A Pumping Lemma for Output Languages of Macro Tree Transducers

Armin Kühnemann

Grundlagen der Programmierung, Institut für Softwaretechnik I,
Fakultät Informatik, Technische Universität Dresden, D–01062 Dresden, Germany,
e–mail: kuehne@orchid.inf.tu–dresden.de

Abstract. The concept of macro tree transducer is a formal model for studying properties of syntax–directed translations. In this paper, for output languages of producing, nondeleting, and noncopying macro tree transducers, we introduce a pumping lemma. We apply the pumping lemma to gain the following result: there is no producing and nondeleting macro tree transducer which computes the set of all monadic trees with double exponential height as output.

Keywords: Macro tree transducer, Primitive Recursion, Pumping Lemma

1 Introduction

Pumping lemmata are useful tools to prove that a given language L is not an element of a class $\mathcal{L}$ of formal languages, where $\mathcal{L}$ usually is defined by a class of grammars or translation schemes. Pumping lemmata have been investigated for example for string languages, tree languages, and graph languages.

First Aho and Ullman have inspected pumping lemmata for output languages of translation schemes in [1], namely for generalized syntax directed translations. Perrault and Ésik have introduced in [17] and [10], respectively, pumping lemmata for (nondeterministic) top–down tree transducers (cf. [18, 19, 4]). Engelfriet, Rozenberg, and Slutzki have presented in [7] a pumping lemma for output languages of deterministic top–down tree–to–string transducers. Kühnemann and Vogler have investigated in [15] a pumping lemma for output languages of restricted attributed tree transducers (cf. [11]). The latter two lemmata had a big influence on the development of the pumping lemma for output languages of restricted macro tree transducers which we present here. The contents of this paper is a part of the technical report [14].

The concept of macro tree transducer (in the following abbreviated by mtt) has been invented by Engelfriet in [5] (cf. also [3, 8]). In [8] it is described that the syntax–directed translations of Irons in [13] can already be seen as mtts with only one function. An mtt can be understood as extended primitive recursive program scheme over trees, where additionally simultaneous recursion and nested function calls in parameter positions are allowed. Clearly, mtts do not represent the class of primitive recursive tree functions. In [9] and [12] the relation between tree transducers and primitive recursive tree functions is examined in detail.

An mtt M takes input trees over a ranked alphabet Σ of input symbols and delivers output trees over a ranked alphabet Δ of output symbols. For this translation process, M uses a ranked alphabet F of functions which have at least rank 1, and a set R of rules which describe the possible derivations. For every function f with rank $n+1$ and for every input symbol σ with rank k there is exactly one rule $f(\sigma(x_1,\ldots,x_k),y_1,\ldots,y_n) \to \rho$, where $x_1,\ldots,x_k$ (and $y_1,\ldots,y_n$) are recursion variables (and context variables, respectively) and ρ is a tree over F, Δ, and $x_1,\ldots,x_k,y_1,\ldots,y_n$, such that for every subtree of ρ of the form $f'(\rho_0,\rho_1,\ldots,\rho_m)$ with $f' \in F$, the tree ρ_0 has to be a variable x_i.

Every mtt M computes a total function from input trees to output trees. This function is called the tree transformation of M. Its output language is defined as the range of the tree transformation of M.

Our pumping lemma is restricted to producing, nondeleting, and noncopying mtts: An mtt is producing, if the right–hand side of every rule contains at least one output symbol. An mtt is nondeleting (noncopying), if every variable (every context variable) of the left–hand side of a rule occurs at least once (at most once, respectively) on the corresponding right–hand side.

The central idea of our pumping lemma is adopted from the pumping lemma for context–free string languages of Bar–Hillel, Perles, and Shamir (cf. [2]). There we have to inspect a derivation tree of a sufficiently long string to deduce new pumped strings. In our case we consider an input tree corresponding to a sufficiently large output tree to obtain new pumped output trees.

For every producing, nondeleting, and noncopying mtt M, a natural number pi_M, called the **p**umping **i**ndex of M, can be constructed. If we choose a tree t from the output language of M which has at least the size pi_M, then every input tree e which can be transformed into t has the following property: e is high enough, such that it has a path p, on which two different nodes x_1 and x_2 can be found, which have the same set of function occurrences that are needed to calculate subtrees of t. Assuming that x_1 is closer to the root of e than x_2, we can define the following tree $\bar{e}$: Roughly speaking, the tree $\bar{e}$ is that part of e which has x_1 as root and from which the subtree rooting at x_2 is pruned. Since the two nodes are compatible with respect to the needed function occurrences, we can construct new input trees by repeating $\bar{e}$ arbitrarily many times. Translating these input trees by M, we obtain new elements of the output language of M.

The decomposition of e induces a decomposition of t into output patterns. These patterns are used to construct the output trees. Thus the pumping process can be described by using only the output patterns. Therefore the applications of the pumping lemma are completely independent of the underlying input trees.

In this paper we apply our pumping lemma to prove the following result: There is no producing and nondeleting mtt which computes the set of all monadic trees with double exponential height as output.

The paper is divided into six sections, from which this one is the first. In Section 2 we fix elementary notions and notations. Section 3 introduces mtts. Section 4 contains the pumping lemma. The application is shown in Section 5. Finally, Section 6 contains a short summary and presents further research topics.

2 Preliminaries

We denote the set of natural numbers including 0 by $\mathbb{N}$. For every $m \in \mathbb{N}$, the set $\{1, \ldots, m\}$ is denoted by $[m]$. We shall use the sets $X = \{x_1, x_2, x_3, \ldots\}$ and $Y = \{y_1, y_2, y_3, \ldots\}$ of *variables* and for every $k \in \mathbb{N}$ the abbreviations $X_k = \{x_1, \ldots, x_k\}$ and $Y_k = \{y_1, \ldots, y_k\}$. For an arbitrary set S, the cardinality of S is denoted by $card(S)$. If $f : S \longrightarrow S$ is a total function, then, for every $i \in \mathbb{N}$, the function $f^i : S \longrightarrow S$ is defined as follows: for every $s \in S$, $f^0(s) = s$ and if $i > 0$, then $f^i(s) = f(f^{i-1}(s))$.

If A is an alphabet, then A^* denotes the set of *strings* over A. The empty string is denoted by ε. For a string v and two lists $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ of strings such that no pair u_i and u_j overlaps in v, we abbreviate by $v[u_1/v_1, \ldots, u_n/v_n]$ the string which is obtained from v by replacing every occurrence of u_i in v by v_i. The resulting string is also denoted by $v[u_i/v_i\,;\, i \in [n]]$. If v is a string, then $v^0 = \varepsilon$ and $v^i = vv^{i-1}$ for every $i > 0$. If V_1 and V_2 are two sets of strings, then $V_1 \cdot V_2 = \{v_1 v_2 \mid v_1 \in V_1,\, v_2 \in V_2\}$.

Let $\Rightarrow$ be a binary relation on some set T. Then, $\Rightarrow^*$ denotes the transitive, reflexive closure of $\Rightarrow$. If $t \Rightarrow^* t'$ for $t, t' \in T$ and if there is no $t'' \in T$ such that $t' \Rightarrow t''$, then t' is called a *normal form of t with respect to* $\Rightarrow$. If the normal form of t with respect to $\Rightarrow$ exists and if it is unique, then it is denoted by $nf(\Rightarrow, t)$.

A *ranked alphabet* is a pair $(\Sigma, rank_\Sigma)$ where Σ is a finite set and $rank_\Sigma : \Sigma \longrightarrow \mathbb{N}$ is a mapping which associates with every symbol $\sigma \in \Sigma$ a natural number $rank_\Sigma(\sigma)$ called the rank of the symbol. If $\sigma \in \Sigma$ with $rank_\Sigma(\sigma) = n$, and Σ is clear from the context, then we also write $\sigma^{(n)}$ or $rank(\sigma) = n$. If the rank function is clear from the context, then it is dropped from the notation. The set of elements with rank n is denoted by $\Sigma^{(n)}$.

Let Σ be a ranked alphabet and let S be a set of symbols. The set of *trees over Σ indexed by S*, denoted by $T\langle\Sigma\rangle(S)$, is the smallest set T such that (i) $S \subseteq T$ and (ii) for every $\sigma \in \Sigma^{(n)}$ with $n \geq 0$ and $t_1, \ldots, t_n \in T$, $\sigma(t_1, \ldots, t_n) \in T$. $T\langle\Sigma\rangle(\emptyset)$ is abbreviated by $T\langle\Sigma\rangle$. We sometimes omit the parentheses for the argument of a symbol $\sigma \in \Sigma^{(1)}$. The following objects can be defined inductively for every tree $t \in T\langle\Sigma\rangle(S)$:

- $height(t) \in \mathbb{N}$ is the *height of t*, where $height(\sigma) = 1$ if $\sigma \in \Sigma^{(0)} \cup S$.
- $size_{S'}(t) \in \mathbb{N}$ is the *size of t with respect to a subset $S' \subseteq \Sigma \cup S$*, i.e. the number of nodes of t which are labeled by a symbol $\sigma \in S'$. If $S' = \Sigma \cup S$, then we abbreviate $size_{S'}(t)$ by $size(t)$.
- $paths(t) \subseteq \mathbb{N}^*$ is the *set of paths of t*. If $t \in \Sigma^{(0)} \cup S$, then $paths(t) = \{\varepsilon\}$. If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 1$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle(S)$, then $paths(t) = \{\varepsilon\} \cup \{ip \mid i \in [n], p \in paths(t_i)\}$.
- $label(t, p) \in \Sigma \cup S$ is the *label of the node of t reached by a path $p \in paths(t)$*.
- $subtree(t, p) \in T\langle\Sigma\rangle(S)$ is the *subtree of t reached by a path $p \in paths(t)$*.
- $t[p \leftarrow t'] \in T\langle\Sigma\rangle(S)$ is the *tree obtained from t by replacing the subtree reached by a path $p \in paths(t)$, by another tree $t' \in T\langle\Sigma\rangle(S)$*.

For every tree $t \in T\langle\Sigma\rangle(S)$ and for every path $p \in paths(t)$, the path p determines exactly one node of t. This node will also be denoted by p.

For every tree $t \in T\langle\Sigma\rangle(S)$ and for every path $p \in paths(t)$ we define the subset $P(t,p) = \{p\} \cdot paths(subtree(t,p))$ of $paths(t)$. For every tree $t \in T\langle\Sigma\rangle(S)$ and for every paths $p, p' \in paths(t)$, such that there is a string p'' with $p' = pp''$, we abbreviate $P(t,p) - P(t,p')$ by $P(t,p-p')$.

A *tree transformation* is a total function $\tau : T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ where Σ and Δ are ranked alphabets. The *output language* $L_{out}(\tau)$ *of* τ is defined as range of τ.

3 Macro Tree Transducers

Definition 3.1 A *macro tree transducer* is a tuple $(F, \Delta, \Sigma, f_{in}, R)$ where

- F is the ranked alphabet of *functions*; for every $f \in F$, $rank_F(f) \geq 1$.
- Δ is the ranked alphabet of *output symbols* with $F \cap \Delta = \emptyset$.
- Σ is the ranked alphabet of *input symbols* with $F \cap \Sigma = \emptyset$.
- $f_{in} \in F^{(1)}$ is the *initial function*.
- R is the finite set of *rules*: For every $n \geq 0$, $f \in F^{(n+1)}$, $k \geq 0$, and $\sigma \in \Sigma^{(k)}$, the set R contains exactly one rule $f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n) \rightarrow \rho$ with $\rho \in RHS(F, \Delta, X_k, Y_n)$.

For every $k \geq 0$ and $n \geq 0$, the set of *right-hand sides over* F, Δ, X_k, *and* Y_n, denoted by $RHS(F, \Delta, X_k, Y_n)$, is the smallest set RHS such that:

(i) For every $r \geq 0$, $\delta \in \Delta^{(r)}$, and $\rho_1, \ldots, \rho_r \in RHS$, $\delta(\rho_1, \ldots, \rho_r) \in RHS$.
(ii) For every $m \geq 0$, $f \in F^{(m+1)}$, $x_i \in X_k$, and $\rho_1, \ldots, \rho_m \in RHS$, $f(x_i, \rho_1, \ldots, \rho_m) \in RHS$.
(iii) For every $i \in [n]$, $y_i \in RHS$. □

The first argument of every function is called the *recursion argument*. The other arguments are called *context parameters*. An mtt translates trees of the set $T\langle\Sigma\rangle$, called *input trees*, into trees of the set $T\langle\Delta\rangle$, called *output trees* (cf. Def. 3.5).

Example 3.2 We define the mtt $M_1 = (F_1, \Delta_1, \Sigma_1, f_{in,1}, R_1)$ with $F_1 = \{f^{(1)}_{in,1}, f^{(2)}, g^{(2)}\}$, $\Delta_1 = \{T^{(3)}, A^{(1)}, B^{(1)}, L^{(1)}, R^{(1)}, E^{(0)}\}$, $\Sigma_1 = \{\sigma^{(2)}, \alpha^{(0)}\}$, and

$$
\begin{aligned}
R_1 = \{ & f_{in,1}(\sigma(x_1, x_2)) \rightarrow T(f(x_1, L(E)), f(x_2, R(E)), g(x_1, g(x_2, E))), \\
& f_{in,1}(\alpha) \rightarrow B(E), \\
& f(\sigma(x_1, x_2), y_1) \rightarrow T(f(x_1, L(y_1)), f(x_2, R(y_1)), g(x_1, g(x_2, E))), \\
& f(\alpha, y_1) \rightarrow B(y_1), \\
& g(\sigma(x_1, x_2), y_1) \rightarrow g(x_1, A(g(x_2, y_1))), \\
& g(\alpha, y_1) \rightarrow A(y_1) \quad \}.
\end{aligned}
$$

M_1 takes a tree e over Σ_1 and it delivers a tree t which has almost the same structure as e. Every node n, which is labeled by σ, is substituted by a node labeled by a ternary output symbol T, where the additional third subtree is a tree over $\{A^{(1)}, E^{(0)}\}$ which is as high as the size of the input subtree rooting at n. Every node n, which is labeled by α, is substituted by an encoding of the reverse path from the root of e to n. The encoding of a reverse path is a tree

over $\{B^{(1)}, L^{(1)}, R^{(1)}, E^{(0)}\}$, where the symbol L (and R) represent the left child (and the right child, respectively) of a node and the symbol B (and E) is the first symbol (and the last symbol, respectively) of each path encoding. □

The set of sentential forms of an mtt $M = (F, \Delta, \Sigma, f_{in}, R)$ usually is defined as set of trees over F, Σ, and Δ, where the recursion argument of every function has to be a subtree of an input tree e and, vice versa, an input subtree may only occur as recursion argument of a function. Rather than this, we use paths of e instead of subtrees of e. Thus we can determine the exact occurrence of a subtree in an input tree. The input tree e itself will parameterize the derivation relation. For later purposes we define the set of sentential forms for an arbitrary subset $P \subseteq paths(e)$ of paths and for a superset $\Delta' \supseteq \Delta$ of output symbols.

Definition 3.3 Let $M = (F, \Delta, \Sigma, f_{in}, R)$ be an mtt, $e \in T\langle\Sigma\rangle$, $P \subseteq paths(e)$, and Δ' be a ranked alphabet with $\Delta \subseteq \Delta'$. The set $SF(F, P, \Delta')$ of (F, P, Δ')–*sentential forms* is the smallest set SF such that:

(i) For every $r \geq 0$, $\delta \in \Delta'^{(r)}$, and $t_1, \ldots, t_r \in SF$, $\delta(t_1, \ldots, t_r) \in SF$.
(ii) For every $n \geq 0$, $f \in F^{(n+1)}$, $p \in P$, and $t_1, \ldots, t_n \in SF$, $f(p, t_1, \ldots, t_n) \in SF$. □

For an mtt $M = (F, \Delta, \Sigma, f_{in}, R)$ and for an input tree $e \in T\langle\Sigma\rangle$, the set of *function occurrences of* e, denoted by $fun(e)$, is the set $F \times paths(e)$. If $f(p, t_1, \ldots, t_n) \in SF(F, P, \Delta')$ is a subtree of another sentential form $sf \in SF(F, P, \Delta')$, then (f, p) is called a *function occurrence of* sf.

We describe the derivation relation of an mtt M with respect to an input tree e. For later purposes, we restrict the derivation relation to work only on particular parts of e. That means, we parameterize the derivation relation with a subset $P \subseteq paths(e)$ and only if $p \in P$, then a function occurrence (f, p) of a sentential form can be derived further.

Definition 3.4 Let $M = (F, \Delta, \Sigma, f_{in}, R)$ be an mtt. Let $e \in T\langle\Sigma\rangle$ and $P \subseteq paths(e)$. Let Δ' be a ranked alphabet with $\Delta \subseteq \Delta'$. The *derivation relation of* M *with respect to* e *and* P, denoted by $\Rightarrow_{M,e,P}$, is a binary relation on $SF(F, paths(e), \Delta')$ defined as follows:
For every $t_1, t_2 \in SF(F, paths(e), \Delta')$, $\quad t_1 \Rightarrow_{M,e,P} t_2$, $\quad$ iff

- there is a $t' \in SF(F, paths(e), \Delta' \cup \{u\})$ in which the 0–ary symbol $u \notin F \cup \Delta'$ occurs exactly once,
- there is an $n \in \mathbb{N}$ and a function $f \in F^{(n+1)}$,
- there is a path $p \in P$ with $label(e, p) = \sigma$ for some $\sigma \in \Sigma^{(k)}$ with $k \in \mathbb{N}$,
- there is a rule $f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n) \to \rho$ in R, and
- there are $t'_1, \ldots, t'_n \in SF(F, paths(e), \Delta')$, such that
- $t_1 = t'[u/f(p, t'_1, \ldots, t'_n)]$ and
- $t_2 = t'[u/\rho[x_i/pi \; ; \; i \in [k]][y_i/t'_i \; ; \; i \in [n]]]$. □

If M or e are known from the context, then we drop the corresponding indices from $\Rightarrow$. If $P = paths(e)$, then we drop P.

According to [8], total deterministic mtts compute total functions from input trees to output trees. This is due to the fact that their derivation relations are confluent and noetherian and thus every sentential form has a unique normal form. This is the basis for the definition of the tree transformation.

Definition 3.5 Let $M = (F, \Delta, \Sigma, f_{in}, R)$ be an mtt. The *tree transformation computed by* M, denoted by $\tau(M)$, is the total function of type $T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ defined as follows. For every $e \in T\langle\Sigma\rangle$, $\tau(M)(e) = nf(\Rightarrow_{M,e}, f_{in}(\varepsilon))$. □

We only state here that $\tau(M)(e) = nf(\Rightarrow'_M, f_{in}(e))$, where $\Rightarrow'_M$ is the usual derivation relation induced by M working on sentential forms which contain subtrees of e.

For an input tree e and for a path p in e we need the set $funset(e,p)$ of those functions $f \in F$, for which there is a function occurrence $(f,p) \in fun(e)$ in a sentential form in every derivation $f_{in}(\varepsilon) \Rightarrow_e^* nf(\Rightarrow_e, f_{in}(\varepsilon))$. Therefore we calculate the normal form nf of $f_{in}(\varepsilon)$ with respect to $P(e, \varepsilon - p)$, i.e. the derivation is blocked at every function occurrence (f,p). Then we define $funset(e,p)$ as set of those functions f, for which there is a function occurrence (f,p) in nf.

Definition 3.6 Let $M = (F, \Delta, \Sigma, f_{in}, R)$ be an mtt. Let $e \in T\langle\Sigma\rangle$ and $p \in paths(e)$. Then we define the *function–set* $funset(e,p) \subseteq F$ *of* e *and* p by

$$funset(e,p) = set_p(nf(\Rightarrow_{e,P(e,\varepsilon-p)}, f_{in}(\varepsilon))),$$

where, for every $t \in SF(F, \{p\}, \Delta)$, $set_p(t)$ is the set of those functions f, such that there is a function occurrence (f,p) in t. □

Example 3.7 Let M_1 be the mtt of Example 3.2 and let $e = \sigma(\sigma(\alpha,\alpha),\alpha)$.

$$\begin{aligned} \text{Then, } funset(e,11) &= set_{11}(nf(\Rightarrow_{e,P(e,\varepsilon-11)}, f_{in,1}(\varepsilon))) \\ &= set_{11}(T(T(f(11,LLE), BRLE, g(11,AE)), \\ &\qquad BRE, g(11,AAAE))) \quad = \{f,g\}. \end{aligned}$$ □

The following definition introduces restrictions for mtts:

Definition 3.8 Let M be an mtt. M is

- *producing*, if the right–hand side of every rule contains at least one output symbol,
- *nondeleting*, if every variable which occurs in the left–hand side of a rule, occurs at least once in the corresponding right–hand side, and
- *noncopying*, if every variable which occurs in the left–hand side of a rule and which represents a context parameter, occurs at most once in the corresponding right–hand side. □

In the rest of this paper we always mean producing and nondeleting mtts, when we talk about mtts. The noncopying–condition will be only used to prove a certain detail of the pumping lemma and is no restriction for our application.

If $\tau(M)$ is a tree transformation computed by an mtt M, then we simply write $L_{out}(M)$ instead of $L_{out}(\tau(M))$ and we call $L_{out}(M)$ the *output language of* M. We denote the *class of output languages* of mtts by MT_{out}.

4 Pumping Lemma for Macro Tree Transducers

Let $M = (F, \Delta, \Sigma, f_{in}, R)$ be an mtt. If we choose a sufficiently large output tree $t \in L_{out}(M)$, then every input tree e with $\tau(M)(e) = t$ is high enough, such that it has a path p from the root of e to a leaf node, on which two different nodes x_1 and x_2 can be found such that

- there exist strings p_1, p_2, and p_3 such that $p = p_1p_2p_3$,
- $x_1 = p_1$ and $x_2 = p_1p_2$, and
- the function–sets $funset(e, p_1)$ and $funset(e, p_1p_2)$ are equal.

In Figure 1 an input tree e, the path $p = p_1p_2p_3$, and the nodes x_1 and x_2 are indicated. We assume here $funset(e, p_1) = \{f_1, f_2\}$. The function occurrences (f_1, p_1) and (f_2, p_1) (and (f_1, p_1p_2) and (f_2, p_1p_2)) are represented by nodes on the horizontal line through x_1 (and x_2, respectively), which are simply labeled by the functions f_1 and f_2. The function occurrence (f_{in}, ε) is represented by a node on the line through the root of e, which is simply labeled by f_{in}.

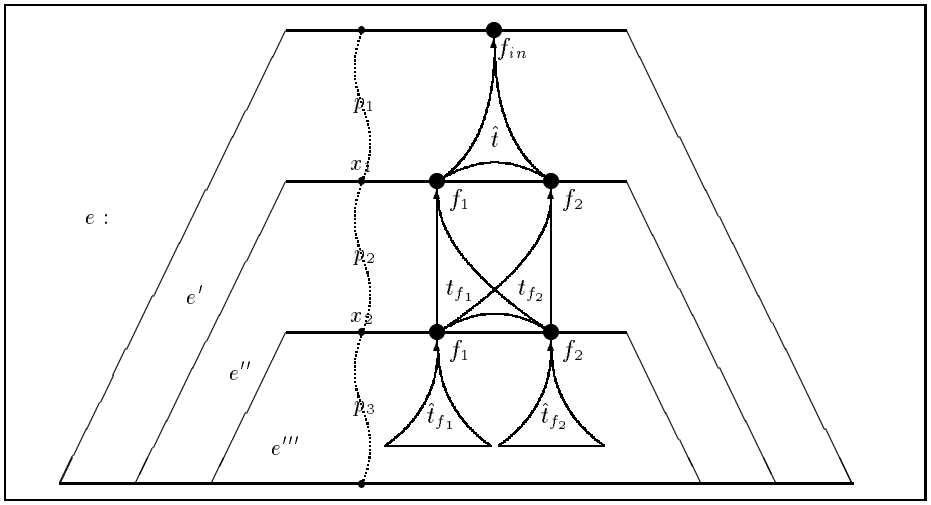

Fig. 1. Input tree e with input patterns and induced output patterns.

The nodes x_1 and x_2 define a decomposition of e into three *input patterns*:

- e' is the tree e without the subtree which has x_1 as root.
- e'' is the tree which has x_1 as root without the subtree which has x_2 as root.
- e''' is the tree which has x_2 as root.

This decomposition of e induces a decomposition of t into a certain *output pattern* $\hat{t}$ and certain *output patterns* t_f and $\hat{t}_f$ for every function $f \in funset(e, p_1)$:

- The tree $\hat{t}$ corresponds to the normal form of $f_{in}(\varepsilon)$ that is calculated only on the nodes of e'.
- For every $f \in funset(e, p_1)$ with $rank(f) = n + 1 \geq 1$, the tree t_f (and $\hat{t}_f$) corresponds to the normal form of $f(p_1, y_1, \ldots, y_n)$ (and $f(p_1p_2, y_1, \ldots, y_n)$, respectively) that is calculated only on the nodes of e'' (and e''', respectively).

Thus, the pattern $\hat{t}$ (and, for every $f \in funset(e, p_1)$, the pattern t_f) can contain subtrees of the form $f'(p_1, t_1, \ldots, t_m)$ (and $f'(p_1p_2, t_1, \ldots, t_m)$, respectively), where $f' \in funset(e, p_1)$ has the rank $m + 1 \geq 1$. Since later we shall define the pumping process in such a way that it only needs the knowledge about the output patterns, the path–information p_1 (and p_1p_2, respectively) in such subtrees, which refers to the input tree e, is no more suitable. Thus we transform $\hat{t}$ (and, for every $f \in funset(e, p_1)$, the pattern t_f) by substituting subtrees of the form $f'(p_1, t_1, \ldots, t_m)$ and $f'(p_1p_2, t_1, \ldots, t_m)$ by $f'(t_1, \ldots, t_m)$, where f' is a new symbol with rank m. For the sake of understanding we choose exactly the functions as names for the new symbols. The lost path–information is compensated by the pumping process.

In Figure 1 these output patterns are indicated. The root of every output pattern is represented by an arrow. Besides output symbols, also the new symbols f_1 and f_2 can occur in $\hat{t}$ and in t_f for every $f \in \{f_1, f_2\}$. Figure 1 represents this fact by integrating f_1 and f_2 at x_1 (and x_2) into $\hat{t}$ (and t_{f_1} and t_{f_2}, respectively). The figure suggests that these symbols appear only as leaf nodes of the patterns, but clearly, in general this is not true, because the underlying functions can have context parameters. However, our intention was to simplify the representation. Additionally, context parameters can occur in t_f and $\hat{t}_f$ for every $f \in \{f_1, f_2\}$. Also this fact is not expressed in the figure.

If we construct new input trees by repeating the input pattern e'' arbitrarily often, we can get new output trees by translating the new input trees. All of them are by definition elements of $L_{out}(M)$. Since $funset(e, p_1) = funset(e, p_1p_2)$, also the input patterns of the new input trees are compatible in the sense that there is the same function–set on, roughly speaking, every level of the input tree. Thus, for every $f \in funset(e, p_1)$, the output pattern t_f must be used for every repetition of e'' to obtain the new output tree.

For every $r \geq 0$, we denote the input tree, which has r repetitions of e'', by e_r and $nf(\Rightarrow_{e_r}, f_{in}(\varepsilon))$ by $\underline{tree}(r)$. For the calculation of an output tree $\underline{tree}(r)$ in the pumping lemma, we shall define a recursive function $\underline{tree}_1$ (and $\underline{tree}_2$), which performs the construction of $\underline{tree}(r)$ by working on a certain repetition of e'' (and on e''', respectively) and by using the output patterns as follows:

We start with the pattern $\hat{t}$, which represents the calculation on e'. Then we apply r times the function $\underline{tree}_1$ to the current intermediate result. Roughly speaking, every application of $\underline{tree}_1$ replaces every occurrence of a symbol f in a current intermediate result by the output pattern t_f, which represents a calculation on a repetition of e'', and substitutes every occurrence of a variable y_i in t_f by the recursive application of $\underline{tree}_1$ on the subtree rooting at the i–th child of the occurrence of f. Every application of $\underline{tree}_1$ yields a new intermediate result. Finally, we apply the function $\underline{tree}_2$ to the last intermediate result. Roughly speaking, the application of $\underline{tree}_2$ replaces every occurrence of a symbol f in this intermediate result by the output pattern $\hat{t}_f$, which represents a calculation on e''', and substitutes every occurrence of a variable y_i in $\hat{t}_f$ by the recursive application of $\underline{tree}_2$ on the subtree rooting at the i–th child of the occurrence of f. This application of $\underline{tree}_2$ yields the output tree $\underline{tree}(r)$.

Example 4.1 Let M_1 be the mtt defined in Example 3.2. We consider the output tree $t = nf(\Rightarrow_e, f_{in}(\varepsilon))$, where $e = \sigma(\sigma(\alpha, \alpha), \alpha)$ is the input tree.

We choose the path $p = 11$ in e with its subpaths $p_1 = 1$, $p_2 = 1$, and $p_3 = \varepsilon$. Thus we consider the two nodes $x_1 = p_1 = 1$ and $x_2 = p_1p_2 = 11$. Note that $funset(e, 1) = \{f, g\} = funset(e, 11)$. If we substitute the pruned subtrees of e by a new symbol w, then we obtain the input patterns $e' = \sigma(w, \alpha)$, $e'' = \sigma(w, \alpha)$, and $e''' = \alpha$ with the nodes reached by the paths ε, 1, and 11, respectively, of e as roots. In fact, we have $e = e'[w/e''[w/e''']]$.

Now we obtain the patterns in the output tree t as follows: we calculate

- the tree $\hat{t}$ as normal form of $f_{in}(\varepsilon)$, where the derivation relation is restricted to the paths ε and 2 of e,
- the tree t_f (and t_g) as normal form of $f(1, y_1)$ (and $g(1, y_1)$, respectively), where the derivation relation is restricted to the paths 1 and 12 of e,
- the tree $\hat{t}_f$ (and $\hat{t}_g$) as normal form of $f(11, y_1)$ (and $g(11, y_1)$, respectively), where the derivation relation is restricted to the path 11 of e,

and replace subtrees of the form $f(1, t')$ and $f(11, t')$ (and $g(1, t')$ and $g(11, t')$) by $f(t')$ (and $g(t')$), where f (and g, respectively) is a new symbol with rank 1:

$$
\begin{aligned}
\hat{t} &= nf(\Rightarrow_{e,\{\varepsilon,2\}}, f_{in}(\varepsilon))[f(1, LE)/fLE,\ g(1, AE)/gAE] \\
&= T(f(1, LE), BRE, g(1, AE))[f(1, LE)/fLE,\ g(1, AE)/gAE] \\
&= T(fLE, BRE, gAE), \\
t_f &= nf(\Rightarrow_{e,\{1,12\}}, f(1, y_1))[f(11, Ly_1)/fLy_1,\ g(11, AE)/gAE] \\
&= T(f(11, Ly_1), BRy_1, g(11, AE))[f(11, Ly_1)/fLy_1,\ g(11, AE)/gAE] \\
&= T(fLy_1, BRy_1, gAE), \\
t_g &= nf(\Rightarrow_{e,\{1,12\}}, g(1, y_1))[g(11, AAy_1)/gAAy_1] \\
&= g(11, AAy_1)[g(11, AAy_1)/gAAy_1] = gAAy_1, \\
\hat{t}_f &= nf(\Rightarrow_{e,\{11\}}, f(11, y_1)) = By_1, \\
\hat{t}_g &= nf(\Rightarrow_{e,\{11\}}, g(11, y_1)) = Ay_1.
\end{aligned}
$$

Since in this example the function symbols do not occur nested in the above normal forms, we are able to present the substitutions in this easy form. In general, the function symbols occur nested and we have to define a recursive transformation function which realizes the substitutions.

Figure 2 shows the output patterns defined above by framing those parts which only consist of output symbols.

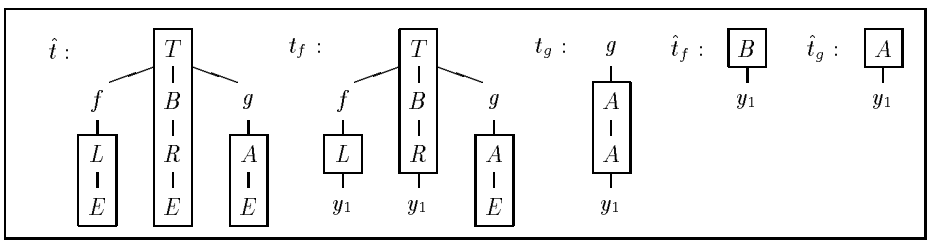

Fig. 2. Output patterns of t.

Now we show the pumping process in the case $r = 1$. Thus we consider the input tree $e = e_1 = e'[w/e''[w/e''']]$. Though just in this case there is no "real" pumping, we have chosen it, because here the whole mechanism can already be studied and the calculation does not become too big. We calculate the corresponding output tree $t = \underline{tree}(1)$ with the help of $\underline{tree}_1$ and $\underline{tree}_2$ as follows:

$$
\begin{aligned}
&\underline{tree}(1)\\
&= \underline{tree}_2(\underline{tree}_1(T(fLE, BRE, gAE)))\\
&= \underline{tree}_2(T(\underline{tree}_1(fLE), BRE, \underline{tree}_1(gAE)))\\
&= \underline{tree}_2(T(T(fLy_1, BRy_1, gAE)[y_1/\underline{tree}_1(LE)], BRE, gAAy_1[y_1/\underline{tree}_1(AE)]))\\
&= \underline{tree}_2(T(T(fLLE, BRLE, gAE), BRE, gAAAE))\\
&= T(T(\underline{tree}_2(fLLE), BRLE, \underline{tree}_2(gAE)), BRE, \underline{tree}_2(gAAAE))\\
&= T(T(By_1[y_1/\underline{tree}_2(LLE)], BRLE, Ay_1[y_1/\underline{tree}_2(AE)]), BRE,\\
&\quad Ay_1[y_1/\underline{tree}_2(AAAE)])\\
&= T(T(BLLE, BRLE, AAE), BRE, AAAAE)
\end{aligned}
$$

In Figure 3 we show the output tree $\underline{tree}(1)$ and its decomposition into (the different parts of) the output patterns. Every part of an output pattern is labeled with the input pattern which causes it. We also show the input tree e_1 and its decomposition into input patterns. □

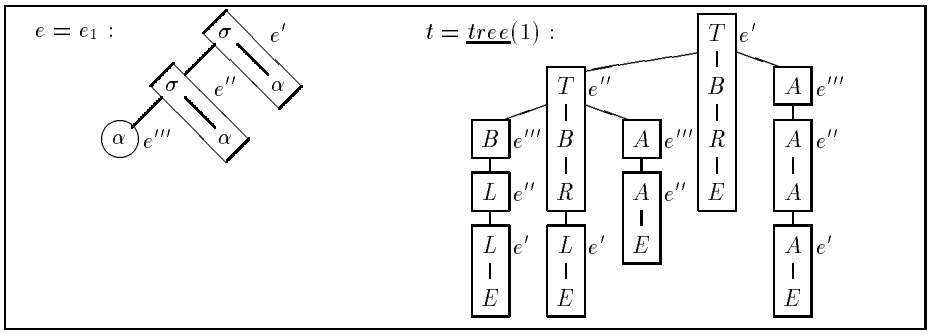

Fig. 3. Decompositions of e_1 and $\underline{tree}(1)$.

For an mtt $M = (F, \Delta, \Sigma, f_{in}, R)$ and an input tree $e \in T\langle\Sigma\rangle$ we need a certain height h_e of e, such that we have a repetition of a function–set on a path. Since there are $2^{card(F)} - 1$ possibilities for nonempty subsets of F, we need at least $h_e = 2^{card(F)}$. This height can be guaranteed, if the height of $t = nf(\Rightarrow_e, f_{in}(\varepsilon))$ is at least $h_t = (maxheight_M)^{h_e}$, where $maxheight_M$ is the maximum height of right–hand sides of rules of M. This result is due to Theorem 3.24 of [8] which relates the heights of input trees and corresponding output trees. Thus, if we denote the maximum rank of output symbols by $maxrank_M$, then the size $\sum_{i=0}^{h_t-1}(maxrank_M)^i$ of t is sufficient for the desired repetition of a function–set. This number is a first approximation of the *pumping index of* M, which is denoted by pi_M. In fact, pi_M has to be even greater to guarantee all statements of the following pumping lemma. In [14] the reader can find a more detailed discussion of the pumping index.

Theorem 4.2 (Pumping Lemma)
Let $M = (F, \Delta, \Sigma, f_{in}, R)$ be a (producing and nondeleting) mtt. There is a natural number $pi_M \geq 1$, such that for every $t \in L_{out}(M)$ with $size(t) \geq pi_M$

- there exists a ranked alphabet $(U, rank_U)$ with $U \subseteq F$, $card(U) \geq 1$, and $rank_U(f) = rank_F(f) - 1$ for every $f \in U$,
- there exists a tree $\hat{t} \in T\langle \Delta \cup U \rangle$ with $size_\Delta(\hat{t}) \geq 1$,
- for every $f \in U$, there exists a tree $t_f \in T\langle \Delta \cup U \rangle(Y_{rank_U(f)})$ with $1 \leq size_\Delta(t_f) < pi_M$ and $size_U(t_f) < pi_M$,
- for every $f \in U$, there exists a tree $\hat{t}_f \in T\langle \Delta \rangle(Y_{rank_U(f)})$ with $1 \leq size_\Delta(\hat{t}_f) < pi_M$,
- for every $f \in U$, the symbol f occurs in $\hat{t}$,
- for every $f \in U$, there is an $f' \in U$ such that f occurs in $t_{f'}$,
- for every $f \in U$ and $y \in Y_{rank_U(f)}$, the symbol y occurs in t_f and $\hat{t}_f$, such that

(1) $t = \underline{tree}(1)$ and
(2) for every $r \geq 0$, $\underline{tree}(r) \in L_{out}(M)$,

where the function $\underline{tree} : I\!N \longrightarrow T\langle \Delta \rangle$ is defined as follows:
For every $r \in I\!N$, $\underline{tree}(r) = \underline{tree}_2(\underline{tree}_1{}^r(\hat{t}))$, with
$\underline{tree}_1 : T\langle \Delta \cup U \rangle \longrightarrow T\langle \Delta \cup U \rangle$ and $\underline{tree}_2 : T\langle \Delta \cup U \rangle \longrightarrow T\langle \Delta \rangle$
are defined for every $n \in I\!N$ and $t_1, \ldots, t_n \in T\langle \Delta \cup U \rangle$ by:
For every $\delta \in \Delta^{(n)}$, $\underline{tree}_1(\delta(t_1, \ldots, t_n)) = \delta(\underline{tree}_1(t_1), \ldots, \underline{tree}_1(t_n))$.
For every $f \in U^{(n)}$, $\underline{tree}_1(f(t_1, \ldots, t_n)) = t_f[y_j / \underline{tree}_1(t_j) \ ; \ j \in [n]]$.
For every $\delta \in \Delta^{(n)}$, $\underline{tree}_2(\delta(t_1, \ldots, t_n)) = \delta(\underline{tree}_2(t_1), \ldots, \underline{tree}_2(t_n))$.
For every $f \in U^{(n)}$, $\underline{tree}_2(f(t_1, \ldots, t_n)) = \hat{t}_f[y_j / \underline{tree}_2(t_j) \ ; \ j \in [n]]$.

If M is additionally noncopying, then

(3) for every $r \geq 0$, $size(\underline{tree}(r+1)) > size(\underline{tree}(r))$,
(4) $size(\underline{tree}(0)) = size_\Delta(\hat{t}) + \sum_{f \in U}(size_{\{f\}}(\hat{t}) \cdot size_\Delta(\hat{t}_f))$, and
(5) $size(\underline{tree}(1)) =$
$size_\Delta(\hat{t}) + \sum_{f \in U}(size_{\{f\}}(\hat{t}) \cdot (size_\Delta(t_f) + \sum_{g \in U}(size_{\{g\}}(t_f) \cdot size_\Delta(\hat{t}_g))))$.

Proof Idea. Let $M = (F, \Delta, \Sigma, f_{in}, R)$ be an mtt. We do not prove the existence of the pumping index pi_M here. Thus, we consider $t \in L_{out}(M)$ with $size(t) \geq pi_M$ and an arbitrary input tree e with $nf(\Rightarrow_e, f_{in}(\varepsilon)) = t$, and only state that then $height(e) \geq 2^{card(F)}$ holds. We choose a longest path $p \in paths(e)$. Then its length is at least $2^{card(F)} - 1$. Since there are exactly $2^{card(F)} - 1$ possibilities for nonempty subsets of F, we can choose strings p_1, p_2, and p_3, with $p = p_1 p_2 p_3$, $p_2 \neq \varepsilon$, and $funset(e, p_1) = funset(e, p_1 p_2) \neq \emptyset$.

We define the ranked alphabet $(U, rank_U)$ as follows: $U = funset(e, p_1)$ and $rank_U(f) = rank_F(f) - 1$ for every $f \in U$. Let $w \notin \Sigma$ with $rank(w) = 0$. We define trees $e', e'' \in T\langle \Sigma \cup \{w\} \rangle$, where both, e' and e'', have exactly one occurrence of w, and $e''' \in T\langle \Sigma \rangle$ with the help of p_1, p_2, and p_3 as follows:

$$e' = e[p_1 \leftarrow w], \quad e'' = subtree(e[p_1 p_2 \leftarrow w], p_1), \quad \text{and} \quad e''' = subtree(e, p_1 p_2).$$

To calculate the output patterns, we restrict the derivation relation of M to the sets $P(e, \varepsilon - p_1)$, $P(e, p_1 - p_1p_2)$, and $P(e, p_1p_2)$, which, roughly speaking, lead from the root of e to the nodes in the three parts e', e'', and e''', respectively. Further we define a function $\underline{trans}$ that transforms sentential forms containing variables of Y_m with $m = max(\{rank_F(f) - 1 \mid f \in F\})$, which are viewed as additional output symbols with rank 0, into trees which are built up from output symbols, symbols of the set U, and variables of Y_m:

$\underline{trans} : SF(F, \{p_1{p_2}^l \mid l \geq 0\}, \Delta \cup Y_m) \longrightarrow T\langle \Delta \cup U\rangle(Y_m)$ is defined by:
For every $n \in I\!N, \delta \in \Delta^{(n)}, t_1, \ldots, t_n \in SF(F, \{p_1{p_2}^l \mid l \geq 0\}, \Delta \cup Y_m)$,
$\quad \underline{trans}(\delta(t_1, \ldots, t_n)) = \delta(\underline{trans}(t_1), \ldots, \underline{trans}(t_n))$.
For every $y \in Y_m$,
$\quad \underline{trans}(y) = y$.
For every $n \in I\!N, f \in F^{(n+1)}, l \in I\!N, t_1, \ldots, t_n \in SF(F, \{p_1{p_2}^l \mid l \geq 0\}, \Delta \cup Y_m)$,
$\quad \underline{trans}(f(p_1{p_2}^l, t_1, \ldots, t_n)) = f(\underline{trans}(t_1), \ldots, \underline{trans}(t_n))$.

With the help of $\underline{trans}$ we can define the desired output patterns as follows:

$\hat{t} \;\; = \underline{trans}(nf(\Rightarrow_{e, P(e, \varepsilon - p_1)}, f_{in}(\varepsilon)))$.
For every $n \in I\!N$ and $f \in U^{(n)}$,
$\quad t_f = \underline{trans}(nf(\Rightarrow_{e, P(e, p_1 - p_1p_2)}, f(p_1, y_1, \ldots, y_n)))$ and
$\quad \hat{t}_f = nf(\Rightarrow_{e, P(e, p_1p_2)}, f(p_1p_2, y_1, \ldots, y_n))$.

We do not prove here the size conditions for the output patterns and the conditions whether certain symbols occur in certain output patterns.

We abbreviate the input tree $e'[w/\underbrace{e''[w/\ldots e''[w/}_{r \text{ times}} e'''\ \underbrace{]\ldots]}_{r \text{ times}}\ ]$ by e_r.

To prove the Statements (1) and (2), we have to verify the following Statement (a) (and (b)), which relates the calculation on the l–th repetition of e'' (on the occurrence of e''') in e_r with the application of $\underline{tree}_1$ (and $\underline{tree}_2$, respectively):

(a) For every $r \geq 0,\ 1 \leq l \leq r,$ and $t \in SF(F, \{p_1{p_2}^{l-1}\}, \Delta)$,
$\quad \underline{tree}_1(\underline{trans}(t)) = \underline{trans}(nf(\Rightarrow_{e_r, P(e_r, p_1{p_2}^{l-1} - p_1{p_2}^l)}, t))$.
(b) For every $r \geq 0$ and $t \in SF(F, \{p_1{p_2}^r\}, \Delta)$,
$\quad \underline{tree}_2(\underline{trans}(t)) = nf(\Rightarrow_{e_r, P(e_r, p_1{p_2}^r)}, t)$.

Statement (a) (and Statement (b)) can be proved by structural induction on the set $SF(F, \{p_1{p_2}^{l-1}\}, \Delta)$ (and $SF(F, \{p_1{p_2}^r\}, \Delta)$, respectively). Then we can show that the following Statement (c), which relates the calculation on the topmost l occurrences of e'' in e_r with the l–fold application of the function $\underline{tree}_1$. It can be proved by finite induction on $0 \leq l \leq r$.

(c) For every $r \geq 0,\ 0 \leq l \leq r,$ and $t \in SF(F, \{p_1\}, \Delta)$,
$\quad {\underline{tree}_1}^l(\underline{trans}(t)) = \underline{trans}(nf(\Rightarrow_{e_r, P(e_r, p_1 - p_1{p_2}^l)}, t))$

Now we can prove for every $r \geq 0$ the equation $\underline{tree}(r) = \underline{tree}_2({\underline{tree}_1}^r(\hat{t})) = nf(\Rightarrow_{e_r}, f_{in}(\varepsilon))$. This equation has the following two desired consequences:

(1) $\underline{tree}(1) = nf(\Rightarrow_{e_1}, f_{in}(\varepsilon)) = nf(\Rightarrow_e, f_{in}(\varepsilon)) = t$.
(2) For every $r \geq 0$, $\underline{tree}(r) = nf(\Rightarrow_{e_r}, f_{in}(\varepsilon)) = \tau(M)(e_r) \in L_{out}(M)$.

We do not prove here the Statements (3), (4), and (5) which concern the sizes of output trees and which require the additional noncopying–condition. □

We conclude this section with observations concerning the restrictions on mtts. The producing–condition does not affect the pumping process itself. But it is necessary for the proof that the output patterns consist of at least one output symbol. This result, however, is an existential fact for the proof of the growth–property (3) of the pumping lemma and for the application in Section 5. We need the nondeleting–condition to prove that every context parameter y of a function f occurs in t_f and $\hat{t}_f$, and that the Statements (3), (4), and (5) are valid. Also the general construction in our proof depends on the nondeleting–condition, because otherwise $funset(e, p') \neq \emptyset$ cannot be guaranteed for the input tree e and for every subpath p' of the chosen path p. The noncopying–condition is only needed in the proof of the Statements (3), (4), and (5). At first sight the growth–property (3) seems to be a trivial consequence of the pumping process, because one could imagine that a growth of input trees automatically involves a growth of the corresponding output trees. But it turned out that mtts in general do not imply this growth–property and that the search for a sufficient condition has become one of the hardest items in the preparation of [14].

5 Application

It is known from Lemma 3.3 of [6] and Theorem 3.24 of [8] that, if M is an mtt, then there is a constant $c > 0$ such that, if $\tau(M)(e) = t$ for an input tree e and an output tree t, then $height(t) \leq c^{height(e)}$ holds. Thus, for example, there does not exist an mtt which calculates the tree transformation $\tau_1 : T\langle\{\sigma^{(2)}, \alpha^{(0)}\}\rangle \longrightarrow T\langle\{B^{(1)}, E^{(0)}\}\rangle$ with $\tau_1(e) = B^{2^{2 \cdot size(e)-1}} E$ for every $e \in T\langle\Sigma\rangle$, since $size(e)$ can grow exponentially in $height(e)$. We also can deduce that there does not exist an mtt which calculates the tree transformation $\tau_2 : T\langle\{\gamma^{(1)}, \alpha^{(0)}\}\rangle \longrightarrow T\langle\{B^{(1)}, E^{(0)}\}\rangle$ with $\tau_2(\gamma^n(\alpha)) = B^{2^{2^n}} E$ for every $n \geq 0$. We only mention here that there is a macro attributed tree transducer (cf. Theorem 4.2 of [16]) which calculates τ_1 and that even no macro attributed tree transducer can realize τ_2.

Now the question arises, whether an arbitrary ranked alphabet Σ and an mtt M, which computes a tree transformation $\tau(M) : T\langle\Sigma\rangle \longrightarrow T\langle\{B^{(1)}, E^{(0)}\}\rangle$ with $L_{out}(M) = \{B^{2^{2^n}} E \mid n \geq 0\}$, exist. The answer to this question for producing and nondeleting mtts is no, because of the following result:

Theorem 5.1 $\{B^{2^{2^n}} E \mid n \geq 0\} \notin MT_{out}$

Proof. We assume that there is an mtt N with $L_{out}(N) = \{B^{2^{2^n}} E \mid n \geq 0\}$. We only mention that, since N is nondeleting and $L_{out}(N)$ is monadic, there is also an mtt $M = (F, \Delta, \Sigma, f_{in}, R)$ with $L_{out}(M) = L_{out}(N)$ and with $F = F^{(1)} \cup F^{(2)}$ (cf. Lemma 3.18 of [14]). Thus, the right–hand side of every rule of M contains at most one occurrence of a context parameter, and therefore M

additionally is noncopying (cf. Observation 3.19 of [14]). Then, by Theorem 4.2, for every $t \in L_{out}(M)$ with $size(t) \geq pi_M$ certain properties hold. We consider the output tree $t = B^{2^{2^{(pi_M{}^2+pi_M+1)}}} E$ with $size(t) = 1 + 2^{2^{(pi_M{}^2+pi_M+1)}} \geq pi_M$.

We choose arbitrarily a nonempty subset $U \subseteq F$, a tree $\hat{t}$, and trees t_f and $\hat{t}_f$ for every $f \in U$, fulfilling the conditions of Theorem 4.2, such that $t = \underline{tree}(1)$. For every such choice we can deduce the contradiction $\underline{tree}(0) \notin L_{out}(M)$:

With Statement (5) and the size conditions of Theorem 4.2 we estimate

$$\begin{aligned}
&1 + 2^{2^{(pi_M{}^2+pi_M+1)}} = size(t) = size(\underline{tree}(1)) \\
&= size_{\Delta}(\hat{t}) + \textstyle\sum_{f \in U}(size_{\{f\}}(\hat{t}) \cdot (size_{\Delta}(t_f) + \sum_{g \in U}(size_{\{g\}}(t_f) \cdot size_{\Delta}(\hat{t}_g)))) \\
&< size_{\Delta}(\hat{t}) + \textstyle\sum_{f \in U}(size_{\{f\}}(\hat{t}) \cdot (pi_M + \sum_{g \in U}(size_{\{g\}}(t_f) \cdot pi_M))) \\
&= size_{\Delta}(\hat{t}) + \textstyle\sum_{f \in U}(size_{\{f\}}(\hat{t}) \cdot (pi_M + (pi_M \cdot size_U(t_f)))) \\
&< size_{\Delta}(\hat{t}) + \textstyle\sum_{f \in U}(size_{\{f\}}(\hat{t}) \cdot (pi_M + pi_M{}^2)) \\
&= size_{\Delta}(\hat{t}) + size_U(\hat{t}) \cdot (pi_M + pi_M{}^2).
\end{aligned}$$

Thus we have the intermediate result $size_U(\hat{t}) > \dfrac{1 + 2^{2^{(pi_M{}^2+pi_M+1)}} - size_{\Delta}(\hat{t})}{pi_M{}^2 + pi_M}$.

With Statement (4) and the size conditions of Theorem 4.2 we estimate

$$\begin{aligned}
&size(\underline{tree}(0)) \\
&= size_{\Delta}(\hat{t}) + \textstyle\sum_{f \in U}(size_{\{f\}}(\hat{t}) \cdot size_{\Delta}(\hat{t}_f)) \\
&\geq size_{\Delta}(\hat{t}) + \textstyle\sum_{f \in U}(size_{\{f\}}(\hat{t}) \cdot 1) \\
&= size_{\Delta}(\hat{t}) + size_U(\hat{t}) \\
&> size_{\Delta}(\hat{t}) + \tfrac{1+2^{2^{(pi_M{}^2+pi_M+1)}} - size_{\Delta}(\hat{t})}{pi_M{}^2+pi_M} \text{ (Intermediate result)} \\
&= \tfrac{1+2^{2^{(pi_M{}^2+pi_M+1)}}}{pi_M{}^2+pi_M} + size_{\Delta}(\hat{t}) \cdot (1 - \tfrac{1}{pi_M{}^2+pi_M}) \\
&\geq 1 + \tfrac{2^{2^{(pi_M{}^2+pi_M+1)}}}{pi_M{}^2+pi_M} + \tfrac{1-1}{pi_M{}^2+pi_M} \qquad (\text{since } size_{\Delta}(\hat{t}) \geq 1 \text{ and } 1 \geq \tfrac{1}{pi_M{}^2+pi_M}) \\
&= 1 + \tfrac{2^{(2^{(pi_M{}^2+pi_M)}+2^{(pi_M{}^2+pi_M)})}}{pi_M{}^2+pi_M} \\
&= 1 + 2^{2^{(pi_M{}^2+pi_M)}} \cdot \tfrac{2^{2^{(pi_M{}^2+pi_M)}}}{pi_M{}^2+pi_M} \\
&> 1 + 2^{2^{(pi_M{}^2+pi_M)}}. \qquad (\text{since } 2^{2^{(pi_M{}^2+pi_M)}} > pi_M{}^2 + pi_M)
\end{aligned}$$

Thus $1 + 2^{2^{(pi_M{}^2+pi_M)}} < size(\underline{tree}(0)) < size(\underline{tree}(1)) = 1 + 2^{2^{(pi_M{}^2+pi_M+1)}}$ results from Statement (3) of Theorem 4.2 and therefore $\underline{tree}(0) \notin L_{out}(M) = \{B^{2^{2^n}} E \mid n \geq 0\}$, contradicting Statement (2) of the pumping lemma. □

In [14] another kind of application can be found which leads to hierarchies of producing and nondeleting mtts with respect to their number of functions.

6 Summary and Further Research Topics

This paper has introduced a pumping lemma for output languages of producing, nondeleting, and noncopying mtts. As application we have got the result that the language $\{B^{2^{2^n}} E \mid n \geq 0\}$ which contains the monadic trees with double exponential height is no output language of a producing and nondeleting mtt.

The paper leaves several further research topics for pumping lemmata for mtts: Are there non–monadic languages which can be proved not to be output languages of mtts with the help of our pumping lemma in a justifiable expense? Are there possibilities to relax the restrictions to mtts and are the perhaps resulting pumping lemmata suitable for certain applications? Finally, a pumping lemma for macro attributed tree transducers (cf. [16]) could be combined from the pumping lemmata for attributed tree transducers (cf. [15]) and mtts.

References

1. A.V. Aho and J.D. Ullman. Translations on a context free grammar. *Inform. and Control*, 19:439–475, 1971.
2. Y. Bar–Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Z. Phonetik. Sprach. Komm.*, 14:143–172, 1961.
3. B. Courcelle and P. Franchi–Zannettacci. Attribute grammars and recursive program schemes. *Theoret. Comput. Sci.*, 17:163–191 and 235–257, 1982.
4. J. Engelfriet. Bottom–up and top–down tree transformations — a comparison. *Math. Syst. Theory*, 9:198–231, 1975.
5. J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory; perspectives and open problems*. New York, Academic Press, 1980.
6. J. Engelfriet. Tree transducers and syntax directed semantics. Technical Report Memorandum 363, Technische Hogeschool Twente, 1981.
7. J. Engelfriet, G. Rozenberg, and G. Slutzki. Tree transducers, L systems, and two–way machines. *J. Comput. Syst. Sci.*, 20:150–202, 1980.
8. J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31:71–145, 1985.
9. J. Engelfriet and H. Vogler. Modular tree transducers. *Theoret. Comput. Sci.*, 78:267–304, 1991.
10. Z. Ésik. Decidability results concerning tree transducers. *Acta Cybernetica*, 5:1–20, 1980.
11. Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.
12. Z. Fülöp, F. Herrmann, S. Vagvölgyi, and H. Vogler. Tree transducers with external functions. *Theoret. Comput. Sci.*, 108:185–236, 1993.
13. E.T. Irons. A syntax directed compiler for ALGOL 60. *Comm. Assoc. Comput. Mach.*, 4:51–55, 1961.
14. A. Kühnemann. A pumping lemma for output languages of macro tree transducers. Technical Report TUD/FI95/08, Technical University of Dresden, 1995.
15. A. Kühnemann and H. Vogler. A pumping lemma for output languages of attributed tree transducers. *Acta Cybernetica*, 11:261–305, 1994.
16. A. Kühnemann and H. Vogler. Synthesized and inherited functions – a new computational model for syntax–directed semantics. *Acta Informatica*, 31:431–477, 1994.
17. C.R. Perrault. Intercalation lemmas for tree transducer languages. *J. Comput. Syst. Sci.*, 13:246–277, 1976.
18. W.C. Rounds. Mappings and grammars on trees. *Math. Syst. Theory*, 4:257–287, 1970.
19. J.W. Thatcher. Generalized^2 sequential machine maps. *J. Comput. Syst. Sci.*, 4:339–367, 1970.

This article was processed using the LaTeX macro package with LLNCS style