Tree Transducers

Niko Paltzer

Programming Systems Lab
Universität des Saarlandes, D-66041 Saarbrücken, Germany
nikopp@ps.uni-sb.de

Seminar Formal Grammars WS 06/07
Advisor: Marco Kuhlmann

Abstract. The concept of top-down tree transducers is known since decades and the purpose of this paper is to give a smooth introduction into this concept. Within this context, we consider the notion of copying, a copying normal form and a corresponding algorithm. This leads to the presentation of the intercalation lemma that is used to establish a language hierarchy on a special class of tree transducers with respect to the copying-bound parameter.

1 Introduction

The concept of top-down tree transducers is known since decades [4] and the purpose of this paper is to give a smooth introduction into this concept. In order to keep things simple, we only consider deterministic transducers. [1]

Tree transducers are designed to manipulate trees and together with the notion of a copying-bound, some interesting language hierarchies can be established. Since the copying-bound is based on dynamic properties, we point out a possibility to derive it statically. Afterward, we consider an analogon for the pumping lemma of context-free grammars, named *intercalation lemma* [4]. We use it to show that a particular class of tree transducers form a language hierarchy with respect to the copying-bound parameter.

The paper is organized as described in the following. Most of the necessary basic definitions are stated in Sec. 2. Section 3 provides some examples to get an idea on how the definitions work together and what is meant by copying. In Sec. 4 we present a *copying normal form* together with an algorithm [2] and propose some modifications to generalize this algorithm. In order to show that the possible amount of copying performed by a transducer is directly related to its expressiveness, we consider the intercalation lemma together with a sample application in Sec. 5.

[1] For a special class of transducers, there also exists an algorithm to transform nondeterministic ones into deterministic ones [4].

2 Definitions

The following basic definitions are mainly taken from [2] and [3]. We slightly modified them wherever we thought it would better suit our requirements. Additional definitions and some notation that are more specific, are introduced in the accordant section.

Definition 1 (Ranked Alphabet). *A ranked alphabet is a finite alphabet Σ with rank function $d_\Sigma : \Sigma \to \mathbb{N}$. By Σ_n we denote the set $\{\sigma \in \Sigma \mid d_\Sigma(\sigma) = n\}$.*

Definition 2 (Trees). *A tree over Σ is either a symbol of rank 0 or a string of the form $\sigma(t_1 \dots t_n)$, where σ has rank n, with $n \geq 1$, and t_i is a tree over Σ (for $1 \leq i \leq n$).* [2]

The set of all trees over Σ is denoted T_Σ, which is (formally) a subset of $(\Sigma \cup \{(,)\})^*$.

Definition 3 (Variables). *We use $V = \{x_1, x_2, x_3, \dots\}$ as a denumerably infinite set of variables, $V_0 = \emptyset$ and, for $n \geq 1$, $V_n = \{x_1, \dots, x_n\}$. In (small) examples we will use x, y, z rather than x_1, x_2, x_3.*

For an alphabet Σ, $n > 0$ and a string containing variables, i.e. $w_0 \in (\Sigma \cup V_n)^*$, and strings $w_1, \dots, w_n \in \Sigma^*$, $w_0[w_1, \dots, w_n]$ denotes the result of substituting w_i for x_i in w_0 (where $1 \leq i \leq n$).

Definition 4 (Top-down Tree Transducer). *A top-down tree transducer*[3] *is a construct $M = (Q, \Sigma, \Delta, q_0, R)$, where*

1. *Q is a finite set of states*
2. *Σ is the ranked input alphabet*
3. *Δ is the ranked output alphabet*
4. *$q_0 \in Q$ is the initial state*
5. *R is a finite set of (transducer) rules*

A rule is of the form $q(\sigma(x_1 \dots x_n)) \to \tau(q_1(x_{i_1}) \dots q_k(x_{i_k}))$ with $n, k \geq 0$, $\sigma \in \Sigma_n$, $\tau \in \Delta_k$, $q, q_1, \dots, q_k \in Q$ and $1 \leq i_m \leq n$ for $1 \leq m \leq k$. [4]

M is deterministic if different rules in R have different left-hand sides.

Definition 5 (Sentential Form). *Let $d_Q : Q \to \mathbb{N}$ be the rank function for the set of states Q (of M) with $d(q) = 1$ for all $q \in Q$.*

A sentential form is an element $t \in T_{\Sigma \cup Q \cup \Delta}$ with the following properties: Either $t = q(t')$ with $q \in Q$ and $t' \in T_\Sigma$ or $t = \delta(t_1 \dots t_n)$ with $\delta \in \Delta$ and $t_1, \dots, t_n$ are sentential forms.

A sentential form is a valid intermediate result of a derivation (Def. 6). The following property might help to imagine how such a sentential form looks like:

[2] The rank of a tree is the number of its direct subtrees.
[3] We will omit the "top-down" from now on.
[4] Note that $q, q_1, \dots, q_k$ and $x_{i_1}, \dots, x_{i_k}$ do not have to be distinct, respectively.

If you pick an arbitrary path of the sentential form from the root to a leave, then this path is *ordered* in the sense that there are some elements of Δ first, followed by one element of Q and some elements of Σ at the end. There are two additional special cases: The path might contain no elements of Δ or it only consists of elements of Δ.

Definition 6 (Derivation). *For sentential forms s_1 and s_2 we write $s_1 \Rightarrow s_2$ if s_2 is obtained from s_1 by applying a rule $q(\sigma(x_1 \dots x_n)) \to \delta(w)$, i.e., replacing a subtree $q(\sigma(t_1 \dots t_n))$ of s_1 by $\delta(w[t_1, \dots, t_n])$.*

As usual, $\Rightarrow^*$ is used to denote derivations, the reflexive and transitive closure of $\Rightarrow$.

Definition 7 (Translation). *The translation of a tree transducer M is defined by*

$$\{(t, t') \in T_\Sigma \times T_\Delta \mid q_0(t) \Rightarrow^* t'\}.$$

We define the language generated by M for an input tree language L to be

$$\mathcal{L}(M, L) = \{t' \in T_\Delta \mid q_0(t) \Rightarrow^* t' \text{ for some } t \in L\}.$$

We denote $\mathcal{L}(M, L)$ by $\mathcal{L}(M)$ if $L = T_\Sigma$.

Definition 8 (State-sequence). *Let $M = (Q, \Sigma, \Delta, q_0, R)$ be a tree transducer, $t \in T_\Sigma$ an input tree and d a subtree of t at level l. Furthermore, let s be a sentential form such that there exists a derivation $\alpha : q(t) \Rightarrow^* s$ and all subtrees of s that have a root node from Q occur at level l (of s).*

We define the state-sequence $\langle q_1 \dots q_k \rangle$, $k \geq 0$, of d with respect to α as the sequence of states $q_i \in Q$ that occur (from left to right) in s as parent nodes of d.

Intuitively, the state-sequence displays how often a particular subtree of the input is copied and which states of the transducer are applied to it.

Definition 9 (Copying). *Let $M = (Q, \Sigma, \Delta, q_0, R)$ be a tree transducer and let $k \geq 1$ be an integer. A derivation $\alpha : q_0(t) \Rightarrow^* t'$ has copying-bound k if, for each node d of t, the length of the state-sequence of d with respect to α is at most k.*

Let L be a tree language. (M, L) has copying-bound k if for each $t' \in M(L)$ there exist $t \in L$ and a derivation $q_0(t) \Rightarrow^* t'$ with copying-bound k. (M, L) is finite copying if it has copying-bound k for some $k \in \mathbb{N}$. [5]

The class of deterministic tree transducers with copying-bound k is denoted by $DT_{fc(k)}$.

[5] We will write that *M is finite copying* if L is clear from the context.

3 Warmup

Having now all those definitions at hand, we can start playing with tree transducers. The fact that a tree transducer has an input and an output alphabet somehow implies that it transforms trees of an input language into trees of an output language. It often makes no sense to define a transducer for an arbitrary input tree language. Otherwise, we could define a nontrivial language and combine it with a trivial transducer. Therefore, we restrict the input language to some tree language $L \subset T_\Sigma$ that is recognized by a finite tree automaton (FTA). The class of recognizable tree languages is denoted by REC and several references concerning FTAs and properties of REC are included in [2].

In order to clarify the definitions and to get some intuition on the behavior of a tree transducer, we will give some small examples in the following.

Example 1. Let $M^1 = (\{q_0, q_1\}, \{\sigma, \tau\}, \{\beta, \gamma, \delta\}, q_0, R^1)$ such that σ, β and γ have rank 1 whereas τ and δ have rank 0. R^1 consists of the rules

$$\begin{aligned} q_0(\sigma(x)) &\to \beta(q_1(x)) \\ q_0(\tau) &\to \delta \\ q_1(\sigma(x)) &\to \gamma(q_0(x)) \\ q_1(\tau) &\to \delta. \end{aligned}$$

The tree transducer M^1 transforms a linear tree $\sigma^n(\tau)$ into an alternating linear tree $\beta(\gamma(\beta(\gamma(\ldots(\delta)))))$ with depth n. Since the left-hand sides of the rules are disjoint, M^1 is deterministic.

Let's have a look at the derivation for input $t = \sigma(\sigma(\sigma(\tau)))$:

$$\begin{aligned} q_0(t) &\Rightarrow \beta(q_1(\sigma(\sigma(\tau)))) \\ &\Rightarrow \beta(\gamma(q_0(\sigma(\tau)))) \\ &\Rightarrow \beta(\gamma(\beta(q_1(\tau)))) \\ &\Rightarrow \beta(\gamma(\beta(\delta)))) \end{aligned}$$

From the derivation we can see that the state-sequence for $\sigma(\sigma(\sigma(\tau)))$ and $\sigma(\tau)$ is $\langle q_0 \rangle$. For $\sigma(\sigma(\tau))$ and τ we have the state-sequence $\langle q_1 \rangle$.

Therefore, this derivation has copying-bound 1. Since this holds for all derivations of possible input trees, M^1 is finite copying with copying-bound 1.

Example 2. Let $M^2 = (\{q_0\}, \{\sigma, \tau\}, \{\beta, \delta\}, q_0, R^2)$ such that β has rank 2, σ has rank 1, τ and δ have rank 0, and R^2 consists of the rules

$$\begin{aligned} q_0(\sigma(x)) &\to \beta(q_0(x)q_0(x)) \\ q_0(\tau) &\to \delta. \end{aligned}$$

The tree transducer M^2 transforms a linear tree $\sigma^n(\tau)$ into a binary tree $\beta(\beta(\ldots(\delta\delta))\beta(\ldots(\delta\delta)))$ with depth n and 2^n leaves labeled δ. Since the left-hand sides of the rules are disjoint, M^2 is deterministic.

For the derivation we add subscripts to the input symbols in order to distinguish different nodes with the same label. Then, the derivation for input $t = \sigma_1(\sigma_2(\tau_1))$ looks as follows:

$$\begin{aligned} q_0(t) \Rightarrow \quad & \beta(q_0(\sigma_2(\tau_1))q_0(\sigma_2(\tau_1))) \\ \Rightarrow^* \ & \beta(\beta(q_0(\tau_1)q_0(\tau_1))\beta(q_0(\tau_1)q_0(\tau_1))) \\ \Rightarrow^* \ & \beta(\beta(\delta\delta)\beta(\delta\delta)) \end{aligned}$$

Investigating the corresponding state-sequences, the subtree with root σ_1 has $\langle q_0 \rangle$, σ_2 has $\langle q_0 q_0 \rangle$ and τ_1 has $\langle q_0 q_0 q_0 q_0 \rangle$, respectively.

This derivation has copying-bound 4. In general, the copying-bound of this example directly corresponds to the number of leaves of the output tree, i.e. 2^n for an input tree of depth n. Therefore, M^2 is not finite copying.

4 Copying Normal Form (CNF)

The definition of copying-bound was founded on derivations which are dynamic artifacts. In order to show that the copying-bound can also be determined statically, we introduce the copying normal form and a corresponding algorithm, which are both described in [2].

Definition 10 (Copying Normal Form). *A tree transducer M is in copying normal form iff for all derivations α and for all state-sequences s of α, all states occur in s at most once.*

The benefit of this normal form is the fact that for any tree transducer M that is in copying normal form, the number of states of M is an upper bound for the copying-bound of M.

Each tree transducer that is deterministic and finite copying can be transformed into an equivalent one that is in copying normal form by applying the following algorithm. Unfortunately, this algorithm may introduce many new states such that the above mentioned upper bound is not very tight.

4.1 Algorithm

In order to state the algorithm in a more or less readable format, we have to introduce some additional notation.

Definition 11 (Notation). *For a rule $q(\sigma(x_1 \dots x_n)) \to \tau(q_1(x_{i_1}) \dots q_k(x_{i_k}))$ we define $SS(q, \sigma, x_j)$ to be the sequence of states that is obtained by deleting all states from $\langle q_1 \dots q_k \rangle$ that are* **not** *applied to x_j.*

An additional superscript l denotes, that the lth state in the sequence is overlined, i.e. $SS^l(q, \sigma, x_j) = \langle q_{i_1} \dots \overline{q_{i_l}} \dots q_{i_m} \rangle$.

The concatenation of two sequences is denoted by the $\circ$-operator, i.e. $\langle q_k \dots q_l \rangle \circ \langle q_m \dots q_n \rangle = \langle q_k \dots q_l q_m \dots q_n \rangle$.

The main idea of the algorithm is to code the state-sequences into the states. Therefore the new states are labeled with state-sequences and exactly one state in such a label is overlined, namely the *active* one.

An agenda A is maintained that keeps track of all newly introduced states, since the rules for these states have to be copied.

Let $M = (Q, \Sigma, \Delta, q_0, R)$ be the tree transducer that should be transformed into copying normal form.

1. The algorithm starts with the introduction of a new initial state $\langle\overline{q_0}\rangle$ and inserts it into A.
2. As long as A is not empty, the algorithm picks an arbitrary state q' out of A, modifies the rules from R that correspond to the overlined state in the label of q' and enters them into R'.
   For $q' = \langle q_{i_1} \dots \overline{q_{i_l}} \dots q_{i_m}\rangle$ and a rule
   $q_{i_l}(\sigma(x_1 \dots x_n)) \to \tau(q_1(x_{i_1}) \dots q_k(x_{i_k}))$ we obtain a rule
   $q'(\sigma(x_1 \dots x_n)) \to \tau(q_1{}'(x_{i_1}) \dots q_k{}'(x_{i_k}))$ with
   $q_j{}' = SS(q_{i_1}\sigma, x_{i_j}) \circ \dots \circ SS^n(q_{i_l}, \sigma, x_{i_j}) \circ \dots \circ SS(q_{i_m}, \sigma, x_{i_j})$ and n is the number of x_{i_h} with $x_{i_h} = x_{i_j}$ for $h \leq j$.
3. Move q' from A to Q' and continue with 2.
4. If A is empty, the algorithm is done and the output is $M' = (Q', \Sigma, \Delta, \langle\overline{q_0}\rangle, R')$.

This algorithm is considered to be correct because the difference between the new and the old tree transducer is just duplication and renaming of the states. Neither the alphabet nor the rules have been changed.

Since there are a lot of indices involved in the description of the algorithm, we will give an example and hope that it will clarify the course of action.

Example 3. Let's have a look at the tree transducer
$M^3 = (\{q, r, s\}, \{\tau, \delta\}, \{\alpha, \beta, \gamma\}, q, R^3)$ such that β has rank 2, τ and α have rank 1 whereas δ and γ have rank 0. R^3 consists of the rules

$$\begin{aligned} q(\tau(x)) &\to \beta(r(x)s(x)) \\ q(\delta) &\to \gamma \\ r(\tau(x)) &\to \beta(s(x)s(x)) \\ r(\delta) &\to \gamma \\ s(\tau(x)) &\to \alpha(s(x)) \\ s(\delta) &\to \gamma. \end{aligned}$$

The algorithm constructs the tree transducer
$M^{3'} = (\{\langle\overline{q}\rangle, \langle\overline{r}s\rangle, \langle r\overline{s}\rangle, \langle r\overline{s}\rangle, \langle\overline{s}ss\rangle, \langle s\overline{s}s\rangle, \langle ss\overline{s}\rangle\}, \Sigma^3, \Delta^3, \langle\overline{q}\rangle, R^{3'})$ and $R^{3'}$ consists of the rules

$$\begin{aligned} \langle\overline{q}\rangle(\tau(x)) &\to \beta(\langle\overline{r}s\rangle(x)\langle r\overline{s}\rangle(x)) \\ \langle\overline{q}\rangle(\delta) &\to \gamma \\ \langle\overline{r}s\rangle(\tau(x)) &\to \beta(\langle\overline{s}ss\rangle(x)\langle s\overline{s}s\rangle(x)) \end{aligned}$$

$$\begin{aligned}
\langle \overline{r}s\rangle(\delta) &\to \gamma \\
\langle r\overline{s}\rangle(\tau(x)) &\to \alpha(\langle ss\overline{s}\rangle(x)) \\
\langle r\overline{s}\rangle(\delta) &\to \gamma \\
\langle \overline{s}ss\rangle(\tau(x)) &\to \alpha(\langle \overline{s}ss\rangle(x)) \\
\langle \overline{s}ss\rangle(\delta) &\to \gamma \\
\langle s\overline{s}s\rangle(\tau(x)) &\to \alpha(\langle s\overline{s}s\rangle(x)) \\
\langle s\overline{s}s\rangle(\delta) &\to \gamma \\
\langle ss\overline{s}\rangle(\tau(x)) &\to \alpha(\langle ss\overline{s}\rangle(x)) \\
\langle ss\overline{s}\rangle(\delta) &\to \gamma.
\end{aligned}$$

Let's try to clarify how the right-hand side of the rule for $\langle \overline{r}s\rangle(\tau(x))$ has been constructed. Since M^3 is deterministic, the decisions we have to make are unambiguous.

1. The rule that we have to copy from R^3 is chosen via the overlined state r in the label and the input symbol τ, i.e. $r(\tau(x)) \to \beta(s(x)s(x))$.
2. In this rule, we have to replace the old states by new ones. The labels for these states depend on $\langle \overline{r}s\rangle$, τ and the variables they are applied to. For r, τ and x, we obtain $\langle ss\rangle$. Since r is overlined, we get $\langle \overline{s}s\rangle$ and $\langle s\overline{s}\rangle$ for the first and the second occurence, respectively. For s, τ and x, we obtain $\langle s\rangle$. The concatenation yields $\langle \overline{s}ss\rangle$ and $\langle s\overline{s}s\rangle$ which raplace the first and second occurence of s, respectively.

Modifications The described algorithm can be modified such that it additionally computes the copying bound of the given tree transducer M or decides that M is not finite copying. We will just give the ideas of these modifications without proving any properties, since this is beyond the range of this paper.

The modification to obtain the copying-bound is rather trivial. Since the generated states are labeled with state-sequences, we just measure the length of these labels and the maximum over all these values is the copying-bound.

To find out if a tree transducer is finite copying or not, we were not able to find such a simple solution. The first idea is to let the algorithm build a directed graph that depends on the rules and this graph is acyclic iff the tree transducer is finite copying. But it is not sufficient just to check for cycles in this graph because cycles also do occur in tree transducers that are finite copying. Otherwise, a tree transducer could not produce an infinite language since it has only finitely many states and rules.

Therefore we have to consider two kinds of cycles. The *good* ones, that occur in finite copying tree transducers, and the *bad* ones, that make the transducer loose this property.

We choose the nodes of the graph to be the states of M. A directed edge (q, q') is marked if q' is applied to a subtree that has been copied due to a rule for q. Then the *bad* cycles in the graph are those that contain at least one marked edge.

The algorithm should build a graph $G = (E, V)$ as follows: [6]

1. When we access a rule r of R, we add each state q that occurs in r to E. For $r = q(\sigma(x_1 \dots x_n)) \to \tau(q_1(x_{i_1}) \dots q_k(x_{i_k}))$ we add edges (q, q_j) to V for $1 \leq j \leq k$. If $x_{i_j} = x_{i_l}$ for some $l \in \{1, \dots, k\}$ then (q, q_j) is marked.
2. After each change (or bunch of changes) of G, we test if we can find a cycle in G that contains at least one marked edge. If we discover such a *bad* cycle, we can abort the computation since M is not finite copying.

Example 4. If we slightly modify one rule of M^3, that was presented in Example 3, we obtain a tree transducer that is no longer finite copying.

Let $M^4 = (\{q, r, s\}, \{\tau, \delta\}, \{\alpha, \beta, \gamma\}, q, R^4)$ such that β has rank 2, τ and α have rank 1 whereas δ and γ have rank 0. R^4 consists of the rules

$$\begin{aligned} q(\tau(x)) &\to \beta(r(x)s(x)) \\ q(\delta) &\to \gamma \\ r(\tau(x)) &\to \beta(s(x)s(x)) \\ r(\delta) &\to \gamma \\ s(\tau(x)) &\to \alpha(q(x)) \\ s(\delta) &\to \gamma. \end{aligned}$$

If we take M^4 as input for the modified algorithm, it will build up a graph that will (at some point in time) look like the one depicted in Fig. 1. The marked edges are labeled with m and we can see that there are two *bad* cycles, namely $q \to s \to q$ and $q \to r \to s \to q$, containing one marked edge and two marked edges, respectively.

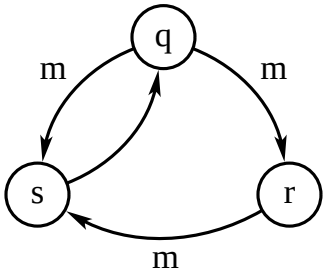

Fig. 1. The graph that is built by the modified CNF algorithm for input M^4.

[6] Insertion of nodes and edges is only done if necessary, i.e. if they are not already in the graph. The only exception: If an unmarked edge is in the graph, it could be replaced by a marked one but not vice versa.

5 Intercalation Lemma

The intercalation lemma for tree transducer languages is the analogon for the pumping lemma for context-free grammars. It can be used to show that a language L is not an element of the yield languages of a class of tree transducers $\mathcal{T}$.

We first give a recap of the pumping lemma for context-free grammars (found in [5]), then state the definition of a yield language and introduce the intercalation lemma for finite copying deterministic tree transducers. (The more general version of the lemma can be found in [4].) We point out the main differences between the two given lemmas afterward.

Lemma 1 (Pumping Lemma for CFGs). *For each context-free language L, there exists a constant $n \in \mathbb{N}$ such that each $z \in L$ with $|z| \geq n$ can be written as $z = uvwxy$ with $|vx| \geq 1, |vwx| \leq n$ and $\forall i \geq 0 : uv^iwx^iy \in L$.*

Figure 2 depicts what you should have in mind when arguing about the pumping lemma for CFGs. Unfortunately, there is no such nice picture for the intercalation lemma as we will see later.

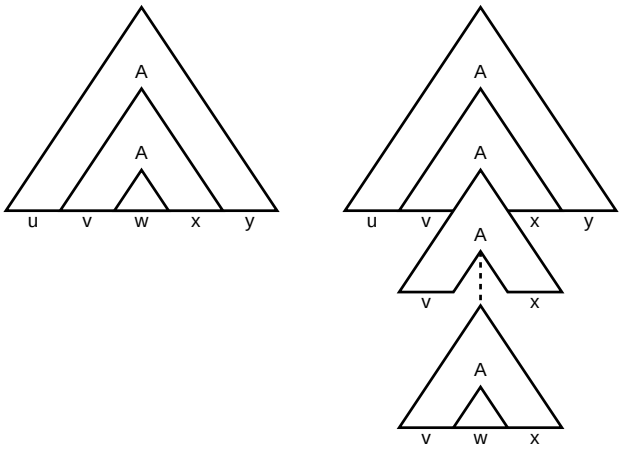

Fig. 2. The picture behind the pumping lemma for CFGs. It is an abstract view of possible derivation trees where A is a nonterminal of the CFG and u, v, w, x and y are sequences of terminals.

Definition 12 (Yield Language). *Let $M = (Q, \Sigma, \Delta, q_0, R)$ be a tree transducer and L a tree language over Σ.*

Then $y\mathcal{L}(M, L) = \{yield(t') \mid q_0(t) \Rightarrow^* t' \text{ for some } t \in L\}$ is the yield language of (M, L). [7]

With $y\mathcal{L}(\mathcal{T}, \mathcal{C}) = \{y\mathcal{L}(M, L) \mid M \in \mathcal{T} \text{ and } L \in \mathcal{C}\}$ we denote the set of yield languages for some class of tree transducers $\mathcal{T}$ and some class of tree languages $\mathcal{C}$.

The string language generated by a CFG G is equivalent to the yield language of the derivation trees of G together with the identity transducer. The identity transducer simply outputs the input tree like the identity function does.

Lemma 2 (Intercalation Lemma). *Let $k > 0$ be an integer. For each tree language $L \in y\mathcal{L}(DT_{fc(k)}, \text{REC})$ there exists a constant $p \in \mathbb{N}$ such that each $z \in L$ with $|z| \geq p$ can be written as $z = x_1 z_1 x_2 z_2 \dots x_s z_s x_{s+1}$ with $(1 \leq s \leq k)$ such that $0 < |z_i| \leq p$ for $1 \leq i \leq s$ and the following holds:*

$\forall n \in \mathbb{N}$. $\exists v_1, \dots, v_s$ such that

1. *$v = x_1 v_1 x_2 v_2 \dots x_s v_s x_{s+1}$*
2. *$v \in L$*
3. *$|v| \geq n$*
4. *$alph(v_i) = alph(z_i)$ for $1 \leq i \leq s$*

Note that $alph(w)$ is the set of symbols occurring in w.

On the one hand this lemma is very similar to the pumping lemma, but on the other hand there are some major differences. One aspect is the fact that there can be up to k parts $x_1 \dots x_s$ where something can be intercalated into the string whereas there are only two parts v and y in a string that can be pumped.

While the pumping lemma clearly predicts how the pumped string looks like, the intercalation lemma is weaker in the sense that it only states, that the set of used symbols does not change when intercalating from z_i to v_i. As already mentioned, this is the reason, why we cannot give such a nice picture for the intercalation lemma. And we will see in Example 6 why the lemma has this restriction.

Theorem 1 (Language Hierarchy). *The yield languages generated by finite copying deterministic tree transducers form a hierarchy with respect to the copying-bound k, i.e.*

$y\mathcal{L}(DT_{fc(k-1)}, \text{REC}) \subsetneq y\mathcal{L}(DT_{fc(k)}, \text{REC})$ *for $k > 1$*

Proof. It is clear that $y\mathcal{L}(DT_{fc(k-1)}, \text{REC}) \subset y\mathcal{L}(DT_{fc(k)}, \text{REC})$, since by the definition of the copying-bound, $DT_{fc(k-1)} \subset DT_{fc(k)}$.

It remains to show that $y\mathcal{L}(DT_{fc(k-1)}, \text{REC}) \neq y\mathcal{L}(DT_{fc(k)}, \text{REC})$ for $k > 1$. For that purpose, we use the language $L_k = \{{a_1}^n {a_2}^n \dots {a_{2k}}^n \mid n \in \mathbb{N}\}$ for $k > 1$.

With Lemma 3 and 4 we show that $L_k \in y\mathcal{L}(DT_{fc(k)}, \text{REC})$ but $L_k \notin y\mathcal{L}(DT_{fc(k-1)}, \text{REC})$, respectively.

Therefore $y\mathcal{L}(DT_{fc(k-1)}, \text{REC}) \neq y\mathcal{L}(DT_{fc(k)}, \text{REC})$.

□

[7] The yield of a tree is the concatenation of all leaves from left to right.

Lemma 3 (Language Hierarchy). *Let* $L_k = \{{a_1}^n {a_2}^n \dots {a_{2k}}^n \mid n \in \mathbb{N}\}$, *then* $L_k \in y\mathcal{L}(DT_{fc(k)}, \mathrm{REC})$ *for* $k > 1$.

Proof. We can prove this lemma by constructing an appropriate tree transducer. Let $M = (\{q_0, \dots, q_k, r_1, \dots, r_{2k}\}, \{a, b, c, d\}, \{\overline{a}, b, c, a_1, \dots, a_{2k}\}, q_0, R)$ where $\overline{a}$ has rank k, b has rank 3, c has rank 2, a has rank 1 and $d, a_1, \dots, a_{2k}$ have rank 0. Furthermore, R contains the following rules for $1 \leq i \leq k$:

$$\begin{aligned} q_0(a(x)) &\to \overline{a}(q_1(x) \dots q_k(x)) \\ q_i(b(xyz)) &\to b(r_{2i-1}(x) q_i(y) r_{2i}(z)) \\ q_i(c(xy)) &\to c(r_{2i-1}(x) r_{2i}(y)) \\ r_{2i-1}(d) &\to a_{2i-1} \\ r_{2i}(d) &\to a_{2i} \end{aligned}$$

The recognizable input language for this tree transducer contains trees of the form $a(b(db(d \dots b(dc(dd)d) \dots d)d))$. The output has the form
$a(b(a_1 b(a_1 \dots b(a_1 c(a_1 a_2)a_2) \dots a_2)a_2) \dots$
$\quad b(a_{2k-1} b(a_{2k-1} \dots b(a_{2k-1} c(a_{2k-1} a_{2k})a_{2k}) \dots a_{2k})a_{2k}))$.

The copying-bound of M is k since copying only occurs in the first rule where the direct subtree of a is copied k times. □

Example 5 shows the construction of Lemma 3 for the case $k = 2$.

Lemma 4 (Language Hierarchy). *Let* $L_k = \{{a_1}^n {a_2}^n \dots {a_{2k}}^n \mid n \in \mathbb{N}\}$, *then* $L_k \notin y\mathcal{L}(DT_{fc(k-1)}, \mathrm{REC})$ *for* $k > 1$.

Proof. We assume that $L_k \in y\mathcal{L}(DT_{fc(k-1)})$ and deduce a contradiction.

According to our assumption, we can apply the intercalation lemma (Lemma 2). Let p be the constant from the lemma and $z = {a_1}^p {a_2}^p \dots {a_{2k}}^p \in L_k$.

According to the lemma there exists a decomposition $z = x_1 z_1 x_2 z_2 \dots x_s z_s x_{s+1}$ with $s \leq k - 1$ and $0 < |z_i| \leq p$.

Since all z_i have to be nonempty, at least one a_i occurs in $z_1, \dots, z_s$. And since $|z_i| \leq p$, each z_i can contain at most two different a_i. Hence, there are at most $2k - 2$ different a_i in $z_1, \dots, z_s$.

It follows that there is some a_j that occurs in $z_1, \dots, z_s$ and some $a_{j'}$ that does not occur in any z_i. Since $alph(z_i) = alph(v_i)$, this $a_{j'}$ does not occur in any v_i.

Since the lemma states that we can produce an arbitrary long word $v \in L_k$, the number of at least one a_j must increase whereas the number of $a_{j'}$ remains p and therefore $v \notin L_k$.

□

Example 5. To illustrate the construction that was done in Lem. 3 we state an example for the case $k = 2$ in the following.

Let $M^5 = (\{q_0, q_1, q_2, r_1, r_2, r_3, r_4\}, \{a, b, c, d\}, \{\overline{a}, b, c, a_1, a_2, a_3, a_4\}, q_0, R^5)$ and R^5 contains the following rules:

$$\begin{aligned}
q_0(a(x)) &\to \overline{a}(q_1(x)q_2(x)) \\
q_1(b(xyz)) &\to b(r_1(x)q_i(y)r_2(z)) \\
q_2(b(xyz)) &\to b(r_3(x)q_i(y)r_4(z)) \\
q_1(c(xy)) &\to c(r_1(x)r_2(y)) \\
q_2(c(xy)) &\to c(r_3(x)r_4(y)) \\
r_1(d) &\to a_1 \\
r_2(d) &\to a_2 \\
r_3(d) &\to a_3 \\
r_4(d) &\to a_4
\end{aligned}$$

Figure 3 shows one sample input and output tree for M^5.

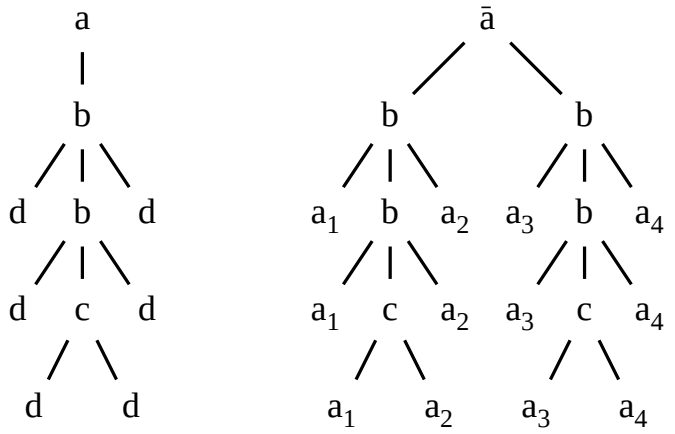

Fig. 3. Tree transducer M^5 generates for the input tree on the left the output tree on the right. And the yield of the output tree is ${a_1}^3{a_2}^3{a_3}^3{a_4}^3$.

A tree transducer may reorder subtrees in any possible way. Therefore, the intercalation lemma is weaker than the pumping lemma for CFGs in the sense that there is not simply a multiplication of substrings for the yield of an intercalated tree, as shown in the following example.

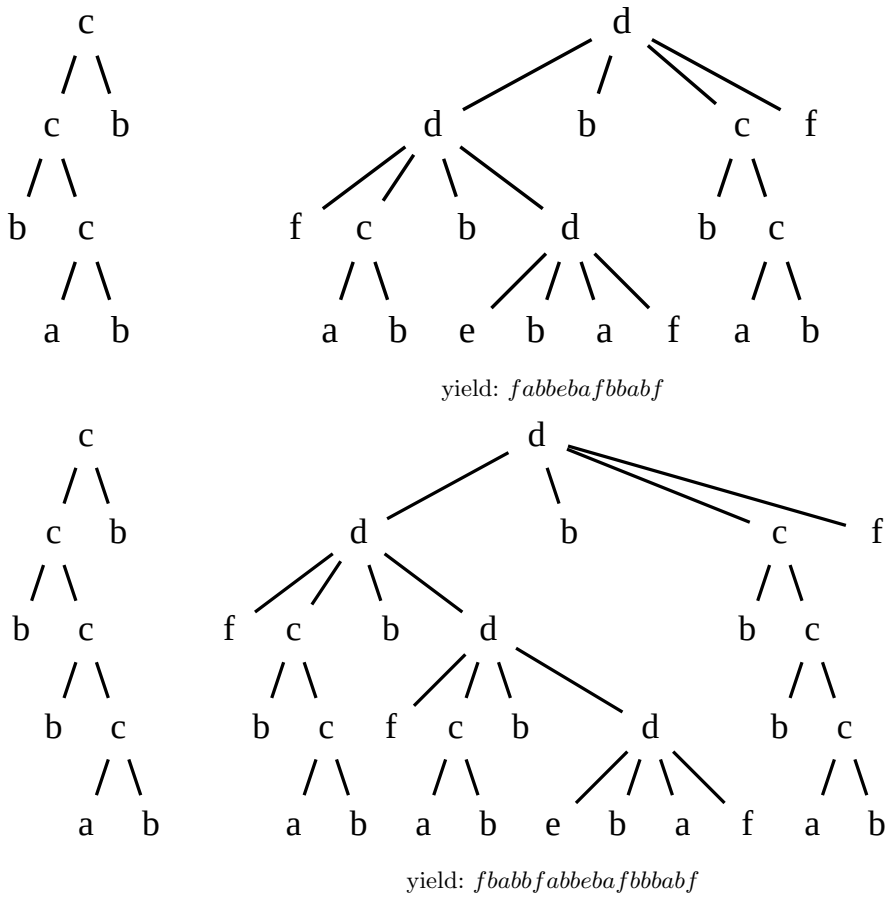

Fig. 4. The top right tree is obtained from the top left one by *intercalation*. The trees below are the corresponding output trees with respect to transducer M^6.

Example 6. This example is adapted from Ex. 1.1 in [3].

Let $M^6 = (\{q\}, \{a, b, c\}, \{a, b, c, d, e, f\}, q, R^6)$ where a, b, e and f have rank 0, c has rank 2, d has rank 4 and R^6 contains the following rules:

$$\begin{aligned} q(c(xy)) &\to d(q(x)r(y)r(x)q(y)) \\ r(c(xy)) &\to c(r(x)r(y)) \\ q(a) &\to e \\ r(a) &\to a \\ q(b) &\to f \\ r(b) &\to b \end{aligned}$$

Figure 4 shows that the only structural relation between the two resulting yields is the fact that no different symbols have been introduced to the yield. But the order in which they appear highly depends on the rules of the transducer and can not be generalized.

References

1. Maneth, S.: Models of Tree Translation. IPA Dissertation Series (2004) 45–80
2. Vugt, N.: Generalized Context-Free Grammars. Master's Thesis, Universiteit Leiden (1996) 43–57
3. Perrault, C. R.: Intercalation theorems for tree transducer languages. STOC '75: Proceedings of seventh annual ACM symposium on Theory of computing (1975) 126–136
4. Engelfriet, J., Rozenberg, G., Slutzki, G.: Tree Transducers, L Systems, and Two-Way Machines. Journal of Computer and System Sciences 20 (1980) 150–176
5. Wegener, I.: Theoretische Informatik - eine algorithmenorientierte Einführung. B.G. Teubner Stuttgart • Leipzig (1999) 157–160