Branching Synchronization Grammars with Nested Tables

Frank Drewes
Department of Computing Science, Umeå University, S–901 87 Umeå, Sweden
drewes@cs.umu.se

Joost Engelfriet
Department of Computer Science, Leiden University
P.O. Box 9612, NL-2300 RA Leiden, The Netherlands
engelfri@liacs.nl

Abstract. A generalization of ET0L systems is introduced: grammars with branching synchronization and nested tables. Branching synchronization grammars with tables of nesting depth n have the same string- and tree-generating power as n-fold compositions of top-down tree transducers.

UMINF 02.22 ISSN 0348-0542

1 Introduction

Context-free Chomsky grammars and context-free Lindenmayer systems certainly belong to the most useful and well-studied types of grammars that have been proposed in formal language theory. They are both sufficiently general in order to describe interesting languages and sufficiently simple in order to guarantee nice mathematical and algorithmic properties (see [RS97] for these and many other types of grammatical systems).

Context-free grammars are particularly well suited to formalize certain aspects of natural language grammar and, today perhaps more importantly, the syntax of programming languages. In contrast, Lindenmayer systems have been proposed as a means to model developmental processes in biology. An important characteristics of such processes is that all parts develop simultaneously. Consequently, Lindenmayer systems employ a parallel mode of rewriting in which all symbols of the current sentential form are replaced in each step. Later, Rozenberg extended context-free Lindenmayer systems (0L systems) by dividing the set of rules into subsets called tables [Roz73a] and adding a distinguished set of terminal symbols [Roz73b]. As a rather immediate consequence, the resulting ET0L systems are more powerful than both context-free grammars and 0L systems.

The tables of an ET0L system provide a global synchronization mechanism: In every derivation step, a table is chosen and all nonterminal symbols in the current sentential form are replaced in parallel, using rules from the chosen table.[1] To discuss an example, let twice(L) denote the language $\{u\$u \mid u \in L\}$, for every language L. The language twice($\{a,b\}^*$) can easily be generated by an ET0L system using the tables

$$\{S \to A\$A\},\ \{A \to aA\},\ \{A \to bA\},\ \{A \to \lambda\}$$

where S and A are nonterminals (of which S is the start symbol), $a, b, \$$ are terminals, and λ denotes the empty string. This language is neither context-free, nor can it be generated by a 0L system.

However, what about the language twice(D), where D is the Dyck language over one pair of parentheses? Recall that D is generated by the context-free grammar with rules $S \to SS$, $S \to (S)$, and $S \to ()$. It seems that twice(D) cannot be generated by an ET0L system since only pairwise corresponding occurrences of S would have to be synchronized. After a few steps, one would expect a derivation to produce the string $SS(S)\$SS(S)$, for instance. In this situation, three potentially different rules r_1, r_2, r_3 must be applied to the six nonterminals, namely r_1 to the first and fourth, r_2 to the second and fifth, and r_3 to the third and sixth occurrence of S. In other words, in order to generate this language it seems to be necessary to synchronize only specific occurrences of nonterminals with each other.

In this paper, we propose an extension of ET0L systems, called branching synchronization grammars with nested tables (branching grammars, for short), in which pairs of nonterminals can, but need not be synchronized with each other. In a branching grammar of nesting depth n (where $n \in \mathbb{N}$), the tables are organized in a tree-like hierarchy of depth n. We exploit this hierarchy in order to provide n increasingly strong levels of synchronization. A pair of nonterminals in a given sentential form may be synchronized at any of these levels, or not at all.

Let us discuss the special case where $n = 1$, which we call branching ET0L system. In contrast to ordinary ET0L systems, the synchronization between some of the nonterminals can be released while the synchronization between others is kept. Technically, this is accomplished by letting the nonterminals accumulate so-called synchronization strings during a derivation. If the synchronization strings of two nonterminals are equal, they are synchronized with each other. Thus, rules from the same table must be applied to them. If the respective synchronization strings differ, the tables can be chosen independently. The nonterminals in the right-hand side of a rule are equipped with a single synchronization symbol each. In every step $u \Rightarrow v$ of a derivation, a nonterminal in v inherits the synchronization string of its predecessor in u, to which the corresponding synchronization symbol from the right-hand side of the applied rule is appended. Thus, the synchronization can be released by providing nonterminals with different synchronization symbols, respectively kept by providing them with the same synchronization symbol.

Returning to the example twice(D) from above, a branching ET0L system for this language uses a new start symbol T and the tables

$$\{T \to S\langle 0\rangle \$ S\langle 0\rangle\},\ \{S \to S\langle 0\rangle S\langle 1\rangle\},\ \{S \to (S\langle 0\rangle)\},\ \{S \to ()\}$$

where the symbols in angular brackets are the synchronization symbols (0 and 1). Now,

[1]We shall speak of terminals and nonterminals because our definition of ET0L systems in Section 3 makes this distinction (in contrast to the original definition of ET0L systems).

the initial phase of a derivation may take the form

$$\begin{array}{rcll} T\langle\rangle & \Rightarrow & S\langle 0\rangle\$S\langle 0\rangle & \text{by table 1} \\ & \Rightarrow & S\langle 00\rangle S\langle 01\rangle\$S\langle 00\rangle S\langle 01\rangle & \text{by table 2} \\ & \Rightarrow & S\langle 000\rangle S\langle 001\rangle(S\langle 010\rangle)\$S\langle 000\rangle S\langle 001\rangle(S\langle 010\rangle) & \text{by tables 2 and 3} \\ & \Rightarrow & \cdots & \end{array}$$

synchronizing only those nonterminals for which synchronization is desired.

Our main result characterizes the language generating power of branching grammars in terms of output languages of top-down tree transducers (td transducers). The investigation of the latter was initiated in the late sixties and early seventies by Rounds and Thatcher [Rou70, Tha70a, Tha70b, Tha73], and later continued by many others [Eng75, Eng77, Bak78a, Bak78b, Bak79, Eng82, FV89, AD94, Sei94, SV95, Dre96, GV96, DF98, DE98, Man98, MN00, Dre01a, MN03]. For an overview on the theory of tree transducers see the books by Gécseg and Steinby [GS84], and Fülöp and Vogler [FV98].

A td transducer transforms input trees into output trees by processing the input tree from the root towards the leaves using a finite number of states. It is known from [Eng76] that there exists a natural relationship between ET0L languages and output languages of td transducers: the ET0L languages are precisely the yields of output languages of td transducers having a monadic input alphabet. Equivalently [ERS80], they are the output languages of top-down tree-to-string transducers with a monadic input alphabet. Intuitively, the states of the transducer are the nonterminals of the ET0L system and the input symbols represent the tables. The synchronization effect is obtained by the copying of (subtrees of) the input tree that takes place in a computation of the transducer.

For instance, the four tables used to generate the language twice$(\{a,b\}^*)$ above may be represented by input symbols $\tau_1,\ldots,\tau_4$ of rank 1. Together with a symbol ϵ of rank 0 this yields a set of monadic input trees each of which represents a unique sequence of tables. Now, the rules of a top-down tree-to-string transducer corresponding to the ET0L system above (where the nonterminals become states, of which S is the start state, and x is a variable ranging over input trees) are[2]

$$\begin{array}{rcl} S[\tau_1[x]] & \to & A[x]\$A[x], \\ A[\tau_2[x]] & \to & aA[x], \\ A[\tau_3[x]] & \to & bA[x], \\ A[\tau_4[x]] & \to & \lambda. \end{array}$$

Thus, in the initial computation step the input tree is copied, which makes sure that the two subcomputations apply the same table sequence.

In general, the fact that the input tree is monadic guarantees that, after m parallel computation steps (for any m), all states will process a copy of the same subtree of the input tree. In other words, similar to an ET0L system there is a strict synchronization covering all states present. This is not any longer the case if we consider arbitrary input trees. For example, if τ_1 was of rank 2 above and the first rule was replaced with

[2] The meaning of these rewriting rules should be self-evident; within the paper we will use a different notation for technical reasons.

$S[\tau_1[x,y]] \to A[x]\$A[y]$, the two subcomputations after the first step would process distinct (and therefore potentially different) subtrees, thus releasing the synchronization. We could, for example, also employ the rule $S[\tau_1[x,y]] \to A[x]\$A[y]\$A[x]$. Then the first subcomputation is synchronized with the third one, but not with the second; as a result, the output strings are of the form $u\$v\$u$ where $u,v \in \{a,b\}^*$. As a more interesting example, let us consider a top-down tree-to-string transducer that generates the language twice(D). It has the following rules, where the input symbols $\tau_1,\dots,\tau_4$ now have rank 2 and represent the tables of the branching ET0L system discussed above:

$$\begin{array}{rcl} T[\tau_1[x,y]] & \to & S[x]\$S[x], \\ S[\tau_2[x,y]] & \to & S[x]S[y], \\ S[\tau_3[x,y]] & \to & (S[x]), \\ S[\tau_4[x,y]] & \to & \lambda. \end{array}$$

A computation of this transducer, corresponding to the derivation of the branching ET0L system considered above, is as follows (for an input tree $s = \tau_1[\tau_2[\tau_2[t,u],\tau_3[v,\epsilon]],\epsilon]$ with arbitrary subtrees t, u, and v):

$$\begin{array}{rcll} T[s] & = & T[\tau_1[\tau_2[\tau_2[t,u],\tau_3[v,\epsilon]],\epsilon]] & \\ & \Rightarrow & S[\tau_2[\tau_2[t,u],\tau_3[v,\epsilon]]]\$S[\tau_2[\tau_2[t,u],\tau_3[v,\epsilon]]] & \text{by rule 1} \\ & \Rightarrow^2 & S[\tau_2[t,u]]S[\tau_3[v,\epsilon]]\$S[\tau_2[t,u]]S[\tau_3[v,\epsilon]] & \text{by rule 2 (twice)} \\ & \Rightarrow^4 & S[t]S[u](S[v])\$S[t]S[u](S[v]) & \text{by rules 2 and 3 (twice)} \\ & \Rightarrow & \cdots & \end{array}$$

Note that the synchronization strings 000, 001, and 010 in the derivation of the branching ET0L system correspond to the positions of the subtrees t, u, and v, respectively, in the input tree t (where 0 means 'left' and 1 means 'right').

The ability of td transducers to implement this type of partial synchronization makes them more powerful than ET0L systems (if viewed as language generating devices). Moreover, it is known from [Eng82] that compositions of $n+1$ td transducers yield a strictly larger class of output languages than compositions of n td transducers. Our main result states that branching grammars of nesting depth n have exactly the same language generating power as compositions of n td transducers. We obtain this result by considering branching *tree* grammars, which are branching grammars in which the right-hand sides of rules are trees, i.e., strings having a term-like structure, with the nonterminals at the leaves (thus generalizing the regular tree grammars).

Apart from being of independent theoretical interest, we expect the main result to be useful for applications in which only the language generating power of compositions of td transducers is needed, while the tree transformations as such are of minor interest (see, e.g., [Dre00, Dre01b]). In such cases it is appropriate to describe the desired language by means of a single grammatical device rather than by a cumbersome and (from the algorithmic point of view) inefficient n stage process using n td transducers.

Furthermore, our characterization can be used to prove new results about output languages of td transducers. We exemplify this by proving the following. Let $\mathrm{TD}^n(\mathrm{REGT})$ denote the set of all tree languages obtained by applying a composition of n td transducers to a regular tree language (and similarly for other classes of input languages).

Thus, according to our main result, $\mathrm{TD}^n(\mathrm{REGT})$ equals the class of tree languages generated by branching tree grammars of nesting depth n and $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ equals the class of string languages generated by branching grammars of nesting depth n. We show that, for every $n \geq 1$, there exists a single tree language $K_{n-1} \in \mathrm{TD}^{n-1}(\mathrm{REGT})$ such that $\mathrm{TD}^n(\mathrm{REGT}) = \mathrm{TD}(\{K_{n-1}\})$. In other words, the whole class $\mathrm{TD}^n(\mathrm{REGT})$ can be generated by td transducers from just one of the elements of $\mathrm{TD}^{n-1}(\mathrm{REGT})$. Moreover, we obtain a similar result for the generated class $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ of string languages: For every $n \geq 0$ there is a single language $L_n \in \mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ such that $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT})) = \mathrm{FST}(\{L_n\})$, where FST denotes the set of all finite state string-to-string transductions. Thus, $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ can be generated by finite state transductions from just one of its elements, which means that this class is a full principal AFL.

The paper is structured as follows. In the next section, we collect some basic notations and definitions. In Sections 3 and 4 we introduce branching grammars and prove some of their basic properties. In Section 5 we define branching tree grammars, recall regular tree grammars and td transducers, and prove a few results about td transducers needed later. Sections 6 and 7 show that output languages of compositions of td transducers can be generated by branching tree grammars, and vice versa. From this we obtain our main result, stated in Section 8. We conclude the paper with Section 9, where the main result is exploited in order to show that $\mathrm{TD}^n(\mathrm{REGT})$ and $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ can be generated from single languages.

2 Preliminaries

The reader is assumed to be familiar with the basic concepts of formal language theory. In what follows we enumerate some of the terminology we use. In particular the reader should notice the way we treat strings of n-tuples.

We denote the set of natural numbers (including 0) by $\mathbb{N}$. For $n \in \mathbb{N}$, $[n]$ denotes $\{1, \ldots, n\}$. An alphabet is a finite set of symbols. As usual, X^* denotes the set of all strings over a set X. We denote the empty string by λ and the length of $s \in X^*$ by $|s|$. The concatenation of two strings s and t is denoted st, and s^n denotes the n-fold concatenation $s \cdots s$. The cardinality of a set X is denoted by $|X|$ and its powerset by $\mathcal{P}(X)$. By X^n we denote the set of all n-tuples $(x_1, \ldots, x_n)$ over elements $x_i \in X$. For $\gamma = (x_1, \ldots, x_n) \in X^n$ and $x \in X$ we let $\gamma.x$ denote the n+1-tuple $(x_1, \ldots, x_n, x)$, and for $k \in \{0, \ldots, n\}$, $\mathrm{first}_k(\gamma) = (x_1, \ldots, x_k)$ restricts γ to its first k components.

Note that, even though we also use the notation s^n for the n-fold concatenation of strings, we do not identify n-tuples with strings of length n. In particular, we will consider the set $(X^n)^*$ of strings of n-tuples. As an example,

$$(x, y, y)(x, y, y)(x, x, x)(y, y, y)$$

is a string of 3-tuples of length 4, different from the string $xyyxyyxxxyyy$ of length 12.

If $\Rightarrow \subseteq X^2$ is a binary relation we let $\Rightarrow^n$ ($n \in \mathbb{N}$) denote the n-fold composition of $\Rightarrow$ with itself, where $\Rightarrow^0$ is the identity on X. Moreover, $\Rightarrow^+$ and $\Rightarrow^*$ denote the transitive and the transitive-reflexive closure of $\Rightarrow$, respectively. For a binary relation $r \subseteq X \times Y$ and $x \in X$ we let $r(x) = \{y \in Y \mid (x, y) \in r\}$. For $X' \subseteq X$, $r(X') = \bigcup_{x \in X} r(x)$. The domain of r is the set $\mathrm{dom}(r)$ of all $x \in X$ such that $r(x) \neq \emptyset$. The composition of r with $r' \subseteq Y \times Z$ is $r' \circ r = \{(x,z) \mid (x,y) \in r \text{ and } (y,z) \in r' \text{ for some } y \in Y\}$.

A *ranked alphabet* is an alphabet Σ such that every symbol $a \in \Sigma$ is given a *rank* in $\mathbb{N}$. For every $k \in \mathbb{N}$, the set of symbols of rank k in Σ is denoted by Σ_k. For a ranked alphabet Σ and a set A disjoint with Σ, the set $\mathrm{T}_\Sigma(A)$ of trees over Σ and A is the smallest subset T of $(\Sigma \cup A)^*$ such that $A \subseteq T$ and, for all $f \in \Sigma_k$ ($k \in \mathbb{N}$) and $t_1, \ldots, t_k \in T$, $f\, t_1 \cdots t_k \in T$. For the sake of readability we often write $f[t_1, \ldots, t_k]$ instead of $f\, t_1 \cdots t_k$. The trees $t_1, \ldots, t_k$ are said to be the *direct subtrees* of the tree $f[t_1, \ldots, t_k]$. The set of subtrees of a tree contains the tree itself as well as all subtrees of its direct subtrees. The set $\mathrm{T}_\Sigma(\emptyset)$ of trees over Σ is denoted by T_Σ. A subset of T_Σ is called a *tree language*.

If we read the leaves of a tree from left to right we obtain a string called its *yield*. Formally, let us reserve a special symbol ϵ of rank 0 to denote the empty string. To avoid confusion we shall assume throughout the paper that ϵ is never used as an ordinary symbol in the considered alphabets. Now, for every symbol a of rank 0

$$\mathrm{yield}(a) = \begin{cases} \lambda & \text{if } a = \epsilon \\ a & \text{otherwise} \end{cases}$$

and, for all trees $f[t_1, \ldots, t_k]$ with $k \geq 1$, $\mathrm{yield}(f[t_1, \ldots, t_k]) = \mathrm{yield}(t_1) \cdots \mathrm{yield}(t_k)$. Thus, $\mathrm{yield}(t)$ is obtained from $t \in \mathrm{T}_\Sigma$ by simply deleting all symbols except for those in $\Sigma_0 \setminus \{\epsilon\}$. Given a class $\mathcal{L}$ of tree languages, we denote by $\mathrm{yield}(\mathcal{L})$ the class of all string languages of the form $\mathrm{yield}(L)$ with $L \in \mathcal{L}$.

Other terminology will be introduced within the paper, when it is needed.

3 Branching grammars

In this section, we introduce branching grammars and illustrate the definition by means of examples. Since branching grammars are a generalization of ET0L systems, we start by recalling the notion of ET0L system.

An *ET0L system* is a tuple $G = (N, T, J, R, S)$ where

- N and T are disjoint alphabets of *nonterminals* resp. *terminals*,
- J is a nonempty alphabet of *table symbols*,
- R, the *table specification*, assigns to every $\tau \in J$ a finite set $R(\tau)$ of rules $A \to \zeta$ such that $A \in N$ and $\zeta \in (N \cup T)^*$, and
- $S \in N$ is the *initial nonterminal*.

Given such an ET0L system, let $\xi_1, \xi_2 \in (N \cup T)^*$ where $\xi_1 = w_0 A_1 w_1 \cdots A_h w_h$ for some $h \in \mathbb{N}$, $w_0, \ldots, w_h \in T^*$, and $A_1, \ldots, A_h \in N$. There is a derivation step $\xi_1 \Rightarrow \xi_2$ if there is some $\tau \in J$ such that $R(\tau)$ contains rules $A_1 \to \zeta_1, \ldots, A_h \to \zeta_h$ with $\xi_2 = w_0 \zeta_1 w_1 \cdots \zeta_h w_h$. Note the parallel mode of derivation which is typical for Lindenmayer systems. In each derivation step all nonterminals are replaced in parallel. Furthermore, all rules applied in the same step must be taken from the same $R(\tau)$, called a *table*. The language $L(G)$ generated by G is the set of all strings $w \in T^*$ such that $S \Rightarrow^* w$. The class of all languages of the form $L(G)$, where G is an ET0L system, is denoted by ET0L. Note that this class includes the context-free languages because G generates the same language as the context-free grammar $(N, T, R(\tau), S)$ if J is a singleton $\{\tau\}$. To see this, it suffices to notice that the parallel derivation mode yields the same terminal strings as the sequential mode employed by context-free Chomsky grammars if only one table is present.

The reader may have noticed that our definition of ET0L systems deviates slightly from the standard definition. Usually, all symbols of an ET0L system are subject to replacement and T is only used as a filter which determines the sentential forms that belong to $L(G)$. However, it is easy to see that both definitions are equivalent.

The tables of an ET0L system implement a very strict kind of synchronization as the nonterminals cannot be replaced independently of each other. Branching grammars generalize this concept of synchronization by introducing different levels of synchronization, so that the synchronization need not be equally strong for all pairs of nonterminals in a given sentential form. In the following, we shall first discuss a special case which generalizes the ET0L system and which we call branching ET0L system. In a derivation of a branching ET0L system nonterminals are either synchronized (level 1), in which case they have to be replaced by the same table, or they are unsynchronized (level 0) and are replaced independently. At each derivation step the synchronization can be retained or released by the use of so-called synchronization symbols. Thus, the only difference between an ET0L system and a branching ET0L system is that, in the latter, the nonterminals in the right-hand sides of rules are equipped with a synchronization symbol each. The formal definition reads as follows.

Definition 3.1 (branching ET0L system) A *branching ET0L system* is a tuple $G = (N, T, I, J, R, S)$ where

- N and T are disjoint alphabets of *nonterminals* resp. *terminals*,
- I and J are nonempty alphabets of *synchronization symbols* resp. *table symbols*,
- R, the *table specification*, assigns to every $\tau \in J$ a finite set $R(\tau)$ of rules $A \to \zeta$ such that $A \in N$ and $\zeta \in ((N \times I) \cup T)^*$, and
- $S \in N$ is the *initial nonterminal.*

Given a branching ET0L system G as in the definition, elements of I^* are called *synchronization strings* and SN_G denotes the set $N \times I^*$ of *synchronized nonterminals* (of G). Thus, a synchronized nonterminal (A, φ) consists of a nonterminal A and a synchronization string φ. The *initial synchronized nonterminal* is (S, λ). The intuitive idea behind the definition of derivations is that the nonterminals accumulate synchronization symbols to synchronization strings. More precisely, in each derivation step $\xi_1 \Rightarrow \xi_2$ every nonterminal in ξ_2 inherits the synchronization string of the replaced nonterminal in ξ_1, to which the corresponding synchronization symbol in the right-hand side of the applied rule is appended. The accumulated synchronization strings control the rewriting as follows: in a derivation step, nonterminals with equal synchronization strings must be replaced using rules from the same table whereas nonterminals with different synchronization strings can be replaced independently of each other. In this way, one is able to release the synchronization. To define this properly, we first define an auxiliary step relation that takes care of the accumulation, as follows. For every $(A, \varphi) \in \mathrm{SN}_G$ and every rule

$$r = A \to v_0(B_1, \alpha_1)v_1 \cdots (B_l, \alpha_l)v_l$$

where $l \in \mathbb{N}$, $A, B_1, \ldots, B_l \in N$, $v_0, \ldots, v_l \in T^*$, and $\alpha_1, \ldots, \alpha_l \in I$, we define

$$(A, \varphi) \Rightarrow_r v_0(B_1, \varphi\alpha_1)v_1 \cdots (B_l, \varphi\alpha_l)v_l.$$

Let us now define derivation steps in general. They are, as in an ET0L system, fully parallel—in each step all nonterminals in the sentential form are replaced simultaneously. Let $\xi_1, \xi_2 \in (\mathrm{SN}_G \cup T)^*$ where $\xi_1 = w_0(A_1, \varphi_1)w_1 \cdots (A_h, \varphi_h)w_h$ for some $(A_1, \varphi_1), \ldots, (A_h, \varphi_h) \in \mathrm{SN}_G$ and $w_0, \ldots, w_h \in T^*$. Then there is a *derivation step* $\xi_1 \Rightarrow \xi_2$ if there are $\tau_1, \ldots, \tau_h \in J$ and rules $r_1 \in R(\tau_1), \ldots, r_h \in R(\tau_h)$ such that

(i) $\xi_2 = w_0\zeta_1 w_1 \cdots \zeta_h w_h$ where $(A_j, \varphi_j) \Rightarrow_{r_j} \zeta_j$ for all $j \in [h]$ and

(ii) for all $i, j \in [h]$, $\varphi_i = \varphi_j$ implies $\tau_i = \tau_j$.

The *language generated by* G is $L(G) = \{w \in T^* \mid (S, \lambda) \Rightarrow^* w\}$.

Clearly, with these definitions an ET0L system is the special case of a branching ET0L system where $|I| = 1$. (Note that, by induction on the length of derivations, every string $\xi_1 = w_0(A_1, \varphi_1)w_1 \cdots (A_h, \varphi_h)w_h$ as above satisfies $|\varphi_1| = \cdots = |\varphi_h|$ if $(S, \lambda) \Rightarrow^* \xi_1$.)

Let us consider an example. The aim is to generate the smallest language $L_0 \subseteq T^* = \{\triangleleft, \triangleright, [,], |\}^*$ such that $\triangleright, \triangleleft \in L_0$ and, for all $u, v \in L_0$, the strings $[u]$, uv, and $[u|u^{\mathrm{r}}]$ are in L_0. Here, u^{r} denotes the *reflection* of u given by $\lambda^{\mathrm{r}} = \lambda$ and, for all $a \in T, u \in T^*$,

$$(au)^{\mathrm{r}} = \begin{cases} u^{\mathrm{r}}\triangleleft & \text{if } a = \triangleright \\ u^{\mathrm{r}}\triangleright & \text{if } a = \triangleleft \\ u^{\mathrm{r}}] & \text{if } a = [\\ u^{\mathrm{r}}[& \text{if } a =] \\ u^{\mathrm{r}}| & \text{if } a = | \end{cases}$$

Thus, up to the case $[u|u^{\mathrm{r}}]$, L_0 is the language of balanced parentheses (Dyck language) over [and] if we remove all $\triangleright$'s and $\triangleleft$'s. In addition, strings in L_0 can contain substrings of the form $[u|v]$ where v is the reflection of u (and u is an element of L_0).

To generate this language we use two synchronization symbols (say $I = \{0, 1\}$), five tables (say $J = \{1, 2, 3, 4, 5\}$), and two nonterminals S, S^{r}, where S is the initial one. Intuitively, if occurrences of S and S^{r} are synchronized with each other and S generates u then S^{r} generates u^{r}. The five tables are the following ones:[3]

$$\begin{array}{rcllll}
R(1) & = & \{ & S \to [S\langle 0\rangle], & S^{\mathrm{r}} \to [S^{\mathrm{r}}\langle 0\rangle] & \}, \\
R(2) & = & \{ & S \to S\langle 0\rangle S\langle 1\rangle, & S^{\mathrm{r}} \to S^{\mathrm{r}}\langle 1\rangle S^{\mathrm{r}}\langle 0\rangle & \}, \\
R(3) & = & \{ & S \to [S\langle 0\rangle|S^{\mathrm{r}}\langle 0\rangle], & S^{\mathrm{r}} \to [S\langle 0\rangle|S^{\mathrm{r}}\langle 0\rangle] & \}, \\
R(4) & = & \{ & S \to \triangleright, & S^{\mathrm{r}} \to \triangleleft & \}, \\
R(5) & = & \{ & S \to \triangleleft, & S^{\mathrm{r}} \to \triangleright & \}.
\end{array}$$

[3]In examples, we denote a synchronized nonterminal (A, φ) by $A\langle\varphi\rangle$ in order to enhance readability.

Consider the second table, for instance. In each rule the nonterminals in the right-hand side are equipped with distinct synchronization symbols. This makes sure that the corresponding subderivations can choose rules independently of each other. On the other hand, if there are two occurrences of $S\langle\varphi\rangle$ (i.e., both are synchronized with each other) and table 2 is applied to them, both will be replaced with $S\langle\varphi 0\rangle S\langle\varphi 1\rangle$ so that the synchronization carries over to the corresponding pairs of replacing nonterminals. Note also what happens if table 2 is applied to $S\langle\varphi\rangle$ and $S^{\mathrm{r}}\langle\varphi\rangle$. These are replaced with $S\langle\varphi 0\rangle S\langle\varphi 1\rangle$ and $S^{\mathrm{r}}\langle\varphi 1\rangle S^{\mathrm{r}}\langle\varphi 0\rangle$ so that the left (right) subderivation of the former will be synchronized with the right (resp. left) subderivation of the latter. In contrast to table 2 the two occurrences of each right-hand side in table 3 are synchronized with each other, which ensures that only substrings of the form $[u|u^{\mathrm{r}}]$ are generated.

A sample derivation, where the subscripts of '$\Rightarrow$' indicate the tables applied to the nonterminals, is

$$
\begin{array}{rcl}
S\langle\rangle & \underset{2}{\Rightarrow} & S\langle 0\rangle S\langle 1\rangle \\
& \underset{1,3}{\Rightarrow} & [S\langle 00\rangle][S\langle 10\rangle|S^{\mathrm{r}}\langle 10\rangle] \\
& \underset{2,1,1}{\Rightarrow} & [S\langle 000\rangle S\langle 001\rangle][[S\langle 100\rangle]|[S^{\mathrm{r}}\langle 100\rangle]] \\
& \underset{4,1,2,2}{\Rightarrow} & [\triangleright[S\langle 0010\rangle]][[S\langle 1000\rangle S\langle 1001\rangle]|[S^{\mathrm{r}}\langle 1001\rangle S^{\mathrm{r}}\langle 1000\rangle]] \\
& \underset{3,4,1,1,4}{\Rightarrow} & [\triangleright[[S\langle 00100\rangle|S^{\mathrm{r}}\langle 00100\rangle]]][[\triangleright[S\langle 10010\rangle]]|[[S^{\mathrm{r}}\langle 10010\rangle]\triangleleft]] \\
& \underset{5,5,2,2}{\Rightarrow} & [\triangleright[[\triangleleft|\triangleright]]][[\triangleright[S\langle 100100\rangle S\langle 100101\rangle]]|[[S^{\mathrm{r}}\langle 100101\rangle S^{\mathrm{r}}\langle 100100\rangle]\triangleleft]] \\
& \underset{1,4,4,1}{\Rightarrow} & [\triangleright[[\triangleleft|\triangleright]]][[\triangleright[[S\langle 1001000\rangle]\triangleright]]|[[\triangleleft[S^{\mathrm{r}}\langle 1001000\rangle]]\triangleleft]] \\
& \underset{4,4}{\Rightarrow} & [\triangleright[[\triangleleft|\triangleright]]][[\triangleright[[\triangleright]\triangleright]]|[[\triangleleft[\triangleleft]]\triangleleft]].
\end{array}
$$

The example illustrates how the synchronization mechanism of branching ET0L systems extends the one of ordinary ET0L systems. In an ET0L system each derivation step consists of choosing a table and applying it to all nonterminals in the current sentential form. Branching ET0L systems are more flexible in that they allow to release the synchronization between certain subderivations (see $R(2)$ above) while retaining the synchronization between others (see $R(3)$).

The example reveals a general property of derivations: The synchronization between nonterminals can never be restored again, once it has been released (see also Lemma 4.1). By definition, being synchronized is an equivalence relation between the nonterminals of a given sentential form. Nonterminals with the same synchronization string belong to the same group. In other words, the nonterminals of the sentential form are divided into disjoint groups (i.e., equivalence classes), where those in the same group are synchronized with each other but those in different groups are not. In a derivation step, every such group gives rise to at most $|I|$ new groups. Such a new group consists of all descendants of the original group that have been supplied with the same last synchronization symbol. In this sense, the groups can split in a derivation step—the synchronization branches. Note that a derivation thus creates a tree of groups of nonterminals. The root is the group consisting of the initial synchronized nonterminal. The descendants of a group are the new ones it gives rise to. This observation will lead to the notion of synchronization trees in Section 7, where it is used to prove the second direction of the main result. Note finally that, in the example above, the number and size of the groups in a sentential form can grow arbitrarily large, due to the repeated application of tables $R(2)$ and $R(3)$, respectively.

Let us now turn to branching grammars. Here, the idea is to have n possible levels of synchronization instead of just one. This is accomplished by replacing J with J^n in the table specification (i.e., tables are now specified by n-tuples of table symbols), and by augmenting the nonterminals in the right-hand sides of rules with elements of I^n instead of I (i.e., with n-tuples of synchronization symbols).

Definition 3.2 (branching grammar) Let $n \in \mathbb{N}$. A *grammar with branching synchronization and nested tables* (*branching grammar*, for short) is a tuple $G = (N, T, I, J, R, S)$ where

- N and T are disjoint alphabets of *nonterminals* resp. *terminals*,
- I and J are nonempty alphabets of *synchronization symbols* resp. *table symbols*,
- R, the *table specification*, assigns to every $\tau \in J^n$ a finite set $R(\tau)$ of rules $A \to \zeta$ such that $A \in N$ and $\zeta \in ((N \times I^n) \cup T)^*$, and
- $S \in N$ is the *initial nonterminal.*

The number n is the *nesting depth* (or just *depth*) of G.

A set of rules $R(\tau)$, $\tau \in J^n$, is called a *table of* G. When there is no risk of confusion we may also call τ a table, and we may use τ to refer to $R(\tau)$. In particular, we say that a rule r is *taken from table* τ if $r \in R(\tau)$. It is important to note that the tables need not be disjoint. Thus, a rule r may be taken from several different tables. Intuitively, the table specification organizes the tables in a tree-like way. This gives rise to the following inductive definition of *supertables.* The supertables at nesting depth n are the actual tables $R(\tau)$ where $\tau \in J^n$. Those at nesting depth k, $0 \leq k < n$, are obtained by taking the union of the supertables at nesting depth $k+1$: $R(\tau) = \bigcup_{j \in J} R(\tau.j)$ for all $\tau \in J^k$. Equivalently, $R(\tau) = \bigcup\{R(\tau') \mid \tau' \in J^n, \mathrm{first}_k(\tau') = \tau\}$. In particular, $R() = \bigcup_{\tau \in J^n} R(\tau)$ is the unique supertable at depth 0, which consists of all rules of G. In the following, we will simply denote $R()$ by R, thus writing, e.g., $r \in R$ in order to express the fact that r occurs in at least one of the tables of G.

The terminology introduced earlier for branching ET0L systems carries over to branching grammars, as follows. Let G be as in Definition 3.2. An element of $(I^n)^*$ is a *synchronization string.* In other words, a synchronization string is a string of n-tuples of synchronization symbols (see the respective definitions in Section 2). An element of $\mathrm{SN}_G = N \times (I^n)^*$, consisting of a nonterminal and a synchronization string, is called a *synchronized nonterminal* (of G). The initial synchronized nonterminal of G is (S, λ).

Derivations will yield sentential forms $\xi = w_0(A_1, \varphi_1)w_1 \cdots (A_h, \varphi_h)w_h$ where $h \in \mathbb{N}$, $w_0, \ldots, w_h \in T^*$, $(A_1, \varphi_1), \ldots, (A_h, \varphi_h) \in \mathrm{SN}_G$, and $|\varphi_1| = \cdots = |\varphi_h|$. As a convention, denoting a string ξ in this manner is from now on always meant to imply that $h \in \mathbb{N}$, $w_0, \ldots, w_h \in T^*$, and $(A_1, \varphi_1), \ldots, (A_h, \varphi_h) \in \mathrm{SN}_G$, where all synchronization strings $\varphi_1, \ldots, \varphi_h$ have equal length. Note that this convention applies to the right-hand sides of rules as well, which are written as $v_0(B_1, \alpha_1)v_1 \cdots (B_l, \alpha_l)v_l$. In this case the synchronization strings α_i are of length 1, of course.

As in a derivation of a branching ET0L system, synchronization strings are accumulated in the nonterminals[4] of a sentential form. Thus, for branching grammars, we use the same definition of the auxiliary step relation (except that now strings of n-tuples of synchronization symbols are concatenated), as follows. For every $(A, \varphi) \in \mathrm{SN}_G$ and every rule

$$r = A \to v_0(B_1, \alpha_1)v_1 \cdots (B_l, \alpha_l)v_l$$

we define

$$(A, \varphi) \Rightarrow_r v_0(B_1, \varphi\alpha_1)v_1 \cdots (B_l, \varphi\alpha_l)v_l.$$

Before we turn to the definition of a general derivation step, where a string contains several terminals and nonterminals (Definition 3.3), let us discuss how synchronization works. Suppose a sentential form contains synchronized nonterminals (A, φ) and (B, ψ). Since derivations start with (S, λ) and are fully parallel, the accumulated synchronization strings φ and ψ have the same length, namely the length m of the derivation. Thus, $\varphi = \alpha_1 \cdots \alpha_m$ and $\psi = \beta_1 \cdots \beta_m$ for certain $\alpha_1, \beta_1, \ldots, \alpha_m, \beta_m \in I^n$. Writing the α_i and β_i as column vectors we can therefore view φ and ψ as $n \times m$-matrices of synchronization symbols:

$$\varphi = \begin{matrix} \alpha_{1,1} & \cdots & \alpha_{1,m} \\ \alpha_{2,1} & \cdots & \alpha_{2,m} \\ \vdots & \ddots & \vdots \\ \alpha_{n,1} & \cdots & \alpha_{n,m} \end{matrix} \qquad \psi = \begin{matrix} \beta_{1,1} & \cdots & \beta_{1,m} \\ \beta_{2,1} & \cdots & \beta_{2,m} \\ \vdots & \ddots & \vdots \\ \beta_{n,1} & \cdots & \beta_{n,m} \end{matrix}$$

where $\alpha_i = (\alpha_{1,i}, \ldots, \alpha_{n,i})$ and $\beta_i = (\beta_{1,i}, \ldots, \beta_{n,i})$ for $i \in [m]$. Now, the number k of rows, counted from the top, up to which both matrices are equal determines how tightly (A, φ) and (B, ψ) are synchronized. We call this number their level of synchronization. The rules to be applied to them must be taken from the same supertable at depth k but can be chosen independently within one such supertable. Thus, if $k = 0$ (i.e., $\alpha_{1,1} \cdots \alpha_{1,m} \neq \beta_{1,1} \cdots \beta_{1,m}$) then the requirement means that both rules must be in R, the supertable at depth 0. Since this is a trivial restriction, there is no synchronization at all in this case. If $k = 1$, i.e., $\alpha_{1,1} \cdots \alpha_{1,m} = \beta_{1,1} \cdots \beta_{1,m}$ but $\alpha_{2,1} \cdots \alpha_{2,m} \neq \beta_{2,1} \cdots \beta_{2,m}$, then the rules must be taken from the same supertable at depth 1. The highest level of synchronization is given by $k = n$, i.e., $\varphi = \psi$, in which case the rules must be taken from the same supertable at depth n, in other words, from the same table. Thus, nesting depth n leads to n increasingly strong levels of synchronization (not counting level 0).

To formalize this, suppose we are given a set X. Recall that, for every n-tuple $\gamma = (x_1, \ldots, x_n) \in X^n$ and every $k \in \{0, \ldots, n\}$, $\mathrm{first}_k(\gamma) = (x_1, \ldots, x_k)$. This extends to strings of n-tuples in the usual way: $\mathrm{first}_k(s) = \mathrm{first}_k(\gamma_1) \cdots \mathrm{first}_k(\gamma_m)$ for all $s = \gamma_1 \cdots \gamma_m \in (X^n)^*$. Hence, first_k maps X^n to X^k and its extension to strings maps $(X^n)^*$ to $(X^k)^*$. With respect to the matrix notation used above this means that first_k yields the first k rows, cutting off rows $k+1, \ldots, n$. Now, for $s, s' \in (X^n)^*$ with $|s| = |s'|$ let

$$\mathrm{level}(s, s') = \max\{k \in \{0, \ldots, n\} \mid \mathrm{first}_k(s) = \mathrm{first}_k(s')\}.$$

[4]We will often speak of nonterminals instead of synchronized nonterminals if there is no danger of confusion.

Note that $\mathrm{level}(s, s) = n$. In particular, $\mathrm{level}(\lambda, \lambda) = n$, which is slightly ambiguous but should not lead to confusion as n will always be clear from the context. Note also that $\mathrm{level}(s, s') = n$ implies that $s = s'$.

Thus, for synchronized nonterminals (A, φ) and (B, ψ), $\mathrm{level}(\varphi, \psi)$ yields their level of synchronization, as discussed informally above. Regarding tables $\tau, \tau' \in J^n$, $\mathrm{first}_k(\tau) = \mathrm{first}_k(\tau')$ means that $R(\tau)$ and $R(\tau')$ are included in the same supertable at depth k. Hence, $\mathrm{level}(\tau, \tau')$ yields the depth of their least common supertable.

The synchronization requirement discussed above can now be reformulated as follows: The rules applied to synchronized nonterminals $(A, \varphi), (B, \psi)$ must be taken from tables τ and τ' such that $\mathrm{level}(\tau, \tau') \geq \mathrm{level}(\varphi, \psi)$, i.e., such that $\mathrm{first}_k(\tau) = \mathrm{first}_k(\tau')$ for $k = \mathrm{level}(\varphi, \psi)$. In general, we may of course have to replace more than just two nonterminals. Then the requirements must be satisfied simultaneously, which leads to the following definition.

Definition 3.3 (derivation and generated language) Let $G = (N, T, I, J, R, S)$ be a branching grammar of nesting depth n and consider strings $\xi_1, \xi_2 \in (\mathrm{SN}_G \cup T)^*$ where $\xi_1 = w_0(A_1, \varphi_1)w_1 \cdots (A_h, \varphi_h)w_h$. Then there is a *derivation step* $\xi_1 \Rightarrow \xi_2$ if there are tables $\tau_1, \ldots, \tau_h \in J^n$ and rules $r_1 \in R(\tau_1), \ldots, r_h \in R(\tau_h)$ such that

(i) $\xi_2 = w_0\zeta_1 w_1 \cdots \zeta_h w_h$ for some $\zeta_1, \ldots, \zeta_h$ with $(A_j, \varphi_j) \Rightarrow_{r_j} \zeta_j$ for all $j \in [h]$ and

(ii) $\mathrm{level}(\tau_i, \tau_j) \geq \mathrm{level}(\varphi_i, \varphi_j)$ for all $i, j \in [h]$.

If condition (ii) is satisfied we say that $\tau_1, \ldots, \tau_h$ *are consistent with* $\varphi_1, \ldots, \varphi_h$.

In situations where several branching grammars are considered, we may write $\Rightarrow_G$ instead of $\Rightarrow$ to avoid confusion. The *language generated by* G is $L(G) = \{w \in T^* \mid (S, \lambda) \Rightarrow^* w\}$. The class of all languages which can be generated by branching grammars of nesting depth n is denoted by BS_n, and $\mathrm{BS} = \bigcup_{n \in \mathbb{N}} \mathrm{BS}_n$ (where BS stands for *branching synchronization*).

Note that, as for ET0L systems and branching ET0L systems, the derivation steps of a branching grammar are fully parallel.

Note also that the choice of $r_1, \ldots, r_h$ in the definition above does not depend on the actual synchronization symbols which appear in $\varphi_1, \ldots, \varphi_h$. The only value that matters is $\mathrm{level}(\varphi_i, \varphi_j)$. In other words, an (injective) renaming or permutation of synchronization symbols does not affect the language generated by a branching grammar. A similar remark holds for the symbols in J, of course. Furthermore, given a branching grammar G one may always enlarge I and J by additional symbols without affecting the generated language. Regarding I this is obvious and for J one can simply define $R(\tau) = \emptyset$ for all tables τ containing new elements of J.

Note finally that in Definition 3.3(ii) we do not require that $\mathrm{level}(\tau_i, \tau_j) = \mathrm{level}(\varphi_i, \varphi_j)$; this corresponds to the fact that the tables τ_i, τ_j have to be chosen from the same supertable at depth $k = \mathrm{level}(\varphi_i, \varphi_j)$, but need not be chosen from different supertables at depth $k + 1$ if $k < n$.

Clearly, for branching grammars of nesting depth 1 the definition of derivations coincides with the one for branching ET0L systems. This is because, in that case, we either have $\varphi_i = \varphi_j$ and thus $\mathrm{level}(\varphi_i, \varphi_j) = 1$ or $\varphi_i \neq \varphi_j$ and thus $\mathrm{level}(\varphi_i, \varphi_j) = 0$ (and similarly for

τ_i and τ_j). Hence, BS_1 is the class of languages generated by branching ET0L systems, thus including the class of all languages generated by ordinary ET0L systems. In fact, similar to the example discussed in connection with branching ET0L systems above, one can easily construct a branching ET0L system which generates the language of all strings $u|u^{\mathrm{r}}$, where u is an element of the Dyck language over two pairs of parentheses (and the reflection -^{r} is extended to the second pair of parentheses in the obvious way). By results of Skyum [Sky76] and Engelfriet, Rozenberg, and Slutzki [ERS80] this language is not an element of ET0L.

Conversely, an ET0L system may be seen as a branching grammar (of any depth ≥ 1) with one synchronization symbol, i.e., $|I| = 1$. Moreover, generalizing the remark that context-free grammars correspond to ET0L systems with only one table, a context-free grammar may be considered to be a branching grammar (of any depth ≥ 0) with only one table symbol, i.e., $|J| = 1$. Furthermore, if a branching grammar has depth 0 then I and J do not play any role because $I^0 = \{()\} = J^0$. This means that $\mathrm{BS}_0 = \mathrm{CF}$ and yields, together with the observations about ET0L above, the following theorem.

Theorem 3.4 $\mathrm{CF} = \mathrm{BS}_0$ and $\mathrm{ET0L} \subsetneq \mathrm{BS}_1$.

In a branching ET0L system, two nonterminals are either synchronized with each other or they are not. If they are not, the tables applied to them are chosen independently of each other. Otherwise the same table must be chosen. In other words, as discussed above, nesting depth 1 results in a single level of synchronization.

Consider again the branching ET0L system discussed as an example above. Suppose that we want to make sure that the (terminating) tables 4 and 5 are applied to all nonterminals in the same derivation step. Intuitively, this requires two groups of tables (or tables of tables, or supertables), namely tables 1–3 on the one hand and tables 4 and 5 on the other. To implement this new level of synchronization we must set $n = 2$ and turn R into R' with[5]

$$
\begin{array}{lcllll}
R'(1,1) & = & \{ & S \to [S\langle\begin{smallmatrix}0\\0\end{smallmatrix}\rangle], & S^{\mathrm{r}} \to [S^{\mathrm{r}}\langle\begin{smallmatrix}0\\0\end{smallmatrix}\rangle] & \},\\
R'(1,2) & = & \{ & S \to S\langle\begin{smallmatrix}0\\0\end{smallmatrix}\rangle S\langle\begin{smallmatrix}0\\1\end{smallmatrix}\rangle, & S^{\mathrm{r}} \to S^{\mathrm{r}}\langle\begin{smallmatrix}0\\1\end{smallmatrix}\rangle S^{\mathrm{r}}\langle\begin{smallmatrix}0\\0\end{smallmatrix}\rangle & \},\\
R'(1,3) & = & \{ & S \to [S\langle\begin{smallmatrix}0\\0\end{smallmatrix}\rangle | S^{\mathrm{r}}\langle\begin{smallmatrix}0\\0\end{smallmatrix}\rangle], & S^{\mathrm{r}} \to [S\langle\begin{smallmatrix}0\\0\end{smallmatrix}\rangle | S^{\mathrm{r}}\langle\begin{smallmatrix}0\\0\end{smallmatrix}\rangle] & \},\\
R'(2,4) & = & \{ & S \to \triangleright, & S^{\mathrm{r}} \to \triangleleft & \},\\
R'(2,5) & = & \{ & S \to \triangleleft, & S^{\mathrm{r}} \to \triangleright & \}.
\end{array}
$$

As a general convention for such examples, tables which are not specified are assumed to be empty. Hence, the supertable $R'(1)$ equals $R'(1,1) \cup R'(1,2) \cup R'(1,3)$ and $R'(2) = R'(2,4) \cup R'(2,5)$. The synchronization on level 1 accomplishes the distinction between terminating and nonterminating rules whereas the one on level 2 (which is carried over from the previous grammar) makes finer distinctions. Since the synchronization symbol corresponding to level 1 is the same for all occurrences of nonterminals in the right-hand sides, the synchronization between nonterminals is never released on this level. This makes sure that rules in $R'(2)$ will not be applied simultaneously with any of the rules in $R'(1)$. Except for this the behaviour is the same as before.

[5] In concrete examples such as this one we will denote the component φ of a synchronized nonterminal $A\langle\varphi\rangle$ in matrix form as discussed earlier.

As the example shows (for a simple case), the set of nonterminals of a sentential form of a branching grammar of nesting depth n is partitioned into groups of mutually synchronized nonterminals on each of the levels $1, \ldots, n$. In fact, nonterminals (A, φ) and (B, ψ) with $\mathrm{level}(\varphi, \psi) \geq k$, i.e., with $\mathrm{first}_k(\varphi) = \mathrm{first}_k(\psi)$, are in the same group at level k. The partition into groups at level $k+1$ is a refinement of the one at level k, i.e., every group at level k is a union of groups at level $k+1$. In particular, in a derivation step the groups can split only in such a way that branching at one level implies branching at all higher levels. Note that each group at level n (consisting of all nonterminals with the same synchronization string) branches into at most $|I^n|$ new groups of descendants at level n.

It follows from our main result (in particular Corollary 8.2), together with the results of [Eng82], that the classes $(\mathrm{BS}_n)_{n\in\mathbb{N}}$ constitute an infinite hierarchy which is strict on each level (see Corollary 8.5). An example in $\mathrm{BS}_2 \setminus \mathrm{BS}_1$ (known from [Eng82]) is the language $L_1 \subseteq \{a, b, \$\}^*$ defined as

$$L_1 = \{w\$w \mid w \in \{a,b\}^* \text{ and } \#_a(w) = 2^n \text{ for some } n \in \mathbb{N}\},$$

where $\#_a(w)$ counts the number of occurrences of a in w. (In fact, the previous example seems to be in $\mathrm{BS}_2 \setminus \mathrm{BS}_1$ as well, but we are not aware of an easy formal argument which shows that this is indeed the case.)

In order to generate L_1 by a branching grammar of depth 2 we use the synchronization symbols $0, 1$ and the table symbols $0, 1, 2$. The nonterminals are S (the initial nonterminal), A, and B, and the tables are

$$\begin{array}{rcl} R(0,0) & = & \{\ S \to A\langle{}^0_0\rangle\$A\langle{}^0_0\rangle\ \}, \\ R(1,0) & = & \{\ A \to A\langle{}^0_0\rangle A\langle{}^0_1\rangle\ \}, \\ R(2,0) & = & \{\ A \to B\langle{}^0_0\rangle aB\langle{}^0_1\rangle\ \}, \\ R(2,1) & = & \{\ B \to bB\langle{}^0_0\rangle\ \}, \\ R(2,2) & = & \{\ B \to \lambda\ \}. \end{array}$$

To understand the rationale behind these rules, notice the following facts. The table $R(0,0)$ will be applied exactly once, namely in the first derivation step. The rule in this table makes sure that the nonterminals on both sides of the \$ are synchronized with each other at level 2. During the remainder of the derivation this property carries over to the pairwise corresponding descendants of these two nonterminals, which guarantees that the same string is generated on both sides of the \$. Now consider the prefix to the left of the \$ and disregard the other half of the string. Clearly, in every sentential form all nonterminals in this part of the string are synchronized with each other at level 1. In other words, in each step the same supertable must be applied to all nonterminals, but within this supertable the tables can be chosen arbitrarily. Hence, the derivation corresponds to a derivation in an ET0L system whose tables are

$$\{A \to AA\} \text{ and } \{A \to BaB, B \to bB, B \to \lambda\}.$$

Consequently, a derivation first duplicates all A's n times, yielding A^{2^n}. When $R(2)$ is applied the first time, each A is replaced with BaB. Finally, the B's can independently produce an arbitrary number of b's each.

4 Some properties

In this section we discuss some of the basic properties of branching grammars and their generated languages. We start with a lemma which states a quite obvious, but nevertheless instructive property of $\mathrm{level}(\varphi, \psi)$. In particular, it shows that the level of synchronization of descendants of nonterminals cannot exceed the level of synchronization of their ancestors, as observed earlier.

Lemma 4.1 Let I be a set and $n \in \mathbb{N}$. For all $\varphi, \psi, \varphi', \psi' \in (I^n)^*$ with $|\varphi| = |\psi|$ and $|\varphi'| = |\psi'|$ it holds that $\mathrm{level}(\varphi\varphi', \psi\psi') = \min(\mathrm{level}(\varphi, \psi), \mathrm{level}(\varphi', \psi'))$.

Proof By the definition of first_k it holds that $\mathrm{first}_k(\varphi\varphi') = \mathrm{first}_k(\varphi)\mathrm{first}_k(\varphi')$ and $\mathrm{first}_k(\psi\psi') = \mathrm{first}_k(\psi)\mathrm{first}_k(\psi')$ for all $k \in \{0, \dots, n\}$. Hence, $\mathrm{first}_k(\varphi\varphi') = \mathrm{first}_k(\psi\psi')$ if and only if $\mathrm{first}_k(\varphi) = \mathrm{first}_k(\psi)$ and $\mathrm{first}_k(\varphi') = \mathrm{first}_k(\psi')$, which proves the lemma. ∎

To see what this means for a derivation $\xi \Rightarrow^* \xi'$ in a branching grammar, note that if nonterminals (A, φ) and (B, ψ) in ξ have descendants in ξ', then these have the form $(A', \varphi\varphi')$ respectively $(B', \psi\psi')$. Hence, if $k = \mathrm{level}(\varphi, \psi)$ is the level of synchronization of the original nonterminals then the level of synchronization of their descendants is $\min(k, \mathrm{level}(\varphi', \psi'))$. In particular, the synchronization level of the descendants cannot exceed the synchronization level of their ancestors. Note also that the synchronization level of descendants of the same ancestor is given by $\mathrm{level}(\varphi', \psi')$ because $\varphi = \psi$ and thus $k = n$ in this case.

Another simple, but useful observation is the following one. Recall from Definition 3.3 that, for a branching grammar $G = (N, T, I, J, R, S)$ of depth n and synchronization strings $\varphi_1, \dots, \varphi_h$ of equal length, tables $\tau_1, \dots, \tau_h$ are said to be consistent with $\varphi_1, \dots, \varphi_h$ if condition (ii) in that definition is satisfied, i.e., if $\mathrm{level}(\tau_i, \tau_j) \geq \mathrm{level}(\varphi_i, \varphi_j)$ for all $i, j \in [h]$. The next lemma states that a consistent choice of tables can always be extended in a consistent way, no matter which additional synchronization strings are to be considered.

Lemma 4.2 Let I and J be sets of synchronization resp. table symbols. Let $\varphi_1, \dots, \varphi_h, \varphi_{h+1}, \dots, \varphi_m \in (I^n)^*$ be synchronization strings of equal length, for $m \in \mathbb{N}$ and $h \in [m]$. If tables $\tau_1, \dots, \tau_h \in J^n$ are consistent with $\varphi_1, \dots, \varphi_h$ then there are $\tau_{h+1}, \dots, \tau_m \in J^n$ such that $\tau_1, \dots, \tau_m$ are consistent with $\varphi_1, \dots, \varphi_m$.

Proof It suffices to consider the case $m = h + 1$. Let $j \in [h]$ be such that $l = \mathrm{level}(\varphi_j, \varphi_{h+1})$ is maximal and set $\tau_{h+1} = \tau_j$. Then, for all $i \in [h]$ and $k \in [n]$, $\mathrm{first}_k(\varphi_i) = \mathrm{first}_k(\varphi_{h+1})$ implies $k \leq l$ and thus $\mathrm{first}_k(\varphi_i) = \mathrm{first}_k(\varphi_{h+1}) = \mathrm{first}_k(\varphi_j)$. Hence

$$\mathrm{level}(\tau_i, \tau_{h+1}) = \mathrm{level}(\tau_i, \tau_j) \geq \mathrm{level}(\varphi_i, \varphi_j) \geq \mathrm{level}(\varphi_i, \varphi_{h+1})$$

as required. ∎

Some of the constructions in the remainder of the paper work correctly only if the branching grammar in question has a certain property. The most important properties we are going to make use of are defined next. They are also of independent interest.

Definition 4.3 Let $G = (N, T, I, J, R, S)$ be a branching grammar of depth n and let $Q \subseteq R$ be a subset of its set of rules.

1. The set of left-hand sides of rules in Q is denoted by $\mathrm{lhs}(Q)$; thus, $\mathrm{lhs}(Q) \subseteq N$.

2. The set of rules Q is *total* if $\text{lhs}(Q) = N$ and *deterministic* if it does not contain distinct rules with the same left-hand side. The grammar G is *total* (*deterministic*) if every nonempty table $R(\tau)$, $\tau \in J^n$, is total (deterministic, respectively).

3. The grammar G is *terminable* if the following holds: for every $\xi \in (\text{SN}_G \cup T)^*$ with $(S, \lambda) \Rightarrow^+ \xi$ there exists some $w \in T^*$ such that $\xi \Rightarrow^* w$.

Note that a branching grammar may fail to be terminable due to unsatisfiable synchronization requirements. For example, if ξ contains nonterminals (A, φ) and (B, φ) but there is no table $\tau \in J^n$ such that $A, B \in \text{lhs}(R(\tau))$ then the derivation blocks as no further derivation step is possible.

The first example in Section 3 (generating the language L_0) is both deterministic and total as every table contains, for every nonterminal, exactly one rule having this nonterminal as its left-hand side. Similarly, the second example is total (since only nonempty tables are taken into account) and deterministic. The third example (generating the language L_1) is deterministic, but not total. In fact, none of its tables is total. Each of the three grammars is terminable; for instance, for a sentential form of the second grammar, the terminating table $R'(2, 4)$ can be applied to all its nonterminals.

What happens if we increase the depth of a branching grammar by dividing the tables into subtables without synchronizing any nonterminals on the new levels? Intuitively, this should not affect the generated language. We show now that this is indeed true. For this, consider two branching grammars $G = (N, T, I, J, R, S)$ and $G' = (N, T, I, J, R', S)$ of depth n respectively $n + m$. For a rule $r = A \to v_0(B_1, \beta_1)v_1 \cdots (B_l, \beta_l)v_l$ in R' let $r|_n$ denote the rule $A \to v_0(B_1, \alpha_1)v_1 \cdots (B_l, \alpha_l)v_l$, where $\alpha_i = \text{first}_n(\beta_i)$ for $i \in [l]$. We say that G' *is essentially* G if

(a) $R(\tau) = \{r|_n \mid r \in R'(\tau)\}$ for every (super)table $\tau \in J^n$ and

(b) $\text{level}(\beta_i, \beta_j) \leq n$ for every right-hand side $v_0(B_1, \beta_1)v_1 \cdots (B_l, \beta_l)v_l$ of a rule in R' and every pair of distinct $i, j \in [l]$.

Intuitively, if G' is essentially G, then a sentential form generated by G' becomes a sentential form of G by deleting the m last rows of synchronization symbols in every nonterminal. Furthermore, if two synchronized nonterminals (A, ψ), (B, ψ') occur at distinct positions in a sentential form of G' then $\text{level}(\psi, \psi') \leq n$. This guarantees that every derivation in G corresponds to some derivation in G'. We can therefore show that both generate the same language, as stated in the following lemma.

Lemma 4.4 If G, G' are branching grammars such that G' is essentially G, then $L(G') = L(G)$. Moreover, G is terminable if and only if G' is terminable.

Proof Let $G = (N, T, I, J, R, S)$ and $G' = (N, T, I, J, R', S)$, where the depth of G is n and that of G' is $n + m$. Let us say that two strings $\xi \in (\text{SN}_G \cup T)^*$ and $\eta \in (\text{SN}_{G'} \cup T)^*$ are *related* if they have the form

$$\xi = w_0(A_1, \varphi_1)w_1 \cdots (A_h, \varphi_h)w_h \quad \text{and} \quad \eta = w_0(A_1, \psi_1)w_1 \cdots (A_h, \psi_h)w_h$$

such that $\text{level}(\varphi_i, \varphi_j) = \text{level}(\psi_i, \psi_j)$ for all distinct $i, j \in [h]$. In other words, the synchronization between nonterminals in ξ is the same as the synchronization between nonterminals in η. Note that the initial synchronized nonterminals of G and G' are related and, by definition, a terminal string is related only to itself.

Consider two related strings $\xi \in (\mathrm{SN}_G \cup T)^*$ and $\eta \in (\mathrm{SN}_{G'} \cup T)^*$ as above. We prove two claims.

Claim 1 For every derivation step $\xi \Rightarrow_G \xi'$ there is a derivation step $\eta \Rightarrow_{G'} \eta'$ such that ξ' and η' are related.

To prove Claim 1 let $\xi \Rightarrow_G \xi' = w_0\zeta_1 w_1 \cdots \zeta_h w_h$ using rules $r_1, \ldots, r_h$ taken from tables $\tau_1, \ldots, \tau_h \in J^n$. For every $i \in [h]$ choose a table $\tau_i' \in J^{n+m}$ and a rule $r_i' \in R'(\tau_i')$ such that $\tau_i = \mathrm{first}_n(\tau_i')$ and $r_i = r_i'|_n$ (which is possible by condition (a)). Then we have

$$\mathrm{level}(\tau_i', \tau_j') \geq \mathrm{level}(\tau_i, \tau_j) \geq \mathrm{level}(\varphi_i, \varphi_j) = \mathrm{level}(\psi_i, \psi_j)$$

for all distinct $i, j \in [h]$. Hence $\eta \Rightarrow \eta' = w_0\zeta_1' w_1 \cdots \zeta_h' w_h$ where $(A_i, \psi_i) \Rightarrow_{r_i'} \zeta_i'$ for all $i \in [h]$. Since $r_i'|_n = r_i$, ζ_i and ζ_i' are equal up to the synchronization strings of their nonterminals. Consider two distinct occurrences (B, φ), (B', φ') of nonterminals in ξ' and let (B, ψ), (B', ψ') be the corresponding nonterminals in η'. To prove that ξ' and η' are related it remains to be shown that $\mathrm{level}(\varphi, \varphi') = \mathrm{level}(\psi, \psi')$.

Let ζ_i, ζ_j and ζ_i', ζ_j' be the substrings of ξ' resp. η' in which the considered nonterminals occur. Then $\varphi = \varphi_i\alpha$ and $\varphi' = \varphi_j\alpha'$ for some $\alpha, \alpha' \in I^n$, and $\psi = \psi_i\beta$ and $\psi' = \psi_j\beta'$ for some $\beta, \beta' \in I^{n+m}$. By the definition of $r_i'|_n$ and $r_j'|_n$, $\alpha = \mathrm{first}_n(\beta)$ and $\alpha' = \mathrm{first}_n(\beta')$ and hence $\mathrm{level}(\alpha, \alpha') = \min(n, \mathrm{level}(\beta, \beta'))$.

We can distinguish two cases. If $i \neq j$ then $\mathrm{level}(\psi_i, \psi_j) = \mathrm{level}(\varphi_i, \varphi_j) \leq n$ as ξ and η are related. Using Lemma 4.1 we get

$$\begin{aligned}
\mathrm{level}(\varphi, \varphi') &= \min(\mathrm{level}(\varphi_i, \varphi_j), \mathrm{level}(\alpha, \alpha')) \\
&= \min(\mathrm{level}(\psi_i, \psi_j), \min(n, \mathrm{level}(\beta, \beta'))) \\
&= \min(\mathrm{level}(\psi_i, \psi_j), \mathrm{level}(\beta, \beta')) \\
&= \mathrm{level}(\psi, \psi').
\end{aligned}$$

If $i = j$ then $\mathrm{level}(\beta, \beta') \leq n$ by condition (b) and so $\mathrm{level}(\alpha, \alpha') = \mathrm{level}(\beta, \beta')$. This yields

$$\begin{aligned}
\mathrm{level}(\varphi, \varphi') &= \min(\mathrm{level}(\varphi_i, \varphi_i), \mathrm{level}(\alpha, \alpha')) \\
&= \mathrm{level}(\alpha, \alpha') \\
&= \mathrm{level}(\beta, \beta') \\
&= \min(\mathrm{level}(\psi_i, \psi_i), \mathrm{level}(\beta, \beta')) \\
&= \mathrm{level}(\psi, \psi').
\end{aligned}$$

Hence, ξ' and η' are related.

Claim 2 For every derivation step $\eta \Rightarrow_{G'} \eta'$ there is a derivation step $\xi \Rightarrow_G \xi'$ such that ξ' and η' are related.

The proof is similar to the proof of Claim 1. Let $\eta \Rightarrow_{G'} \eta' = w_0\zeta_1' w_1 \cdots \zeta_h' w_h$ using rules $r_1', \ldots, r_h'$ taken from tables $\tau_1', \ldots, \tau_h' \in J^{n+m}$. For $i \in [h]$ let $\tau_i = \mathrm{first}_n(\tau_i')$. Then the rule $r_i = r_i'|_n$ is in $R(\tau_i)$ by condition (a). Furthermore,

$$\mathrm{level}(\tau_i, \tau_j) = \min(n, \mathrm{level}(\tau_i', \tau_j')) \geq \min(n, \mathrm{level}(\psi_i, \psi_j)) = \mathrm{level}(\varphi_i, \varphi_j)$$

for all $i, j \in [h]$. Consequently, $\xi \Rightarrow_G \xi' = w_0\zeta_1 w_1 \cdots \zeta_h w_h$ where $(A_i, \varphi_i) \Rightarrow_{r_i} \zeta_i$ for all $i \in [h]$. The situation is now identical to the one obtained in the proof of Claim 1, and hence the same arguments show that ξ' and η' are related. This completes the proof of Claim 2.

By induction, both claims carry over to derivations of arbitrary length. Since the initial synchronized nonterminals of G and G' are related, and a terminal string is related only to itself, this shows that $L(G') = L(G)$.

Finally, to see that G is terminable if and only if G' is terminable, notice that $(S, \lambda) \Rightarrow_G^+ \xi$ implies $(S, \lambda) \Rightarrow_{G'}^+ \eta$ for some η such that ξ and η are related (by Claim 1). If G' is terminable then there is some $w \in T^*$ such that $\eta \Rightarrow_{G'}^* w$, and hence $\xi \Rightarrow_G^* w$ (by Claim 2). Therefore, G is terminable if G' is terminable. For the other direction the argument is similar, which completes the proof. ∎

The following lemma uses the previous one in order to make grammars deterministic while keeping other desirable properties, by increasing the nesting depth of tables by one.

Lemma 4.5 For every branching grammar G of depth n there is a deterministic branching grammar G' of depth $n+1$ such that $L(G') = L(G)$. The construction preserves totality and terminability.

Proof Let $G = (N, T, I, J, R, S)$. As mentioned after Definition 3.3, I and J can be enlarged as needed. Since this obviously preserves totality and terminability, we may assume that $\mathcal{P}(R) \subseteq J$, and that $[m] \subseteq I$ where m is the maximal number of occurrences of nonterminals in the right-hand side of a rule in R.

To construct G', consider a rule

$$r = A \to v_0(B_1, \alpha_1)v_1 \cdots (B_l, \alpha_l)v_l$$

in R. We turn r into a new rule r' as follows:

$$r' = A \to v_0(B_1, \alpha_1.1)v_1 \cdots (B_s, \alpha_l.l)v_l.$$

Thus, no nonterminals are synchronized at synchronization level $n+1$.

Now, let $G' = (N, T, I, J, R', S)$ be the branching grammar of depth $n+1$ with the following tables. If $\tau' \in J^{n+1}$ is of the form $\tau' = \tau.Q$ for some $\tau \in J^n$ and some deterministic $Q \subseteq R(\tau)$ with $\mathrm{lhs}(Q) = \mathrm{lhs}(R(\tau))$, then $R'(\tau') = \{r' \mid r \in Q\}$. Otherwise, $R'(\tau') = \emptyset$. Intuitively, every table of G is split up into all its deterministic subtables, but the synchronization disregards this.

By construction, the supertables at depth n in G' are the tables of G. More precisely, $R(\tau) = \{r'|_n \mid r' \in R'(\tau)\}$ for all $\tau \in J^n$. Furthermore, in the right-hand sides of rules there is no synchronization at level $n+1$ between any pair of distinct nonterminals. In other words, G' is essentially G. By Lemma 4.4 this implies $L(G') = L(G)$ and that G' is terminable if G is. Finally, note that totality is preserved thanks to the condition $\mathrm{lhs}(Q) = \mathrm{lhs}(R(\tau))$ in the definition of R'. This completes the proof. ∎

A direct consequence of Lemma 4.5 is, of course, that $\mathrm{BS}_n \subseteq \mathrm{BS}_{n+1}$ for all $n \in \mathbb{N}$. Somewhat more interesting is the fact that the lemma can be used in order to implement copying: for all languages $L \in \mathrm{BS}_n$ the language $c_*(L) = \{(w\$)^m \mid w \in L, m \geq 1\}$ is in BS_{n+1}.[6] To see this, let $G = (N, T, I, J, R, S)$ be a deterministic branching grammar of

[6] This is one of the constructions used in [Eng82] in order to show that the hierarchy which, according to our main result, equals $(\mathrm{BS}_n)_{n \in \mathbb{N}}$ is strict on each level.

depth $n+1$ that generates L. To generate $c_*(L)$ we simply take a new initial nonterminal $S' \notin N$ and add two new tables $R(p^{n+1})$ and $R(q^{n+1})$, where p and q are new table symbols. Let $R(p^{n+1}) = \{S' \to (S', \alpha)(S', \alpha),\ S' \to (S', \alpha)\}$ and $R(q^{n+1}) = \{S' \to (S, \alpha)\$\}$ for some arbitrary $\alpha \in I^{n+1}$. (One could, in fact, also add the new rules to two distinct ones of the original tables of G. In this way one can avoid using new table symbols, provided that $|J| \geq 2$.) It should be clear that the resulting branching grammar G' generates $c_*(L)$: by applying $R(p^{n+1})$ a number of times, and then $R(q^{n+1})$ once, a sentential form $(S, \varphi)\$ \cdots (S, \varphi)\$$ is obtained which consists of $m \geq 1$ copies of $(S, \varphi)\$$. As the derivation continues, the determinism of the original grammar guarantees that only sentential forms $\xi\$ \cdots \xi\$$ are generated, where ξ is a sentential form of G (except for the fact that the prefix φ has been added to the synchronization strings of all nonterminals) which is copied m times. Note that the example discussed at the end of the previous section is based on a similar, though somewhat simpler, construction.

5 Tree grammars and tree transducers

We shall now focus on the generation of trees rather than strings. According to the definition of trees in Section 2, a tree is a particular kind of string. Therefore, the definition of branching tree grammars is very simple. They are just a special case of branching grammars, as follows.

A branching grammar $G = (N, \Sigma, I, J, R, S)$ is a *branching tree grammar* if its terminal alphabet Σ is ranked and the right-hand side of each rule is a tree in $\mathrm{T}_\Sigma(N \times I^n)$. Obviously, this implies that every sentential form is a tree in $\mathrm{T}_\Sigma(\mathrm{SN}_G)$. In particular, $L(G) \subseteq \mathrm{T}_\Sigma$. We denote by BST the class of tree languages generated by branching tree grammars, and by BST_n the class of tree languages generated by branching tree grammars of nesting depth n.

The following example of nesting depth 1 is similar to the first example in Section 3. Let $\Sigma_2 = \{\bullet, |\}$, $\Sigma_1 = \{\circ\}$, and $\Sigma_0 = \{\triangleright, \triangleleft\}$. We construct a branching tree grammar that generates all trees which, if drawn in the usual way, have the property that every node labelled $|$ is the root of a vertically symmetric subtree. Again, we need nonterminals S and S^{r} and two synchronization symbols. The tables are similar to the earlier example as well:

$$
\begin{array}{lcllll}
R(1) & = & \{ & S \to \circ[S\langle 0\rangle], & S^{\mathrm{r}} \to \circ[S^{\mathrm{r}}\langle 0\rangle] & \}, \\
R(2) & = & \{ & S \to \bullet[S\langle 0\rangle, S\langle 1\rangle], & S^{\mathrm{r}} \to \bullet[S^{\mathrm{r}}\langle 1\rangle, S^{\mathrm{r}}\langle 0\rangle] & \}, \\
R(3) & = & \{ & S \to |[S\langle 0\rangle, S^{\mathrm{r}}\langle 0\rangle], & S^{\mathrm{r}} \to |[S\langle 0\rangle, S^{\mathrm{r}}\langle 0\rangle] & \}, \\
R(4) & = & \{ & S \to \triangleright, & S^{\mathrm{r}} \to \triangleleft & \}, \\
R(5) & = & \{ & S \to \triangleleft, & S^{\mathrm{r}} \to \triangleright & \}.
\end{array}
$$

A derivation is shown in Figure 1. To save space the synchronization strings are written beneath the nonterminals.

One can turn this example into an example of nesting depth 2 by dividing the set of tables into two supertables consisting of $R(1), R(2), R(3)$ on the one hand and $R(4), R(5)$ on the other, while choosing the synchronization in such a way that it is never released on the first level and is as above on the second level (similar to the second example in Section 3). In this way, the derivations are restricted to yield only fully balanced trees,

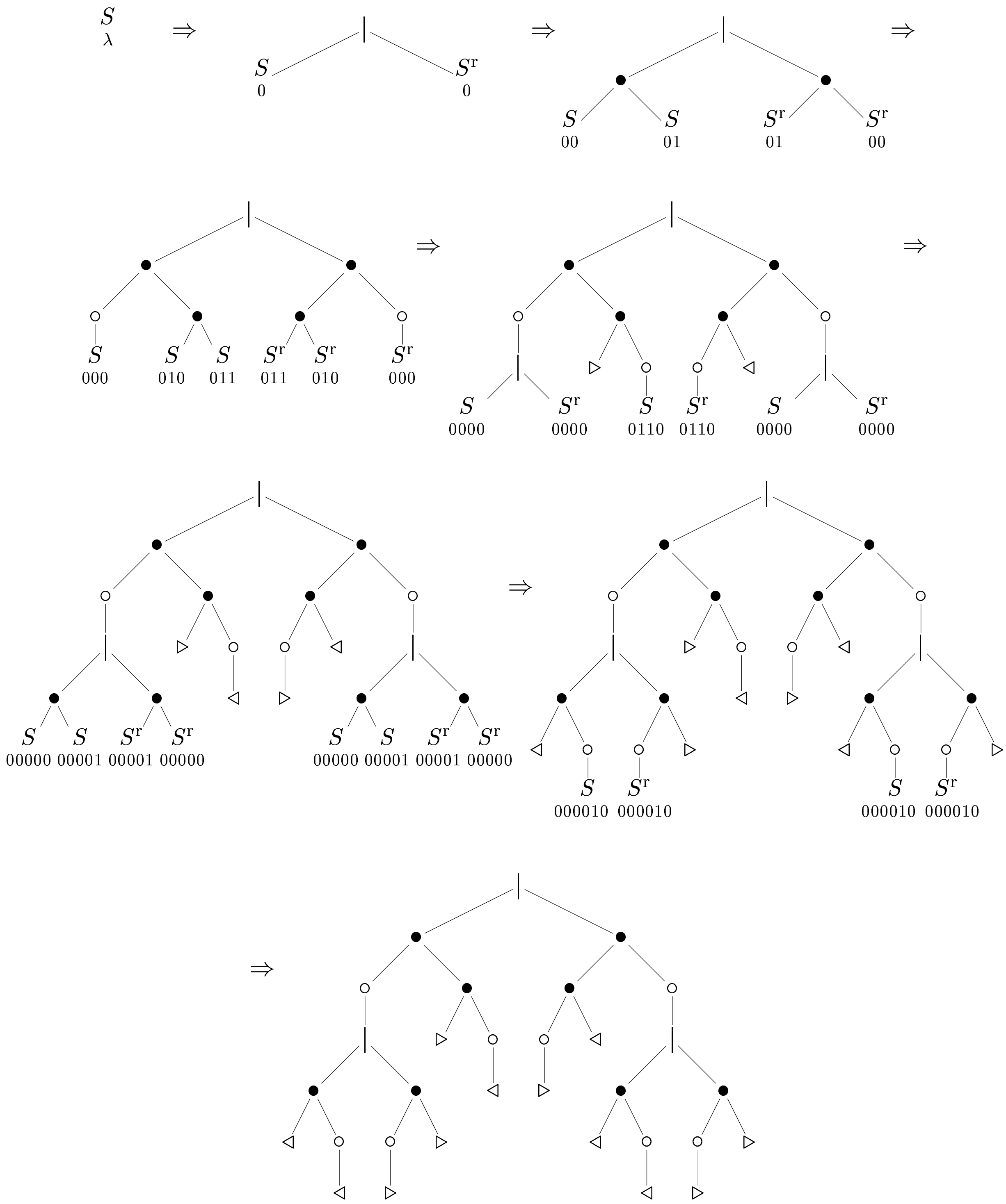


Figure 1: A derivation

since the terminating rules in $R(4)$ and $R(5)$ must be applied to all nonterminals at the same time.

Next, as another example, we shall discuss a family of tree languages $L_0, L_1, \ldots$ generated by branching tree grammars of depth $0, 1, \ldots$. Let e be a symbol of rank 0 and $\sigma_0, \sigma_1, \ldots$ symbols of rank 2. The tree language L_0 is simply $\mathrm{T}_{\{\sigma_0, e\}}$. For $i \geq 1$, L_i is the tree language $\{t[\![e \leftarrow t']\!] \mid t \in L_{i-1}, t' \in \mathrm{T}_{\{\sigma_i, e\}}\}$, where $t[\![e \leftarrow t']\!]$ denotes the result of substituting t' for every occurrence of e in t. Thus, according to this definition, a tree in L_i consists of an arbitrary tree in L_{i-1}, in which copies of a single tree over $\{\sigma_i, e\}$ are substituted for the leaves. Equivalently, the tree can be divided into layers 0 (topmost) up to i (lowest), the kth layer being a (possibly empty) forest with nodes labelled σ_k, such that each layer consists of copies of a single tree.

Clearly, $L_0 \in \mathrm{BST}_0$; we need one nonterminal S_0 and the rules $S_0 \to e$, $S_0 \to \sigma_0[S_0, S_0]$. The language L_1 is generated by a branching grammar of nesting depth 1, where $I = \{0, 1\}$ and $J = \{\mathit{next}, 0, a, b\}$. For this, we use nonterminals S_0 and S_1, where S_0 is still the initial one, and the following table specification:

$$\begin{array}{lcl} R(\mathit{next}) & = & \{S_0 \to S_1\langle 0 \rangle,\ S_1 \to e\} \\ R(0) & = & \{S_0 \to \sigma_0[S_0\langle 0 \rangle, S_0\langle 0 \rangle],\ S_0 \to S_0\langle 0 \rangle\} \\ R(a) & = & \{S_1 \to \sigma_1[S_1\langle 0 \rangle, S_1\langle 1 \rangle]\} \\ R(b) & = & \{S_1 \to S_1\langle 0 \rangle\}. \end{array}$$

Thus, layer 0 is generated nondeterministically by table $R(0)$ whereas the first (and only) level of synchronization is exploited to make sure that the trees of layer 1, generated by tables $R(a)$ and $R(b)$, are identical. Table $R(\mathit{next})$ is used to switch from one layer to the next, and to terminate. Strictly speaking, table $R(b)$ is not needed, but its presence makes it easier to see the pattern in the next step. In fact, table $R(b)$ allows all nonterminals S_1 to terminate at the same time; instead, they could all switch to layer 2 when it comes to the generation of L_2.

To generate L_2, we let $J = \{\mathit{next}, 0, 1, a, b\}$, add a new nonterminal S_2, and define the tables as follows:

$$\begin{array}{lcl} R(\mathit{next}, \mathit{next}) & = & \{S_0 \to S_1\langle \binom{0}{0} \rangle,\ S_1 \to S_2\langle \binom{0}{0} \rangle,\ S_2 \to e\} \\ R(0, 0) & = & \{S_0 \to \sigma_0[S_0\langle \binom{0}{0} \rangle, S_0\langle \binom{0}{0} \rangle],\ S_0 \to S_0\langle \binom{0}{0} \rangle\} \\ R(1, a) & = & \{S_1 \to \sigma_1[S_1\langle \binom{0}{0} \rangle, S_1\langle \binom{0}{1} \rangle]\} \\ R(1, b) & = & \{S_1 \to S_1\langle \binom{0}{0} \rangle\} \\ R(a, a) & = & \{S_2 \to \sigma_2[S_2\langle \binom{0}{0} \rangle, S_2\langle \binom{1}{1} \rangle]\} \\ R(b, b) & = & \{S_2 \to S_2\langle \binom{0}{0} \rangle\}. \end{array}$$

Thus, layer 0 is generated nondeterministically as before, while the second and first level of synchronization are used to obtain arbitrary but identical trees in layer 1 and 2, respectively.

In general, to generate L_i, we let $I = \{0, 1\}$, $J = \{\mathit{next}, 0, \ldots, i-1, a, b\}$, we take nonterminals $S_0, \ldots, S_i$ with initial nonterminal S_0, and we define the tables

$$
\begin{aligned}
R(next, \ldots, next) &= \{S_0 \to S_1 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle, \ldots,\ S_{i-1} \to S_i \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle,\ S_i \to e\} \\[2ex]
R(0, \ldots, 0) &= \{S_0 \to \sigma_0[S_0 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle, S_0 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle],\ S_0 \to S_0 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle\} \\[2ex]
R(1, \ldots, 1, a) &= \{S_1 \to \sigma_1[S_1 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \\ 0 \end{smallmatrix} \right\rangle, S_1 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{smallmatrix} \right\rangle]\} \\
R(1, \ldots, 1, b) &= \{S_1 \to S_1 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle\} \\[2ex]
R(2, \ldots, 2, a, a) &= \{S_2 \to \sigma_2[S_2 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{smallmatrix} \right\rangle, S_2 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 1 \end{smallmatrix} \right\rangle]\} \\
R(2, \ldots, 2, b, b) &= \{S_2 \to S_2 \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle\} \\
&\ \ \vdots \\
R(\underbrace{k, \ldots, k}_{i-k}, \underbrace{a, \ldots, a}_{k}) &= \{S_k \to \sigma_k[S_k \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle, S_k \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 1 \end{smallmatrix} \right\rangle]\} \\
R(k, \ldots, k, b, \ldots, b) &= \{S_k \to S_k \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle\} \\
&\ \ \vdots \\
R(i-1, a, \ldots, a) &= \{S_{i-1} \to \sigma_{i-1}[S_{i-1} \left\langle \begin{smallmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle, S_{i-1} \left\langle \begin{smallmatrix} 0 \\ 1 \\ \vdots \\ 1 \end{smallmatrix} \right\rangle]\} \\
R(i-1, b, \ldots, b) &= \{S_{i-1} \to S_{i-1} \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle\} \\[2ex]
R(a, \ldots, a) &= \{S_i \to \sigma_i[S_i \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle, S_i \left\langle \begin{smallmatrix} 1 \\ \vdots \\ 1 \end{smallmatrix} \right\rangle]\} \\
R(b, \ldots, b) &= \{S_i \to S_i \left\langle \begin{smallmatrix} 0 \\ \vdots \\ 0 \end{smallmatrix} \right\rangle\}.
\end{aligned}
$$

Note that there are k 1's in the right-hand side of the rule of table $R(k, \ldots, k, a, \ldots, a)$.

As before, layer 0 is generated nondeterministically, while the levels $i, i-1, \ldots, 1$ of synchronization are used to obtain arbitrary but identical trees in layers $1, 2, \ldots, i$, respectively. More precisely, for $k \in [i]$, when the nonterminals S_k are introduced by table $R(next, \ldots, next)$ they are all synchronized at a level $\geq i-k+1$. Then, during the generation of the tree in layer k, they are synchronized at a level $\geq i-k+1$ or at level $i-k$, depending on whether or not they occur at the same node of that tree, respectively.

This ends our presentation of examples of branching tree grammars. Let us now recall the definition of regular tree grammars. A *regular tree grammar* is a quadruple $G = (N, \Sigma, R, S)$ where N is the alphabet of nonterminals, Σ is the ranked alphabet of terminals, $S \in N$ is the initial nonterminal, and R is a finite set of rules $A \to \zeta$ where $A \in N$ and $\zeta \in \mathrm{T}_\Sigma(N)$. Hence, G is a context-free Chomsky grammar with the additional requirements that the terminal alphabet Σ is ranked and every right-hand side of a rule in R is a tree in $\mathrm{T}_\Sigma(N)$. The derivation relation $\Rightarrow$ and the generated language $L(G)$, which is called a *regular tree language*, are defined as usual. The class of all regular tree languages is denoted by REGT. Obviously, regular tree grammars can be identified with branching tree grammars of depth 0 (see also Theorem 3.4). Hence, $\mathrm{REGT} = \mathrm{BST}_0$.

It is well known that the class of context-free languages is the class of yields of regular tree languages. A similar result relates BS_n and BST_n.

Lemma 5.1 For every $n \in \mathbb{N}$, $\mathrm{BS}_n = \mathrm{yield}(\mathrm{BST}_n)$.

Proof Consider a tree $\xi = w_0 A_1 w_1 \cdots A_h w_h$ where $A_1, \dots, A_h$ are symbols of rank 0, and trees $\zeta_1, \dots, \zeta_h$. It follows by structural induction on ξ that, if $\mathrm{yield}(\xi) = v_0 A_1 v_1 \cdots A_h v_h$ then $\mathrm{yield}(w_0 \zeta_1 w_1 \cdots \zeta_h w_h) = v_0 \mathrm{yield}(\zeta_1) v_1 \cdots \mathrm{yield}(\zeta_h) v_h$. Hence, for every branching tree grammar $G = (N, \Sigma, I, J, R, S)$ the branching grammar $G' = (N, \Sigma_0 \setminus \{\epsilon\}, I, J, R', S)$ obtained from G by replacing every rule $A \to \zeta$ with $A \to \mathrm{yield}(\zeta)$ satisfies $L(G') = \mathrm{yield}(L(G))$. This proves that $\mathrm{yield}(\mathrm{BST}_n) \subseteq \mathrm{BS}_n$. Conversely, if G' is given it is easy to turn every right-hand side ζ' into a tree ζ such that $\mathrm{yield}(\zeta) = \zeta'$. Following the argument above, this yields a branching tree grammar G (of the same depth) such that $L(G') = \mathrm{yield}(L(G))$, which finishes the proof. ■

In order to recall top-down tree transducers, and also for later use, we need a few additional notations regarding substitution in trees. For a tree $t \in \mathrm{T}_\Sigma$, pairwise distinct symbols $a_1, \dots, a_k \in \Sigma_0$, and trees $s_1, \dots, s_k$ we denote by $t[\![a_1 \leftarrow s_1, \dots, a_k \leftarrow s_k]\!]$ the tree t' obtained by substituting s_i for every occurrence of a_i in t ($i \in [k]$). Formally, if $t = a_i$ then $t' = s_i$ and if $t = f[t_1, \dots, t_m]$ where $f \notin \{a_1, \dots, a_k\}$ then

$$t' = f[t_1[\![a_1 \leftarrow s_1, \dots, a_k \leftarrow s_k]\!], \dots, t_m[\![a_1 \leftarrow s_1, \dots, a_k \leftarrow s_k]\!]].$$

To increase the readability of formulas we may use set notation to denote substitutions, i.e., $t[\![A \leftarrow B \mid P]\!]$ where $A \leftarrow B$ is a generic expression including free variables, and P is a condition that determines the range of the free variables. For example, $t[\![a_1 \leftarrow s_1, \dots, a_k \leftarrow s_k]\!]$ may be abbreviated as $t[\![a_i \leftarrow s_i \mid i \in [k]]\!]$. Sometimes even this notation results in formulas which are hard to read—in particular if nested substitutions occur. In these cases we may even omit the part "$\mid P$", indicating the range of variables in the text instead.

Substitution will often be used in order to indicate the structure of a tree by decomposing it into a "context part" and a finite number of subtrees. For this, let us designate a countably infinite set $X = \{x_1, x_2, \dots\}$ of *variables*, special symbols of rank 0 that are only used for this purpose. In the following, we denote a tree s in the form $t[\![s_1, \dots, s_k]\!]$ in order to indicate that $s = t[\![x_1 \leftarrow s_1, \dots, x_k \leftarrow s_k]\!]$, where every variable x_i ($i \in [k]$) occurs in t exactly once (and no further variables x_j, $j > k$, occur in t). In other words, every s_i corresponds to a specific occurrence of that subtree in s. Note that, with this notation, derivation steps in a branching tree grammar $G = (N, \Sigma, I, J, R, S)$ of depth n can be characterized as follows. Let $\xi_1, \xi_2 \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$ where $\xi_1 = t[\![(A_1, \varphi_1), \dots, (A_h, \varphi_h)]\!]$.[7]

[7]Similarly to the string case, we adopt here and in the following the convention that the notation $t[\![(A_1, \varphi_1), \dots, (A_h, \varphi_h)]\!]$ is meant to imply that t does not contain any further nonterminals.

There is a derivation step $\xi_1 \Rightarrow_G \xi_2$ if and only if there are tables $\tau_1, \dots, \tau_h \in J^n$ which are consistent with $\varphi_1, \dots, \varphi_h$ and rules $r_i \in R(\tau_i)$ $(i \in [h])$ such that $\xi_2 = t[\![\zeta_1, \dots, \zeta_h]\!]$ where $(A_i, \varphi_i) \Rightarrow_{r_i} \zeta_i$ for all $i \in [h]$.

We shall also make use of language substitution, i.e., the extension of $[\![\cdots]\!]$ to languages of trees. The definition is similar to the one above: For a tree $t \in \mathrm{T}_\Sigma$, pairwise distinct symbols $a_1, \dots, a_k \in \Sigma_0$, and tree languages $L_1, \dots, L_k$, $t[\![a_1 \leftarrow L_1, \dots, a_k \leftarrow L_k]\!]$ yields the tree language L' given as follows. If $t = a_i$ then $L' = L_i$ and if $t = f[t_1, \dots, t_m]$ where $f \notin \{a_1, \dots, a_k\}$ then

$$L' = \{f[t'_1, \dots, t'_m] \mid t'_i \in t_i[\![a_1 \leftarrow L_1, \dots, a_k \leftarrow L_k]\!] \text{ for } i \in [m]\}.$$

Note that this is OI-substitution, which is the usual notion of substitution in formal language theory. With this kind of language substitution distinct occurrences of a_i in t may be replaced with different trees in L_i. More precisely, if $t = t_0[\![a_{i_1}, \dots, a_{i_l}]\!]$ contains exactly these l occurrences of symbols in $\{a_1, \dots, a_k\}$ then $t[\![a_1 \leftarrow L_1, \dots, a_k \leftarrow L_k]\!]$ equals the set of all trees of the form $t_0[\![s_1, \dots, s_l]\!]$ such that $s_j \in L_{i_j}$ for all $j \in [l]$.

For a tree language L we define

$$L[\![a_1 \leftarrow L_1, \dots, a_k \leftarrow L_k]\!] = \bigcup_{t \in L} t[\![a_1 \leftarrow L_1, \dots, a_k \leftarrow L_k]\!].$$

The following lemma, which states that substitution in trees is associative, is well known (cf. [Tha70a, Lemma 7.8] or [ES77, Corollary 2.4.2]).

Lemma 5.2 Let $t, s_1, \dots, s_k, s'_1, \dots, s'_l$ be trees and $a_1, \dots, a_k, b_1, \dots, b_l$ symbols of rank 0, where both the a_i $(i \in [k])$ and the b_j $(j \in [l])$ are pairwise distinct. If $b_1, \dots, b_l$ do not occur in t then

$$t[\![a_i \leftarrow s_i[\![b_j \leftarrow s'_j]\!]]\!] = t[\![a_i \leftarrow s_i]\!][\![b_j \leftarrow s'_j]\!]$$

where i ranges over $[k]$ and j over $[l]$. Similarly, if $L, L_1, \dots, L_k, L'_1, \dots, L'_l$ are tree languages such that $b_1, \dots, b_l$ do not occur in the trees in L then

$$L[\![a_i \leftarrow L_i[\![b_j \leftarrow L'_j]\!]]\!] = L[\![a_i \leftarrow L_i]\!][\![b_j \leftarrow L'_j]\!].$$

∎

For an alphabet Q and $k \in \mathbb{N}$, let $\langle Q, k \rangle$ denote the alphabet consisting of all pairs $\langle q, i \rangle$ where $q \in Q$ and $i \in [k]$. We use this in our definition of top-down tree transducers which is given next.

Definition 5.3 (td transducer) A *top-down tree transducer* (td transducer, for short) is a tuple $td = (\Sigma, \Sigma', Q, R, q_0)$ where

- Σ and Σ' are ranked alphabets of *input* resp. *output symbols*,
- Q is a ranked alphabet of *states*, all of rank 1,
- $q_0 \in Q$ is the *initial state*, and
- R is a finite set of rules of the form $qf \to \zeta$ where $q \in Q$, $f \in \Sigma_k$ for some $k \in \mathbb{N}$, and $\zeta \in \mathrm{T}_{\Sigma'}(\langle Q, k \rangle)$.

For trees $\xi_1, \xi_2 \in \mathrm{T}_{\Sigma \cup Q \cup \Sigma'}$, we write $\xi_1 \Rightarrow_{td} \xi_2$ (or simply $\xi_1 \Rightarrow \xi_2$) if there is a rule $qf \to \zeta$ in R such that $\xi_1 = \eta[\![qf[s_1, \ldots, s_k]]\!]$ and $\xi_2 = \eta[\![\zeta[\![\langle q', i\rangle \leftarrow q's_i \mid q' \in Q, i \in [k]]\!]]\!]$ for suitable trees $\eta \in \mathrm{T}_{\Sigma \cup Q \cup \Sigma'}(X)$ and $s_1, \ldots, s_k \in \mathrm{T}_\Sigma$, where k is the rank of f. The *top-down tree transduction (td transduction) computed by td*, denoted by td as well, is the set of all pairs $(s, s') \in \mathrm{T}_\Sigma \times \mathrm{T}_{\Sigma'}$ such that $q_0 s \Rightarrow^* s'$.

By the definition, every rule of a td transducer $td = (\Sigma, \Sigma', Q, R, q_0)$ can be written as $qf \to t[\![\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle]\!]$, where $q \in Q$, $f \in \Sigma_k$ for some $k \in \mathbb{N}$, $t \in \mathrm{T}_{\Sigma'}(X)$, and $\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle \in \langle Q, k\rangle$. The assumption $t \in \mathrm{T}_{\Sigma'}(X)$ implies that the right-hand side does not contain any further occurrences of elements of $\langle Q, k\rangle$ than $\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle$. We shall adopt this as a general convention in the following, i.e., if a rule is denoted in the form $qf \to t[\![\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle]\!]$ then $t \in \mathrm{T}_{\Sigma'}(X)$ (and, of course, $\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle \in \langle Q, k\rangle$).
Given a td transducer as in the definition and a state $q \in Q$ we let td_q denote the td transducer $(\Sigma, \Sigma', Q, R, q)$ obtained from td by taking q as its initial state. A rule of the form $qf \to \zeta$ is a *qf-rule*. A td tree transducer $td = (\Sigma, \Sigma', Q, R, q_0)$ is

- *total* if R contains at least one qf-rule for every $q \in Q$ and $f \in \Sigma$,
- *deterministic* if R contains at most one qf-rule for every $q \in Q$ and $f \in \Sigma$, and
- a *finite state relabelling* if the right-hand side of every qf-rule, where $f \in \Sigma_k$, has the form $g[\langle q_1, 1\rangle, \ldots, \langle q_k, k\rangle]$ (thus, finite state relabellings do not change the structure of input trees).

A td transduction is total (deterministic) if there exists a total (resp. deterministic) td transducer that computes it. Similarly, a td transduction is a finite state relabelling if there is a finite state relabelling computing it. The set of all td transductions is denoted by TD. Moreover, tTD, dTD, and RELAB denote the sets of total td transductions, deterministic td transductions, and finite state relabellings, respectively.
The following lemma gives convenient recursive formulations of how a td transducer computes its output trees.

Lemma 5.4 Let $td = (\Sigma, \Sigma', Q, R, q_0)$ be a td transducer, $q \in Q$, and $s = f[s_1, \ldots, s_k] \in \mathrm{T}_\Sigma$.

(1) For every tree $s' \in \mathrm{T}_{\Sigma'}$, there is a computation $qs \Rightarrow^* s'$ if and only if there exist a rule $qf \to t[\![\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle]\!]$ in R and, for all $j \in [l]$, computations $q_j s_{i_j} \Rightarrow^* s'_j$ such that $s' = t[\![s'_1, \ldots, s'_l]\!]$.

(2) Let $L_{qf} = \{\zeta \mid (qf \to \zeta) \in R\}$, i.e., L_{qf} is the set of all right-hand sides of qf-rules in R. Then
$$td_q(s) = L_{qf}[\![\langle q', i\rangle \leftarrow td_{q'}(s_i) \mid q' \in Q, i \in [k]]\!].$$

Proof The first claim is well known (see, e.g., [Eng75, Lemma 1.2]). We only prove the second, which is obtained from the first one, as follows. To show that $td_q(s) \subseteq L_{qf}[\![\langle q', i\rangle \leftarrow td_{q'}(s_i) \mid q' \in Q, i \in [k]]\!]$, consider a tree $s' \in td_q(s)$. By the definition of $td_q(s)$, $qs \Rightarrow^* s'$. By part (1) of the lemma this means that $s' = t[\![s'_1, \ldots, s'_l]\!]$ for some rule

$qf \to t[\![\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle]\!]$ in R and computations $q_j s_{i_j} \Rightarrow^* s'_j$. In other words, the tree $\zeta = t[\![\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle]\!]$ is in L_{qf} and $s'_j \in td_{q_j}(s_{i_j})$ for all $j \in [l]$. Hence we get

$$\begin{array}{rcl} s' & = & t[\![s'_1, \ldots, s'_l]\!] \\ & \in & \zeta[\![\langle q', i\rangle \leftarrow td_{q'}(s_i) \mid q' \in Q, i \in [k]]\!] \\ & \subseteq & L_{qf}[\![\langle q', i\rangle \leftarrow td_{q'}(s_i) \mid q' \in Q, i \in [k]]\!]. \end{array}$$

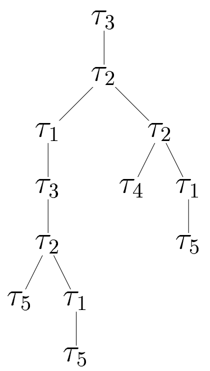
Conversely, by definition $s' \in L_{qf}[\![\langle q', i\rangle \leftarrow td_{q'}(s_i) \mid q' \in Q, i \in [k]]\!]$ means that there is a tree $\zeta \in L_{qf}$ such that $s' \in \zeta[\![\langle q', i\rangle \leftarrow td_{q'}(s_i) \mid q' \in Q, i \in [k]]\!]$. If $\zeta = t[\![\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle]\!]$ then there must be $s'_j \in td_{q_j}(s_{i_j})$ for $j \in [l]$ such that $s' = t[\![s'_1, \ldots, s'_l]\!]$. Hence, by (1), $s' \in td_q(s)$ as claimed. ■

Given a class $\mathcal{C}$ of tree transductions (i.e., a class of binary relations on trees such as TD) and a class $\mathcal{L}$ of tree languages, we denote by $\mathcal{C}(\mathcal{L})$ the class of all tree languages of the form $tr(L)$ where $tr \in \mathcal{C}$ and $L \in \mathcal{L}$. This construction can be iterated: $\mathcal{C}^0(\mathcal{L}) = \mathcal{L}$ and, for $n \in \mathbb{N}$, $\mathcal{C}^{n+1}(\mathcal{L}) = \mathcal{C}(\mathcal{C}^n(\mathcal{L}))$.

Our main result (Theorem 8.1) states that $\mathrm{TD}^n(\mathrm{REGT}) = \mathrm{BST}_n$ for all $n \in \mathbb{N}$. For $n = 0$ we have already observed this fact above. The following example shows that the tree language generated by the branching tree grammar of depth 1 in the beginning of this section is of the form $td(\mathrm{T}_\Sigma)$ for a td transducer td, and is thus in TD(REGT) (since T_Σ is regular for every ranked alphabet Σ). In fact, the td transducer is even total and deterministic. We choose as input alphabet the symbols $\tau_1, \ldots, \tau_5$, where τ_2 is of rank 2, τ_1 and τ_3 are of rank 1, and τ_4 and τ_5 are of rank 0. Intuitively, the symbols τ_i help synchronize the selection of tables $R(i)$ (see also the discussion in Section 1). We choose two states q, q^{r} corresponding to the nonterminals of the grammar. Thus, q is the initial state. The rules are

$$\begin{array}{ll} q\tau_1 \to \circ[\langle q, 1\rangle], & q^{\mathrm{r}}\tau_1 \to \circ[\langle q^{\mathrm{r}}, 1\rangle], \\ q\tau_2 \to \bullet[\langle q, 1\rangle, \langle q, 2\rangle], & q^{\mathrm{r}}\tau_2 \to \bullet[\langle q^{\mathrm{r}}, 2\rangle, \langle q^{\mathrm{r}}, 1\rangle], \\ q\tau_3 \to |[\langle q, 1\rangle, \langle q^{\mathrm{r}}, 1\rangle], & q^{\mathrm{r}}\tau_3 \to |[\langle q, 1\rangle, \langle q^{\mathrm{r}}, 1\rangle], \\ q\tau_4 \to \triangleright, & q^{\mathrm{r}}\tau_4 \to \triangleleft, \\ q\tau_5 \to \triangleleft, & q^{\mathrm{r}}\tau_5 \to \triangleright. \end{array}$$

Notice that the copying of subtrees (as is done by the rules with left-hand side $q\tau_3$) results in synchronization whereas the choice of different subtrees (as in the rules with left-hand side $q\tau_2$) means to release the synchronization. Intuitively, the number i in a pair $\langle q, i\rangle$ or $\langle q^{\mathrm{r}}, i\rangle$ in the right-hand side of a rule corresponds to the synchronization symbol $i - 1$ in the respective rule of the branching tree grammar. An input tree which yields the output tree generated by the sample derivation in Figure 1 is shown below.



We shall now prove that the language-generating power of total td transducers is the same as the one of general td transducers as long as we consider a class $\mathcal{L}$ of input tree languages which is closed under finite state relabellings (i.e., $\mathrm{RELAB}(\mathcal{L}) \subseteq \mathcal{L}$). Note that closure under finite state relabellings implies closure under intersection with regular tree languages because, for every regular tree language $L \subseteq \mathrm{T}_\Sigma$, the partial identity $\{(t,t) \mid t \in L\}$ is a finite state relabelling.

Lemma 5.5 If a class $\mathcal{L}$ of tree languages is closed under finite state relabellings then $\mathrm{TD}(\mathcal{L}) = \mathrm{tTD}(\mathcal{L})$.

Proof The proof technique is similar to the one used to prove the lemma in [Eng78]. Consider a tree language $td(L)$ where $L \in \mathcal{L}$ and $td = (\Sigma, \Sigma', Q, R, q_0)$ is a td transducer. Since the domain of a td transduction is regular (see Theorem II.1 of [Rou70]) and $\mathcal{L}$ is closed under intersection with regular tree languages, we may assume that $L \subseteq \mathrm{dom}(td)$.

Intuitively, the idea behind the following construction is to modify L in such a way that (an appropriately changed version of) td can avoid running into dead ends. For this, we index the symbols in the trees in L with sets of states of td in order to indicate which states (and, hence, which rules) can process a subtree successfully.

For every $q \in Q$ let $\mathrm{dom}(q) = \mathrm{dom}(td_q)$. As mentioned above, the tree languages $\mathrm{dom}(q)$ are regular. Now, let $\overline{\Sigma}$ be the ranked alphabet where $\overline{\Sigma}_k = \{f_{Q_1 \cdots Q_k} \mid f \in \Sigma_k,\ Q_1, \ldots, Q_k \subseteq Q\}$ for all $k \in \mathbb{N}$, and let $r \colon \mathrm{T}_\Sigma \to \mathrm{T}_{\overline{\Sigma}}$ be the mapping defined as follows. For every tree $s = f[s_1, \ldots, s_k] \in \mathrm{T}_\Sigma$, $r(s) = f_{Q_1 \cdots Q_k}[r(s_1), \ldots, r(s_k)]$ where $Q_i = \{q \in Q \mid s_i \in \mathrm{dom}(q)\}$ for all $i \in [k]$. Using the fact that $\mathrm{dom}(q)$ is regular for every $q \in Q$ it is a straightforward exercise to show that r is a finite state relabelling. Thus, $r(L) \in \mathcal{L}$.

To finish the proof, we turn td into a total td transducer $td' = (\overline{\Sigma}, \Sigma', Q, R', q_0)$ such that $td'(r(L)) = td(L)$. For every symbol $f_{Q_1 \cdots Q_k} \in \overline{\Sigma}$ and every rule

$$qf \to t[\![\langle q_1, i_1 \rangle, \ldots, \langle q_l, i_l \rangle]\!]$$

in R such that $q_j \in Q_{i_j}$ for all $j \in [l]$ we let R' contain the rule

$$qf_{Q_1 \cdots Q_k} \to t[\![\langle q_1, i_1 \rangle, \ldots, \langle q_l, i_l \rangle]\!].$$

If this does not result in a total td transducer, we include arbitrary dummy rules for the missing left-hand sides. Thus, td' is total by construction. We show the following by induction on the structure of the tree s:

> For all $q \in Q$, $s \in \mathrm{dom}(q)$, and $s' \in \mathrm{T}_{\Sigma'}$, there is a computation $qs \Rightarrow_{td}^{*} s'$ if and only if there is a computation $qr(s) \Rightarrow_{td'}^{*} s'$.

The proof of this claim completes the proof of the theorem, due to the assumption that $L \subseteq \mathrm{dom}(td) = \mathrm{dom}(q_0)$.

In order to establish the claim, let $s = f[s_1, \ldots, s_k]$ and $r(s) = f_{Q_1 \cdots Q_k}[r_1, \ldots, r_k]$. Assume first that there is a computation

$$qs \Rightarrow_{td} t[\![q_1 s_{i_1}, \ldots, q_l s_{i_l}]\!] \Rightarrow_{td}^{*} t[\![s'_1, \ldots, s'_l]\!]$$

(cf. Lemma 5.4(1)). The existence of this computation proves that $s_{i_j} \in \mathrm{dom}(q_j)$ and hence, using the induction hypothesis, that $q_j r_{i_j} \Rightarrow_{td'}^{*} s'_j$ for all $j \in [l]$. Moreover, it implies that $q_j \in Q_{i_j}$ for all $j \in [l]$. Consequently, R' contains the rule $qf_{Q_1 \cdots Q_k} \to t[\![\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle]\!]$, resulting in the required computation

$$qf_{Q_1 \cdots Q_k}[r_1, \ldots, r_k] \Rightarrow_{td'} t[\![q_1 r_{i_1}, \ldots, q_l r_{i_l}]\!] \Rightarrow^*_{td'} t[\![s'_1, \ldots, s'_l]\!].$$

Conversely, assume that such a computation is given. By the argument above, the assumption $s \in \mathrm{dom}(q)$ implies that there exists a $qf_{Q_1 \cdots Q_k}$-rule in R' which is not a dummy rule. Hence, none of the $qf_{Q_1 \cdots Q_k}$-rules in R' is a dummy rule. In particular, the rule applied in the first step of the given computation has been constructed from the rule $qf \to t[\![\langle q_1, i_1\rangle, \ldots, \langle q_l, i_l\rangle]\!]$ in R, yielding a computation step $qs \Rightarrow_{td} t[\![q_1 s_{i_1}, \ldots, q_l s_{i_l}]\!]$. Since $q_j \in Q_{i_j}$ for all $j \in [l]$, the induction hypothesis yields computations $q_j s_{i_j} \Rightarrow^*_{td} s'_j$. Combining these we get

$$qs \Rightarrow_{td} t[\![q_1 s_{i_1}, \ldots, q_l s_{i_l}]\!] \Rightarrow^*_{td} t[\![s'_1, \ldots, s'_l]\!],$$

which finishes the proof. ∎

As a consequence, we find that $\mathrm{TD}^n(\mathrm{REGT}) = \mathrm{tTD}^n(\mathrm{REGT})$. This result will play an important role in the proof of our main result.

Theorem 5.6 $\mathrm{TD}^n(\mathrm{REGT}) = \mathrm{tTD}^n(\mathrm{REGT})$ for all $n \in \mathbb{N}$.

Proof By [GS84, Corollary III.6.6] we have $\mathrm{RELAB}(\mathrm{REGT}) \subseteq \mathrm{REGT}$, and by Theorem III.3.15 of the same book, TD is closed under composition with finite state relabellings. In other words, $\mathrm{TD}^n(\mathrm{REGT})$ is closed under finite state relabellings for all $n \in \mathbb{N}$. Hence the theorem follows from Lemma 5.5 by induction on n. ∎

6 From td transducers to branching tree grammars

We shall now prove the first direction of our main result, namely that $\mathrm{TD}^n(\mathrm{REGT}) \subseteq \mathrm{BST}_n$ for all $n \in \mathbb{N}$. It is convenient (and interesting in itself) to prove that these languages can even be generated by branching tree grammars that are total and terminable. The proof will be by induction on n. Recall that Lemma 4.5 (which obviously also holds for the tree case) makes it possible to turn a total terminable branching tree grammar of depth n into a total terminable branching tree grammar of depth $n+1$ which generates the same language and is deterministic in addition. Furthermore, Theorem 5.6 allows us to restrict our attention to total td transducers. Exploiting these facts, it mainly remains to be proved that every language of the form $td(L(G))$, where td is a total td transducer and G is a total deterministic terminable branching tree grammar, can be generated by a total terminable branching tree grammar G' of the same depth (see Lemma 6.3).

For the rest of this section we shall therefore consider an arbitrary but fixed branching tree grammar $G = (N, \Sigma, I, J, R, S)$ of depth n and a td transducer $td = (\Sigma, \Sigma', Q, R_{td}, q_0)$. The grammar G is assumed to be total, deterministic and terminable, and td is assumed to be total.

In order to construct G' we extend td to input symbols $(A, \varphi) \in \mathrm{SN}_G$ (of rank 0) by defining $q(A, \varphi) \Rightarrow (\langle q, A\rangle, \varphi)$ for all $q \in Q$, where $(\langle q, A\rangle, \varphi)$ is considered to be an output symbol of rank 0. Consequently, td (and thus every td_q) is now total on $\mathrm{T}_\Sigma(\mathrm{SN}_G)$. In other words, $td_q(\xi)$ is nonempty for every $q \in Q$ and $\xi \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$. Moreover, if td

is deterministic then $td_q(\xi)$ is a singleton. Note that Lemma 5.4 is valid even for the extended td transducer td.

The new grammar is $G' = (N', \Sigma', I, J, R', S')$ where $N' = Q \times N$, $S' = \langle q_0, S\rangle$, and R' is defined as follows. For $\tau \in J^n$, $R'(\tau)$ consists of all rules $\langle q, A\rangle \to \zeta'$ such that $\zeta' \in td_q(\zeta)$ where $q \in Q$ and $A \to \zeta$ is a rule in $R(\tau)$. Thus, the construction is rather standard—the right-hand side of a new rule is obtained by "running" td in the given state on the right-hand side of the original rule. Note that G' is total because both G and td are total; and if td is deterministic, then so is G'. We are going to show that $L(G') = td(L(G))$ and that G' is terminable.

We need a few auxiliary results. The first one states that td distributes over tree substitution.

Lemma 6.1 Let $\xi \in \mathrm{T}_\Sigma(\{(A_1, \varphi_1), \ldots, (A_m, \varphi_m)\})$, where $(A_1, \varphi_1), \ldots, (A_m, \varphi_m) \in \mathrm{SN}_G$ are pairwise distinct. Let $\xi_i \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$ for $i \in [m]$, and let $q \in Q$. Then

$$td_q(\xi[\![(A_i, \varphi_i) \leftarrow \xi_i \mid i \in [m]]\!]) = td_q(\xi)[\![(\langle q', A_i\rangle, \varphi_i) \leftarrow td_{q'}(\xi_i) \mid q' \in Q, i \in [m]]\!].$$

Proof Until the end of the proof, let q' and i range over Q and $[m]$, respectively. We prove the lemma by structural induction on ξ. For the induction basis assume that $\xi = (A_{i_0}, \varphi_{i_0})$ for some $i_0 \in [m]$. This yields

$$\begin{aligned} td_q(\xi[\![(A_i, \varphi_i) \leftarrow \xi_i]\!]) &= td_q(\xi_{i_0}) \\ &= (\langle q, A_{i_0}\rangle, \varphi_{i_0})[\![(\langle q', A_i\rangle, \varphi_i) \leftarrow td_{q'}(\xi_i)]\!] \\ &= td_q(\xi)[\![(\langle q', A_i\rangle, \varphi_i) \leftarrow td_{q'}(\xi_i)]\!], \end{aligned}$$

as required.

For the induction step let $\xi = f[\eta_1, \ldots, \eta_k]$ for some $f \in \Sigma_k$ and direct subtrees $\eta_1, \ldots, \eta_k$. Let $L_{qf} = \{\zeta \mid qf \to \zeta \in R_{td}\}$ and let q'' and j range over Q resp. $[k]$. Then

$$\begin{aligned} & td_q(\xi[\![(A_i, \varphi_i) \leftarrow \xi_i]\!]) \\ = \;& td_q(f[\eta_1[\![(A_i, \varphi_i) \leftarrow \xi_i]\!], \ldots, \eta_k[\![(A_i, \varphi_i) \leftarrow \xi_i]\!]]) && \text{(definition of } [\![\cdots]\!]) \\ = \;& L_{qf}[\![\langle q'', j\rangle \leftarrow td_{q''}(\eta_j[\![(A_i, \varphi_i) \leftarrow \xi_i]\!])]\!] && \text{(Lemma 5.4(2))} \\ = \;& L_{qf}[\![\langle q'', j\rangle \leftarrow td_{q''}(\eta_j)[\![(\langle q', A_i\rangle, \varphi_i) \leftarrow td_{q'}(\xi_i)]\!]]\!] && \text{(induction hypothesis)} \\ = \;& L_{qf}[\![\langle q'', j\rangle \leftarrow td_{q''}(\eta_j)]\!][\![(\langle q', A_i\rangle, \varphi_i) \leftarrow td_{q'}(\xi_i)]\!] && \text{(Lemma 5.2)} \\ = \;& td_q(\xi)[\![(\langle q', A_i\rangle, \varphi_i) \leftarrow td_{q'}(\xi_i)]\!] && \text{(Lemma 5.4(2))} \end{aligned}$$

which completes the proof. ∎

We can now prove a central lemma of this section, which relates G, G', and td by a square of the form

$$\begin{array}{ccc} \xi & \Rightarrow_G & \xi' \\ {\scriptstyle td}\downarrow & & \downarrow{\scriptstyle td} \\ \eta & \Rightarrow_{G'} & \eta' \,. \end{array}$$

We prove that, given ξ, η, and η' we can construct ξ', given ξ, ξ', and η' we can construct η, and given ξ, ξ', and η we can construct η'.

Lemma 6.2 Let $k \in \mathbb{N}$ and $\xi \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$.

(1) For all $\eta, \eta' \in \mathrm{T}_{\Sigma'}(\mathrm{SN}_{G'})$ such that $\eta \in td(\xi)$ and $\eta \Rightarrow_{G'}^k \eta'$, there exists some $\xi' \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$ such that $\xi \Rightarrow_G^k \xi'$ and $\eta' \in td(\xi')$.

(2) For all $\xi' \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$ and $\eta' \in \mathrm{T}_{\Sigma'}(\mathrm{SN}_{G'})$ such that $\eta' \in td(\xi')$ and $\xi \Rightarrow_G^k \xi'$, there exists some $\eta \in \mathrm{T}_{\Sigma'}(\mathrm{SN}_{G'})$ such that $\eta \Rightarrow_{G'}^k \eta'$ and $\eta \in td(\xi)$.

(3) For all $\xi' \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$ and $\eta \in \mathrm{T}_{\Sigma'}(\mathrm{SN}_{G'})$ such that $\eta \in td(\xi)$ and $\xi \Rightarrow_G^k \xi'$, there exists some $\eta' \in \mathrm{T}_{\Sigma'}(\mathrm{SN}_{G'})$ such that $\eta \Rightarrow_{G'}^k \eta'$ and $\eta' \in td(\xi')$.

Proof The statements are trivial for $k = 0$. Therefore, it suffices to prove the lemma for single derivation steps, i.e., for $\Rightarrow_G$ and $\Rightarrow_{G'}$ instead of $\Rightarrow_G^k$ and $\Rightarrow_{G'}^k$. Then the lemma follows by induction on k.

Since G is deterministic and total, for every synchronized nonterminal (A, φ) and every nonempty table $\tau \in J^n$ there is exactly one rule $r \in R(\tau)$ which applies to (A, φ) (i.e., whose left-hand side is A). Thus, for every $\tau \in J^n$ and $(A, \varphi) \in \mathrm{SN}_G$ there is a uniquely determined tree $\zeta \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$ such that $(A, \varphi) \Rightarrow_r \zeta$ for $r \in R(\tau)$. In the following, let us denote this tree by $\tau(A, \varphi)$. We first prove the claim below.

Claim Let $\eta \in \mathrm{T}_{\Sigma'}(\{(\langle q, A_i\rangle, \varphi_i) \mid q \in Q, i \in [m]\})$ where $(A_1, \varphi_1), \ldots, (A_m, \varphi_m) \in \mathrm{SN}_G$ are pairwise distinct and $\varphi_1, \ldots, \varphi_m$ have equal length, and let $\eta' \in \mathrm{T}_{\Sigma'}(\mathrm{SN}_{G'})$. There is a derivation step $\eta \Rightarrow_{G'} \eta'$ if and only if

$$\eta' \in \eta[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow td_q(\tau_i(A_i, \varphi_i)) \mid q \in Q, i \in [m]]\!]$$

for some tables $\tau_1, \ldots, \tau_m \in J^n$ which are consistent with $\varphi_1, \ldots, \varphi_m$.

To prove the claim, let $\eta = t_\eta[\![(\langle q_1, A_{i_1}\rangle, \varphi_{i_1}), \ldots, (\langle q_h, A_{i_h}\rangle, \varphi_{i_h})]\!]$ where $q_1, \ldots, q_h \in Q$ and $i_1, \ldots, i_h \in [m]$. Note that the q_j resp. i_j need not be distinct. We consider the two directions separately.

($\Rightarrow$) By the definition of derivation steps we have $\eta \Rightarrow_{G'} \eta'$ only if there are tables $\tau'_1, \ldots, \tau'_h \in J^n$ which are consistent with $\varphi_{i_1}, \ldots, \varphi_{i_h}$ such that $\eta' = t_\eta[\![\zeta_1, \ldots, \zeta_h]\!]$ where $(\langle q_j, A_{i_j}\rangle, \varphi_{i_j}) \Rightarrow_{r_j} \zeta_j$ for some $r_j \in R'(\tau'_j)$ ($j \in [h]$). By the construction of R', the latter means that $\zeta_j \in td_{q_j}(\tau'_j(A_{i_j}, \varphi_{i_j}))$ for all $j \in [h]$. This yields

$$\eta' \in \eta[\![(\langle q_j, A_{i_j}\rangle, \varphi_{i_j}) \leftarrow td_{q_j}(\tau'_j(A_{i_j}, \varphi_{i_j})) \mid j \in [h]]\!].$$

Since $\tau'_1, \ldots, \tau'_h$ are consistent with $\varphi_{i_1}, \ldots, \varphi_{i_h}$ it holds in particular that $\tau'_r = \tau'_s$ for all $r, s \in [h]$ such that $i_r = i_s$. It is thus sound to define $\tau_{i_j} = \tau'_j$ for all $j \in [h]$. By Lemma 4.2 this choice of tables can be extended in order to cover even those $i \in [m]$ which do not occur among $i_1, \ldots, i_h$, yielding tables $\tau_1, \ldots, \tau_m$ which are consistent with $\varphi_1, \ldots, \varphi_m$. Consequently, we get

$$\begin{aligned}\eta' &\in \eta[\![(\langle q_j, A_{i_j}\rangle, \varphi_{i_j}) \leftarrow td_{q_j}(\tau'_j(A_{i_j}, \varphi_{i_j})) \mid j \in [h]]\!] \\ &= \eta[\![(\langle q_j, A_{i_j}\rangle, \varphi_{i_j}) \leftarrow td_{q_j}(\tau_{i_j}(A_{i_j}, \varphi_{i_j})) \mid j \in [h]]\!] \\ &= \eta[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow td_q(\tau_i(A_i, \varphi_i)) \mid q \in Q, i \in [m]]\!]\end{aligned}$$

as claimed.

($\Leftarrow$) This direction is quite obvious. We have

$$\begin{aligned}
\eta' &\in \eta[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow td_q(\tau_i(A_i, \varphi_i)) \mid q \in Q, i \in [m]]\!] \\
&= \eta[\![(\langle q_j, A_{i_j}\rangle, \varphi_{i_j}) \leftarrow td_{q_j}(\tau_{i_j}(A_{i_j}, \varphi_{i_j})) \mid j \in [h]]\!] \\
&= \{t_\eta[\![\zeta_1, \ldots, \zeta_h]\!] \mid \zeta_j \in td_{q_j}(\tau_{i_j}(A_{i_j}, \varphi_{i_j})) \text{ for all } j \in [h]\}
\end{aligned}$$

where $\tau_{i_1}, \ldots, \tau_{i_h}$ are consistent with $\varphi_{i_1}, \ldots, \varphi_{i_h}$ (which is obvious as $\tau_1, \ldots, \tau_m$ are consistent with $\varphi_1, \ldots, \varphi_m$). This proves that there is a derivation step $\eta \Rightarrow_{G'} \eta'$ and completes the proof of the claim.

Let us now turn to the proof of the three parts of the lemma. Let $\xi \in \mathrm{T}_\Sigma(\{\langle A_1, \varphi_1\rangle, \ldots, \langle A_m, \varphi_m\rangle\})$ where $\langle A_1, \varphi_1\rangle, \ldots, \langle A_m, \varphi_m\rangle \in \mathrm{SN}_G$ and $\varphi_1, \ldots, \varphi_m$ have equal length.

(1) By the claim above and Lemma 6.1, $\eta \Rightarrow_{G'} \eta'$ implies

$$\begin{aligned}
\eta' &\in \eta[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow td_q(\tau_i(A_i, \varphi_i)) \mid q \in Q, i \in [m]]\!] \\
&\subseteq td(\xi)[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow td_q(\tau_i(A_i, \varphi_i)) \mid q \in Q, i \in [m]]\!] \\
&= td(\xi[\![(A_i, \varphi_i) \leftarrow \tau_i(A_i, \varphi_i) \mid i \in [m]]\!])
\end{aligned}$$

for tables $\tau_1, \ldots, \tau_m$ which are consistent with $\varphi_1, \ldots, \varphi_m$. This proves the first part of the lemma, taking $\xi' = \xi[\![(A_i, \varphi_i) \leftarrow \tau_i(A_i, \varphi_i) \mid i \in [m]]\!]$, since $\xi \Rightarrow_G \xi[\![(A_i, \varphi_i) \leftarrow \tau_i(A_i, \varphi_i) \mid i \in [m]]\!]$.

(2) If $\xi \Rightarrow_G \xi'$ then $\xi' = \xi[\![(A_i, \varphi_i) \leftarrow \tau_i(A_i, \varphi_i) \mid i \in [m]]\!]$ for tables $\tau_1, \ldots, \tau_m$ which are consistent with $\varphi_1, \ldots, \varphi_m$. Thus, $\eta' \in td(\xi')$ means

$$\eta' \in td(\xi)[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow td_q(\tau_i(A_i, \varphi_i)) \mid q \in Q, i \in [m]]\!]$$

by Lemma 6.1. In other words, there exists some tree $\eta \in td(\xi)$ such that $\eta' \in \eta[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow td_q(\tau_i(A_i, \varphi_i)) \mid q \in Q, i \in [m]]\!]$. Now, the claim proved in the beginning yields $\eta \Rightarrow_{G'} \eta'$, as claimed.

(3) Again, $\xi \Rightarrow_G \xi'$ means $\xi' = \xi[\![(A_i, \varphi_i) \leftarrow \tau_i(A_i, \varphi_i) \mid i \in [m]]\!]$ for tables $\tau_1, \ldots, \tau_m$ which are consistent with $\varphi_1, \ldots, \varphi_m$. For all $q \in Q$ and $i \in [m]$, choose some tree $\zeta_{q,i} \in td_q(\tau_i(A_i, \varphi_i))$ (which is possible due to the totality of td) and define $\eta' = \eta[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow \zeta_{q,i} \mid q \in Q, i \in [m]]\!]$. By our claim, $\eta \Rightarrow_{G'} \eta'$. Furthermore, since $\eta \in td(\xi)$, Lemma 6.1 yields

$$\begin{aligned}
\eta' &= \eta[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow \zeta_{q,i} \mid q \in Q, i \in [m]]\!] \\
&\in td(\xi)[\![(\langle q, A_i\rangle, \varphi_i) \leftarrow td_q(\tau_i(A_i, \varphi_i)) \mid q \in Q, i \in [m]]\!] \\
&= td(\xi'),
\end{aligned}$$

which completes the proof of (3) and hence the proof of the lemma. ■

We are now ready to show that $L(G') = td(L(G))$. To see that $L(G') \subseteq td(L(G))$ consider some tree $t \in L(G')$. Then $(\langle q_0, S\rangle, \lambda) \Rightarrow^+_{G'} t$, and since $td((S, \lambda)) = \{(\langle q_0, S\rangle, \lambda)\}$, Lemma 6.2(1) states that there exists some $\xi \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$ such that $(S, \lambda) \Rightarrow^+_G \xi$ and $t \in td(\xi)$. We cannot yet conclude that $t \in td(L(G))$ because ξ may contain nonterminals (which td deletes). However, as G is terminable, there is a tree $s \in L(G)$ such that

$\xi \Rightarrow_G^* s$. Now, Lemma 6.2(3) yields a tree $t' \in td(s) \subseteq td(L(G))$ such that $t \Rightarrow_{G'}^* t'$. But since t is terminal the latter implies $t = t'$, which proves that $t \in td(L(G))$, as required.

Conversely, let $t \in td(L(G))$. Consider some tree $s \in \mathrm{T}_\Sigma$ such that $(S, \lambda) \Rightarrow_G^* s$ (i.e., $s \in L(G)$) and $t \in td(s)$. By Lemma 6.2(2) this implies the existence of a tree $\eta \in td((S, \lambda)) = \{(\langle q_0, S\rangle, \lambda)\}$ such that $\eta \Rightarrow_{G'}^* t$. In other words, $t \in L(G')$, as claimed.

Finally, let us check that G' is terminable. Consider a derivation $(\langle q_0, S\rangle, \lambda) \Rightarrow_{G'}^+ \eta$. By Lemma 6.2(1) there exists a tree $\xi \in \mathrm{T}_\Sigma(\mathrm{SN}_G)$ such that $(S, \lambda) \Rightarrow_G^+ \xi$ and $\eta \in td(\xi)$. Since G is terminable, $\xi \Rightarrow_G^* s$ for some tree $s \in \mathrm{T}_\Sigma$. Now Lemma 6.2(3) yields a tree $t \in \mathrm{T}_{\Sigma'}(\mathrm{SN}_{G'})$ such that $\eta \Rightarrow_{G'}^* t$ and $t \in td(s)$. In fact, $t \in \mathrm{T}_{\Sigma'}$ as $s \in \mathrm{T}_\Sigma$, and hence G' is terminable.

The following lemma summarizes what we have proved so far in this section.

Lemma 6.3 Let G be a total deterministic terminable branching tree grammar, and let td be a total td transducer. Then a total terminable branching tree grammar G' of the same nesting depth can be constructed such that $L(G') = td(L(G))$. If td is deterministic, then so is G'.

Together with Theorem 5.6 and Lemma 4.5, this yields the main result of this section.

Theorem 6.4 Let $n \in \mathbb{N}$. For every tree language $L \in \mathrm{TD}^n(\mathrm{REGT})$ there is a total terminable branching tree grammar G of nesting depth n such that $L(G) = L$.

Proof We proceed by induction on n. For $n = 0$ it has been observed earlier that $\mathrm{BST}_0 = \mathrm{REGT} = \mathrm{TD}^0(\mathrm{REGT})$. Furthermore, it is well known that every regular tree grammar can be turned into one that is total and terminable, without affecting the generated language. (Given a regular tree grammar (N, Σ, R, S), call a nonterminal $A \in N$ *useful* if $A = S$ or $S \Rightarrow^* \xi[\![A]\!] \Rightarrow^* t$ for some trees $\xi[\![A]\!] \in \mathrm{T}_\Sigma(N)$ and $t \in \mathrm{T}_\Sigma$. Now, remove all nonterminals that are not useful, together with the rules in which they occur. Obviously, the resulting grammar generates the same language while being terminable and total.)

Now, consider some $n > 0$. By Theorem 5.6 we have $L \in \mathrm{tTD}^n(\mathrm{REGT})$. Let therefore $L_0 \in \mathrm{tTD}^{n-1}(\mathrm{REGT})$ and $td \in \mathrm{tTD}$ be such that $L = td(L_0)$. By the induction hypothesis, $L_0 = L(G_0)$ for some total terminable branching tree grammar G_0 of depth $n-1$. Using Lemma 4.5 this yields a total, terminable, and deterministic branching tree grammar G_0' of depth n such that $L(G_0') = L_0$. Hence, by Lemma 6.3 there is a total terminable branching tree grammar G of depth n such that $L(G) = td(L_0)$, as claimed. ∎

7 From branching tree grammars to td transducers

The aim of this section is to prove the converse of Theorem 6.4: $\mathrm{BST}_n \subseteq \mathrm{TD}^n(\mathrm{REGT})$ for all $n \in \mathbb{N}$. Since $\mathrm{BST}_0 = \mathrm{REGT}$ by definition, this holds for $n = 0$. Hence, it suffices to study the case $n \geq 1$. For the rest of this section, let us therefore consider an arbitrary but fixed branching tree grammar $G = (N, \Sigma, I, J, R, S)$ of depth $n \geq 1$. Without loss of generality, assume that $I = \{0, \ldots, d-1\}$ for some $d \in \mathbb{N}$.

The translation of G to a composition of n td transducers (applied to a regular tree language) makes use of the notion of *synchronization trees* to be defined below. The set of synchronization trees depends only on I, J, and n. Intuitively, the synchronization trees represent all correctly synchronized choices of tables in the derivations of G. We will need $n-1$ td transducers (applied to a regular tree language) in order to generate the set of synchronization trees for the given I, J, and n. Then, another td transducer td_G, whose states are the nonterminals of G and whose rules are obtained from those of G in a rather direct manner, exploits the information in the synchronization trees (which it takes as input), in order to generate the language $L(G)$.

For the formal definition of synchronization trees, we need a few prerequisites. By 'num' we denote the bijection $\mathrm{num}\colon I^n \to [d^n]$ given by $\mathrm{num}(i_1,\dots,i_n) = 1 + \sum_{j=1}^{n} i_j \cdot d^{n-j}$ for all $i_1,\dots,i_n \in I$. Thus, 'num' simply interprets $(i_1,\dots,i_n)$ as a natural number written in base-d notation (and adds 1 in order to get a number in $[d^n]$ rather than in $\{0,\dots,d^n-1\}$).

Let $\Sigma_{\langle n\rangle}$ denote the ranked alphabet consisting of a symbol $\perp$ of rank 0 and all $\tau \in J^n$ viewed as symbols of rank d^n. Thus, the internal symbols of a tree $s \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$ are tables of nesting depth n, and every such symbol has as many children as there are possibilities to branch out in the synchronization. Among others, this can be exploited in order to address the nodes of s by synchronization strings $\varphi \in (I^n)^*$. For this, we define the set $\mathrm{internal}(s) \subseteq (I^n)^*$ of *internal nodes* of s, the table at such a node, and the subtree rooted at the node inductively, as follows. If $s = \perp$ then $\mathrm{internal}(s) = \emptyset$. If $s = \tau[s_1,\dots,s_{d^n}]$ for some $\tau \in J^n$ and $s_1,\dots,s_{d^n} \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$, then

- $\mathrm{internal}(s) = \{\lambda\} \cup \{\alpha\varphi \mid \alpha \in I^n, \varphi \in \mathrm{internal}(s_{\mathrm{num}(\alpha)})\}$,
- $s(\lambda) = \tau$ and $s/\lambda = s$, and
- $s(\alpha\varphi) = s_{\mathrm{num}(\alpha)}(\varphi)$ and $s/\alpha\varphi = s_{\mathrm{num}(\alpha)}/\varphi$ for all $\alpha \in I^n$ and $\varphi \in \mathrm{internal}(s_{\mathrm{num}(\alpha)})$.

Note that, for $\varphi \in \mathrm{internal}(s)$, if $\varphi = \varphi_1\varphi_2$ and $s_1 = s/\varphi_1$ for some $\varphi_1,\varphi_2 \in (I^n)^*$, then $s(\varphi) = s_1(\varphi_2)$ and $s/\varphi = s_1/\varphi_2$. This can be shown in a straightforward way using induction on the length of φ_1. The reader should furthermore notice that s is uniquely determined by specifying a finite, prefix-closed set $\mathrm{internal}(s)$ of synchronization strings and, for every $\varphi \in \mathrm{internal}(s)$, the table $s(\varphi)$. This is because, intuitively, every $\varphi = \varphi'\alpha \notin \mathrm{internal}(s)$ with $\varphi' \in \mathrm{internal}(s)$ and $\alpha \in I^n$, corresponds to a leaf of s labelled $\perp$.

We can now define synchronization trees.

Definition 7.1 (synchronization tree) The set of *n-synchronization trees* (synchronization trees, for short) is the set SYNC_n of all trees $s \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$ such that

$$\mathrm{level}(s(\varphi), s(\varphi')) \geq \mathrm{level}(\varphi, \varphi')$$

for all $\varphi, \varphi' \in \mathrm{internal}(s)$ of equal length.

Thus, the requirement is that the tables at nodes φ, φ' are consistent with φ, φ'. We now show that SYNC_n can be generated by a composition of $n-1$ td transducers applied to a regular tree language.

Theorem 7.2 $\mathrm{SYNC}_n \in \mathrm{TD}^{n-1}(\mathrm{REGT})$.

Proof We prove the theorem by induction on n. For $n = 1$ it suffices to notice that $\mathrm{SYNC}_1 = \mathrm{T}_{\Sigma_{\langle n\rangle}}$. Indeed, if $n = 1$ in Definition 7.1 then $\mathrm{level}(\varphi,\varphi') = 1$ means $\varphi = \varphi'$ and thus $\mathrm{level}(s(\varphi), s(\varphi')) = 1$, which shows that all trees in $\mathrm{T}_{\Sigma_{\langle 1\rangle}}$ are 1-synchronization trees.

Now, let $n \geq 1$ and assume that $\mathrm{SYNC}_n \in \mathrm{TD}^{n-1}(\mathrm{REGT})$. We construct a td transducer whose application to SYNC_n yields SYNC_{n+1}. Let $td = (\Sigma_{\langle n\rangle}, \Sigma_{\langle n+1\rangle}, \{q\}, R, q)$ where R consists of all rules

$$q\,\tau \to \tau.j[\underbrace{\langle q,1\rangle,\ldots,\langle q,1\rangle}_{d \text{ times}},\ldots,\underbrace{\langle q,d^n\rangle,\ldots,\langle q,d^n\rangle}_{d \text{ times}}]$$

and $q\,\tau \to \bot$, where $\tau \in J^n$ and $j \in J$, as well as the rule $q\,\bot \to \bot$.

We have to prove that $td(\mathrm{SYNC}_n) = \mathrm{SYNC}_{n+1}$. For this, we first note the following claim, which is a direct consequence of Lemma 5.4 and the definition of R.

Claim 1 Let $s \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$ and $s' \in \mathrm{T}_{\Sigma_{\langle n+1\rangle}}$. Then $s' \in td(s)$ if and only if $s' = \bot$ or $s(\lambda) = \mathrm{first}_n(s'(\lambda))$ and $s'/\alpha \in td(s/\mathrm{first}_n(\alpha))$ for all $\alpha \in I^{n+1}$.

Using Claim 1, we show the following.

Claim 2 Let $s \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$ and $s' \in \mathrm{T}_{\Sigma_{\langle n+1\rangle}}$. Then $s' \in td(s)$ if and only if, for all $\varphi \in \mathrm{internal}(s')$, $\mathrm{first}_n(\varphi) \in \mathrm{internal}(s)$ and $s(\mathrm{first}_n(\varphi)) = \mathrm{first}_n(s'(\varphi))$.

We proceed by structural induction on s'. The claim holds for $s' = \bot$ as in this case $\mathrm{internal}(s') = \emptyset$ and $s' \in td(s)$ for all $s \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$. Assume now that $s' \neq \bot$; we consider both directions of the claimed equivalence separately.

($\Rightarrow$) Let $\varphi \in \mathrm{internal}(s')$. If $\varphi = \lambda$ then $s(\mathrm{first}_n(\varphi)) = s(\lambda) = \mathrm{first}_n(s'(\varphi))$ by Claim 1. If $\varphi = \alpha\varphi'$ for some $\alpha \in I^{n+1}$ and $\varphi' \in (I^{n+1})^*$, then $s'/\alpha \in td(s/\mathrm{first}_n(\alpha))$ by Claim 1. Hence, using the induction hypothesis and the fact that $\varphi' \in \mathrm{internal}(s'/\alpha)$, we find that $\mathrm{first}_n(\varphi') \in \mathrm{internal}(s/\mathrm{first}_n(\alpha))$ and $s/\mathrm{first}_n(\alpha)(\mathrm{first}_n(\varphi')) = \mathrm{first}_n(s'/\alpha(\varphi'))$. Hence, since $\mathrm{first}_n(\varphi) = \mathrm{first}_n(\alpha)\mathrm{first}_n(\varphi')$, it follows that $\mathrm{first}_n(\varphi) \in \mathrm{internal}(s)$ and $s(\mathrm{first}_n(\varphi)) = \mathrm{first}_n(s'(\varphi))$, as claimed.

($\Leftarrow$) As we have $s(\lambda) = \mathrm{first}_n(s'(\lambda))$, by Claim 1 it suffices to show that $s'/\alpha \in td(s/\mathrm{first}_n(\alpha))$ for all $\alpha \in I^{n+1}$. Consider some $\alpha \in I^{n+1}$ and let $\varphi' \in \mathrm{internal}(s'/\alpha)$ and $\varphi = \alpha\varphi'$. Then $\varphi \in \mathrm{internal}(s')$ and we have that $\mathrm{first}_n(\varphi) \in \mathrm{internal}(s)$, $\mathrm{first}_n(\varphi') \in \mathrm{internal}(s/\mathrm{first}_n(\alpha))$, and that $s/\mathrm{first}_n(\alpha)(\mathrm{first}_n(\varphi')) = s(\mathrm{first}_n(\varphi)) = \mathrm{first}_n(s'(\varphi)) = \mathrm{first}_n(s'/\alpha(\varphi'))$. This shows that the induction hypothesis applies to s'/α and $s/\mathrm{first}_n(\alpha)$, which yields $s'/\alpha \in td(s/\mathrm{first}_n(\alpha))$, as required. This completes the proof of Claim 2.

We can now prove that $td(\mathrm{SYNC}_n) = \mathrm{SYNC}_{n+1}$. First, consider some $s \in \mathrm{SYNC}_n$ and let $s' \in td(s)$. For $\varphi, \varphi' \in \mathrm{internal}(s')$ of equal length, if $\mathrm{level}(\varphi,\varphi') = n+1$ then $\varphi = \varphi'$ and hence $\mathrm{level}(s'(\varphi), s'(\varphi')) = n+1$. Otherwise, $\mathrm{level}(\varphi,\varphi') = \mathrm{level}(\mathrm{first}_n(\varphi), \mathrm{first}_n(\varphi'))$ and Claim 2 yields

$$\begin{aligned}
\mathrm{level}(s'(\varphi), s'(\varphi')) &\geq \mathrm{level}(\mathrm{first}_n(s'(\varphi)), \mathrm{first}_n(s'(\varphi')))\\
&= \mathrm{level}(s(\mathrm{first}_n(\varphi)), s(\mathrm{first}_n(\varphi')))\\
&\geq \mathrm{level}(\mathrm{first}_n(\varphi), \mathrm{first}_n(\varphi'))\\
&= \mathrm{level}(\varphi,\varphi').
\end{aligned}$$

Thus, $s' \in \mathrm{SYNC}_{n+1}$. Conversely, let $s' \in \mathrm{SYNC}_{n+1}$ and define a tree $s \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$ as follows: $\mathrm{internal}(s)$ is the (finite, prefix-closed) set $\{\mathrm{first}_n(\varphi) \mid \varphi \in \mathrm{internal}(s')\}$ and $s(\mathrm{first}_n(\varphi)) =$

$\mathrm{first}_n(s'(\varphi))$ for all $\varphi \in \mathrm{internal}(s')$. This definition is consistent as $\mathrm{first}_n(\varphi) = \mathrm{first}_n(\varphi')$ for $\varphi, \varphi' \in \mathrm{internal}(s')$ implies $\mathrm{first}_n(s'(\varphi)) = \mathrm{first}_n(s'(\varphi'))$ since $s' \in \mathrm{SYNC}_{n+1}$. By Claim 2 we have $s' \in td(s)$, so it remains to be shown that $s \in \mathrm{SYNC}_n$. By construction, $\psi, \psi' \in \mathrm{internal}(s)$ only if there are $\varphi, \varphi' \in \mathrm{internal}(s')$ with $\psi = \mathrm{first}_n(\varphi)$ and $\psi' = \mathrm{first}_n(\varphi')$. This yields $s \in \mathrm{SYNC}_n$ since

$$\begin{aligned}
\mathrm{level}(s(\psi), s(\psi')) &= \mathrm{level}(\mathrm{first}_n(s'(\varphi)), \mathrm{first}_n(s'(\varphi'))) \\
&= \min(n, \mathrm{level}(s'(\varphi), s'(\varphi'))) \\
&\geq \min(n, \mathrm{level}(\varphi, \varphi')) \\
&= \mathrm{level}(\mathrm{first}_n(\varphi), \mathrm{first}_n(\varphi')) \\
&= \mathrm{level}(\psi, \psi'),
\end{aligned}$$

which completes the proof of the theorem. ■

We shall now define the td transducer td_G which completes the implementation of the branching tree grammar $G = (N, \Sigma, I, J, R, S)$ by a series of n td transducers. Let $td_G = (\Sigma_{\langle n \rangle}, \Sigma, N, R', S)$ where R' consists of all rules

$$A\,\tau \to t[\![\langle B_1, \mathrm{num}(\alpha_1)\rangle, \ldots, \langle B_l, \mathrm{num}(\alpha_l)\rangle]\!]$$

such that $\tau \in J^n$ and

$$A \to t[\![(B_1, \alpha_1), \ldots, (B_l, \alpha_l)]\!]$$

is a rule in $R(\tau)$. Note that there are no rules in R' with left-hand side $A\,\bot$. Note also that if G is deterministic, then so is td_G.

In order to show that $td_G(\mathrm{SYNC}_n) = L(G)$, we must establish a correspondence between derivations in G and synchronization trees. Let $(A, \varphi) \in \mathrm{SN}_G$ and $s \in \mathrm{SYNC}_n$. A derivation in G starting with (A, φ) is said to be *s-synchronized* if, for every step $t[\![(A_1, \varphi\varphi_1), \ldots, (A_h, \varphi\varphi_h)]\!] \Rightarrow t[\![\zeta_1, \ldots, \zeta_h]\!]$ of this derivation using tables $\tau_1, \ldots, \tau_h$, it holds for all $i \in [h]$ that $\varphi_i \in \mathrm{internal}(s)$ and $\tau_i = s(\varphi_i)$.

Lemma 7.3 For all $s \in \mathrm{SYNC}_n$ and $s' \in td_G(s)$ there is an s-synchronized derivation $(S, \lambda) \Rightarrow_G^* s'$.

Proof We prove the following, more general statement, by induction on s.

Claim Let $A \in N$, $s \in \mathrm{SYNC}_n$, and $s' \in \mathrm{T}_\Sigma$. If $A[s] \Rightarrow_{td_G}^* s'$, then there is an s-synchronized derivation $(A, \varphi) \Rightarrow_G^* s'$ for every $\varphi \in (I^n)^*$.

To prove the claim, let $\varphi \in (I^n)^*$ and assume that there is a computation

$$A[s] \Rightarrow_{td_G} t[\![B_1[s_1], \ldots, B_l[s_l]]\!] \Rightarrow_{td_G}^* t[\![s'_1, \ldots, s'_l]\!] \qquad \text{(cf. Lemma 5.4).}$$

By the construction of td_G this implies that $\lambda \in \mathrm{internal}(s)$ and $R(s(\lambda))$ contains a rule

$$A \to t[\![(B_1, \alpha_1), \ldots, (B_l, \alpha_l)]\!]$$

where $s_i = s/\alpha_i$ for all $i \in [l]$. Hence, by the induction hypothesis, there is an s/α_i-synchronized derivation $(B_i, \varphi\alpha_i) \Rightarrow_G^* s'_i$ for every $i \in [l]$. We claim that these derivations can be combined (in the obvious way) to an s-synchronized derivation

$$(A, \varphi) \Rightarrow_G t[\![(B_1, \varphi\alpha_1), \ldots, (B_l, \varphi\alpha_l)]\!] \Rightarrow_G^* t[\![s'_1, \ldots, s'_l]\!].$$

To see that this is a correct derivation, let $(X, \varphi\alpha_i\psi)$ and $(Y, \varphi\alpha_j\psi')$ be nonterminals in one of the intermediate steps, where $i, j \in [l]$. Then the tables applied to them are $s/\alpha_i(\psi) = s(\alpha_i\psi)$ resp. $s/\alpha_j(\psi') = s(\alpha_j\psi')$ as $(B_i, \varphi\alpha_i) \Rightarrow_G^* s'_i$ is s/α_i-synchronized and $(B_j, \varphi\alpha_j) \Rightarrow_G^* s'_j$ is s/α_j-synchronized. Since $\mathrm{level}(s(\alpha_i\psi), s(\alpha_j\psi')) \geq \mathrm{level}(\alpha_i\psi, \alpha_j\psi') = \mathrm{level}(\varphi\alpha_i\psi, \varphi\alpha_j\psi')$ this shows that the choice of tables is consistent with the synchronization strings of nonterminals in every step, i.e., every step satisfies Definition 3.3(ii). In passing, we have also shown that the table applied to a nonterminal $(X, \varphi\alpha_i\psi)$ $(i \in [l])$ in some step of the derivation is $s(\alpha_i\psi)$. In addition, the table applied to (A, φ) in the first step was $s(\lambda)$, which proves that the derivation is s-synchronized. ■

Lemma 7.4 For all $s \in \mathrm{SYNC}_n$ and $s' \in \mathrm{T}_\Sigma$, if there is an s-synchronized derivation $(S, \lambda) \Rightarrow_G^* s'$, then $s' \in td_G(s)$.

Proof Again, we prove a slightly more general claim using induction on s.

Claim Let $A \in N$, $s \in \mathrm{SYNC}_n$, and $s' \in \mathrm{T}_\Sigma$. If there is an s-synchronized derivation $(A, \varphi) \Rightarrow_G^* s'$ for some $\varphi \in (I^n)^*$ then $A[s] \Rightarrow_{td_G}^* s'$.

For the proof, consider an s-synchronized derivation

$$(A, \varphi) \Rightarrow_G t[\![(B_1, \varphi\alpha_1), \ldots, (B_l, \varphi\alpha_l)]\!] \Rightarrow_G^* t[\![s'_1, \ldots, s'_l]\!].$$

Since $s/\alpha_i(\psi) = s(\alpha_i\psi)$ for all $i \in [l]$ and $\alpha_i\psi \in \mathrm{internal}(s)$, each of the subderivations $(B_i, \varphi\alpha_i) \Rightarrow_G^* s'_i$ (where $i \in [l]$) is s/α_i-synchronized. By the induction hypothesis this implies that $B_i[s/\alpha_i] \Rightarrow_{td_G}^* s'_i$ for all $i \in [l]$. Moreover, since the rule $A \to t[\![(B_1, \alpha_1), \ldots, (B_l, \alpha_l)]\!]$ is in $R(s(\lambda))$, we get

$$A[s] \Rightarrow_{td_G} t[\![B_1[s/\alpha_1], \ldots, B_l[s/\alpha_l]]\!] \Rightarrow_{td_G}^* t[\![s'_1, \ldots, s'_l]\!],$$

as claimed. ■

Lemma 7.3 already yields the inclusion $td_G(\mathrm{SYNC}_n) \subseteq L(G)$. Lemma 7.4 provides us with the converse inclusion if we can show that every derivation in G is s-synchronized for some $s \in \mathrm{SYNC}_n$. We prove this now.

Lemma 7.5 Every derivation $(S, \lambda) \Rightarrow_G^* s'$ is s-synchronized for some $s \in \mathrm{SYNC}_n$.

Proof Consider a derivation $D = (\xi_1 \Rightarrow_G \xi_2 \Rightarrow_G \cdots \Rightarrow_G \xi_m)$ where $\xi_1 = (S, \lambda)$. We construct an n-synchronization tree s such that D is s-synchronized, as follows. Let $\mathrm{internal}(s)$ be the (finite, prefix-closed) set of all $\varphi \in (I^n)^*$ such that (A, φ) occurs in ξ_i for some $A \in N$ and $i \in [m-1]$. Furthermore, if τ is the table applied to this occurrence of (A, φ) in the derivation step $\xi_i \Rightarrow_G \xi_{i+1}$ then $s(\varphi) = \tau$. (More formally, if the i-th step in D is $\xi_i = t[\![(A_1, \varphi_1), \ldots, (A_h, \varphi_h)]\!] \Rightarrow_G t[\![\zeta_1, \ldots, \zeta_h]\!] = \xi_{i+1}$ where $(A_j, \varphi_j) \Rightarrow_{r_j} \zeta_j$ for every $j \in [h]$, using tables $\tau_1, \ldots, \tau_h$, then $s(\varphi_j) = \tau_j$ for every $j \in [h]$.)

To see that the definition of s is consistent note that, if there are two occurrences of nonterminals $(A, \varphi), (B, \varphi)$ with the same synchronization string in ξ_i resp. ξ_j then $i = j$ (since the length of synchronization strings in ξ_i is $i - 1$ for all $i \in [m]$). Hence, by Definition 3.3(ii), the same tables are applied to both occurrences.

Moreover, s is an n-synchronization tree: If $\varphi, \psi \in \mathrm{internal}(s)$ have the same length, say i, then tables $s(\varphi)$ and $s(\psi)$ were applied to some nonterminals (A, φ) and (B, ψ) in ξ_{i+1}. By Definition 3.3(ii) this implies $\mathrm{level}(s(\varphi), s(\psi)) \geq \mathrm{level}(\varphi, \psi)$.

This completes the proof because D is s-synchronized by construction. ■

Lemmas 7.4 and 7.5 yield the inclusion $L(G) \subseteq td_G(\mathrm{SYNC}_n)$. Thus, we have shown the following result.

Theorem 7.6 For every branching tree grammar $G = (N, \Sigma, I, J, R, S)$ of depth n there is a top-down tree transducer td_G such that $L(G) = td_G(\mathrm{SYNC}_n)$. If G is deterministic, then so is td_G. (Note that SYNC_n depends on I and J).

It follows from Theorems 7.2 and 7.6 that $\mathrm{BST}_n \subseteq \mathrm{TD}^n(\mathrm{REGT})$.

8 Main results

Combining Theorem 6.4 with Theorems 7.2 and 7.6, we get the main result of this paper.

Theorem 8.1 For every $n \in \mathbb{N}$, $\mathrm{BST}_n = \mathrm{TD}^n(\mathrm{REGT})$.

Using this, a characterization of the string languages in BS_n follows immediately from Lemma 5.1.

Corollary 8.2 For every $n \in \mathbb{N}$, $\mathrm{BS}_n = \mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$.

In particular, the class BS_1 of languages generated by branching ET0L systems (see Definition 3.1) is equal to yield(TD(REGT)).

In Theorem 6.4 we have shown that the constructed branching tree grammar is total and terminable. This leads to the following normal form result for branching grammars.

Theorem 8.3 For every branching grammar G there is a total terminable branching grammar G' of the same nesting depth such that $L(G') = L(G)$. If G is a branching tree grammar, then so is G'. If G is deterministic, then so is G'.

Proof For tree grammars this follows directly from Theorems 8.1 and 6.4. Hence, since the constructions in the proof of Lemma 5.1 obviously preserve totality and terminability, it also holds for ordinary branching grammars. Preservation of determinism will be shown in the proof of the next theorem. ■

Finally, we consider a characterization of the languages generated by deterministic branching grammars. We will use dBS_n to denote the class of languages generated by deterministic branching grammars of depth n, and similarly dBST_n for the corresponding tree languages.

Theorem 8.4 For every $n \geq 1$, $\mathrm{dBST}_n = \mathrm{dTD}(\mathrm{TD}^{n-1}(\mathrm{REGT}))$ and
$\mathrm{dBS}_n = \mathrm{yield}(\mathrm{dTD}(\mathrm{TD}^{n-1}(\mathrm{REGT})))$.

Proof Let $\mathrm{BST}_n[tt]$ denote the class of tree languages generated by total terminable branching tree grammars of depth n, and similarly in the deterministic case. By Theorem 6.4 and Lemma 4.5, $\mathrm{TD}^{n-1}(\mathrm{REGT}) \subseteq \mathrm{BST}_{n-1}[tt] \subseteq \mathrm{dBST}_n[tt]$. Consequently $\mathrm{dTD}(\mathrm{TD}^{n-1}(\mathrm{REGT})) \subseteq \mathrm{dTD}(\mathrm{dBST}_n[tt])$ which, by Lemma 6.3, is included in $\mathrm{dBST}_n[tt]$. This shows that $\mathrm{dTD}(\mathrm{TD}^{n-1}(\mathrm{REGT})) \subseteq \mathrm{dBST}_n$. The inclusion in the other direction is immediate from Theorems 7.2 and 7.6. The result for ordinary branching grammars follows from the fact that the constructions in the proof of Lemma 5.1 preserve determinism. This also shows that $\mathrm{dBST}_n \subseteq \mathrm{dBST}_n[tt]$ and $\mathrm{dBS}_n \subseteq \mathrm{dBS}_n[tt]$: the deterministic case of Theorem 8.3. ∎

In particular, as in the nondeterministic case, the class dBS_1 of languages generated by deterministic branching ET0L systems (which contains the class of EDT0L languages) is equal to yield(dTD(REGT)).

We know from Lemma 4.5 that $\mathrm{BS}_n \subseteq \mathrm{dBS}_{n+1}$. Thus Corollary 8.2 and Theorem 8.4, together with Theorem 3.14 of [Eng82], lead to the following proper inclusions.

Corollary 8.5 For every $n \in \mathbb{N}$, $\mathrm{BS}_n \subsetneq \mathrm{dBS}_{n+1} \subsetneq \mathrm{BS}_{n+1}$.

9 Generation from a single language

In this final section we use our main result to prove two new results about the tree and string languages that are generated by compositions of top-down tree transducers, i.e., about the classes $\mathrm{TD}^n(\mathrm{REGT})$ and $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$, respectively. Instead of proving these results in all detail, which would result in a rather technical and cumbersome presentation, we put the focus on the main constructions and ideas, and sketch their correctness proofs only on a somewhat informal level.

The first result is that for every $n \geq 1$ there is a single tree language $K_{n-1} \in \mathrm{TD}^{n-1}(\mathrm{REGT})$ such that $\mathrm{TD}^n(\mathrm{REGT}) = \mathrm{TD}(\{K_{n-1}\})$. In other words, the tree languages in the class $\mathrm{TD}(\mathrm{TD}^{n-1}(\mathrm{REGT}))$ can be generated by top-down tree transducers from just one element of $\mathrm{TD}^{n-1}(\mathrm{REGT})$. The second result is that for every n there is a single language $L_n \in \mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ such that $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT})) = \mathrm{FST}(\{L_n\})$, where FST is the class of finite state transductions. This means that the languages in $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ can be generated by finite state transducers from just one of its elements. In other words, $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ which is known to be a full AFL [Bak78b, Theorem 13], is a full *principal* AFL.

The tree language K_{n-1} mentioned above is, in fact, the language SYNC_n of n-synchronization trees with I and J both equal to $\{0,1\}$ (recall from the previous section that SYNC_n depends on I and J). The inclusion $\mathrm{TD}(\{\mathrm{SYNC}_n\}) \subseteq \mathrm{TD}^n(\mathrm{REGT})$ is immediate from Theorem 7.2. To prove that $\mathrm{TD}^n(\mathrm{REGT}) \subseteq \mathrm{TD}(\{\mathrm{SYNC}_n\})$ with $I = J = \{0,1\}$, it follows from our main result (Theorem 8.1) and from Theorem 7.6 that it suffices to show that every tree language in BST_n can be generated by a branching tree grammar with $I = J = \{0,1\}$. This is proved in the following two normal form lemmas, which are of interest in their own right (and hence are stated for arbitrary branching grammars).

We need some additional terminology that defines the rows of the matrix corresponding to a synchronization string. Define for every $\alpha = (i_1, \ldots, i_n) \in I^n$ and every $k \in [n]$, $\mathrm{row}_k(\alpha) = i_k$. This extends to strings of n-tuples in the usual way: $\mathrm{row}_k(\varphi) = \mathrm{row}_k(\alpha_1) \cdots \mathrm{row}_k(\alpha_m)$ for all $\varphi = \alpha_1 \cdots \alpha_m \in (I^n)^*$. Note that $\mathrm{row}_k(\varphi) \in I^*$ and that $|\mathrm{row}_k(\varphi)| = |\varphi|$. Note also that φ is uniquely determined by $\mathrm{row}_1(\varphi), \ldots, \mathrm{row}_n(\varphi)$, and, more generally, that $\mathrm{first}_k(\varphi)$ is uniquely determined by $\mathrm{row}_1(\varphi), \ldots, \mathrm{row}_k(\varphi)$ for every $k \in [n]$.

In the first normal form lemma it is shown that I can be turned into $\{0,1\}$, while J

remains the same.

Lemma 9.1 For every branching grammar $G = (N, T, I, J, R, S)$ of depth n there is a branching grammar $G' = (N', T, \{0,1\}, J, R', S)$ of depth n such that $L(G') = L(G)$. If G is a branching tree grammar, then so is G'.

Proof Since I can always be enlarged, we may assume that $|I| = 2^q$ for some $q > 1$. Let 'code' be a bijection between I and the set of all strings in $\{0,1\}^*$ of length q. We now extend 'code' to I^n: for $\alpha = (i_1, \ldots, i_n) \in I^n$ define $\mathrm{code}(\alpha) = \varphi$, where φ is the unique synchronization string in $(\{0,1\}^n)^*$ of length q, such that $\mathrm{row}_k(\varphi) = \mathrm{code}(i_k)$ for every $k \in [n]$. Furthermore, for $m \in [q]$, we define $\mathrm{code}_m(\alpha) \in \{0,1\}^n$ to be the mth element of $\mathrm{code}(\alpha)$, i.e., $\mathrm{code}(\alpha) = \mathrm{code}_1(\alpha) \cdots \mathrm{code}_q(\alpha)$.

The idea of the construction is to build $\mathrm{code}(\alpha)$ in q steps of the new grammar, where the mth step produces $\mathrm{code}_m(\alpha)$. The set N' of nonterminals of G' consists of all nonterminals of G and of all symbols of the form $B^{\tau,r,t,m}$ where τ is a table of G, $r \in R(\tau)$, B is the tth nonterminal in the right-hand side of r, and $m \in [q-1]$. Note that we could as well take $\langle \tau, r, t, m \rangle$ instead of $B^{\tau,r,t,m}$, but the latter is more convenient. The table specification R' is defined as follows. For every table $\tau \in J^n$, $R'(\tau)$ is obtained from $R(\tau)$ by replacing every rule $r\colon A \to v_0(B_1, \alpha_1)v_1 \cdots (B_l, \alpha_l)v_l$ by the rule $r'\colon A \to v_0(B_1^{\tau,r,1,1}, \mathrm{code}_1(\alpha_1))v_1 \cdots (B_l^{\tau,r,l,1}, \mathrm{code}_1(\alpha_l))v_l$, and all the rules $B_t^{\tau,r,t,m} \to (B_t^{\tau,r,t,m+1}, \mathrm{code}_{m+1}(\alpha_t))$ for $t \in [l]$ and $m \in [q-2]$, and, finally, the rules $B_t^{\tau,r,t,q-1} \to (B_t, \mathrm{code}_q(\alpha_t))$ for $t \in [l]$.

To see that this construction is correct, extend 'code' to a (string) homomorphism from $(I^n)^*$ to $(\{0,1\}^n)^*$ in the canonical way. Now let us consider a derivation $(S, \lambda) \Rightarrow_G^* w_0(A_1, \varphi_1)w_1 \cdots (A_h, \varphi_h)w_h$ in G. It can be shown by induction on its length that in G' there is a derivation $(S, \lambda) \Rightarrow_{G'}^* w_0(A_1, \mathrm{code}(\varphi_1))w_1 \cdots (A_h, \mathrm{code}(\varphi_h))w_h$. Each step of the first derivation is simulated by a corresponding step of the second derivation in which every rule r is simulated by rule r' (taken from the same table), followed by $q-1$ steps of the second derivation in which the rules for the new nonterminals are applied (taken from the same tables). Of course, these $q-1$ steps are not needed if all rules r have terminal right-hand sides (in which case $r' = r$). It is important to realize that the level of synchronization between two nonterminals A_i and A_j is the same in both sentential forms, i.e., $\mathrm{level}(\mathrm{code}(\varphi_i), \mathrm{code}(\varphi_j)) = \mathrm{level}(\varphi_i, \varphi_j)$. This is because, for every $\varphi \in (I^n)^*$ and every $k \in [n]$,

$$\mathrm{row}_k(\mathrm{code}(\varphi)) = \mathrm{code}(\mathrm{row}_k(\varphi)) \tag{1}$$

where, in the right-hand side of the equation, code is viewed as a homomorphism $I^* \to \{0,1\}^*$. Thus, since code is injective, $\mathrm{row}_k(\mathrm{code}(\varphi_i)) = \mathrm{row}_k(\mathrm{code}(\varphi_j))$ iff $\mathrm{row}_k(\varphi_i) = \mathrm{row}_k(\varphi_j)$, and hence $\mathrm{first}_k(\mathrm{code}(\varphi_i)) = \mathrm{first}_k(\mathrm{code}(\varphi_j))$ iff $\mathrm{first}_k(\varphi_i) = \mathrm{first}_k(\varphi_j)$, which implies that the levels of synchronization are the same. The proof of equation (1) is straightforward. In fact, since $\mathrm{row}_k(\varphi\psi) = \mathrm{row}_k(\varphi)\mathrm{row}_k(\psi)$, it suffices to show that $\mathrm{row}_k(\mathrm{code}(\alpha)) = \mathrm{code}(\mathrm{row}_k(\alpha))$ for $\alpha \in I^n$, which holds by definition. Thus, the level of synchronization between A_i and A_j is the same in both sentential forms, and hence the same tables can be applied to these nonterminals in both derivations. Moreover, since by Lemma 4.1 the synchronization between descendants of A_i and A_j can only decrease, these tables can also be used in the next $q-1$ steps of the second derivation.

Vice versa, it can be shown that all derivations of G' are of the form above. ∎

In the second normal form lemma we show that J can be turned into $\{0,1\}$, while I remains the same.

Lemma 9.2 For every branching grammar $G = (N, T, I, J, R, S)$ of depth n there is a branching grammar $G' = (N', T, I, \{0,1\}, R', S)$ of depth n such that $L(G') = L(G)$. If G is a branching tree grammar, then so is G'.

Proof The idea of the proof is similar to the one of the previous lemma. It is also similar to the proof of [Roz73b, Theorem 5] (together with the proof of [Roz73a, Theorem 9]) stating that every ET0L language can be generated by an ET0L system with two tables, which here is the special case $|I| = 1$. Since J can always be enlarged, we may assume that $|J| = 2^q$ for some $q > 1$. Let 'code' be a bijection between J and the set of all strings in $\{0,1\}^*$ of length q. We extend 'code' to tables in J^n, as in the proof of the previous lemma: for $\tau = (j_1, \dots, j_n) \in J^n$, $\mathrm{code}(\tau)$ is the unique string $\beta \in (\{0,1\}^n)^*$ of length q, such that $\mathrm{row}_k(\beta) = \mathrm{code}(j_k)$ for every $k \in [n]$. And again, for $m \in [q]$, $\mathrm{code}_m(\tau) \in \{0,1\}^n$ denotes the mth element of $\mathrm{code}(\tau)$.

As in the proof of the previous lemma, the idea of the construction is to build $\mathrm{code}(\tau)$ in q steps of the new grammar, but this time nondeterministically. This will be done in the nonterminals, by augmenting them with a prefix of $\mathrm{code}(\tau)$ as their second component. In step $m \in [q]$, $\mathrm{code}_m(\tau)$ is added to the second component, resulting in $\mathrm{code}_1(\tau) \cdots \mathrm{code}_m(\tau)$. Accordingly, we let the set N' of nonterminals of G' consist of all nonterminals of G and of all symbols of the form $\langle A, \beta \rangle$ with $A \in N$ and $\beta \in (\{0,1\}^n)^*$ of length at most q (and $\beta \neq \lambda$).

Let α_0 be an arbitrary, but fixed, element of I^n. For every table $\tau' \in \{0,1\}^n$ of G', $R'(\tau')$ contains for every $A \in N$ the rule $A \to (\langle A, \tau' \rangle, \alpha_0)$ and all the rules $\langle A, \beta \rangle \to (\langle A, \beta\tau' \rangle, \alpha_0)$ with $|\beta| < q$. Thus, τ' corresponds to $\mathrm{code}_m(\tau)$ in the previous paragraph. In addition, $R'(\tau')$ contains all the rules $\langle A, \beta \rangle \to \zeta$ where $|\beta| = q$ and $A \to \zeta$ is a rule in the table $R(\mathrm{code}^{-1}(\beta))$ of G. We observe here that the rules of the last kind could be put in just one, arbitrarily chosen, table of G'.

Let $\gamma\colon (I^n)^* \to (I^n)^*$ be the homomorphism defined by $\gamma(\alpha) = \alpha_0^q \alpha$ for every $\alpha \in I^n$. Consider a derivation $(S, \lambda) \Rightarrow_G^* w_0(A_1, \varphi_1)w_1 \cdots (A_h, \varphi_h)w_h$ in G. It can be shown by induction on its length that there is a derivation $(S, \lambda) \Rightarrow_{G'}^* w_0(A_1, \gamma(\varphi_1))w_1 \cdots (A_h, \gamma(\varphi_h))w_h$ in G'. Note that A_i and A_j have the same level of synchronization in both sentential forms, i.e., $\mathrm{level}(\gamma(\varphi_i), \gamma(\varphi_j)) = \mathrm{level}(\varphi_i, \varphi_j)$, because the α_0's do not influence the synchronization. Each step of the first derivation is simulated by $q+1$ steps of the second derivation. If a rule $A \to \zeta$ taken from a table τ is applied in that step of the first derivation, then, in the second derivation, rules are applied that are taken from the tables $\mathrm{code}_1(\tau), \dots, \mathrm{code}_q(\tau)$ and an arbitrary table. The last rule that is applied is $\langle A, \mathrm{code}(\tau) \rangle \to \zeta$. Vice versa, it can be shown that all derivations of G' are of the form above.

Observe that if $\mathrm{level}(\gamma(\varphi_i), \gamma(\varphi_j)) = k$, then in the first q steps of the second derivation, as above, the synchronization level does not change and hence, at each such step, only tables τ_i' and τ_j' of G' can be applied to A_i and A_j, respectively, such that $\mathrm{first}_k(\tau_i') = \mathrm{first}_k(\tau_j')$. Consequently, $(A_i, \gamma(\varphi_i))$ and $(A_j, \gamma(\varphi_j))$ are replaced by $(\langle A_i, \beta \rangle, \gamma(\varphi_i)\alpha_0^q)$ and $(\langle A_j, \delta \rangle, \gamma(\varphi_j)\alpha_0^q)$ with $\mathrm{first}_k(\beta) = \mathrm{first}_k(\delta)$. Hence $\mathrm{first}_k(\mathrm{code}^{-1}(\beta)) = \mathrm{first}_k(\mathrm{code}^{-1}(\delta))$ and so, in the next step of the second derivation, only the application of tables τ_β and τ_δ of G (to A_i and A_j, respectively) can be simulated that are properly synchronized, i.e., for which $\mathrm{first}_k(\tau_\beta) = \mathrm{first}_k(\tau_\delta)$. ■

Convention From now on, let $I = J = \{0, 1\}$. In particular, SYNC_n is the set of n-synchronization trees with $I = J = \{0, 1\}$.

Theorem 9.3 For every $n \geq 1$, $\mathrm{TD}^n(\mathrm{REGT}) = \mathrm{TD}(\{\mathrm{SYNC}_n\})$.

Proof Since, by Theorem 7.2, $\mathrm{SYNC}_n \in \mathrm{TD}^{n-1}(\mathrm{REGT})$, it holds that $\mathrm{TD}(\{\mathrm{SYNC}_n\}) \subseteq \mathrm{TD}^n(\mathrm{REGT})$. Conversely, by Theorem 6.4, $\mathrm{TD}^n(\mathrm{REGT}) \subseteq \mathrm{BST}_n$. By the two lemmas above and Theorem 7.6, $\mathrm{BST}_n \subseteq \mathrm{TD}(\{\mathrm{SYNC}_n\})$. ∎

One consequence of this theorem is that $\mathrm{SYNC}_n \in \mathrm{TD}^{n-1}(\mathrm{REGT}) - \mathrm{TD}^{n-2}(\mathrm{REGT})$, because applying TD to $\mathrm{SYNC}_n \in \mathrm{TD}^{n-2}(\mathrm{REGT})$ would give that $\mathrm{TD}^n(\mathrm{REGT}) \subseteq \mathrm{TD}^{n-1}(\mathrm{REGT})$, contradicting the properness of the top-down tree transducer hierarchy [Eng82] (see Corollary 8.5). Thus, the synchronization tree languages separate the levels of this hierarchy.

Another consequence is that $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT})) = \mathrm{yield}(\mathrm{TD}(\{\mathrm{SYNC}_n\}))$. It even follows that $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT})) = \mathrm{CTPD}(\{\mathrm{SYNC}_n\})$, where CTPD is the class of (tree-to-string) transductions realized by the CTPD transducers of [ERS80]. In fact, by Corollary 4.6 of [ERS80] and Lemma 2.4 of [Eng82], $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT})) = \mathrm{CTPD}(\mathrm{TD}^{n-1}(\mathrm{REGT}))$ and so $\mathrm{CTPD}(\{\mathrm{SYNC}_n\}) \subseteq \mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$. The other direction follows from the fact (shown in [ERS80, Theorem 4.5]) that $\mathrm{yield}(\mathrm{TD}(\mathcal{L})) \subseteq \mathrm{CTPD}(\mathcal{L})$ for any class $\mathcal{L}$ of tree languages, and hence $\mathrm{yield}(\mathrm{TD}(\{\mathrm{SYNC}_n\})) \subseteq \mathrm{CTPD}(\{\mathrm{SYNC}_n\})$.

In the remainder of this section we will show that $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ is a full principal AFL, for every n. We assume the reader to be familiar with AFL and AFA theory (see [Gin75]) and with the CTPD transducer of [ERS80].

Recall from [ERS80] that a CTPD transducer is a nondeterministic tree-walking automaton with an output tape and a pushdown. At the current node of the input tree it can test the label of the node and the symbol on top of the pushdown. When it moves down to a child of the node it simultaneously pushes one symbol on the pushdown, and when it moves up to the parent of the node it simultaneously pops the top symbol of the pushdown. It starts its computations at the root of the input tree, with the initial pushdown symbol on its pushdown. Note that, as a consequence of the synchronization of the tree walk and the pushdown instructions, the length of the pushdown always equals the length of the path from the root to the current node of the tree. The transduction computed by a CTPD transducer M will also be denoted M.

The above equation $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT})) = \mathrm{CTPD}(\{\mathrm{SYNC}_n\})$ gives a machine model for $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$, and hence for BS_n: it is the CTPD transducer, viewed as an acceptor of its output tape, guessing an input synchronization tree from SYNC_n in its storage. It is well known from AFA theory that any "reasonable" machine model accepts a class of languages that is a full AFL. Moreover, if the machine model is "finitely encoded", then the language class is a full principal AFL, which means that the class is generated by finite state transducers from one of its languages (see [Gin75, Chapter 5]). By definition, a machine model is finitely encoded if the set of all its possible tests and instructions on the storage may be assumed to be finite[8]. The CTPD automaton has storage tests

[8] Viewing that set as an alphabet, let L_n be the language consisting of all "successful" test and instruction sequences (meaning that such a sequence can be executed on a synchronization tree in the CTPD storage). A CTPD transducer M can be simulated by a finite state transducer F that receives a string from L_n as input (instead of an input tree from SYNC_n); F simulates the state behaviour and output instructions of M directly, but consults its input string for the simulation of M's storage.

label $=\sigma$ and top $=\gamma$ (which test the label of the current node and the top of the pushdown, respectively), and it has storage instructions up, stay(γ), and down$_i(\gamma)$: move up and pop, stay at the current node and change the topmost pushdown symbol into γ, move down to the ith child and push γ, respectively. Since the input alphabet of M is fixed to be $\Sigma_{\langle n\rangle}$, it only needs the finitely many tests label $=\sigma$ with $\sigma \in \Sigma_{\langle n\rangle}$, and it only needs the instructions down$_i(\gamma)$ with $i \in [2^n]$. Thus, to show that finitely many tests and instructions suffice, it remains to argue that we may restrict the pushdown alphabet to be $\{0,1\}$ (just as in the case of ordinary pushdown automata). To simulate a CTPD transducer with pushdown alphabet Γ by one with pushdown alphabet $\{0,1\}$, the straightforward idea is, again, to code each pushdown symbol by a string in $\{0,1\}^*$ of length q, where $|\Gamma| \leq 2^q$. Since the tree walk and the pushdown are synchronized, this is possible only if each node of the input tree is "stretched" into q nodes. This is defined formally as follows. In the definition, leaves(s) denotes the set of leaves of a tree $s \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$, addressed by synchronization strings, i.e., leaves($\perp$) $=\{\lambda\}$ and for $s \neq \perp$, leaves(s) $=\{\varphi\alpha \mid \varphi \in \mathrm{internal}(s), \alpha \in I^n, \varphi\alpha \notin \mathrm{internal}(s)\}$.

Let $\delta = (0,\ldots,0) = \mathrm{num}^{-1}(1) \in I^n$, where num is defined as in Section 7. For a tree $s \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$ and $q \in \mathbb{N}$ with $q > 1$, we define the *q-stretching* $\mathrm{stretch}_q(s)$ of s to be the tree $s' \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$ for which

(1) internal(s') consists of all synchronization strings $\delta^{q-1}\alpha_1\delta^{q-1}\cdots\alpha_{m-1}\delta^{q-1}\alpha_m\delta^j$ (with $m \geq 0$ and $\alpha_1,\ldots,\alpha_m \in I^n$) such that *either* $\alpha_1\cdots\alpha_m \in \mathrm{internal}(s)$ and $0 \leq j \leq q-1$, *or* $\alpha_1\cdots\alpha_m \in \mathrm{leaves}(s)$ and $0 \leq j \leq q-2$, and

(2) the labels are defined as follows: for every node $\alpha_1\cdots\alpha_m \in \mathrm{internal}(s)$, we let $s'(\delta^{q-1}\alpha_1\delta^{q-1}\cdots\alpha_m\delta^{q-1}) = s(\alpha_1\cdots\alpha_m)$, and all other internal nodes of s' have the label τ_0, an arbitrary, fixed element of J^n.

Intuitively, each node $\alpha_1\cdots\alpha_m$ of s (internal node or leaf) corresponds to the node $\delta^{q-1}\alpha_1\delta^{q-1}\cdots\alpha_m\delta^{q-1}$ of s', which has a chain of $q-1$ internal nodes above it, viz. the nodes $\delta^{q-1}\alpha_1\delta^{q-1}\cdots\alpha_m\delta^j$ with $0 \leq j \leq q-2$. Figure 2 shows an example, where $n=1$ and $q=3$.

Lemma 9.4 For every tree $s \in \mathrm{T}_{\Sigma_{\langle n\rangle}}$ and $q > 1$, $\mathrm{stretch}_q(s)$ is an n-synchronization tree if and only if s is an n-synchronization tree.

Proof It should be clear that level$(\delta^{q-1}\alpha_1\delta^{q-1}\cdots\alpha_m\delta^{q-1}, \delta^{q-1}\beta_1\delta^{q-1}\cdots\beta_m\delta^{q-1})$ is equal to level$(\alpha_1\cdots\alpha_m, \beta_1\cdots\beta_m)$, which is $\leq$ level$(s(\alpha_1\cdots\alpha_m), s(\beta_1\cdots\beta_m))$ if s or $\mathrm{stretch}_q(s)$ is an n-synchronization tree. Since all other internal nodes of $\mathrm{stretch}_q(s)$ have label τ_0, this proves the lemma. ∎

In other words, $\mathrm{stretch}_q(\mathrm{SYNC}_n) = \mathrm{SYNC}_n \cap \mathrm{stretch}_q(\mathrm{T}_{\Sigma_{\langle n\rangle}})$.

Lemma 9.5 For every CTPD transducer M there is a CTPD transducer M' with pushdown alphabet $\{0,1\}$ such that $M'(\mathrm{SYNC}_n) = M(\mathrm{SYNC}_n)$.

Proof It should be intuitively clear that if M is a CTPD transducer with a pushdown alphabet Γ such that $|\Gamma| \leq 2^q$, $q > 1$, then there is a CTPD transducer M' with pushdown alphabet $\{0,1\}$ such that $M'(\mathrm{stretch}_q(s)) = M(s)$ for every $s \in \mathrm{SYNC}_n$, and $M'(s') = \emptyset$ for every $s' \in \mathrm{SYNC}_n - \mathrm{stretch}_q(\mathrm{SYNC}_n)$, i.e., every s' that is not of the form $\mathrm{stretch}_q(s)$.

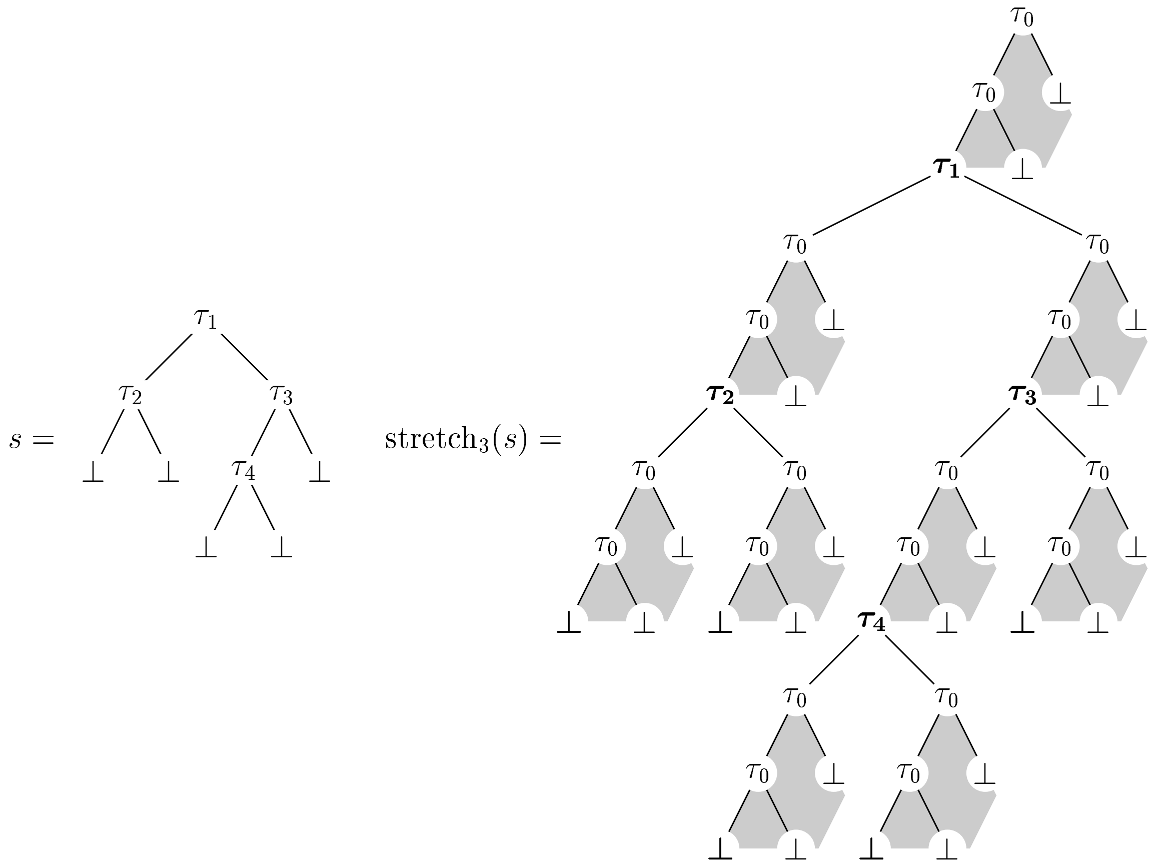


Figure 2: Stretching a synchronization tree

Then, by Lemma 9.4, $M'(\mathrm{SYNC}_n) = M(\mathrm{SYNC}_n)$. To see in more detail how M' works, let 'code' be an injection of Γ into the set of all strings in $\{0,1\}^*$ of length q. The CTPD transducer M' starts with a depth-first tree walk, checking that $s' = \mathrm{stretch}_q(s)$ for some $s \in \mathrm{T}_{\Sigma_{\langle n \rangle}}$. This can be implemented in a straightforward way, assuming (without loss of generality) that, additionally, $q \geq n$: at a current node with label $\neq \tau_0$, the topmost n symbols of the pushdown can be used to store the number of children that have already been visited during this walk (note that every internal node has rank 2^n). After checking that $s' = \mathrm{stretch}_q(s)$, M' simulates the computation of M on s, pushing and popping $\mathrm{code}(\gamma)$ instead of γ. To simulate M, M' first executes $q-1$ instructions $\mathrm{down}_1(0)$, where we assume that the initial pushdown symbol of M is coded as 0^q and that M' has initial pushdown symbol 0. Thus, M' is now at the node of s' that corresponds to the root of s. Then, M' simulates every instruction and test of M by an appropriate sequence of instructions and tests. An up instruction is simulated by q up instructions. A $\mathrm{stay}(\gamma)$ instruction is simulated by $q-1$ up instructions, followed by a stay instruction and $q-1$ down_1 instructions that produce $\mathrm{code}(\gamma)$ on the pushdown. A $\mathrm{down}_i(\gamma)$ instruction is simulated by a down_i instruction and $q-1$ down_1 instructions, that push $\mathrm{code}(\gamma)$ on the pushdown. A test $\mathrm{label} = \sigma$ is simulated by that same test, and finally, a test $\mathrm{top} = \gamma$ is simulated by $q-1$ up instructions, together with q $\mathrm{top} = 0$ and $\mathrm{top} = 1$ tests to verify that $\mathrm{code}(\gamma)$ is on the pushdown, followed by $q-1$ down_1 instructions (that repair the pushdown). ∎

Theorem 9.6 For every $n \geq 1$, $\mathrm{yield}(\mathrm{TD}^n(\mathrm{REGT}))$ is a full principal AFL.

Proof After Theorem 9.3 we have proved that yield(TD^n(REGT)) = CTPD({SYNC_n}). By Lemma 9.5 this shows that yield(TD^n(REGT)) has a machine model (or AFA) that is finitely encoded. It is straightforward to define this AFA formally. Thus, by [Gin75, Theorem 5.2.1], yield(TD^n(REGT)) is a full principal AFL. ■

References

[AD94] Yves André and Max Dauchet. Decidability of equivalence for a class of non-deterministic tree transducers. *RAIRO Informatique Théoretique et Applications/Theoretical Informatics and Applications*, 28:447–463, 1994.

[Bak78a] Brenda S. Baker. Generalized syntax directed translation, tree transducers, and linear space. *SIAM Journal of Computing*, 7:376–391, 1978.

[Bak78b] Brenda S. Baker. Tree transducers and tree languages. *Information and Control*, 37:241–266, 1978.

[Bak79] Brenda S. Baker. Composition of top-down and bottom-up tree transductions. *Information and Control*, 41:186–213, 1979.

[DE98] Frank Drewes and Joost Engelfriet. Decidability of the finiteness of ranges of tree transductions. *Information and Computation*, 145:1–50, 1998.

[DF98] G. Dányi and Zoltán Fülöp. Compositions with superlinear deterministic top-down tree transducers. *Theoretical Computer Science*, 194:57–85, 1998.

[Dre96] Frank Drewes. A lower bound on the growth of functions computed by tree transducers. *Fundamenta Informaticae*, 26:267–286, 1996.

[Dre00] Frank Drewes. Tree-based picture generation. *Theoretical Computer Science*, 246:1–51, 2000.

[Dre01a] Frank Drewes. The complexity of the exponential output size problem for top-down and bottom-up tree transducers. *Information and Computation*, 169:264–283, 2001.

[Dre01b] Frank Drewes. Tree-based generation of languages of fractals. *Theoretical Computer Science*, 262:377–414, 2001.

[Eng75] Joost Engelfriet. Bottom-up and top-down tree transformations—a comparison. *Mathematical Systems Theory*, 9:198–231, 1975.

[Eng76] Joost Engelfriet. Surface tree languages and parallel derivation trees. *Theoretical Computer Science*, 2:9–27, 1976.

[Eng77] Joost Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10:289–303, 1977.

[Eng78] Joost Engelfriet. On tree transducers for partial functions. *Information Processing Letters*, 7:170–172, 1978.

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

[ERS80] Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. Tree transducers, L systems, and two-way machines. *Journal of Computer and System Sciences*, 20:150–202, 1980.

[ES77] Joost Engelfriet and Erik Meineche Schmidt. IO and OI. I. *Journal of Computer and System Sciences*, 15:328–353, 1977.

[FV89] Zoltán Fülöp and Sándor Vágvölgyi. Variants of top-down tree transducers with look-ahead. *Mathematical Systems Theory*, 21:125–145, 1989.

[FV98] Zoltán Fülöp and Heiko Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. Springer, 1998.

[Gin75] Seymour Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland, Amsterdam, 1975.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

[GV96] Pál Gyenizse and Sándor Vágvölgyi. Compositions of deterministic bottom-up, top-down, and regular look-ahead tree transformations. *Theoretical Computer Science*, 156:71–97, 1996.

[Man98] Sebastian Maneth. The generating power of total deterministic tree transducers. *Information and Computation*, 147:111–144, 1998.

[MN00] Sebastian Maneth and Frank Neven. Structured document transformations based on XSL. In R. Connor and A. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming - Revised Papers DBPL'99*, volume 1949 of *Lecture Notes in Computer Science*, pages 80–98, 2000.

[MN03] Wim Martens and Frank Neven. Typechecking top-down uniform unranked tree transducers. In *Proc. 9th Intl. Conf. on Database Theory (ICDT 2003)*, Lecture Notes in Computer Science, 2003. To appear.

[Rou70] William C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4:257–287, 1970.

[Roz73a] Grzegorz Rozenberg. T0L systems and languages. *Information and Control*, 23:262–283, 1973.

[Roz73b] Grzegorz Rozenberg. Extension of tabled 0L systems and languages. *International Journal of Computer and Information Sciences*, 2:311–334, 1973.

[RS97] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*, volume 1–3. Springer, 1997.

[Sei94] Helmut Seidl. Equivalence of finite-valued tree transducers is decidable. *Mathematical Systems Theory*, 27:285–346, 1994.

[Sky76] Sven Skyum. Decomposition theorems for various kinds of languages parallel in nature. *SIAM Journal of Computing*, 5:284–296, 1976.

[SV95] Giora Slutzki and Sándor Vágvölgyi. Deterministic top-down tree transducers with iterated look-ahead. *Theoretical Computer Science*, 143:285–308, 1995.

[Tha70a] James W. Thatcher. Generalized^2 sequential machine maps. *Journal of Computer and System Sciences*, 4:339–367, 1970.

[Tha70b] James W. Thatcher. There's a lot more to finite automata theory than you would have thought. In *Proc. 4th Ann. Princeton Conf. on Information Sciences and Systems*, pages 263–276, 1970. Revised version published as [Tha73].

[Tha73] James W. Thatcher. Tree automata: an informal survey. In A.V. Aho, editor, *Currents in the Theory of Computing*, pages 143–172. Prentice Hall, 1973.