Acta Cybernetica, Vol. 11, No. 4, Szeged, 1994

A Pumping Lemma for Output Languages of Attributed Tree Transducers

A. Kühnemann* H. Vogler*

Abstract

An attributed tree transducer is a formal model for studying properties of attribute grammars. In this paper we introduce and prove a pumping lemma for output languages of noncircular, producing, and visiting attributed tree transducers. We apply this pumping lemma to gain two results: (1) there is no noncircular, producing, and visiting attributed tree transducer which computes the set of all monadic trees with exponential height as output and (2) there is a hierarchy of noncircular, producing, and visiting attributed tree transducers with respect to their number of attributes.

1 Introduction

In formal language theory we are often confronted with the task to decide, whether a given language L is an element of a class $\mathcal{L}$ of languages, where $\mathcal{L}$ usually is defined by a class of grammars or translation schemes. If L is an element of $\mathcal{L}$, then we have to specify a grammar or a translation scheme which generates L. If L is not an element of $\mathcal{L}$, then sometimes we can use necessary conditions which every language in $\mathcal{L}$ has to fulfill. With the help of these conditions we can try to deduce a contradiction to the assumption that L is an element of $\mathcal{L}$. Pumping lemmata are such necessary conditions which have been proven to be very useful tools.

Pumping lemmata have been invented for different kinds of languages, for example string languages, graph and hypergraph languages, picture languages, and tree transducer languages.

*Institut für Softwaretechnik I, Fakultät Informatik, Technische Universität Dresden, D–01062 Dresden, Germany, e–mail: {ak15, hv3}@irs.inf.tu-dresden.de

In the case of string languages we can observe the following evolution of pumping lemmata: Scheinberg has used in [Sch60] a proof technique which can be seen as a predecessor of the well known pumping lemma for context–free languages of Bar–Hillel, Perles, and Shamir [BPS61]. The structure of the latter pumping lemma has served as pattern for most of the existing pumping lemmata in the literature and therefore it seems to be the root of the research about pumping lemmata. Since it also has influenced our pumping lemma, we present here a short version of the lemma's central statement and we recall its proof idea:

For every context–free grammar G there is a natural number n_G, called the pumping index of G, such that for every string z which is an element of the language $L(G)$ generated by G and which has at least the length n_G, the following holds. There is a decomposition $z = uvwxy$, such that v or x is not the empty string and such that for every natural number j, the pumped string uv^jwx^jy is an element of $L(G)$.

The proof can be sketched as follows: We choose a sufficiently long string z of $L(G)$, such that its derivation tree e 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 are labeled by the same nonterminal symbol. 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 x_1 and x_2 have the same label, we can construct for every natural number j a new derivation tree, by repeating $\bar{e}$ j times. Taking the yield of these derivation trees, we obtain new elements of $L(G)$.

As stated above, the pumping lemma of Bar–Hillel, Perles, and Shamir is only a necessary condition for the context–freeness of a string language. Thus there exist non–context–free languages which fulfill the requirements of the pumping lemma. In the sequel more and more stronger pumping lemmata for context–free string languages have been invented. Most of them, however, represent no sufficient condition for context–freeness. For example, in the Ogden–Lemma (cf. [Ogd68]) we can designate distinguished positions in the pumped string. This allows us to concentrate on those substrings, in which pumping is effective. Bader and Moura have developed in [BM82] a stronger version, the Generalized Ogden–Lemma, where additionally positions in the pumped string can be excluded. In the paper of Bader and Moura it is also shown that there is no stronger version of the Generalized Ogden–Lemma which exactly characterises the context–free string languages.

Wise has introduced in [Wis76] his Strong Pumping Lemma which is a necessary and sufficient condition for context–free string languages. The central idea of this lemma is to pump sentential forms of a grammar for a context–free language L instead of pumping terminal strings of L. The Strong Pumping Lemma of Wise represents another method to prove that a certain language is not context–free by assuming that it is context–free and by applying the lemma. In contrast to the other pumping lemmata stated above, this application guarantees the existence of a contradiction, because the Strong Pumping Lemma characterises the class of context–free languages. Clearly, it depends on the skill of the researcher, whether he can construct this contradiction, yes or no.

There also exist pumping lemmata for subclasses of the class of context–free languages: Boonyavatana and Slutzki have invented pumping lemmata for linear context- -free and nonterminal bounded string languages in [BS86a] and [BS86b], respectively. Yu has developed in [Yu89] a pumping lemma for deterministic context–free languages. Ehrenfeucht, Parikh, and Rozenberg have introduced in [EPR81] the Block Pumping Lemma as characterisation of regular string languages.

There are also pumping lemmata in the area of context– free graph and hypergraph languages: Kreowski (cf. [Kre79]) and Habel (cf. [Hab89]) have invented pumping lemmata for edge–replacement and hyperedge–replacement languages, respectively. These pumping lemmata require a certain size of the pumped graphs. In comparison with them, the Maximum Path Length Pumping Lemma for edge–replacement languages of Kuske (cf. [Kus91,Kus93]) needs a certain length of a path in the pumped graphs.

Another kind of language paradigm are the picture languages. Hinz has developed in [Hin90] pumping lemmata for certain subclasses of picture languages.

First Aho and Ullman have inspected pumping lemmata for output languages of translation schemes in [AU71], namely for generalized syntax directed translations. Perrault and Ésik have introduced in [Per76] and [Ési80], respectively, pumping lemmata for (nondeterministic) top–down tree transducers (cf. [Rou70, Tha70,Eng75]). The results of Ésik also appear in the book of Gécseg and Steinby (cf. [GS83]). Engelfriet, Rozenberg, and Slutzki have presented in [ERS80] a pumping lemma for deterministic top–down tree–to–string transducers which has a structure that is closely related to the pumping lemma for context–free string languages. The proof of this lemma had a big influence on the development of the pumping lemma for attributed tree transducers which we present in this paper.

The concept of attributed tree transducer has been invented by Fülöp in [Fül81]; it is a formal model for studying properties of attribute grammars introduced by Knuth in [Knu68]. Attributed tree transducers are abstractions of attribute grammars in the sense that they take trees over an arbitrary ranked alphabet of input symbols rather than derivation trees as argument, and that the values of the attributes are also trees over a ranked alphabet of output symbols.

Like in attribute grammars, the set of attributes is partitioned into the set of synthesized and inherited attributes which are associated to the input symbols and which compute their values in a bottom–up manner and in a top–down manner, respectively. In contrast to attribute grammars, to every input symbol the whole set of attributes is associated; this means that all attributes are available at any node of any input tree. Roughly speaking, computing the value of a synthesized attribute occurrence of a node x of an input tree, the values of the inherited attribute occurrences of x and of the synthesized attribute occurrences of its sons (if they exist) may be used and, computing the value of an inherited attribute occurrence of x, the values of the inherited attribute occurrences of its father (if it exists) and of the synthesized attribute occurrences of x and of its brothers may be used. This refers to the usual Bochmann Normal Form of attribute grammars [Boc76].

In this paper we consider only total deterministic attributed tree transducers: For every node x of an input tree which is labeled by a particular input symbol

and for every synthesized attribute s, the computation of the attribute occurrence of s at x is fixed by exactly one rule. Similarly, for every node x which is labeled by a particular input symbol and for every inherited attribute i, the computation of the attribute occurrence of i at the j–th son of x is fixed by exactly one rule.

As in attribute grammars, these dependencies can induce circularities among the attribute occurrences of an input tree. We restrict the attributed tree transducers to be noncircular and we designate a synthesized attribute as initial attribute. Thus we designate an initial attribute occurrence at the root of every input tree of which the value will be the output tree. Then every attributed tree transducer M computes a total function from input trees to output trees. This function is called the tree transformation of M. The output language of an attributed tree transducer M is defined as the range of the tree transformation of M.

As stated at the beginning of the introduction, pumping lemmata can help us to prove that a certain language is not an element of a class of languages. But not only pumping lemmata have been used to solve such a kind of problem: Fülöp and Vágvölgyi have shown in [FV91] by means of a direct proof that a particular tree transformation (which is induced by a bottom–up tree transducer; cf. [Eng75]) cannot be computed by an attributed tree transducer. Maybe the proof of Fülöp and Vágvölgyi can be generalized to a proof of a kind of pumping lemma. But we do not follow here this line of generalization and return to the development of a pumping lemma for a particular class of attributed tree transducers.

We restrict our pumping lemma to special attributed tree transducers, namely producing and visiting (and noncircular) attributed tree transducers. An attributed tree transducer is producing, if every rule application delivers at least one new output symbol. An attributed tree transducer is visiting, if for every input tree and for every node x of it, the value of at least one attribute occurrence of x is needed to compute the value of the initial synthesized attribute occurrence at the root.

The main idea of our pumping lemma for output languages of producing and visiting attributed tree transducers is adopted from the proof of the pumping lemma for context–free string languages that was outlined at the beginning of this introduction. In the case of context–free string languages we have to inspect a derivation tree of a sufficiently long string to deduce new pumped strings. Here we have to consider input trees belonging to a sufficiently large output tree to obtain new pumped output trees: For every producing and visiting attributed tree transducer M, a natural number n_M, called the pumping index of M, can be constructed. If we choose an output tree t from the output language of M which has at least n_M nodes, 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 attribute occurrences that are needed to calculate the initial attribute occurrence at the root of e. 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 attribute occurrences, we can construct new input trees by repeat-

ing $\bar{e}$ arbitrarily many times. Translating these input trees by M, we obtain new elements of the output language of M.

The proof is based on the observation that the decomposition of the input tree e induces a decomposition of the output tree t into output patterns and that these patterns are used to construct the new output trees. Thus the pumping process itself can be described by using only the output patterns. Therefore the applications of the pumping lemma are completely independend of the underlying input trees.

In this paper we apply our pumping lemma to prove the following two results:

- There is no noncircular, producing, and visiting attributed tree transducer which computes the set of all monadic trees with exponential height as output.
- There is a hierarchy of noncircular, producing, and visiting attributed tree transducers with respect to their number of attributes.

This paper is divided into five sections, from which this one is the first. In Section 2 we fix all the notions and notations, especially about attributed tree transducers, which are necessary for the remaining sections. Section 3 contains the pumping lemma together with its proof. In Section 4 we show the two applications of the pumping lemma. Finally, in Section 5 the reader can find a short summary and a presentation of further research topics.

2 Preliminaries

In this section we collect the notations, notions, and definitions which are used throughout this paper. Most of the definitions are taken from [KV94], some of them with a slight modification.

2.1 General notations

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 $[m]$, thus $[0]$ denotes the empty set $\emptyset$. The empty word is denoted by ε. For an arbitrary set S, the cardinality of S is denoted by $card(S)$ and the set of all subsets of S is denoted by $\mathcal{P}(S)$. If S is a subset of $\mathbb{N}$, then $max(S)$ denotes the maximum of S; $max(\emptyset)$ is defined as 0. A relation $f \subseteq A \times B$ is a *partial function*, if for every $(a, b_1) \in f$ and $(a, b_2) \in f$, the elements b_1 and b_2 are equal. Such a partial function is denoted by $f : A \dashrightarrow B$.

If A is an alphabet, then A^* denotes the set of words over A. 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 $v[u_i/v_i \,;\, i \in [n]]$. $|p|$ denotes the length of a string p over an alphabet which should be known from the context. If P_1 and P_2 are two sets of strings, then $P_1 \cdot P_2 := \{p_1 p_2 \mid p_1 \in P_1, p_2 \in P_2\}$.

Let $\Rightarrow$ be a binary relation on some set T. Then, $\Rightarrow^*$ and $\Rightarrow^+$ denote the transitive, reflexive closure of $\Rightarrow$ and the transitive closure of $\Rightarrow$, respectively. Let $n \in \mathbb{N} - \{0\}$. If $t_j \in T$ for every $j \in [n+1]$ and if $t_j \Rightarrow t_{j+1}$ for every $j \in [n]$, then the sequence $t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow t_{n+1}$ is called a *derivation*. If only the first element t_1 and the last element t_{n+1} of a derivation are important, we also use the notation $t_1 \Rightarrow^+ t_{n+1}$. Note that there can exist more than one derivation $t_1 \Rightarrow^+ t_{n+1}$. 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* $\Rightarrow$. In general t can have either no or one or more than one normal form. If the normal form of t exists and if it is unique, then it is denoted by $nf(\Rightarrow, t)$. The relation $\Rightarrow$ is *confluent*, if for every $t, t_1, t_2 \in T$ with $t \Rightarrow^* t_1$ and $t \Rightarrow^* t_2$, there is an $t' \in T$ such that $t_1 \Rightarrow^* t'$ and $t_2 \Rightarrow^* t'$. It is *noetherian* or *terminating*, if there is no infinite derivation of $\Rightarrow$. If $\Rightarrow$ is noetherian and confluent, then for every $t \in T$, the normal form of t exists and it is unique.

2.2 Ranked alphabets, trees, and tree transformations

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 a natural number 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 $\sigma^{(n)}$ and $rank(\sigma) = n$. 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 $\Sigma^{(n)}$.

For a ranked alphabet Σ, the set of *trees over* Σ, denoted by $T\langle\Sigma\rangle$, is the smallest subset $T \subseteq (\Sigma \cup \{(,),,\})^*$ such that for every $\sigma \in \Sigma^{(n)}$ with $n \geq 0$ and $t_1, \ldots, t_n \in T$, the string $\sigma(t_1, \ldots, t_n) \in T$. For a symbol $\sigma \in \Sigma^{(0)}$ we simply write σ instead of $\sigma()$.

The following functions are defined inductively on the structure of trees in $T\langle\Sigma\rangle$ (here, the induction base is a special case of the induction step):

- $height : T\langle\Sigma\rangle \longrightarrow \mathbb{N}$ delivers the *height of a tree* $t \in T\langle\Sigma\rangle$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $height(\sigma(t_1, \ldots, t_n)) = 1 + max(\{height(t_i) \mid i \in [n]\})$.

- $size_{\Sigma'} : T\langle\Sigma\rangle \longrightarrow \mathbb{N}$ delivers the *size of a tree* $t \in T\langle\Sigma\rangle$ *with respect to a subset* $\Sigma' \subseteq \Sigma$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $size_{\Sigma'}(\sigma(t_1, \ldots, t_n)) = 1 + \sum_{i \in [n]} size_{\Sigma'}(t_i)$, if $\sigma \in \Sigma'$,
  $size_{\Sigma'}(\sigma(t_1, \ldots, t_n)) = \sum_{i \in [n]} size_{\Sigma'}(t_i)$, if $\sigma \notin \Sigma'$.
  If $\Sigma' = \Sigma$, then we abbreviate $size_{\Sigma'}$ by $size$.

- $paths : T\langle\Sigma\rangle \longrightarrow \mathcal{P}(\mathbb{N}^*)$ delivers the *set of paths of a tree* $t \in T\langle\Sigma\rangle$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $paths(\sigma(t_1, \ldots, t_n)) = \{\varepsilon\} \cup \{p \mid p = ip', i \in [n], p' \in paths(t_i)\}$.

- *label* : $T\langle\Sigma\rangle \times \mathbb{N}^* - \rightarrow \Sigma$ delivers the *label of the node of a tree* $t \in T\langle\Sigma\rangle$ *reached by a path* $p \in paths(t)$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $label(\sigma(t_1, \ldots, t_n), p) = \sigma$, if $p = \varepsilon$,
  $label(\sigma(t_1, \ldots, t_n), p) = label(t_i, p')$, if $p = ip'$ for some $i \in [n]$.

- *subtree* : $T\langle\Sigma\rangle \times \mathbb{N}^* - \rightarrow T\langle\Sigma\rangle$ delivers the *subtree of a tree* $t \in T\langle\Sigma\rangle$ *reached by a path* $p \in paths(t)$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $subtree(\sigma(t_1, \ldots, t_n), p) = \sigma(t_1, \ldots, t_n)$, if $p = \varepsilon$,
  $subtree(\sigma(t_1, \ldots, t_n), p) = subtree(t_i, p')$, if $p = ip'$ for some $i \in [n]$.

- *repl* : $T\langle\Sigma\rangle \times \mathbb{N}^* \times T\langle\Sigma\rangle - \rightarrow T\langle\Sigma\rangle$ delivers the *tree obtained from a tree* $t \in T\langle\Sigma\rangle$ *by replacing the subtree reached by a path* $p \in paths(t)$, *by another tree* $t' \in T\langle\Sigma\rangle$.
  If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 0$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then
  $repl(\sigma(t_1, \ldots, t_n), p, t') = t'$, if $p = \varepsilon$,
  $repl(\sigma(t_1, \ldots, t_n), p, t') = \sigma(t_1, \ldots, repl(t_i, p', t'), \ldots, t_n)$, if $p = ip'$ for some $i \in [n]$.
  In the following we use the more convenient notation $t[p \leftarrow t']$ instead of $repl(t, p, t')$.

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

Let Σ be a ranked alphabet, $t \in T\langle\Sigma\rangle$, and let U be another ranked alphabet with $rank(u) = 0$ for every $u \in U$ and with $U \cap \Sigma = \emptyset$. A tree $t' \in T\langle\Sigma \cup U\rangle$ is called a *pattern in* $t \in T\langle\Sigma\rangle$, if there is a symbol $v \notin \Sigma$ with $rank(v) = 0$, there is a tree $t'' \in T\langle\Sigma \cup \{v\}\rangle$, and for every $u \in U$ there is a tree $t_u \in T\langle\Sigma\rangle$, such that $t = t''[v/t'[u/t_u \ ; \ u \in U]]$.

A *tree transformation* is a total function $\tau : T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ where Σ and Δ are ranked alphabets.

2.3 Attributed Tree Transducers

In this subsection we define the syntax of so called *si*–tree transducers and the derivation relations which are induced by them. In [Gie88] *si*–tree transducers are called full attributed tree transducers. Though *si*–tree transducers are an extension of attributed tree transducers in the sense of [Fül81], we also use simply the notion attributed tree transducer for an *si*–tree transducer. If we restrict the transducers to be noncircular, then their derivation relations are confluent and noetherian, and every noncircular transducer computes a tree transformation.

A system of attributes is the first component in the definition of an attributed tree transducer M. We specify a ranked input alphabet Σ. Then, intuitively, M takes an argument e where e is a tree over Σ, called *input tree*, on which the evaluation of attribute values is performed. An *output tree* is built up over a ranked alphabet Δ of working symbols. The derivations of M will start with an initial

synthesized attribute s_{in} and with an extra marker *root* on top of the input tree where *root* is a new symbol of rank 1. If e is an input tree, then in anology to [KV94] we call the tree $\tilde{e} = root(e)$ the *control tree*, because it controls the derivation of the transducer (cf. Figure 1). The role of the marker *root* is explained after defining the derivation relation. Of course, the kernel of the definition of an attributed tree transducer is the finite set of rewrite rules. The possible right-hand sides of rules are fixed at the end of the definition.

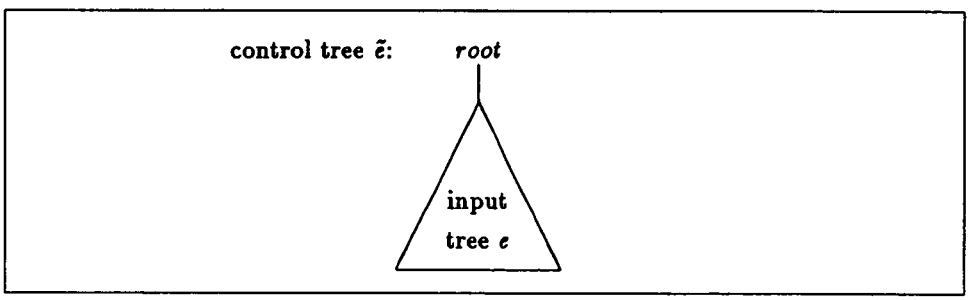


Figure 1: The input tree e and the control tree $\tilde{e}$.

We mention already here that, similarly to top–down tree transducers, we designate the argument position of every attribute to contain the control tree $\tilde{e}$. Additionally, in attributed tree transducers the control tree $\tilde{e}$ is associated with a path through $\tilde{e}$. Actually, in the argument of an attribute, only a path through $\tilde{e}$ will occur, the control tree itself will parameterize the derivation relation (cf. Definition 2.6).

Definition 2.1 An *si-tree transducer* is a tuple $(\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ where

- $\mathcal{A} = (A, A_s, A_i)$ is a system of attributes, where
  - A is a ranked alphabet of *attributes*; for every $a \in A$, $rank_A(a) = 1$.
  - $A_s \subseteq A$ and $A_i \subseteq A$ are the disjoint sets of *synthesized attributes* and *inherited attributes*, respectively, with $A = A_s \cup A_i$.
- Δ is the ranked alphabet of *working symbols* (or: *output symbols*) with $A \cap \Delta = \emptyset$.
- Σ is the ranked alphabet of *input symbols* with $A \cap \Sigma = \emptyset$.
- $s_{in} \in A_s$ is the *initial attribute*.
- *root* is a symbol of rank 1, called the *root marker*, where $root \notin A \cup \Delta \cup \Sigma$.

- $R = \bigcup_{\sigma \in \Sigma \cup \{root\}} R_\sigma$ is a finite set of *rules*, defined by Conditions 1. and 2.

1. The set R_{root} contains exactly one rule of the form

$$s_{in}(z) \to \rho$$

with $\rho \in RHS(A_s, \emptyset, \Delta, root)$.
For every $i \in A_i$, the set R_{root} contains exactly one rule of the form

$$i(z1) \to \rho$$

with $\rho \in RHS(A_s, \emptyset, \Delta, root)$.

2. For every $\sigma \in \Sigma^{(k)}$ with $k \geq 0$ and for every $s \in A_s$, the set R_σ contains exactly one rule of the form

$$s(z) \to \rho$$

with $\rho \in RHS(A_s, A_i, \Delta, \sigma)$.
For every $\sigma \in \Sigma^{(k)}$ with $k \geq 0$, for every $i \in A_i$ and for every $j \in [k]$, the set R_σ contains exactly one rule of the form

$$i(zj) \to \rho$$

with $\rho \in RHS(A_s, A_i, \Delta, \sigma)$.

For every $G_s \subseteq A_s$, $G_i \subseteq A_i$, and $\sigma \in \Sigma \cup \{root\}$ with $rank(\sigma) = k \geq 0$, the set of *σ-right-hand sides over G_s, G_i and Δ*, denoted by $RHS(G_s, G_i, \Delta, \sigma)$, is the smallest subset RHS of $(G_s \cup G_i \cup \Delta \cup [k] \cup \{z, (,), ,\})^*$ such that the following three conditions hold:

(i) For every $\delta \in \Delta^{(r)}$ with $r \geq 0$, and $\rho_1, \ldots, \rho_r \in RHS$, the tree $\delta(\rho_1, \ldots, \rho_r) \in RHS$.

(ii) For every $s \in G_s$, $j \in [k]$, the tree $s(zj) \in RHS$.

(iii) For every $i \in G_i$, the tree $i(z) \in RHS$. □

For an *si*-tree transducer $M = (A, \Delta, \Sigma, s_{in}, root, R)$, we fix the following notions and notations.

- The set $\Sigma \cup \{root\}$ is denoted by Σ_+.
- In the rules of R, the symbol z is called *path variable*.
- For every $\sigma \in \Sigma^{(k)}$, the set of *inside attribute occurrences* of σ, denoted by $in(\sigma)$, is the set $\{s(z) \mid s \in A_s\} \cup \{i(zj) \mid i \in A_i, j \in [k]\}$. The set of *inside attribute occurrences* of *root*, denoted by $in(root)$, is the set $\{s_{in}(z)\} \cup \{i(z1) \mid i \in A_i\}$. The set of *outside attribute occurrences* of σ, denoted by $out(\sigma)$, is the set $\{i(z) \mid i \in A_i\} \cup \{s(zj) \mid s \in A_s, j \in [k]\}$. The set of *outside attribute occurrences* of *root*, denoted by $out(root)$, is the set $\{s(z1) \mid s \in A_s\}$. The set of *attribute occurrences* of $\sigma \in \Sigma_+$, denoted by $att(\sigma)$, is the set $in(\sigma) \cup out(\sigma)$.

- For $a \in A$, $\sigma \in \Sigma_+^{(k)}$ and $\eta \in \{zj \mid j \in [k] \cup \{\varepsilon\}\}$, we call a rule of R_σ with the left-hand side $a(\eta)$ an (a, η, σ)-rule. The right-hand side of this rule is denoted by $rhs(a, \eta, \sigma)$. We note that only outside attribute occurrences of σ appear in $rhs(a, \eta, \sigma)$ and that for every $a(\eta) \in in(\sigma)$, there is exactly one (a, η, σ)-rule in R.

Example 2.2 We define the *si*-tree transducer $M_1 = (\mathcal{A}, \Delta, \Sigma, s, root, R)$ with:
$\Delta = \{B^{(1)}, T^{(2)}, L^{(1)}, R^{(1)}, E^{(0)}\}$,
$\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$,
$\mathcal{A} = (A, A_s, A_i)$ with $A = \{s, i\}$, $A_s = \{s\}$, and $A_i = \{i\}$, and
$R = R_{root} \cup R_\sigma \cup R_\alpha$ is the following set of rules:

$$
\begin{array}{lcll}
R_{root} = & \{s(z) & \to B(s(z1)), & (1) \\
& i(z1) & \to E \quad \} & (2) \\
R_\sigma = & \{s(z) & \to T(s(z1), s(z2)), & (3) \\
& i(z1) & \to L(i(z)), & (4) \\
& i(z2) & \to R(i(z)) \quad \} & (5) \\
R_\alpha = & \{s(z) & \to B(i(z)) \quad \} & (6)
\end{array}
$$

The *si*-tree transducer M_1 takes a binary tree e over the ranked alphabet $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$ as argument and it delivers a tree t which has the same structure as e, but in which every leaf node n is substituted by an encoding of the reverse path from the root of e to n. The encoding of a reverse path is a monadic tree over the ranked alphabet $\{B^{(1)}, L^{(1)}, R^{(1)}, E^{(0)}\}$, where the symbol L (and R) represent the left son (and the right son, respectively) of a node and the symbol B (and E) is the first symbol (and the last symbol, respectively) of each path encoding (cf. Figure 2). □

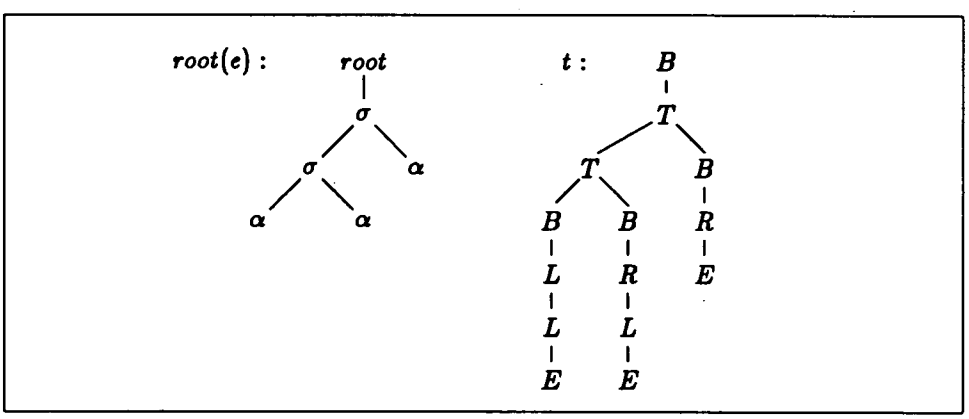


Figure 2: The control tree $\tilde{e}$ and the calculated output tree t.

Observation 2.3

1. Top–down tree transducers [Rou70,Tha70,Eng75] are *si*–tree transducers without inherited attributes.

2. Attributed tree transducers [Fül81] are *si*–tree transducers in which, for every inherited attribute i, the right–hand side of the $(i, z1, root)$–rule is a tree over Δ. In accordance to [Gie88] *si*–tree transducers are *full attributed tree transducers*. But in the sequel we also use simply the notion attributed tree transducer. □

Before working out the definition of the derivation relation, we first introduce a uniform classification scheme for subclasses of *si*–tree transducers which are induced by the number of attributes.

Definition 2.4

- Let $k_s \in \mathbb{N} - \{0\}$ and $k_i \in \mathbb{N}$. An $s_{(k_s)}i_{(k_i)}$*–tree transducer* M is an *si*–tree transducer with at most k_s synthesized attributes and with at most k_i inherited attributes.

- An *s–tree transducer* is an $s_{(k_s)}i_{(0)}$–tree transducer for some $k_s \in \mathbb{N} - \{0\}$, i.e., an *si*–tree transducer without inherited attributes. □

In the next definition we inductively describe the set of all sentential forms of attributed tree transducers. For a given control tree $\tilde{e} = root(e)$ with $e \in T\langle\Sigma\rangle$, a sentential form is a tree over attributes, working symbols, and paths through $\tilde{e}$. Moreover, the argument of an attribute is always a path through $\tilde{e}$ and vice versa a path may only occur in the argument of an attribute.

Definition 2.5 Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes. Moreover, let $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$ and let Δ' be a ranked alphabet with $\Delta \subseteq \Delta'$. The set of $(A, s_{in}, paths(\tilde{e}), \Delta')$*–sentential forms*, denoted by $SF(A, s_{in}, paths(\tilde{e}), \Delta')$, is defined inductively as follows where we abbreviate $SF(A, s_{in}, paths(\tilde{e}), \Delta')$ by SF.

(i) For every $\delta \in \Delta'^{(r)}$ with $r \geq 0$ and $t_1, \ldots, t_r \in SF$, the tree $\delta(t_1, \ldots, t_r) \in SF$.

(ii) For every $a \in A$ and $p \in paths(\tilde{e})$ with $p \neq \varepsilon$, the tree $a(p) \in SF$.

(iii) The tree $s_{in}(\varepsilon) \in SF$. □

Notice that the tree $\tilde{e}$ does not occur in sentential forms. It is only needed to define the set of paths of $\tilde{e}$.

For an attributed tree transducer $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ with system $\mathcal{A} = (A, A_s, A_i)$ of attributes and for a tree $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$, the set of *attribute occurrences* of $\tilde{e}$, denoted by $att(\tilde{e})$, is the set $\{s_{in}(\varepsilon)\} \cup \{a(p) \mid a \in A,$

$p \in paths(\tilde{e}), p \neq \epsilon\}$. If $\tilde{e} = root(e)$ for a particular tree $e \in T\langle\Sigma\rangle$, then we define $att(e) = att(\tilde{e}) - \{s_{in}(e)\}$.

Let $e' \in T\langle\Sigma_+ \cup \{w\}\rangle$ with exactly one occurrence of a symbol $w \notin \Sigma_+$ be a pattern in a control tree $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$, such that $e' = subtree(\tilde{e}[p' \leftarrow w], p)$ holds for some paths $p, p' \in paths(\tilde{e})$. The set of *inside attribute occurrences of e' with respect to $\tilde{e}$* is the set $(\{s(p) \mid s \in A_s\} \cup \{i(p') \mid i \in A_i\}) \cap att(\tilde{e})$. The set of *outside attribute occurrences of e' with respect to $\tilde{e}$* is the set $(\{i(p) \mid i \in A_i\} \cup \{s(p') \mid s \in A_s\}) \cap att(\tilde{e})$. (The intersection with $att(\tilde{e})$ is necessary to handle the case $p = \epsilon$.) If the underlying control tree $\tilde{e}$ is clear from the context, then we simply use the notions inside and outside attribute occurrences of e'.

Now we describe the derivation relation of an attributed tree transducer M with respect to a control tree $\tilde{e}$. For later purposes, we restrict the derivation relation to work only on particular parts of $\tilde{e}$ parameterizing the derivation relation with a subset $P \subseteq paths(\tilde{e})$.

Definition 2.6 Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes. Let $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$ and $P \subseteq paths(\tilde{e})$. The *derivation relation of M with respect to $\tilde{e}$ and P*, denoted by $\Rightarrow_{M,\tilde{e},P}$, is a binary relation on $SF(A, s_{in}, paths(\tilde{e}), \Delta)$ defined as follows:
For every $t_1, t_2 \in SF(A, s_{in}, paths(\tilde{e}), \Delta)$, $t_1 \Rightarrow_{M,\tilde{e},P} t_2$, iff

- there is a $t' \in SF(A, s_{in}, paths(\tilde{e}), \Delta \cup \{u\})$ in which the 0–ary symbol $u \notin A \cup \Delta$ occurs exactly once,
- there is an attribute $a \in A$,
- there is a path $p \in paths(\tilde{e})$,

such that $t_1 = t'[u/a(p)]$ and if one of the following two conditions holds:

1. - a is a synthesized attribute,
   - $p \in P$ and $label(\tilde{e}, p) = \sigma$ for some $\sigma \in \Sigma_+^{(k)}$ with $k \geq 0$,
   - there is a rule $a(z) \rightarrow \rho$ in R_σ, and
   - $t_2 = t'[u/\rho[z/p]]$.
2. - a is an inherited attribute,
   - $p = p'j$ for some $p' \in P$, $label(\tilde{e}, p') = \sigma$ for some $\sigma \in \Sigma_+^{(k)}$ with $k \geq 1$, and $j \in [k]$,
   - there is a rule $a(zj) \rightarrow \rho$ in R_σ, and
   - $t_2 = t'[u/\rho[z/p']]$. □

Note that in case 2. the path p itself needs not to be in P. This is important for the later construction in the pumping lemma. If M or $\tilde{e}$ are known from the context, we drop the corresponding indices from $\Rightarrow$. If $P = paths(\tilde{e})$, then we drop P.

Before presenting an example derivation we have to explain the special role of the marker *root*. It allows us to handle the calculation of the values of inherited attribute occurrences at the root of an input tree e like all the other attribute occurrences of e. Taking the control tree $root(e)$, we can specify the value of an inherited attribute occurrence at the root of e by a rule in R_{root}. In particular, the inherited attribute occurrences at the root of e may depend on the synthesized attribute occurrences at the root of e. This mechanism has also been used in [KV94]. It is more general than the solution presented in [Fül81], where special trees in $T\langle\Delta\rangle$ are used to specify the values of the inherited attribute occurrences at the root of e.

Example 2.7 Let M_1 be the attributed tree transducer defined in Example 2.2 and let $\tilde{e} = root(\sigma(\sigma(\alpha,\alpha),\alpha))$ be the control tree. We abbreviate $\Rightarrow_{M_1,\tilde{e},paths(\tilde{e})}$ by $\Rightarrow$. The number of the applied rule is indicated as a subscript. The control tree and the calculated output tree are also shown in Figure 2.

$$
\begin{aligned}
& s(e)\\
\Rightarrow_{(1)}\ & B(s(1))\\
\Rightarrow_{(3)}\ & B(T(s(11),s(12)))\\
\Rightarrow_{(3)}\ & B(T(T(s(111),s(112)),s(12)))\\
\Rightarrow_{(6)}\ & B(T(T(B(i(111)),s(112)),s(12)))\\
\Rightarrow_{(4)}\ & B(T(T(B(L(i(11))),s(112)),s(12)))\\
\Rightarrow_{(4)}\ & B(T(T(B(L(L(i(1)))),s(112)),s(12)))\\
\Rightarrow_{(2)}\ & B(T(T(B(L(L(E))),s(112)),s(12)))\\
\Rightarrow^{+}\ & B(T(T(B(L(L(E))),B(R(L(E)))),B(R(E))))
\end{aligned}
$$

□

2.4 Noncircular attributed tree transducers

Since an attributed tree transducer can be circular (in the same sense as an attribute grammar), we can conclude that, in general, the derivation relations of attributed tree transducers are not noetherian (cf., e.g., [Ems91] for an example of a circular attributed tree transducer.) However, noncircular attributed tree transducers induce noetherian derivation relations. The notion of circularity is taken from [Fül81]:

Definition 2.8 Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an si–tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes.

1. M is *circular* if
   - there is an $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$
   - there is an $a(p) \in SF(A, s_{in}, paths(\tilde{e}), \Delta)$ with $a \in A$ and $p \in paths(\tilde{e})$,
   - there is a $t \in SF(A, s_{in}, paths(\tilde{e}), \Delta \cup \{u\})$ in which the 0–ary symbol $u \notin A \cup \Delta$ occurs exactly once,

such that $a(p) \Rightarrow^{+}_{M,\tilde{e}} t[u/a(p)]$.

2. M is *noncircular* if it is not circular. □

For the definition of the tree transformation computed by an attributed tree transducer we use the following result (cf. Theorem 3.17 of [KV94]).

Lemma 2.9 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*-tree transducer. If M is noncircular, then for every $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$, the relation $\Rightarrow_{M,\tilde{e}}$ is confluent and noetherian. □

Since the derivation relations of noncircular attributed tree transducers are confluent and noetherian, every sentential form has a unique normal form. This is the basis for the definition of the tree transformation which is computed by an attributed tree transducer.

Definition 2.10 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be a noncircular *si*- tree transducer. 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 $e \in T\langle\Sigma\rangle$,

$$\tau(M)(e) = nf(\Rightarrow_{M,root(e)}, s_{in}(e)).$$ □

In the rest of this paper, we always mean noncircular attributed tree transducers when we talk about attributed tree transducers.

For a given control tree $\tilde{e}$, for a given derivation $s_{in}(e) \Rightarrow^{+}_{\tilde{e}} t$ (abbreviated by d), where $t = nf(\Rightarrow_{\tilde{e}}, s_{in}(e))$, and for a given path p in $\tilde{e}$ we define the set $attset(d,p)$ of those attributes a, for which there are attribute occurrences $a(p)$ in a sentential form during the derivation d. This concept is the same as the concept of *state-set* described in [ERS80], however, we use another way of definition.

Definition 2.11 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*-tree transducer with system $A = (A, A_s, A_i)$ of attributes. Let $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$. Let d be the derivation $s_{in}(e) = t_0 \Rightarrow_{\tilde{e}} t_1 \Rightarrow_{\tilde{e}} \ldots \Rightarrow_{\tilde{e}} t_n = nf(\Rightarrow_{\tilde{e}}, s_{in}(e))$ with $n \geq 1$ derivation steps, and let $p \in paths(\tilde{e})$. Then we define the *attribute-set of* d *and* p, denoted by $attset(d,p)$, by

$$\bigcup_{j=0}^{n} attset'(t_j, p) \qquad \text{where}$$

$attset' : SF(A, s_{in}, paths(\tilde{e}), \Delta) \times paths(\tilde{e}) \longrightarrow \mathcal{P}(A)$ is defined as follows:
For every $\delta \in \Delta^{(r)}$, $r \geq 0$, $t_1, \ldots, t_r \in SF(A, s_{in}, paths(\tilde{e}), \Delta)$, $p \in paths(\tilde{e})$,
$attset'(\delta(t_1, \ldots, t_r), p) = \bigcup_{j=1}^{r} attset'(t_j, p)$.
For every $a(p') \in att(\tilde{e})$, $p \in paths(\tilde{e})$, if $p = p'$, then
$attset'(a(p'), p) = \{a\}$.
For every $a(p') \in att(\tilde{e})$, $p \in paths(\tilde{e})$, if $p \neq p'$, then
$attset'(a(p'), p) = \emptyset$. □

Example 2.12 Let M_1 be the attributed tree transducer defined in Example 2.2 and let $\tilde{e} = root(\alpha)$ be the control tree.
Let $d = (s(e) \Rightarrow_{\tilde{e}} B(s(1)) \Rightarrow_{\tilde{e}} B(B(i(1))) \Rightarrow_{\tilde{e}} B(B(E)))$ be a derivation.
Then $attset(d, e) = attset'(s(e), e) = \{s\}$
and $attset(d, 1) = attset'(B(s(1)), 1) \cup attset'(B(B(i(1))), 1) = \{s, i\}$ hold. □

In fact, the attribute–set of a path does not depend on the chosen derivation.

Lemma 2.13 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer. Let d_1 and d_2 be two derivations $s_{in}(e) \Rightarrow^+_{\tilde{e}} nf(\Rightarrow_{\tilde{e}}, s_{in}(e))$ for some $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$. Then, for every path $p \in paths(\tilde{e})$, the sets $attset(d_1, p)$ and $attset(d_2, p)$ are equal. □

Definition 2.14 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer. Let $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$ and let $p \in paths(\tilde{e})$. The *attribute–set of* $\tilde{e}$ *and* p, denoted by $attset(\tilde{e}, p)$, is the set $attset(d, p)$ for some derivation $d = (s_{in}(e) \Rightarrow^+_{\tilde{e}} nf(\Rightarrow_{\tilde{e}}, s_{in}(e)))$. □

2.5 Producing and visiting attributed tree transducers

The pumping lemma in the next section is only valid for special kinds of attributed tree transducers. In the following definition we introduce the concepts of producing (every rule application produces at least one new output symbol), and visiting (every node of a control tree is visited by at least one attribute) tree transducers.

Definition 2.15 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer. M is

- *producing*, if, for every rule $\lambda \to \rho$ in R, the size of ρ with respect to Δ is at least 1, i.e., $size_\Delta(\rho) \geq 1$,
- *visiting*, if, for every control tree $\tilde{e} \in \{root(e) \mid e \in T\langle\Sigma\rangle\}$ and for every $p \in paths(\tilde{e})$, the attribute–set of $\tilde{e}$ and p is not empty, i.e., $attset(\tilde{e}, p) \neq \emptyset$.

□

In the rest of this paper we always mean producing and visiting (and noncircular) attributed tree transducers, when we talk about attributed tree transducers. We denote the *classes of tree transformations computed* by (noncircular, producing, and visiting) *si*–tree transducers, $s_{(k_s)}i_{(k_i)}$– tree transducers, and *s*–tree transducers by SIT, $S_{(k_s)}I_{(k_i)}T$, and ST, respectively.

2.6 Output languages of attributed tree transducers

The pumping lemma which we introduce in the next section, deals with output languages of tree transformations of attributed tree transducers. The output language of a tree transformation τ is defined as the range of τ.

Definition 2.16 Let $\tau : T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ be a tree transformation. The *output language of* τ, denoted by $L_{out}(\tau)$ is defined as follows:

$$L_{out}(\tau) = \{t \in T\langle\Delta\rangle \mid \text{there is an } e \in T\langle\Sigma\rangle \text{ such that } \tau(e) = t\}. \qquad \square$$

If $\tau(M)$ is a tree transformation computed by an attributed tree transducer M, we simply write $L_{out}(M)$ instead of $L_{out}(\tau(M))$ and we simply call $L_{out}(M)$ the *output language of* M instead of the output language of the tree transformation computed by M.

We denote the *classes of output languages* of (noncircular, producing, and visiting) si–tree transducers, $s_{(k_s)}i_{(k_i)}$–tree transducers, and s–tree transducers by SIT_{out}, $S_{(k_s)}I_{(k_i)}T_{out}$, and ST_{out}, respectively.

If we want to prove that a certain tree transformation τ is not an element of the class SIT, then the output language $L_{out}(\tau)$ can be very useful. It would suffice to show with the help of the pumping lemma presented in the next section that $L_{out}(\tau) \notin SIT_{out}$. Thus, since $L_{out}(\tau)$ is not the range of an si–tree transducer, τ cannot be the tree transformation computed by an si–tree transducer.

For the sake of convenience, we now omit the parantheses for arguments of monadic output symbols in the rest of the paper; the parantheses for arguments of attributes remain.

Example 2.17 Let M_1 be the attributed tree transducer defined in Example 2.2 and let d be the derivation of Example 2.7.

Thus, in the following we write rule (1) of M_1 in the form $s(z) \rightarrow B\,s(z1)$. Note that there are still parantheses in the attribute occurrence $s(z1)$. The notation $s(z) \rightarrow T(s(z1), s(z2))$ of rule (3) is left unchanged, because T is a binary output symbol.

In anology we write the last but one sentential form of d that was shown in Example 2.7 as $B\,T(T(B\,L\,L\,E, s(112)), s(12))$. $\square$

3 Pumping lemma for attributed tree transducers

Before presenting the pumping lemma for si–tree transducers and working out the proof formally, we want to illustrate the central idea and show an example. Although the pumping lemma only deals with output trees and not with the control trees corresponding to them via a tree transformation, the control trees play an important part.

Let M be an attributed tree transducer. If we choose a sufficiently large output tree t, then every control tree $\tilde{e} = root(e)$ with $\tau(M)(e) = t$ is high enough, such that it has a path p, on which two different nodes x_1 and x_2 can be found such that (cf. Figure 3)

- there exist strings p_1, p_2, and p_3 such that $|p_2| > 0$ and $p = p_1p_2p_3$,

- x_1 and x_2 can be reached from the root by p_1 and p_1p_2, respectively, i.e., $x_1 = node(\tilde{e}, p_1)$ and $x_2 = node(\tilde{e}, p_1p_2)$, and
- the attribute–sets $attset(\tilde{e}, p_1)$ and $attset(\tilde{e}, p_1p_2)$ are equal.

These two nodes define a decomposition of $\tilde{e}$ into three input patterns e', e'', and e'''. Intuitively,

- e' is the tree $\tilde{e}$ 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.

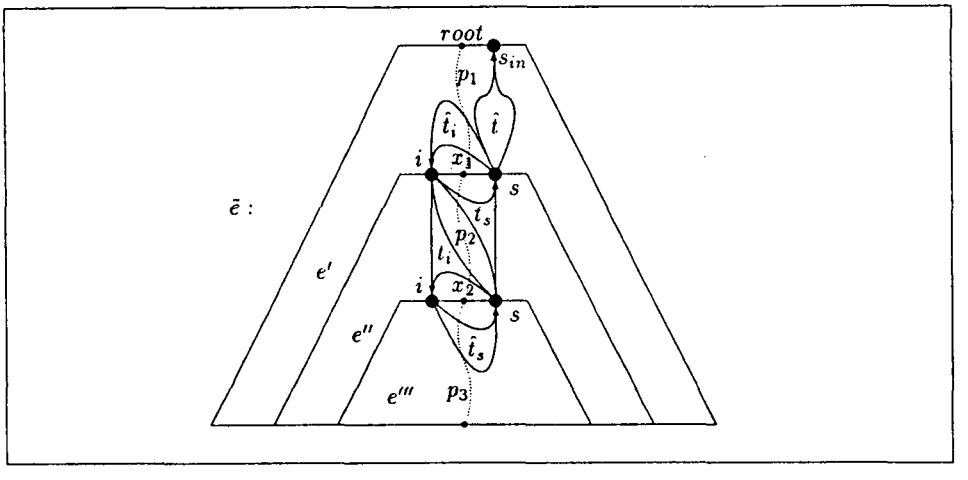


Figure 3: Control tree $\tilde{e}$ with input patterns and induced output patterns.

This decomposition of the control tree $\tilde{e}$ induces a decomposition of the output tree t into a certain output pattern $\hat{t}$, certain output patterns t_s and $\hat{t}_s$ for every synthesized attribute s, and certain output patterns t_i and $\hat{t}_i$ for every inherited attribute i. Roughly speaking, these patterns correspond to normal forms of certain attribute occurrences of the patterns e', e'', and e'''. More precisely,

- The tree $\hat{t}$ corresponds to the normal form of $s_{in}(\varepsilon)$ that is calculated only on the nodes of e'.
- For every synthesized attribute s in the attribute–set of the two relevant nodes x_1 and x_2, the tree t_s (and $\hat{t}_s$) corresponds to the normal form of $s(p_1)$ (and $s(p_1p_2)$, respectively) that is calculated only on the nodes of e'' (and e''', respectively).

- For every inherited attribute i in the attribute- -set of the two relevant nodes x_2 and x_1, the tree t_i (and $\hat{t}_i$) corresponds to the normal form of $i(p_1p_2)$ (and $i(p_1)$, respectively) that is calculated only on the nodes of e'' (and e', respectively).

In Figure 3 these output patterns are indicated; the root of every output pattern is represented by an arrow. The reader should not be misleaded by the cycles among the pieces of the final output tree: we consider noncircular attributed tree transducers and, only for the sake of simplicity of the figure, we show only one inherited attribute and one synthesized attribute; thus, dependencies are folded and suggest cycles which are not there.

If we construct new control trees by repeating the pattern e'' arbitrarily often, then we can get new output trees by translating the new control trees. All of them are by definition elements of $L_{out}(M)$. The output patterns t_s and t_i must be used for every repetition of e'' to obtain the new output tree. Figure 4 shows the situation in which e'' is repeated twice.

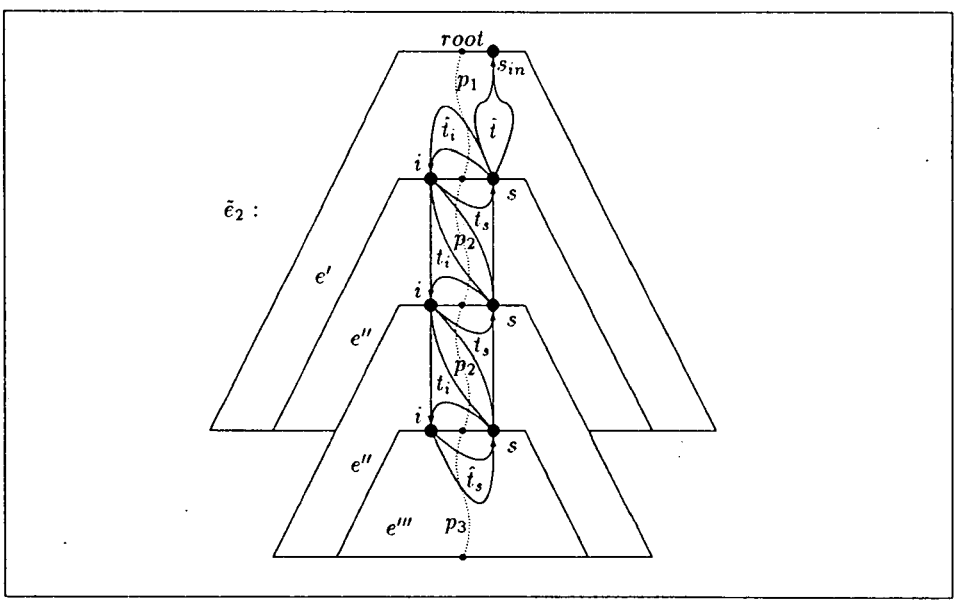


Figure 4: Control tree with two repetitions of e'' and output patterns.

In the pumping lemma we use a recursive function $\underline{tree'}$ which walks through the patterns of the control tree and builds up the output using the output patterns

defined above.

Note that for the pumping process it is not necessary that the nodes x_1 and x_2 are labeled by the same symbol, in contrast to the pumping lemma for context–free languages (cf. for example [BPS61]). This is due to the fact that we only deal with ranked alphabets rather than heterogeneous signatures; thus only the rank of the symbols is important when building up trees.

We show the input patterns, the output patterns and the pumping process in the following example.

Example 3.1 Let M_1 be the *si*–tree transducer defined in Example 2.2. For simplicity we repeat the rules of M_1, omitting superfluous paranthesis:

$$
\begin{array}{lcll}
R_{root} & = & \{s(z) & \rightarrow B\, s(z1), \\
 & & i(z1) & \rightarrow E \qquad\qquad\quad \} \\
R_\sigma & = & \{s(z) & \rightarrow T(s(z1), s(z2)), \\
 & & i(z1) & \rightarrow L\, i(z), \\
 & & i(z2) & \rightarrow R\, i(z) \qquad\quad \} \\
R_\alpha & = & \{s(z) & \rightarrow B\, i(z) \qquad\quad \}
\end{array}
$$

Although the pumping lemma only guarantees to work with an output tree t with $size(t) \geq n_{M_1}$ for a certain natural number n_{M_1} (which is called the pumping index of M_1), it often also works for smaller output trees. Nevertheless, the pumping index is needed in the proof of the pumping lemma. In this example we have $n_{M_1} = 2^{15}$. The reader can check this after having read Definition 3.2.

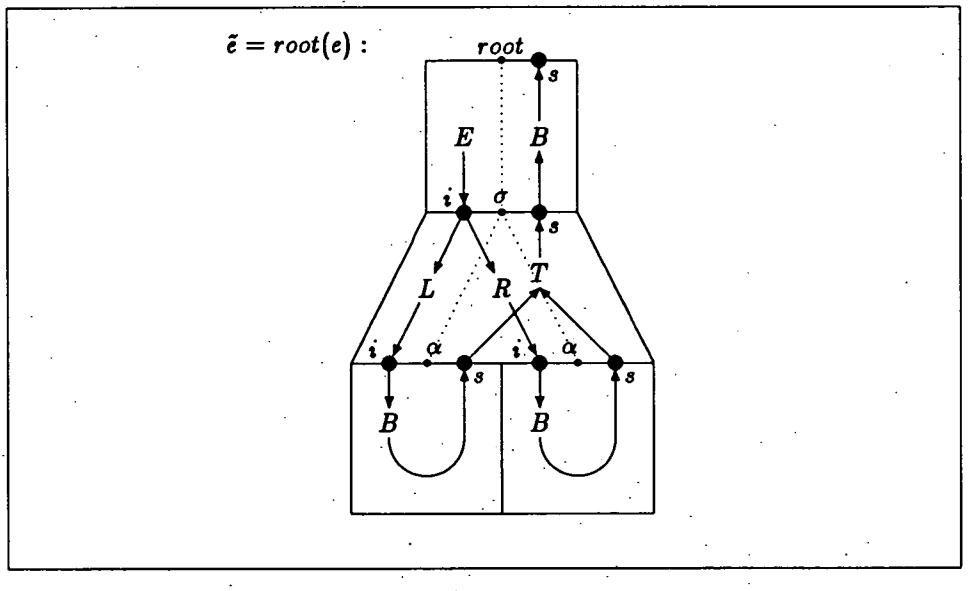


Figure 5: Control tree $\tilde{e}$ with right–hand sides of rules.

Here we take the smaller tree $t = nf(\Rightarrow_{\tilde{e}}, s_{in}(e))$, where $\tilde{e} = root(\sigma(\alpha, \alpha))$ is the control tree. In Figure 5 the control tree $\tilde{e}$ is shown by dotted lines, where additionally the right-hand sides of those rules are incorporated which are necessary to compute the values of the attribute occurrences of $\tilde{e}$.

Now we consider the two nodes $node(\tilde{e}, 1)$ and $node(\tilde{e}, 11)$ of the control tree $\tilde{e}$ which can be reached from the root of $\tilde{e}$ by paths 1 and 11. Note that $\sigma = label(\tilde{e}, 1)$, $\alpha = label(\tilde{e}, 11)$, and $attset(\tilde{e}, 1) = attset(\tilde{e}, 11) = \{s, i\}$. In this case we have chosen the path $p = 11$ with its subpaths $p_1 = 1$, $p_2 = 1$, and $p_3 = \varepsilon$. In Figure 6 we show three patterns in $\tilde{e}$ with the nodes reached by the paths ε, 1, and 11, respectively, as roots. Again the right-hand sides of rules are incorporated into the figure.

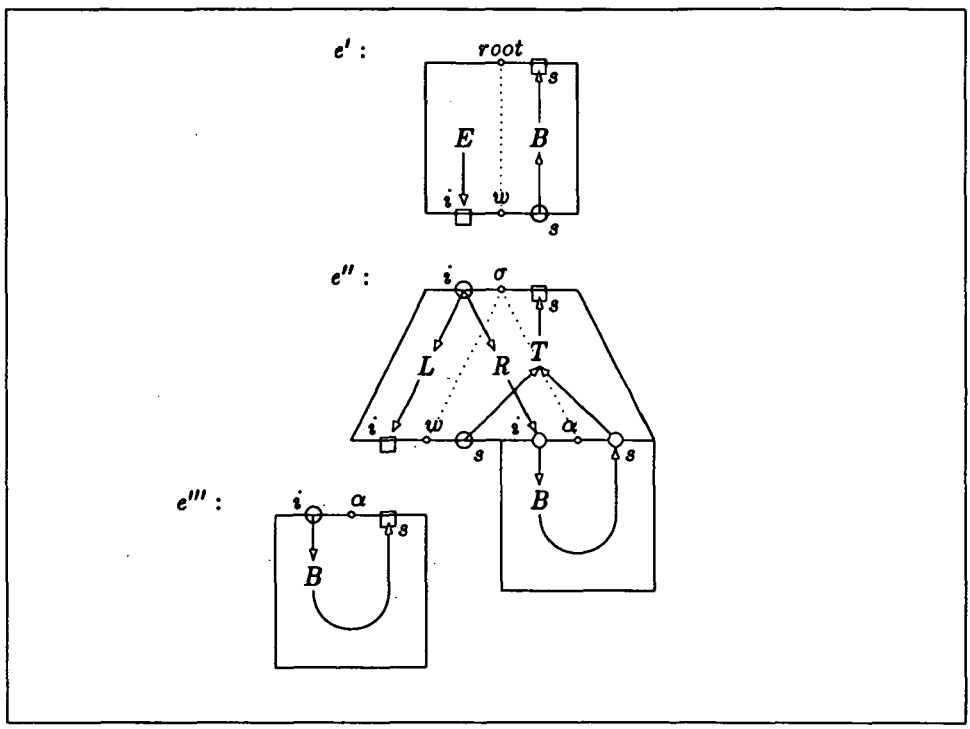


Figure 6: Input patterns e', e'' and e''' with right-hand sides of rules.

For later purposes, in Figure 7 we also show the control tree $\tilde{e}$ and the patterns e', e'', and e''' framing those parts of the patterns which only consist of input symbols. In fact, we have $\tilde{e} = e'[w/e''[w/e''']]$.

With these preparations we can obtain the patterns in the output tree t as follows: Roughly speaking, for each of the patterns e', e'', and e''', we calculate the values of the inside attribute occurrences as function in the values of the outside attribute occurrences. Therefore we can use the dependencies among the attribute occurrences presented in Figure 6, where the outside attribute occurrences and the inside attribute occurrences are depicted as non-filled cycles and non- filled boxes,

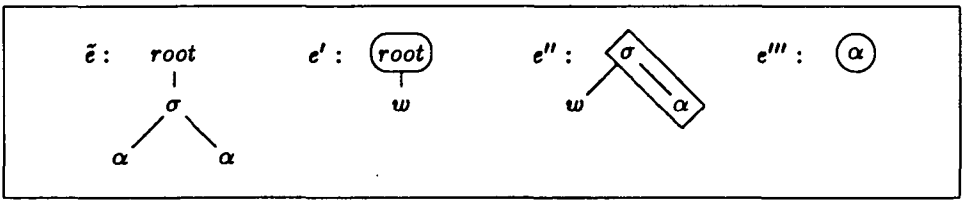


Figure 7: Control tree $\tilde{e}$ and its decomposition.

respectively, whereas the other attribute occurrences are depicted as filled cycles. More precisely, we calculate

- the values $\hat{t}$ and $\hat{t}_i$ of the inside attribute occurrences $s(\varepsilon)$ and $i(1)$ of e', respectively, as function in the value of the outside attribute occurrence $s(1)$ of e',
- the values t_s and t_i of the inside attribute occurrences $s(1)$ and $i(11)$ of e'', respectively, as function in the values of the outside attribute occurrences $s(11)$ and $i(1)$ of e'',
- and the value $\hat{t}_s$ of the inside attribute occurrence $s(11)$ of e''' as function in the value of the outside attribute occurrence $i(11)$ of e''',

and replace the synthesized attribute occurrences $s(1)$ and $s(11)$ by the symbol s with rank 0 and the inherited attribute occurrences $i(1)$ and $i(11)$ by the symbol i with rank 0. For the sake of understanding we choose exactly the attributes as names for the new symbols. Based on the rank, the reader can retrieve whether symbols or attributes are concerned at a time. The values of the output patterns are as follows:

$$
\begin{array}{lcrcl}
\hat{t} & = & nf(\Rightarrow_{\tilde{e},\{\varepsilon\}}, s(\varepsilon))[s(1)/s] & = & B\ s, \\
\hat{t}_i & = & nf(\Rightarrow_{\tilde{e},\{\varepsilon\}}, i(1))[s(1)/s] & = & E, \\
t_s & = & nf(\Rightarrow_{\tilde{e},\{1,12\}}, s(1))[s(11)/s,\ i(1)/i] & = & T(s, B\ R\ i), \\
t_i & = & nf(\Rightarrow_{\tilde{e},\{1,12\}}, i(11))[s(11)/s,\ i(1)/i] & = & L\ i, \\
\hat{t}_s & = & nf(\Rightarrow_{\tilde{e},\{11\}}, s(11))[i(11)/i] & = & B\ i.
\end{array}
$$

In Figure 8 we show the output tree t and the output patterns defined above. For later purposes we also frame the parts of the patterns which only consist of output symbols.

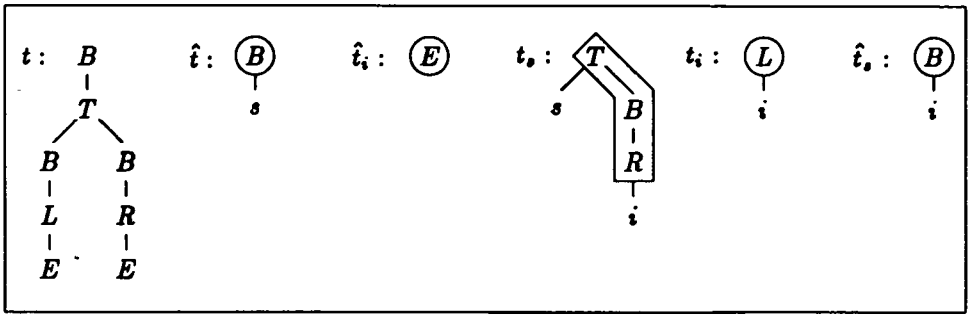


Figure 8: Output tree t and output patterns.

Now we show the pumping process in the cases in which

(i) the pattern e'' is dropped ($r = 0$),

(ii) the pattern e'' occurs once ($r = 1$), and

(iii) the pattern e'' occurs twice ($r = 2$).

Thus we have the control tree

(i) $\tilde{e}_0 = e'[w/e''']$, if $r = 0$,

(ii) $\tilde{e} = \tilde{e}_1 = e'[w/e''[w/e''']]$, if $r = 1$, and

(iii) $\tilde{e}_2 = e'[w/e''[w/e''[w/e''']]]$, if $r = 2$.

For every $0 \leq r \leq 2$, the normal form $nf(\Rightarrow_{\tilde{e}_r}, s_{in}(e))$ is denoted by $\underline{tree}(r)$. It can also be calculated using the above defined patterns of t as follows:

We start with the pattern $\hat{t} = Bs$ that corresponds to the attribute occurrence $s(\varepsilon)$, and replace the symbol s by the function call $\underline{tree'}(s, r, 1)$. Roughly speaking, the recursive function $\underline{tree'}$ moves through the different patterns of $\tilde{e}_r$ and it constructs the output using the output patterns. Every function call of $\underline{tree'}$ delivers one output pattern, in which the symbols s and i are replaced by new function calls of $\underline{tree'}$.

The function $\underline{tree'}$ has three parameters. The first parameter is one of the symbols s or i. It indicates, whether we have to use one of the patterns t_s or $\hat{t}_s$ (in case of the symbol s), or one of the patterns t_i or $\hat{t}_i$ (in case of the symbol i). The other two parameters are natural numbers. The second parameter r indicates the number of repetitions of e'' in the control tree $\tilde{e}_r$. It is constant during the calculation of a certain output tree. The third parameter l indicates the level of the input pattern, where $\underline{tree'}$ currently works. $l = 0$ means the pattern e', $1 \leq l \leq r$ means the l–th repetition of the pattern e'', and $l = r + 1$ means the pattern e'''.

If $1 \leq l \leq r$, then $\underline{tree'}$ uses the pattern $t_s = T(s, BRi)$ (or $t_i = Li$); this pattern corresponds to the normal form which is calculated only on the nodes of the pattern e'' starting with the attribute occurrence $s(1)$ (or $i(11)$, respectively).

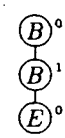
If $l = r+1$, then $\underline{tree'}$ uses the pattern $\hat{t}_s = Bi$; this pattern corresponds to the normal form which is calculated only on the nodes of the pattern e''' starting with the attribute occurrence $s(11)$.

If $l = 0$, then $\underline{tree'}$ uses the pattern $\hat{t}_i = E$; this pattern corresponds to the normal form which is calculated only on the nodes of the pattern e' starting with the attribute occurrence $i(1)$.

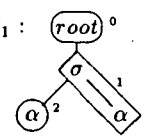
If l is the current level of the function $\underline{tree'}$, then every occurrence of the symbol s (or i) in the produced output pattern is replaced by a function call $\underline{tree'}(s, r, l+1)$ (or $\underline{tree'}(i, r, l-1)$, respectively), because $\underline{tree'}$ has to move one level down (or up, respectively) in $\tilde{e}_r$.

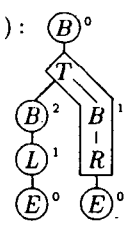
$\underline{tree}(0)$
$= B\ \underline{tree'}(s,0,1)$
$= B\ B\ \underline{tree'}(i,0,0)$
$= B\ B\ E$

$\tilde{e}_0$: 

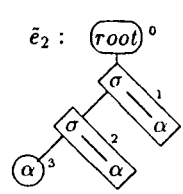
$\underline{tree}(0)$: 

$\underline{tree}(1)$
$= B\ \underline{tree'}(s,1,1)$
$= B\ T(\underline{tree'}(s,1,2), B\ R\ \underline{tree'}(i,1,0))$
$= B\ T(B\ \underline{tree'}(i,1,1), B\ R\ E)$
$= B\ T(B\ L\ \underline{tree'}(i,1,0), B\ R\ E)$
$= B\ T(B\ L\ E, B\ R\ E)$

$\tilde{e} = \tilde{e}_1$: 

$t = \underline{tree}(1)$: 

$\underline{tree}(2)$
$= B\ \underline{tree'}(s,2,1)$
$= B\ T(\underline{tree'}(s,2,2), B\ R\ \underline{tree'}(i,2,0))$
$= B\ T(T(\underline{tree'}(s,2,3),$
$\quad B\ R\ \underline{tree'}(i,2,1)), B\ R\ E)$
$= B\ T(T(B\ \underline{tree'}(i,2,2),$
$\quad B\ R\ L\ \underline{tree'}(i,2,0)), B\ R\ E)$
$= B\ T(T(B\ L\ \underline{tree'}(i,2,1),$
$\quad B\ R\ L\ E), B\ R\ E)$
$= B\ T(T(B\ L\ L\ \underline{tree'}(i,2,0),$
$\quad B\ R\ L\ E), B\ R\ E)$
$= B\ T(T(B\ L\ L\ E, B\ R\ L\ E), B\ R\ E)$

$\tilde{e}_2$: 

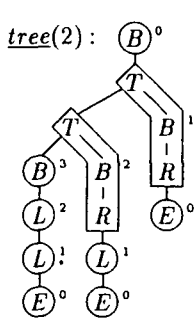
$\underline{tree}(2)$: 

Figure 9: Calculations of $\underline{tree}(r)$ for $0 \leq r \leq 2$ and decompositions of $\tilde{e}_r$ and $\underline{tree}(r)$.

In Figure 9 we show besides the calculations of the output trees $\underline{tree}(0)$, $\underline{tree}(1)$, and $\underline{tree}(2)$, their decompositions into the output patterns. Every output pattern is labeled with the level $0 \leq l \leq r+1$ of the input pattern which causes it. We also show the control trees, corresponding to the output trees, and their decompositions into input patterns which are labeled with their level. □

As stated in the last example, the pumping process only guarantees to work for output trees which are large enough. This requirement is satisfied, if the size of the output tree is at least the pumping index of the given attributed tree transducer. Recall that we only consider noncircular, producing, and visiting attributed tree transducers.

Definition 3.2 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with k_s synthesized and k_i inherited attributes. We define

$$\begin{aligned} c_M &= max\{size_A(\rho) \mid (\lambda \to \rho) \in R\} \\ &\quad \text{(maximum number of attribute occurrences in right-hand sides)}, \\ l_M &= max\{size_\Delta(\rho) \mid (\lambda \to \rho) \in R\} \\ &\quad \text{(maximum number of output symbols in right-hand sides)}, \\ m_M &= max\{rank(\sigma) \mid \sigma \in \Sigma\} \\ &\quad \text{(maximum rank of input symbols)}, \end{aligned}$$

and the *pumping index* n_M *of* M as:

$$n_M = 1 + l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j \quad \text{where} \quad n'_M = \sum_{j=0}^{2^{k_i}\cdot(2^{k_s}-1)} (m_M)^j.$$

□

In the proof of the pumping lemma we need the fact that the subtree e of a control tree $root(e)$ has at least some particular height; the desired height is $2^{k_i} \cdot (2^{k_s}-1)+2$ (cf. the proof of Theorem 3.4 for an argumentation on this number). If, for an attributed tree transducer M and for a derivation $s_{in}(e) \Rightarrow^+_{root(e)} t$, the size of t is at least the pumping index n_M, then e has the desired height.

Lemma 3.3 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with k_s synthesized attributes, k_i inherited attributes, and with pumping index n_M. Let $t \in L_{out}(M)$.

If $size(t) \geq n_M$, then for every $e \in T\langle\Sigma\rangle$ such that $t = nf(\Rightarrow_{root(e)}, s_{in}(e))$, the height $height(e) \geq 2^{k_i} \cdot (2^{k_s}-1) + 2$.

Proof. Consider $t \in L_{out}(M)$ with $size(t) \geq n_M$. We examine a control tree $\tilde{e} = root(e)$ with a certain derivation $s_{in}(e) \Rightarrow^+_{\tilde{e}} t$. We abbreviate this derivation by d and the number of derivation steps of d by $length(d)$. The proof consists of a sequence of five implications. First, we list these implications and afterwards we give some explanations.

(1) If $size(t) \geq n_M = 1 + l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$,
then $length(d) \geq 1 + \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$.

(2) If $length(d) \geq 1 + \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$, then $card(att(\tilde{e})) \geq 2 + (k_s + k_i) \cdot n'_M$.

(3) If $card(att(\tilde{e})) \geq 2 + (k_s + k_i) \cdot n'_M$, then $card(att(e)) \geq 1 + (k_s + k_i) \cdot n'_M$.

(4) If $card(att(e)) \geq 1 + (k_s + k_i) \cdot n'_M$, then $size(e) \geq 1 + n'_M$.

(5) If $size(e) \geq 1 + n'_M = 1 + \sum_{j=0}^{2^{k_i}\cdot(2^{k_s}-1)} (m_M)^j$,
then $height(e) \geq 2^{k_i} \cdot (2^{k_s} - 1) + 2$.

(1) Since l_M is the maximum number of output symbols in the right-hand sides of the rules of M, $\sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ rule applications can produce at most $l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ output symbols. Hence, since $size(t) \geq 1 + l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$, it needs at least $1 + \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ rule applications to generate t.

(2) Since every attribute occurrence can call at most c_M other attribute occurrences in one derivation step, $1 + (k_s + k_i) \cdot n'_M$ different attribute occurrences of $\tilde{e}$ can cause at most $\sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ rule applications during the whole derivation d. To understand this fact, we can construct the calling tree of d with attribute occurrences of $\tilde{e}$ as nodes: the root of this tree is labeled $s_{in}(e)$; every node of the tree labeled $a(p)$ has as many sons as there are attribute occurrences in t' with $a(p) \Rightarrow_{\tilde{e}} t'$; the sons are labeled by the different attribute occurrences. It is easy to observe that the length $length(d)$ of the derivation d is equal to the size of the calling tree. Under the assumption that there are at most $1 + (k_s + k_i) \cdot n'_M$ different attribute occurrences of $\tilde{e}$, the height of the calling tree is at most $1 + (k_s + k_i) \cdot n'_M$, because M is noncircular. Thus its size is at most $\sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$. Hence, since $length(d) \geq 1 + \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$, we have at least $2 + (k_s + k_i) \cdot n'_M$ different attribute occurrences of $\tilde{e}$.

(3) At the root of $\tilde{e}$ we only have the attribute occurrence $s_{in}(e)$, thus there exist at least $1 + (k_s + k_i) \cdot n'_M$ attribute occurrences of e.

(4) Since M has $k_s + k_i$ attributes, an input tree e with n'_M nodes can only have $(k_s + k_i) \cdot n'_M$ attribute occurrences. Hence, since $card(att(e)) \geq 1 + (k_s + k_i) \cdot n'_M$, we must have $size(e) \geq 1 + n'_M$.

(5) Since m_M is the maximal rank of the input symbols, an input tree with height $2^{k_i} \cdot (2^{k_s} - 1) + 1$ can only have the size $\sum_{j=0}^{2^{k_i}\cdot(2^{k_s}-1)} (m_M)^j$. Hence, since

$size(e) \geq 1 + n'_M = 1 + \sum_{j=0}^{2^{k_i}\cdot(2^{k_s}-1)} (m_M)^j$, we must have $height(e) \geq 2^{k_i} \cdot (2^{k_s} - 1) + 2$. □

Theorem 3.4 (Pumping Lemma)
Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an si-tree transducer with system $A = (A, A_s, A_i)$ of attributes and pumping index n_M.
For every $t \in L_{out}(M)$ with $size(t) \geq n_M$

- there exist three ranked alphabets
  - $(U_s, rank_{U_s})$ with $U_s \subseteq A_s$, $card(U_s) \geq 1$, and $rank_{U_s}(s) = 0$ for every $s \in U_s$,
  - $(U_i, rank_{U_i})$ with $U_i \subseteq A_i$ and $rank_{U_i}(i) = 0$ for every $i \in U_i$, and
  - $U = U_s \cup U_i$,
- there exists $\hat{t} \in T\langle \Delta \cup U_s \rangle - T\langle \Delta \rangle$ with $size_\Delta(\hat{t}) \geq 1$,
- for every $i \in U_i$, there exists a tree $\hat{t}_i \in T\langle \Delta \cup U_s \rangle$with $size_\Delta(\hat{t}_i) \geq 1$,
- for every $s \in U_s$, there exists a tree $t_s \in T\langle \Delta \cup U \rangle$ with $1 \leq size_\Delta(t_s) < n_M$,
- for every $i \in U_i$, there exists a tree $t_i \in T\langle \Delta \cup U \rangle$ with $1 \leq size_\Delta(t_i) < n_M$,
- for every $s \in U_s$, there exists a tree $\hat{t}_s \in T\langle \Delta \cup U_i \rangle$ with $1 \leq size_\Delta(\hat{t}_s) < n_M$,

with

- for every $s \in U_s$, the symbol s occurs in $\hat{t}$ or there is an $i' \in U_i$ such that s occurs in $\hat{t}_{i'}$,
- for every $s \in U_s$, there is an $s' \in U_s$ such that s occurs in $t_{s'}$ or there is an $i' \in U_i$ such that s occurs in $t_{i'}$,
- for every $i \in U_i$, there is an $s' \in U_s$ such that i occurs in $t_{s'}$ or there is an $i' \in U_i$ such that i occurs in $t_{i'}$,
- for every $i \in U_i$, there is an $s' \in U_s$ such that i occurs in $\hat{t}_{s'}$,

such that $t = \underline{tree}(1)$ and for every $r \geq 0$, the tree $\underline{tree}(r) \in L_{out}(M)$. The function

$\underline{tree} : I\!N \longrightarrow T\langle \Delta \rangle$

is for every $r \geq 0$ defined by $\underline{tree}(r) = \hat{t}\,[s/\underline{tree}'(s, r, 1)\ ;\ s \in U_s]$, where the partial function

$\underline{tree'} : U \times \boldsymbol{N} \times \boldsymbol{N} \to T\langle\Delta\rangle$ is defined as follows:
For every $s \in U_s$ and $r \geq 0$, if $l \in [r]$,
$\underline{tree'}(s,r,l) = t_s[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s,\ i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.
For every $s \in U_s$ and $r \geq 0$, if $l = r+1$,
$\underline{tree'}(s,r,l) = \hat{t}_s[i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.
For every $i \in U_i$ and $r \geq 0$, if $l \in [r]$,
$\underline{tree'}(i,r,l) = t_i[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s,\ i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.
For every $i \in U_i$ and $r \geq 0$, if $l = 0$,
$\underline{tree'}(i,r,l) = \hat{t}_i[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s]$.

Proof. Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an si-tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes, k_s synthesized attributes, and k_i inherited attributes.

Consider $t \in L_{out}(M)$ with $size(t) \geq n_M$. By Lemma 3.3 we know that, for every control tree $\tilde{e} = root(e)$ with $s_{in}(\varepsilon) \Rightarrow_{\tilde{e}}^{+} t$, the condition $height(e) \geq 2^{k_i} \cdot (2^{k_s} - 1) + 2$ holds.

We choose a control tree $\tilde{e} = root(e)$, a derivation $d = (s_{in}(\varepsilon) \Rightarrow_{\tilde{e}}^{+} t)$, and a path p with maximal length from the root of $\tilde{e}$ to a leaf of $\tilde{e}$. Then we know that $|p| \geq 2^{k_i} \cdot (2^{k_s} - 1) + 2 > 2^{k_i} \cdot (2^{k_s} - 1)$. Note that here it would have been sufficient to have $|p| \geq 2^{k_i} \cdot (2^{k_s} - 1) + 1$, but later in the proof of the size conditions for the output patterns we again make use of the pumping index n_M to avoid the definition of a new constant. Otherwise we would have had another formulation of the pumping lemma with two constants (like in [BPS61], there the constants are called p and q).

Since there are exactly 2^{k_i} possibilities to choose an arbitrary subset of the k_i inherited attributes and since there are exactly $2^{k_s} - 1$ possibilities to choose an arbitrary, nonempty subset of the k_s synthesized attributes, we have that

$$card(\{attset(\tilde{e}, p') \mid p' \neq \varepsilon, \text{ and } p' \text{ is a prefix of } p\}) \leq 2^{k_i} \cdot (2^{k_s} - 1).$$

Since $|p| > 2^{k_i} \cdot (2^{k_s} - 1)$, there must exist strings $p_1 \neq \varepsilon$, $p_2 \neq \varepsilon$ and p_3 with $p = p_1 p_2 p_3$, such that

$$attset(\tilde{e}, p_1) = attset(\tilde{e}, p_1 p_2).$$

We choose p_1, p_2, and p_3 such that $|p_2 p_3|$ is minimal. This means that we take the first repetition of $attset(\tilde{e}, p')$, where p' is a prefix of p, beginning from the leaf at p. Then we know that $|p_2 p_3| \leq 2^{k_i} \cdot (2^{k_s} - 1)$, because otherwise there is another repetition of $attset$ in that part of p between $node(\tilde{e}, p_1)$ and $node(\tilde{e}, p_1 p_2 p_3)$, in contradiction to $|p_2 p_3|$ being minimal.

We define the subsets $U_s \subseteq A_s$ and $U_i \subseteq A_i$, such that

$$U_s = attset(\tilde{e}, p_1) \cap A_s \qquad \text{and} \qquad U_i = attset(\tilde{e}, p_1) \cap A_i.$$

In fact, $card(U_s) \geq 1$, because M is visiting and thus every symbol of the control tree must be visited by a synthesized attribute.

Let $w \notin \Sigma_+$ with $rank(w) = 0$. We define trees $e' \in T\langle\Sigma_+ \cup \{w\}\rangle$ and $e'' \in T\langle\Sigma \cup \{w\}\rangle$, 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:

$$\begin{aligned} e' &= \tilde{e}[p_1 \leftarrow w] \\ e'' &= subtree(\tilde{e}[p_1p_2 \leftarrow w], p_1) \\ e''' &= subtree(\tilde{e}, p_1p_2) \end{aligned}$$

Then the representation $\tilde{e} = e'[w/e''[w/e''']]$ holds. The reader can find these patterns of $\tilde{e}$ and the paths leading to them in Figure 3.

In the sequel we need the sets P_1, P_2, and P_3 of paths, which lead from the root of $\tilde{e}$ to the nodes in the three parts e', e'', and e''', respectively:

$$\begin{aligned} P_1 &= paths(e') - \{p_1\} \\ P_2 &= (\{p_1\} \cdot paths(e'')) - \{p_1p_2\} \\ P_3 &= \{p_1p_2\} \cdot paths(e''') \end{aligned}$$

Note that the path p_1 leading to the root of e'' is excluded from P_1 and that the path p_1p_2 leading to the root of e''' is excluded from P_2.

Now we calculate, roughly speaking, the values of the inside attribute occurrences of the patterns e', e'', and e''' as functions in the values of the outside attribute occurrences of the same patterns in order to gain the desired output patterns that are needed for the pumping process. Therefore we restrict the derivation relation of M to the sets P_1, P_2, and P_3, respectively, as it is defined in Definition 2.6.

For the definition of the output patterns, we use symbols from the ranked alphabets $(U_s, rank_{U_s})$ and $(U_i, rank_{U_i})$ with $rank_{U_s}(s) = 0$ for every $s \in U_s$, with $rank_{U_i}(i) = 0$ for every $i \in U_i$, and with $U = U_s \cup U_i$. We choose exactly the attributes as names for the symbols, to emphasize their strong connection, although they have different ranks. It is easy to decide from the context in which the names occur, whether symbols or attributes are concerned at a time.

Now we can define (cf. Figure 3):

$$\hat{t} = nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))[s'(p_1)/s' \,;\, s' \in U_s].$$

For every $s \in U_s$,

$$\begin{aligned} t_s &= nf(\Rightarrow_{\tilde{e},P_2}, s(p_1))[s'(p_1p_2)/s' \,;\, s' \in U_s,\ i'(p_1)/i' \,;\, i' \in U_i] \quad \text{and} \\ \hat{t}_s &= nf(\Rightarrow_{\tilde{e},P_3}, s(p_1p_2))[i'(p_1p_2)/i' \,;\, i' \in U_i]. \end{aligned}$$

For every $i \in U_i$,

$$\begin{aligned} t_i &= nf(\Rightarrow_{\tilde{e},P_2}, i(p_1p_2))[s'(p_1p_2)/s' \,;\, s' \in U_s,\ i'(p_1)/i' \,;\, i' \in U_i] \quad \text{and} \\ \hat{t}_i &= nf(\Rightarrow_{\tilde{e},P_1}, i(p_1))[s'(p_1)/s' \,;\, s' \in U_s]. \end{aligned}$$

Note that, by the definition of $\Rightarrow_{\tilde{e},P_2}$, the inherited attribute occurrences $i'(p_1)$ cannot be evaluated and thus may occur in $nf(\Rightarrow_{\tilde{e},P_2}, s(p_1))$ and $nf(\Rightarrow_{\tilde{e},P_2}, i(p_1p_2))$. The same holds for $\Rightarrow_{\tilde{e},P_3}$ and the inherited attribute occurrences $i'(p_1p_2)$ that may occur in $nf(\Rightarrow_{\tilde{e},P_3}, s(p_1p_2))$.

By this definition, every pattern has the type, which is required by the pumping lemma. We only have to check that $\hat{t} \notin T\langle\Delta\rangle$. Again the reason is that every symbol of the control tree must be visited by synthesized attributes. Thus, one of the synthesized attribute occurrences $s(p_1)$ must be called directly from $s_{in}(\varepsilon)$

via a sequence of attribute occurrences of e'. Otherwise the derivation would never reach e''.

Now we prove the size conditions for the patterns:

(a) $size_\Delta(\hat{t}) \geq 1$:

We have $size_\Delta(\hat{t}) \geq 1$, because $p_1 \neq \varepsilon$ and thus there must be at least one rule application to calculate $nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(e)) \neq s_{in}(e)$. Note that M is producing.

(b) For every $i \in U_i$, $size_\Delta(\hat{t}_i) \geq 1$:

We have $size_\Delta(\hat{t}_i) \geq 1$, because $nf(\Rightarrow_{\tilde{e},P_1}, i(p_1))$ can only consist of output symbols and attribute occurrences $s'(p_1)$ and thus cannot be equal to $i(p_1)$. Again there is at least one rule application to calculate the normal form.

(c) For every $s \in U_s$, $1 \leq size_\Delta(t_s) < n_M$:

We have $size_\Delta(t_s) \geq 1$, because $p_2 \neq \varepsilon$ and thus there must be at least one rule application to calculate $nf(\Rightarrow_{\tilde{e},P_2}, s(p_1)) \neq s(p_1)$. We have $size_\Delta(t_s) < n_M$, because the calculation of $nf(\Rightarrow_{\tilde{e},P_2}, s(p_1))$ only takes place on the part e'' of the control tree. Since p is a longest path in $\tilde{e}$, its subpath p_2p_3 with $|p_2p_3| \leq 2^{k_i} \cdot (2^{k_s} - 1)$ is a longest path of $e''[w/e''']$ and thus e'' can have no path with a length greater than $2^{k_i} \cdot (2^{k_s} - 1)$. Then $height(e'') \leq 2^{k_i} \cdot (2^{k_s} - 1) + 1$ and (with a reverse argumentation to fix $height(e)$ in the proof of Lemma 3.3 we get $size(e'') \leq \sum_{j=0}^{2^{k_i} \cdot (2^{k_s}-1)} (m_M)^j = n'_M$, we have less than $1 + (k_s + k_i) \cdot n'_M$ attribute occurrences of e'', we have less than $\sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j$ rule applications to generate t_s and thus $size_\Delta(t_s) < l_M \cdot \sum_{j=0}^{(k_s+k_i)\cdot n'_M} (c_M)^j < n_M$.

(d) For every $i \in U_i$, $1 \leq size_\Delta(t_i) < n_M$:

We have $size_\Delta(t_i) \geq 1$, because $p_2 \neq \varepsilon$ and thus there must be at least one rule application to calculate $nf(\Rightarrow_{\tilde{e},P_2}, i(p_1p_2)) \neq i(p_1p_2)$. The proof for $size_\Delta(t_i) < n_M$ is analogous to that in (c).

(e) For every $s \in U_s$, $1 \leq size_\Delta(\hat{t}_s) < n_M$:

We have $size_\Delta(\hat{t}_s) \geq 1$, because $nf(\Rightarrow_{\tilde{e},P_3}, s(p_1p_2))$ can only consist of output symbols and attribute occurrences $i'(p_1p_2)$ and thus cannot be equal to $s(p_1p_2)$. Again there is at least one rule application to calculate the normal form. We have $size_\Delta(\hat{t}_s) < n_M$, because the calculation of $nf(\Rightarrow_{\tilde{e},P_3}, s(p_1p_2))$ only takes place on the part e''' of the control tree. Since p is a longest path in $\tilde{e}$, its subpath p_3 with $|p_3| < |p_2p_3| \leq 2^{k_i} \cdot (2^{k_s} - 1)$ is a longest path of e'''. Now we can apply the same argumentation as in (c).

In the next step we have to check, whether the symbols $s \in U_s$ and $i \in U_i$ occur at least once in the desired patterns of t. We show the proof only for the occurrences of $s \in U_s$ in the tree $\hat{t}$ or in a tree $\hat{t}_{i'}$ for some $i' \in U_i$. The other cases can be treated analogous. The proof works by contradiction:

If there is an $s \in U_s$ such that s does not occur in $\hat{t}$ and not in $\hat{t}_{i'}$ for every $i' \in U_i$, then, by the definition of the patterns of t, the attribute occurrence $s(p_1)$ does not occur in $nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))$ and not in $nf(\Rightarrow_{\tilde{e},P_1}, i'(p_1))$ for every $i' \in U_i$. But the calculation of these normal forms are the only parts of the derivation d, in which the attribute occurrence $s(p_1)$ can be introduced into the derivation. Thus $s(p_1)$ cannot occur in d, in contradiction to $s \in attset(d, p_1) = attset(\tilde{e}, p_1)$.

The last item of this proof is to show that $t = \underline{tree}(1)$ and that for every $r \geq 0$, the property $\underline{tree}(r) \in L_{out}(M)$ holds.

We abbreviate the control tree, which is built up by repeating $r \geq 0$ times the pattern e'', by $\tilde{e}_r$:

$$\tilde{e}_r = e'[w/\underbrace{e''[w/\ldots e''[w/}_{r \text{ times}} e''' \underbrace{]\ldots]}_{r \text{ times}}]$$

Thus, in particular, $\tilde{e} = \tilde{e}_1$.

First we have to verify the following Statements (1a) and (1b) concerning the function $\underline{tree'}$:

(1a) For every $s \in U_s$, $r \geq 0$, $1 \leq l \leq r+1$, $\underline{tree'}(s, r, l) = nf(\Rightarrow_{\tilde{e}_r}, s(p_1 p_2^{l-1}))$.
(1b) For every $i \in U_i$, $r \geq 0$, $0 \leq l \leq r$, $\underline{tree'}(i, r, l) = nf(\Rightarrow_{\tilde{e}_r}, i(p_1 p_2^l))$.

Since M is noncircular, there must exist an order in which, for every $r \geq 0$, the attribute occurrences of the set $\{s(p_1 p_2^l) \mid s \in U_s, 0 \leq l \leq r\} \cup \{i(p_1 p_2^l) \mid i \in U_i, 0 \leq l \leq r\}$ can be evaluated. This order induces an order θ on the set $\{\underline{tree'}(s, r, l) \mid s \in U_s, 1 \leq l \leq r+1\} \cup \{\underline{tree'}(i, r, l) \mid i \in U_i, 0 \leq l \leq r\}$ of function calls and thus it is guaranteed that the recursive function $\underline{tree'}$ is well defined.

If, for example, the evaluation of $\underline{tree'}(s, r, l)$ forces us to evaluate $\underline{tree'}(s', r, l+1)$, then, for every $1 \leq l \leq r$, the attribute occurrence $s'(p_1 p_2^l)$ has to appear earlier than the attribute occurrence $s(p_1 p_2^{l-1})$ in an order of the above attribute occurrences. But this is guaranteed, because in this case t_s must contain a symbol s' (compare the definition of $\underline{tree'}$ in Theorem 3.4) and by the definition of t_s, we must have an attribute occurrence $s'(p_1 p_2)$ in $nf(\Rightarrow_{\tilde{e},P_2}, s(p_1))$. Hence, $s'(p_1 p_2)$ must be evaluated before $s(p_1)$ and thus, for every $1 \leq l \leq r$, $s'(p_1 p_2^l)$ must be evaluated before $s(p_1 p_2^{l-1})$.

Now we take an arbitrary such order θ of function calls which can be considered as a string of length $(r+1) \cdot card(U)$. Then we can prove the Statements (1a) and (1b) by finite (mathematical) induction on ν with $1 \leq \nu \leq (r+1) \cdot card(U)$, i.e., ν is a position in this string. Depending on the function call at position ν, we have to prove either the statement $\underline{tree'}(s, r, l) = nf(\Rightarrow_{\tilde{e}_r}, s(p_1 p_2^{l-1}))$ (if the ν–th function call is $\underline{tree'}(s, r, l)$) or the statement $\underline{tree'}(i, r, l) = nf(\Rightarrow_{\tilde{e}_r}, i(p_1 p_2^l))$ (if the ν–th function call is $\underline{tree'}(i, r, l)$). If we want to prove the statement for the function call at position ν in θ, then we can use the induction hypothesis which says that, for every function call at position ν' with $1 \leq \nu' < \nu$, the corresponding statement holds.

Case (a): The function call at position ν is $\underline{tree'}(s, r, l)$ with $s \in U_s$, $r \geq 0$, and $1 \leq l \leq r+1$. Thus we have to prove the statement $\underline{tree'}(s, r, l) = nf(\Rightarrow_{\tilde{e}_r}, s(p_1 p_2^{l-1}))$. There are two cases:

Case I: $1 \leq l \leq r$

$$
\begin{aligned}
& \underline{tree'}(s, r, l) \\
= \; & t_s[s'/\underline{tree'}(s', r, l+1) \; ; \; s' \in U_s, \; i'/\underline{tree'}(i', r, l-1) \; ; \; i' \in U_i] \\
& \qquad \text{(Definition of } \underline{tree'}\text{)} \\
= \; & t_s[s'/nf(\Rightarrow_{\bar{z}_r}, s'(p_1 p_2^l)) \; ; \; s' \in U_s, \; i'/nf(\Rightarrow_{\bar{z}_r}, i'(p_1 p_2^{l-1})) \; ; \; i' \in U_i] \\
& \qquad \text{(Induction Hypothesis for function} \\
& \qquad \text{calls with positions less than } \nu\text{)} \\
= \; & nf(\Rightarrow_{\bar{z}_r}, t_s[s'/s'(p_1 p_2^l) \; ; \; s' \in U_s, \; i'/i'(p_1 p_2^{l-1}) \; ; \; i' \in U_i]) \\
= \; & nf(\Rightarrow_{\bar{z}_r}, s(p_1 p_2^{l-1})) \qquad \text{(Calculation on the } l\text{-th occurrence of } e''\text{)}
\end{aligned}
$$

Case II: $l = r + 1$

$$
\begin{aligned}
& \underline{tree'}(s, r, r+1) \\
= \; & \hat{t}_s[i'/\underline{tree'}(i', r, r) \; ; \; i' \in U_i] \qquad \text{(Definition of } \underline{tree'}\text{)} \\
= \; & \hat{t}_s[i'/nf(\Rightarrow_{\bar{z}_r}, i'(p_1 p_2^r)) \; ; \; i' \in U_i] \qquad \text{(Induction Hypothesis for function} \\
& \qquad \text{calls with positions less than } \nu\text{)} \\
= \; & nf(\Rightarrow_{\bar{z}_r}, \hat{t}_s[i'/i'(p_1 p_2^r) \; ; \; i' \in U_i]) \\
= \; & nf(\Rightarrow_{\bar{z}_r}, s(p_1 p_2^r)) \qquad \text{(Calculation on } e'''\text{)}
\end{aligned}
$$

Case (b): The function call at position ν is $\underline{tree'}(i, r, l)$ with $i \in U_i$, $r \geq 0$, and $0 \leq l \leq r$. Thus we have to prove the statement $\underline{tree'}(i, r, l) = nf(\Rightarrow_{\bar{z}_r}, i(p_1 p_2^l))$. There are two cases:

Case I: $1 \leq l \leq r$

$$
\begin{aligned}
& \underline{tree'}(i, r, l) \\
= \; & t_i[s'/\underline{tree'}(s', r, l+1) \; ; \; s' \in U_s, \; i'/\underline{tree'}(i', r, l-1) \; ; \; i' \in U_i] \\
& \qquad \text{(Definition of } \underline{tree'}\text{)} \\
= \; & t_i[s'/nf(\Rightarrow_{\bar{z}_r}, s'(p_1 p_2^l)) \; ; \; s' \in U_s, \; i'/nf(\Rightarrow_{\bar{z}_r}, i'(p_1 p_2^{l-1})) \; ; \; i' \in U_i] \\
& \qquad \text{(Induction Hypothesis for function} \\
& \qquad \text{calls with positions less than } \nu\text{)} \\
= \; & nf(\Rightarrow_{\bar{z}_r}, t_i[s'/s'(p_1 p_2^l) \; ; \; s' \in U_s, \; i'/i'(p_1 p_2^{l-1}) \; ; \; i' \in U_i]) \\
= \; & nf(\Rightarrow_{\bar{z}_r}, i(p_1 p_2^l)) \qquad \text{(Calculation on the } l\text{-th occurrence of } e''\text{)}
\end{aligned}
$$

Case II: $l = 0$

$$
\begin{aligned}
& \underline{tree'}(i, r, 0) \\
= \; & \hat{t}_i[s'/\underline{tree'}(s', r, 1) \; ; \; s' \in U_s] \qquad \text{(Definition of } \underline{tree'}\text{)} \\
= \; & \hat{t}_i[s'/nf(\Rightarrow_{\bar{z}_r}, s'(p_1)) \; ; \; s' \in U_s] \qquad \text{(Induction Hypothesis for function} \\
& \qquad \text{calls with positions less than } \nu\text{)} \\
= \; & nf(\Rightarrow_{\bar{z}_r}, \hat{t}_i[s'/s'(p_1) \; ; \; s' \in U_s]) \\
= \; & nf(\Rightarrow_{\bar{z}_r}, i(p_1)) \qquad \text{(Calculation on } e'\text{)}
\end{aligned}
$$

Then we can prove for every $r \geq 0$ the equation $\underline{tree}(r) = nf(\Rightarrow_{\bar{z}_r}, s_{in}(e))$:

$$
\begin{aligned}
& \underline{tree}(r) \\
= \; & \hat{t}\,[s/\underline{tree}'(s,r,1)\ ;\ s \in U_s] && \text{(Definition of } \underline{tree}\text{)} \\
= \; & nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))[s(p_1)/s\ ;\ s \in U_s][s/\underline{tree}'(s,r,1)\ ;\ s \in U_s] \\
& && \text{(Definition of } \hat{t}\text{)} \\
= \; & nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))[s(p_1)/\underline{tree}'(s,r,1)\ ;\ s \in U_s] \\
= \; & nf(\Rightarrow_{\tilde{e},P_1}, s_{in}(\varepsilon))[s(p_1)/nf(\Rightarrow_{\tilde{e}_r}, s(p_1))\ ;\ s \in U_s] && \text{(Statement (1a))} \\
= \; & nf(\Rightarrow_{\tilde{e}_r,P_1}, s_{in}(\varepsilon))[s(p_1)/nf(\Rightarrow_{\tilde{e}_r}, s(p_1))\ ;\ s \in U_s] && \text{(Subterm } e' \text{ in } \tilde{e}_r \\
& && \text{unchanged)} \\
= \; & nf(\Rightarrow_{\tilde{e}_r}, s_{in}(\varepsilon))
\end{aligned}
$$

This equation has the two desired consequences that finish the proof of the pumping lemma:

- $\underline{tree}(1) = nf(\Rightarrow_{\tilde{e}_1}, s_{in}(\varepsilon)) = nf(\Rightarrow_{\tilde{e}}, s_{in}(\varepsilon)) = t.$
- For every $r \geq 0$, $\underline{tree}(r) = nf(\Rightarrow_{\tilde{e}_r}, s_{in}(\varepsilon)) \in L_{out}(M)$, because $\tau(M)(e'_r) = \underline{tree}(r)$ where $\tilde{e}_r = root(e'_r)$. □

We want to conclude this section with an observation concerning the requirements of the attributed tree transducers to be producing and visiting.

If we had dropped the "producing–condition", then the pumping process itself would not have been affected. But it would have been impossible to prove that the output patterns consist of at least one output symbol. In the next section we shall see that the applications of the pumping lemma demonstrated there, are no more feasible without this size–condition.

If we had dropped the "visiting–condition", then the proof of the pumping lemma itself would have been impossible. Since for the control tree $\tilde{e}$ and for every subpath p' of the chosen path p, $attset(\tilde{e}, p') \neq \emptyset$ cannot be guaranteed, the following construction is no more feasible.

4 Applications

Our pumping lemma is usable for the output language of every noncircular, producing, and visiting attributed tree transducer. But, if we take output languages which are constructed over an arbitrary output alphabet, then the application of the pumping lemma is very difficult. Hence we apply our pumping lemma only to output languages with monadic trees.

The following Theorem 4.1 is a specialised version of our pumping lemma for the case of monadic output languages. Observation 4.2 makes a statement about the number of occurrences of the output patterns in the trees $\underline{tree}(0)$ and $\underline{tree}(1)$ in the case of monadic output languages. We use this theorem and this observation in the following proofs.

4.1 Pumping lemma for monadic output languages

In order to simplify the study of this paper we state here a complete monadic version of the pumping lemma instead of giving only the additional conditions.

Theorem 4.1 Let $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ be an *si*–tree transducer with system $\mathcal{A} = (A, A_s, A_i)$ of attributes, pumping index n_M, and $\Delta = \Delta^{(1)} \cup \Delta^{(0)}$. For every $t \in L_{out}(M)$ with $size(t) \geq n_M$

- there exist three ranked alphabets
  - $(U_s, rank_{U_s})$ with $U_s \subseteq A_s$, $card(U_s) \geq 1$, and $rank_{U_s}(s) = 0$ for every $s \in U_s$,
  - $(U_i, rank_{U_i})$ with $U_i \subseteq A_i$ and $rank_{U_i}(i) = 0$ for every $i \in U_i$, and
  - $U = U_s \cup U_i$,
  - with $card(U_s) = card(U_i)$ or $card(U_s) = card(U_i) + 1$,
- there exist $u \in U_s$ and $\hat{t} \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $size_\Delta(\hat{t}) \geq 1$,
- for every $i \in U_i$, there exist $u \in \Delta^{(0)} \cup U_s$ and $\hat{t}_i \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $size_\Delta(\hat{t}_i) \geq 1$,
- for every $s \in U_s$, there exist $u \in U$ and $t_s \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $1 \leq size_\Delta(t_s) < n_M$,
- for every $i \in U_i$, there exist $u \in U$ and $t_i \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $1 \leq size_\Delta(t_i) < n_M$,
- for every $s \in U_s$, there exist $u \in \Delta^{(0)} \cup U_i$ and $\hat{t}_s \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ with $1 \leq size_\Delta(\hat{t}_s) < n_M$,

such that

- exactly one tree of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ is of type $T\langle\Delta\rangle$, such that
  if $card(U_s) = card(U_i)$, then there is exactly one $i \in U_i$ such that $\hat{t}_i \in T\langle\Delta\rangle$,
  if $card(U_s) = card(U_i) + 1$, then there is exactly one $s \in U_s$ such that $\hat{t}_s \in T\langle\Delta\rangle$,
- for every $s \in U_s$, the symbol s occurs in exactly one tree of the set $\{\hat{t}\} \cup \{\hat{t}_{i'} \mid i' \in U_i\}$,
- for every $s \in U_s$, the symbol s occurs in exactly one tree of the set $\{t_{s'} \mid s' \in U_s\} \cup \{t_{i'} \mid i' \in U_i\}$,
- for every $i \in U_i$, the symbol i occurs in exactly one tree of the set $\{t_{s'} \mid s' \in U_s\} \cup \{t_{i'} \mid i' \in U_i\}$,
- for every $i \in U_i$, the symbol i occurs in exactly one tree of the set $\{\hat{t}_{s'} \mid s' \in U_s\}$,

such that $t = \underline{tree}(1)$ and for every $r \geq 0$, the tree $\underline{tree}(r) \in L_{out}(M)$. The function

$\underline{tree} : \mathbb{N} \longrightarrow T\langle\Delta\rangle$

is for every $r \geq 0$ defined by $\underline{tree}(r) = \hat{t}\,[s/\underline{tree'}(s,r,1)\ ;\ s \in U_s]$, where the partial function

$\underline{tree'} : U \times \mathbb{N} \times \mathbb{N} - \!\!\rightarrow T\langle\Delta\rangle$ is defined as follows:

For every $s \in U_s$ and $r \geq 0$, if $l \in [r]$,

$\underline{tree'}(s,r,l) = t_s[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s,\ i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.

For every $s \in U_s$ and $r \geq 0$, if $l = r+1$,

$\underline{tree'}(s,r,l) = \hat{t}_s[i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.

For every $i \in U_i$ and $r \geq 0$, if $l \in [r]$,

$\underline{tree'}(i,r,l) = t_i[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s,\ i'/\underline{tree'}(i',r,l-1)\ ;\ i' \in U_i]$.

For every $i \in U_i$ and $r \geq 0$, if $l = 0$,

$\underline{tree'}(i,r,l) = \hat{t}_i[s'/\underline{tree'}(s',r,l+1)\ ;\ s' \in U_s]$.

Proof. We only have to prove the additional conditions of the pumping lemma. The proof is based on the proof of Theorem 3.4. Thus we make use of some notions which were introduced there.

We first prove the correctness of the substitutions of "occurs in a tree" in Theorem 3.4 by "occurs in exactly one tree". We show the proof only for the occurrence of s in the tree $\hat{t}$ or in a tree $\hat{t}_{i'}$. The other cases can be treated analogous. The proof works by contradiction:

Assume that there is an $s \in U_s$ such that s occurs in at least two different trees of the set $\{\hat{t}\} \cup \{\hat{t}_{i'} \mid i' \in U_i\}$. Then, by the definition of the patterns of t, the attribute occurrence $s(p_1)$ occurs in two different normal forms of $nf(\Rightarrow_{\hat{\varepsilon},P_1}, s_{in}(\varepsilon))$ and $nf(\Rightarrow_{\hat{\varepsilon},P_1}, i'(p_1))$ for $i' \in U_i$. The calculation of these normal forms correspond to different parts of the derivation $s_{in}(\varepsilon) \Rightarrow_{\hat{\varepsilon}}^{+} t$. Thus $s(p_1)$ occurs in two different sentential forms of the derivation $s_{in}(\varepsilon) \Rightarrow_{\hat{\varepsilon}}^{+} t$. There must exist $t_1, t_2 \in (\Delta^{(1)})^+$ with $s_{in}(\varepsilon) \Rightarrow_{\hat{\varepsilon}}^{+} t_1 s(p_1) \Rightarrow_{\hat{\varepsilon}}^{+} t_1 t_2 s(p_1) \Rightarrow_{\hat{\varepsilon}}^{+} t$. Consequently, M is circular, which is a contradiction. The conditions that

- there exist $u \in U_s$ and $\hat{t} \in T\langle\Delta^{(1)} \cup \{u\}\rangle$,
- for every $i \in U_i$, there exist $u \in \Delta^{(0)} \cup U_s$ and $\hat{t}_i \in T\langle\Delta^{(1)} \cup \{u\}\rangle$, and
- for every $s \in U_s$, there exist $u \in \Delta^{(0)} \cup U_i$ and $\hat{t}_s \in T\langle\Delta^{(1)} \cup \{u\}\rangle$

are direct consequences of the pumping lemma, because Δ is monadic.

- For every $s \in U_s$, there exist $u \in U$ and $t_s \in T\langle\Delta^{(1)} \cup \{u\}\rangle$ and
- for every $i \in U_i$, there exist $u \in U$ and $t_i \in T\langle\Delta^{(1)} \cup \{u\}\rangle$,

because each of the $card(U)$ symbols of U occurs in exactly one of the $card(U)$ trees of the set $\{t_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\}$, and because each of these trees can contain at most one (and thus exactly one) of the symbols.

We know that each of the $card(U_i)$ symbols of U_i occurs in exactly one of the $card(U_s)$ trees of the set $\{\hat{t}_s \mid s \in U_s\}$, and that each of these trees can contain at most one of the symbols. Thus we must have $card(U_s) \geq card(U_i)$. We also know that each of the $card(U_s)$ symbols of U_s occurs in exactly one of the $card(U_i) + 1$ trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\}$, and that $\hat{t}$ contains exactly one and each of the other trees can contain at most one of the symbols. Thus we must have $card(U_i) \geq card(U_s) - 1$. We can conclude that $card(U_s) = card(U_i)$ or $card(U_s) = card(U_i) + 1$ holds.

If $card(U_s) = card(U_i)$, then every tree $\hat{t}_s$ contains exactly one of the symbols of U_i and every tree $\hat{t}_i$ except one of them contains exactly one of the symbols of U_s. Thus there is exactly one $i \in U_i$ with $\hat{t}_i \in T\langle\Delta\rangle$.

If $card(U_s) = card(U_i)+1$, then every tree $\hat{t}_i$ contains exactly one of the symbols of U_s and every tree $\hat{t}_s$ except one of them contains exactly one of the symbols of U_i. Thus there is exactly one $s \in U_s$ with $\hat{t}_s \in T\langle\Delta\rangle$. □

Observation 4.2 Let $M = (A, \Delta, \Sigma, s_{in}, root, R)$ be an *si*-tree transducer with system $A = (A, A_s, A_i)$ of attributes and $\Delta = \Delta^{(1)} \cup \Delta^{(0)}$. Then in Theorem 4.1,

1. $\underline{tree}(0)$ is built up, using each of the trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ exactly once and
2. $t = \underline{tree}(1)$ is built up, using each of the trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ exactly once.

Proof. Again we make use of some notions which were introduced in the proof of Theorem 3.4.

(a) The tree $\hat{t}$ is used exactly once in $\underline{tree}(0)$ and $\underline{tree}(1)$, because $\hat{t}$ is introduced calling the function $\underline{tree'}$ the first time and nowhere else.

(b) The argumentation for the statement that the trees of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ are used at most once in $\underline{tree}(0)$ works as follows by contradiction:

W.l.o.g. we assume that a tree $\hat{t}_i$ is used twice (or more than twice). Then the calculation of $nf(\Rightarrow_{\hat{\varepsilon},P_1}, i(p_1))$ corresponds to different parts of the derivation $s_{in}(\varepsilon) \Rightarrow^+_{\hat{\varepsilon}_0} \underline{tree}(0)$. Thus $i(p_1)$ occurs in different sentential forms of the derivation $s_{in}(\varepsilon) \Rightarrow^+_{\hat{\varepsilon}_0} \underline{tree}(0)$. There must exist $t_1, t_2 \in (\Delta^{(1)})^+$ with $s_{in}(\varepsilon) \Rightarrow^+_{\hat{\varepsilon}_0} t_1 i(p_1) \Rightarrow^+_{\hat{\varepsilon}_0} t_1 t_2 i(p_1) \Rightarrow^+_{\hat{\varepsilon}_0} \underline{tree}(0)$. Consequently, M is circular, which is a contradiction.

(c) The same argumentation can be applied for the proof of the statement that the trees of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ are used at most once in $\underline{tree}(1)$.

(d) The argumentation for the statement that the trees of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ are used at least once in $\underline{tree}(0)$ works as follows by contradiction:

By Theorem 4.1 we have $card(U_s) = card(U_i)$ or $card(U_s) = card(U_i) + 1$. We show the proof only for the case $card(U_s) = card(U_i)$. The other case can be proved analogous.

We let $k = card(U_s) = card(U_i)$, $U_s = \{s_1, \ldots, s_k\}$, and $U_i = \{i_1, \ldots, i_k\}$.

Assume that not all of the desired output patterns occur in $\underline{tree}(0)$. The number of used trees $\hat{t}_i$ with $i \in U_i$ and the number of used trees $\hat{t}_s$ with $s \in U_s$ has to be equal, because the process of building up $\underline{tree}(0)$ starts with $\hat{t}$, it must end with the only tree $\hat{t}_i \in T\langle\Delta\rangle$ by Theorem 4.1, and the use of trees $\hat{t}_i$ with $i \in U_i$ and of trees $\hat{t}_s$ with $s \in U_s$ must alternate, as can be seen observing the function $\underline{tree'}$.

Thus we can assume that there is a $k' \in [k-1]$, such that only the patterns $\hat{t}_{i_1}, \ldots, \hat{t}_{i_{k'}}, \hat{t}_{s_1}, \ldots, \hat{t}_{s_{k'}}$ occur in $\underline{tree}(0)$ (possibly by renaming the trees). We construct a circularity in $\tilde{e}_0$ with the remaining patterns $\hat{t}_{i_{k'+1}}, \ldots, \hat{t}_{i_k}, \hat{t}_{s_{k'+1}}, \ldots, \hat{t}_{s_k}$ which can not be of type $T\langle\Delta\rangle$, as follows:

Because of Theorem 4.1 and because the symbols $s_1, \ldots, s_{k'}, i_1, \ldots, i_{k'}$ must occur in the patterns which are used to construct $\underline{tree}(0)$, we know:

- For every j with $k'+1 \leq j \leq k$, the tree $\hat{t}_{i_j} \in T\langle\Delta^{(1)} \cup \{s_{k'+1}, \ldots, s_k\}\rangle$,
- for every j with $k'+1 \leq j \leq k$, the tree $\hat{t}_{s_j} \in T\langle\Delta^{(1)} \cup \{i_{k'+1}, \ldots, i_k\}\rangle$,
- and every symbol $s_{k'+1}, \ldots, s_k, i_{k'+1}, \ldots, i_k$ must occur in exactly one tree of the set $\{\hat{t}_{i_{k'+1}}, \ldots, \hat{t}_{i_k}, \hat{t}_{s_{k'+1}}, \ldots, \hat{t}_{s_k}\}$.

Thus, possibly by renaming the trees, there must exist $k'' \in [k-k']$ with:

- For every j with $k'+1 \leq j \leq k'+k''$, the tree $\hat{t}_{i_j} \in T\langle\Delta^{(1)} \cup \{s_j\}\rangle$,
- for every j with $k'+1 \leq j \leq k'+k''-1$, the tree $\hat{t}_{s_j} \in T\langle\Delta^{(1)} \cup \{i_{j+1}\}\rangle$,
- and $\hat{t}_{s_{k'+k''}} \in T\langle\Delta^{(1)} \cup \{i_{k'+1}\}\rangle$.

By the definition of the patterns in the proof of Theorem 3.4 we know that these patterns correspond to normal forms of certain attribute occurrences and we can construct a derivation on the control tree $\tilde{e}_0$ as follows:

$$
\begin{array}{ll}
 & i_{k'+1}(p_1) \\
\Rightarrow^+_{\tilde{e}_0} & \hat{t}_{i_{k'+1}} s_{k'+1}(p_1) \\
\Rightarrow^+_{\tilde{e}_0} & \hat{t}_{i_{k'+1}} \hat{t}_{s_{k'+1}} i_{k'+2}(p_1) \\
\vdots & \\
\Rightarrow^+_{\tilde{e}_0} & \hat{t}_{i_{k'+1}} \hat{t}_{s_{k'+1}} \ldots \hat{t}_{i_{k'+k''}} s_{k'+k''}(p_1) \\
\Rightarrow^+_{\tilde{e}_0} & \hat{t}_{i_{k'+1}} \hat{t}_{s_{k'+1}} \ldots \hat{t}_{i_{k'+k''}} \hat{t}_{s_{k'+k''}} i_{k'+1}(p_1)
\end{array}
$$

We can conclude that M is circular, which is a contradiction. An example situation which would be a consequence of the assumption that not all of the desired output patterns occur in $\underline{tree}(0)$ is shown in Figure 10.

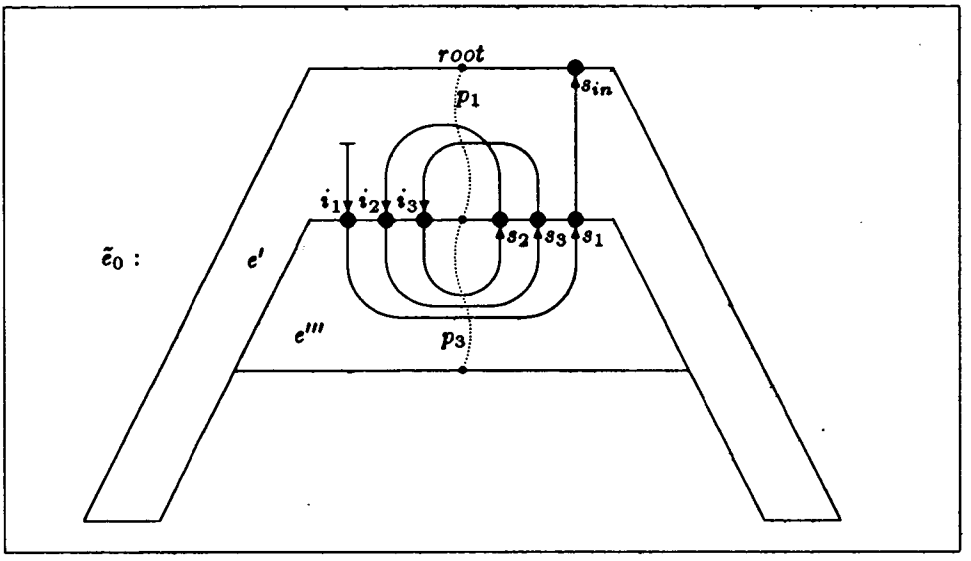


Figure 10: Circularity in $\tilde{e}_0$ with $k = 3$, $k' = 1$ and $k'' = 2$.

(e) The trees of the set $\{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ are used at least once in $\underline{tree}(1)$, because these patterns correspond to parts of the derivation $s_{in}(e) \Rightarrow_{\tilde{e}}^{+} t = \underline{tree}(1)$ by the definition of the output patterns in the proof of Theorem 3.4. □

4.2 Arithmetic Proof

It is known from Lemma 4.1 of [Fül81] that, if M is an attributed tree transducer and if $\tau(M)(e) = t$ for an input tree e and an output tree t, then there is a constant $c > 0$ such that $height(t) \leq c \cdot size(e)$ holds. Thus, there cannot exist an attributed tree transducer M, which calculates the tree transformation $\tau(M) : T\langle\{\gamma^{(1)}, \alpha^{(0)}\}\rangle \longrightarrow T\langle\{B^{(1)}, E^{(0)}\}\rangle$ with $\tau(M)(\gamma^n\alpha) = B^{2^n}E$ for every $n \geq 0$. We only mention here that there is a macro tree transducer (cf. Example 4.3 of [EV85]) which calculates this tree transformation.

If we do not restrict the input trees to be monadic trees, then the lemma of Fülöp says nothing about whether an attributed tree transducer M' exists computing the tree transformation $\tau(M') : T\langle\Sigma\rangle \longrightarrow T\langle\{B^{(1)}, E^{(0)}\}\rangle$ with $L_{out}(M') = \{B^{2^n}E \mid n \geq 0\}$. Such a producing and visiting attributed tree transducer cannot exist, because we can use our pumping lemma to prove that $\{B^{2^n}E \mid n \geq 0\} \notin SIT_{out}$ holds.

We call the following kind of proof arithmetic proof, because we use arithmetic arguments while applying the pumping lemma.

Theorem 4.3 $\{B^{2^n}E \mid n \geq 0\} \notin SIT_{out}$

Proof. Assume that there is an *si*-tree transducer $M = (\mathcal{A}, \Delta, \Sigma, s_{in}, root, R)$ with system $\mathcal{A} = (A, A_s, A_i)$ of attributes and $L_{out}(M) = \{B^{2^n}E \mid n \geq 0\}$. By Theorem 4.1, for every $t \in L_{out}(M)$ with $size(t) \geq n_M$, where $n_M \geq 1$ is the pumping index of M, certain properties hold. Consider $t = B^{2^{n_M \cdot card(A)}}E$; clearly, $size(t) \geq n_M$.

According to Theorem 4.1 there exist $U_s \subseteq A_s$ with $card(U_s) \geq 1$, $U_i \subseteq A_i$, a tree $\hat{t}$, trees $\hat{t}_i, t_i$ for every $i \in U_i$, and trees $\hat{t}_s, t_s$ for every $s \in U_s$ fulfilling the conditions of Theorem 4.1, such that $t = \underline{tree}(1)$.

$t = \underline{tree}(1)$ is built up, using each of the trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ exactly once, because of Observation 4.2. $\underline{tree}(0)$ is built up, using each of the trees of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\}$ exactly once, because of Observation 4.2.

Thus we can estimate $size(\underline{tree}(0))$ with the size conditions of Theorem 4.1 as follows:

$$
\begin{aligned}
& size(\underline{tree}(0)) \\
= \;& size(\underline{tree}(1)) - \textstyle\sum_{s \in U_s} size_\Delta(t_s) - \sum_{i \in U_i} size_\Delta(t_i) \\
\geq \;& 2^{n_M \cdot card(A)} + 1 - (n_M - 1) \cdot (card(U_s) + card(U_i)) && (size_\Delta(t_s) \leq n_M - 1, \\
& && \; size_\Delta(t_i) \leq n_M - 1) \\
\geq \;& 2^{n_M \cdot card(A)} + 1 - (n_M - 1) \cdot (card(A_s) + card(A_i)) \\
= \;& 2^{n_M \cdot card(A)} + 1 - (n_M - 1) \cdot card(A) \\
\geq \;& 2^{n_M \cdot card(A)} + 1 - (n_M \cdot card(A) - 1) \\
> \;& 2^{n_M \cdot card(A)} + 1 - 2^{n_M \cdot card(A) - 1} \\
= \;& 2^{n_M \cdot card(A) - 1} + 1, \qquad \text{and}
\end{aligned}
$$

$$
\begin{aligned}
& size(\underline{tree}(0)) \\
= \;& size(\underline{tree}(1)) - \textstyle\sum_{s \in U_s} size_\Delta(t_s) - \sum_{i \in U_i} size_\Delta(t_i) \\
\leq \;& 2^{n_M \cdot card(A)} + 1 - (card(U_s) + card(U_i)) && (size_\Delta(t_s) \geq 1, \\
& && \; size_\Delta(t_i) \geq 1) \\
< \;& 2^{n_M \cdot card(A)} + 1.
\end{aligned}
$$

Note that the requirement of M to be producing is necessary for this part of the proof.

Thus $2^{n_M \cdot card(A) - 1} + 1 < size(\underline{tree}(0)) < 2^{n_M \cdot card(A)} + 1$ and therefore $\underline{tree}(0) \notin L_{out}(M) = \{B^{2^n}E \mid n \geq 0\}$, contradicting the assumption. □

4.3 Structural Proof

In contrast to the (easier) arithmetic proofs, we want to demonstrate here, how structural properties of a certain output language can be used while applying the pumping lemma for attributed tree transducers. We use the results of this subsection to present a hierarchy for attributed tree transducers with bounded number of attributes.

Lemma 4.4 For every $k \geq 1$, $\{(BD^n)^{2k+1}E \mid n \geq 0\} \notin S_{(k)}I_{(k)}T_{out}$.

Proof. Let $k \geq 1$. Assume that there is an *si*– tree transducer $M = (A, \Delta, \Sigma, s_{in}, root, R)$ with system $A = (A, A_s, A_i)$ of attributes, $L_{out}(M) = \{(BD^n)^{2k+1}E \mid n \geq 0\}$ and with k synthesised attributes and k inherited attributes. By Theorem 4.1, for every $t \in L_{out}(M)$ with $size(t) \geq n_M$, where $n_M \geq 1$ is the pumping index of M, certain properties hold. Consider $t = (BD^{n_M \cdot (2k+1)})^{2k+1}E$; clearly $size(t) \geq n_M$.

According to Theorem 4.1 there exist $U_s \subseteq A_s$ with $card(U_s) \geq 1$, $U_i \subseteq A_i$ with $card(U_s) = card(U_i)$ or $card(U_s) = card(U_i) + 1$. Additionally, there exist a pattern $\hat{t}$, patterns $\hat{t}_i, t_i$ for every $i \in U_i$, and patterns $\hat{t}_s, t_s$ for every $s \in U_s$ fulfilling the conditions of Theorem 4.1, such that $t = \underline{tree}(1)$.

$t = \underline{tree}(1)$ is built up, using each of the patterns of the set $\{\hat{t}\} \cup \{\hat{t}_i \mid i \in U_i\} \cup \{\hat{t}_s \mid s \in U_s\} \cup \{t_i \mid i \in U_i\} \cup \{t_s \mid s \in U_s\}$ exactly once, because of Observation 4.2. In the following, we simply identify these patterns with the sequence of their output symbols from the root to the leaf by dropping the symbols $s \in U_s$ and $i \in U_i$. This notation is slightly inaccurate, but easier to read. We let $k_1 = card(U_s)$, $k_2 = card(U_i)$, $U_s = \{s_1, \ldots, s_{k_1}\}$, and $U_i = \{i_1, \ldots, i_{k_2}\}$.

Case 1: $k_1 = k_2$

In this case we can represent t as follows, where for every $l \in [k_1]$, $t^{(l)}$ is a sequence of patterns taken from the $3k_1$ patterns $t_{s_1}, \ldots, t_{s_{k_1}}, t_{i_1}, \ldots, t_{i_{k_1}}, \hat{t}_{s_1}, \ldots, \hat{t}_{s_{k_1}}$:

$$t = \underline{tree}(1) = \hat{t}\, t^{(1)}\, \hat{t}_{i_1}\, t^{(2)}\, \hat{t}_{i_2} \ldots t^{(k_1)}\, \hat{t}_{i_{k_1}}$$

For every $l \in [k_1]$, the tree $t^{(l)}$ is built up from at least one pattern. It is constructed from at most $2k_1 + 1$ patterns, if the other trees $t^{(l')}$ are built up from exactly one pattern, because each pattern can only be used once, according to Observation 4.2. Since for every $j \in [k_1]$, $1 \leq size_\Delta(t_{s_j}) < n_M$, $1 \leq size_\Delta(t_{i_j}) < n_M$, and $1 \leq size_\Delta(\hat{t}_{s_j}) < n_M$, we know for every $l \in [k_1]$:

$$1 \leq size_\Delta(t^{(l)}) < (2k_1 + 1) \cdot n_M \leq (2k + 1) \cdot n_M$$

Thus every sequence $t^{(l)}$ can overlap at most two parts of successive symbols D in $\underline{tree}(1)$. The k_1 sequences together can overlap at most $2k_1 \leq 2k$ parts of successive symbols D in $\underline{tree}(1)$. Since there are $2k+1$ parts of successive symbols D in $\underline{tree}(1)$, there must exist one subsequence

$$b = BD^{n_M \cdot (2k+1)}B \quad \text{or} \quad b = BD^{n_M \cdot (2k+1)}E$$

of $\underline{tree}(1)$ which completely is a part of $\hat{t}$ or of a tree $\hat{t}_{i_l}$ for some $l \in [k_1]$.

We present an example situation with $k = k_1 = 2$ and with a subsequence $b = B\,D \ldots D\,B$ in $\hat{t}_{i_1}$:

$$\underbrace{B\,D\ldots}_{\hat{t}}\underbrace{\ldots D\,B\,D\ldots}_{t^{(1)}}\underbrace{\ldots D\,\overbrace{B\,D\ldots\ldots D\,B}^{b}\,D\ldots}_{\hat{t}_{i_1}}\underbrace{\ldots D\,B\,D\ldots}_{t^{(2)}}\underbrace{\ldots D\,E}_{\hat{t}_{i_2}}$$

This subsequence b must appear in $\underline{tree}(0)$, because $\underline{tree}(0)$ is built up, using each of the patterns $\hat{t}, \hat{t}_{i_1}, \ldots, \hat{t}_{i_{k_1}}, \hat{t}_{s_1}, \ldots, \hat{t}_{s_{k_1}}$ exactly once by Observation 4.2.

(It is not important for this proof that the relative positions of these patterns can change from $\underline{tree}(1)$ to $\underline{tree}(0)$.)

The patterns $t_{s_1}, \ldots, t_{s_{k_1}}, t_{i_1}, \ldots, t_{i_{k_1}}$ do not appear in $\underline{tree}(0)$ any more. These patterns can only contain symbols D and B, because $size_\Delta(\hat{t}_{i_{k_1}}) \geq 1$ and thus the last symbol E must be a part of $\hat{t}_{i_{k_1}}$.

If there is a symbol B in one of these patterns, then the number of symbols B decreases and thus $\underline{tree}(0) \notin \{(BD^n)^{2k+1}E \mid n \geq 0\}$, contradicting the assumption.

If these patterns only contain symbols D, then the number of symbols D decreases and the number of symbols B is constant. Thus we must have a block $b' = B\ D \ldots D\ B$ or $b' = B\ D \ldots D\ E$ with less than $n_M \cdot (2k+1)$ successive symbols D. Since b and b' have a different number of successive symbols D, we have $\underline{tree}(0) \notin \{(BD^n)^{2k+1}E \mid n \geq 0\}$, contradicting the assumption.

Note that the last steps of the above argumentation need the requirement of M to be producing.

Case 2: $k_1 = k_2 + 1$

In this case we can represent t as follows, where for every $l \in [k_1]$, $t^{(l)}$ is a sequence of patterns taken from the $3k_1 - 1$ patterns $t_{s_1}, \ldots, t_{s_{k_1}}, t_{i_1}, \ldots, t_{i_{k_1-1}}, \hat{t}_{s_1}, \ldots, \hat{t}_{s_{k_1}}$:

$$t = \underline{tree}(1) = \hat{t}\, t^{(1)}\, \hat{t}_{i_1}\, t^{(2)}\, \hat{t}_{i_2} \ldots t^{(k_1-1)}\, \hat{t}_{i_{k_1-1}}\, t^{(k_1)}$$

For every $l \in [k_1 - 1]$, the tree $t^{(l)}$ is built up from at least one pattern, and $t^{(k_1)}$ is built up from at least two patterns. For every $l \in [k_1 - 1]$, the tree $t^{(l)}$ is constructed from at most $2k_1 - 1$ patterns, if the other trees $t^{(l')}$ with $l' \in [k_1 - 1]$ are built up from exactly one pattern and $t^{(k_1)}$ is built up from exactly two patterns, because each pattern can only be used once, according to Observation 4.2. The tree $t^{(k_1)}$ is constructed from at most $2k_1$ patterns, if the other trees $t^{(l')}$ with $l' \in [k_1 - 1]$ are built up from exactly one pattern. Then we can apply the same argumentation as in Case 1. □

Lemma 4.5 For every $k \geq 1$, $\{(BD^n)^{2k}E \mid n \geq 0\} \notin S_{(k)}I_{(k-1)}T_{out}$.

Proof. The proof of this lemma is analogous to the proof of Lemma 4.4. □

The following lemma completes the requirements for the desired hierarchy of attributed tree transducers.

Lemma 4.6

- For every $k \geq 1$, $\{(BD^n)^{2k}E \mid n \geq 0\} \in S_{(k)}I_{(k)}T_{out}$.
- For every $k \geq 0$, $\{(BD^n)^{2k+1}E \mid n \geq 0\} \in S_{(k+1)}I_{(k)}T_{out}$.

Proof. For every $k \geq 1$ we define an *si*-tree transducer $M^{(2k)} = (\mathcal{A}^{(2k)}, \Delta, \Sigma, s_1, root, R^{(2k)})$ with:
$\Delta = \{B^{(1)}, D^{(1)}, E^{(0)}\}$,
$\Sigma = \{\gamma^{(1)}, \alpha^{(0)}\}$,
$\mathcal{A}^{(2k)} = (A^{(2k)}, A_s^{(2k)}, A_i^{(2k)})$ with $A^{(2k)} = \{s_1, \ldots, s_k, i_1, \ldots, i_k\}$,

$A_s^{(2k)} = \{s_1, \ldots, s_k\}$, and $A_i^{(2k)} = \{i_1, \ldots, i_k\}$, and
$R^{(2k)} = R_{root}^{(2k)} \cup R_\gamma^{(2k)} \cup R_\alpha^{(2k)}$ with:

$$\begin{array}{lcll} R_{root}^{(2k)} & = & \{s_1(z) \rightarrow B\, s_1(z1) & \} \cup \\ & & \{i_j(z1) \rightarrow B\, s_{j+1}(z1) & \mid j \in [k-1]\} \cup \\ & & \{i_k(z1) \rightarrow E & \} \\ R_\gamma^{(2k)} & = & \{s_j(z) \rightarrow D\, s_j(z1) & \mid j \in [k]\} \cup \\ & & \{i_j(z1) \rightarrow D\, i_j(z) & \mid j \in [k]\} \\ R_\alpha^{(2k)} & = & \{s_j(z) \rightarrow B\, i_j(z) & \mid j \in [k]\} \end{array}$$

For every $k \geq 0$ we define an *si*–tree transducer
$M^{(2k+1)} = (\mathcal{A}^{(2k+1)}, \Delta, \Sigma, s_1, root, R^{(2k+1)})$ with:
$\Delta = \{B^{(1)}, D^{(1)}, E^{(0)}\}$,
$\Sigma = \{\gamma^{(1)}, \alpha^{(0)}\}$,
$\mathcal{A}^{(2k+1)} = (A^{(2k+1)}, A_s^{(2k+1)}, A_i^{(2k+1)})$ with $A^{(2k+1)} = \{s_1, \ldots, s_{k+1}, i_1, \ldots, i_k\}$,
$A_s^{(2k+1)} = \{s_1, \ldots, s_{k+1}\}$, and $A_i^{(2k+1)} = \{i_1, \ldots, i_k\}$, and
$R^{(2k+1)} = R_{root}^{(2k+1)} \cup R_\gamma^{(2k+1)} \cup R_\alpha^{(2k+1)}$ with:

$$\begin{array}{lcll} R_{root}^{(2k+1)} & = & \{s_1(z) \rightarrow B\, s_1(z1) & \} \cup \\ & & \{i_j(z1) \rightarrow B\, s_{j+1}(z1) & \mid j \in [k]\} \\ R_\gamma^{(2k+1)} & = & \{s_j(z) \rightarrow D\, s_j(z1) & \mid j \in [k+1]\} \cup \\ & & \{i_j(z1) \rightarrow D\, i_j(z) & \mid j \in [k]\} \\ R_\alpha^{(2k+1)} & = & \{s_j(z) \rightarrow B\, i_j(z) & \mid j \in [k]\} \cup \\ & & \{s_{k+1}(z) \rightarrow E & \} \end{array}$$

Clearly, for every $k \geq 1$, $L_{out}(M^{(k)}) = \{(BD^n)^k E \mid n \geq 0\}$. Thus we can conclude the statement of the lemma. □

From Lemma 4.4, Lemma 4.5, and Lemma 4.6 we gain the following hierarchy for classes of output languages of *si*–tree transducers with bounded number of attributes:

Theorem 4.7 $S_{(k)}I_{(k-1)}T_{out} \subset S_{(k)}I_{(k)}T_{out} \subset S_{(k+1)}I_{(k)}T_{out}$, for every $k \geq 1$. □

This theorem can be transformed into the following theorem that presents a hierarchy for classes of tree transformations of *si*–tree transducers with bounded number of attributes (cf. also Figure 11):

Theorem 4.8 $S_{(k)}I_{(k-1)}T \subset S_{(k)}I_{(k)}T \subset S_{(k+1)}I_{(k)}T$, for every $k \geq 1$. □

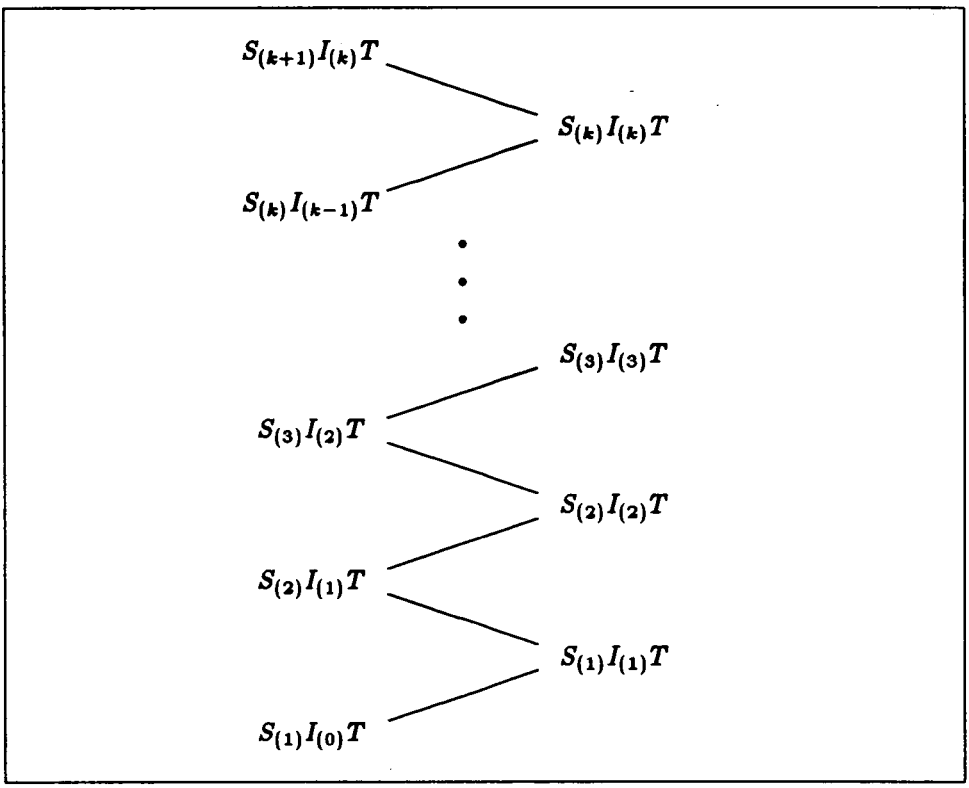


Figure 11: Hierarchy of tree transformation classes.

5 Summary and further research topics

In this paper we have developed a pumping lemma for output languages of noncircular, producing, and visiting attributed tree transducers. We have restricted the applications of the pumping lemma to monadic output languages yielding two results for attributed tree transducers. In particular,

- we have proved that the language $\{B^{2^n}E \mid n \geq 0\}$ can be no output language of a noncircular, producing, and visiting attributed tree transducer, using our pumping lemma together with arithmetic properties of this language, and
- we have proved a hierarchy for noncircular, producing, and visiting attributed tree transducers with bounded number of attributes, using our pumping lemma together with structural properties of languages.

There are several further research topics in the area of pumping lemmata for attributed tree transducers and other kinds of tree transducers:

- Are there non-monadic languages which can be proved not to be output languages of attributed tree transducers with the help of our pumping lemma in

a justifiable expense? In the case of non-monadic languages the proofs become very much harder, because the output patterns can no more be treated like concatenated strings as in the proof of Lemma 4.4. The output patterns are non-monadic trees which occur in a non- monadic output tree. The main problem is to find a complete case analysis for all possibilities to construct an output tree with output patterns. Then we have to derive a contradiction for every case. Additionally we have the difficulty that output patterns can occur more than once in an output tree $\underline{tree}(1)$, as can be seen in Figure 9. Thus in the case of non- monadic output languages there is no helping observation as Observation 4.2.

- A similar pumping lemma as for attributed tree transducers can be developed for macro tree transducers (cf. [EV85]). It will be introduced in another paper which is in preparation (cf. [Küh94]). Is it possible to use this pumping lemma in a proof that the difference set SI_fT–S_fT of subclasses of macro attributed tree transducers is not empty, as it was conjectured in [KV94]?
- As next step it should be possible to construct a pumping lemma for macro attributed tree transducers (cf. [KV94]) as combination of the lemmata for attributed tree transducers and macro tree transducers. Then as special case of it we have a pumping lemma for the class SI_fT and perhaps it is possible to prove that the difference set S_fT–SI_fT is not empty, as it also was conjectured in [KV94].

Acknowledgement
We would like to thank Zoltan Fülöp for carefully reading an earlier version of this paper and for making useful suggestions on its contents.

References

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

[BM82] C. Bader and A. Moura. A generalization of Ogden's lemma. *J. Assoc. Comput. Mach.*, 29:404–407, 1982.

[Boc76] G. Bochmann. Semantic evaluation from left to right. *Comm. Assoc. Comput. Sci.*, 19:55–62, 1976.

[BPS61] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Z. Phonetik. Sprach. Komm.*, 14:143–172, 1961.

[BS86a] R. Boonyavatana and G. Slutzki. A generalized Ogden's lemma for linear context-free languages. *Bulletin of the EATCS*, 28:20–26, 1986.

[BS86b] R. Boonyavatana and G. Slutzki. Ogden's lemma for nonterminal bounded languages. *RAIRO*, 20:457–471, 1986.

[Ems91] K. Emser–Loock. Integration von attributierten Grammatiken und primitiv–rekursiven Programmschemata. Master Thesis, RWTH Aachen, 1991.

[Eng75] J. Engelfriet. Bottom–up and top–down tree transformations — a comparison. *Math. Syst. Theory*, 9:198–231, 1975.

[EPR81] A. Ehrenfeucht, R. Parikh, and G. Rozenberg. Pumping lemmas for regular sets. *SIAM J. Comput.*, 10:536–541, 1981.

[ERS80] J. Engelfriet, G. Rozenberg, and G. Slutzki. Tree transducers, L systems, and two–way machines. *J. Comput. Syst. Sci.*, 20:150–202, 1980.

[Ési80] Z. Ésik. Decidability results concerning tree transducers. *Acta Cybernetica*, 5:1–20, 1980.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31:71–145, 1985.

[Fül81] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[FV91] Z. Fülöp and S. Vágvölgyi. Attributed tree transducers cannot induce all deterministic bottom–up tree transformations. manuscript, to appear in Information and Computation, 1991.

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25:355–423, 1988.

[GS83] F. Gécseg and M. Steinby. *Tree Automata.* Akademiai Kiado, Budapest, 1983.

[Hab89] A. Habel. *Hyperedge replacement: grammars and languages.* PhD thesis, University of Bremen, 1989.

[Hin90] F. Hinz. *Erzeugung von Bildsprachen durch Chomsky–Grammatiken — Entscheidbarkeits– und Komplexitätsfragen.* PhD thesis, RWTH Aachen, 1990.

[Knu68] D.E. Knuth. Semantics of context–free languages. *Math. Syst. Theory*, 2:127–145, 1968.

[Kre79] H.-J. Kreowski. A pumping lemma for context–free graph languages. *Lect. Not. Comp. Sci.*, 73:270–283, 1979.

[Küh94] A. Kühnemann. A pumping lemma for output languages of macro tree transducers. Technical report, Technical University of Dresden, 1994. in preparation.

[Kus91] S. Kuske. Ein Pumping–Lemma für Kantenersetzungssprachen bezüglich maximaler Weglänge. Master Thesis, University of Bremen, 1991.

[Kus93] S. Kuske. A maximum path length pumping lemma for edge–replacement languages. In *FCT'93*, pages 342–351. Springer-Verlag, 1993. LNCS 710.

[KV94] A. Kühnemann and H. Vogler. Synthesized and inherited functions — a new computational model for syntax–directed semantics. *Acta Informatica*, 31:431–477, 1994.

[Ogd68] W. Ogden. A helpful result for proving inherent ambiguity. *Math. Syst. Theory*, 2:191–194, 1968.

[Per76] C.R. Perrault. Intercalation lemmas for tree transducer languages. *J. Comput. Syst. Sci.*, 13:246–277, 1976.

[Rou70] W.C. Rounds. Mappings and grammars on trees. *Math. Syst. Theory*, 4:257–287, 1970.

[Sch60] S. Scheinberg. Note on the boolean properties of context free languages. *Inform. and Control*, 3:372–375, 1960.

[Tha70] J.W. Thatcher. Generalized^2 sequential machine maps. *J. Comput. Syst. Sci.*, 4:339–367, 1970.

[Wis76] D. S. Wise. A strong pumping lemma for context–free languages. *Theoret. Comp. Sci.*, 3:359–369, 1976.

[Yu89] S. Yu. A pumping lemma for deterministic context–free languages. *Inform. Proc. Letters*, 31:47–51, 1989.

Received March 26, 1994