, under Kleene star (by applying respectively the operations $\sigma(L_1L_2)$ and $\sigma(L\sigma(L \cdots \sigma(LL) \cdots))$ for some new σ of rank 2 to the input languages).

The classes yT(REC) and ET0L are known to be substitution-closed full AFL's [9, 49]. ET0L is even a full hyper-AFL [10, 53], i.e., closed under iterated substitution [7], but yT(REC) is not, as will be shown in Theorem 5.2 and Corollary 5.3 (using insertion of regular languages it is easy to see that we may allow the yT transducer to have ET0L right-hand sides, but any class larger than ET0L is beyond the power of yT(REC)). ET0L is a full-principal AFL (cf. [12]) and the same is true for yT(REC). In fact the ct–pd machine viewed as an acceptor for yT(REC) languages (Corollary 4.6) can easily be formulated as a finitely encoded AFA (note that all languages in yT(REC) can be produced by transducers with the fixed input alphabet $\Sigma = \{0, 1\} = \Sigma_0 = \Sigma_1 = \Sigma_2$ by a straightforward coding argument).

As note above the classes yDT(REC), EDT0L, $yT_{(k)}$(REC), ET0L$_{(k)}$, $yDT_{(k)}$(REC), and EDT0L$_{(k)}$ are closed under all AFL operations except inverse homomorphism; yDT(REC) and EDT0L are also closed under deterministic gsm mappings (even 2-way, as will be shown later). It follows from Theorem 3.2.14 that yDT(REC) and EDT0L are not closed under inverse homomorphism. The above-mentioned bounded classes are not closed under deterministic gsm mappings (and hence not under inverse homomorphism); in fact, for each k the language $\{(a^nb)^{2^k} \mid n \geqslant 0\}$ is generated by an EDT0L$_{(1)}$ system with rules $q_0(\sigma x) \to (q_1(x)b)^{2^k}$, $q_1(\sigma x) \to aq_1(x)$, $q_1(\tau x) \to \lambda$ and this language can be transformed by a deterministic gsm into the language $L_k = \{a_1{}^n a_2{}^n \cdots a_{2^k}^n \mid n \geqslant 0\}$ which is not in $yT_{(k-1)}$(REC) by Theorem 3.2.5.

We now state the closure properties of the finite-copying and metalinear classes.

(5.1) THEOREM. *Let* $k \geqslant 1$.

(i) $yT_{\mathrm{fc}(k)}$(REC) *and* yT_{fc}(REC) *are substitution-closed full* AFL's; *the latter is not full principal.*

(ii) $yT_{\mathrm{ml}(k)}(\mathrm{REC})$ *and* $yT_{\mathrm{ml}}(\mathrm{REC})$ *are concatenation-closed full semi-*AFL's; *the latter is not full principal.*

(iii) $\mathrm{ET0L}_{\mathrm{FIN}(k)}$ *and* $\mathrm{ET0L}_{\mathrm{ml}(k)}$ *are full semi-*AFL's; $\mathrm{ET0L}_{\mathrm{FIN}}$ *is a substitution-closed full* AFL *and* $\mathrm{ET0L}_{\mathrm{ml}}$ *is a concatenation-closed full semi-*AFL, *both not full principal.*

Proof. For a proof of (iii) the reader is referred to the "monadic" literature [35, 45, 50, 62]. To prove (i) and (ii) it suffices (by the discussion preceding this theorem) to show that these classes are closed under regular substitution (this follows directly from Lemma 3.3.2) and that $yT_{\mathrm{fc}(k)}(\mathrm{REC})$ is substitution closed. Note that nonprincipality follows from the hierarchy result of Theorem 3.2.5.

We argue now that $yT_{\mathrm{fc}(k)}(\mathrm{REC})$ is closed under substitution. Let (M, L) be a deterministic top-down tree transformation system of copying-bound k with $M = (Q, \Sigma, \Delta, q_0, R)$ and $L \in \mathrm{REC}$. Let, for each $a \in \Delta$, $M_a(L_a)$ be a language in $yT_{\mathrm{fc}(k)}(\mathrm{REC})$. To obtain a top-down tree transformation system (M', L') with copying-bound k, which generates the substitution of the L_a for a in L, we first change L into L' as follows. Let $\Delta = \{a_1, \ldots, a_n\}$ and assume (without loss of generality) that a right-hand side of a rule in R contains at most one occurrence of each symbol of Δ. L' is obtained from L by adding nk additional subtrees at each node of a tree $t \in L$, such that the ith k-tuple of new subtrees consists of trees in L_{a_i}; correspondingly the ranks of the elements of Σ are increased by nk. Clearly L' is recognizable. The new transducer M' simulates M and, whenever M outputs some symbol $a \in \Delta$, it simulates the corresponding M_a on one of the additional trees of L_a. More precisely, suppose that M' arrives at the jth copy of a node of (the modified) t, $1 \leqslant j \leqslant k$, and that M would output a_i at this step; then we want M' to operate as M_{a_i} on the jth element of the ith k-tuple of the additional subtrees. In order to know at which copy of the node M' arrives, M' keeps track of the state-sequence of M at the nodes of the input tree and also its position in this state-sequence (which can easily be done due to the determinism of M). In this way the nodes in the additional subtrees are copied at most k times, and thus M' will have copying-bound k on L'. ∎

Note that AFA formulations of all these classes can easily be derived from the corresponding machines in Section 4 (cf. [41] for the monadic case).

Note also that, due to these closure properties, the counterexamples of Theorem 3.2.5 can now be changed into languages over a two-letter alphabet; thus $\{(a^n b)^{2k} \mid n \geqslant 0\}$ is in $\mathrm{ET0L}_{\mathrm{ml}(k)}$ but not in $yT_{\mathrm{fc}(k-1)}(\mathrm{REC})$.

It follows from Theorem 5.1 and Theorem 3.2.14 that $yT_{\mathrm{fc}}(\mathrm{REC})$ is the largest full AFL (even the largest class closed under inverse homomorphism) inside $yDT(\mathrm{REC})$, and that $\mathrm{ET0L}_{\mathrm{FIN}}$ is the largest full AFL inside EDT0L. Similarly it follows from Corollary 3.2.12 that CF is the largest full AFL inside $yT_{\mathrm{ml}}(\mathrm{REC})$ and REG the largest in $\mathrm{ET0L}_{\mathrm{ml}}$. The results on $\mathrm{ET0L}_{\mathrm{ml}(k)}$ and $\mathrm{ET0L}_{\mathrm{FIN}(k)}$ are also optimal: they are not closed under concatenation [62].

The difference between (i) and (iii) of Theorem 5.1 shows the power of unbounded rank of the input symbols (as used in the proof of Theorem 5.1). On the other hand, it is known that $\mathrm{ET0L}_{\mathrm{FIN}(k)}$ and $\mathrm{ET0L}_{\mathrm{ml}(k)}$ are full principal for each k [35, 50, 62], but in the tree case this is an open question. We do not know whether there exists a fixed

input alphabet Σ such that all languages in $yT_{fc(k)}$(REC) can be obtained from transformation systems with this input alphabet (this would give principality by finite encoding of the corresponding dct transducers). The trick that can be used in the case of yT(REC) to replace each node of rank n by $n-1$ nodes of rank 2, increases the copying-bound of the transformation system. If the above question could be answered positively then we could even find (by a refinement of the proof of Theorem 4.9) one transducer $M \in yT_{fc(k)}$ such that $yT_{fc(k)}(\text{REC}) = \{M(L) \mid L \in \text{REC}\}$; see Section 7 of [50] for the monadic case of this result. An alternative solution would of course be to restrict all trees to have maximal rank 2. By the trick mentioned it follows that this restriction does not influence the union families. One can show straightforwardly that $yT_{fc(k)}(\text{REC}_2)$ is a full-principal AFL (where the subscript 2 denotes the restriction) and that a single transducer can be found which generates it by varying the input language. It is not clear any more whether this rank-restricted class is substitution closed. Note finally that in the terminology of GSDT this discussion amounts to the question whether, for GSDT with copying-bound k, the underlying context-free grammar can always be taken in Chomsky normal form (cf. [2]).

By methods similar to those in the proof of Theorem 5.1(i) it can be shown that $yT_{fc(k)}$(REC), yT_{fc}(REC), and yT(REC) are even super-AFL's [32], i.e., closed under nested iterated substitution. In the next theorem we show that the nesting is essential, i.e., that these classes are not hyper-AFL's.

(5.2) THEOREM. *Let L be a language over alphabet A, b a letter not in A, and let $f(L)$ denote the language $\{a_1 w a_2 w \cdots a_n w \mid a_i \in A,\ a_1 a_2 \cdots a_n \in L,\ w \in b^*\}$. If $f(L) \in yT$(REC), then $L \in$ ET0L.*

Proof. Let $f(L) = M(K)$ with $K \in$ REC and $M = (Q, \Sigma, \Delta, q_0, R)$ in yT. A sketch of the proof is as follows. We note first that, since yT(REC) and ET0L are both closed under gsm mappings, we may assume that no string of L contains two consecutive occurrences of the same symbol.

Suppose now that a rule of the form $q(\sigma(x_1 \cdots x_m)) \to \cdots p_1(x_i) \cdots p_2(x_j) \cdots$ with $i \neq j$ is used in a derivation of a string $a_1 w a_2 w \cdots a_n w$ for some very long w, such that $p_1(t_i) \overset{*}{\Rightarrow} \cdots a_r w a_{r+1} \cdots$ and t_j is very high (where $t_1, t_2, \ldots, t_m$ are the direct subtrees of the node to which the rule is applied). We will argue that such a situation cannot happen. Let us first change all subderivations which operate on t_j, in such a way that the same rule is applied when M arrives in the same state at different copies of the same node of t_j, i.e., all derivations on t_j are made "deterministic" ("uniform" in the terminology of [44]). This also changes the generated string $a_1 w \cdots a_n w$, however, since the derivation on t_i is kept fixed, the new string still "has the same w." It should be clear that to such (sub)derivations the pumping Theorem 3.2.4 is applicable (see [44]). Hence, since t_j is high, we can pump the substrings generated by the derivations on t_j, and moreover (because w is long and the substrings are short, and because of the first assumption of our proof) it follows from the preservation of the alphs that only the w's are pumped. This contradicts the fact that the w is kept fixed by the derivation on t_i. Hence such a rule is not needed.

By putting the appropriate information on the nodes of the input tree by a finite-state relabeling and directly "substituting" derivations that produce at most one element of A and derivations on small subtrees, we can therefore obtain the language L as $M'(K')$, where M' contains only rules of the form $q(\sigma(x_1 \cdots x_n)) \to v$ or $q(\sigma(x_1 \cdots x_n)) \to v_1 p(x_i) v_2$ with $v, v_1, v_2 \in \Delta^*$. It is straightforward to turn such a yT transducer into an ET0L system. ▮

(5.3) COROLLARY. *yT(REC) is not a hyper-*AFL. ET0L *is the only hyper-*AFL *included in yT*(REC).

Proof. Let $\mathfrak{L} \subseteq yT$(REC) be a hyper-AFL, i.e., closed under iterated substitution. We shall show that $\mathfrak{L} \subseteq$ ET0L (and hence $\mathfrak{L} =$ ET0L and the corollary is proved). Let $L \in \mathfrak{L}$. Let h be the homomorphism such that $h(a) = ab$ for all $a \in A$ and $h(b) = b$, where A is the alphabet of L. Then $\bigcup_{n \geq 0} h^n(L)$, i.e., the iterated application of h to L, is equal to $f(L)$, as defined in Theorem 5.2. Hence, since $\mathfrak{L}$ is a hyper-AFL, $f(L) \in \mathfrak{L}$. It now follows from Theorem 5.2 that $L \in$ ET0L. Hence $\mathfrak{L} \subseteq$ ET0L. ▮

We note that this property distinguishes yT(REC) rather sharply from the class of indexed languages. The latter is a full hyper-AFL containing a proper hierarchy of full hyper-AFL's [22, 27].

We also note that it can be shown in a similar way that yDT(REC) is not closed under "deterministic substitution" [8] and that EDT0L is the largest class inside yDT(REC) closed under that operation.

In the rest of this section we shall consider closure under dcs (and cs) transducers, i.e., closure under 2-way gsm mappings. We shall make an essential use of the following composition result on top-down tree transducers with regular look-ahead.

(5.4) THEOREM. *DT^R, DT^R_{fc} and DT^R_{ml} are closed under composition. In particular, for $k, n \geqslant 1$, $DT^R_{\text{fc}(k)} \circ DT^R_{\text{fc}(n)} \subseteq DT^R_{\text{fc}(kn)}$ and similarly for* ml.

Proof. Closure of DT^R under composition is proved in [19]. Unfortunately the proof does not preserve finite copying. It is, however, straightforward to prove this result directly by the standard method of applying a transducer M_2 to the right-hand sides of the rules of a transducer M_1. The regular look-ahead of M_1 and M_2 can easily be combined (using the fact that $M^{-1}(L)$ is recognizable if L is), and the eventual problem that M_1 deletes subtrees to be checked by M_2 can be handled by an additional regular look-ahead. Clearly, if M_1 copies a node k times and M_2 copies each of these copies n times, then the newly constructed transducer will copy the original node kn times. It is easy to see that metalinearity is preserved. A formal proof is left to the reader. ▮

Together with Lemma 3.2.3, Theorem 5.4 shows that if $\mathfrak{L}$ is a class of tree languages closed under finite-state relabelings (in particular if $\mathfrak{L} =$ REC), then $T_{\text{fc}}(\mathfrak{L})$ and $T_{\text{ml}}(\mathfrak{L})$ are closed under T_{fc} and T_{ml} tree translations, respectively. This result can be viewed (and we will show this later) as a generalization of the monadic case [41, 35, 11]: if $\mathfrak{L}$ is, say, a full semi-AFL, then DCS($\mathfrak{L}$) is closed under dcs translations and $\text{DCS}_{\text{fp}}(\mathfrak{L})$ under dcs_{fp} translations (Corollary 5.8).

The essential construction in the proof of the closure of tree transformation languages under dcs transducers is the simulation of a dcs transducer on the yield of a tree by a dct transducer on that tree, as given in the next theorem.

(5.5) THEOREM. *If $\mathfrak{L}$ is a class of tree languages closed under deterministic top-down finite-state relabelings, then* $\mathrm{DCS}(y\mathfrak{L}) \subseteq yDT^R_{\mathrm{fc}}(\mathfrak{L})$ *and* $\mathrm{DCS}_{\mathrm{fp}}(y\mathfrak{L}) \subseteq yDT^R_{\mathrm{ml}}(\mathfrak{L})$, *and in particular, for* $k \geqslant 1$, $\mathrm{DCS}_{\mathrm{fc}(k)}(y\mathfrak{L}) \subseteq yDT^R_{\mathrm{fc}(2k)}(\mathfrak{L})$, *and similarly for* fp *and* ml. *If $\mathfrak{L}$ is closed under arbitrary finite-state relabelings, then* $\mathrm{DCS}(y\mathfrak{L}) \subseteq yT_{\mathrm{fc}}(\mathfrak{L})$ *and* $\mathrm{DCS}_{\mathrm{fp}}(y\mathfrak{L}) \subseteq yT_{\mathrm{ml}}(\mathfrak{L})$ *and bounds are doubled as before. If $\mathfrak{L}$ is closed moreover under insertion of regular languages, then* $\mathrm{CS}(y\mathfrak{L}) \subseteq yT(\mathfrak{L})$.

Proof. Let $L \in \mathfrak{L}$ be a tree language over Σ and let $M = (Q, \Sigma_0, \Delta, \delta, q_0, F)$ be a cs transducer with input alphabet Σ_0. We first change L into L' by the (deterministic top-down) finite-state relabeling N that relabels σ by $\langle \sigma, i \rangle$ if the corresponding node is the ith son of its father (or, arbitrary, $i = 1$ if it is the root). Since M works on strings we let δ be a function from $Q \times \Sigma_0$ into the finite subsets of $Q \times \{\text{left, stay, right}\} \times \Delta^*$. The simulation of M be a ct transducer M' proceeds as follows. If M is at a leaf of the tree, then M' is at the same leaf (and stays there if M does). If M moves one leaf to the right, then M' moves so to, say, a little bit to the right of the leaf (see Fig. 8) and moves up keeping the arcs of the tree at its left hand; after some time it will move down again, still keeping the arcs at his left hand, until it arrives at a leaf (where M is already waiting!). Similarly, if M moves left, then M' moves a little bit left and then moves up and down again, keeping the arcs at his right hand this time. Clearly if M is deterministic, then so is M'. For a fixed arc of the input tree t with subtree t_0 hanging down from that arc, if $\mathrm{yield}(t) = w_1\,\mathrm{yield}(t_0)w_2$ and M crosses the boundary between w_1 and $\mathrm{yield}(t_0)$ n_1 times and the one between $\mathrm{yield}(t_0)$ and w_2 n_2 times, then the arc is crossed $n_1 + n_2$ times.

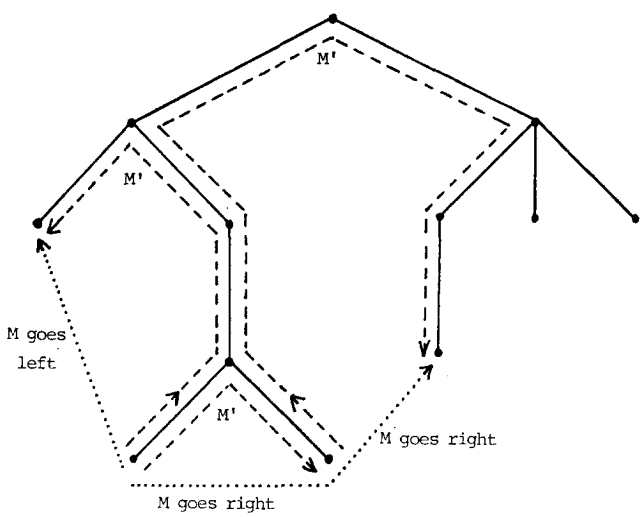

FIG. 8. Simulation of a cs transducer by a ct transducer.

Hence if M is k-crossing, then M' is $2k$-crossing. In the finite-pass case the construction should be slightly changed so that M' will move up to the root and down again whenever it is at the leftmost or rightmost leaf of the tree (we leave this to the reader).

The formal construction of a ct transducer M' such that $M'(L') = M(L)$ is as follows. Let $M' = (Q', \Sigma \times \{1, 2, \ldots, \max\}, \Delta, \delta', q_0', F)$, where max is the maximal rank of a symbol in Σ, $Q' = Q \cup (Q \times \{1, \ldots, \max\} \times \{\text{left, right}\} \times \{\text{up, down}\})$, $q_0' = \langle q_0, -, \text{right, down}\rangle$ ("$-$" denotes an arbitrary element of $\{1, \ldots, \max\}$) and δ' is defined by the following requirements.

If $\delta(q_1, \sigma)$ contains (q_2, left, w), then $\delta'(q_1, \langle\sigma, j\rangle)$ contains $(\langle q_2, j, \text{left, up}\rangle, \text{up}, w)$.

If $\delta(q_1, \sigma)$ contains (q_2, stay, w), then $\delta'(q_1, \langle\sigma, j\rangle)$ contains (q_2, stay, w).

If $\delta(q_1, \sigma)$ contains (q_2, right, w), then $\delta'(q_1, \langle\sigma, j\rangle)$ contains $(\langle q_2, j, \text{right, up}\rangle, \text{up}, w)$.

Moreover,

$$\begin{aligned}
&\delta'(\langle q, i, \text{left, up}\rangle, \langle\sigma, j\rangle)\\
&\quad = (\langle q, j, \text{left, up}\rangle, \text{up}, \lambda) && \text{if } i = 1,\\
&\quad = (\langle q, -, \text{left, down}\rangle, \text{down}(i-1), \lambda) && \text{if } i > 1,\\
&\delta'(\langle q, -, \text{left, down}\rangle, \langle\sigma, j\rangle)\\
&\quad = (\langle q, -, \text{left, down}\rangle, \text{down}(n), \lambda) && \text{if } \sigma \in \Sigma_n,\ n \geqslant 1,\\
\text{and}\quad &\quad = (q, \text{stay}, \lambda) && \text{if } \sigma \in \Sigma_0,\\
&\delta'(\langle q, i, \text{right, up}\rangle, \langle\sigma, j\rangle)\\
&\quad = (\langle q, j, \text{right, up}\rangle, \text{up}, \lambda) && \text{if } \sigma \in \Sigma_n \text{ and } i = n,\\
&\quad = (\langle q, -, \text{right, down}\rangle, \text{down}(i+1), \lambda) && \text{if } \sigma \in \Sigma_n,\ i < n,\\
&\delta'(\langle q, -, \text{right, down}\rangle, \langle\sigma, j\rangle)\\
&\quad = (\langle q, -, \text{right, down}\rangle, \text{down}(1), \lambda) && \text{if } \sigma \in \Sigma_n,\ n \geqslant 1,\\
&\quad = (q, \text{stay}, \lambda) && \text{if } \sigma \in \Sigma_0.
\end{aligned}$$

Note that, when going up, M' knows from which son it came. It should be clear now that $M'(L') = M(L)$. The first part of the theorem follows now from Theorem 4.7, the second part from Lemma 3.3.4 and the third part from Corollary 4.6. ∎

By taking $\mathfrak{L} = \text{REC}$ it follows that the images of the context-free languages under 2-way gsm mappings are contained in the top-down tree transformation languages.

(5.6) COROLLARY.

$$\text{ET0L}_{\text{ml}} \subsetneq \text{DCS}_{\text{fp}}(\text{CF}) \subseteq yT_{\text{ml}}(\text{REC}).$$
$$\text{ET0L}_{\text{FIN}} \subsetneq \text{DCS}(\text{CF}) \subseteq yT_{\text{fc}}(\text{REC}).$$
$$\text{CS}(\text{REG}) \subsetneq \text{CS}(\text{CF}) \subseteq yT(\text{REC}).$$

Proof. The inclusions follow from Corollary 4.11 and the second part of the previous theorem. Proper inclusions follow from the existence of a context-free language not in EDT0L. ∎

We conjecture that the remaining inclusions of this corollary are also proper.

In [35] it is shown that, for any full semi-AFL $\mathfrak{L}$, $\mathrm{DCS}_{\mathrm{fp}}(\mathfrak{L})$ can be obtained by iterating the process of putting control on linear context-free grammars, starting with control from $\mathfrak{L}$, such that the kth iteration corresponds to 2^{k-1} passes, i.e., $\mathrm{DCS}_{\mathrm{fp}}(\mathfrak{L}) = \bigcup_n \mathrm{DCS}^n_{\mathrm{fp}(1)}(\mathfrak{L})$ or $\mathrm{ET0L}_{\mathrm{ml}}(\mathfrak{L}) = \bigcup_n \mathrm{ET0L}^n_{\mathrm{ml}(1)}(\mathfrak{L})$. Consequently $\mathrm{DCS}_{\mathrm{fp}}(\mathrm{CF})$ is equal to the hierarchy $\bigcup_k \mathfrak{L}_k$ considered in [39], and in particular $\mathfrak{L}_k = \mathrm{DCS}_{\mathrm{fp}(2^{k-1})}(\mathrm{CF})$. Hence by the previous theorem and its corollary, $\mathfrak{L}_k \subseteq yT_{\mathrm{ml}(2^k)}(\mathrm{REC})$. Several results in [39] can be understood in this light, in particular the examples to prove the hierarchy proper, which are essentially the L_{2^k} of Theorem 3.2.5.

It is shown in [35, 62] that $\mathrm{DCS}_{\mathrm{fp}}(\mathfrak{L})$ is the closure of $\mathfrak{L}$ under homomorphic replications [30, 33]. This implies that $\mathrm{DCS}_{\mathrm{fp}}(\mathrm{CF})$ contains the simple matrix languages of [37] and the controlled pushdown automata languages of [38]. In [38] it is shown that $\mathrm{DCS}_{\mathrm{fp}}(\mathrm{CF})$ is a proper hierarchy, using the counterexamples of Theorem 3.2.5.

In the next theorem we state the main result of this section.

(5.7) THEOREM. *If $\mathfrak{L}$ is a class of tree languages closed under finite-state relabelings, then $yDT(\mathfrak{L})$ and $yT_{\mathrm{fc}}(\mathfrak{L})$ are closed under* dcs *transducers, and $yT_{\mathrm{ml}}(\mathfrak{L})$ is closed under finite-pass* dcs *transducers. In particular, for $k, m \geq 1$, $\mathrm{DCS}_{\mathrm{fc}(m)}(yT_{\mathrm{fc}(k)}(\mathfrak{L})) \subseteq yT_{\mathrm{fc}(2mk)}(\mathfrak{L})$ and similarly for* fp *and* ml.

Proof. Recall that $yT_{\mathrm{fc}}(\mathfrak{L}) = yDT_{\mathrm{fc}}(\mathfrak{L})$ be Lemma 3.2.3, and similarly for ml. Since $\mathfrak{L}$ is closed under finite-state relabelings, the classes $DT(\mathfrak{L})$, $DT_{\mathrm{fc}(k)}(\mathfrak{L})$ and $\mathrm{DT}_{\mathrm{ml}(k)}(\mathfrak{L})$ are closed under deterministic top-down finite-state relabelings (which are in $\mathrm{DT}_{\mathrm{ml}(1)}$) by Theorem 5.4 and Lemma 3.3.4. Hence the first part of Theorem 5.5 is applicable. Thus $\mathrm{DCS}(yDT(\mathfrak{L})) \subseteq yDT^R_{\mathrm{fc}}(DT(\mathfrak{L})) \subseteq yDT(\mathfrak{L})$ by Theorem 5.4 and Lemma 3.3.4. Similarly, $\mathrm{DCS}_{\mathrm{fc}(m)}(yDT_{\mathrm{fc}(k)}(\mathfrak{L})) \subseteq yDT^R_{\mathrm{fc}(2m)}(DT_{\mathrm{fc}(k)}(\mathfrak{L})) \subseteq yDT_{\mathrm{fc}(2mk)}(\mathfrak{L})$, and analogously for fp and ml. ∎

Note that $yT_{\mathrm{ml}}(\mathrm{REC})$ is not closed under dcs transducers by Corollary 3.2.13.

The monadic case of the above theorem is stated next.

(5.8) COROLLARY. *If $\mathfrak{L}$ is a class of languages closed under sequential machine mappings, then* $\mathrm{EDT0L}(\mathfrak{L})$ *and* $\mathrm{ET0L}_{\mathrm{FIN}}(\mathfrak{L})$ *are closed under* dcs *transducers, and* $\mathrm{ET0L}_{\mathrm{ml}}(\mathfrak{L})$ *is closed under finite-pass* dcs *transducers.*

From $\mathrm{ET0L}_{\mathrm{FIN}}(\mathfrak{L}) = \mathrm{DCS}(\mathfrak{L})$ we obtain that $\mathrm{DCS}_{\mathrm{fc}(k)}(\mathrm{DCS}_{\mathrm{fc}(m)}(\mathfrak{L})) \subseteq \mathrm{DCS}_{\mathrm{fc}(2km)}(\mathfrak{L})$, and similarly for $\mathrm{DCS}_{\mathrm{fp}}$ (as proved in [35], see also [41]). The results on ET0L can also be understood by saying that $\mathrm{ET0L}_{\mathrm{FIN}}$ and $\mathrm{ET0L}_{\mathrm{ml}}$ are "closed under control," i.e., $\mathrm{ET0L}_{\mathrm{FIN}}(\mathrm{ET0L}_{\mathrm{FIN}}) = \mathrm{ET0L}_{\mathrm{FIN}}$ and $\mathrm{ET0L}_{\mathrm{ml}}(\mathrm{ET0L}_{\mathrm{ml}}) = \mathrm{ET0L}_{\mathrm{ml}}$.

Thus $yDT(\mathrm{REC})$, $yT_{\mathrm{fc}}(\mathrm{REC})$, EDT0L and $\mathrm{ET0L}_{\mathrm{FIN}}$ are all closed under deterministic 2-way gsm mappings and $\mathrm{ET0L}_{\mathrm{FIN}}$ is the smallest such class containing the regular languages. Similarly $yT_{\mathrm{ml}}(\mathrm{REC})$, $\mathrm{DCS}_{\mathrm{fp}}(\mathrm{CF})$ and $\mathrm{ET0L}_{\mathrm{ml}}$ are closed under finite-pass dcs transducers, and $\mathrm{ET0L}_{\mathrm{ml}}$ is the smallest such class containing REG.

We finally note that similar results cannot be obtained for CS–PD or DCS–PD. For

CS–PD, it follows directly from the copying theorems in [24] that ET0L (and yT(REC)) is not closed under dcs transducers. It is known that the language $\{(a^n b)^{2^n} \mid n \geqslant 0\}$ is in DCS–PD(LIN)[7], but not in EDT0L. Consequently, DCS–PD(REG), which is closed under dcs transducers by Corollary 5.8, is not closed under dcs–pd transducers, and thus dcs–pd transducers are not closed under composition, or equivalently EDT0L is not closed under control.

6. Macro Grammars

A rule $q(\sigma x) \to w_1 q_1(x)\, w_2 q_2(x) \cdots w_n q_n(x) w_{n+1}$ of an ET0L system can be interpreted in two ways: as a rewriting rule (as we did until now), but also as part of a fixed point equation saying that if v_i is the q_i-translation of x then $w_1 v_1 w_2 v_2 \cdots w_n v_n w_{n+1}$ is the q-translation of σx. The same holds for top-down tree transducers. Macro grammars [28] are an appropriate tool to compute the fixed point of an ET0L system, in a stepwise fashion. In [13] is was shown that the linear basic macro grammars compute precisely all EDT0L fixed points, whereas the extended version of these grammars compute the ET0L fixed points (we note that, dually, EDT0L systems may be viewed as computing the fixed point of linear basic macro grammars, cf. [25]). Thus, as can also be seen from the diagram of Fig. 7, ET0L systems are a particular case of both tree transformation systems and macro grammars (the monadic and the linear case, respectively).

The reason that we include a discussion of macro grammars in this paper is that they can be generalized in a straightforward way such that they are able to generate the top-down tree transformation languages. Actually, the generalized model is a particular type of bottom-up tree transducer (of which the linear basic macro grammar is the monadic case). Note that the fixed-point computation of an EDT0L language may be viewed as a bottom-up process; in [5, Vol. II] the working of a GSDT is actually defined by a bottom-up algorithm.

Thus, both macro grammars and 2-way automata can be considered (after appropriate generalization) as a general framework for the classes of Fig. 7.

In this section we relate the bounds on ET0L systems, as studied in Section 3, to natural bounds on macro grammars. We show that the state-bound (and copying-bound) of an ET0L system corresponds to the maximal number of arguments of the nonterminals in a macro grammar. For each k there exists an OI macro language that can be generated with nonterminals of k arguments but not less. In fact the linear basic language L_{k+1} of Theorem 3.2.5 is such a language. Hence the number of arguments of the macro grammar gives rise to a proper hierarchy in the class of OI macro languages (see [47]). It turns out that ET0L_{FIN} systems correspond to noncopying linear basic macro grammars, whereas to obtain ET0L_{ml} no nonterminal of the macro grammar should "combine" some of its arguments into one argument of another nonterminal (by the application of a rule). This shows that the bounds on ET0L systems considered in the previous sections are also natural from the macro point of view.

We shall treat EDT0L and its correspondence to the linear basic macro grammars, and leave the case of ET0L to the reader.

(6.1) DEFINITION. A *linear basic macro grammar* is a 4-tuple $G = (N, \Delta, S, R)$, where N is a ranked alphabet of nonterminals, Δ is the terminal alphabet, $S \in N$ is the initial nonterminal (of rank 0), and R is a finite set of rules of one of the following forms.

(a) $A(x_1, \ldots, x_m) \to w_0 B(w_1, \ldots, w_n) w_{n+1}$ with $n, m \geqslant 0$, $A \in N_m$, $B \in N_n$, and $w_0, w_1, \ldots, w_{n+1} \in (X_m \cup \Delta)^*$.

(b) $A(x_1, \ldots, x_m) \to w$ with $m \geqslant 0$, $A \in N_m$, and $w \in (X_m \cup \Delta)^*$.

Sentential forms of G are of the form $v_0 A(v_1, \ldots, v_m) v_{m+1}$ with $A \in N_m$ and $v_0, v_1, \ldots, v_{m+1} \in \Delta^*$. Application of a rule $A(x_1, \ldots, x_m) \to t$, denoted by $\Rightarrow$, to this sentential form results in the new sentential form $v_0 t[v_1, \ldots, v_m] v_{m+1}$. Derivations are denoted by $\overset{*}{\Rightarrow}$. The *language generated* by G is $L(G) = \{v \in \Delta^* \mid S^* \overset{*}{\Rightarrow} v\}$. ▮

The class of languages generated by linear basic macro grammars is denoted by LB. We now define the relevant restrictions on these grammars.

(6.2) DEFINITION. Let $G = (N, \Delta, S, R)$ be a linear basic macro grammar. For $k \geqslant 0$, G is *k-argument* if all its nonterminals are of rank at most k. G is *iterative* if in all its rules of type (a), see Definition 6.1, $w_0 = w_{n+1} = \lambda$. G is *double linear* if each x_i occurs at most once in the right-hand side of a given rule. G is *triple linear* if it is double linear and in each rule of type (a) each w_i $(1 \leqslant i \leqslant n)$ contains at most one element of X_m, and $w_0, w_{n+1} \in \Delta^*$. ▮

"k-argument" will be denoted by subscript (k), "iterative" by I and "double" and "triple linear" by L^2 and L^3, respectively. Thus $IL^2B_{(k)}$ denotes the class of languages generated by iterative k-argument double linear basic macro grammars.

We note that ILB = LB, $IL^2B = L^2B$ and $IL^3B = L^3B$. (Provide each nonterminal with two new arguments x_0 and x_∞ and change a rule of type (a) into $A(x_0, x_1, \ldots, x_m, x_\infty) \to B(x_0 w_0, w_1, \ldots, w_n, w_{n+1} x_\infty)$ and a rule of type (b) into $A(x_0, x_1, \ldots, x_m, x_\infty) \to x_0 w x_\infty$). This is not true for the k-argument classes.

The correspondence between the bounds on EDT0L systems and those on linear basic macro grammars is stated in the next theorem.

(6.3) THEOREM. (i) *For* $k \geqslant 1$, $EDT0L_{(k)} = ILB_{(k)}$, $ET0L_{FIN(k)} = IL^2B_{(k)}$ *and* $ET0L_{ml(k)} = IL^2B_{(k)}$.

(ii) $EDT0L = LB$, $ET0L_{FIN} = L^2B$ *and* $ET0L_{ml} = L^3B$.

Proof. It suffices to show (i). We shall use refinements of the ideas of [13].

To prove that $EDT0L_{(k)} \subseteq ILB_{(k)}$, let $M = (Q, \Sigma, \Delta, q_0, R)$ be an $EDT0L_{(k)}$ system. To simulate M by an ILB grammar we shall use the state-sets of M as nonterminals and for each state in the state-set we will keep a string derivable from this state in an argument of the nonterminal. Thus the grammar will be k-argument. Assume that the elements of Q are ordered in some fixed way; we shall always write the elements of a subset of Q in that order. Construct the ILB grammar $G = (N, \Delta, S, R_G)$, where N is the set of all subsets of Q with cardinality at most k (the rank of a subset is its cardinality), $S = \varnothing$ and R_G contains the rule $\{q_1, \ldots, q_r\}(x_1, \ldots, x_r) \to \{p_1, \ldots, p_s\}(w_1, \ldots, w_s)$ iff there

exists $\sigma \in \Sigma$ and $u_1, \ldots, u_s \in (\Delta \cup \{q_1(x), \ldots, q_r(x)\})^*$ such that, for $1 \leqslant i \leqslant s$, $p_i(\sigma x) \to u_i$ is in R and w_i is the result of replacing each $q_j(x)$ by x_j in u_i (for $1 \leqslant j \leqslant r$), i.e., $u_i = w_i[q_1(x), \ldots, q_r(x)]$; moreover R_G contains the rule $\{q_0\}(x) \to x$.

It can be proved by induction that $\varnothing \overset{*}{\Rightarrow} \{q_1, \ldots, q_r\}(v_1, \ldots, v_r)$ in G (with $v_i \in \Delta^*$) if and only if there exists $v \in \Sigma^*$ such that $q_i(v) \overset{*}{\Rightarrow} v_i$ (for $1 \leqslant i \leqslant r$) and the derivation $q_1(v) \cdots q_r(v) \overset{*}{\Rightarrow} v_1 \cdots v_r$ has state-bound k. This shows that $L(G) = L(M)$.

The proof that $\mathrm{EDT0L}_{\mathrm{FIN}(k)} \subseteq \mathrm{IL}^2\mathrm{B}_{(k)}$ is entirely similar. This time the nonterminals are the state-sequences of the EDT0L system, and in the above construction the set-notation $\{q_1, \ldots, q_r\}$ should be replaced by the sequence-notation $\langle q_1, \ldots, q_r \rangle$. The only other change is that in the condition for a rule to be in R_G the requirement $u_1 u_2 \cdots u_s \in \Delta^* q_1(x)\, \Delta^* q_2(x) \cdots \Delta^* q_r(x) \Delta^*$ should be added. The same statement as above can now be proved with respect to copying-bound k. To show that $\mathrm{ET0L}_{\mathrm{ml}(k)} \subseteq \mathrm{IL}^3\mathrm{B}_{(k)}$ one has to delete the rule $\langle q_0 \rangle(x) \to x$ and replace each right-hand side of the form $\langle q_0 \rangle(w)$ by w.

We now show that $\mathrm{ILB}_{(k)} \subseteq \mathrm{EDT0L}_{(k)}$. Let $G = (N, \Delta, S, R)$ be an $\mathrm{ILB}_{(k)}$ grammar. By viewing the variables x_i as new terminal symbols it should be clear that each nonterminal $A \in N_m$ generates a languages $L_A(G) = \{w \in (\Delta \cup X_m)^* \mid A(x_1, \ldots, x_m) \overset{*}{\Rightarrow} w\}$; cf. [28]. We shall construct a regular controlled EDT0L system M that generates strings of $L_A(G)$ for all $A \in N$. The variables x_i will be used as states of M (and consequently M has state-bound k) and an application of a rule $A(x_1, \ldots, x_m) \to B(w_1, \ldots, w_n)$ will be simulated using the fact that if $B(x_1, \ldots, x_n) \overset{*}{\Rightarrow} w$, then $A(x_1, \ldots, x_m) \Rightarrow B(w_1, \ldots, w_n) \overset{*}{\Rightarrow} w[w_1, \ldots, w_n]$. The correct order of application of these rules will be ensured by a regular control language. Formally we construct $M = (Q, \Sigma, \Delta, q_0, R_M)$ such that $Q = \{q_0, q_1, \ldots, q_k\}$ (where $q_1, \ldots, q_k$ simulate $x_1, \ldots, x_k$, and q_0 is new), $\Sigma = R$ and the elements of R_M are obtained as follows.

If σ is the rule $A(x_1, \ldots, x_m) \to B(w_1, \ldots, w_n)$, then the rule $q_i(\sigma x) \to w_i[q_1(x), \ldots, q_m(x)]$ is in R_M for each i, $1 \leqslant i \leqslant n$. If σ is the rule $A(x_1, \ldots, x_m) \to w$, then the rule $q_0(\sigma x) \to w[q_1(x), \ldots, q_m(x)]$ is in R_M.

It can easily be proved by induction that $A(x_1, \ldots, x_m) \overset{*}{\Rightarrow}_v w$ if and only if $q_0(v^{\mathrm{rev}}) \overset{*}{\Rightarrow}_\pi w[q_1(\lambda), \ldots, q_m(\lambda)]$, where v denotes the sequence of rules applied in the derivation $A(x_1, \ldots, x_m) \overset{*}{\Rightarrow} w$, and v^{rev} its reverse. Hence $L(G) = M(L)$, where L is the (obviously regular) language of all $v \in \Sigma^*$ such that $S \overset{*}{\Rightarrow}_v w$ for some $w \in \Delta^*$. Hence, by Theorem 3.2.2, $L(G) \in \mathrm{EDT0L}_{(k)}$. If G is double linear and $A(x_1, \ldots, x_m) \overset{*}{\Rightarrow} w$, then w contains each x_i at most once; hence, by the above statement, each q_i occurs at most once in each (parallel) sentential form of M; consequently M has index k. If G is triple linear, then M is metalinear. ∎

The definition of an *extended* LB grammar (ELB grammar) is obtained from Definition 6.1 by allowing $w_1, \ldots, w_n$ in rules of type (a) to be finite subsets of $(X_m \cup \Delta)^*$. It was shown in [13] that ELB = ET0L. The proof of Theorem 6.3 can be extended to show that, for $k \geqslant 1$, $\mathrm{ET0L}_{(k)} = \mathrm{IELB}_{(k)}$.

Another obvious way to extend LB grammars is by allowing control. It should be clear from the proof of Theorem 6.3 that this theorem can easily be extended to control languages.

We now discuss the link between macro grammars and top-down tree transducers.

An iterative linear basic macro grammar can also be viewed as a particular kind of bottom-up tree-to-string transducer (with monadic input tree): if σ denotes the rule $A(x_1,\ldots,x_m) \to B(w_1,\ldots,w_n)$, then the corresponding rule for the bottom-up transducer would look like $\sigma(A(x_1,\ldots,x_m)) \to B(w_1,\ldots,w_n)$, where A and B are states of the transducer and σ is a unary input symbol. Generalizing to arbitrary input trees we obtain a type of bottom-up tree transducer with rules of the form $\sigma(A_1(x_{1,1},\ldots,x_{1,m_1}),\ldots,A_k(x_{k,1},\ldots,x_{k,m_k})) \to B(w_1,\ldots,w_n)$, where σ is an input symbol of rank k, symbols $A_1,\ldots,A_k$, B are states of the transducer, and $w_1,\ldots,w_n$ are strings of terminals and variables $x_{i,j}$. Intuitively, the transducer arrives at the top of the ith subtree of σ in state A_i, holding m_i translations of that subtree, and then moves up to σ in state B, computing the n translations of the tree in terms of those of the subtree (using the w_i). Thus it computes several translations of each subtree simultaneously, instead of just one (as in the case of the classical bottom-up tree transducer [18]). Such bottom-up tree transducers were investigated recently in [64]. Their relationship to the top-down tree-to-string transducer should be clear. It is left to the reader to prove the analogue of Theorem 6.3.

Theorem 6.3 and 3.2.5 show that ILB is a proper hierarchy with respect to the number of arguments. In the rest of this section we show that the class OI of arbitrary OI macro languages (see [28] for a definition) is also a hierarchy with respect to the number of arguments, with the same counterexamples as those in Theorem 3.2.5; see [47]. We start with LB. Let $L_k = \{a_1^n \cdots a_{2k}^n \mid n \geqslant 0\}$.

(6.4) LEMMA. *For $k \geqslant 2$, L_k is in $\mathrm{LB}_{(k-1)}$ but not in $\mathrm{LB}_{(k-2)}$.*

Proof. To see that L_k is in $\mathrm{LB}_{(k-1)}$ consider the grammar with rules

$$S \to A(\lambda,\ldots,\lambda),$$
$$A(x_1,\ldots,x_{k-1}) \to a_1 A(a_2 x_1 a_3,\ a_4 x_2 a_5,\ldots,\ a_{2k-2} x_{k-1} a_{2k-1}) a_{2k},$$
$$A(x_1,\ldots,x_{k-1}) \to x_1 x_2 \cdots x_{k-1}.$$

Note that this grammar is even triple linear.

To prove that $L_k \notin LB_{(k-2)}$ we shall prove that if $L_k \in \mathrm{LB}_{(s)}$ then $L_{k-1} \in \mathrm{ILB}_{(s)}$. This shows that if $L_k \in \mathrm{LB}_{(k-2)}$ then $L_{k-1} \in \mathrm{ILB}_{(k-2)}$ which is false by Theorems 6.3 and 3.2.5.

Assume that $L_k \in \mathrm{LB}_{(s)}$ for some $s \geqslant 0$ and let $G = (N, \Delta, S, R)$ be an $\mathrm{LB}_{(s)}$ grammar generating L_k. We first change G in such a way that for each sentential form $v_0 A(v_1,\ldots,v_m)v_{m+1}$ it knows $\mathrm{alph}(v_i)$ for $0 \leqslant i \leqslant m+1$. It should be clear that this information can be "added" to the nonterminal A (by creating a new nonterminal A_d for each possible piece of information d, where d is, say, a mapping from $\{x_0, x_1,\ldots,x_{m+1}\}$ into the subsets of Δ). Suppose that the new G contains a rule $A_d(x_1,\ldots,x_m) \to w_0 B_e(w_1,\ldots,w_n)w_{n+1}$ such that e informs us that the string to the left of B contains a symbol different from a_1 (i.e., $e(x_0) \not\subseteq \{a_1\}$). This means that, when this rule is used in a derivation of a string $a_1^p a_2^p \cdots a_{2k}^p$, the substring a_1^p has already been generated and, therefore, the rest of the string is completely determined. We can therefore change the above rule into $A_d(x_1,\ldots,x_m) \to w_0 w[w_1,\ldots,w_n]w_{n+1}$, where $w \in (\Delta \cup X_n)^*$ is any string such that $B_e(x_1,\ldots,x_n) \overset{*}{\Rightarrow} w$. A similar reasoning applies to the right context of $B_e(\cdots)$

and a_{2k}. Hence, after these changes, all rules of type (a) are of the form $A_d(x_1, \ldots, x_m) \to w_0 B_e(w_1, \ldots, w_n) w_{n+1}$ with $w_0 \in a_1^*$ and $w_{n+1} \in a_{2k}^*$. By replacing each a_1 and each a_{2k} in the rules by λ, we obtain an $\mathrm{ILB}_{(s)}$ grammar for the language $\{a_2{}^n a_3{}^n \cdots a_{2k-1}^n \mid n \geqslant 0\}$ and hence, by renumbering the a_i, for L_{k-1}. This shows the lemma. ∎

Let $\mathrm{OI}_{(k)}$ denote the class of all OI macro languages that can be generated by OI macro grammars for which the nonterminals have at most rank k. The next theorem shows that $\{\mathrm{OI}_{(k)}\}$ is a proper hierarchy.

(6.5) THEOREM. *For $k \geqslant 2$, L_k is in $\mathrm{LB}_{(k-1)}$ but not in $\mathrm{OI}_{(k-2)}$.*

Proof. It was shown in the previous lemma that $L_k \in \mathrm{LB}_{(k-1)}$. It should be clear that $L_2 \notin \mathrm{OI}_{(0)}$, because $\mathrm{OI}_{(0)} = \mathrm{CF}$. For $k \geqslant 3$ we shall show that if $L_k \in \mathrm{OI}_{(k-2)}$, then $L_k \in \mathrm{LB}_{(k-2)}$ which is false by the previous lemma.

As mentioned in [24] it was shown in [28] that if an OI macro language has "property P1," then it is in LB. A language L over alphabet Δ has property P1 if the following holds: if xuy, $xu'y$, $x'uy'$, and $x'u'y'$ are in L, then $u = u'$ or $\langle x, y\rangle = \langle x', y'\rangle$. The construction involved in the proof of the above fact (see Lemma 4.3.6 of [28]) changes each rule $A(x_1, \ldots, x_m) \to w$ into all possible rules $A(x_1, \ldots, x_m) \to w'$, where w' is obtained from w by substituting for each occurrence of a nonterminal, except one, a string generated by that nonterminal (cf. the proof of Lemma 6.4). Clearly this construction preserves the number of arguments (actually in [28] a normal form theorem is applied which preserves the number of arguments n only if $n \geqslant 2$; it is however easy to provide a similar proof for the case $n = 1$). This shows that for $n \geqslant 1$ if $L \in \mathrm{OI}_{(n)}$ and L has property P1, then $L \in \mathrm{LB}_{(n)}$. It is quite easy to see that, for $k \geqslant 3$, L_k has property P1. Hence, for $k \geqslant 3$, if $L_k \in \mathrm{OI}_{(k-2)}$, then $L_k \in \mathrm{LB}_{(k-2)}$. This proves the theorem. ∎

It can be proved by similar methods that L_k cannot be generated by an IO macro grammar with less than $k - 1$ arguments.

CONCLUSION

A survey has been given of the relationship between the top-down tree transducer and the ct–pd transducer, and several of their varieties obtained by determinism, monadic input alphabet (ET0L systems), copying-bound and metalinearity. Using results of [26] it was shown that many classes of tree transformation languages and indexed languages can be obtained by simple variations in the machine model, showing the similarities and differences between these two kinds of languages, which are related via the ET0L languages.

For each of the transducers considered in this paper one may investigate closure under composition and, in case of a negative answer, whether their iteration gives rise to a proper hierarchy of classes of languages. Using the results of this paper (and in particular the concept of finite copying) it is shown in [23] that a proper hierarchy is obtained for the top-down tree transducer, the cs–pd transducer (= ET0L mapping) and the cs transducer (= 2-way gsm).

We mention the following areas of further research, apart from problems discussed in the paper such as restriction of rank (see after Theorem 5.1) and the properness of the inclusions in Corollary 5.6.

(1) Polynomial copying. As mentioned before, a polynomial bound on the copying power of deterministic top-down tree transducers was investigated in [4], and, for the monadic case, in [6]. It is worthwhile to investigate the corresponding classes of languages more deeply and to find a (pebble) machine model for them.

(2) Extensions to both the ct–pd transducer and the s–pd machine. A generalization of both these machines might lead to a formal model of the syntax-directed translation of non-context-free languages. One possibility is to generalize the pushdown of the ct–pd transducer to be a stack or even an s–pd (still synchronized with the input pointer, of course).

REFERENCES

1. A. V. AHO AND J. D. ULLMAN, Properties of syntax directed translations, *J. Comput. System Sci.* **3** (1969), 319–334.
2. A. V. AHO AND J. D. ULLMAN, Syntax directed translations and the pushdown assembler, *J. Comput. System Sci.* **3** (1969), 37–56.
3. A. V. AHO AND J. D. ULLMAN, A characterization of two-way deterministic classes of languages, *J. Comput. System Sci.* **4** (1970), 523–538.
4. A. V. AHO AND J. D. ULLMAN, Translations on a context-free grammar, *Inform. Contr.* **19** (1971), 439–475.
5. A. V. AHO AND J. D. ULLMAN, "The Theory of Parsing, Translation and Compiling," Vol. I, II, Prentice–Hall, Englewood Cliffs, N. J., 1973.
6. A. ARNOLD AND M. DAUCHET, Translations de forêts reconaissables monadiques; forêts corégulières, *RAIRO Inform. Théor.* **10** (1976), 5–28.
7. P. R. J. ASVELD, Controlled iteration grammars and full hyper-AFL's, *Inform. Contr.* **34** (1977), 248–269.
8. P. R. J. ASVELD AND J. ENGELFRIET, Iterated deterministic substitution, *Acta Informatica* **8** (1977), 285–302.
9. B. S. BAKER, Tree transducers and tree languages, *Inform. Contr.* **37** (1978), 241–266. See also [65, 66].
10. P. A. CHRISTENSEN, Hyper-AFL's and ET0L systems, *in* "*L* Systems" (G. Rozenberg and A. Salomaa, Eds.), pp. 254–257, Lecture Notes in Computer Science No. 15, Springer–Verlag, Berlin, 1974.
11. M. P. CHYTIL AND V. JÁKL, Serial composition of 2-way finite-state transducers and simple programs on strings, *in* "Automata, Languages and Programming; Fourth Colloquium, Turku" (A. Salomaa and M. STEINBY, Eds.), pp. 135–147, Lecture Notes in Computer Science No. 52, Springer–Verlag, Berlin, 1977.
12. K. CULIK II, On some families of languages related to developmental systems, *Internat. J. Comput. Math.* **4** (1974), 31–42.
13. P. DOWNEY, "Formal Languages and Recursion Schemes," Harvard University, Report TR-16-74, 1974.
14. A. EHRENFEUCHT AND G. ROZENBERG, On some context-free languages that are not deterministic ET0L languages; *RAIRO Inform. Théor.* **11** (1977), 273–291.
15. A. EHRENFEUCHT AND G. ROZENBERG, Three useful results concerning *L* languages without interactions, *in* "*L* Systems" (G. Rozenberg and A. Salomaa, Eds.), pp. 72–77, Lecture Notes in Computer Science No. 15, Springer–Verlag, Berlin, 1974.

16. A. EHRENFEUCHT, G. ROZENBERG, AND D. VERMEIR, On ET0L Systems with Rank, *J. Comput. System Sci.* **19** (1979), 237–255.
17. R. W. EHRICH AND S. S. YAU, Two-way sequential transductions and stack automata, *Inform. Contr.* **18** (1971), 404–446.
18. J. ENGELFRIET, Bottom-up and top-down tree transformations: A comparison, *Math. Systems Theory* **9** (1975), 198–231.
19. J. ENGELFRIET, Top-down tree transducers with regular look-ahead, *Math. Systems Theory* **10** (1977), 289–303.
20. J. ENGELFRIET, Surface tree languages and parallel derivation trees, *Theor. Comput. Sci.* **2** (1976), 9–27.
21. J. ENGELFRIET, "Tree Automata and Tree Grammars," Lecture Notes, DAIMI FN-10, University of Aarhus, 1975.
22. J. ENGELFRIET, Macro grammars, Lindenmayer systems and other copying devices, *in* "Automata, Languages and Programming; Fourth Colloquium, Turku" (A. Salomaa and M. Steinby, Eds.), pp. 221–229, Lecture Notes in Computer Science No. 52, Springer–Verlag, Berlin, 1977.
23. J. ENGELFRIET, "Three Hierarchies of Transducers," Memorandum 217, Twente University of Technology, 1978.
24. J. ENGELFRIET AND S. SKYUM, Copying theorems, *Inform. Proc. Letters* **4** (1976), 157–161.
25. J. ENGELFRIET AND E. MEINECHE SCHMIDT, IO and OI, I, *J. Comput. System Sci.* **15** (1977), 328–353; II, *J. Comput. System Sci.* **16** (1978), 67–99.
26. J. ENGELFRIET, E. MEINECHE SCHMIDT, AND J. VAN LEEUWEN, Stack Machines and Classes of Nonnested Macro Languages, *J. Assoc. Comput. Mach.* **27** (1980), 96–117.
27. J. ENGELFRIET AND G. SLUTZKI, Classes of stack controlled automata, in preparation.
28. M. J. FISCHER, "Grammars with Macro-like Productions," Ph. D. thesis, Harvard University, 1968; (see also "9th Conference on Switching and Automata Theory," pp. 131–142).
29. S. GINSBURG AND G. ROZENBERG, T0L schemes and control sets, *Inform. Contr.* **27** (1975), 109–125.
30. S. GINSBURG AND E. H. SPANIER, AFL with the semilinear property, *J. Comput. System Sci.* **5** (1971), 363–396.
31. S. A. GREIBACH, Checking automata and one-way stack languages, *J. Comput. System Sci.* **3** (1969), 196–217.
32. S. A. GREIBACH, Full AFLs and nested iterated substitution, *Inform. Contr.* **16** (1970), 7–35.
33. S. A. GREIBACH, Syntactic operators on full semi AFLs, *J. Comput. System Sci.* **6** (1972), 30–76.
34. S. A. GREIBACH, Control sets on context-free grammar forms, *J. Comput. System Sci.* **15** (1977), 35–98.
35. S. A. GREIBACH, One-way finite visit automata, *Theor. Comput. Sci.* **6** (1978), 175–221.
36. J. E. HOPCROFT AND J. D. ULLMAN, "Formal Languages and Their Relation to Automata," Addison–Wesley, Reading, Mass., 1969.
37. O. H. IBARRA, Simple matrix languages, *Inform. Contr.* **17** (1970), 359–394.
38. O. H. IBARRA, Controlled pushdown automata, *Inform. Sci.* **6** (1973), 327–342.
39. N. A. KHABBAZ, A geometric hierarchy of languages, *J. Comput. System Sci.* **8** (1974), 142–157.
40. N. A. KHABBAZ, Control sets on linear grammars, *Inform. Contr.* **25** (1974), 206–221.
41. D. KIEL, Two-way a-transducers and AFL, *J. Comput. System Sci.* **10** (1975), 88–109.
42. R. J. NELSON, "Introduction to Automata," Wiley, New York, 1968.
43. W. F. OGDEN AND W. C. ROUNDS, Composition of n transducers, *in* "Proceedings, 4th ACM Symposium on Theory of Computing," pp. 198–206, 1972.
44. C. R. PERRAULT, Intercalation lemmas for tree transducer languages, *J. Comput. System Sci.* **13** (1976), 246–277.
45. V. RAJLICH, Absolutely parallel grammars and two-way finite-state transducers, *J. Comput. System Sci.* **6** (1972), 324–342.
46. V. RAJLICH, Bounded-crossing transducers, *Inform. Contr.* **27** (1975), 329–335.

47. V. J. RAYWARD-SMITH, Hypergrammars: An extension to macro grammars, *J. Comput. System Sci.* **14** (1977), 130–149.
48. W. C. ROUNDS, Mappings and grammars on trees, *Math. Systems Theory* **4** (1970), 257–287.
49. G. ROZENBERG, Extension of tabled 0L-systems and languages, *Internat. J. Comp. Inform. Sci.* **2** (1973), 311–336.
50. G. ROZENBERG AND D. VERMEIR, On ET0L systems of finite index, *Inform. Contr.* **38** (1978), 103–133.
51. G. ROZENBERG AND D. VERMEIR, "On Metalinear ET0L systems," Report 77-02, UIA, University of Antwerp, 1975; *Fundamenta Informatica*, in press.
52. A. SALOMAA, "Formal Languages," Academic Press, New York, 1973.
53. A. SALOMAA, "Macros, Iterated Substitution and Lindenmayer AFL's," DAIMI PB-18, University of Aarhus, 1973.
54. R. SIROMONEY, On equal matrix languages, *Inform. Contr.* **14** (1969), 135–151.
55. R. SIROMONEY, Finite-turn checking automata, *J. Comput. System Sci.* **5** (1971), 549–559.
56. J. W. THATCHER, Generalized^2 sequential machine maps, *J. Comput. System Sci.* **4** (1970), 339–367 (see also IBM Report RC 2466).
57. J. W. THATCHER, Tree automata: an informal survey, *in* "Currents in the Theory of Computing" (A. V. Aho, Ed.), pp. 143–172, Prentice–Hall, Englewood Cliffs, 1973.
58. J. W. THATCHER, Characterizing derivation trees of context-free grammars through a generalization of finite automata theory, *J. Comput. System Sci.* **1** (1967), 317–322.
59. J. W. THATCHER AND J. B. WRIGHT, Generalized finite automata theory with an application to a decision problem of second-order logic, *Math. Systems Theory* **2** (1968), 57–81.
60. J. VAN LEEUWEN, A study of complexity in hyper-algebraic families, *in* "Automata, Languages, Development" (A. Lindenmayer and G. Rozenberg, Eds.), pp. 323–333, North–Holland, Amsterdam, 1976.
61. J. VAN LEEUWEN, Variations of a new machine model, *in* "Proceedings, 17th Ann. IEEE Symposium on Foundations of Computer Science, Houston, 1976," pp. 228–235.
62. M. LATTEUX, Substitutions dans les EDT0L-systèmes ultralinéaires, *Inform. Contr.* **42** (1979), 194–260.
63. M. LATTEUX, "EDT0L-systèmes ultralinéaires et opérateurs associés," Publication 100 du Laboratoire de Calcul, Université de Lille, 1977.
64. E. LILIN, "Une généralisation des transducteurs d'états finis d'arbres: les S-transducteurs," Thèse de troisième cycle, Université de Lille, 1978.
65. B. S. BAKER, Generalized syntax directed translation, tree transducers, and linear space, *SIAM J. Comput.* **7** (1978), 376–391.
66. B. S. BAKER, Composition of top-down and bottom-up tree transductions, *Inform. Contr.* **41** (1979), 186–213.