A Comparison of Pebble Tree Transducers with Macro Tree Transducers

Joost Engelfriet and Sebastian Maneth

Leiden University, LIACS, PO Box 9512, 2300 RA Leiden, The Netherlands
E-mail: {engelfri, maneth}@liacs.nl

Technical Report 2002–03

Abstract. The n-pebble tree transducer was recently proposed as a model for XML query languages. The four main results on deterministic transducers are: First, (1) the translation τ of an n-pebble tree transducer can be realized by a composition of $n+1$ 0-pebble tree transducers. Next, the pebble tree transducer is compared with the macro tree transducer, a well-known model for syntax-directed semantics, with decidable type checking. The 0-pebble tree transducer can be simulated by the macro tree transducer, which, by the first result, implies that (2) τ can be realized by an $(n+1)$-fold composition of macro tree transducers. Conversely, every macro tree transducer can be simulated by a composition of 0-pebble tree transducers. Together these simulations prove that (3) the composition closure of n-pebble tree transducers equals that of macro tree transducers (and that of 0-pebble tree transducers). Similar results hold in the nondeterministic case. Finally, (4) the output languages of deterministic n-pebble tree transducers form a hierarchy with respect to the number n of pebbles.

1 Introduction

Trees appear in science in many contexts. For instance, they are used to represent the structure of a composed object: the object is obtained by applying a certain operation (at the root of the tree) to its components (represented by the subtrees); such a tree corresponds to the derivation tree of a grammar generating the object. Another more recent example is XML, a general data format for structured documents; there, the interest is in the structure of the tree itself. Natural application areas of trees are (we mention only four) (i) linguistics (phrase structure), (ii) compilers (derivation trees, or parse trees), (iii) functional programming (terms), and (iv) databases (XML documents).

Let us now consider the translation of trees into other trees. It plays an important role in each of the four areas: (i) for natural language translation (see, e.g., [KMMM,MMM01]) (ii) for the specification of the syntax-directed semantics of a programming language, and its implementation in a compiler, cf. [Iro61,Knu68, KV97,WM95], (iii) for functional programs working on tree structured data, cf., e.g., [Vog91], and (iv) for the specification and implementation of XML transformation (e.g., XSLT; cf. [MN00,BMN02]) and XML query languages [Via01]. Now, consider the (sequential) composition of tree translations. It appears in applications in a natural way: e.g., as multi-pass compilers, as model for deforestation in functional languages [Küh98,Voi02] and as implementation of queries to a (possibly iterated) view of an (XML) database.

This paper is concerned with tree translations and compositions of them. In particular, we study the relationship between the n-pebble tree transducer, introduced in [MSV00,MSV] as a model for XML query languages (cf. also [Via01]), and the macro tree transducer [Eng80,Eng81,CF82,EV85,EV86,FV98] which is a model for syntax-directed semantics. We first discuss the pebble tree transducer (in the terminology used within this paper, which differs slightly from that in [MSV00,MSV]) and then the macro tree transducer.

An n-pebble tree transducer (n-ptt) is a finite state device that translates ordered ranked trees (which might be codings of XML documents). Its reading head is a pointer to a node of the input tree and can be moved to another node along the edges of the input tree. The n-ptt is equipped with n pebbles, marked $1, \dots, n$, which can be dropped at or lifted from the current node (pointed at by the reading head). A computation starts in the initial state with the reading head at the root node, and no pebbles on the input tree. The ptt can test (in its current state) the label of the current node, its "position" (i.e., whether it is the root node or the jth child of a node, $j \geq 1$), and the presence of the pebbles at the current node. Depending on the test, it generates an output tree, at the leaves of which new computations can be spawned (which will each have their "own" copy of the input tree, with pebbles and reading head). This means that, in terms of the output tree, the basic operation inherent in a computation step of an n-ptt is the replacement of leaves by trees ("first order tree substitution"). When a new computation is spawned, the ptt can change its state and either move the reading head to a neighboring node, or lift/drop a pebble at the current node. Pebbles must be used in a stack-like fashion: if $l \leq n$ pebbles are on the tree, then pebble l can be lifted (if it is present at the current node) or pebble $l+1$ can be dropped at the current node (if $l+1 \leq n$). We note here that in the model of [MSV00,MSV,Via01] the reading head is considered to be a pebble too; thus, our n-pebble tree transducer is there called an $(n+1)$-pebble tree transducer.

As observed in [MSV00], the pebble tree transducer can be obtained from the tree-walking automaton of [AU71] (see also [ERS80]) by adding pebbles and the ability to generate output trees rather than strings. We observe here that the deterministic pebble tree transducer without pebbles, i.e., the 0-ptt, is very closely related to the attribute grammar: a well-known compiler writing formalism (see, e.g., [DJL88,AM91,Paa95]). Here, the attributes of the attribute grammar should have trees as values (in which case it is also called an attributed tree transducer [EF81,Fül81,FV98]). This relationship was discussed in [Eng86], where the 0-ptt is called an RT(Tree-walk) transducer (see also [EV86]). Thus, 0-ptts are essentially attribute grammars, and n-ptts could be viewed as "attribute grammars with pebbles". If we further restrict the 0-ptt in such a way that the reading head may only move down in each computation step, then we obtain the classical top-down tree transducer [Rou70,Tha70,GS97], as mentioned in [MSV00].

For a pebble tree transducer, the restriction of input and output to monadic trees gives rise to a natural transducer model for string translation which was considered in [EM02b]. For some of the results of the present paper we will mention the corresponding results for pebble string transducers, but for more details the reader is referred to [EM02b]. String automata that use pebbles in a stack-like fashion (which basically means that the pebbles have nested life times) were introduced in [GH96] and extended to trees in [EH99] (see also [NSV01]).

The macro tree transducer (mtt) is also a finite state device that translates trees into trees. It can be obtained by combining the top-down tree transducer and the macro grammar [Fis68], i.e., the states of the top-down tree transducer may have parameters of type output tree, and thus computations can be spawned at non-leaf nodes of the output tree. Now, when the mtt executes a move at such a node v, it is replaced by an output tree which may spawn new computations, and in which each leaf labeled by the formal parameter y_j is replaced by the corresponding actual parameter, i.e., the jth subtree of v ("second-order tree substitution"). Just as for the top-down tree transducer, the reading head of the macro tree transducer can only move down. This implies that deterministic macro tree transducers do not have nonterminating computations, as opposed to deterministic pebble tree transducers.

Note that it is well known that (in the total deterministic case) all attributed tree transducers can be simulated by macro tree transducers [Fra82,CF82,FV99,EM99], and that the composition closures of the two coincide (cf., e.g., Chapter 6 of [FV98]). This suggests that (in the deterministic case) 0-ptts can be simulated by mtts, and that their composition closures coincide: one of our results. Macro tree transducers are well studied in tree transducer theory, and about their composition closure many attractive properties are known; for instance: it has decidable type checking [EV85], the translations can be computed in linear time (in the sum of the sizes of input and output tree) [Man02], and the output languages form a full AFL and have decidable emptiness and finiteness problems [DE98].

Before we discuss our results, let us consider the relationship of tree transducers to (XML based) databases, cf. [Via01,MSV00]. In terms of databases, tree transducers can be seen as a query language: the input tree is the current content of the database and the output tree is a result of the query that is computed by the transducer. Of course the result can be input to another query; this corresponds to the sequential composition of two tree transducers. In fact, the application of a query q to a database D (a set of inputs) is often used to define a derived version of the database, called the "view of D under q". This corresponds to the output language $\tau(R)$ of a tree transducer τ taking a set R of input trees. We will assume (as in [MSV00]) that R is a regular tree language (corresponding to a database type constraint as defined, e.g., by a DTD or a specialized DTD in XML).

Our first main result is completely independent of macro tree transducers. It is a result about pebble tree transducers only: The translation of an n-pebble tree transducer can be realized by the composition of $n+1$ zero-pebble tree transducers. In fact, the use of the first pebble can be simulated by (pre-)composing with the translation of a deterministic zero-pebble tree transducer. In terms of databases this means that a user who understands the concept of a view and that of a 0-pebble query (computed by a 0-ptt) need not be bothered with queries of n-pebble tree transducers for $n > 0$, i.e., need not know about pebbles at all. Moreover, we observe that it is a desirable property of a query language to be closed under composition: it means that querying a view (i.e., the result of a previous query) gives a result for which there is a direct query on the original database. Thus, it is natural to define the query language of a class of tree translations as its composition closure. Note that the class of pebble tree translations is *not* closed under composition (both in the deterministic and the nondeterministic case). For the composition closure of pebble tree transducers the first result implies that it is equal to the composition closure of zero-pebble tree transducers. Hence, as query languages in the above sense, the pebble tree transducer and the zero-pebble tree transducer are equally expressive.

Our second main result is that every pebble tree transducer can be simulated by a composition of macro tree transducers. In the nondeterministic case, to simulate n pebbles, $n+1$ mtts are needed in the composition and the mtts must be extended by the ability to remain at a node, instead of strictly moving down in each step. Since such a transducer can loop, it can have nonterminating computations. In the deterministic case, n pebbles can be simulated by the composition of $n+1$ (conventional) deterministic mtts. Also, a simulation in the converse direction is possible: for every macro tree transducer there is a composition of 0-pebble tree transducers which realizes the same translation. This gives our third main result: the composition closure of n-pebble tree transducers equals that of macro tree transducers, i.e., as query languages both formalisms have the same power. Since mtts always terminate, the simulations prove that compositions of deterministic pebble tree transducers can be transformed into ones that always terminate. Technically speaking, this is one of the key results of this paper.

Our fourth main result concerns the power of defining views, or, equivalently, the power to generate output languages (for deterministic transducers): $n + 1$ pebbles give strictly more views than n pebbles, i.e., there is a hierarchy with respect to the number n of pebbles, of the output languages of n-pebble tree transducers. The proof is based on the "mtt-hierarchy" of (string) output languages of n-fold compositions of mtts that was recently proved in [EM02a]. The result strengthens the hierarchy of translations of n-pebble tree transducers, which follows from an obvious size-to-height relationship for such translations (viz., the height of the output tree is polynomially bounded in the size of the input tree, with exponent $n + 1$). The proof uses counter examples that are monadic, and thus also proves that there is a hierarchy of output languages of n-pebble string transducers, as already presented in [EM02b]. Moreover, it is shown that nondeterminism gives more views: even without pebbles a nondeterministic (0-)ptt can compute a view that cannot be computed by any composition of deterministic pebble tree transducers.

Finally, we address the type checking problem for compositions of pebble tree transducers; it is the question whether all output documents in a view satisfy a given type (i.e., a regular tree language). Since it is well known that inverse type inference for compositions of macro tree transducers is solvable [EV85], our second main result provides an alternative proof of the main result of [MSV00] that type checking for pebble tree transducers is decidable. We also obtain an extension from [DE98]: "almost always" type checking is solvable for compositions of pebble tree transducers; it is the question whether all output documents in a view, except finitely many, satisfy a given type (and if so, to produce the list of exceptions).

The structure of this paper is as follows. The Preliminaries (Section 2) fix basic notations and definitions, mainly concerning trees, tree substitution, and tree grammars. Section 3 presents the definition of the n-pebble tree transducer (with a comparison to the original definition of [MSV00] in Subsection 3.1), and proves some of its elementary properties. In particular, the size-to-height relationship for ptts is proved, and then applied to show that there is a proper hierarchy of translations and that the class of pebble tree translations is not closed under composition. Subsections 3.2 and 3.3 compare ptts to attribute grammars and to the $\mathrm{RT}(S)$ transducers of [Eng86,EV86] (with S = Tree-walk). Section 4 proves our first result, the decomposition of an n-pebble tree translation into $n + 1$ zero-pebble tree translations. In Section 5 pebble tree transducers are compared with macro tree transducers. In particular, our second and third main results are proved there. In Section 6 the output languages of pebble tree transducers are investigated; it is proved that these languages form a proper hierarchy with respect to the number of pebbles. Section 7 discusses type checking, and almost always type checking. The paper ends with conclusions and suggestions for further research in Section 8.

Even when not explicitly mentioned in the lemmas and theorems, all our results are effective.

2 Preliminaries

The set $\{0, 1, \dots\}$ of natural numbers is denoted by $\mathbb{N}$. The empty set is denoted by $\varnothing$. For $k, l \in \mathbb{N}$, $[k]$ denotes the set $\{1, \dots, k\}$ and $[k, l]$ denotes the set $\{k, \dots, l\}$. For a set A, $|A|$ is the cardinality of A, $\mathcal{P}(A)$ is the set of subsets of A, A^* is the set of all strings over A, and A^+ is the set of nonempty strings over A. The empty string is denoted by ε. If the elements of A are strings themselves, then we might write a string $w \in A^*$ as $w = [a_1; a_2; \dots; a_n]$ with $a_i \in A$; in particular, we will then use λ to denote the empty string (of strings), i.e., λ has a different type than

ε. The length of a string w is denoted $|w|$, and the ith symbol in w is denoted by $w(i)$. For $n \geq 0$, $A^{\leq n}$ denotes the set $\{w \in A^* \mid |w| \leq n\}$.

For sets A and B, their cartesian product is $A \times B = \{(a,b) \mid a \in A, b \in B\}$. An ordered pair (a,b) will also be denoted $\langle a,b \rangle$, and $A \times B$ will also be denoted by $\langle A,B \rangle$.

For a binary relation R and a set A, $R(A)$ denotes the set $\{y \mid \exists x \in A : (x,y) \in R\}$ and $R^{-1}(A)$ denotes the set $\{x \mid \exists y \in A : (x,y) \in R\}$. Moreover, for a class $\mathcal{R}$ of binary relations and a class of sets $\mathcal{A}$, $\mathcal{R}(\mathcal{A})$ denotes the class of sets $\{R(A) \mid R \in \mathcal{R}, A \in \mathcal{A}\}$. The composition of two (binary) relations R and S, denoted by $R \circ S$, is the set of pairs $\{(x,z) \mid$ there is a y with $(x,y) \in R$ and $(y,z) \in S\}$. For $n \geq 0$, the n-fold composition of R with itself is denoted R^n. The reflexive, transitive closure and the transitive closure of R are denoted R^* and R^+, respectively. For classes of relations $\mathcal{R}$ and $\mathcal{S}$, $\mathcal{R} \circ \mathcal{S}$ denotes the class of relations $\{R \circ S \mid R \in \mathcal{R}, S \in \mathcal{S}\}$. For $n \geq 1$, $\mathcal{R}^n$ denotes $\mathcal{R} \circ \cdots \circ \mathcal{R}$ (n times) and $\mathcal{R}^*$ denotes the class $\bigcup_{n \geq 1} \mathcal{R}^n$.

For a binary relation $\Rightarrow \subseteq A \times A$ over a set A, we will call, for $a, a' \in A$, a derivation $a \Rightarrow^* a'$ a *computation* (*by* $\Rightarrow$ *starting with* a). Moreover, a computation starting with a can also be infinite. A computation is *complete* if it is either infinite or of the form $a \Rightarrow^* a' \not\Rightarrow$, i.e., there is no $a'' \in A$ such that $a' \Rightarrow a''$; in the latter case, a' is the *result* of the computation.

2.1 Ranked Sets and Trees

A set Σ together with a mapping $\text{rank}_\Sigma \colon \Sigma \to \mathbb{N}$ is called a *ranked set*. For $k \geq 0$, $\Sigma^{(k)}$ is the set $\{\sigma \in \Sigma \mid \text{rank}_\Sigma(\sigma) = k\}$; we also write $\sigma^{(k)}$ to indicate that $\text{rank}_\Sigma(\sigma) = k$. For a set A, $\langle \Sigma, A \rangle$ is the ranked set $\Sigma \times A$ with $\text{rank}_{\langle \Sigma, A \rangle}(\langle \sigma, a \rangle) = \text{rank}_\Sigma(\sigma)$ for every $\langle \sigma, a \rangle \in \langle \Sigma, A \rangle$.

Let Σ be a ranked set. The set of *trees* over Σ, denoted by T_Σ, is the smallest set of strings $T \subseteq (\Sigma \cup \{(,\,),\, ,\})^*$ such that $\Sigma^{(0)} \subseteq T$ and if $\sigma \in \Sigma^{(k)}$, $k \geq 1$, and $t_1, \ldots, t_k \in T$, then $\sigma(t_1, \ldots, t_k) \in T$. For a set A, the set of trees over Σ *indexed by* A, denoted by $T_\Sigma(A)$, is the set $T_{\Sigma \cup A}$, where for every $a \in A$, $\text{rank}_A(a) = 0$. For the rest of this paper we choose the set of *parameters* to be $Y = \{y_1, y_2, \ldots\}$. For $m \geq 0$, Y_m denotes the set $\{y_1, \ldots, y_m\}$. Thus, $T_\Sigma(Y)$ is the set of trees over Σ with parameters.

For every tree $t \in T_\Sigma$, the set of *nodes* of t, denoted by $V(t)$, is the subset of $\mathbb{N}^*$ that is inductively defined as follows: if $t = \sigma(t_1, \ldots, t_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $t_i \in T_\Sigma$ for all $i \in [k]$, then $V(t) = \{\varepsilon\} \cup \{iu \mid u \in V(t_i), i \in [k]\}$. Thus, ε represents the root of a tree and for a node u the ith child of u is represented by ui. The *size* of t is its number of nodes, i.e., $\text{size}(t) = |V(t)|$, and the *height* of t is the number of nodes on a longest path of t, i.e., $\text{height}(\sigma(t_1, \ldots, t_k)) = 1 + \max\{\text{height}(t_i) \mid i \in [k]\}$.

The *label* of t at node u is denoted by $t[u]$; we also say that $t[u]$ occurs in t (at u). The *rank* of u is the rank of its label $t[u]$; in particular, u is a *leaf* if it has no children, i.e., if it has rank zero. If $u = vw$ with $w \in \mathbb{N}^*$, then v is an *ancestor* of u and u is a *descendant* of v; if $w \neq \varepsilon$, then v is a *proper* ancestor of u and u is a *proper* descendant of v. The *subtree* of t *at* node u is denoted by t/u; a subtree t/ui is called a subtree *of* node u. The *substitution* of the tree $s \in T_\Sigma$ at node u in t is denoted by $t[u \leftarrow s]$; it means that the subtree t/u is replaced by s. Formally, these notions can be defined as follows: $t[\varepsilon]$ is the first symbol of t (in Σ), $t/\varepsilon = t$, $t[\varepsilon \leftarrow s] = s$, and if $t = \sigma(t_1, \ldots, t_k)$, $i \in [k]$, and $u \in V(t_i)$, then $t[iu] = t_i[u]$, $t/iu = t_i/u$, and $t[iu \leftarrow s] = \sigma(t_1, \ldots, t_i[u \leftarrow s], \ldots, t_k)$.

Let $u \in \mathbb{N}^*$. For every $j \geq 1$, u is the *parent* of uj, denoted by $\text{parent}(uj)$, and j is the *child number* of uj, denoted by $\text{childno}(uj)$. Moreover, we define $\text{childno}(\varepsilon) = 0$.

Let Σ be a ranked alphabet. For a tree $t \in T_\Sigma$, yt denotes the *yield* of t, i.e., the string in $(\Sigma^{(0)} - \{e\})^*$ obtained by reading the leaves of t from left to right, omitting nodes labeled by the special symbol e of rank 0 (e.g., for $t = \sigma(a, \sigma(e, b))$, $yt = t[1]t[22] = ab$). The string yt can be obtained recursively as follows; if $t = e$ then $yt = \varepsilon$, if $t \in \Sigma^{(0)} - \{e\}$ then $yt = t$, and if $t = \sigma(t_1, \ldots, t_k)$, $k \geq 1$, $\sigma \in \Sigma^{(k)}$, and $t_1, \ldots, t_k \in T_\Sigma$, then $yt = yt_1 \cdots yt_k$.

A ranked alphabet Σ is *monadic* if all its symbols are of rank one, except the special symbol e of rank zero, i.e., if $\Sigma = \Sigma^{(1)} \cup \{e^{(0)}\}$; a tree in T_Σ is a *monadic tree.* For a monadic tree $t = a_1(a_2(\cdots a_m(e)))$, pt denotes the *path of* t, i.e., the string $a_1 \cdots a_m \in (\Sigma^{(1)})^*$.

2.2 Tree Substitution

First, we define string substitution: For strings $v, w_1, \ldots, w_n \in A^*$ and distinct $a_1, \ldots, a_n \in A$, we denote by $v[a_1 \leftarrow w_1, \ldots, a_n \leftarrow w_n]$ the result of (simultaneously) substituting w_i for every occurrence of a_i in v. Note that the substitution $[a_1 \leftarrow w_1, \ldots, a_n \leftarrow w_n]$ is a homomorphism on strings. Let P be a condition on a and w such that $\{(a, w) \mid P\}$ is a partial function. Then we use, similar to set notation, $[a \leftarrow w \mid P]$ to denote the substitution $[L]$, where L is the list of all $a \leftarrow w$ for which condition P holds. Since trees are strings, we can use ordinary string substitution to replace leaves in a tree: for α of rank zero, $t[\alpha \leftarrow s]$ is the tree obtained from t by replacing each node labeled α by the tree s. This type of tree substitution (i.e., replacing leaves) is often called “first-order tree substitution”; note that top-down tree transducers and also pebble tree transducers are based on this type of substitution.

Recall from the previous subsection that for a node u of t, $t[u \leftarrow s]$ is the tree obtained by replacing in t the subtree rooted at u by s. This type of tree substitution (i.e., replacing a subtree) is also often called first-order tree substitution. Note that if $\{u_1, \ldots, u_n\}$ is the set of all α-labeled nodes in t and α is of rank zero, then $t[\alpha \leftarrow s] = t[u_1 \leftarrow s] \cdots [u_n \leftarrow s]$.

We now turn to a different type of substitution, which is used in macro tree transducers: “second-order tree substitution”. It means to replace in a tree a symbol of arbitrary rank by a tree s. Here, the question arises how to deal with the subtrees of a symbol of rank $k \geq 1$ that is replaced. We use, at leaves of s, the (formal) parameters $y_1, \ldots, y_k$ as placeholders for the 1st, ..., kth subtrees of the symbol being replaced.

As for first-order tree substitution, let us first define the explicit replacement of a node u in t. Let k be the rank of u, i.e., $t[u] \in \Sigma^{(k)}$, and let s be a tree with parameters in Y_k, i.e., $s \in T_\Sigma(Y_k)$. Then the *second-order substitution* of s at u in t, denoted by $t[\![u \leftarrow s]\!]$, is the tree obtained by replacing in t the subtree rooted at u by s, in which each y_j is replaced by the jth subtree t/uj of u in t; thus, $t[\![u \leftarrow s]\!]$ can be defined in terms of first-order substitution as

$$t[\![u \leftarrow s]\!] = t[u \leftarrow s[y_j \leftarrow t/uj \mid j \in [k]]].$$

Note, by the way, that $t[\![u \leftarrow s]\!] = t[u \leftarrow s]$ in the case that s does not contain parameters.

Next, we define the second-order tree substitution of all σ's (of rank k) in t by the tree $s \in T_\Sigma(Y_k)$. Let $\sigma_1, \ldots, \sigma_n$ be distinct elements of Σ, $n \geq 1$, and for each $i \in [n]$ let s_i be a tree in $T_\Sigma(Y_{k_i})$, where $k_i = \text{rank}_\Sigma(\sigma_i)$. The *second-order tree substitution* of σ_i by s_i in t, denoted by $t[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ is inductively defined as follows (abbreviating $[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ by $[\![\ldots]\!]$). For $t = \sigma(t_1, \ldots, t_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $t_1, \ldots, t_k \in T_\Sigma$, (i) if $\sigma = \sigma_i$ for an $i \in [n]$, then $t[\![\ldots]\!] = s_i[y_j \leftarrow t_j[\![\ldots]\!] \mid j \in [k]]$ and (ii) otherwise $t[\![\ldots]\!] = \sigma(t_1[\![\ldots]\!], \ldots, t_k[\![\ldots]\!])$. We will say that $[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ is a second-order tree substitution over Σ. Note that it is a mapping from T_Σ to T_Σ. In fact, it is a tree homomorphism [GS84]. Let P be a condition on σ and s such that $\{(\sigma, s) \mid P\}$ is a partial function. Then we use $[\![\sigma \leftarrow s \mid P]\!]$ to denote the substitution $[\![L]\!]$, where L is the list of all $\sigma \leftarrow s$ for which condition P holds. In second-order tree substitutions we use for the relabeling $\sigma \leftarrow \delta(y_1, \ldots, y_k)$ of $\sigma^{(k)}$ by $\delta^{(k)}$ the abbreviation $\sigma \leftarrow \delta$; note that this is, in fact, a string substitution.

We will use elementary properties of second-order substitution (both $t[\![u \leftarrow s]\!]$ and $t[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$) without proof. For instance, (just as ordinary substitution) second-order tree substitution is associative (by the closure of tree homomorphisms under composition, cf. Theorem IV.3.7 of [GS84]), i.e., $t[\![\sigma \leftarrow s]\!][\![\sigma \leftarrow s']\!] = t[\![\sigma \leftarrow s[\![\sigma \leftarrow s']\!]]\!]$ and if $\sigma' \neq \sigma$ then $t[\![\sigma \leftarrow s]\!][\![\sigma' \leftarrow s']\!] = t[\![\sigma' \leftarrow s', \sigma \leftarrow s[\![\sigma' \leftarrow s']\!]]\!]$, and similarly for the general case (cf. Sections 3.4 and 3.7 of [Cou83]).

It should be clear that $t[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ can be obtained from t by the iterative application of one-node substitutions $t'[\![u \leftarrow s_i]\!]$. More precisely, let $\Phi = t[\![\sigma_1 \leftarrow s_1, \ldots, \sigma_n \leftarrow s_n]\!]$ and define the binary relation $\Rightarrow_\Phi$ on trees as follows: $t_1 \Rightarrow_\Phi t_2$ if $t_2 = t_1[\![u \leftarrow s_i]\!]$ for some $i \in [n]$ and some $u \in V(t_1)$ with $t_1[u] = \sigma_i$. Note that $\Rightarrow_\Phi^*$ is a congruence, i.e., if $t_i \Rightarrow_\Phi^* t_i'$ then $\sigma(t_1, \ldots, t_k) \Rightarrow_\Phi^* \sigma(t_1', \ldots, t_k')$. Using this and the definition of the second-order tree substitution Φ, it is straightforward to show (by induction on the structure of t) that $t \Rightarrow_\Phi^* t\Phi$.

2.3 Tree Languages and Tree Grammars

Let Σ be a ranked alphabet. A *tree language* (over Σ) is a subset of T_Σ. Both yield and path (defined in Subsection 2.1) are extended to tree languages in the obvious way, i.e., for $L \subseteq T_\Sigma$, $yL = \{yt \mid t \in L\}$ and $pL = \{pt \mid t \in L\}$ (note that pL is only defined if Σ is monadic). For a class $\mathcal{L}$ of tree languages, $y\mathcal{L} = \{yL \mid L \in \mathcal{L}\}$ and $p\mathcal{L} = \{pL \mid L \in \mathcal{L}\}$.

A *regular tree grammar* is a tuple $G = (N, \Sigma, S_0, P)$ where N is a finite set of nonterminals, Σ is a ranked alphabet, $S_0 \in P$ is the initial nonterminal, and P is a finite set of productions of the form $A \to \zeta$ with $A \in N$ and $\zeta \in T_\Sigma(N)$. For trees $\xi, \xi' \in T_\Sigma(N)$, $\xi \Rightarrow_G \xi'$ if $\xi' = \xi[u \leftarrow \zeta]$ for a leaf u of ξ labeled by $A \in N$ and a production $A \to \zeta$ in P. The tree language generated by G is $L(G) = \{t \in T_\Sigma \mid S_0 \Rightarrow_G^* t\}$. The class of all regular tree languages is denoted by REGT.

We assume the reader to be familiar with the elementary properties of the regular tree languages (see, e.g., [GS84,GS97]).

3 Pebble Tree Transducers

In this section the n-pebble tree transducer (n-ptt) is defined, and two easy results about them are proved. The first one is a normal form for the rules of n-ptts (Lemma 2). After that, we give several examples of n-ptts. Then the second result is proved: a size-to-height relationship for translations of n-ptts (Lemma 7). Using this relationship (and the examples of before), it is shown that there is a proper hierarchy of translations of n-ptts, with respect to the number n of pebbles, and that the class of ptt translations is not closed under composition. In Subsection 3.1 the differences between our definition of n-pebble tree transducer and the original one of [MSV00] are discussed. In Subsection 3.2 it is shown that, under certain conditions, 0-pebble tree transducers are attribute grammars; to be precise, that noncircular deterministic 0-pebble tree transducers compute the same total functions as attribute grammars. Finally, in Subsection 3.3, we explain how n-ptts fit into the framework of RT(S) transducers of [Eng86,EV86]. These subsections are independent from the rest of the paper, and therefore can be skipped.

An n-pebble tree transducer is a finite state device that takes an (ordered, ranked) tree as input and generates a tree as output. It processes the input tree starting in the initial state with its reading head at the root node (i.e., with the root node as "current node"). It then walks on the input tree, from node to node, using n pebbles to find its way. Depending on the current state, the label of the current node and its child number (that is, 0 for the root and $j \geq 1$ for a node that is the jth child of its parent), and on the presence of the pebbles $1, \ldots, n$ at the current node, the transducer can generate a tree as output; the leaves of that tree may contain state-instruction pairs that determine how to proceed. The possible instructions are to move to one of the neighbors of the current node (i.e., to a parent or a child) or to stay there, or to lift or drop a pebble. The pebbles $1, \ldots, n$ are used in a stack-like fashion, i.e., if $l \leq n$ pebbles are on the tree, then at most two instructions concerning pebbles are available: either drop pebble $l+1$ (if $l+1 \leq n$) or lift pebble l (if it is present at the current node).

An n-ptt can be seen as a particular type of functional program: each state is a function with one parameter. The parameter is the "input configuration" h which contains the current node of the input tree and the positions of the pebbles. The function body consists of a case distinction on the input configuration h; more precisely, the case distinction is on $\text{test}(h)$, see below, which is a triple consisting of the label of the current node, the information about which pebbles are at the current node, and the child number of the current node. The function body may contain recursive calls to other functions, and generates output of type output tree.

Definition 1. For $n \geq 0$, an *n-pebble tree transducer* (for short, n-ptt) is a tuple $M = (\Sigma, \Delta, Q, q_0, R)$, where Σ and Δ are ranked alphabets of *input* and *output symbols*, respectively, Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, and R is a finite set of *rules*. A rule is of the form $\langle q, \sigma, b, j \rangle \to \zeta$ where ζ is of one of the two forms

$$\zeta = \begin{cases} \langle q', \varphi \rangle \\ \delta(\langle q_1, \text{stay} \rangle, \ldots, \langle q_k, \text{stay} \rangle) \end{cases}$$

for $q \in Q$, $\sigma \in \Sigma$, $b \in \{0,1\}^{\leq n}$, $j \in [0, J]$ with $J = \max\{\text{rank}_\Sigma(\sigma) \mid \sigma \in \Sigma\}$, $q' \in Q$, $\varphi \in I_{\sigma,b,j}$, $\delta \in \Delta^{(k)}$, $k \geq 0$, and $q_1, \ldots, q_k \in Q$. The set $I_{\sigma,b,j}$ of *instructions* is defined as

$$\{\text{stay}\} \cup \{\text{up} \mid j \neq 0\} \cup \{\text{down}_i \mid i \in [\nu]\} \cup \{\text{drop} \mid l < n\} \cup \{\text{lift} \mid l \geq 1, b(l) = 1\}$$

where $\nu = \text{rank}_\Sigma(\sigma)$ and $l = |b|$. A rule r as above is called $\langle q, \sigma, b, j \rangle$-rule or q-rule, and its right-hand side ζ is denoted by $\text{rhs}(r)$. For a subset Q' of Q, a q-rule with $q \in Q'$ is also called Q'-rule.

If Σ and Δ are monadic then M is *monadic.* If for every q, σ, b, and j there is at most one $\langle q, \sigma, b, j \rangle$-rule in R, then M is *deterministic* (for short, M is an n-$\underline{\text{d}}$ptt). If there is at least one such rule then M is *total.* □

If an n-ptt M is monadic (recall the definition of monadic trees from Subsection 2.1) and if we view monadic trees as strings, then the resulting string-to-string translations realized by monadic n-ptts are the same as those realized by the *two-way n-pebble string transducers* of [EM02b] (and similarly for the deterministic transducers). Viewing a monadic tree t as a string corresponds to taking its path pt, i.e., the string $a_1 \cdots a_m$ for $t = a_1(a_2(\cdots a_m(e) \cdots))$.

Let us now discuss how, for a given input tree $s \in T_\Sigma$, the n-ptt M computes an output tree. An (*n-pebble*) *input configuration* (*on s*) is a pair $h = (u, \pi)$, where $u \in V(s)$ and $\pi \in V(s)^{\leq n}$. The set of all n-pebble input configurations on s is denoted by $\text{IC}_{n,s}$. The input configuration (u, π) means that the reading head of M is at node u, that there are $l = |\pi|$ pebbles on the tree, and that the pebbles $1, \ldots, l$ are present at the nodes $\pi(1), \ldots, \pi(l)$, respectively.

By 'testing' the configuration h, M can determine the label σ of the current node u, the bit string b (of length l) that has the ith bit set iff the ith pebble is at

u, and the child number j of u (see Subsection 2.1 for the notion of child number). Thus, we define $\mathrm{test}(h)$ as the triple (σ, b, j), where $\sigma = s[u]$, $b(i) = (\pi(i) = u)$ for $i \in [l]$, and $j = \mathrm{childno}(u)$. For $\mathrm{test}(h) = (\sigma, b, j)$ and an instruction $\varphi \in I_{\sigma,b,j}$, the *execution of φ on h*, denoted by $\varphi(h)$, is the input configuration defined as

$$\varphi(h) = \varphi((u,\pi)) = \begin{cases} (u,\pi) & \text{if } \varphi = \mathrm{stay} \\ (\mathrm{parent}(u),\pi) & \text{if } \varphi = \mathrm{up} \\ (ui,\pi) & \text{if } \varphi = \mathrm{down}_i \\ (u,\pi u) & \text{if } \varphi = \mathrm{drop} \\ (u,[\pi(1);\ldots;\pi(l-1)] & \text{if } \varphi = \mathrm{lift} \end{cases}$$

Note that π is a string of strings and that $[\pi(1);\ldots;\pi(l-1)]$ is the string consisting of the strings $\pi(1),\ldots,\pi(l-1)$; cf. the beginning of the Preliminaries. Thus, $[\pi(1);\ldots;\pi(l-1)]$ is the unique π' such that $\pi = \pi' u$.

A *configuration of M on s* is a pair $\langle q, h\rangle \in \langle Q, \mathrm{IC}_{n,s}\rangle$. It means that q is the current state and h is the current input configuration. The set $\langle Q, \mathrm{IC}_{n,s}\rangle$ of all configurations of M on s is denoted $C_{M,s}$. A rule $\langle q,\sigma,b,j\rangle \to \zeta$ of M is *applicable to* $\langle q,h\rangle$ if $(\sigma,b,j) = \mathrm{test}(h)$. A *sentential form* (*of M on s*) is a tree in $T_\Delta(C_{M,s})$, containing the already produced output and the configurations at which the computation of M may continue.

The *computation relation of M on s*, denoted by $\Rightarrow_{M,s}$, is the binary relation over $T_\Delta(C_{M,s})$ defined as follows: for $\xi, \xi' \in T_\Delta(C_{M,s})$, $\xi \Rightarrow_{M,s} \xi'$ iff there are

(N) a leaf v of ξ labeled by $\langle q,h\rangle \in C_{M,s}$, and
(R) a rule $\langle q,\sigma,b,j\rangle \to \zeta$ in R applicable to $\langle q,h\rangle$

such that $\xi' = \xi[v \leftarrow \eta]$ where η equals

– $\langle q', \varphi(h)\rangle$ if $\zeta = \langle q', \varphi\rangle$, and
– $\delta(\langle q_1,h\rangle,\ldots,\langle q_k,h\rangle)$ if $\zeta = \delta(\langle q_1,\mathrm{stay}\rangle,\ldots,\langle q_k,\mathrm{stay}\rangle)$.

Note that $\xi' = \xi[v \leftarrow \zeta[h]_{M,s}]$ where

$$[h]_{M,s} = [\langle q',\varphi\rangle \leftarrow \langle q', \varphi(h)\rangle \mid q' \in Q, \varphi \in I_{\mathrm{test}(h)}]. \qquad (\#)$$

A computation of M on an input tree s always starts at the root node ε of s, and with no pebbles present; in other words, the initial configuration is $\langle q_0, h_0\rangle$, where the initial input configuration h_0 is defined as (ε, λ). Recall, from the beginning of the Preliminaries, that ε denotes the empty string, and that λ is used to denote the empty string of strings. The *translation realized by M*, denoted by τ_M, is defined as

$$\tau_M = \{(s,t) \in T_\Sigma \times T_\Delta \mid \langle q_0, h_0\rangle \Rightarrow^*_{M,s} t\}.$$

Two transducers are *equivalent*, if they realize the same translation. The class of all translations realized by n-ptts is denoted by n-PTT, and in the deterministic case by n-DPTT. The unions $\bigcup_{n\geq 0} n$-PTT and $\bigcup_{n\geq 0} n$-DPTT are denoted by PTT and DPTT, respectively. It should be clear that for a deterministic n-ptt M, τ_M is a function (cf. Lemma 20 where this fact is proved for the more general case of deterministic n-pebble *macro* tree transducers).

Note that for $n \geq 0$, n-PTT(REGT) denotes the class of all tree languages $\tau_M(R) = \{t \mid (s,t) \in \tau_M \text{ for some } s \in R\}$ where M is an n-ptt and R is a regular tree language. This is the class of *output languages* of n-PTT. From the point of view of databases it is the class of views corresponding to queries realized by n-ptts (on some type R). In fact, we will use similar terminology for any class of tree transducers.

Since pebble tree transducers, just as regular tree grammars, are based on first-order tree substitution, it is quite obvious to see that for a fixed input tree the computations of an n-ptt can be simulated by a regular tree grammar. Formally, let $M = (\Sigma, \Delta, Q, q_0, R)$ be an n-ptt and let $s \in T_\Sigma$ be an input tree. As stated in Proposition 3.5 of [MSV00], there is a regular tree grammar $G_{M,s}$ such that its derivations correspond to the computations by $\Rightarrow_{M,s}$. In fact, the nonterminals of $G_{M,s}$ are the configurations $\langle q, h\rangle$ in $C_{M,s}$, with initial nonterminal $\langle q_0, h_0\rangle$, and if $\langle q, h\rangle \Rightarrow_{M,s} \xi$ then $G_{M,s}$ has the production $\langle q, h\rangle \to \xi$. Clearly, $G_{M,s}$ generates the tree language $\tau_M(s) \subseteq T_\Delta$.

PTTs with general rules. When constructing the rules of a ptt, it is convenient not to be restricted to the two forms of possible right-hand sides of Definition 1, i.e., either "navigation" (viz. $\langle q, \varphi\rangle$) or "output one symbol" (viz. $\delta(\langle q_1, \mathrm{stay}\rangle, \dots, \langle q_k, \mathrm{stay}\rangle)$). It should be intuitively clear that we can allow any tree ζ over output symbols and symbols $\langle q, \varphi\rangle$ as right-hand side of a rule, without changing the expressiveness of the model. Roughly speaking, such a right-hand side ζ can be simulated by a subprogram that generates ζ, using only rules with right-hand sides of the above two kinds (navigation or output).

A rule of the form $\langle q, \sigma, b, j\rangle \to \zeta$ with $\zeta \in T_\Delta(\langle Q, I_{\sigma,b,j}\rangle)$ is a *general rule*, and an n-ptt *with general rules* is a tuple $M = (\Sigma, \Delta, Q, q_0, R)$ where R is a finite set of general rules (and the rest is as for an n-ptt). For M, the notions 'deterministic', 'total', and 'monadic' are defined in the same way as for an n-ptt. Recall the definition of the computation of an n-ptt. The computation relation for a ptt with general rules is defined as follows: $\xi \Rightarrow_{M,s} \xi'$ iff there are (N) a leaf v of ξ labeled by $\langle q, h\rangle \in C_{M,s}$, and (R) a rule $\langle q, \sigma, b, j\rangle \to \zeta$ in R applicable to $\langle q, h\rangle$, such that

$$\xi' = \xi[v \leftarrow \zeta[h]_{M,s}],$$

where $[h]_{M,s}$ is defined in equation (#) above.

Lemma 2. For every n-ptt M with general rules there is an equivalent n-ptt M'. If M is deterministic, then so is M'.

Proof. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be an n-ptt with general rules. The construction of the rules of the n-ptt M' is similar to the construction of productions in normal form for a regular tree grammar (cf. Lemma 3.4 of [GS84]). Let $M' = (\Sigma, \Delta, Q \cup Q_\mathrm{r}, q_0, R')$ be defined as follows. Consider a rule $\langle q, \sigma, b, j\rangle \to \zeta$ in R. Let (ζ, ε) be a state in Q_r and let the rule

$$\langle q, \sigma, b, j\rangle \to \langle (\zeta, \varepsilon), \mathrm{stay}\rangle$$

be in R'. For every $w \in V(\zeta)$ let (ζ, w) be a state in Q_r and let the rule

$$\langle (\zeta, w), \sigma, b, j\rangle \to \zeta[w](\langle (\zeta, w1), \mathrm{stay}\rangle, \dots, \langle (\zeta, wk), \mathrm{stay}\rangle)$$

be in R', where k is the rank of the label $\zeta[w]$ of w. Obviously, M' is an n-ptt and $\tau_{M'} = \tau_M$.

Actually, this lemma is just an easy special case of Theorem 16 in Section 5 (more precisely, the case that all states of the "n-pmtt" M have rank zero; then M is an n-ptt with general rules). Thus, the proof of Theorem 16 contains a formal correctness proof of the above construction. □

Convention 3. From now on, when defining an n-ptt (or n-dptt) we tacitly give the definition of one with general rules, without explicitly mentioning that Lemma 2 should be applied in order to obtain an equivalent n-ptt (or n-dptt). Note that from this point of view Lemma 2 is a normal form result.

Examples. We now give several examples of pebble tree transducers. We start with deterministic transducers without pebbles: In Example 4 two deterministic 0-pebble tree transducers are defined, such that their composition has an exponential size-to-height relationship; this will be used later in this section to prove that PTT and DPTT are not closed under composition. In Example 5, a deterministic monadic n-ptt, $n \in \mathbb{N}$, is defined which has polynomial size increase with exponent $n+1$; it will be used later in this section to prove that the translations of n-ptts and of n-dptts form hierarchies with respect to the number n of pebbles. Finally, in Example 6, an example of a nondeterministic 0-pebble transducer is given that translates each input tree into infinitely many different output trees; this example will play a special role in Section 5.

Example 4. Let $\Sigma = \{a^{(1)}, e^{(0)}\}$ and $\Delta = \{\sigma^{(2)}, e^{(0)}\}$. The first 0-dptt M_1 translates a monadic tree (in T_Σ) of size $m+1$ (i.e., a tree s with $ps = a^m$, cf. the definition of the "path" ps of a monadic tree s in Subsection 2.1) into a full binary tree (in T_Δ) with 2^m leaves. Let $M_1 = (\Sigma, \Delta, \{q\}, q, R_1)$ where, for $j \in \{0,1\}$, R consists of the following (general) rules

$$\begin{aligned}
&\langle q, a, \lambda, j\rangle \to \sigma(\langle q, \mathrm{down}_1\rangle, \langle q, \mathrm{down}_1\rangle)\\
&\langle q, e, \lambda, j\rangle \to e.
\end{aligned}$$

Obviously, the tree $t_m = \tau_{M_1}(a^m(e))$ is a full binary tree with 2^m leaves, i.e., with yield $yt_m = e^{2^m}$.

The next 0-dptt M_2 translates a binary tree (in T_Δ) with m leaves into a monadic tree (in T_Σ) of size $m+1$, i.e., into the tree $a^m(e)$. Let $M_2 = (\Delta, \Sigma, \{d, d', u\}, d, R_2)$ and let the following (general) rules be in R_2.

$$\begin{array}{lll}
\langle d, \sigma, \lambda, j\rangle & \to \langle d, \mathrm{down}_1\rangle & \text{for } j \in [0,2]\\
\langle d, e, \lambda, 1\rangle & \to a(\langle d', \mathrm{up}\rangle) & \\
\langle d', \sigma, \lambda, j\rangle & \to \langle d, \mathrm{down}_2\rangle & \text{for } j \in [0,2]\\
\langle d, e, \lambda, 2\rangle & \to a(\langle u, \mathrm{up}\rangle) & \\
\langle d, e, \lambda, 0\rangle & \to a(e) & \\
 & & \\
\langle u, \sigma, \lambda, 1\rangle & \to \langle d', \mathrm{up}\rangle & \\
\langle u, \sigma, \lambda, 2\rangle & \to \langle u, \mathrm{up}\rangle & \\
\langle u, \sigma, \lambda, 0\rangle & \to e &
\end{array}$$

Obviously, M_2 performs a depth-first left-to-right tree traversal on its input tree $s \in T_\Sigma$, outputting an a for each leaf (labeled e) of s. Each σ-labeled node is visited three times by M_2 (in states d, d', and u, respectively) and each e-labeled node is visited once (in state d).

Finally, consider the composition

$$\tau = \tau_{M_1} \circ \tau_{M_2} = \{(a^m(e), a^{2^m}(e)) \mid m \in \mathbb{N}\}.$$

The size of $\tau(s)$ is $2^{\mathrm{size}(s)-1} + 1$, i.e., τ is of exponential size increase. Thus, τ has a non-polynomial size-to-height relationship (because the height of a monadic tree equals its size). □

Recall from Definition 1 that an n-ptt is monadic if its input and output alphabets are monadic. The next example presents, for $n \in \mathbb{N}$, the monadic n-dptt M_n such that

$$\tau_{M_n} = \{(a^{m-1}(e), a^{k-1}(e)) \mid k = m^{n+1}\},$$

i.e., it has polynomial size increase with exponent $n+1$. It will be proved later (Lemma 7) that this is indeed the maximal size increase of a monadic n-ptt.

Example 5. Let $\Sigma = \Delta = \{a^{(1)}, e^{(0)}\}$. Let M_0 be a 0-dptt that realizes the identity on all input trees in T_Σ: M_0 has set of states $Q_0 = \{q_0\}$ and, for $j \in \{0,1\}$, it has the rules

$$\begin{aligned}\langle q_0, a, \lambda, j\rangle &\to a(\langle q_0, \text{down}_1\rangle)\\ \langle q_0, e, \lambda, j\rangle &\to e\end{aligned}$$

For every $n \geq 0$ we now define inductively the $(n+1)$-dptt M_{n+1} which, above each symbol in an output tree of M_n, inserts a copy of the corresponding input tree (more precisely, of the monadic piece a^{m-1} of the input tree $a^{m-1}(e)$). The idea of the construction is as follows. Whenever M_n generates an output symbol δ, the new $(n+1)$-dptt M_{n+1} instead drops a pebble at the current node u, and changes into a new state q_{up}. In state q_{up} it moves to the root of the input tree s. Then it changes into the state q_{down} in which it moves down to the leaf of s, copying each a of the input tree. Finally, it changes into the state q_{find} and searches for the node with the most recently placed pebble, i.e., the node u. Once at u, it lifts the pebble, outputs δ, and proceeds according to the rules of M_n (doing the same as above whenever output is generated).

For $n \geq 0$ define $M_{n+1} = (\Sigma, \Delta, Q_{n+1}, q_0, R_{n+1})$ with

- $Q_{n+1} = Q_n \cup \{q_c \mid q \in Q_n, c \in \{\text{up}, \text{down}, \text{find}, \text{back}\}\}$
- For every rule $r = (\langle q, \sigma, b, j\rangle \to \zeta)$ in R_n: if $\zeta \in \langle Q_n, I_{\sigma,b,j}\rangle$ then let r be in R_{n+1}, and otherwise (i.e., $\zeta = e$ or $\zeta = a(\langle q', \text{stay}\rangle)$ with $q' \in Q$) let the rules

$$\begin{aligned}\langle q, \sigma, b, j\rangle &\to \langle q_{\text{up}}, \text{drop}\rangle\\ \langle q_{\text{back}}, \sigma, b, j\rangle &\to \zeta\end{aligned}$$

be in R_{n+1}. For every $q \in Q_n$, $b \in \{0,1\}^{\leq n+1}$, and $b' \in \{0,1\}^{\leq n}$ let the following rules be in R_{n+1}:

$$\begin{array}{lll}\langle q_{\text{up}}, \sigma, b, 1\rangle & \to \langle q_{\text{up}}, \text{up}\rangle & \text{for } \sigma \in \Sigma\\ \langle q_{\text{up}}, \sigma, b, 0\rangle & \to \langle q_{\text{down}}, \text{stay}\rangle & \text{for } \sigma \in \Sigma\\ \langle q_{\text{down}}, a, b, j\rangle & \to a(\langle q_{\text{down}}, \text{down}_1\rangle) & \text{for } j \in \{0,1\}\\ \langle q_{\text{down}}, e, b, j\rangle & \to \langle q_{\text{find}}, \text{stay}\rangle & \text{for } j \in \{0,1\}\\ \langle q_{\text{find}}, \sigma, b'0, 1\rangle & \to \langle q_{\text{find}}, \text{up}\rangle & \text{for } \sigma \in \Sigma\\ \langle q_{\text{find}}, \sigma, b'1, j\rangle & \to \langle q_{\text{back}}, \text{lift}\rangle & \text{for } \sigma \in \Sigma, j \in \{0,1\}.\end{array}$$

Clearly, M_{n+1} is deterministic, i.e., $\tau_{M_{n+1}} \in (n+1)$-DPTT. Let us now show that M_{n+1} has polynomial size increase with exponent $n+2$. Consider an input tree $s = a^{m-1}(e)$, $m \geq 1$. Then $\tau_{M_0}(s) = s$. The 1-dptt M_1 inserts a^{m-1} above each of the m symbols of $\tau_{M_0}(s)$, i.e., $\tau_{M_1}(s)$ has $k - 1 = (m-1)m + (m-1)$ occurrences of a, and thus its size is $k = m^2 = \text{size}(s)^2$. In general we get

$$\begin{aligned}\text{size}(\tau_{M_{n+1}}(s)) &= (\text{size}(s) - 1) \cdot \text{size}(\tau_{M_n}(s)) + \text{size}(\tau_{M_n}(s))\\ &= \text{size}(s) \cdot \text{size}(\tau_{M_n}(s))\\ &= \text{size}(s)^{n+2}.\end{aligned}$$

Finally note that instead of defining M_n recursively, it would have also been possible to give a direct construction of an n-dptt that realizes the same translation as M_n: it systematically generates all possible configurations in which all n pebbles are present, starting with all the pebbles and the reading head at the root node and ending with all the pebbles and the reading head at the leaf, generating an a for each such configuration. Obviously, there are $\text{size}(s)^{n+1}$ such configurations. □

Example 6. Let Σ be a ranked alphabet, $J = \max\{\text{rank}_\Sigma(\sigma) \mid \sigma \in \Sigma\}$, and let $\Delta = \Sigma \cup \{\bar{\sigma}^{(1)} \mid \sigma \in \Sigma\}$. Let $\text{mon}_\Sigma \subseteq T_\Sigma \times T_\Delta$ be the translation consisting of all pairs (s,t) such that t is obtained from s by inserting, above each σ-labeled node in s, an arbitrary number of unary symbols $\bar{\sigma}$ (we use 'mon' to stand for "monadic insertion"). The following nondeterministic 0-ptt M_Σ realizes the translation mon_Σ.

Let $M_\Sigma = (\Sigma, \Delta, \{q\}, q, R)$ where, for every $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $j \in [0, J]$, the following rules are in R.

$$\langle q, \sigma, \lambda, j\rangle \to \bar{\sigma}(\langle q, \text{stay}\rangle)$$
$$\langle q, \sigma, \lambda, j\rangle \to \sigma(\langle q, \text{down}_1\rangle, \ldots, \langle q, \text{down}_k\rangle)$$

It should be clear that indeed $\tau_{M_\Sigma} = \text{mon}_\Sigma$.

Note that mon_Σ is an instance of a "regular insertion" (see, e.g., Section 2.3 of [Eng82]), which inserts strings (seen as monadic trees) of an arbitrary regular language R_σ above each symbol σ of an input tree. $\square$

Size-to-Height Relationship of PTT Translations. In the next lemma we show an elementary property of the translation realized by an n-ptt: for a given input tree, the height of an output tree is either unbounded or it is polynomially bounded by the size of the input tree, where the exponent of the polynomial is $n+1$. This is due to the fact that the number of possible configurations on the input tree is polynomially bounded by its size.

Lemma 7. Let M be an n-ptt. There is a $c > 0$ such that for every input tree s, if $\tau_M(s)$ is finite then $\text{height}(t) \leq c \cdot \text{size}(s)^{n+1}$ for every output tree $t \in \tau_M(s)$.

Proof. Let $M = (\Sigma, \Delta, Q, q_0, R)$ and $s \in T_\Sigma$. We claim that if $\tau_M(s)$ is finite then $\text{height}(t) \leq |C_{M,s}|$ for every $t \in \tau_M(s)$. Since the number of configurations of M on s is at most $|Q| \cdot \text{size}(s) \cdot (\text{size}(s) + 1)^n$ (state, current node, and the position of the n pebbles), this shows the lemma for, e.g., $c = |Q| \cdot 2^n$.

To prove the claim, consider the regular tree grammar $G'_{M,s}$ with set of nonterminals $C_{M,s}$, initial nonterminal $\langle q_0, h_0\rangle$, and all productions $\langle q, h\rangle \to \delta(\langle q_1, h_1\rangle, \ldots, \langle q_k, h_k\rangle)$ such that $\delta \in \Delta^{(k)}$, $k \geq 0$, and $\langle q, h\rangle \Rightarrow^*_{M,s} \delta(\langle q_1, h_1\rangle, \ldots, \langle q_k, h_k\rangle)$. It should be clear that the language $L(G'_{M,s})$ generated by $G'_{M,s}$ equals $\tau_M(s)$. It should also be clear, by the usual pumping argument (see, e.g., Proposition 5.2 of [GS97]), that if $t \in L(G'_{M,s})$ has height larger than $|C_{M,s}|$, which is the number of nonterminals of $G'_{M,s}$, then $L(G'_{M,s})$ is infinite.

We note that the proof would work as well with $G_{M,s}$, discussed above after the definition of τ_M, but is even more apparent with $G'_{M,s}$ which generates exactly one output symbol at each derivation step (and thus corresponds to a nondeterministic finite state tree automaton). $\square$

The fact that translations of n-ptts have polynomial size-to-height relationship of input to output tree (Lemma 7), has two immediate consequences:

(1) Hierarchies of Translations. Recall from Example 5 the deterministic monadic n-ptt M_{n+1}, $n \in \mathbb{N}$, and note that $\text{height}(t) = \text{size}(t)$ for every monadic tree t. As was shown in the example, $\text{height}(\tau_{M_{n+1}}(s)) = \text{size}(s)^{n+2}$, which means that there is no c such that $\text{height}(\tau_{M_{n+1}}(s)) \leq c \cdot \text{size}(s)^{n+1}$ for every input tree s. By Lemma 7 we obtain that $\tau_{M_{n+1}}$ cannot be realized by any n-dptt, i.e., $\tau_{M_{n+1}} \notin n\text{-DPTT}$. This proves that

$$\tau_{M_{n+1}} \in (n+1)\text{-DPTT} - n\text{-DPTT},$$

i.e., there is a proper hierarchy of translations of deterministic n-ptts with respect to the number n of pebbles.

In fact, by Lemma 7, even

$$(n+1)\text{-DPTT} - n\text{-PTT} \neq \varnothing,$$

which means that also the translations of nondeterministic n-ptts form a proper hierarchy with respect to the number n of pebbles.

(2) Nonclosure under Composition. Recall from Example 4 the two 0-dptts M_1 and M_2. As was shown in the example, the composition $\tau = \tau_{M_1} \circ \tau_{M_2}$ has exponential size-to-height relationship. Thus, by Lemma 7, τ cannot be realized by any n-ptt, and therefore 0-DPTT $\circ$ 0-DPTT $\nsubseteq$ PTT which means that

DPTT and PTT are not closed under composition.

As discussed in the Introduction, it is an undesirable property of a query language not to be closed under composition: it means that querying a view (i.e., the result of a previous query) might give a result for which there is no direct query on the original database. For this reason, one may argue that the query language of pebble tree transducers determines the classes DPTT* and PTT* of (deterministic and nondeterministic) queries, rather than DPTT and PTT, respectively. Note further, that in the case of monadic trees, the class of two-way pebble string translations corresponding to DPTT *is* closed under composition, as was shown in Theorem 2 of [EM02b] (and so is the class corresponding to 0-DPTT).

3.1 Comparison with the Model of Milo, Suciu, and Vianu

In this subsection our definition of n-pebble tree transducer (Definition 1) is compared to the original definition of [MSV00]. This comparison is not needed in order to understand the remainder of the paper, and hence can be skipped.

The n-pebble tree transducer of [MSV00] translates binary trees, using n pebbles named $1, \ldots, n$. The pebbles are put on the input tree in the order of their names, i.e., if there are l pebbles on the tree, then pebble l is the most recently placed pebble, called the *current pebble*. It acts as the reading head and moves according to the label of the node on which it is (the *current node*), the current state, and the absence or presence of the various other pebbles on the current node. In other words, there are up to $n-1$ "real" pebbles that are tested in the transitions, plus the additional current pebble (the "reading-head-pebble"). To place a new pebble means that the current pebble l remains at the current node, and pebble $l+1$, which now becomes the current pebble, is placed on the root of the input tree. To pick the current pebble $l+1$ means to remove it, making pebble l the current one. In terms of a model with a reading head in place of the current pebble these two operations can be seen as follows: (1) first a pebble is dropped at the node of the reading head, and then the reading head jumps to the root and (2) the reading head jumps to the node of the highest numbered pebble, and then this pebble is lifted.

Our model of n-pebble tree transducer (Definition 1) has a reading head and additionally has n pebbles, that it may drop/lift at the current node, which is the node pointed at by the reading head. Moreover, our transducer has the ability to check whether the current node is the root node, viz. checking, in the left-hand side of a rule, whether the child number equals zero: "is the current node the child of no node?", i.e., "is it the root node?". This is a natural choice because the transducer can check whether the current node is a leaf (by the rank of the node label), i.e., it can recognize the bottom boundary of the input tree, so it should also be able to recognize the top boundary of the input tree, i.e., its root. In the model of [MSV00] a root check can be implemented by placing an extra pebble on the root (or by having a special root symbol). Note that the explicit test for the child number j that is present in the left-hand side of a rule of our transducer, is also present in the model of [MSV00] for $j \neq 0$: it occurs when the applicability of an up$_j$-instruction (with $j = 1, 2$) is determined. Since we are particularly interested in deterministic transducers, it seems more appropriate to explicitly include this test in the left-hand side of a rule, because it leads to a natural definition of determinism: for each left-hand side there should be at most one rule.

Let n-MSV denote the class of tree translations realized by the $(n+1)$-pebble tree transducers of [MSV00] (i.e., having n "real" pebbles), where we drop the restriction to binary ranked alphabets. Denote by n-PTT$_{\text{no-root}}$ the class of tree translations that can be realized by the n-pebble tree transducers obtained from Definition 1 by removing the root-check, i.e., by requiring that if $\langle q, \sigma, b, 0\rangle \to \zeta$ is a rule, then $\langle q, \sigma, b, j\rangle \to \zeta$ is also a rule, for all possible $j \geq 1$. Below we prove the following inclusions, for $n \geq 0$:

$$n\text{-MSV} \subseteq n\text{-PTT} \subseteq (n+1)\text{-PTT}_{\text{no-root}} \subseteq (n+1)\text{-MSV}, \qquad (*)$$

also for the deterministic case.

First inclusion of $(*)$: the move transition $(q, \text{place-new-pebble})$ of an n-MSV transducer can be simulated by an n-ptt by first dropping a pebble and changing into a new state r, and then in r to move up to the root node (recognized by the root-check), at which we change into the state q. The move transition $(q, \text{pick-current-pebble})$ is simulated by changing into state r and then, as before, to move to the root node. Now we search the tree for the highest numbered pebble, which can be realized by a depth-first left-to-right traversal of the tree (cf., e.g., Example 3.3 of [MSV00], and our Example 4). Once arrived at the node that has the highest numbered pebble, we lift it and change to state q.

Second inclusion of $(*)$: To simulate the root-check of an n-PTT, the $(n+1)$-PTT$_{\text{no-root}}$ drops a pebble in its initial configuration, i.e., at the root node; then the root-check is simply realized by checking the presence of this pebble.

Third inclusion of $(*)$: A (q, drop) transition of an $(n+1)$-PTT$_{\text{no-root}}$ can be simulated by an $(n+1)$-MSV transducer in the following way. First place a new pebble, by a transition $(r, \text{place-new-pebble})$. This means that current pebble l remains at the current node, and the new current pebble $l+1$ (the reading-head-pebble) will be at the root. Now search for the pebble l and move to state q once it is found. A (q, lift) transition of an $(n+1)$-PTT$_{\text{no-root}}$ is simulated by a $(q, \text{pick-current-pebble})$ transition of an $(n+1)$-MSV transducer.

Clearly, the above implies that MSV $= \bigcup_{n \geq 0} n$-MSV $=$ PTT and hence our results about the class PTT directly carry over to the class MSV (and similarly in the deterministic case). On the other hand, our results that depend on the number n of pebbles, i.e., results about the classes n-PTT and n-DPTT, should be handled with care when translating them into the model of [MSV00].

3.2 0-PTTs are Attribute Grammars

In this subsection it is shown that 0-dptts and attribute grammars are closely related formalisms and, under certain conditions, realize the same class of translations. Since we do not use this in the remainder of the paper, the subsection can be skipped.

Attribute grammars were introduced by Knuth in [Knu68] to model syntax-directed semantics. They are now the basis of many compiler-compiler systems (see, e.g., [DJL88]). An attribute grammar can be seen as a device which translates the set of trees (i.e., the free algebra) over a many-sorted signature. This is, in fact, the set of derivation trees of a context-free grammar G: the sorts are the nonterminals of G and the symbols are the productions of G (see Section 3 of [GTWW77]). The output trees are interpreted in a semantic domain, i.e., they are viewed as expressions denoting objects in that domain. Thus, an attribute grammar defines a tree-to-object translation. If the interpretation of the output trees is dropped, then an attribute grammar defines a tree-to-tree translation [EF81]. We will only consider one-sorted signatures from now on, for the sake of simplicity. Then the resulting (uninterpreted) attribute grammars are also called attributed tree transducers [Fül81,FV98].

The table in Figure 1 shows the correspondence between deterministic zero pebble tree transducers and attribute grammars (seen as attributed tree transducers).

0-dptt	attribute grammar
states	attributes
initial state	designated attribute at the root
rules	semantic rules that define the attributes

Fig. 1. Correspondence between 0-dptts and attribute grammars.

Attribute grammars (for short, AGs) are total deterministic, and even required to have no infinite computations starting with any sentential form, i.e., they are "noncircular", which, in the 0-ptt notation, means that there is no computation $\langle q, h\rangle \Rightarrow^+_{M,s} \xi$ where $\langle q, h\rangle$ occurs in ξ. This implies that AGs define total functions. Formally, a 0-ptt M is *noncircular*, if there are no input tree s, configuration $c \in C_{M,s}$, and sentential form ξ of M on s such that $c \Rightarrow^+_{M,s} \xi$ and c occurs in ξ (such a configuration c will also be called "circular", cf. Section 5.2).

To understand the formal definition of an attribute grammar as a special type of 0-dptt, we first extend the 0-ptt formalism to have rules with left-hand side $\langle q, \sigma, \tau, \lambda, j\rangle$ where τ is the label of the parent of the current node (or '$-$' if $j = 0$). Clearly, this extension does not change the power of 0-dptts (a 0-dptt M' can simulate an extended one M, because in state q at node u, M' can visit u's parent u', move down to u into state (q, τ) where τ is the label of u', and then apply the $\langle q, \sigma, \tau, \lambda, j\rangle$-rule of M). Furthermore, we allow the extended 0-dptts to use in the right-hand side of a $\langle q, \sigma, \tau, \lambda, j\rangle$-rule the new instruction updown_i, with $1 \le i \le \mathrm{rank}_\Sigma(\tau)$, which is simply a subroutine for moving to the parent of the current node u and then to the ith child (i.e., to the ith sibling of u).

Next we restrict the extended 0-dptts: The attributes (states) are divided into inherited attributes (i-states) and synthesized attributes (s-states). Now the restriction says that the

- rules for s-states are: $\langle q, \sigma, \tau, \lambda, j\rangle$-rules that disregard τ and j and have no up instruction in the right-hand side (and no updown_i), and
- rules for i-states are: $\langle q, \sigma, \tau, \lambda, j\rangle$-rules that disregard σ, have no down_i instruction in the right-hand side, but are allowed to use updown_i.

The extended 0-ptts that fulfill the above two conditions and additionally are total deterministic and noncircular, are called *attributed tree transducers* (for short att). Note that for the $\langle q, \sigma, \tau, \lambda, j\rangle$-rules to disregard, e.g., the symbol σ, means that all $\langle q, \sigma, \tau, \lambda, j\rangle$-rules for $\sigma \in \Sigma$ have the same right-hand side. Note also that, intuitively, the first condition means that for each s-state, at a σ-labeled node, there is a unique applicable rule, and the second condition means that for each i-state, at a jth child of a τ-labeled node, there is a unique applicable rule. Moreover, from an s-state it is not possible to move up, and from an i-state it is not possible to move down, respectively.

Finally note that an attribute grammar is usually specified by giving for each input symbol σ (i.e., each production of the underlying context-free grammar)

$$\begin{cases} \text{all rules } \langle\, q\,, \sigma \quad, [\tau\,, \lambda \quad, j]\,\rangle \to \zeta & q \text{ synthesized} \\ \text{all rules } \langle\, q\,, [\sigma']\,, \sigma\,\,, [\lambda]\,, j\,\,\rangle \to \zeta & q \text{ inherited} \end{cases}$$

where the brackets '[' and ']' around the symbols mean that they are not present in the actual left-hand side of the attribute grammar rule (which is the same as

$$\begin{array}{lll}
\langle d,\sigma\rangle \to \langle d,\mathrm{down}_1\rangle & \langle t,\sigma,1\rangle \to a(\langle d,\mathrm{updown}_2\rangle) & \langle u,\sigma,1\rangle \to \langle d,\mathrm{updown}_2\rangle \\
\langle d,e\rangle \to \langle t,\mathrm{stay}\rangle & \langle t,\sigma,2\rangle \to a(\langle u,\mathrm{up}\rangle) & \langle u,\sigma,2\rangle \to \langle u,\mathrm{up}\rangle \\
 & \langle t,\_,0\rangle \to a(e) & \langle u,\_,0\rangle \to e
\end{array}$$

Fig. 2. An att (with s-state d and i-states t and u) equivalent to M_2 of Example 4.

disregarding it). Figure 2 shows the rules, in this attribute grammar notation, of an att that computes the same translation as the 0-dptt M_2 of Example 4, in a similar way.

Clearly, for every att there is an equivalent noncircular 0-dptt, because an att is an extended 0-dptt. We now show that also the converse holds, i.e., that for every noncircular 0-dptt M that realizes a total function, there is an equivalent att; this proves that such 0-dptts and atts have the same power.

Theorem 8. A total function from T_Σ to T_Δ can be realized by an attributed tree transducer iff it can be realized by a noncircular 0-dptt.

Proof. As stated before, every att is an (extended) noncircular 0-dptt, by definition. It remains to show that for every noncircular 0-dptt M that realizes a total function, there is an equivalent att A. Since τ_M is a total function, we may assume that M is total: this can be achieved by simply adding (dummy) rules for the left-hand sides that do not have a rule (note that these rules will never be applied).

Let $M = (\Sigma, \Delta, Q, q_0, R)$ and let $J = \max\{\mathrm{rank}_\Sigma(\sigma) \mid \sigma \in \Sigma\}$. Note that M is not extended. The att A is constructed as follows:

- s-states: (q,j) with $q \in Q$ and $j \in [0,J]$; initial state: $(q_0,0)$
- i-states: (q,φ) with $q \in Q$ and $\varphi \in \{\mathrm{stay},\mathrm{up}\}$
- rules for s-states:
  For every $\langle q,\sigma,\lambda,j\rangle \to \zeta$ in R and $(\tau,j') \in (\Sigma \times [J]) \cup \{(-,0)\}$, let
  $$\langle (q,j),\sigma,\tau,\lambda,j'\rangle \to \zeta'$$
  be a rule of A, where
  $$\zeta' = \begin{cases} \delta(\langle (q_1,j),\mathrm{stay}\rangle,\dots,\langle (q_k,j),\mathrm{stay}\rangle) & \text{if } \zeta = \delta(\langle q_1,\mathrm{stay}\rangle,\dots,\langle q_k,\mathrm{stay}\rangle) \\ \langle (q',j),\mathrm{stay}\rangle & \text{if } \zeta = \langle q',\mathrm{stay}\rangle \\ \langle (q',i),\mathrm{down}_i\rangle & \text{if } \zeta = \langle q',\mathrm{down}_i\rangle \\ \langle (q',\mathrm{stay}),\mathrm{stay}\rangle & \text{if } \zeta = \langle q',\mathrm{up}\rangle \end{cases}$$
- rules for i-states:
  For every $q \in Q$, $\sigma \in \Sigma$, and $(\tau,j) \in (\Sigma \times [J]) \cup \{(-,0)\}$, let
  $$\begin{array}{ll} \langle (q,\mathrm{stay}),\sigma,\tau,\lambda,j\rangle \to \langle (q,\mathrm{up}),\mathrm{up}\rangle & \text{for } j \neq 0 \\ \langle (q,\mathrm{up}),\sigma,\tau,\lambda,j\rangle \to \langle (q,j),\mathrm{stay}\rangle & \end{array}$$
  be rules of A. Furthermore, A has the (dummy) rule $\langle (q,\mathrm{stay}),\sigma,-,\lambda,0\rangle \to \langle p,\mathrm{stay}\rangle$ where p is an arbitrary state of A.

Note that the rules of A even disregard τ, and do not contain the updown_i instructions. It should be clear that A is equivalent to M, i.e., $\tau_A = \tau_M$. Intuitively, whenever M is in state q at node u, the att A will be in s-state $(q,\mathrm{childno}(u))$ at the same node u. This property is obviously preserved by down and stay moves: If M moves down to its ith child ui into state q', then A moves down to ui into s-state (q',i), and if M stays at u in state q', then A stays at u in s-state $(q',\mathrm{childno}(u))$. Now, if M moves up into state q', then A cannot move up directly, because $(q,\mathrm{childno}(u))$ is an s-state (only i-states are allowed to move up). Thus, A first changes into the i-state (q',stay), then moves up into the i-state (q',up), and finally does a stay move into the s-state (q',j), where $j = \mathrm{childno}(\mathrm{parent}(u))$. It is not difficult to see that A is noncircular, because M is. □

Note that for an attributed tree transducer it is well known that the height of the output tree is linear in the size of the input tree (cf., e.g., Lemma 5.40 of [FV98]); this corresponds to the case $n = 0$ of Lemma 7.

Attribute grammars can also be defined as nondeterministic and partial devices. In fact, the attributed tree transducer of [Fül81] is defined nondeterministically. In [Kam83,FM00] it is shown that domains of (deterministic) partial AGs are the languages recognized by universal tree-walking automata, which, essentially, are the acceptor version of 0-dptts. We finally note that the relationship between 0-ptts and attribute grammars was already pointed out in Section 3 of [Eng86], where 0-ptts are called RT(Tree-walk) transducers; these transducers are discussed in the next subsection.

3.3 Relationship to Grammars with Storage

In this subsection we explain that the n-ptt is an instance of the regular tree S transducer, for a storage type S. This is only needed to understand some of our references to the literature, and hence can be skipped.

Grammars, automata, and transducers with storage have been considered in [Eng86,EV86,EV88], both for strings and for trees. The special case of string automata with storage was extensively investigated in AFL and AFA theory [Gin75]. Here we discuss the regular tree transducers with storage, or RT(S) transducers, where S is an arbitrary storage type (such as the Counter, the Pushdown, or the Stack). Basically, an RT(S) transducer is a $\underline{\text{r}}$egular $\underline{\text{t}}$ree grammar (see Subsection 2.3) of which the nonterminals are viewed as the states of the transducer. Moreover, with each occurrence of a nonterminal in a sentential form a storage configuration of S is associated, and the productions of the grammar are extended with tests and instructions of S that operate on these configurations. Thus, the derivations of the grammar are controlled by the storage configurations. The RT(S) transducer receives one of a set of designated initial storage configurations of S as input (associated with the initial nonterminal), and produces the generated tree as output. This means that it translates initial configurations into trees.

As observed already in the Introduction (and at the end of the previous subsection), the 0-ptt is the same as the RT(Tree-walk) transducer of [Eng86], i.e., the RT(S) transducer where S is the storage type Tree-walk. A storage configuration of Tree-walk consists of an input tree s, together with an input configuration on s, as defined for the 0-ptt, i.e., a node u of s; it is an initial storage configuration if u is the root of s, in which case it is identified with s (and thus, the RT(Tree-walk) transducer indeed translates trees into trees). The tests of the storage type Tree-walk allow to test the label and child number of the node u, and its instructions are the instructions of the 0-ptt, i.e., up, stay, and down$_i$. As an example of a production of an RT(Tree-walk) transducer, consider

$$A[\text{label} = \sigma?\text{childno} = 3?] \;\to\; \delta(\alpha, B[\text{down}_2], C[\text{up}]).$$

Intuitively, this production means that a nonterminal (or state) A which has storage configuration (s, u) where s is an input tree and u a node of s with label σ and child number 3, can be replaced by the right-hand side, in which the nonterminals (or states) B and C have storage configurations $(s, u2)$ and $(s, \text{parent}(u))$, respectively. Thus, it corresponds to the rule $\langle A, \sigma, \lambda, 3\rangle \to \delta(\alpha, \langle B, \text{down}_2\rangle, \langle C, \text{up}\rangle)$ of a 0-ptt.

It should now be clear to the reader that the storage type Tree-walk can easily be extended to the storage type n-Pebble, for every $n \in \mathbb{N}$, such that the RT(n-Pebble) transducer is precisely the n-ptt. Hence, all results for RT(S) transducers proved in, e.g., [Eng86,EV86,EV88] hold in particular for n-ptts.

Another storage type of interest is Tree (denoted TR in [EV86,EV88]): it is Tree-walk without the instructions stay and up, and without the test on child number. We observe here that the RT(Tree) transducer is precisely the top-down tree transducer.

In [EV86,EV88], also context-free tree transducers with storage, or CFT(S) transducers, are investigated. They are defined in the same way as RT(S) transducers, except that context-free tree grammars rather than regular tree grammars are used. In particular, the CFT(Tree) transducer is (a notational variant of) the macro tree transducer. Thus, in this paper, we compare RT(n-Pebble) transducers with CFT(Tree) transducers.

We finally note that with every storage type S is associated the storage type P(S) of pushdowns of S-configurations. It is easy to see (see Section 6(7) of [Eng86]) that every RT(Tree-walk) transducer, i.e., every 0-ptt, can be simulated by an RT(P(Tree)) transducer: roughly speaking, the nodes that are on the path from the root to the current node are pushed on the stack; thus, a down$_i$ instruction is simulated by a push(down$_i$) instruction, which pushes node ui on the pushdown (if u was the node on top of the pushdown), and an up instruction is simulated by popping the pushdown. It is shown in [EV86] that, under certain conditions, the RT(P(S)) transducer has the same power as the CFT(S) transducer.

4 Decomposition of Pebble Tree Transducers

In this section it is proved that each n-pebble tree transducer M can be decomposed into the $(n+1)$-fold composition of 0-pebble tree transducers; more precisely, the first n 0-ptts of the composition are deterministic, and the last one is nondeterministic (and they are all deterministic if M is). This means that for a pebble transducer, a pebble can be simulated by the application of a translation of a deterministic 0-ptt. Thus, instead of taking care of many pebbles at the same time (viz. programming an n-ptt) one can simply consider pebble transducers without pebbles, and sequentially compose them. Note that in the string case an analogous result holds, but with one pebble rather than zero: each n-pebble string transducer can be realized by the composition of n 1-pebble string transducers (Theorem 1 of [EM02b]). The idea of the proof in the string case is similar to, but easier than, the one for trees in this section. The one pebble is really needed: deterministic 0-pebble string transducers are closed under composition (because they are the two-way finite state transducers [CJ77]).

Let us sketch the proof of this decomposition. Let M be an n-ptt, $n \geq 1$. We want to discuss how to decompose M's translation τ_M into the composition of a fixed total function EncPeb, realized by a deterministic 0-ptt, and an $(n-1)$-ptt M'. The idea of the function EncPeb is to add information about the position of the first pebble of M to the input tree. More precisely, the input tree is enlarged by adding to each node, as an additional (last) subtree, a copy of the input tree in which that node is marked. The computation of M on an input tree s is simulated by the $(n-1)$-ptt M' on the input tree EncPeb(s). As long as M has no pebbles on s, M' simulates it on the original nodes of s, of which the labels are primed to distinguish them from the new nodes of EncPeb(s). However, when M drops the first pebble on node v of s, M' instead enters the new subtree of v and walks to the marked node, corresponding to v. In that subtree M' behaves just like M, using pebble i as pebble $i+1$ of M. If M checks for the presence of its first pebble, then M' checks whether the current node is marked. If M lifts its first pebble, then M' returns to v by walking up to the first primed node.

There is one difficulty in the construction sketched above, and that is the precise definition of EncPeb(s). Suppose that, as suggested above, each additional subtree is indeed a precise copy of the input tree, with one node marked by barring its label. Then it is easy to see that EncPeb can be realized by a dptt M_1 with one pebble. In fact, M_1 has states q_0, q_1, and q_2, and the following rules (with $\sigma \in \Sigma^{(k)}$, $j \geq 0$, $j' > 0$, and $b \in \{0, 1\}$):

$$\begin{aligned}
\langle q_0, \sigma, \lambda, j\rangle &\to \sigma'(\langle q_0, \mathrm{down}_1\rangle, \ldots, \langle q_0, \mathrm{down}_k\rangle, \langle q_1, \mathrm{drop}\rangle)\\
\langle q_1, \sigma, b, j'\rangle &\to \langle q_1, \mathrm{up}\rangle\\
\langle q_1, \sigma, b, 0\rangle &\to \langle q_2, \mathrm{stay}\rangle\\
\langle q_2, \sigma, 0, j\rangle &\to \sigma(\langle q_2, \mathrm{down}_1\rangle, \ldots, \langle q_2, \mathrm{down}_k\rangle)\\
\langle q_2, \sigma, 1, j\rangle &\to \overline{\sigma}(\langle q_2, \mathrm{down}_1\rangle, \ldots, \langle q_2, \mathrm{down}_k\rangle)
\end{aligned}$$

Thus, to generate the additional tree, M_1 drops its pebble at the current node, walks to the root, and copies the input tree, putting a bar on the label of the node that carries the pebble.

However, it can be proved that this mapping EncPeb can*not* be realized by a zero-pebble ptt. For this reason, we instead define EncPeb in such a way that the new subtree of node v is a "folded" copy of the input tree s, obtained from s by turning v into the root node. This is done by reversing the direction of the edges on the path from the root to v, i.e., by inverting the parent-child relationship between all ancestors of v. It is not difficult to see that this EncPeb *can* be realized by a zero-pebble ptt (see also Example 3.7 of [MSV]): to generate the new subtree it can just copy the input tree starting at the current node v and "walking away" from v. It should also be clear that the $(n-1)$-ptt M' can still simulate M on this folding of s, provided some additional information is added to the labels of the (ex-)ancestors of v that allows M' to reconstruct the form of s, and, thus, to turn a walk on s into a walk on the folding of s. This information can easily be produced by the zero-pebble ptt. Note that the simulation of the dropping and lifting of the first pebble has even become easier: when it is dropped, M' just moves down one step (to the root of the new subtree), and when it is lifted, M' just moves up one step.

We now give a more precise description of the mapping EncPeb, to prepare for its formal definition. For every input tree s of M, EncPeb(s) has all nodes of the original tree s, but additionally each node v of rank k in the tree s, has rank $k+1$ in EncPeb(s) and its $(k+1)$th subtree is the tree s_v^{dir}, obtained by adding the "re$\underline{\text{dir}}$ection information" mentioned above to the labels of the folding s_v of the input tree s at v. We first describe how the intermediate tree s_v is constructed from s, and then show how to relabel it in order to obtain the tree s_v^{dir}. The tree s_v is obtained from s by inverting the parent-child relationship of all ancestors of u. More precisely, if u is an ancestor of v in s, then, in s_v, the parent of u is swapped with its ith child, where $i = \mathrm{swap}_v(u)$ and

$$\mathrm{swap}_v(u) = \begin{cases} k+1 & \text{if } u = v \\ l & \text{if } v = ulv' \text{ for } l \in \mathbb{N} \text{ and some } v' \in \mathbb{N}^* \end{cases}$$

with $k = \mathrm{rank}_\Sigma(s[v])$. Since v itself has no child that is an ancestor of v, its parent is added as a new, $(k+1)$th child. If u is the root node, then it has no parent, but in order to keep the ranks of the new symbols in s_v^{dir} as uniform as possible, we assume an imaginary parent of u, labeled by a dummy symbol \$. Clearly M' will never visit these \$-labeled nodes in EncPeb(s), because that would correspond to an up instruction of M at the root node, which does not exist.

We now discuss how to relabel s_v in order to obtain the tree s_v^{dir}. Let u be an ancestor of v. Since in s_v the parent of u was swapped with its ith child, $i = \mathrm{swap}_v(u)$, also the corresponding move instructions of the $(n-1)$-ptt M' have to be swapped. We capture this "swapping information" by the set d_i, defined as

$$d_i = \{(\mathrm{up}, \mathrm{down}_i), (\mathrm{down}_i, \mathrm{up})\}.$$

Also, the child number j of u (in s) may have changed in s_v. Thus, for M' to have complete information about the original order of the ancestors of v, we include both d_i and the original child number j of u in the label of the corresponding node in s_v^{dir}. Hence, s_v^{dir} is obtained from s_v by relabeling, for every ancestor u of v, the node corresponding to u by $(s[u], \text{childno}(u), d_{\text{swap}_v(u)})$.

Note that the node of s_v^{dir} corresponding to v, i.e., its root, is marked in the sense that it is the unique node of s_v^{dir} with label (σ, j, d_i) such that $i = \text{rank}_\Sigma(\sigma) + 1$. Note also that, in fact, the child number information is superfluous: if a node of s_v^{dir} has label (σ, j, d_i) and its ith child has label $(\sigma', j', d_{i'})$, then $j = i'$ (and if its ith child has label \$, then $j = 0$). Moreover, even the d_i information is superfluous, because i is the number of the unique child that is an (ex-)ancestor of v (or has label \$). Thus, it would have sufficed to mark all (ex-)ancestors of v. However, the addition of this information simplifies the formal definition of M'.

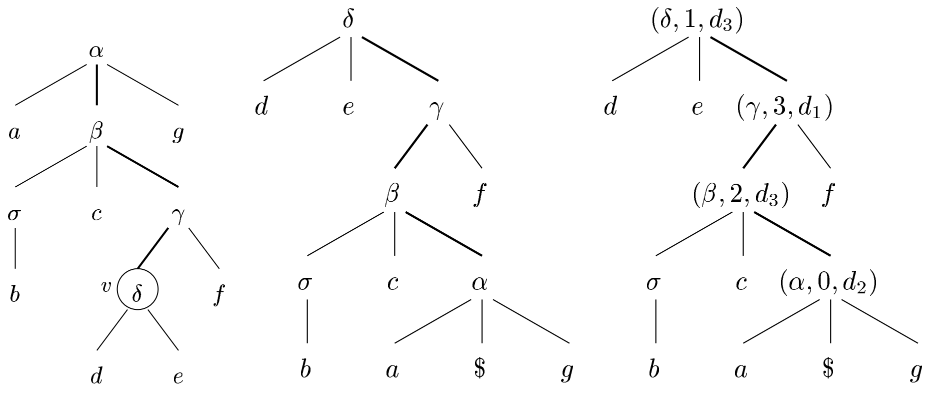

Fig. 3. The trees s, s_v, and s_v^{dir}.

Figure 3 shows a tree s in which the node $v = 231$ is encircled, the corresponding tree s_v which is obtained from s by turning v into the root node and reversing the order of the ancestors of v, as described above, and the tree s_v^{dir} obtained from s_v by relabeling each ancestor of v by the correct triple (σ, j, d_i). As an example of the translation EncPeb, consider Figure 4 which shows the tree $s = \alpha(\beta, \gamma(\delta))$ together with the tree EncPeb(s).

Formally, the tree EncPeb(s) is defined as follows. First, define for every $v \in V(s)$ the function enc_v that maps every $u \in V(s)$ to the corresponding node in the subtree s_v^{dir} of EncPeb(s). Let w be the longest common ancestor of u and v, let $u' \in \mathbb{N}^*$ such that $u = wu'$, and let $w_1 = v, w_2, \ldots, w_m = w$, $m \geq 1$, be the nodes on the path from v to w (i.e., w_i is a child of w_{i+1} for $1 \leq i < m$). Then

$$\text{enc}_v(u) = v(k+1)\text{swap}_v(w_1) \cdots \text{swap}_v(w_{m-1})u'$$

with $k = \text{rank}_\Sigma(s[v])$. Figure 5 shows the nodes u, v, and w_i in the tree s. Obviously, enc_v is an encoding, i.e., for every $u, u' \in V(s)$

(P0) $\text{enc}_v(u) = \text{enc}_v(u')$ iff $u = u'$.

Using $\text{enc}_v(u)$ we can define the set of nodes of EncPeb(s) as

$$\begin{aligned} V(\text{EncPeb}(s)) = V(s) &\cup \{\text{enc}_v(u) \mid u, v \in V(s)\} \\ &\cup \{\text{enc}_v(\varepsilon)\text{swap}_v(\varepsilon) \mid v \in V(s)\}. \end{aligned}$$

The labels of the nodes of EncPeb(s) are as follows. Note that nodes in $V(s)$ are labeled by primed copies of the corresponding symbols of Σ, because their rank in

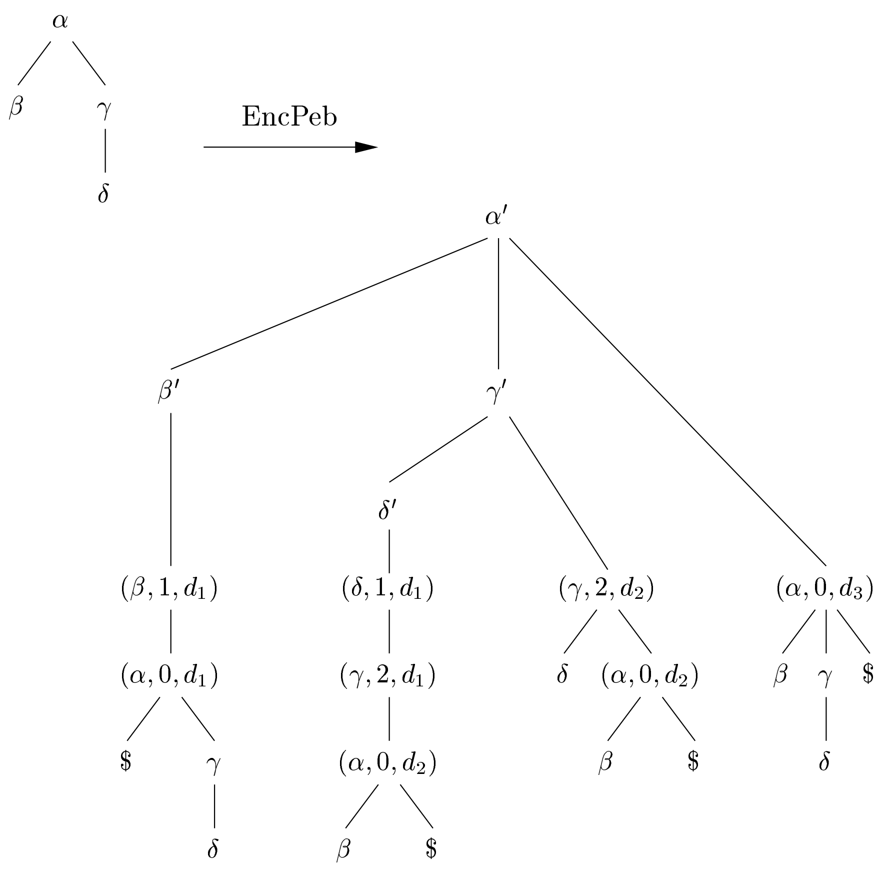

Fig. 4. The trees $s = \alpha(\beta, \gamma(\delta))$ and $\text{EncPeb}(s)$.

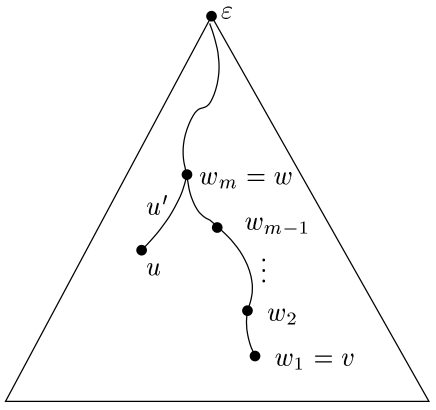

Fig. 5. The nodes u, v, and w_i in the tree s.

EncPeb(s) has increased by one. Denote the tree EncPeb(s) by s'. Then, for every $u, v \in V(s)$,

(P1) $s'[u] = \sigma'$ where $\sigma = s[u]$
(P2) if u is not an ancestor of v then $s'[\mathrm{enc}_v(u)] = s[u]$
(P3) if u is an ancestor of v then $s'[\mathrm{enc}_v(u)] = (s[u], \mathrm{childno}(u), d_{\mathrm{swap}_v(u)})$
(P4) $s'[\mathrm{enc}_v(\varepsilon)\mathrm{swap}_v(\varepsilon)] = \$$.

Note that the information childno(u) is available at node $\mathrm{enc}_v(u)$ of s'. If u is an ancestor of v this is by definition of the relabeling, viz. P3, and otherwise, by the definition of $\mathrm{enc}_v(u)$, we get

(P5) if u is not an ancestor of v then $\mathrm{childno}(u) = \mathrm{childno}(\mathrm{enc}_v(u))$.

In the next lemma the 0-dptt M_{EncPeb} realizing EncPeb is constructed, and, for a given n-ptt M, the $(n-1)$-ptt M' is constructed such that the composition of $\tau_{M_{\mathrm{EncPeb}}}$ and $\tau_{M'}$ equals the translation τ_M realized by M.

Lemma 9. For every $n \geq 1$, n-PTT $\subseteq$ 0-DPTT $\circ$ $(n-1)$-PTT and
n-DPTT $\subseteq$ 0-DPTT $\circ$ $(n-1)$-DPTT.

Proof. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be an n-ptt, and let $J = \max\{\mathrm{rank}_\Sigma(\sigma) \mid \sigma \in \Sigma\}$. We will define the deterministic 0-ptt M_{EncPeb} and the $(n-1)$-ptt M' such that $\tau_M = \tau_{M_{\mathrm{EncPeb}}} \circ \tau_{M'}$. The 0-ptt M_{EncPeb} realizes the mapping EncPeb described above this lemma, i.e., it adds to each node v of rank k of an input tree s, as $(k+1)$th subtree, the tree s_v^{dir} (cf. Figure 3). It has initial state q which copies the current node v of rank k (adding a prime to its label), and spawns the generation of s_v^{dir} as $(k+1)$th subtree, in state q_∞. In the subtree s_v^{dir}, M_{EncPeb} uses states q_ν, $\nu \in [J]$, to denote that the previously processed node had child number ν. Finally, it has a state q_{id} that realizes the identity.

Define $M_{\mathrm{EncPeb}} = (\Sigma, \Gamma, S, q, P)$ with

$$\begin{aligned}\Gamma = \Sigma &\cup \{\sigma'^{(k+1)} \mid \sigma \in \Sigma^{(k)}, k \geq 0\}\\ &\cup \{(\sigma, j, d_i)^{(k)} \mid \sigma \in \Sigma^{(k)}, k \geq 0, i \in [k], j \in [0, J]\}\\ &\cup \{(\sigma, j, d_{k+1})^{(k+1)} \mid \sigma \in \Sigma^{(k)}, k \geq 0, j \in [0, J]\}\\ &\cup \{\$^{(0)}\}\end{aligned}$$

and $S = \{q, q_\infty, q_1, \ldots, q_J, q_{\mathrm{id}}\}$. For every $\sigma \in \Sigma^{(k)}$, $k \geq 0$, $j \in [0, J]$, and $\nu \in [k]$ let the following rules be in P.

$$\begin{aligned}\langle q, \sigma, \lambda, j\rangle &\to \sigma'(\langle q, \mathrm{down}_1\rangle, \ldots, \langle q, \mathrm{down}_k\rangle, \langle q_\infty, \mathrm{stay}\rangle)\\ \langle q_\infty, \sigma, \lambda, j\rangle &\to (\sigma, j, d_{k+1})(\langle q_{\mathrm{id}}, \mathrm{down}_1\rangle, \ldots, \langle q_{\mathrm{id}}, \mathrm{down}_k\rangle, \xi_j)\\ \langle q_\nu, \sigma, \lambda, j\rangle &\to (\sigma, j, d_\nu)(\langle q_{\mathrm{id}}, \mathrm{down}_1\rangle, \ldots, \langle q_{\mathrm{id}}, \mathrm{down}_{\nu-1}\rangle, \xi_j,\\ &\qquad\qquad \langle q_{\mathrm{id}}, \mathrm{down}_{\nu+1}\rangle, \ldots, \langle q_{\mathrm{id}}, \mathrm{down}_k\rangle)\\ \langle q_{\mathrm{id}}, \sigma, \lambda, j\rangle &\to \sigma(\langle q_{\mathrm{id}}, \mathrm{down}_1\rangle, \ldots, \langle q_{\mathrm{id}}, \mathrm{down}_k\rangle)\end{aligned}$$

where $\xi_j = \$$ if $j = 0$, and $\xi_j = \langle q_j, \mathrm{up}\rangle$ if $j \in [J]$. This ends the construction of M_{EncPeb}. It should be clear that indeed $\tau_{M_{\mathrm{EncPeb}}}(s) = \mathrm{EncPeb}(s)$ for every $s \in T_\Sigma$. In particular this implies that the properties P1 – P5 (stated before the lemma) hold for $s' = \tau_{M_{\mathrm{EncPeb}}}(s)$.

We now define the $(n-1)$-ptt $M' = (\Gamma, \Delta, Q, q_0, R')$. Since, in the correctness proof, we will need to know which rules r' in R' were constructed from the rule $r \in R$, we will call r' *related to* r if it is constructed from r. Then R' is defined as $\{r' \mid \exists r \in R : r' \text{ is related to } r\}$.

Let $q \in Q$, $\sigma \in \Sigma^{(k)}$, $k \geq 0$, $b \in \{0,1\}^{\leq n}$, $j \in [0, J]$, and let $r = (\langle q, \sigma, b, j\rangle \to \zeta)$ be a rule in R. The new rules of M' are defined by the following case distinction on the bit string b.

– (zero pebbles) $b = \lambda$: If $\zeta \neq \langle q', \mathrm{drop}\rangle$ for any $q' \in Q$ then let the rule $\langle q, \sigma', \lambda, j\rangle \to \zeta$ be related to r, and otherwise let the rule $\langle q, \sigma', \lambda, j\rangle \to \langle q', \mathrm{down}_{k+1}\rangle$ be related to r.

– (first pebble not at current node) $b = 0b'$ for some $b' \in \{0,1\}^{\leq n-1}$: Let the rule $\langle q, \sigma, b', j\rangle \to \zeta$ be related to r, and, for every $i \in [k]$ and $j' \in [J+1]$, let the rule

$$\langle q, (\sigma, j, d_i), b', j'\rangle \to \zeta[\langle q', \varphi\rangle \leftarrow \langle q', \varphi'\rangle \mid q' \in Q, (\varphi, \varphi') \in d_i]$$

be related to r.

– (first pebble at current node) $b = 1b'$ for some $b' \in \{0,1\}^{\leq n-1}$: If $b' = \lambda$ and $\zeta = \langle q', \mathrm{lift}\rangle$ for $q' \in Q$, then let, for every $j' \in [J+1]$, the rule $\langle q, (\sigma, j, d_{k+1}), b', j'\rangle \to \langle q', \mathrm{up}\rangle$ be related to r, and otherwise let, for every $j' \in [J+1]$, the rule

$$\langle q, (\sigma, j, d_{k+1}), b', j'\rangle \to \zeta[\langle q', \varphi\rangle \leftarrow \langle q', \varphi'\rangle \mid q' \in Q, (\varphi, \varphi') \in d_{k+1}]$$

be related to r. (Remark: the rules with $j' \neq k+1$ are useless, but their presence simplifies the correctness proof.)

This concludes the construction of M'. Clearly, M' is deterministic if M is.

Let $s \in T_\Sigma$ and $s' = \tau_{M_{\mathrm{EncPeb}}}(s)$. In order to prove the correctness of the construction, we extend the notion of relatedness from rules to sentential forms: For $\xi \in T_{\Delta \cup C_{M,s}}$ and $\xi' \in T_{\Delta \cup C_{M',s'}}$: ξ is related to ξ' if $\xi' = \xi[\mathrm{enc}]$, where $[\mathrm{enc}]$ is the substitution $[\langle q, h\rangle \leftarrow \langle q, \mathrm{enc}(h)\rangle \mid q \in Q, h \in \mathrm{IC}_{n,s}]$ and the "encoded" input configuration $\mathrm{enc}(h) \in \mathrm{IC}_{n-1,s}$ of M' is defined as follows: if $h = (u, \lambda)$ with $u \in V(s)$ then $\mathrm{enc}(h) = h$, and if $h = (u, vv_1 \cdots v_l)$ with $u, v, v_1, \ldots, v_l \in V(s)$ and $l \in [0, n-1]$ then

$$\mathrm{enc}(h) = \mathrm{enc}(u, vv_1 \cdots v_l) = (\mathrm{enc}_v(u), \mathrm{enc}_v(v_1) \cdots \mathrm{enc}_v(v_l)).$$

Note that for every rule $r' \in R'$ there is precisely one rule r in R related to r' which we denote by $\mathrm{rel}(r')$. We first show, in Claim 1, that if a rule is applicable to a configuration, then there is related rule applicable to the related configuration, and vice versa.

Claim 1: Let $\langle q, h\rangle \in C_{M,s}$ and $r \in R$.

r is applicable to $\langle q, h\rangle$ iff there is a rule $r' \in R'$ such that
$\mathrm{rel}(r') = r$ and r' is applicable to $\langle q, \mathrm{enc}(h)\rangle$.

Case 1, $h = (u, \lambda)$ for $u \in V(s)$: Let $\sigma = s[u]$ and $j = \mathrm{childno}(u)$. Then, r is applicable to $\langle q, h\rangle$ iff its left-hand side is $\langle q, \sigma, \lambda, j\rangle$. By the definition of R' this is iff there is an $r' \in R'$ with $\mathrm{rel}(r') = r$ and left-hand side $\langle q, \sigma', \lambda, j\rangle$. Since $\mathrm{enc}(h) = h$ and, by P1, $s'[u] = \sigma'$, this is iff r' is applicable to $\langle q, \mathrm{enc}(h)\rangle$.

Case 2, $h = (u, vv_1 \cdots v_l)$ for $u, v, v_1, \ldots, v_l \in V(s)$ and $l \in [0, n-1]$: Let $\sigma = s[u]$, $p \in \{0,1\}$ with $p = 1$ iff $v = u$, $b' \in \{0,1\}^l$ with $b'(\mu) = 1$ iff $v_\mu = u$ for $\mu \in [l]$, and $j = \mathrm{childno}(u)$. We distinguish two subcases.

Case (i), u is not an ancestor of v: Since $p = 0$ (because $u \neq v$), r is applicable to $\langle q, h\rangle$ iff its left-hand side is $\langle q, \sigma, 0b', j\rangle$. By the definition of R' this is iff there is an $r' \in R'$ with $\mathrm{rel}(r') = r$ and left-hand side $\langle q, \sigma, b', j\rangle$. Since $\mathrm{enc}(h) = (\mathrm{enc}_v(u), \mathrm{enc}_v(v_1) \cdots \mathrm{enc}_v(v_l))$, $s'[\mathrm{enc}_v(u)] = s[u]$ by P2, $\mathrm{childno}(\mathrm{enc}_v(u)) = \mathrm{childno}(u)$ by P5, and $\mathrm{enc}_v(v_\mu) = \mathrm{enc}_v(u)$ iff $v_\mu = u$ for $\mu \in [l]$ by P0, this is iff r' is applicable to $\langle q, \mathrm{enc}(h)\rangle$.

Case (ii), u is an ancestor of v: Let $j' = \mathrm{childno}(\mathrm{enc}_v(u))$ and $i = \mathrm{swap}_v(u)$. Now r is applicable to $\langle q, h\rangle$ iff its left-hand side is $\langle q, \sigma, pb', j\rangle$. By the definition of R' this is iff there is an r' in R' with $\mathrm{rel}(r') = r$ and left-hand side $\langle q, (\sigma, j, d_i), b', j'\rangle$ (note that, by the definition of $\mathrm{swap}_v(u)$, $i \in [k]$ if $p = 0$ and $i = k+1$ otherwise).

Since $\mathrm{enc}(h) = (\mathrm{enc}_v(u), \mathrm{enc}_v(v_1) \cdots \mathrm{enc}_v(v_l))$, $s'[\mathrm{enc}_v(u)] = (\sigma, j, d_i)$ by P3, and $\mathrm{enc}_v(v_\mu) = \mathrm{enc}_v(u)$ iff $v_\mu = u$ for $\mu \in [l]$ by P0, this is iff r' is applicable to $\langle q, \mathrm{enc}(h)\rangle$, which concludes the proof of Claim 1.

Next we prove a claim about the result of applying related rules r and r' to related configurations. More precisely, the claim shows that the application of related rules to related configurations yields related sentential forms. Recall, for an input configuration h, the definition (#) of the substitution $[h]_{M,s}$ at the end of Section 3; we will denote it here by $[h]$, and similarly we denote $[\mathrm{enc}(h)]_{M',s'}$ by $[\mathrm{enc}(h)]$.

Claim 2: Let $\langle q, h\rangle \in C_{M,s}$, $r \in R$ applicable to $\langle q, h\rangle$, and $r' \in R'$ applicable to $\langle q, \mathrm{enc}(h)\rangle$, with $r = \mathrm{rel}(r')$. Then $\mathrm{rhs}(r')[\mathrm{enc}(h)] = \mathrm{rhs}(r)[h][\mathrm{enc}]$.

Let $\sigma \in \Sigma^{(k)}$, $k \geq 0$, $b \in \{0,1\}^{\leq n}$, and $j \in [0, J]$ such that $(\sigma, b, j) = \mathrm{test}(h)$. Thus, r is a $\langle q, \sigma, b, j\rangle$-rule.

If $\mathrm{rhs}(r) \in T_{\Delta \cup \langle Q, \mathrm{stay}\rangle}$ then $\mathrm{rhs}(r') = \mathrm{rhs}(r)$ and, since there are only stay instructions, applying the substitution $[h][\mathrm{enc}]$ is equivalent to applying $[\langle q, \mathrm{stay}\rangle \leftarrow \langle q, \mathrm{enc}(h)\rangle \mid q \in Q]$ which, for the same reason, is equivalent to applying $[\mathrm{enc}(h)]$.

If $\mathrm{rhs}(r) = \langle q', \varphi\rangle$ with $\varphi \in I_{\sigma,b,j} - \{\mathrm{stay}\}$ then we distinguish the following three cases. Let $u \in V(s)$.

Case 1, $\varphi = \mathrm{drop}$: If $h = (u, \lambda)$ then $\mathrm{enc}(h) = h$ and $\mathrm{rhs}(r') = \langle q', \mathrm{down}_{k+1}\rangle$. Thus, $\mathrm{rhs}(r')[\mathrm{enc}(h)] = \langle q', \mathrm{down}_{k+1}(h)\rangle = \langle q', (u(k+1), \lambda)\rangle$ which, by the definition of enc, equals $\langle q', \mathrm{enc}(u,u)\rangle = \langle q', \mathrm{drop}(h)\rangle[\mathrm{enc}] = \mathrm{rhs}(r)[h][\mathrm{enc}]$. If $h = (u, vv_1 \cdots v_l)$ for $v, v_1, \ldots, v_l \in V(s)$ and $l \geq 0$, then $\mathrm{rhs}(r') = \mathrm{rhs}(r)$. Thus, $\mathrm{rhs}(r')[\mathrm{enc}(h)] = \langle q', \mathrm{drop}(\mathrm{enc}_v(u), \mathrm{enc}_v(v_1) \cdots \mathrm{enc}_v(v_l))\rangle = \langle q', (\mathrm{enc}_v(u), \mathrm{enc}_v(v_1) \cdots \mathrm{enc}_v(v_l)\mathrm{enc}_v(u))\rangle = \langle q', \mathrm{enc}(u, vv_1 \cdots v_l u)\rangle = \langle q', \mathrm{drop}(h)\rangle[\mathrm{enc}] = \mathrm{rhs}(r)[h][\mathrm{enc}]$.

Case 2, $\varphi = \mathrm{lift}$: If $h = (u, u)$ then $\mathrm{rhs}(r') = \langle q', \mathrm{up}\rangle$ and $\mathrm{enc}(h) = (\mathrm{enc}_u(u), \lambda) = (u(k+1), \lambda)$. Consequently, $\mathrm{rhs}(r')[\mathrm{enc}(h)] = \langle q', \mathrm{up}(u(k+1), \lambda)\rangle = \langle q', (u, \lambda)\rangle = \langle q', \mathrm{enc}(u, \lambda)\rangle = \langle q', \mathrm{lift}(h)\rangle[\mathrm{enc}] = \mathrm{rhs}(r)[h][\mathrm{enc}]$.

If $h = (u, vv_1 \cdots v_l u)$ for $v, v_1, \ldots, v_l \in V(s)$ and $l \geq 0$ then $\mathrm{rhs}(r') = \mathrm{rhs}(r)$. Hence, $\mathrm{rhs}(r')[\mathrm{enc}(h)] = \langle q', \mathrm{lift}(\mathrm{enc}_v(u), \mathrm{enc}_v(v_1) \cdots \mathrm{enc}_v(v_l)\mathrm{enc}_v(u))\rangle = \langle q', (\mathrm{enc}_v(u), \mathrm{enc}_v(v_1) \cdots \mathrm{enc}_v(v_l))\rangle = \langle q', \mathrm{enc}(u, vv_1 \cdots v_l)\rangle = \langle q', \mathrm{lift}(h)\rangle[\mathrm{enc}] = \mathrm{rhs}(r)[h][\mathrm{enc}]$.

Case 3, $\varphi \in \{\mathrm{up}, \mathrm{down}_1, \ldots, \mathrm{down}_k\}$: If $h = (u, \lambda)$ then $\mathrm{rhs}(r') = \mathrm{rhs}(r)$, $\mathrm{enc}(h) = h$, and $\mathrm{enc}(\varphi(h)) = \varphi(h)$. Thus, on $\mathrm{rhs}(r)$, $[\mathrm{enc}(h)] = [h] = [h][\mathrm{enc}]$. If $h = (u, vv_1 \cdots v_l)$ for $v, v_1, \ldots, v_l \in V(s)$ and $l \geq 0$ then we distinguish the following two cases, where p denotes the string $vv_1 \cdots v_l$ and p' denotes $\mathrm{enc}_v(v_1) \cdots \mathrm{enc}_v(v_l)$.

Case (i), u is not an ancestor of v: Then $\mathrm{rhs}(r') = \mathrm{rhs}(r)$, i.e., it suffices to show that $\varphi(\mathrm{enc}(h)) = \mathrm{enc}(\varphi(h))$. Now $\mathrm{enc}(h) = (\mathrm{enc}_v(u), p') = (v(k'+1)\mathrm{swap}_v(w_1) \cdots \mathrm{swap}_v(w_{m-1})u', p')$, where k' is the rank of $s[v]$, $w_1 = u, w_2, \ldots, w_m$ are the nodes on the path from v to the longest common ancestor w_m of u and v, and $u = w_m u'$. Since u is not an ancestor of v, $u' \in \mathbb{N}^+$. Thus, applying φ to $\mathrm{enc}(h)$ amounts to applying it to u', and hence to u. For a node z, define $\varphi(z) = \mathrm{parent}(z)$ if $\varphi = \mathrm{up}$, and $\varphi(z) = zi$ if $\varphi = \mathrm{down}_i$. Then $\varphi(\mathrm{enc}(h)) = (v(k'+1)\mathrm{swap}_v(w_1) \cdots \mathrm{swap}_v(w_{m-1})\varphi(u'), p') = \mathrm{enc}(w_m\varphi(u'), p) = \mathrm{enc}(\varphi(w_m u'), p) = \mathrm{enc}(\varphi(u), p) = \mathrm{enc}(\varphi(h))$.

Case (ii), u is an ancestor of v: If $\varphi = \mathrm{up}$ then $\mathrm{rhs}(r') = \langle q', \mathrm{down}_i\rangle$ where $i = \mathrm{swap}_v(u)$ by P3 and the definition of r'. Thus, we must show that $\mathrm{enc}(\mathrm{up}(h)) = \mathrm{down}_i(\mathrm{enc}(h))$. Now $\mathrm{up}(h) = (\bar{u}, p)$ where $\bar{u} = \mathrm{parent}(u)$. Thus, $\mathrm{enc}_v(\bar{u}) = v(k'+1)\mathrm{swap}_v(w_1) \cdots \mathrm{swap}_v(w_{m-1})$, where $w_1 = v, \ldots, w_m = \bar{u}$ are the nodes on the path from v to $\bar{u}$. This implies that $w_{m-1} = u$ and $\mathrm{swap}_v(w_{m-1}) = i$, i.e., $\mathrm{enc}(\bar{u}, p) = \mathrm{down}_i(v(k'+1)\mathrm{swap}_v(w_1) \cdots \mathrm{swap}_v(w_{m-2}), p') = \mathrm{down}_i(\mathrm{enc}(h))$.

If $\varphi = \mathrm{down}_i$ for $i \in [k]$, then we distinguish whether or not ui is an ancestor of v. If ui is not an ancestor of v, then $\mathrm{rhs}(r') = \mathrm{rhs}(r)$ and we must show that $\mathrm{enc}(\varphi(h)) = \varphi(\mathrm{enc}(h))$. Since $\mathrm{enc}_v(ui) = \mathrm{enc}_v(u)i$ we get $\mathrm{enc}(\mathrm{down}_i(h)) = \mathrm{enc}(ui, p) = (\mathrm{enc}_v(ui), p') = (\mathrm{enc}_v(u)i, p') = \mathrm{down}_i(\mathrm{enc}_v(u), p') = \mathrm{down}_i(\mathrm{enc}(h))$.

If ui is an ancestor of v, then $\mathrm{rhs}(r') = \langle q', \mathrm{up}\rangle$, i.e., we must show that $\mathrm{enc}(\mathrm{down}_i(h)) = \mathrm{up}(\mathrm{enc}(h))$. Now $\mathrm{down}_i(h) = (ui, p)$ and $\mathrm{enc}(h) = (v(k'+1)\mathrm{swap}_v($

$w_1)\cdots\mathrm{swap}_v(w_{m-1}),p')$, where k' is the rank of $s[v]$ and $w_1 = v,\ldots,w_m = u$ are the nodes on the path from v to u; thus, $\mathrm{up}(\mathrm{enc}(h)) = (v(k'{+}1)\mathrm{swap}_v(w_1)\cdots\mathrm{swap}_v(w_{m-2}),p')$. Since $w_{m-1} = ui$, this equals $\mathrm{enc}(ui,p) = \mathrm{down}_i(\mathrm{enc}(h))$.

This concludes the proof of Claim 2.

The next claim shows that the application of related rules to the same node in related sentential forms (i.e., ξ and ξ' with $\xi' = \xi[\mathrm{enc}]$), yields again related sentential forms. Recall the definition of $\Rightarrow_{M,s}$ from Section 3: If $\xi \Rightarrow_{M,s} \tilde{\xi}$ then there is a leaf ρ in ξ such that $\tilde{\xi} = \xi[\rho \leftarrow \zeta[h]]$, where $\xi[\rho] = \langle q,h\rangle \in C_{M,s}$ and ζ is the right-hand side of a rule r of M applicable to $\langle q,h\rangle$; we say that "$\xi \Rightarrow_{M,s} \tilde{\xi}$ by rule r at node ρ".

Claim 3: Let $\xi \in T_{\Delta\cup C_{M,s}}$ and $\eta \in T_{\Delta\cup C_{M',s'}}$ with $\eta = \xi[\mathrm{enc}]$. If $\xi \Rightarrow_{M,s} \tilde{\xi}$ by rule $r \in R$ at node $\rho \in V(\xi)$ and $\eta \Rightarrow_{M',s'} \tilde{\eta}$ by rule r' at node ρ, with $r = \mathrm{rel}(r')$, then $\tilde{\eta} = \tilde{\xi}[\mathrm{enc}]$.

Note that if $\xi[\rho] = \langle q,h\rangle$ then, by the definition of [enc], $\eta[\rho] = \langle q,\mathrm{enc}(h)\rangle$. Now Claim 3 can be proved using Claim 2 as follows:

$$\begin{aligned}
\tilde{\eta} &= \xi[\mathrm{enc}][\rho \leftarrow \mathrm{rhs}(r')[\mathrm{enc}(h)]] & \\
&= \xi[\mathrm{enc}][\rho \leftarrow \mathrm{rhs}(r)[h][\mathrm{enc}]] & \text{(by Claim 2)}\\
&= \xi[\rho \leftarrow \mathrm{rhs}(r)[h]][\mathrm{enc}] & \text{(associativity of substitution)}\\
&= \tilde{\xi}[\mathrm{enc}].
\end{aligned}$$

Last but not least, it is shown in the final claim of this proof that relatedness (viz. the application of [enc]) is preserved in arbitrary computations of M and M'.

Claim 4: Let $l \geq 0$ and $\eta \in T_{\Delta\cup C_{M',s'}}$. Then

$$\langle q_0,h_0\rangle \Rightarrow^l_{M',s'} \eta \quad \text{iff} \quad \exists\xi : \langle q_0,h_0\rangle \Rightarrow^l_{M,s} \xi \text{ and } \xi[\mathrm{enc}] = \eta.$$

The proof of Claim 4 is by induction on the length l of the computations. For $l = 0$ the statement is obvious because $\langle q_0,h_0\rangle[\mathrm{enc}] = \langle q_0,h_0\rangle$. Let us now prove the induction step.

First, the 'if' part: Let $\xi,\tilde{\xi}$ be sentential forms of M on s such that

$$\langle q_0,h_0\rangle \Rightarrow^l_{M,s} \xi \Rightarrow_{M,s} \tilde{\xi},$$

and let $\rho \in V(\xi)$, $\langle q,h\rangle \in C_{M,s}$, and $r \in R$ be the involved node, configuration, and rule, respectively, of the last step of the computation. Let $\tilde{\eta} = \tilde{\xi}[\mathrm{enc}]$. By induction, $\langle q_0,h_0\rangle \Rightarrow^l_{M',s'} \eta$ with $\eta = \xi[\mathrm{enc}]$. It follows from the definition of [enc] that $\eta[\rho] = \langle q,\mathrm{enc}(h)\rangle$. By Claim 1 there is a rule r' applicable to $\langle q,\mathrm{enc}(h)\rangle$ with $\mathrm{rel}(r') = r$. Hence $\eta \Rightarrow_{M',s'} \tilde{\xi}[\mathrm{enc}] = \tilde{\eta}$ by Claim 3.

Second, the 'only if' part: Let $\eta,\tilde{\eta}$ be sentential forms of M' on s' such that

$$\langle q_0,h_0\rangle \Rightarrow^l_{M',s'} \eta \Rightarrow_{M',s'} \tilde{\eta},$$

and let $\rho \in V(\eta)$, $\langle q,h'\rangle \in C_{M',s'}$, and $r' \in R'$ be the involved node, configuration, and rule, respectively, of the last step of the computation. By induction, there exists ξ such that $\langle q_0,h_0\rangle \Rightarrow^l_{M,s} \xi$ and $\xi[\mathrm{enc}] = \eta$. Hence, by the definition of [enc], $h' = \mathrm{enc}(h)$ for some $h \in \mathrm{IC}_{n,s}$ and, using Claim 1, $\mathrm{rel}(r')$ is applicable to $\langle q,h\rangle$ at node ρ of ξ. Let $\tilde{\xi}$ be the result of that application. Then $\tilde{\xi}[\mathrm{enc}] = \tilde{\eta}$ by Claim 3. This ends the proof of Claim 4.

Since $\xi[\mathrm{enc}] = t$ iff $\xi = t$, for $t \in T_\Delta$, it follows immediately from Claim 4 that $\tau_{M'}(s') = \tau_M(s)$. Furthermore, since $s' = \tau_{M_{\mathrm{EncPeb}}}(s)$ we obtain that $\tau_{M_{\mathrm{EncPeb}}} \circ \tau_{M'} = \tau_M$. □

From Lemma 9 we obtain the decomposition result of this section, our first main theorem: every n-ptt can be decomposed into the composition of $n+1$ 0-ptts, and similarly in the deterministic case. In more detail, the first n translations of this composition are in fact (very simple) deterministic transducers: they all realize the total function EncPeb.

Theorem 10. For every $n \geq 1$, n-PTT $\subseteq$ 0-PTT^{n+1} and n-DPTT $\subseteq$ 0-DPTT^{n+1}.

A consequence of Theorem 10 is the equality of the composition closure of all ptts with the composition closure of all 0-ptts, and similarly in the deterministic case.

Corollary 11. PTT* = 0-PTT* and DPTT* = 0-DPTT*.

In terms of databases, Corollary 11 means that the query language of pebble tree transducers, i.e., the composition closure PTT* (DPTT*), is equal to the query language of 0-pebble tree transducers.

We note here that the key result of [MSV00] is that inverse n-ptt translations preserve the regular tree languages, i.e., if $\tau \in n$-PTT and $R \in$ REGT, then $\tau^{-1}(R) \in$ REGT. It follows from Theorem 10 that, in fact, it suffices to show this for 0-ptts.

5 Pebble Tree Transducers and Macro Tree Transducers

In this section we compare the model of pebble tree transducers with that of macro tree transducers, well known from tree language theory [Eng80,CF82,EV85,FV98]. Since, according to Subsection 3.2, 0-pebble tree transducers can be thought of as attribute grammars, the (total deterministic) zero pebble case is closely related to the well-known comparison of attributed tree transducers with macro tree transducers (see, e.g., [Eng81,CF82,EM99,FV99]).

The main result is that an n-pebble tree transducer can be simulated by the composition of $n+1$ macro tree transducers (for short, mtts). Moreover, it is shown that mtts can be simulated by compositions of ptts. Thus, the composition closure of all ptts is equal to the composition closure of all mtts. To be precise, in the nondeterministic case, the mtts must additionally be allowed to use stay instructions ("stay-mtts"). These are the second and third main results of this paper.

Let us now discuss these results in more detail. The macro tree transducer can be obtained from the 0-ptt in the following way: First, consider a 0-ptt M that uses no up or stay instructions, i.e., only down instructions. If we additionally allow M to have general rules (with arbitrary right-hand sides in $T_{\Delta \cup \langle Q, \text{down} \rangle}$), then M is a top-down tree transducer [Rou70,Tha70,AU71,Eng82,GS97] (cf. also the discussion on top-down tree transducers in Subsection 3.1 of [MSV]). Now, by adding parameters (of type output tree) to the states of the top-down tree transducer, we obtain the *macro tree transducer* (for short, mtt). A nice consequence of the fact that mtts have no stay and up instructions, is that they have no infinite computations, i.e., they terminate for every input tree. It was proved in the previous section (Corollary 11) that the composition closure of all ptts is equal to the composition closure of all 0-ptts. Hence, in order to prove the equivalence to the composition closure of all mtts, it suffices to show how to simulate 0-ptts by mtts and vice versa.

In order to formalize the simulation of 0-ptts by mtts, we first define a more general model which is of interest on its own: the n-pebble *macro* tree transducer (for short, n-pmtt). It is obtained from the n-ptt by adding parameters to the states. Then, an mtt is a 0-pmtt that uses only down instructions. In order to prove that a 0-ptt can be simulated by an mtt we first eliminate the up instructions by the use of parameters (Lemma 34), thus obtaining a 0-pmtt without up instructions, but which still uses stay instructions: a "stay-mtt". Using Theorem 10, this shows that n-PTT $\subseteq$ sMTT^{n+1}, where sMTT denotes the class of translations realized by stay-mtts (and similarly for the deterministic classes).

In the deterministic case we prove, in Theorem 31, that stay moves can be eliminated from deterministic stay-mtts, i.e., the translation of a 0-dptt can be realized by a deterministic macro tree transducer, and hence an n-dptt can be realized by the $(n+1)$-fold composition of deterministic macro tree transducers (Theorem 35). As suggested in the Introduction, Theorem 31 is, technically speaking, one of the key results of this paper: it involves removing nonterminating computations (which stay at a node of the input tree) from the stay-mtt; this is done in several intermediate stages in the proof of Theorem 31.

In the nondeterministic case it can be shown that stay-mtts are "close" to mtts, in particular that they have the same output languages (which is of interest for the type checking problem) and that in a composition of stay-mtts, all except the first can be mtts (Theorems 30 and 29, respectively). The reason why a nondeterministic stay-mtt cannot always be simulated by an mtt is that $\tau_M(s)$ may be infinite, i.e., there are stay-mtts M that generate infinitely many output trees for one input tree s. A prototypic example of such a transducer is the nondeterministic 0-ptt M_Σ of Example 6 that realizes the translation mon_Σ: it inserts above each σ-labeled node u of the input tree $s \in T_\Sigma$, arbitrarily many nodes labeled by the (unary) symbol $\bar{\sigma}$. In fact, this translation can be used in order to simulate an arbitrary stay-mtt M by an mtt: first M_Σ translates s into the "(arbitrarily) blown up" version $s' \in \mathrm{mon}_\Sigma(s)$ of s by inserting unary nodes, and then a macro tree transducer M' can be constructed that on s' simulates the stay-mtt M (on s): If M does a stay move, then M' moves down on the unary (barred) nodes. Thus, sMTT $\subseteq$ MON$\circ$MTT (Lemma 27), where MON is the class of all translations mon_Σ.

The structure of this section is as follows. In Subsection 5.1, pebble macro tree transducers are defined and some of their basic properties are proved. Subsection 5.2 deals in particular with properties of deterministic pmtts. Subsection 5.3 defines macro tree transducers and stay-mtts, and investigates their relationship. Subsection 5.4 presents the simulation of ptts by compositions of (stay-) macro tree transducers. Finally, in Subsection 5.5 the simulation of (stay-) macro tree transducers by compositions of ptts is presented, and it is proved that the composition closures of ptts and (stay-) mtts coincide.

5.1 Pebble Macro Tree Transducers

The *n-pebble macro tree transducer* (for short, n-pmtt) is obtained from the n-ptt by allowing each state to have a finite number of *parameters* $y_1, \dots, y_m$ of type output tree (in addition to the, implicit, parameter of type "input configuration"). Moreover, the right-hand side of a rule of an n-pmtt is an arbitrary tree over output symbols, state-instruction pairs $\langle q', \varphi \rangle$ of the same rank as q', and parameters. For instance, $\langle q, \mathrm{up} \rangle(\alpha, \sigma(y_1, \langle q', \mathrm{down}_1 \rangle))$ is a possible right-hand side (for a state of rank ≥ 1), where q and q' are of rank 2 and 0, respectively. Viewing an n-pmtt as a functional program this means that each state (of rank m) is a function with $m+1$ parameters, and in the function body each case of the case distinction consists of an arbitrary expression over output symbols, function calls, and parameters. Recall from Subsection 2.1 that Y_m denotes the set $\{y_1, \dots, y_m\}$.

Definition 12. For $n \geq 0$, an *n-pebble macro tree transducer* is a tuple $M = (\Sigma, \Delta, Q, q_0, R)$, where Σ and Δ are ranked alphabets of *input* and *output symbols*, respectively, Q is a ranked alphabet of *states*, $q_0 \in Q^{(0)}$ is the *initial state*, and R

is a finite set of *rules* of the form

$$\langle q,\sigma,b,j\rangle(y_1,\dots,y_m)\to\zeta,$$

where $q\in Q^{(m)}$, $m\geq 0$, $\sigma\in\Sigma$, $b\in\{0,1\}^{\leq n}$, $j\in[0,J]$ with $J=\max\{\mathrm{rank}_\Sigma(\sigma)\mid\sigma\in\Sigma\}$, and $\zeta\in T_{\Delta\cup\langle Q,I_{\sigma,b,j}\rangle}(Y_m)$. A rule r as above is called $\langle q,\sigma,b,j\rangle$-rule or q-rule, and its right-hand side ζ is denoted by $\mathrm{rhs}(r)$. For a subset Q' of Q, a q-rule with $q\in Q'$ is also called Q'-rule.

If for every q, σ, b, and j there is at most one $\langle q,\sigma,b,j\rangle$-rule in R, then M is *deterministic* (for short, M is an n-dpmtt). If there is at least one such rule then M is *total*. □

Note that an n-ptt with general rules (cf. Lemma 2) is the special case of an n-pmtt in which each state has rank zero, i.e., has no parameters. For an n-pmtt M, the ranked set of all *configurations* of M on s, denoted by $C_{M,s}$, is defined as $\langle Q,\mathrm{IC}_{n,s}\rangle$ (recall, from the beginning of Subsection 2.1, that this means that $\langle q,h\rangle\in\langle Q,\mathrm{IC}_{n,s}\rangle$ has the same rank as q). A rule $\langle q,\sigma,b,j\rangle(y_1,\dots,y_m)\to\zeta$ of M is applicable to a configuration $\langle q,h\rangle$ if $(\sigma,b,j)=\mathrm{test}(h)$. A *sentential form* (of M on s) is a tree over $\Delta\cup C_{M,s}$.

Let ξ be a sentential form and $u\in V(\xi)$. Then u is *outside in* ξ if no proper ancestor of u is labeled by a configuration. The *computation relation* of M on $s\in T_\Sigma$ is defined as follows: For $\xi,\xi'\in T_{\Delta\cup C_{M,s}}$, $\xi\Rightarrow_{M,s}\xi'$ iff there are

(N) a node v outside in ξ labeled by $\langle q,h\rangle\in C^{(m)}_{M,s}$, $m\geq 0$, and
(R) a rule $\langle q,\sigma,b,j\rangle(y_1,\dots,y_m)\to\zeta$ in R applicable to $\langle q,h\rangle$

such that $\xi'=\xi[\![v\leftarrow\zeta[\![h]\!]_{M,s}]\!]$ where

$$[\![h]\!]_{M,s}=[\![\langle q',\varphi\rangle\leftarrow\langle q',\varphi(h)\rangle\mid q'\in Q,\varphi\in I_{\mathrm{test}(h)}]\!]. \qquad (\#)$$

Recall from Subsection 2.2 that $\xi[\![v\leftarrow\eta]\!]$ denotes $\xi[v\leftarrow\eta[y_j\leftarrow t/vj\mid j\in[m]]]$. Recall also that the substitution $[\![h]\!]_{M,s}$ is just a relabeling: every node labeled $\langle q',\varphi\rangle$ is relabeled by $\langle q',\varphi(h)\rangle$.

The *translation* τ_M *realized by* M is defined in the same way as for an n-ptt. The class of all translations realized by n-pmtts is denoted by n-PMTT. If the transducers are deterministic, then the respective class is denoted by n-DPMTT. The unions of these classes over $n\in\mathbb{N}$ are denoted PMTT and DPMTT, respectively. Note that $n\text{-PTT}\subseteq n\text{-PMTT}$, and similarly for the deterministic case.

Example 13. In order to demonstrate that the addition of parameters gives a proper extension to pebble tree transducers, we construct a deterministic 0-pebble macro tree transducer that realizes a translation that has an exponential size-to-height relationship, and therefore cannot be realized by any pebble tree transducer by Lemma 7. Let $M=(\Sigma,\Sigma,\{q_0^{(0)},q^{(1)}\},q_0,R)$ where $\Sigma=\{a^{(1)},e^{(0)}\}$ and let R consist of the following four rules.

$$\begin{array}{lcl}
\langle q_0,a,\lambda,0\rangle & \to & \langle q,\mathrm{down}_1\rangle(\langle q,\mathrm{down}_1\rangle(e))\\
\langle q_0,e,\lambda,0\rangle & \to & a(e)\\
\langle q,a,\lambda,1\rangle(y_1) & \to & \langle q,\mathrm{down}_1\rangle(\langle q,\mathrm{down}_1\rangle(y_1))\\
\langle q,e,\lambda,1\rangle(y_1) & \to & a(e)
\end{array}$$

Now, let us consider how M computes the output tree $\tau_M(s)$, for the input tree $s=a(a(e))$:

$$\begin{aligned}
\langle q_0,h_0\rangle=\langle q_0,(\varepsilon,\lambda)\rangle &\Rightarrow_{M,s}\langle q,(1,\lambda)\rangle(\langle q,(1,\lambda)\rangle(e))\\
&\Rightarrow_{M,s}\langle q,(2,\lambda)\rangle(\langle q,(2,\lambda)\rangle(\langle q,(1,\lambda)\rangle(e)))\\
&\Rightarrow_{M,s}a(\langle q,(2,\lambda)\rangle(\langle q,(1,\lambda)\rangle(e)))\\
&\Rightarrow_{M,s}a(a(\langle q,(1,\lambda)\rangle(e)))\\
&\Rightarrow_{M,s}a(a(\langle q,(2,\lambda)\rangle(\langle q,(2,\lambda)\rangle(e))))\\
&\Rightarrow^2_{M,s}a(a(a(a(e)))).
\end{aligned}$$

It should be clear that $\tau_M = \{(a^m(e), a^{2^m}(e)) \mid m \in \mathbb{N}\}$. Thus, τ_M is not of polynomial size-to-height increase and therefore

$$0\text{-DPMTT} - \text{PTT} \neq \varnothing.$$

□

In the sequel we will also apply $\Rightarrow_{M,s}$ to trees with parameters, i.e, trees in $T_{\Delta\cup C_{M,s}}(Y)$; then, the parameters are just viewed as output symbols of rank zero.

Note that, by the requirement in (N) that v is outside, the order in which configurations in a tree $\xi \in T_{\Delta\cup C_{M,s}}$ are replaced is top-down; in other words, ξ is evaluated in a "call-by-name" (or "lazy") fashion: the value of an actual parameter is not evaluated until the "function-call" has been evaluated and the parameter is needed. In terms of macro tree grammars this order of replacement is called "outside-in", or "OI" for short (cf., e.g., [Fis68,ES77]). Macro tree grammars (also called context-free tree grammars) can be obtained from a pmtt by removing the tree-walk facility (then the configurations become the states, viz. the nonterminals). Just as the computations of an n-ptt can be simulated by a regular tree grammar, as shown in the beginning of Section 3, it is possible to obtain, for a fixed input tree s, a computation by $\Rightarrow_{M,s}$ (for a pmtt M) as the derivation of a macro tree grammar $G_{M,s}$: The (ranked) nonterminals of $G_{M,s}$ are the configurations $\langle q,h\rangle$ in $C_{M,s}$ and if $\langle q,h\rangle(y_1,\dots,y_m) \Rightarrow_{M,s} \alpha$ then $G_{M,s}$ has the production $\langle q,h\rangle(y_1,\dots,y_m) \to \alpha$. For macro tree grammars the OI requirement is superfluous, i.e., the same tree language is generated with unrestricted order of replacement (see Theorem 4.1.2 of [Fis68]; see also Section 3.2 of [EV85]). This implies that also for pmtts the outside-in requirement in (N) can be dropped, without changing τ_M. We keep the restriction because it is technically more convenient.

As explained in Subsection 3.3, n-ptts are the same as RT(n-Pebble) transducers. From the previous paragraph it should be clear that we just have to replace the regular tree grammar (RT) by the context-free tree grammar (CFT) in order to obtain a formalism that is equivalent to the n-pmtt: the CFT(n-Pebble) transducer. In particular, the 0-pmtt is the same as the CFT(Tree-walk) transducer, which is related to the so-called macro attributed tree transducer of [KV94,FV98] in the same way as the 0-ptt is related to the attribute grammar (see Subsection 3.2).

Convention 14. In order to make the rules of n-pmtts more readable, we fix the convention (both for the n-ptts of Definition 1 and the n-pmtts of Definition 12) that stay instructions may be omitted, i.e., instead of $\langle q, \text{stay}\rangle$ for a state q, we may simply write q.

Since pmtts have stay moves, their rules $\langle q,\sigma,b,j\rangle(y_1,\dots,y_m) \to \zeta$ can be restricted in such a way that each ζ has one of the forms

$$\zeta = \begin{cases} \langle q',\varphi\rangle(\langle q_1,\text{stay}\rangle(y_1,\dots,y_m),\dots,\langle q_k,\text{stay}\rangle(y_1,\dots,y_m)) & \text{(navigation)} \\ \delta(\langle q_1,\text{stay}\rangle(y_1,\dots,y_m),\dots,\langle q_k,\text{stay}\rangle(y_1,\dots,y_m)) & \text{(output)} \\ y_\mu & \text{(parameter selection)} \end{cases}$$

A pmtt is in *normal form* if the right-hand side of each of its rules has one of the above three forms. Using Convention 14, this means that the right-hand side of an n-pmtt rule is either a parameter, or of one of the following two forms:

– $\langle q',\varphi\rangle(q_1(y_1,\dots,y_m),\dots,q_k(y_1,\dots,y_m))$ or
– $\delta(q_1(y_1,\dots,y_m),\dots,q_k(y_1,\dots,y_m))$.

It will be proved in the next theorem (Theorem 16) that every pmtt can be put into normal form. This shows that the pmtt can also be viewed as a very simple extension of the ptt in its original form (i.e., without general rules).

To prove Theorem 16 we will use the following basic lemma (also to be used in the proof of Theorem 31). It shows that a stay instruction in the right-hand side of a rule can be expanded by "applying" an appropriate rule. This is similar to the well-known technique of applying a production of a context-free grammar to the right-hand side of another production. Note that the occurrence of the stay instruction need not be outside.

Lemma 15. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be an n-pmtt M, $n \geq 0$, let

$$\begin{aligned} r_1 &= \langle q_1, \sigma, b, j\rangle(y_1, \ldots, y_{m_1}) \to \zeta_1 \text{ and} \\ r_2 &= \langle q_2, \sigma, b, j\rangle(y_1, \ldots, y_{m_2}) \to \zeta_2 \end{aligned}$$

be rules of M, and let $u \in V(\zeta_1)$ have label $\zeta_1[u] = \langle q_2, \text{stay}\rangle$. Assume, moreover, that r_2 is the unique rule in R with left-hand side $\langle q_2, \sigma, b, j\rangle(y_1, \ldots, y_{m_2})$. Let $M' = (\Sigma, \Delta, Q, q_0, R')$ be the n-pmtt with $R' = \{r' \mid r \in R\}$ where $r' = r$ for $r \neq r_1$, and

$$r_1' = \langle q_1, \sigma, b, j\rangle(y_1, \ldots, y_{m_1}) \to \zeta_1[\![u \leftarrow \zeta_2]\!]$$

(i.e., M' is obtained from M by changing rule r_1 into r_1').
Then $\tau_{M'} = \tau_M$.

Proof. We may assume that $q_1 \neq q_2$, that $u \in V(\zeta_1)$ in r_1 is the unique occurrence of the state q_2 in the right-hand sides of the rules of M, that r_2 is the unique q_2-rule in R, and that q_2 is not the initial state. In fact, if this is not the case, then change $\zeta_1[u]$ into $\langle \bar{q}_2, \text{stay}\rangle$, and add the rule $\langle \bar{q}_2, \sigma, b, j\rangle(y_1, \ldots, y_{m_2}) \to \zeta_2$ to R, where $\bar{q}_2$ is a new state.

Note that, consequently, if $\langle q_0, h_0\rangle \Rightarrow^*_{M,s} \xi$ and $\langle q_2, h\rangle$ occurs in ξ, then $\text{test}(h) = (\sigma, b, j)$, as can easily be shown by induction on the length of the derivation. This means that r_2 is applicable to $\langle q_2, h\rangle$.

We also note that $r_1 \neq r_2$ and hence q_2 does not occur in ζ_2. This implies that for every $\xi \in T_{\Delta \cup C_{M,s}}$ there exists $\tilde{\xi} \in T_{\Delta \cup C_{M,s}}$ such that $\xi \Rightarrow^*_{M,s} \tilde{\xi}$ by q_2-rules only (i.e., by applications of r_2) and $\tilde{\xi}$ has no outside occurrences of configurations $\langle q_2, h\rangle$, $h \in \text{IC}_{n,s}$. To see this, let us say that an occurrence of $\langle q_2, h\rangle$ in a sentential form is *almost outside* if none of its ancestors is labeled $\langle q, h'\rangle$ with $q \neq q_2$. It should now be clear that after applying r_2 to all outside occurrences of configurations $\langle q_2, h\rangle$ in the sentential form ξ, the maximal number of almost outside occurrences of configurations $\langle q_2, h\rangle$ on a path of the sentential form has decreased. Thus, $\tilde{\xi}$ is obtained after repeating this process at most $\text{height}(\xi)$ times.

Let $s \in T_\Sigma$. In order to prove the correctness of M', i.e., that $\tau_{M'}(s) = \tau_M(s)$, first a claim is proved. Part (1) of the claim shows how to simulate M by M': if a rule r other than r_2 is applied by M then M' can apply the corresponding rule r', and if rule r_2 is applied then M' need not apply a rule, because the involved trees are equal under the substitution Ψ (defined in the Claim); intuitively, Ψ carries out all M's computation steps for configurations $\langle q_2, h\rangle$, $h \in \text{IC}_{n,s}$. The second part of the Claim shows how to simulate M' by M; it uses the fact mentioned above: starting with any sentential form ξ of M, there is a computation by $\Rightarrow_{M,s}$ (using rule r_2 only) such that the resulting tree $\tilde{\xi}$ has no outside occurrences of configurations $\langle q_2, h\rangle$.

Claim: Let the substitution Ψ be defined as

$$\Psi = [\![\langle q_2, h\rangle \leftarrow \zeta_2[\![h]\!] \mid h \in \text{IC}_{n,s}]\!]$$

where $[\![h]\!] = [\![h]\!]_{M,s} = [\![h]\!]_{M',s}$ is defined as in (#) above (below Definition 12).

(1) Let $\xi, \xi' \in T_{\Delta \cup C_{M,s}}$ such that $\xi \Rightarrow_{M,s} \xi'$ by the rule r at node v of ξ. If $r = r_2$ then $\xi\Psi = \xi'\Psi$, and if $r \neq r_2$ then $\xi\Psi \Rightarrow_{M',s} \xi'\Psi$ by the rule r' at node v of $\xi\Psi$.

(2) For $\eta, \eta' \in T_{\Delta \cup C_{M',s}}$ and $\xi \in T_{\Delta \cup C_{M,s}}$, if $\eta \Rightarrow_{M',s} \eta'$ and $\xi\Psi = \eta$ then there exists ξ' such that $\xi \Rightarrow^*_{M,s} \xi'$ and $\xi'\Psi = \eta'$.

Proof of part (1): $\xi \Rightarrow_{M,s} \xi'$ by r at v. By the definition of $\Rightarrow_{M,s}$ this means that v is outside in ξ and has label $\langle q,h\rangle \in C_{M,s}$, such that $\xi' = \xi[\![v \leftarrow \zeta[\![h]\!]]\!]$ where ζ is the right-hand side of the rule r, which is applicable to $\langle q,h\rangle$.

If $r = r_2$, then $q = q_2$ and $\zeta = \zeta_2$, and so $\xi'\Psi = \xi[\![v \leftarrow \zeta_2[\![h]\!]]\!]\Psi = \xi\Psi$, because v has label $\langle q_2,h\rangle$ and q_2 does not occur in ζ_2.

If $r \neq r_2$ then $q \neq q_2$ because r_2 is the only q_2-rule. Note that since v is outside in ξ, it is also outside in $\xi\Psi$ and $\xi\Psi/v = (\xi/v)\Psi$. This implies that $(\xi\Psi)[v] = \xi[v] = \langle q,h\rangle$. Thus, the rule r' of M', which has the same left-hand side as r, is applicable to $\xi\Psi$ at v. Let η' be the result of that application. Hence, $\xi\Psi \Rightarrow_{M',s} \eta'$. Note also that $\xi'\Psi = \xi[\![v \leftarrow \zeta[\![h]\!]]\!]\Psi = \xi\Psi[\![v \leftarrow \zeta[\![h]\!]\Psi]\!]$ because v is outside and does not have label $\langle q_2,h\rangle$. We now distinguish two cases.

If $r \neq r_1$, then $r' = r$ and $\xi'\Psi = \xi\Psi[\![v \leftarrow \zeta[\![h]\!]]\!]$ because q_2 does not occur in ζ. Since this equals η', $\xi\Psi \Rightarrow_{M',s} \xi'\Psi$.

If $r = r_1$, then $q = q_1$ and $\zeta = \zeta_1$, and $r' = r'_1$. In this case we obtain that $\xi'\Psi = \xi\Psi[\![v \leftarrow \zeta_1[\![h]\!]\Psi]\!] = \xi\Psi[\![v \leftarrow (\zeta_1[\![u \leftarrow \zeta_2]\!])[\![h]\!]]\!] = \eta'$.

Proof of part (2): If $\eta \Rightarrow_{M',s} \eta'$ then there is a node v outside in η such that $\eta[v] = \langle q,h\rangle \in C_{M,s}$ and there is a rule r' in R' with right-hand side ζ that is applicable to $\langle q,h\rangle$ such that $\eta' = \eta[\![v \leftarrow \zeta[\![h]\!]]\!]$. If $\xi \in T_{\Delta \cup C_{M,s}}$ such that $\xi\Psi = \eta$, then, by the remark above this Claim, there exists $\tilde{\xi}$ such that $\xi \Rightarrow^*_{M,s} \tilde{\xi}$ only by q_2-rules, and $\tilde{\xi}$ has no outside occurrences of configurations $\langle q_2,h'\rangle$, $h' \in \mathrm{IC}_{n,s}$. By part (1) of this Claim, $\tilde{\xi}\Psi = \xi\Psi = \eta$. Consider the outside occurrence v of $\langle q,h\rangle$ in η. Since the application of Ψ to $\tilde{\xi}$ does not replace any outside occurrences of configurations $\langle q_2,h'\rangle$ (because there are none), $\tilde{\xi}\Psi[v] = \tilde{\xi}[v]$. Let ξ' be the result of applying the rule r of M to $\tilde{\xi}$ at v. Then $\eta = \tilde{\xi}\Psi \Rightarrow_{M',s} \xi'\Psi$ by applying r' at v, according to part (1) of this Claim. Hence $\xi'\Psi = \eta'$, which concludes the proof of the Claim.

We are now ready to prove that $\tau_{M'} = \tau_M$. First, $\tau_M(s) \subseteq \tau_{M'}(s)$: If $\langle q_0,h_0\rangle \Rightarrow^*_{M,s} t \in T_\Delta$ then, by part (1) of the Claim above, $\langle q_0,h_0\rangle = \langle q_0,h_0\rangle\Psi \Rightarrow^*_{M',s} t\Psi = t$ (where Ψ is as in the Claim). Second, $\tau_{M'}(s) \subseteq \tau_M(s)$: Assume that $\langle q_0,h_0\rangle \Rightarrow^*_{M',s} t \in T_\Delta$. Then, by part (2) of the Claim, $\langle q_0,h_0\rangle \Rightarrow^*_{M,s} \xi$ for some $\xi \in T_{\Delta \cup C_{M,s}}$ with $\xi\Psi = t$. As mentioned before the Claim, there exists a $\tilde{\xi}$ such that $\xi \Rightarrow^*_{M,s} \tilde{\xi}$ by q_2-rules, $\tilde{\xi}$ has no outside occurrences of configurations $\langle q_2,h\rangle$, and $\tilde{\xi}\Psi = \xi\Psi$ by part (1) of the Claim. Since $\tilde{\xi}\Psi \in T_\Delta$, $\tilde{\xi}$ has no outside occurrences of configurations $\langle q,h\rangle$ with $q \neq q_2$ (by the definition of Ψ). Hence, $\tilde{\xi} \in T_\Delta$ and $\langle q_0,h_0\rangle \Rightarrow^*_{M,s} \tilde{\xi} = \tilde{\xi}\Psi = t$. □

In the next theorem we prove that for every pmtt M there is an equivalent pmtt M' in normal form. In particular, if all states of M are of rank 0 (i.e., M is an n-ptt with general rules), then M' is a ptt (without general rules). Thus, this result encompasses Lemma 2.

Theorem 16. For every n-pmtt M there is an equivalent n-pmtt M' in normal form. If M is deterministic, then so is M'. If all states of M are of rank 0, then M' is an n-ptt.

Proof. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be an n-pmtt. Intuitively, M' uses stay moves to generate the right-hand side ζ of a q-rule of M node by node (in states (ζ, w, m) for node w of ζ, where m is the rank of q). Note that if M' simulates a computation of M, then parts of the right-hand sides of the rules of M might never be generated by M', because of the outside-in order of applying rules. This is, however, no problem, due to Lemma 15.

Define $M' = (\Sigma, \Delta, Q \cup Q_{\mathrm{r}} \cup Q_{\mathrm{p}}, q_0, R')$ as follows. Consider a rule

$$\rho = \langle q, \sigma, b, j\rangle(y_1, \ldots, y_m) \to \zeta \text{ in } R.$$

For every $\mu \in [m]$, let p^m_μ be a state in Q_{p} of rank m and let the rule

$$\langle p^m_\mu, \sigma, b, j\rangle(y_1, \ldots, y_m) \to y_\mu$$

be in R'. Let (ζ, ε, m) be a state in Q_{r} of rank m and let the rule

$$\rho' = \langle q, \sigma, b, j\rangle(y_1, \ldots, y_m) \to \langle(\zeta, \varepsilon, m), \mathrm{stay}\rangle(p^m_1(y_1, \ldots, y_m), \ldots, p^m_m(y_1, \ldots, y_m))$$

be in R'. For every $w \in V(\zeta)$ let (ζ, w, m) be a state in Q_{r} of rank m and let the rule

$$\begin{aligned}\langle(\zeta, w, m), \sigma, b, j\rangle&(y_1, \ldots, y_m) \to \\ &\zeta[w]((\zeta, w1, m)(y_1, \ldots, y_m), \ldots, (\zeta, wk, m)(y_1, \ldots, y_m))\end{aligned}$$

be in R', where k is the rank of $\zeta[w]$. Obviously, M' is in normal form (note that we have used Convention 14).

The correctness of M', i.e., the equality $\tau_{M'} = \tau_M$, is based on Lemma 15. In fact, it should be clear that if Lemma 15 is applied iteratively to a rule $r_1 = \rho'$ for all appropriate $(Q_{\mathrm{r}} \cup Q_{\mathrm{p}})$-rules r_2, the original rule ρ is reobtained. More precisely, by first applying m Q_{p}-rules the rule ρ' is transformed into the rule

$$\langle q, \sigma, b, j\rangle(y_1, \ldots, y_m) \to \langle(\zeta, \varepsilon, m), \mathrm{stay}\rangle(y_1, \ldots, y_m),$$

and then size(ζ) applications of Q_{r}-rules transform this rule into ρ (generating ζ in a way similar to a regular tree grammar).

Thus, by Lemma 15, M' is equivalent with the n-pmtt $M'' = (\Sigma, \Delta, Q \cup Q_{\mathrm{r}} \cup Q_{\mathrm{p}}, q_0, R'')$ where R'' is the union of R and all $(Q_{\mathrm{r}} \cup Q_{\mathrm{p}})$-rules of M'. Since, obviously, the states in $Q_{\mathrm{r}} \cup Q_{\mathrm{p}}$ do not occur in the sentential forms of M'' that are generated from $\langle q_0, h_0\rangle$, M'' is equivalent to M. □

In some proofs it will be convenient to deal with total transducers. Therefore, we show in the next lemma that every transducer can be made total, without changing the translation; this is done by simply adding, for each missing q-rule, a rule with $\langle q, \mathrm{stay}\rangle$ as (root of the) right-hand side.

Lemma 17. For every n-pmtt M, $n \geq 0$, there is an equivalent total n-pmtt M'. If M is deterministic, then so is M'.

Proof. Let $M = (\Sigma, \Delta, Q, q_0, R)$ and let $J = \max\{\mathrm{rank}_\Sigma(\sigma) \mid \sigma \in \Sigma\}$. Define $M' = (\Sigma, \Delta, Q, q_0, R')$, where $R' = R \cup C$ and for every $\sigma \in \Sigma$, $q \in Q^{(m)}$, $m \geq 0$, $b \in \{0,1\}^{\leq n}$, and $j \in [0, J]$ such that there is no $\langle q, \sigma, b, j\rangle$-rule in R, let the rule

$$\langle q, \sigma, b, j\rangle(y_1, \ldots, y_m) \to \langle q, \mathrm{stay}\rangle(y_1, \ldots, y_m)$$

be in C. Clearly, M' is equivalent to M: $\tau_M \subseteq \tau_{M'}$ because $R \subseteq R'$. To see that $\tau_{M'} \subseteq \tau_M$, let $s \in T_\Sigma$ and let $\xi, \xi' \in T_{\Delta \cup C_{M',s}} = T_{\Delta \cup C_{M,s}}$. If $\xi \Rightarrow_{M',s} \xi'$ by a rule in R then also $\xi \Rightarrow_{M,s} \xi'$ by the same rule, and if $\xi \Rightarrow_{M',s} \xi'$ by a rule in C, then $\xi' = \xi$ and thus, $\xi \Rightarrow^*_{M,s} \xi'$. Hence, $\langle q_0, h_0\rangle \Rightarrow^*_{M',s} t \in T_\Delta$ implies that $\langle q_0, h_0\rangle \Rightarrow^*_{M,s} t$ and thus $\tau_{M'} \subseteq \tau_M$. □

5.2 Deterministic Pebble Macro Tree Transducers

In this subsection some basic properties of deterministic pmtts are proved. First, a general lemma about binary relations that are "one-step confluent" is proved. Then it is shown that the computation relation of a dpmtt M is one-step confluent. Together this implies that M either halts or computes forever on a given input tree, and that τ_M is a function. Finally it is proved that a computation of M is infinite if it has a "cycle".

Consider a deterministic pmtt M and an input tree s. It should be intuitively clear that for a sentential form ξ of M on s, either all complete computations by $\Rightarrow_{M,s}$ starting with ξ are infinite, or they are all finite, of the same length, and with the same result (recall, from the Preliminaries, the definition of a complete computation). This is proved in the following two lemmas, based on the fact that $\Rightarrow_{M,s}$ is one-step confluent. A binary relation $\Rightarrow$ is *one-step confluent* if $\xi \Rightarrow \xi_1$ and $\xi \Rightarrow \xi_2$ for $\xi_1 \neq \xi_2$ implies that there is a ξ' with $\xi_1 \Rightarrow \xi'$ and $\xi_2 \Rightarrow \xi'$. This is a particular confluence property which implies, e.g., that $\Rightarrow$ is subcommutative [Klo92] (called 'strongly confluent' in [DJ90]). Though not explicitly mentioned, the result that one-step confluence implies the statement of the following lemma, seems to be folklore within the area of term rewriting; nevertheless, we present a formal proof.

Lemma 18. Let A be a set, $\Rightarrow \subseteq A \times A$ a binary relation that is one-step confluent, and let $\xi \in A$. Either the complete computations by $\Rightarrow$ starting with ξ are all infinite, or they are all finite, of the same length, and with the same result.

Proof. Consider two complete computations, both starting with $\xi \in A$. If one of the computations is finite, then by Claim 1 the other computation is also finite, and has the same length and the same result.

Claim 1: If $\xi \Rightarrow^i \xi_1$ and $\xi \Rightarrow^j \xi_2$ for $0 \leq i \leq j$, $\xi_1, \xi_2 \in A$, and $\xi_1 \not\Rightarrow$ (i.e., there is no $\tilde{\xi} \in A$ such that $\xi_1 \Rightarrow \tilde{\xi}$), then $j = i$ and $\xi_2 = \xi_1$.

We prove Claim 1 by induction on i. For $i = 0$, $\xi \not\Rightarrow$ and thus $j = i$ and $\xi_2 = \xi_1 = \xi$. For $i+1 \geq 1$, $\xi \Rightarrow^{i+1} \xi_1$ means that there is a ξ' such that $\xi \Rightarrow \xi' \Rightarrow^i \xi_1$. Since $j \geq i+1$, there is a ξ'' such that $\xi \Rightarrow \xi'' \Rightarrow^{j-1} \xi_2$. If $\xi'' = \xi'$ then, by induction, $j - 1 = i$, i.e., $j = i + 1$, and $\xi_2 = \xi_1$. Now let $\xi'' \neq \xi'$. By one-step confluence there is a $\bar{\xi}$ such that $\xi' \Rightarrow \bar{\xi}$ (which implies $i \geq 1$) and $\xi'' \Rightarrow \bar{\xi}$. By Claim 2, $\bar{\xi} \Rightarrow^{i-1} \xi_1$ and thus $\xi'' \Rightarrow^i \xi_1$. Then, by induction (applied to ξ''), $j - 1 = i$, i.e., $j = i + 1$, and $\xi_2 = \xi_1$, which concludes the proof of Claim 1.

Claim 2: Let $k \geq 1$ and $\xi, \xi', \eta \in A$. If $\xi \Rightarrow^k \xi' \not\Rightarrow$ and $\xi \Rightarrow \eta$ then $\eta \Rightarrow^{k-1} \xi'$.

The claim is proved by induction on k. For $k = 1$ it follows from the one-step confluence of $\Rightarrow$ that $\eta = \xi'$ and thus $\eta \Rightarrow^0 \xi'$. For $k + 1$, there is a ξ_1 such that $\xi \Rightarrow \xi_1 \Rightarrow^k \xi'$. If $\eta = \xi_1$ then the claim holds. Otherwise, by one-step confluence, there must be an η_1 such that $\xi_1 \Rightarrow \eta_1$ and $\eta \Rightarrow \eta_1$. By induction $\eta_1 \Rightarrow^{k-1} \xi_1$ and thus $\eta \Rightarrow^k \xi'$. □

The following easy lemma shows that, for a dpmtt M and an input tree s, the computation relation $\Rightarrow_{M,s}$ is one-step confluent.

Lemma 19. For every dpmtt M and input tree s, $\Rightarrow_{M,s}$ is one-step confluent.

Proof. We have to show that for $\xi, \xi_1, \xi_2 \in T_{\Delta \cup C_{M,s}}$ with $\xi_1 \neq \xi_2$:

$$\text{if } \xi \Rightarrow_{M,s} \xi_1 \text{ and } \xi \Rightarrow_{M,s} \xi_2, \text{ then } \exists \xi' \text{ with } \xi_1 \Rightarrow_{M,s} \xi' \text{ and } \xi_2 \Rightarrow_{M,s} \xi'.$$

If $\xi \Rightarrow_{M,s} \xi_l$ for $l \in [2]$ then there are $v_1, v_2 \in V(\xi)$ and $\zeta_1, \zeta_2 \in T_{\Delta \cup C_{M,s}}$ such that $\xi_l = \xi[v_l \leftarrow \zeta_l]$ for $l \in [2]$. Since M is deterministic there is at most one rule applicable to $\xi[v_l]$. Thus, $v_1 = v_2$ would imply the contradiction $\xi_1 = \xi_2$. Hence,

$v_1 \neq v_2$. Moreover, by the "outside" requirement in (N), v_2 is not an ancestor of v_1, and v_1 is not an ancestor of v_2. Hence $\xi_1/v_2 = \xi/v_2$ and $\xi_2/v_1 = \xi/v_1$ and thus, for $l \in [2]$, $\xi_l \Rightarrow_{M,s} \xi'$, where $\xi' = \xi[v_l \leftarrow \zeta_l \mid l \in [2]]$. $\square$

An immediate consequence of Lemmas 18 and 19 is that τ_M is a (partial) function, because if $(s,t),(s,t') \in \tau_M$, then $\langle q_0, h_0\rangle \Rightarrow^*_{M,s} t$ is a finite complete computation and therefore, by Lemma 18, $t' = t$.

Lemma 20. For every dpmtt M, τ_M is a function.

In fact, Lemmas 19 and 20 were already proved for a more general formalism (see Subsection 3.3): In the proof of Lemma 3.14 of [EV86] it is shown that the derivation relation of a deterministic CFT(S) transducer (where S is an arbitrary storage type) is one-step confluent. Thus, Lemma 19 is the special case that $S = n$-Pebble. Similarly, Lemma 20 is a special case of Theorem 3.15 of [EV86].

Since the number of configurations of a dpmtt M is finite, every infinite computation by M must have repetitions of a configuration. In fact the repetitions will be in such a way that a configuration c will "cycle", i.e., it will compute a tree that contains c itself at an outside occurrence (c is "circular"). The next easy lemma states that circular configurations lead to infinite computations.

Consider a deterministic n-pmtt M and an input tree s of M. A configuration $c \in C^{(m)}_{M,s}$, $m \geq 0$, is *circular* if there is a $t \in T_{\Delta \cup C_{M,s}}(Y_m)$ such that

– $c(y_1,\ldots,y_m) \Rightarrow^+_{M,s} t$ and
– c occurs outside in t.

We now show how to apply a computation starting with some configuration, to an outside occurrence of that configuration in a sentential form. Then, the iterative application of such computations, applied to a node generated by the previous computation, is formalized ("pumping").

Application of a computation: Consider a computation $c(y_1,\ldots,y_m) \Rightarrow^+_{M,s} t$ (where t not necessarily contains c) and consider a tree $\xi \in T_{\Delta \cup C_{M,s}}(Y_m)$ that has an outside occurrence v of c. It follows from the definition of $\Rightarrow_{M,s}$ and by induction, that $\xi \Rightarrow^+_{M,s} \xi[\![v \leftarrow t]\!]$. (In fact, if u is outside in t', then vu is outside in $\xi[\![v \leftarrow t']\!]$ and $\xi[\![v \leftarrow t']\!][\![vu \leftarrow \zeta[\![h]\!]_{M,s}]\!] = \xi[\![v \leftarrow t'[\![u \leftarrow \zeta[\![h]\!]_{M,s}]\!]]\!]$.)

Iteration of applications: If a sentential form ξ_0 (of M on s) has an outside occurrence v_0 of $c_1 \in C^{(m_1)}_{M,s}$, $m_1 \geq 0$, and for every $i \geq 1$ there are t_i and $c_{i+1} \in C^{(m_{i+1})}_{M,s}$, $m_{i+1} \geq 0$, such that $c_i(y_1,\ldots,y_{m_i}) \Rightarrow^+_{M,s} t_i$ and t_i has an outside occurrence v_i of c_{i+1}, then by composing the corresponding computations of the form $\xi \Rightarrow^+_{M,s} [\![v \leftarrow t]\!]$, we obtain the infinite computation

$$\xi_0 \Rightarrow^+_{M,s} \underbrace{\xi_0[\![v_0 \leftarrow t_1]\!]}_{\xi_1} \Rightarrow^+_{M,s} \underbrace{\xi_1[\![v_0 v_1 \leftarrow t_2]\!]}_{\xi_2} \Rightarrow^+_{M,s} \cdots$$
$$\Rightarrow^+_{M,s} \underbrace{\xi_i[\![v_0 v_1 \cdots v_i \leftarrow t_{i+1}]\!]}_{\xi_{i+1}} \Rightarrow^+_{M,s} \cdots . \qquad (\$)$$

Lemma 21. Let M be a dpmtt, s an input tree of M, and ξ a sentential form of M on s. If there exists a ξ' such that $\xi \Rightarrow^*_{M,s} \xi'$ and ξ' contains an outside occurrence of a circular configuration c, then every complete computation by $\Rightarrow_{M,s}$ starting with ξ is infinite.

Proof. Let v be the outside occurrence of c in ξ'. Since c is circular, there exists a t such that $c(y_1,\ldots,y_m) \Rightarrow^+_{M,s} t$ and t has an outside occurrence v of c. Let $\xi_0 = \xi'$, $v_0 = v$, and, for $i \geq 1$, let $t_i = t$, $c_i = c$, and $v_i = v$. Then there is an infinite computation of the form ($), displayed above. Thus there is an infinite computation starting with ξ' and hence one starting with ξ. This implies, by Lemmas 18 and 19, that *every* computation by $\Rightarrow_{M,s}$ starting with ξ is infinite. $\square$

It can be shown that, in fact, the implication in this lemma is an equivalence, i.e., if a complete computation of M starting with ξ is infinite, then ξ leads to a circular configuration. Thus, infinite computations are due to "cycles".

5.3 Macro Tree Transducers (with and without stay moves)

An obvious way to make sure that a 0-pebble macro tree transducer M has no infinite computations, is to disallow up and stay instructions, or, in other words, to only allow down instructions. The transducer model obtained from the 0-pmtt in this way, is the *macro tree transducer* of [Eng80,Eng81,CF82,EV85], defined next.

Definition 22. Let M be a 0-pmtt such that the rules of M contain no up instructions. Then M is a *stay-macro tree transducer* (for short, stay-mtt). If, moreover, the rules of M contain no stay instructions (i.e., there are only down instructions) then M is a *macro tree transducer* (for short mtt, and dmtt if M is deterministic). If all states of an mtt are of rank zero, then it is a *top-down tree transducer.*

As an example of a (deterministic) macro tree transducer, reconsider the 0-dpmtt M of Example 13: it has no up and no stay moves, i.e., it is a dmtt.

The class of all translations realized by stay-mtts is denoted sMTT, and DsMTT for deterministic stay-mtts. The class of all translations realized by mtts is denoted by MTT, and DMTT for deterministic mtts. The class of all translations realized by total deterministic mtts is denoted by $\mathrm{D_tMTT}$. Note that translations realized by total deterministic mtts are total functions. Note also that the analogue of Lemma 17 does not hold for mtts. In fact, $\mathrm{D_tMTT}$ is the class of all total functions in DMTT. We denote by T and DT ($\mathrm{D_tT}$) the classes of translations realized by top-down tree transducers and (total) deterministic top-down tree transducers, respectively.

It follows from the definition that top-down tree transducers are 0-ptts (with general rules) that only use down instructions. Thus, by Lemma 2, we obtain the obvious fact that top-down tree transducers can be simulated by 0-ptts, as observed in [MSV00] and stated in the next lemma.

Lemma 23. $\mathrm{T} \subseteq \text{0-PTT}$ and $\mathrm{DT} \subseteq \text{0-DPTT}$.

Usually (see, e.g., [EV85,FV98]) the rules of an mtt are defined as rewrite rules in which variables of the form x_i represent the down_i instructions. Also, the child number j is not present in the left-hand sides of mtt rules; clearly this information can be incorporated into the states of an mtt, i.e., in order to transform an mtt M defined in the pmtt formalism as above into one defined in the conventional way, new states (q, j) would be introduced, for every state q of M and possible child number j, and the initial state would be $(q_0, 0)$. From this it also follows, as observed in Subsection 3.3, that the mtt is in fact the CFT(Tree) transducer, and that the top-down tree transducer is the RT(Tree) transducer.

Since, in the definition of the computation relation of an n-pmtt, we have fixed in (N) the order in which rules are applied to be outside-in (OI), this also fixes the order for an mtt to be OI (or, equivalently, unrestricted; see Corollary 3.13 of [EV85] and cf. the discussion after Example 13). Macro tree transducers with the inside-out (IO) order of rule application have also been studied in the literature. In the total deterministic case there is no difference between the OI and IO translations. We also note that $\mathrm{MTT}^* = \mathrm{MTT}_{\mathrm{IO}}^*$, where $\mathrm{MTT}_{\mathrm{IO}}$ denotes the class of all IO translations realized by macro tree transducers (cf. Theorem 7.3 of [EV85]), and similarly in the deterministic case.

We now cite two well-known facts about macro tree transducers.

Fact 24. Inverses of (compositions of) macro tree transducers preserve the regular tree languages, i.e., if $\tau \in \mathrm{MTT}^*$ and $R \in \mathrm{REGT}$, then $\tau^{-1}(R) \in \mathrm{REGT}$.

Fact 25. For an output language K of a composition of macro tree transducers, i.e., for $K \in \mathrm{MTT}^*(\mathrm{REGT})$,

(i) it is decidable whether or not K is empty, and
(ii) it is decidable whether or not K is finite; moreover, if the answer is yes, the list of elements of K can be computed.

Fact 24 is proved in Theorem 7.4 of [EV85]. It immediately implies Fact 25(i): since $K = \tau(R)$ is empty iff $\tau^{-1}(T_\Delta) \cap R$ is empty, the result follows from the fact that REGT is closed under intersection, and that emptiness of regular tree languages is decidable (cf. [GS84]). Fact 25(ii) is shown in Theorem 4.5 of [DE98].

In the remainder of this subsection, we relate the new class sMTT to the well-known class MTT of translations realized by mtts. In particular, it is proved in Theorem 31 that, in the deterministic case, stay-mtts realize the same class of translations as mtts, i.e., $\mathrm{DsMTT} = \mathrm{DMTT}$, and it is proved in Theorem 29 that, in the nondeterministic case, compositions of n stay-mtts can be realized by the composition of one stay-mtt and $n-1$ mtts, i.e., $\mathrm{sMTT}^n \subseteq \mathrm{sMTT} \circ \mathrm{MTT}^{n-1}$. In the nondeterministic case, which is proved first, the main proof is rather straightforward (Lemma 27), while the deterministic case (Theorem 31) has a quite involved proof.

Due to nondeterminism and the presence of stay moves, a stay-mtt M can generate infinitely many output trees for one particular input tree (see Example 6). This implies that M's translation can*not* be realized by an mtt, because, due to the absence of stay moves, in every computation step of an mtt a node of the input tree is "consumed"; hence, an mtt translates each input tree into a finite number of output trees. In order to eliminate stay moves from nondeterministic stay-mtts, we consider the translation mon_Σ (of Example 6) that inserts unary $\bar{\sigma}$'s above each symbol σ of a tree. Then, we can decompose M into mon_Σ followed by an mtt M'.

Notation 26. Let MON be the class of all mon_Σ for all ranked alphabets Σ.

Note that the 0-ptt M_Σ of Example 6 that realizes mon_Σ is also a stay-mtt. Thus, $\mathrm{MON} \subseteq \text{0-PTT}$ and $\mathrm{MON} \subseteq \mathrm{sMTT}$.

In the next lemma it is shown how to remove the stay instructions from a stay-mtt, by pre-composing with a translation in MON.

Lemma 27. $\mathrm{sMTT} \subseteq \mathrm{MON} \circ \mathrm{MTT}$.

Proof. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a 0-pmtt without up instructions. We construct a macro tree transducer M' such that $\mathrm{mon}_\Sigma \circ \tau_{M'} = \tau_M$. The idea of the construction of M' is as follows. Instead of staying at some σ-labeled node u of the input tree s, the new transducer M' will move down on the monadic piece of $\bar{\sigma}$-labeled nodes that are present above the σ-labeled node v in $\mathrm{mon}_\Sigma(s)$ that corresponds to u. In order to know, until we arrive at v, the child number of v in the original tree s, we keep this information in the states of M'. That is, states of the form (q, j) are used to simulate sequences of stay moves; this is done only on barred symbols, i.e., there are no rules for states of the form (q, j) and input symbols σ. As soon as there is a non-stay instruction, i.e., a down_i instruction into state q, we change into a state of the form (q, down_i). Such a state will move down the remaining monadic piece of $\bar{\sigma}$'s, and at the σ-labeled node v it will execute the down_i move into state q.

Let $M' = (\Gamma, \Delta, Q \cup Q', q_0, R')$ with $\Gamma = \Sigma \cup \bar{\Sigma}$, $\bar{\Sigma} = \{\bar{\sigma}^{(1)} \mid \sigma \in \Sigma\}$, and $Q' = Q \cup \langle Q, [0, J]\rangle \cup \langle Q, \mathrm{down}\rangle$, where 'down' denotes the set $\{\mathrm{down}_i \mid i \in [J]\}$ and $J = \max\{\mathrm{rank}_\Sigma(\sigma) \mid \sigma \in \Sigma\}$.

Let $\langle q, \sigma, \lambda, j\rangle(y_1, \dots, y_m) \to \zeta$ be a rule in R. Then let the rules

$$\begin{aligned}\langle q, \bar{\sigma}, \lambda, j\rangle(y_1, \dots, y_m) \quad &\to \zeta \Phi_j \Psi \\ \langle (q, j), \bar{\sigma}, \lambda, 1\rangle(y_1, \dots, y_m) &\to \zeta \Phi_j \Psi\end{aligned}$$

be in R', where the substitutions Φ_j and Ψ are defined as

$$\begin{aligned}\Phi_j &= [\![\langle q', \text{stay}\rangle \leftarrow \langle (q', j), \text{down}_1\rangle \mid q' \in Q]\!] \\ \Psi &= [\![\langle q', \text{down}_i\rangle \leftarrow \langle (q', \text{down}_i), \text{down}_1\rangle \mid q' \in Q, i \in [J]]\!].\end{aligned}$$

Moreover, for every $q \in Q^{(m)}$, $m \geq 0$, $\sigma \in \Sigma^{(k)}$, $k \geq 1$, and $i \in [k]$, let the rules

$$\begin{aligned}\langle (q, \text{down}_i), \bar{\sigma}, \lambda, 1\rangle(y_1, \dots, y_m) &\to \langle (q, \text{down}_i), \text{down}_1\rangle(y_1, \dots, y_m) \\ \langle (q, \text{down}_i), \sigma, \lambda, 1\rangle(y_1, \dots, y_m) &\to \langle q, \text{down}_i\rangle(y_1, \dots, y_m)\end{aligned}$$

be in R'. Obviously, the rules of M' do not contain stay instructions anymore, and thus M' is an mtt.

Before we prove the correctness of the construction of M', we need some auxiliary notions. Let $s \in T_\Sigma$ and $s' \in \text{mon}_\Sigma(s)$. Recall that s' is obtained from s by inserting above each σ-labeled node u, an arbitrary number of nodes u' labeled $\bar{\sigma}$ (of rank 1), which are "associated" with u. We now define the function dec, which maps each node u' of s' to the associated node u of s: Let $u' = i_1 \cdots i_m \in V(s')$ with $i_1, \dots, i_m \in [J]$ and $m \geq 0$. Define $\text{dec}(u') = i_{\nu_1} \cdots i_{\nu_n}$, where $\nu_1 < \cdots < \nu_n$, $n \geq 0$, are all indices $\mu \in [m]$ such that $s'[i_1 \cdots i_{\mu-1}] \in \Sigma$. Finally, we define the substitution $[\![\text{dec}]\!]$ which changes a sentential form of M' into one of M by relabeling the configurations of M' appropriately. Let $[\![\text{dec}]\!] = [\![Q]\!][\![\text{down}]\!]$ where $[\![Q]\!]$ denotes the substitution

$$\begin{aligned}[\![\langle r, (u', \lambda)\rangle \leftarrow \langle q', (\text{dec}(u'), \lambda)\rangle \mid\; & q' = r \text{ for } r \in Q \text{ and} \\ & q' = q \text{ for } r = (q, j) \in Q \times [0, J]]\!]\end{aligned}$$

and

$$[\![\text{down}]\!] = [\![\langle (q, \text{down}_i), (u', \lambda)\rangle \leftarrow \langle q, (\text{dec}(u')i, \lambda)\rangle \mid q \in Q, i \in [J]]\!].$$

In the sequel, we will also apply dec to input configurations h' of M', i.e., if $h' = (u', \lambda)$ then $\text{dec}(h') = (\text{dec}(u'), \lambda)$.

Next we state, without proof, two obvious properties about configurations that occur in sentential forms η of M' on an input tree $s' \in T_\Gamma$ with $s' \in \text{mon}_\Sigma(s)$ and $s \in T_\Sigma$. Since both properties are about (the child numbers in s) of <u>n</u>odes of s', we call them N1 and N2. Let $\langle q_0, h_0\rangle \Rightarrow^*_{M',s'} \eta$ and let $\langle p, (u', \lambda)\rangle$ be a configuration that occurs in η. Then

(N1) if $p = (q, j) \in (Q \times [0, J])$ then $\text{childno}(\text{dec}(u')) = j$, and
(N2) if $p \in Q$ then $\text{childno}(\text{dec}(u')) = \text{childno}(u')$.

Before it is proved, in Claims 2 and 3 that M' is correct, i.e., that $\text{mon}_\Sigma \circ \tau_{M'} = \tau_M$, we first relate in Claim 1 the right-hand side $\zeta \Phi_j \Psi$ of a $(Q \cup Q \times [0, J])$-rule of M' to the right-hand side ζ of the corresponding rule of M.

<u>Claim 1:</u> Let $s \in T_\Sigma$, $s' \in \text{mon}_\Sigma(s)$, $h = (u, \lambda) \in \text{IC}_{0,s}$, and $h' = (u', \lambda) \in \text{IC}_{0,s'}$ such that $\text{dec}(h') = h$ and $s'[u'] \in \bar{\Sigma}$. Let $[\![h]\!]$ denote $[\![h]\!]_{M,s}$, and let $[\![h']\!]$ denote $[\![h']\!]_{M',s'}$. Finally, let $\sigma = s[u]$ and $j = \text{childno}(u)$. For every $\zeta \in T_{\Delta \cup I_{\sigma,\lambda,j}}(Y_m)$, $m \geq 0$,

$$\zeta \Phi_j \Psi [\![h']\!][\![\text{dec}]\!] = \zeta [\![h]\!].$$

The proof of Claim 1 is by induction on the structure of ζ. If $\zeta = y \in Y_m$ then $\zeta \Phi_j \Psi [\![h']\!][\![\text{dec}]\!] = y = \zeta [\![h]\!]$, because none of the substitutions replaces parameters. Let $l \geq 0$ and $\zeta_1, \dots, \zeta_l \in T_{\Delta \cup I_{\sigma,\lambda,j}}(Y_m)$.

If $\zeta = \delta(\zeta_1, \ldots, \zeta_l)$ with $\delta \in \Delta^{(l)}$, then $\zeta \Phi_j \Psi[\![h']\!][\![\mathrm{dec}]\!] = \delta(\zeta_1 \Phi_j \Psi[\![h']\!][\![\mathrm{dec}]\!], \ldots, \zeta_l \Phi_j \Psi[\![h']\!][\![\mathrm{dec}]\!])$ which, by induction, is $\delta(\zeta_1[\![h]\!], \ldots, \zeta_l[\![h]\!]) = \delta(\zeta_1, \ldots, \zeta_l)[\![h]\!] = \zeta[\![h]\!]$.

If $\zeta = \langle q, \mathrm{stay}\rangle(\zeta_1, \ldots, \zeta_l)$ with $q \in Q^{(l)}$ then

$$\begin{aligned} \zeta\Phi_j\Psi &= \langle (q,j), \mathrm{down}_1\rangle(\zeta_1\Phi_j\Psi, \ldots, \zeta_l\Phi_j\Psi) \quad \text{and} \\ \zeta\Phi_j\Psi[\![h']\!][\![\mathrm{dec}]\!] &= \langle q, \mathrm{dec}(\mathrm{down}_1(h'))\rangle(\zeta_1\Phi_j\Psi[\![h']\!][\![\mathrm{dec}]\!], \ldots, \zeta_l\Phi_j\Psi[\![h']\!][\![\mathrm{dec}]\!]). \end{aligned}$$

By induction, and since $\mathrm{dec}(\mathrm{down}_1(h')) = \mathrm{dec}(h') = h$ (note that $s'[u'] = \bar{\sigma}$), this equals $\langle q, h\rangle(\zeta_1[\![h]\!], \ldots, \zeta_l[\![h]\!]) = \zeta[\![h]\!]$.

If $\zeta = \langle q, \mathrm{down}_i\rangle(\zeta_1, \ldots, \zeta_l)$ then

$$\begin{aligned} \zeta\Phi_j\Psi &= \langle (q,\mathrm{down}_i), \mathrm{down}_1\rangle(\zeta_1\Phi_j\Psi, \ldots, \zeta_l\Phi_j\Psi) \quad \text{and} \\ \zeta\Phi_j\Psi[\![h']\!][\![\mathrm{dec}]\!] &= \langle q, \mathrm{down}_i(\mathrm{dec}(\mathrm{down}_1(h')))\rangle(\zeta_1\Phi_j\Psi[\![h']\!][\![\mathrm{dec}]\!], \ldots, \zeta_l\Phi_j\Psi[\![h']\!][\![\mathrm{dec}]\!]). \end{aligned}$$

By induction, and since $\mathrm{dec}(\mathrm{down}_1(h')) = h$, this is $\langle q, \mathrm{down}_i(h)\rangle(\zeta_1[\![h]\!], \ldots, \zeta_l[\![h]\!]) = \zeta[\![h]\!]$, which concludes the proof of Claim 1.

Next, it is proved that $\mathrm{mon}_\Sigma \circ \tau_{M'} \subseteq \tau_M$. In fact, since $\langle q_0, h_0\rangle[\![\mathrm{dec}]\!] = \langle q_0, h_0\rangle$ and $t[\![\mathrm{dec}]\!] = t$ for $t \in T_\Delta$, this follows by induction from Claim 2.

Claim 2: Let $s \in T_\Sigma$ and $s' \in \mathrm{mon}_\Sigma(s)$. For every $\eta, \eta' \in T_{\Delta \cup C_{M',s'}}$ with $\langle q_0, h_0\rangle \Rightarrow^*_{M',s'} \eta$, if $\eta \Rightarrow_{M',s'} \eta'$ by a $(Q \cup Q \times [0,J])$-rule then $\eta[\![\mathrm{dec}]\!] \Rightarrow_{M,s} \eta'[\![\mathrm{dec}]\!]$, and if $\eta \Rightarrow_{M',s'} \eta'$ by a $\langle Q, \mathrm{down}\rangle$-rule then $\eta[\![\mathrm{dec}]\!] = \eta'[\![\mathrm{dec}]\!]$.

Let $v \in V(\eta)$ with $\eta[v] = \langle p, h'\rangle$ and $h' = (u', \lambda)$, $u' \in V(s')$, such that $\eta' = \eta[\![v \leftarrow \zeta'[\![h']\!]]\!]$ where ζ' is the right-hand side of a rule applicable to $\langle p, h'\rangle$. Let $u = \mathrm{dec}(u')$.

Case $p \in (Q \cup Q \times [0,J])$: Then $s'[u'] = \bar{\sigma}$ with $\sigma = s[u]$, because p-rules are only defined for barred input symbols. If $p = (q,j) \in Q \times [0,J]$ then, by N1, $\mathrm{childno}(u) = j$. If $p = q \in Q$ then, by N2, $\mathrm{childno}(u) = \mathrm{childno}(u')$. This means that in both cases $\zeta' = \zeta\Phi_j\Psi$ where ζ is the right-hand side of a $\langle q, \sigma, \lambda, j\rangle$-rule r of M and $j = \mathrm{childno}(u)$. Since $[\![\mathrm{dec}]\!]$ is a relabeling of configurations, v is outside in $\eta[\![\mathrm{dec}]\!]$ and labeled by the configuration $\langle q, (u, \lambda)\rangle$. Thus, r can be applied to v: $\eta[\![\mathrm{dec}]\!] \Rightarrow_{M,s} \eta[\![\mathrm{dec}]\!][\![v \leftarrow \zeta[\![h]\!]]\!]$. By Claim 1 the latter equals $\eta[\![\mathrm{dec}]\!][\![v \leftarrow \zeta'[\![h']\!][\![\mathrm{dec}]\!]]\!] = \eta[\![v \leftarrow \zeta'[\![h']\!]]\!][\![\mathrm{dec}]\!] = \eta'[\![\mathrm{dec}]\!]$, which proves the claim for this case.

Case $p = (q, \mathrm{down}_i) \in \langle Q^{(m)}, \mathrm{down}\rangle$, $m \geq 0$: By the definition of $[\![\mathrm{dec}]\!]$ this implies that $\eta[\![\mathrm{dec}]\!][v] = \langle q, (ui, \lambda)\rangle$. Moreover, $\zeta'[\![h']\!]$ is either equal to $\langle (q, \mathrm{down}_i), (u'1, \lambda)\rangle(y_1, \ldots, y_m)$, with $s'[u'] \in \bar{\Sigma}$, or equal to $\langle q, (u'i, \lambda)\rangle(y_1, \ldots, y_m)$, with $s'[u'] \in \Sigma$. In both cases, the application of $[\![\mathrm{dec}]\!]$ gives $\langle q, (ui, \lambda)\rangle(y_1, \ldots, y_m)$, which proves that $\eta[\![\mathrm{dec}]\!] = \eta'[\![\mathrm{dec}]\!]$. This ends the proof of Claim 2.

It remains to prove that $\tau_M \subseteq \mathrm{mon}_\Sigma \circ \tau_{M'}$. This will follow from Claim 3. Denote by $M'(Q)$ the restriction of M' to $(Q \cup Q \times [0,J])$-rules and denote by $M'(Q, \mathrm{down})$ its restriction to $\langle Q, \mathrm{down}\rangle$-rules. Intuitively, to simulate a computation step of M, M' first applies all possible $\langle Q, \mathrm{down}\rangle$-rules, and then it applies a $(Q \cup Q \times [0,J])$-rule. Let $\Rightarrow$ denote

$$\Rightarrow^*_{M'(Q,\mathrm{down}),s'} \circ \Rightarrow_{M'(Q),s'} .$$

Claim 3: Let $n \geq 0$, $s \in T_\Sigma$, and $s' \in \mathrm{mon}_\Sigma(s)$ such that for every $u \in V(s)$ $|\mathrm{dec}^{-1}(u)| \geq n+1$. Let $\xi \in T_{\Delta \cup C_{M,s}}$. If $\langle q_0, h_0\rangle \Rightarrow^n_{M,s} \xi$ then there exists an $\eta \in T_{\Delta \cup C_{M',s'}}$ such that

1. $\langle q_0, h_0\rangle \Rightarrow^n \eta$ and
2. $\eta[\![\mathrm{dec}]\!] = \xi$.

The proof of Claim 3 is by induction on n. If $n = 0$ then the statement holds for $\eta = \langle q_0, h_0\rangle$. Now consider the following computation of length $n+1$.

$$\langle q_0, h_0\rangle \Rightarrow^n_{M,s} \xi \Rightarrow_{M,s} \xi'.$$

By induction there exists an η such that $\langle q_0, h_0\rangle \Rightarrow^n \eta$ and $\eta[\![\mathrm{dec}]\!] = \xi$. Let $v \in V(\xi)$ and $\langle q, h\rangle \in C^{(m)}_{M,s}$, $m \geq 0$, such that $\xi[v] = \langle q, h\rangle$ and $\xi' = \xi[\![v \leftarrow \zeta[\![h]\!]]\!]$ where ζ is the right-hand side of a rule of M applicable to $\langle q, h\rangle$. By the definition of $[\![\mathrm{dec}]\!]$, $\eta[v] = \langle p, h'\rangle$ with (i) $p = q$, or (ii) $p = (q, j)$ with $j \in [0, J]$, or (iii) $p = (q, \mathrm{down}_i)$. Let $h = (u, \lambda)$ and $h' = (u', \lambda)$. We now show that there exists an η' such that $\eta \Rightarrow \eta'$ and $\eta'[\![\mathrm{dec}]\!] = \xi'$.

Cases (i) and (ii): Then $\mathrm{dec}(u') = u$. In case (i) it follows from N2 that childno(u') = childno(u), and in case (ii), i.e., $p = (q, j)$, it follows from N1 that j = childno(u). Since, in the computation $\langle q_0, h_0\rangle \Rightarrow^n \eta$ exactly n steps by $\Rightarrow_{M'(Q),s'}$ have been applied, $s'[u']$ must be a barred symbol, because there are $\geq n + 1$ of them, by the condition $|\mathrm{dec}^{-1}(u)| \geq n + 2$. Thus, $s'[u'] = \bar{\sigma}$ with $\sigma = s[u]$. Hence M' has a rule with right-hand side $\zeta' = \zeta\Phi_j\Psi$ which is applicable to $\langle p, h'\rangle$. We obtain $\eta \Rightarrow_{M'(Q),s'} \eta[\![v \leftarrow \zeta'[\![h']\!]]\!] = \eta'$. The application of $[\![\mathrm{dec}]\!]$ to η' gives, using Claim 1, $\eta[\![\mathrm{dec}]\!][\![v \leftarrow \zeta'[\![h']\!][\![\mathrm{dec}]\!]]\!] = \xi[\![v \leftarrow \zeta[\![h]\!]]\!] = \xi'$. This ends the proof of the claim for this case.

Case (iii) $p = (q, \mathrm{down}_i)$: Then $\eta \Rightarrow^+_{M'(Q,\mathrm{down}),s'} \eta''$ where η'' is the same as η except that $\eta''[v] = \langle q, (u''i, \lambda)\rangle$ with $s'[u''] \in \Sigma$, $\mathrm{dec}(u'') = \mathrm{dec}(u')$, and $\mathrm{dec}(u''i) = \mathrm{dec}(u')i = u$. By Claim 2, $\eta''[\![\mathrm{dec}]\!] = \eta[\![\mathrm{dec}]\!] = \xi$ and $\langle q_0, h_0\rangle \Rightarrow^n \eta''$. Now, to the configuration $\langle q, (u''i, \lambda)\rangle$ of η'' we can apply one step of $\Rightarrow_{M'(Q),s'}$, as shown in case (i), to obtain $\eta'' \Rightarrow_{M'(Q),s'} \eta'$ with $\eta'[\![\mathrm{dec}]\!] = \xi'$. This concludes the proof of Claim 3.

It should be obvious how to show that for every $s \in T_\Sigma$ there exists $s' \in \mathrm{mon}_\Sigma(s)$ such that $\tau_M(s) \subseteq \tau_{M'}(s')$: If $\langle q_0, h_0\rangle \Rightarrow^n_{M,s} t \in \tau_M(s)$, $n \geq 1$, then let $s' \in \mathrm{mon}_\Sigma(s)$ be as required in Claim 3. By Claim 3, applied to $\xi = t$, there exists η such that $\langle q_0, h_0\rangle \Rightarrow^*_{M',s'} \eta$ (because $\Rightarrow\ \subseteq\ \Rightarrow^*_{M',s}$) and $\eta[\![\mathrm{dec}]\!] = t$. Since $t \in T_\Delta$, $\eta = t$. Hence $t \in \tau_{M'}(s')$. □

The next small lemma shows that sMTT is closed under post-composition with MON. It will be needed to prove Theorems 29 and 30.

Lemma 28. sMTT ∘ MON ⊆ sMTT.

Proof. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a stay-mtt. We will construct the stay-mtt M' such that $\tau_{M'} = \tau_M \circ \mathrm{mon}_\Delta$. The idea of defining M' is to replace each output symbol δ (of rank m) in the right-hand side of a rule of M by a new state q_δ (of rank m), which will generate an arbitrary number of $\bar{\delta}$'s followed by the δ, i.e., a tree of the form $\bar{\delta}(\cdots\bar{\delta}(\delta(y_1, \ldots, y_m)))$.

Let $M' = (\Sigma, \Delta \cup \{\bar{\delta} \mid \delta \in \Delta\}, Q', q_0, R')$ where $Q' = Q \cup \{q^{(m)}_\delta \mid \delta \in \Delta^{(m)}, m \geq 0\}$. For every rule $\langle q, \sigma, b, j\rangle(y_1, \ldots, y_m) \to \zeta$ in R, let the rule

$$\langle q, \sigma, \lambda, j\rangle(y_1, \ldots, y_m) \to \zeta\Psi$$

be in R', where the substitution Ψ is defined as

$$\Psi = [\![\delta \leftarrow \langle q_\delta, \mathrm{stay}\rangle \mid \delta \in \Delta]\!].$$

Moreover, for every $\delta \in \Delta^{(m)}$, $m \geq 0$, $\sigma \in \Sigma$, and $j \in [0, J]$ let the rules

$$\begin{aligned}\langle q_\delta, \sigma, \lambda, j\rangle(y_1, \ldots, y_m) &\to \bar{\delta}(q_\delta)\\ \langle q_\delta, \sigma, \lambda, j\rangle(y_1, \ldots, y_m) &\to \delta(y_1, \ldots, y_m)\end{aligned}$$

be in R'.

A formal proof of the correctness of M' is left to the reader. □

For compositions of stay-mtts we obtain, from Lemmas 27 and 28, that stay moves can be removed from all transducers in the compositions, except the first one, as stated in the next theorem.

Theorem 29. For every $n \geq 1$, $\text{sMTT}^{n+1} = \text{sMTT} \circ \text{MTT}^n$.

Proof. By induction on n. For $n = 1$, $\text{sMTT}^2 \subseteq \text{sMTT} \circ \text{MON} \circ \text{MTT}$ by Lemma 27, which is included in $\text{sMTT} \circ \text{MTT}$ by Lemma 28. Now for $n+1$, $\text{sMTT}^{n+2} = \text{sMTT} \circ \text{sMTT}^{n+1}$ is included in $\text{sMTT}^2 \circ \text{MTT}^n$ by induction. By the case $n = 1$ the latter is included in $\text{sMTT} \circ \text{MTT} \circ \text{MTT}^n = \text{sMTT} \circ \text{MTT}^{n+1}$. □

It should be clear that the class REGT of regular tree languages is closed under MON, i.e., that $\text{MON}(\text{REGT}) \subseteq \text{REGT}$ (take the regular tree grammar in normal form, i.e., with at most one terminal symbol in the right-hand side of each production; for every production $A \to \sigma(A_1, \ldots, A_k)$ add all productions $A \to \bar{\sigma}(A_\sigma)$, $A_\sigma \to \bar{\sigma}(A_\sigma)$, and $A_\sigma \to \sigma(A_1, \ldots, A_k)$.) Thus we obtain from Theorem 29 and Lemma 27 that (compositions of) stay-mtts define the same output languages as (compositions of) mtts.

Theorem 30. For every $n \geq 1$, $\text{sMTT}^n(\text{REGT}) = \text{MTT}^n(\text{REGT})$.

Since $\text{MON} \subseteq \text{sMTT}$, we also obtain from Theorem 29 and Lemma 27 that $\text{sMTT}^* = \text{MON} \circ \text{MTT}^*$.

For deterministic stay-mtts we prove in the next theorem (and in the remainder of this subsection) that stay moves can be removed, i.e., the respective classes of translations coincide. As mentioned before, since the proof involves the nontrivial task of removing infinite computations, it is a key result of this paper.

Theorem 31. DsMTT = DMTT.

Proof. We have to show that DsMTT $\subseteq$ DMTT. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a 0-dpmtt without up instructions and let $J = \max\{\text{rank}_\Sigma(\sigma) \mid \sigma \in \Sigma\}$. We will construct the 0-dpmtt M' that has down instructions only, i.e., a dmtt, by removing the stay instructions that appear in the right-hand sides of the rules of M. Roughly speaking, this is done by applying rules to the stay instructions in a right-hand side, while keeping track of possible circular configurations, and forcing $\tau_{M'}(s)$ to be undefined if in the computation $\langle q_0, h_0\rangle \Rightarrow^*_{M,s}$ there is a sentential form that has an outside occurrence of a circular configuration (recall the notion of a circular configuration from Subsection 5.2, and see Lemma 21). Before M' is defined, we construct several intermediate 0-dpmtts: first N which has information about circular configurations, then N' which does not have circular configurations anymore, then N'' which does not execute stay instructions anymore, and finally M' which has only down instructions.

By Lemma 17 we may assume that M is total. First, we construct the 0-dpmtt N which is equivalent to M, but additionally keeps information in its states about which states have been passed, while staying at a particular node of the input tree.

Define $N = (\Sigma, \Delta, Q_N, (q_0, \varnothing), R_N)$ where $Q_N = \langle Q, \mathcal{P}(Q)\rangle$ and for every $(q, F) \in Q_N^{(m)}$, $m \geq 0$, $\sigma \in \Sigma$, $j \in [0, J]$, and rule $\langle q, \sigma, \lambda, j\rangle(y_1, \ldots, y_m) \to \zeta$ in R, the rule

$$\langle (q, F), \sigma, \lambda, j\rangle(y_1, \ldots, y_m) \to \zeta[\![\text{stay}_{q,F}]\!][\![\text{down}]\!]$$

is in R_N, where the substitutions $[\![\text{stay}_{q,F}]\!]$ and $[\![\text{down}]\!]$ are defined as

$$\begin{aligned}
[\![\text{stay}_{q,F}]\!] &= [\![\langle q', \text{stay}\rangle \leftarrow \langle (q', F \cup \{q\}), \text{stay}\rangle \mid q' \in Q]\!], \\
[\![\text{down}]\!] &= [\![\langle q', \text{down}_i\rangle \leftarrow \langle (q', \varnothing), \text{down}_i\rangle \mid q' \in Q, i \in [J]]\!].
\end{aligned}$$

Since N has, besides the additional sets F in its states, exactly the same rules as M, it obviously realizes the same translation as M, i.e., $\tau_N = \tau_M$. In fact, it can be shown easily that for all $\xi_1, \xi_2 \in T_{\Delta \cup C_{N,s}}(Y)$,

(C1) if $\xi_1 \Rightarrow_{N,s} \xi_2$ then $\xi_1[\![\text{no F's}]\!] \Rightarrow_{M,s} \xi_2[\![\text{no F's}]\!]$, and for all $\eta_1, \eta_2 \in T_{\Delta \cup C_{M,s}}(Y)$, and $\xi_1 \in T_{\Delta \cup C_{N,s}}(Y)$ with $\xi_1[\![\text{no F's}]\!] = \eta_1$,

(C2) if $\eta_1 \Rightarrow_{M,s} \eta_2$ then $\exists \xi_2 \in T_{\Delta \cup C_{N,s}}(Y) : \xi_2[\![\text{no F's}]\!] = \eta_2$ and $\xi_1 \Rightarrow_{N,s} \xi_2$, where the substitution $[\![\text{no F's}]\!]$ is defined as

$$[\![\langle (q,F), h\rangle \leftarrow \langle q,h\rangle \mid \langle (q,F),h\rangle \in C_{N,s}]\!].$$

By induction on the length of the computations, C1 implies $\tau_N \subseteq \tau_M$ and C2 implies $\tau_M \subseteq \tau_N$. Note that N is total because M is total.

The following claim expresses that the sets F in the states of N contain the intended states of M, i.e., those that were entered while staying at a particular node.

Claim: Let $s \in T_\Sigma$. If $\langle (q_0, \varnothing), h_0\rangle \Rightarrow^*_{N,s} \xi \in T_{\Delta \cup C_{N,s}}$ and $\langle (q,F),h\rangle \in C_{N,s}$ occurs outside in ξ, then for every $r \in F$ of rank $m \geq 0$,

(a) there is an $\eta \in T_{\Delta \cup C_{M,s}}$ such that $\langle q_0, h_0\rangle \Rightarrow^*_{M,s} \eta$ and $\langle r, h\rangle$ occurs outside in η, and
(b) there is an $\eta' \in T_{\Delta \cup C_{M,s}}(Y_m)$ such that $\langle r,h\rangle(y_1,\ldots,y_m) \Rightarrow^+_{M,s} \eta'$ and $\langle q,h\rangle$ occurs outside in η'.

Since this claim is intuitively obvious, but its proof is technically rather involved, we postpone its proof until after the present proof.

We now use the information in the states of N to remove its circular configurations, i.e., its infinite computations (cf. Lemma 21). Define $Q_{\text{cycle}} = \{(q,F) \in Q_N \mid q \in F\}$. We remove all rules for $(q,F) \in Q_{\text{cycle}}$ from R_N, thus obtaining the 0-dpmtt N'.

Formally, let $N' = (\Sigma, \Delta, Q_N, (q_0, \varnothing), R_{N'})$, where $R_{N'}$ is the set of all p-rules in R_N with $p \in Q_N - Q_{\text{cycle}}$.

It is straightforward to prove the correctness of the definition of N', i.e., that $\tau_{N'} = \tau_N$: Since $R_{N'} \subseteq R_N$, it clearly holds that $\tau_{N'} \subseteq \tau_N$. To prove that $\tau_N \subseteq \tau_{N'}$, let $s \in T_\Sigma$ and consider a complete computation $\xi_0 = \langle (q_0,\varnothing), h_0\rangle \Rightarrow_{N,s} \xi_1 \Rightarrow_{N,s} \cdots \Rightarrow_{N,s} \xi_n \in T_\Delta$. Then, for $i \in [0,n]$, ξ_i has no outside occurrence of $\langle p,h\rangle \in C_{N,s}$ with $p \in Q_{\text{cycle}}$. To see this, assume to the contrary that some ξ_i has an outside occurrence of $\langle (q,F),h\rangle \in C^{(m)}_{N,s}$ with $q \in F$ and $m \geq 0$. Then by the Claim above, there are $\eta, \eta' \in T_{\Delta \cup C_{M,s}}(Y)$ such that $\langle q_0,h_0\rangle \Rightarrow^*_{M,s} \eta$, $\langle q,h\rangle$ occurs outside in η, $\langle q,h\rangle(y_1,\ldots,y_m) \Rightarrow^+_{M,s} \eta'$, and $\langle q,h\rangle$ occurs outside in η', i.e., $\langle q,h\rangle$ is circular. By Lemma 21 this implies that the complete computations by $\Rightarrow_{M,s}$ starting with $\langle q_0,h_0\rangle$ are infinite, and hence that $\tau_M(s)$ is undefined. Since $\tau_N = \tau_M$ this contradicts the existence of the finite complete computation $\xi_0 \Rightarrow^*_{N,s} \xi_n$. Thus, only rules of N' are applied in the computation $\xi_0 \Rightarrow^*_{N,s} \xi_n$, which means that $\xi_0 \Rightarrow^*_{N',s} \xi_n$, and therefore $\tau_N \subseteq \tau_{N'}$. This ends the proof of the correctness of N'.

Next, the 0-dpmtt $N'' = (\Sigma, \Delta, Q_N, (q_0, \varnothing), R_{N''})$ is defined by iteratively applying rules to the stay instructions that appear in the right-hand side of each rule r of N'. This is done with the use of Lemma 15, changing N' gradually into N'' by iterating the following procedure. Initially, $N'' = N'$ and $R_{N''} = R_{N'}$. Now consider a rule $r = \langle (q,F), \sigma, \lambda, j\rangle(y_1,\ldots,y_m) \to \zeta$ in $R_{N''}$ and change it into the rule $\bar{r} = \langle (q,F), \sigma, \lambda, j\rangle(y_1,\ldots,y_m) \to \zeta\Phi_{\sigma,j}$ where

$$\Phi_{\sigma,j} = [\![\langle (q',F'), \text{stay}\rangle \leftarrow \zeta' \mid \langle (q',F'), \sigma, \lambda, j\rangle(y_1,\ldots,y_{m'}) \to \zeta' \text{ is in } R_{N''}]\!].$$

Note that if $\langle (q',F'), \text{stay}\rangle$ occurs in the right-hand side ζ of r, then F' has larger cardinality than the F in the left-hand side of r (and thus the same holds for $\bar{r}$).

Clearly, the new rule $\bar{r}$ can be obtained from the old rule r by iterated application of Lemma 15 (see the last paragraph of Subsection 2.2). Thus, by that lemma, an equivalent 0-dpmtt is obtained. After changing, in this way, every rule r into $\bar{r}$, the minimal cardinality of all state sets F such that $\langle (q,F), \mathrm{stay}\rangle$ occurs in a right-hand side of a rule in $R_{N''}$ for some q with $(q,F) \notin Q_{\mathrm{cycle}}$ has increased. Hence, after repeating this process at most $|Q|$ times, the only $\langle (q,F), \mathrm{stay}\rangle$ that occur in right-hand sides of rules satisfy $(q,F) \in Q_{\mathrm{cycle}}$ (for which there are no rules in $R_{N''}$). The resulting 0-dpmtt is, by definition, N''.

Last but not least, we define the dmtt M'. This is done by removing the stay instructions that appear in the rules of N''. Since N'' has no rules for states in Q_{cycle}, we can, in order to construct M', replace each stay instruction in a rule of N'' by a down_1 instruction (or remove the rule, if the input symbol has rank zero).

Formally, Let $M' = (\Sigma, \Delta, Q_N, (q_0, \varnothing), R')$ where R' is defined as follows. Let $r = \langle p, \sigma, \lambda, j\rangle(y_1, \ldots, y_m) \to \zeta$ with $\sigma \in \Sigma^{(k)}$ and $k \geq 0$ be a rule in $R_{N''}$. If $k = 0$ and $\zeta \in T_\Delta(Y_m)$ then let r be in R'. If $k \geq 1$ then let the rule

$$\langle p, \sigma, \lambda, j\rangle(y_1, \ldots, y_m) \to \zeta[\![\langle p', \mathrm{stay}\rangle \leftarrow \langle p', \mathrm{down}_1\rangle \mid p' \in Q_{\mathrm{cycle}}]\!]$$

be in R'. Obviously, M' is a dmtt. It is straightforward to show that $\tau_{M'} = \tau_{N''} = \tau_M$. □

In the remainder of this subsection, the Claim in the proof of Theorem 31 is proved. The uninterested reader can skip directly to Subsection 5.4.

For the proof of the Claim we need two technical lemmas, which are presented now. They state two general facts about pmtts. The first one is about the decomposition of computations, and the second one is about how to find the rule that generated a particular symbol during a computation. The first is needed to prove the second.

Consider a pmtt M, an input tree s, and a sentential form $\xi = \tau(\xi_1, \ldots, \xi_n)$ with $\tau \in C_{M,s}^{(n)}$. The first lemma states that a computation $\xi \Rightarrow_{M,s}^* \eta$ can be decomposed into $m+1$ computations by $\Rightarrow_{M,s}$ starting with $\tau(y_1, \ldots, y_n)$ and with $\xi_1, \ldots, \xi_n$ (for some $m \geq 0$). A similar result holds for macro grammars (cf. Theorem 4.1.1 of [Fis68], where only the case that η is terminal is considered). Note that the second item of Lemma 32 implies that

$$\tau(\xi_1, \ldots, \xi_n) \Rightarrow_{M,s}^{k_0} \eta_{\mathrm{lin}}[y_j \leftarrow \xi_{\pi(j)} \mid j \in [m]]$$

and that the third item implies that

$$\eta_{\mathrm{lin}}[y_j \leftarrow \xi_{\pi(j)} \mid j \in [m]] \Rightarrow_{M,s}^{k_1 + \cdots + k_m} \eta_{\mathrm{lin}}[y_j \leftarrow \eta_j \mid j \in [m]] = \eta.$$

Lemma 32. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a pmtt. Let $\tau \in (\Delta \cup C_{M,s})^{(n)}$, $n \geq 1$, and $\eta, \xi_1, \ldots, \xi_n \in T_{\Delta \cup C_{M,s}}(Y)$. If $\tau(\xi_1, \ldots, \xi_n) \Rightarrow_{M,s}^k \eta$, $k \geq 0$, then there exists a tree $\eta_{\mathrm{lin}} \in T_{\Delta \cup C_{M,s}}(Y_m)$, $m \geq 0$, such that η_{lin} is linear in Y_m (i.e., each $y \in Y_m$ appears at most once in η_{lin}), and there exist a mapping $\pi : [m] \to [n]$, trees $\eta_1, \ldots, \eta_m \in T_{\Delta \cup C_{M,s}}(Y)$, and $k_0, k_1, \ldots, k_m \in \mathbb{N}$ such that

- $\eta = \eta_{\mathrm{lin}}[y_j \leftarrow \eta_j \mid j \in [m]]$,
- $\tau(y_1, \ldots, y_n) \Rightarrow_{M,s}^{k_0} \eta_{\mathrm{lin}}[y_j \leftarrow y_{\pi(j)} \mid j \in [m]]$,
- for every $j \in [m]$, $\xi_{\pi(j)} \Rightarrow_{M,s}^{k_j} \eta_j$,
- $k_0 + k_1 + \cdots + k_m = k$, and
- for every $j \in [m]$, if y_j does not occur outside in η_{lin} then $k_j = 0$.

Proof. The proof is by induction on the length k of the computation of η. It is obvious for $k = 0$: take $\eta_{\mathrm{lin}} = \tau(y_1, \ldots, y_n)$, $m = n$, π is the identity on $[n]$, $\eta_j = \xi_j$ and $k_j = 0$ for $j \in [0, n]$. Now consider the following computation of length $k+1$

$$\tau(\xi_1, \ldots, \xi_n) \Rightarrow_{M,s}^k \eta \Rightarrow_{M,s} \eta'. \qquad (*)$$

By induction, $\eta = \eta_{\text{lin}}[y_j \leftarrow \eta_j \mid j \in [m]]$ where η_{lin} and $\eta_1, \dots, \eta_m$ satisfy the conditions of the lemma, for certain $\pi : [m] \to [n]$ and $k_0, \dots, k_m \in \mathbb{N}$. Let v be the node in η to which a rule is applied in the last step of the computation $(*)$.

Case 1: $v \in V(\eta_{\text{lin}})$ and $\eta_{\text{lin}}[v] \notin Y_m$. Hence, $\eta_{\text{lin}}[v] = \eta[v]$ and therefore we can apply the rule of the last step in $(*)$ to η_{lin}: $\eta_{\text{lin}} \Rightarrow_{M,s} \tilde{\eta}$ with $\eta' = \tilde{\eta}[y_j \leftarrow \eta_j \mid j \in [m]]$. Let m' be the number of occurrences of parameters in the tree $\tilde{\eta}$. Next, we "linearize" (in the parameters) the tree $\tilde{\eta}$: let $\eta'_{\text{lin}} \in T_{\Delta \cup C_{M,s}}(Y_{m'})$ and $\tilde{\pi} : [m'] \to [m]$ such that

$$\tilde{\eta} = \eta'_{\text{lin}}[y_j \leftarrow y_{\tilde{\pi}(j)} \mid j \in [m']].$$

Note that for every $j \in [m]$, if $\tilde{\pi}^{-1}(j)$ is not a singleton (i.e., if y_j does not occur exactly once in $\tilde{\eta}$) then y_j occurs at a descendant of v in η_{lin}, and so, by the last condition of the lemma, $k_j = 0$. This shows that $k_{\tilde{\pi}(1)} + \cdots + k_{\tilde{\pi}(m')} = k_1 + \cdots + k_m$. Now define $\eta'_j = \eta_{\tilde{\pi}(j)}$ and $k'_j = k_{\tilde{\pi}(j)}$ for $j \in [m']$, and define $k'_0 = k_0 + 1$ and $\pi' = \tilde{\pi} \circ \pi : [m'] \to [n]$. Then

$$\begin{aligned} \eta' &= \eta'_{\text{lin}}[y_j \leftarrow y_{\tilde{\pi}(j)} \mid j \in [m']][y_j \leftarrow \eta_j \mid j \in [m]] \\ &= \eta'_{\text{lin}}[y_j \leftarrow \underbrace{\eta_{\tilde{\pi}(j)}}_{=\eta'_j} \mid j \in [m']] \end{aligned}$$

and $\tau(y_1, \dots, y_n) \Rightarrow^{k_0}_{M,s} \eta_{\text{lin}}[y_j \leftarrow y_{\pi(j)} \mid j \in [m]] \Rightarrow_{M,s} \tilde{\eta}[y_j \leftarrow y_{\pi(j)} \mid j \in [m]]$. The latter tree equals $\eta'_{\text{lin}}[y_j \leftarrow y_{\tilde{\pi}(j)} \mid j \in [m']][y_j \leftarrow y_{\pi(j)} \mid j \in [m]] = \eta'_{\text{lin}}[y_j \leftarrow y_{\pi(\tilde{\pi}(j))} \mid j \in [m']]$ which is equal to $\eta'_{\text{lin}}[y_j \leftarrow y_{\pi'(j)} \mid j \in [m']]$. Thus, the "primed versions" of the first four conditions of the lemma hold. It remains to prove the last condition of the lemma. Let $j \in [m']$. Clearly, if y_j does not occur outside in η'_{lin} then $y_{\tilde{\pi}(j)}$ does not occur outside in η_{lin} and hence $k_{\tilde{\pi}(j)} = 0$ by the last condition for η_{lin}.

Case 2: $v \notin V(\eta_{\text{lin}})$ or $\eta_{\text{lin}}[v] \in Y_m$. This means that there is a $j_0 \in [m]$ such that y_{j_0} occurs outside in η_{lin}, $\eta_{j_0} \Rightarrow_{M,s} \eta'_{j_0}$, and $\eta' = \eta_{\text{lin}}[y_j \leftarrow \eta_j \mid j \in [m] - \{j_0\}][y_{j_0} \leftarrow \eta'_{j_0}]$. Hence, for $k'_{j_0} = k_{j_0} + 1$ (and everything else the same) the statement of the lemma holds. □

The second lemma is based on the following technical notions. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a pmtt and let $s \in T_\Sigma$. For a symbol α

– $\xi \in T_{\Delta \cup C_{M,s}}(Y)$ *computes* α if there is a ξ' such that $\xi \Rightarrow^*_{M,s} \xi'$ and ξ' has an outside occurrence of α, and
– $c \in C^{(m)}_{M,s}$ (with $m \geq 0$) *directly computes* α if there is a $\xi \in T_{\Delta \cup C_{M,s}}(Y_m)$ such that $c(y_1, \dots, y_m) \Rightarrow_{M,s} \xi$, α occurs in ξ, and ξ computes α.

If, in the first definition, $\xi \Rightarrow^k_{M,s} \xi'$ then we say that ξ k-computes α. If ξ k-computes α for $\xi = c(y_1, \dots, y_m)$, $c \in C^{(m)}_{N,s}$, and $m \geq 0$, then we say that c k-computes α. Clearly, computing configurations is transitive, that is, for configurations a, b, and c and $k_1, k_2 \in \mathbb{N}$, if a k_1-computes b and b k_2-computes c, then a $(k_1 + k_2)$-computes c (and similarly, for a replaced by a tree ξ).

Consider now Lemma 33. Intuitively, the lemma states that if configuration c computes another configuration d, then a rule r must have been applied, which contains d in its right-hand side ζ (in the lemma, r is the rule of M that is applicable to c'). To be more precise, d is in $\zeta[\![h]\!]_{M,s}$ where h is the input configuration of c.

Recall the Claim in the proof of Theorem 31. In the proof of that claim we will apply Lemma 33 to $d = \langle (q, F), h \rangle$ with $F \neq \varnothing$; then, by the definition of the rules of M, r must have $\langle (q, F), \text{stay} \rangle$ in its right-hand side, and thus c' must equal $\langle (q', F'), h \rangle$ for some $(q', F') \in Q_M$. Note that, in Lemma 33, c' is not necessarily different from c.

Lemma 33. Let M be a pmtt and let s be an input tree of M. Let $c, d \in C_{M,s}$ with $c \neq d$ and let $k' \in \mathbb{N}$. If c k'-computes d, then there are $c' \in C_{M,s}$ and $k'' < k'$ such that (1) c k''-computes c' and (2) c' directly computes d.

Proof. The proof is by induction on k'. Let m be the rank of c. Since $c \neq d$, $k' \geq 1$, i.e., there is a computation

$$c(y_1, \ldots, y_m) \Rightarrow_{M,s} \xi \Rightarrow^k_{M,s} \xi'$$

where $k = k' - 1$, $\xi, \xi' \in T_{\Delta \cup C_{M,s}}(Y_m)$, and d occurs outside in ξ'. If d occurs in ξ then the lemma holds for $c' = c$ and $k'' = 0$. Consider now the case that d does not occur in ξ. Since ξ k-computes d, we can apply the claim below to obtain a configuration $\tilde{c}$ in ξ and $\tilde{k}, l \in \mathbb{N}$ such that ξ $\tilde{k}$-computes $\tilde{c}$, $\tilde{c}$ l-computes d, and $\tilde{k} + l \leq k$. Now, $\tilde{c} \neq d$ because $\tilde{c}$ occurs in ξ and d does not. Since $l < k'$, we can apply the induction hypothesis (to $\tilde{c}$, l, and d). Hence, there are $\tilde{c}'$ and $l' < l$ such that $\tilde{c}$ l'-computes $\tilde{c}'$ and $\tilde{c}'$ directly computes d. Since $c(y_1, \ldots, y_m) \Rightarrow_{M,s} \xi$ and ξ $\tilde{k}$-computes $\tilde{c}$, c $(\tilde{k}+1)$-computes $\tilde{c}$. By the transitivity of computing configurations we obtain that c $(\tilde{k} + 1 + l')$-computes $\tilde{c}'$. It follows from $\tilde{k} + l \leq k$ and $l' < l$ that $\tilde{k} + l' < k = k' - 1$, and therefore $\tilde{k} + 1 + l' < k'$. Thus, the lemma holds for $c' = \tilde{c}'$ and $k'' = \tilde{k} + 1 + l'$. It remains to prove the claim.

<u>*Claim:*</u> Let $\xi \in T_{\Delta \cup C_{M,s}}(Y)$ and $k \in \mathbb{N}$. If ξ k-computes d, then there are $\tilde{k}, l \in \mathbb{N}$ and a configuration $\tilde{c}$ in ξ such that $\tilde{k} + l \leq k$, ξ $\tilde{k}$-computes $\tilde{c}$, and $\tilde{c}$ l-computes d.

The proof is by induction on the structure of ξ. Since ξ k-computes d there is an η such that $\xi \Rightarrow^k_{M,s} \eta$ and η has an outside occurrence of d. This implies that $\xi \notin Y$, i.e., ξ is of the form $\tau(\xi_1, \ldots, \xi_n)$ for $\tau \in (\Delta \cup C_{M,s})^{(n)}$, $n \geq 0$, and $\xi_1, \ldots, \xi_n \in T_{\Delta \cup C_{M,s}}(Y)$.

We now apply Lemma 32 to the computation $\tau(\xi_1, \ldots, \xi_n) \Rightarrow^k_{M,s} \eta$, and obtain a tree $\eta_{\text{lin}} \in T_{\Delta \cup C_{M,s}}(Y_m)$, $m \geq 0$, which is linear in Y_m, a mapping $\pi : [m] \to [n]$, $\eta_1, \ldots, \eta_m \in T_{\Delta \cup C_{M,s}}(Y)$, and $k_0, k_1, \ldots, k_m \in \mathbb{N}$ such that (1) $\eta = \eta_{\text{lin}}[y_j \leftarrow \eta_j \mid j \in [m]]$, (2) $\tau(y_1, \ldots, y_n) \Rightarrow^{k_0}_{M,s} \eta_{\text{lin}}[y_j \leftarrow y_{\pi(j)} \mid j \in [m]]$, (3) for every $j \in [m]$, $\xi_{\pi(j)} \Rightarrow^{k_j}_{M,s} \eta_j$, and (4) $k_0 + k_1 + \cdots + k_m = k$.

Case (i): d occurs outside in η_{lin}. Then $\tau \in C_{M,s}$ and τ k_0-computes d. Hence, for $\tilde{k} = 0$, $l = k_0$, and $\tilde{c} = \tau$ the claim holds.

Case (ii): d does not occur outside in η_{lin}. Since d occurs outside in η, there must be a $j \in [m]$ such that y_j occurs outside in η_{lin} and d occurs outside in η_j. This implies that $\xi_{\pi(j)}$ k_j-computes d. By induction there are $\tilde{k}, l \in \mathbb{N}$ and a configuration $\tilde{c}$ in $\xi_{\pi(j)}$ such that $\tilde{k} + l \leq k_j$, $\xi_{\pi(j)}$ $\tilde{k}$-computes $\tilde{c}$, and $\tilde{c}$ l-computes d. Since $\xi = \tau(\xi_1, \ldots, \xi_n) \Rightarrow^{k_0}_{M,s} \eta_{\text{lin}}[y_j \leftarrow \xi_{\pi(j)} \mid j \in [m]] = \eta''$ and y_j occurs outside in η_{lin} (at ρ), every outside node (v) in $\xi_{\pi(j)}$ is also outside in η'' (at ρv). Hence, since $\xi_{\pi(j)}$ $\tilde{k}$-computes $\tilde{c}$, we obtain that η'' $\tilde{k}$-computes $\tilde{c}$ and so ξ $(k_0 + \tilde{k})$-computes $\tilde{c}$. It follows from $\tilde{k} + l \leq k_j$ that $(k_0 + \tilde{k}) + l \leq k_0 + k_j$, which is $\leq k$ because $\sum_{\mu \in [m]} k_\mu = k$. This concludes the proof of the claim and hence of the lemma.

Proof of the Claim in the proof of Theorem 31. For ease of reference we repeat the claim.

<u>*Claim:*</u> Let $s \in T_\Sigma$. If $\langle (q_0, \varnothing), h_0 \rangle \Rightarrow^*_{N,s} \xi \in T_{\Delta \cup C_{N,s}}$ and $\langle (q, F), h \rangle \in C_{N,s}$ occurs outside in ξ, then for every $r \in F$ of rank $m \geq 0$,

(a) there is an $\eta \in T_{\Delta \cup C_{M,s}}$ such that $\langle q_0, h_0 \rangle \Rightarrow^*_{M,s} \eta$ and $\langle r, h \rangle$ occurs outside in η, and
(b) there is an $\eta' \in T_{\Delta \cup C_{M,s}}(Y_m)$ such that $\langle r, h \rangle(y_1, \ldots, y_m) \Rightarrow^+_{M,s} \eta'$ and $\langle q, h \rangle$ occurs outside in η'.

The proof is by induction on the length k of the given computation. Assume it holds for all $i < k$ and consider a state $r \in F$. The application of Lemma 33 to $c = \langle (q_0, \varnothing), h_0 \rangle$, $d = \langle (q, F), h \rangle$, and $k' = k$ gives an $i < k$ and a configuration c' such that c i-computes c' and c' directly computes d. Since $\langle (q, F), \text{stay} \rangle$ must appear in the right-hand side of the rule applicable to c', it follows from the definition of the rules of N that c' is of the form $\langle (q', F'), h \rangle$ with $F = \{q'\} \cup F'$. It follows from C1 (in the proof of Theorem 31) that there is an $\tilde{\eta}$ such that $\langle q_0, h_0 \rangle \Rightarrow^*_{M,s} \tilde{\eta}$ and $\tilde{\eta}$ has an outside occurrence of $\langle q', h \rangle$, and that there is an $\tilde{\eta}'$ such that $\langle q', h \rangle (y_1, \dots, y_n) \Rightarrow^+_{M,s} \tilde{\eta}'$ and $\langle q, h \rangle$ occurs outside (at ρ) in $\tilde{\eta}'$ (where n is the rank of q'). If $q' = r$, then the claim holds for $\eta = \tilde{\eta}$ and $\eta' = \tilde{\eta}'$.

Consider now the case that $q' \neq r$. Since $F = F' \cup \{q'\}$, r must be in F'. We apply the induction hypothesis to $\langle (q_0, \varnothing), h_0 \rangle$, i, and $\langle (q', F'), h \rangle$ to obtain an η such that $\langle q_0, h_0 \rangle \Rightarrow^*_{M,s} \eta$ and $\langle r, h \rangle$ occurs outside in η, and an $\bar{\eta}$ with $\langle r, h \rangle (y_1, \dots, y_m) \Rightarrow^+_{M,s} \bar{\eta}$ and $\langle q', h \rangle$ occurs outside in $\bar{\eta}$ (at v). Thus, part (a) of the claim holds. Since $\langle q', h \rangle (y_1, \dots, y_n) \Rightarrow^+_{M,s} \tilde{\eta}'$, there is a computation $\bar{\eta} \Rightarrow^+_{M,s} \bar{\eta}[\![v \leftarrow \tilde{\eta}']\!] = \eta'$ and $\langle q, h \rangle$ occurs outside in η' (at $v\rho$). This proves (b) and concludes the proof of the Claim. □

5.4 Simulation of PTTs by Macro Tree Transducers

In this subsection it is proved that, by the use of parameters, we can remove all up instructions from a 0-ptt M, thus obtaining a stay-mtt that realizes the same translation as M.

In fact, this result is already known. It was proved in [EV86] in the setting of transducers with storage. As discussed at the end of Subsection 3.3, 0-PTT = RT(Tree-walk) and hence 0-PTT ⊆ RT(P(Tree)). By Theorem 5.14, Corollary 5.21, and Theorem 4.18 of [EV86], RT(P(Tree)) ⊆ CFT(Tree$_{\text{id}}$). Here, 'id' indicates the addition of an identity instruction (Definition 3.7 of [EV86]) and thus the possibility to stay at a node. Since, as observed in Subsection 3.3, CFT(Tree) = MTT, it should be clear that CFT(Tree$_{\text{id}}$) is precisely the class sMTT. In the same way it follows that 0-DPTT ⊆ DsMTT, because the proofs preserve determinism. Since the proof in [EV86] is complicated by the fact that it is shown for arbitrary storage types, we present here a direct proof for completeness sake.

Since DsMTT = DMTT by Theorem 31, the fact that 0-DPTT ⊆ DsMTT proves that 0-DPTT ⊆ DMTT (and this *is* a new result). For total functions this result was also proved in [EV86] (Theorem 5.16); in the noncircular case (see Subsection 3.2) it is the well-known fact that attribute grammars can be simulated by macro tree transducers [Fra82,CF82,FV99,EM99].

Lemma 34. 0-PTT ⊆ sMTT and 0-DPTT ⊆ DsMTT.

Proof. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a 0-ptt and let $q^1, \dots, q^m$ be the states in Q. We want to construct a 0-pmtt M' without up instructions that realizes the same translation as M. The idea of M' is to replace each up instruction into state q^ν, by the selection of the parameter y_ν. Hence, if the current node is v, then in the νth parameter position of a state of M', we have to compute what M does at the parent of v. Obviously, if v is the root node, then there is no parent, and therefore the corresponding states of M' have no parameters. More precisely, M' has states $(q, 0)$ of rank zero which are used if the current node is the root node, and if the current node is not the root node, then M' uses states (q, m) of rank m. For every move of M from v to its jth child vj, M' computes in the νth parameter position of the new state (q, m) what happens if M moves back to v into state q^ν. Thus, the parameters are used in a stack-like fashion, to keep a history of the computations of all states for all ancestors of the current node; in that way moving up into state

q^ν is realized by M' by selecting the parameter y_ν, and therefore M' has no up instructions. Note that this kind of stack technique was invented by Rounds (cf. Theorem 7 of [Rou70], which was generalized in Lemma 5.4 of [EV86]).

Let us now define M'. Let $M' = (\Sigma, \Delta, Q', (q_0, 0), R')$, where $Q' = \{(q,\mu)^{(\mu)} \mid q \in Q, \mu \in \{0, m\}\}$ and $R' = \{\mathrm{rel}(r) \mid r \in R\}$. For every rule $r \in R$ the related rule $\mathrm{rel}(r)$ is defined as follows. Let $r = (\langle q, \sigma, \lambda, j\rangle \to \zeta)$ with $q \in Q$, $\sigma \in \Sigma$, and $j \in [0, J]$. Then

$$\mathrm{rel}(r) = \begin{cases} \langle (q,0), \sigma, \lambda, 0\rangle & \to \mathrm{trans}_0(\zeta), \quad \text{if } j = 0 \\ \langle (q,m), \sigma, \lambda, j\rangle(y_1, \ldots, y_m) & \to \mathrm{trans}_m(\zeta), \quad \text{otherwise} \end{cases}$$

where, for $\mu \in \{0, m\}$, $\mathrm{trans}_\mu(\zeta) =$

- y_ν if $\zeta = \langle q^\nu, \mathrm{up}\rangle$
- $\delta((r_1, \mu)(y_1, \ldots, y_\mu), \ldots, (r_k, \mu)(y_1, \ldots, y_\mu))$ if $\zeta = \delta(r_1, \ldots, r_k)$ with $\delta \in \Delta^{(k)}$, $k \geq 0$, and $r_1, \ldots, r_k \in Q$,
- and if $\zeta = \langle q', \varphi\rangle$ with $\varphi \in \{\mathrm{down}_i \mid i \in [J]\} \cup \{\mathrm{stay}\}$ then it equals
  - $\langle (q', m), \mathrm{down}_i\rangle((q^1, \mu)(y_1, \ldots, y_\mu), \ldots, (q^m, \mu)(y_1, \ldots, y_\mu))$ if $\varphi = \mathrm{down}_i$
  - $\langle (q', \mu), \mathrm{stay}\rangle(y_1, \ldots, y_\mu)$ if $\varphi = \mathrm{stay}$.

Obviously, if M is deterministic, then so is M'.

Let $s \in T_\Sigma$. Before we prove the correctness of the construction of M' we need some auxiliary notions. Define the full m-ary "stack tree" (fmt) that is generated by M' in order to keep track of the computations at ancestors as follows. For a configuration $\langle q, (u, \lambda)\rangle$ of M, the tree $\mathrm{fmt}(\langle q, (u,\lambda)\rangle) \in T_{C_{M',s}}$ is defined as $\langle (q, 0), h\rangle$ if $u = \varepsilon$, and otherwise as $\langle (q,m), h\rangle(\mathrm{fmt}(\langle q^1, \mathrm{up}(h)\rangle), \ldots, \mathrm{fmt}(\langle q^m, \mathrm{up}(h)\rangle))$, where $h = (u, \lambda)$. We can now define the substitution Φ, that allows us to extend the notion of relatedness from rules to sentential forms:

$$\Phi = [\langle q, (u,\lambda)\rangle \leftarrow \mathrm{fmt}(\langle q, (u, \lambda)\rangle) \mid q \in Q, u \in V(s)].$$

Two sentential forms $\xi \in T_{\Delta \cup C_{M,s}}$ and $\xi' \in T_{\Delta \cup C_{M',s}}$ are related, if $\xi' = \xi\Phi$.

Claim 1: For $c \in C_{M,s}$ and $r \in R$, r is applicable to c iff $\mathrm{rel}(r)$ is applicable to $c\Phi[\varepsilon]$.

Let $c = \langle q, (u, \lambda)\rangle$ and $r = \langle q, \sigma, \lambda, j\rangle \to \zeta$. The rule r is applicable to c iff $s[u] = \sigma$ and $j = \mathrm{childno}(u)$ iff $\mathrm{rel}(r)$ is applicable to $c\Phi[\varepsilon]$ because, for $u = \varepsilon$, $\langle q, (u,\lambda)\rangle\Phi[\varepsilon] = \langle (q, 0), (u, \lambda)\rangle$ and the left-hand side of $\mathrm{rel}(r)$ is $\langle (q,0), \sigma, \lambda, 0\rangle$, and for $u \neq \varepsilon$, $\langle q, (u, \lambda)\rangle\Phi[\varepsilon] = \langle (q, m), (u, \lambda)\rangle$ and the left-hand side of $\mathrm{rel}(r)$ is $\langle (q, m), \sigma, \lambda, j\rangle(y_1, \ldots, y_m)$. This proves Claim 1.

By Claim 2 below, the application of related rules to the same node in related sentential forms yields again related sentential forms. Now, if $\xi_1 \Rightarrow_{M,s} \xi_2$ by rule r at node ρ and ξ_1' is related to ξ_1, then by Claim 1 $\mathrm{rel}(r)$ is applicable to ξ_1' at ρ because $\xi_1'[\rho] = (\xi_1[\rho])\Phi[\varepsilon]$, and, by Claim 2, ξ_2' is related to ξ_2 where $\xi_1' \Rightarrow_{M',s} \xi_2'$ by $\mathrm{rel}(r)$. Thus, if $\langle q_0, h_0\rangle \Rightarrow^*_{M,s} t \in T_\Delta$, then $\langle (q_0, 0), h_0\rangle \Rightarrow^*_{M',s} t$ because $\langle q_0, h_0\rangle$ is related to $\langle (q_0, 0), h_0\rangle$. This means that $\tau_M \subseteq \tau_{M'}$. Similarly, $\langle (q_0, 0), h_0\rangle \Rightarrow^*_{M',s} t \in T_\Delta$ implies that $\langle q_0, h_0\rangle \Rightarrow^*_{M,s} t$ and thus $\tau_{M'} \subseteq \tau_M$. It remains to prove Claim 2.

Claim 2: Let $\xi_1, \xi_2 \in T_{\Delta \cup C_{M,s}}$ and $\xi_1', \xi_2' \in T_{\Delta \cup C_{M',s}}$ such that ξ_1 and ξ_1' are related. If $\xi_1 \Rightarrow_{M,s} \xi_2$ by rule $r \in R$ at node ρ in ξ_1 and $\xi_1' \Rightarrow_{M',s} \xi_2'$ by rule $\mathrm{rel}(r)$ at node ρ in ξ_1', then ξ_2 and ξ_2' are related.

Let $\xi_1[\rho] = \langle q, (u, \lambda)\rangle$ and $r = \langle q, \sigma, \lambda, j\rangle \to \zeta$. Let $\mu = 0$ if $u = \varepsilon$ and otherwise $\mu = m$. Then $\xi_1'/\rho = \mathrm{fmt}(\langle q, (u, \lambda)\rangle) = \langle (q, \mu), (u, \lambda)\rangle(t_1, \ldots, t_\mu)$ with $t_i = \mathrm{fmt}(\langle q^i, \mathrm{up}(u, \lambda)\rangle)$ for $i \in [\mu]$.

If $\zeta = \langle q', \mathrm{stay}\rangle$ then $\xi_2 = \xi_1[\rho \leftarrow \langle q', (u, \lambda)\rangle]$ and $\mathrm{rel}(r)$ has right-hand side $\langle (q', \mu), \mathrm{stay}\rangle(y_1, \ldots, y_\mu)$. Then $\xi_1' \Rightarrow_{M',s} \xi_2' = \xi_1'[\rho \leftarrow \langle (q', \mu), (u, \lambda)\rangle(t_1, \ldots, t_\mu)] = \xi_1'[\rho \leftarrow \mathrm{fmt}(\langle q', (u, \lambda)\rangle)] = \xi_1[\rho \leftarrow \langle q', (u, \lambda)\rangle]\Phi = \xi_2\Phi$.

If $\zeta = \delta(r_1,\dots,r_k)$ then $\xi_2 = \xi_1[\rho \leftarrow \delta(\langle r_1,(u,\lambda)\rangle,\dots,\langle r_k,(u,\lambda)\rangle)]$ and $\mathrm{rel}(r)$ has right-hand side $\delta((r_1,\mu)(y_1,\dots,y_\mu),\dots,(r_k,\mu)(y_1,\dots,y_\mu))$. Then $\xi_1' \Rightarrow_{M',s}$ $\xi_2' = \xi_1'[\rho \leftarrow \delta(\langle(r_1,\mu),(u,\lambda)\rangle(t_1,\dots,t_\mu),\dots,\langle(r_k,\mu),(u,\lambda)\rangle(t_1,\dots,t_\mu))]$
$= \xi_1'[\rho \leftarrow \delta(\mathrm{fmt}(\langle r_1,(u,\lambda)\rangle),\dots,\mathrm{fmt}(\langle r_k,(u,\lambda)\rangle))]$
$= \xi_1[\rho \leftarrow \delta(\langle r_1,(u,\lambda)\rangle,\dots,\langle r_k,(u,\lambda)\rangle)]\Phi = \xi_2\Phi$.

If $\zeta = \langle q',\mathrm{down}_i\rangle$ then $\xi_2 = \xi_1[\rho \leftarrow \langle q',(ui,\lambda)\rangle]$ and $\mathrm{rel}(r)$ has right-hand side $\langle(q',m),\mathrm{down}_i\rangle((q^1,\mu)(y_1,\dots,y_\mu),\dots,(q^m,\mu)(y_1,\dots,y_\mu))$. Then $\xi_1' \Rightarrow_{M',s} \xi_2' =$ $\xi_1'[\rho \leftarrow \langle(q',m),(ui,\lambda)\rangle(\langle(q^1,\mu),(u,\lambda)\rangle(t_1,\dots,t_\mu),\dots,\langle(q^m,\mu),(u,\lambda)\rangle(t_1,\dots,t_\mu))]$ $= \xi_1'[\rho \leftarrow \mathrm{fmt}(\langle q',(ui,\lambda)\rangle)] = \xi_1[\rho \leftarrow \langle q',(ui,\lambda)\rangle]\Phi = \xi_2\Phi$.

If $\zeta = \langle q',\mathrm{up}\rangle$ then $u = u'i$ for some $u' \in V(s)$ and $i \in [J]$, and $\xi_2 = \xi_1[\rho \leftarrow$ $\langle q',(u',\lambda)\rangle]$. The right-hand side of $\mathrm{rel}(r)$ is y_ν for $\nu \in [m]$ such that $q^\nu = q'$. Thus $\xi_1' \Rightarrow_{M',s} \xi_2' = \xi_1'[\rho \leftarrow t_\nu] = \xi_1'[\rho \leftarrow \mathrm{fmt}(\langle q^\nu,(u',\lambda)\rangle)] = \xi_1[\rho \leftarrow \langle q',(u',\lambda)\rangle]\Phi = \xi_2\Phi$. □

Now, from Theorem 10 together with Lemma 34 and Theorem 31 we obtain our second main result: every n-ptt can be simulated by the composition of $n+1$ stay-mtts (mtts in the deterministic case). Note that, as for Theorem 10, the first n translations are realized by (very simple) total deterministic mtts: they all realize EncPeb $\in \mathrm{D_tMTT}$.

Theorem 35. For every $n \geq 1$, n-PTT $\subseteq$ sMTT^{n+1} and n-DPTT $\subseteq$ DMTT^{n+1}.

By Theorem 29 and Lemma 27 the nondeterministic part of this theorem implies that n-PTT $\subseteq$ MON $\circ$ MTT^{n+1}. The deterministic part of Theorem 35 is, in fact, optimal, i.e., n-DPTT is *not* included in DMTT^n. This will follow immediately from Theorem 41 in Section 6.

5.5 Simulation of Macro Tree Transducers by PTTs

In the previous subsection it was shown how to simulate n-ptts by compositions of stay-mtts, and by compositions of dmtts in the deterministic case. Now we show the converse direction, namely, how to simulate a stay-mtt by a composition of 0-ptts, and a deterministic mtt by a composition of 0-dptts. This result, together with the converse simulation of the previous subsection, proves that ptts and stay-mtts have the same composition closure (and that dptts and dmtts have the same composition closure). Hence, the classes of output languages of compositions of ptts and of mtts coincide.

Recall that by Lemma 27, sMTT $\subseteq$ MON $\circ$ MTT. Since MON $\subseteq$ 0-PTT by Example 6, this means that sMTT $\subseteq$ 0-PTT$\circ$MTT. Thus, it will suffice to consider the simulation of mtts by ptts.

In order to prove that an mtt can be simulated by compositions of ptts, we use a well-known decomposition result of (total deterministic) mtts into compositions of top-down tree transducers and so-called "YIELD mappings" (see, e.g., [ES77,Eng80]). Recall from Definition 22 that a top-down tree transducer is an mtt M *without* parameters, i.e., with each state of rank zero. The configurations of a top-down tree transducer are always at the leaves of a sentential form, in contrast to an mtt whose configurations can also be at non-leaf nodes of a sentential form. This means that a top-down tree transducer can simulate the state behavior of an mtt, but only at the leaves of its sentential forms, because it cannot carry out the second-order tree substitution inherent in a computation step of an mtt (viz. applying a rule to a configuration of rank > 0). Now, YIELD mappings carry out second-order tree substitution. Altogether, a total deterministic mtt M can be simulated by first running a (total deterministic) top-down tree transducer that realizes M's state behavior and generates a special intermediate tree, and then applying a YIELD mapping to that tree (realizing the second-order tree substitution inherent in M's computation). In other words, $\mathrm{D_tMTT} \subseteq \mathrm{D_tT} \circ \mathrm{YIELD}$ (Proposition 4.17 of [CF82]; cf. also Theorem 4.8 of [EV85]). Partialness and nondeterminism of an mtt can be handled by post-composing a total deterministic mtt with a corresponding top-down tree transducer (Corollary 6.12 of [EV85]), i.e., $\mathrm{MTT} \subseteq \mathrm{D_tMTT} \circ \mathrm{T}$ and $\mathrm{DMTT} \subseteq \mathrm{D_tMTT} \circ \mathrm{DT}$. Thus, we obtain (cf. also Theorem 7.3 of [EV85])

$$\mathrm{MTT} \subseteq (\mathrm{T} \cup \mathrm{YIELD})^3 \quad \text{and} \quad \mathrm{DMTT} \subseteq (\mathrm{DT} \cup \mathrm{YIELD})^3.$$

As stated in Lemma 23, top-down tree transducers can be realized by 0-ptts; we now prove, in Lemma 36, that YIELD mappings can be realized by 0-ptts. For attribute grammars (see Subsection 3.2) these results are well known: top-down tree transducers can be simulated by attribute grammars [CF82] and so can YIELD mappings (shown in Theorem 1.3 of [Eng81], without correctness proof, and in Corollary 6.24 of [FV98], using an indirect proof). Together with the above decomposition result this will allow us to prove the equality of the composition closure of ptts and stay-mtts, in Theorem 38.

Let us now define YIELD mappings and show that they can be realized by 0-ptts. A YIELD mapping Y_f is a mapping from T_Σ to $T_\Delta(Y)$ defined by a mapping f from $\Sigma^{(0)}$ to $T_\Delta(Y)$, for ranked alphabets Σ and Δ. It realizes the semantics of first-order tree substitution in the following way:

(i) for $\alpha \in \Sigma^{(0)}$, $Y_f(\alpha) = f(\alpha)$ and
(ii) for $\sigma \in \Sigma^{(k)}$, $s_1, \ldots, s_k \in T_\Sigma$, and $k \geq 1$,
$Y_f(\sigma(s_1, \ldots, s_k)) = Y_f(s_1)[y_\mu \leftarrow Y_f(s_{\mu+1}) \mid \mu \in [k-1]]$.

The class of all YIELD mappings is denoted by YIELD.

Intuitively, to compute the tree $Y_f(s)$ for some $s = \sigma(s_1, \ldots, s_k) \in T_\Sigma$, the mapping Y_f has to be applied to the first subtree s_1, and in the resulting tree each parameter y_μ, $\mu \in [k-1]$, has to be replaced by Y_f applied to the $(\mu+1)$th subtree $s_{\mu+1}$. Note that if f is a mapping from $\Sigma^{(0)}$ to $T_\Delta(Y_m)$, $m \geq 0$, then Y_f is a mapping from T_Σ to $T_\Delta(Y_m)$.

As a small example of a YIELD mapping, consider the ranked alphabet Σ with $\Sigma^{(0)} = \{a, b, c\}$, $\Sigma^{(2)} = \{\sigma\}$ and the mapping f from $\Sigma^{(0)}$ to $T_\Delta(Y_1)$ with $\Delta = \{a^{(1)}, b^{(1)}, c^{(1)}, e^{(0)}\}$, and $f(a) = a(y_1)$, $f(b) = b(y_1)$, and $f(c) = c(e)$. Now let $s = \sigma(a, \sigma(b, c))$. Then $Y_f(s)$ is a (monadic tree) representation of the yield abc of the tree s, namely,

$$\begin{aligned} Y_f(s) &= f(a)[y_1 \leftarrow Y_f(\sigma(b,c))] \\ &= a(y_1)[y_1 \leftarrow \underbrace{f(b)[y_1 \leftarrow c(e)]}_{b(c(e))}] \\ &= a(b(c(e))). \end{aligned}$$

Note that, in general, a YIELD mapping Y_f is realized by a dmtt M_f with one state q and rules

$$\begin{aligned} \langle q, \alpha, \lambda, j\rangle(y_1, \ldots, y_m) &\to f(\alpha) \\ \langle q, \sigma, \lambda, j\rangle(y_1, \ldots, y_m) &\to \langle q, \mathrm{down}_1\rangle(\langle q, \mathrm{down}_2\rangle(y_1, \ldots, y_m), \ldots, \\ &\qquad \langle q, \mathrm{down}_k\rangle(y_1, \ldots, y_m), y_k, \ldots, y_m), \end{aligned}$$

where f is a mapping from $\Sigma^{(0)}$ to $T_\Delta(Y_m)$, and $y_k, \ldots, y_m$ is empty if $m < k$.

We now show that YIELD mappings can be realized by 0-dptts.

Lemma 36. YIELD $\subseteq$ 0-DPTT.

Proof. Let Σ and Δ be ranked alphabets, $J = \max\{\mathrm{rank}_\Sigma(\sigma) \mid \sigma \in \Sigma\}$, $m \geq 0$, and let f be a mapping from $\Sigma^{(0)}$ to $T_\Delta(Y_m)$. We now define the deterministic 0-ptt $M = (\Sigma, \Delta \cup \{y_\mu^{(0)} \mid \mu \in [m]\}, Q, q, R)$ such that $\tau_M = Y_f$. Let $Q =$

$\{q, q_1, \ldots, q_m, q'_1, \ldots, q'_m\}$. For the state q, let the following rules be in R.

$$\begin{array}{ll}\langle q, \sigma, \lambda, j\rangle \to \langle q, \mathrm{down}_1\rangle & \text{for } \sigma \in \Sigma^{(k)}, k \geq 1, \text{ and } j \in [0, J] \\ \langle q, \alpha, \lambda, j\rangle \to f(\alpha)[y_\mu \leftarrow \langle q_\mu, \mathrm{stay}\rangle \mid \mu \in [m]] & \text{for } \alpha \in \Sigma^{(0)} \text{ and } j \in [0, J].\end{array}$$

Intuitively, starting in a configuration $\langle q, (u, \lambda)\rangle$, M will compute the tree $Y_f(s/u)$ when restricted to input configurations (v, λ) where v is a descendant of u, i.e., $v = uv'$ with $v' \in V(s/u)$. However, in place of a parameter y_μ this tree will have a configuration $\langle q_\mu, (u, \lambda)\rangle$; such a configuration computes the actual parameter tree which should replace y_μ during M's computation of $Y_f(s)$. For every $\mu \in [m]$ let the rules

$$\begin{array}{ll}\langle q_\mu, \sigma, \lambda, 1\rangle \to \langle q'_\mu, \mathrm{up}\rangle & \text{for } \sigma \in \Sigma \\ \langle q_\mu, \sigma, \lambda, j\rangle \to \langle q_\mu, \mathrm{up}\rangle & \text{for } \sigma \in \Sigma \text{ and } j \in [2, J] \\ \langle q_\mu, \sigma, \lambda, 0\rangle \to y_\mu & \text{for } \sigma \in \Sigma\end{array}$$

$$\begin{array}{ll}\langle q'_\mu, \sigma, \lambda, j\rangle \to \langle q, \mathrm{down}_{\mu+1}\rangle & \text{for } \sigma \in \Sigma^{(k)}, \mu+1 \leq k, \text{ and } j \in [0, J] \\ \langle q'_\mu, \sigma, \lambda, j\rangle \to \langle q_\mu, \mathrm{stay}\rangle & \text{for } \sigma \in \Sigma^{(k)}, \mu+1 > k, \text{ and } j \in [0, J]\end{array}$$

be in R. Intuitively, in a configuration $\langle q_\mu, (u, \lambda)\rangle$, M computes for y_μ the actual parameter tree at u, which is the $(\mu+1)$th child of u's parent u' if u is a first child and u' has a $(\mu+1)$th child, and otherwise is the actual parameter tree at u' (cf. the rules of the dmtt M_f shown below the definition of YIELD).

We now prove the correctness of the construction of M. Let $s \in T_\Sigma$. It must be shown that $\tau_M(s) = Y_f(s)$. In what follows, let $\Rightarrow_{M,s}$ be denoted by $\Rightarrow$. By the claim below, $\langle q, h_0\rangle \Rightarrow^* Y_f(s)[y_\mu \leftarrow \langle q_\mu, h_0\rangle \mid \mu \in [m]] = \xi$. Since $\langle q_\mu, h_0\rangle \Rightarrow y_\mu$ for every $\mu \in [m]$, $\xi \Rightarrow^* Y_f(s)[y_\mu \leftarrow y_\mu \mid \mu \in [m]] = Y_f(s)$. Thus, $\langle q, h_0\rangle \Rightarrow^* Y_f(s)$.

In the remainder of this proof we will write $\langle q, u\rangle$ instead of $\langle q, (u, \lambda)\rangle$.

<u>*Claim:*</u> For every $u \in V(s)$, $\langle q, u\rangle \Rightarrow^* Y_f(s/u)[y_\mu \leftarrow \langle q_\mu, u\rangle \mid \mu \in [m]]$.

The proof of the claim is by induction on the size of s/u.

Case 1, u is a leaf: Let $\alpha = s[u]$. Then $\langle q, u\rangle \Rightarrow f(\alpha)[y_\mu \leftarrow \langle q_\mu, u\rangle \mid \mu \in [m]]$. By the definition of Y_f, $f(\alpha) = Y_f(\alpha)$, and, since u is a leaf, $Y_f(\alpha) = Y_f(s/u)$, which proves the claim for this case.

Case 2, u is not a leaf: By the definition of the q-rule of M for symbols of positive rank, $\langle q, u\rangle \Rightarrow \langle q, u1\rangle$. By induction

$$\langle q, u1\rangle \Rightarrow^* Y_f(s/u1)[y_\mu \leftarrow \langle q_\mu, u1\rangle \mid \mu \in [m]] = \xi.$$

What M computes in a configuration $\langle q_\mu, u1\rangle$ depends on the numbers $\mu+1$ and k, where k is the rank of $s[u]$: If $\mu + 1 > k$ then

$$\langle q_\mu, u1\rangle \Rightarrow \langle q'_\mu, u\rangle \Rightarrow \langle q_\mu, u\rangle,$$

and if $\mu + 1 \leq k$ then

$$\begin{array}{rll}\langle q_\mu, u1\rangle \Rightarrow & \langle q'_\mu, u\rangle & \\ \Rightarrow & \langle q, u(\mu+1)\rangle & \\ \Rightarrow^* & Y_f(s/u(\mu+1))[y_\nu \leftarrow \langle q_\nu, u(\mu+1)\rangle \mid \nu \in [m]] & \text{(by induction)} \\ \Rightarrow^* & Y_f(s/u(\mu+1))[y_\nu \leftarrow \langle q_\nu, u\rangle \mid \nu \in [m]] & \text{(because } \mu+1 \geq 2).\end{array}$$

Thus, there is a computation starting from ξ (displayed above), of the form

$$\begin{array}{l}\xi \Rightarrow^* Y_f(s/u1)\ [y_\mu \leftarrow Y_f(s/u(\mu+1))\Psi \mid \mu \in [m], \mu+1 \leq k] \\ \qquad\qquad\qquad [y_\mu \leftarrow \langle q_\mu, u\rangle \mid \mu \in [m], \mu+1 > k],\end{array}$$

where $\Psi = [y_\nu \leftarrow \langle q_\nu, u\rangle \mid \nu \in [m]]$. This is equal to

$$Y_f(s/u1)[y_\mu \leftarrow Y_f(s/u(\mu+1)) \mid \mu \in [m], \mu+1 \leq k]\Psi.$$

Since "$\mu \in [m], \mu + 1 \leq k$" means the same as "$\mu \in [k-1]$" we obtain, by the definition of Y_f, that the above equals $Y_f(s/u)\Psi$. This concludes the proof of the claim and of the lemma. □

Consider again the example of a YIELD mapping Y_f given above the previous lemma and the tree $s = \sigma(a, \sigma(b, c))$. Let M be the 0-dptt obtained by the construction in the proof of the previous lemma (and let $\Rightarrow$ denote $\Rightarrow_{M,s}$). Then

$$\langle q, \varepsilon\rangle \Rightarrow \langle q, 1\rangle \Rightarrow a(\langle q_1, 1\rangle) \Rightarrow a(\langle q_1', \varepsilon\rangle) \Rightarrow a(\langle q, 2\rangle) \Rightarrow$$
$$a(\langle q, 21\rangle) \Rightarrow a(b(\langle q_1, 21\rangle)) \Rightarrow a(b(\langle q_1', 2\rangle)) \Rightarrow a(b(\langle q, 22\rangle)) \Rightarrow a(b(c(e))),$$

which is the correct tree $Y_f(s)$, as shown in the example.

Lemma 37. sMTT $\subseteq$ 0-PTT^4 and DMTT $\subseteq$ 0-DPTT^3.

Proof. By Lemma 27, sMTT $\subseteq$ MON $\circ$ MTT which is included in 0-PTT $\circ$ MTT by Example 6. As mentioned above, MTT $\subseteq$ (T $\cup$ YIELD)^3 which is included in 0-PTT^3 by Lemmas 23 and 36. Hence sMTT $\subseteq$ 0-PTT^4. In the deterministic case, DMTT $\subseteq$ (DT$\cup$YIELD)^3 which is included in 0-DPTT^3 by Lemmas 23 and 36. □

It was proved at the end of Section 4 that the composition closure of n-ptts equals the one of 0-ptts, i.e., PTT* = 0-PTT* (and DPTT* = 0-DPTT* in the deterministic case). We are now ready to prove our third main result, namely, that these classes equal the composition closure of stay-mtts (and of dmtts in the deterministic case).

Theorem 38. PTT* = 0-PTT* = sMTT* and
DPTT* = 0-DPTT* = DMTT*.

Proof. By Corollary 11, PTT* = 0-PTT* and DPTT* = 0-DPTT*. We now show that 0-PTT* = sMTT* and 0-DPTT* = DMTT*. By Theorem 35 and Lemma 37, 0-PTT $\subseteq$ sMTT $\subseteq$ 0-PTT* and 0-DPTT $\subseteq$ DMTT $\subseteq$ 0-DPTT*. This implies the required equalities. □

In terms of databases Theorem 38 shows that, as query languages, ptts and mtts have the same expressiveness. For total functions, it was already known that total deterministic macro tree transducers and (noncircular) attributed tree transducers have the same composition closure (see Chapter 6 of [FV98]).

In the deterministic case, we have also proved that DPTT* $\subseteq$ 0-DPTT$^*_{\text{term}}$, where the latter is the class of translations realized by 0-dptts that have no infinite computations, i.e., they are *term*inating: first simulate the dptts by (compositions of) dmtts, then decompose the dmtts into (deterministic) top-down tree transducers and YIELD mappings following the results in [EV85], and finally simulate those by (compositions of) 0-ptts, using Lemmas 23 and 36, respectively (obviously, the constructions in the proofs of these two lemmas give terminating 0-dptts; in fact, they are even noncircular, see Subsection 3.2). Note that it is not clear whether or not infinite computations can be removed directly from an n-dptt, i.e., whether or not DPTT $\subseteq$ DPTT$_{\text{term}}$.

In [MSV00] it is stated as an open problem whether PTT contains all bottom-up tree translations (denoted B, and DB in the deterministic case). Note that we obtain from Lemmas 23, 36, and 37 that DB $\subseteq$ 0-DPTT^3 and B $\subseteq$ 0-PTT^2 because DB $\subseteq$ DMTT (Corollary 6.16 of [EV85]) and B $\subseteq$ T $\circ$ YIELD (Theorem 5.16 and Lemma 5.5 of [EV85]).

If we consider the class of output languages of PTT*, then by the previous theorem, PTT*(REGT) = sMTT*(REGT) and by Theorem 30 this equals MTT*(REGT). Thus, PTT* and MTT* define the same class of output languages.

Corollary 39. $\mathrm{PTT}^*(\mathrm{REGT}) = \mathrm{MTT}^*(\mathrm{REGT})$.

As stated in Fact 25, emptiness and finiteness of tree languages in $\mathrm{MTT}^*(\mathrm{REGT})$ are decidable. In Section 7 on type checking we will use these facts to show that "type checking" and "almost always type checking" are decidable for languages in $\mathrm{MTT}^*(\mathrm{REGT})$. Through Corollary 39 this provides an alternative proof of the main result of [MSV00] that type checking is decidable for languages in $\mathrm{PTT}^*(\mathrm{REGT})$.

6 Pebble Hierarchies for Deterministic PTTs

In this section we consider for deterministic pebble tree transducers the following question: Is the (deterministic) pebble tree transducer with $n+1$ pebbles more powerful than the one with n pebbles? As the power of the pebble tree transducer we consider its ability

(i) to translate,
(ii) to generate output tree languages, and
(iii) to generate output string languages.

The first two aspects are important for database theory (translations are queries, and output tree languages are views) and the third one is mainly of interest for formal language theory. Note that an *output string language* is obtained from an output tree language by taking the yields of its trees; thus, it is of the form $y\tau_M(R) = \{yt \mid (s,t) \in \tau_M \text{ for some } s \in R\}$ where M is a pebble tree transducer and $R \in$ REGT.

In Section 3 it was shown already that the number n of pebbles gives rise to a proper hierarchy of translations; in other words, with respect to (i), the dptt with $n+1$ pebbles is strictly more powerful than the one with n pebbles. In this section we show that also with respect to (iii), and hence also (ii), $n+1$ pebbles are strictly more powerful than n. More precisely, for the classes $y(n\text{-DPTT}(\mathrm{REGT}))$ of output string languages of n-dptts, there is a proper hierarchy with respect to n, i.e., $y((n-1)\text{-DPTT}(\mathrm{REGT})) \subsetneq y(n\text{-DPTT}(\mathrm{REGT}))$ for all $n \geq 1$. We call this the "dptt-hierarchy".

Recall from Theorem 35 that $(n-1)\text{-DPTT} \subseteq \mathrm{DMTT}^n$. The properness of the dptt-hierarchy will be proved using a 'bridge theorem' for the classes $y\mathrm{D_tMTT}^n($ REGT), viz. Theorem 18 of [EM02a]. This bridge theorem provides a method to obtain languages that are *not* in $y\mathrm{D_tMTT}^n(\mathrm{REGT})$. In [EM02a] it was used to prove that the "(total) dmtt-hierarchy" is proper, i.e., $y\mathrm{D_tMTT}^n(\mathrm{REGT}) \subsetneq y\mathrm{D_tMTT}^{n+1}(\mathrm{REGT})$: Theorem 23 of [EM02a]. Here we will use it to show that $y(n\text{-DPTT}(\mathrm{REGT}))$ contains languages not in $y\mathrm{DMTT}^n(\mathrm{REGT})$, and hence not in $y((n-1)\text{-DPTT}(\mathrm{REGT}))$. Since the dptt-hierarchy involves non-total functions, we first prove that totality is irrelevant for output languages of DMTT^n, i.e., that for every $n \geq 1$,

$$y\mathrm{DMTT}^n(\mathrm{REGT}) = y\mathrm{D_tMTT}^n(\mathrm{REGT}). \qquad (*)$$

We show, by induction on n, that $\mathrm{DMTT}^n(\mathrm{REGT}) = \mathrm{D_tMTT}^n(\mathrm{REGT})$, i.e., even the corresponding classes of tree languages coincide. The proof is based on Theorem 6.18 of [EV85] which says that $\mathrm{DMTT} = \mathrm{FTA} \circ \mathrm{D_tMTT}$, where FTA is the class of identity functions restricted to regular tree languages, i.e., applying a function in FTA is the same as taking the intersection with a regular tree language. For $n = 1$ this implies that $\mathrm{DMTT}(\mathrm{REGT}) = \mathrm{D_tMTT}(\mathrm{FTA}(\mathrm{REGT}))$. Since regular tree languages are closed under intersection (cf., e.g., [GS84]), $\mathrm{FTA}(\mathrm{REGT}) =$ REGT and hence $\mathrm{D_tMTT}(\mathrm{FTA}(\mathrm{REGT})) = \mathrm{D_tMTT}(\mathrm{REGT})$. For $n+1$, it follows from Theorem 6.18 of [EV85] and by induction that $\mathrm{DMTT}^{n+1}(\mathrm{REGT}) = \mathrm{D_tMTT}(\mathrm{FTA}(\mathrm{D_tMTT}^n(\mathrm{REGT})))$. Now $\mathrm{FTA}(\mathrm{D_tMTT}^n(\mathrm{REGT}))$ equals $\mathrm{D_tMTT}^n($

REGT): for $R_{\text{in}}, R_{\text{out}} \in$ REGT and $\tau \in \mathrm{D_tMTT}^n$, $\tau(R_{\text{in}}) \cap R_{\text{out}} = \tau(R_{\text{in}} \cap \tau^{-1}(R_{\text{out}}))$ and $R_{\text{in}} \cap \tau^{-1}(R_{\text{out}})$ is in REGT by Fact 24 and the fact that REGT is closed under intersection.

Let us now state the bridge theorem of [EM02a], in terms of non-total dmtts. To do this we first define the notion of δ-completeness. Let A and B be disjoint alphabets. Consider a string w of the form

$$w = w_1 a_1 w_2 a_2 \cdots a_{l-1} w_l a_l w_{l+1}$$

with $l \geq 0$, $a_1, \ldots, a_l \in A$, and $w_1, \ldots, w_{l+1} \in B^*$. Define the string $\mathrm{res}_A(w) \in A^*$ as $a_1 \cdots a_l$. If all $w_2, \ldots, w_l$ are pairwise different, then w is a *δ-string for* $a_1 \cdots a_l$. Let $L \subseteq A^*$ and $L' \subseteq (A \cup B)^*$. If L' contains, for every $w \in L$, a δ-string for w, then L' is called *δ-complete for L.*

Lemma 40. (Theorem 18 of [EM02a]) Let A and B be disjoint alphabets, and let $L \subseteq A^*$ and $L' \subseteq (A \cup B)^*$ be languages such that L' is δ-complete for L and $\mathrm{res}_A(L') = L$.

(a) For every $n \geq 1$, if $L' \in y\mathrm{DMTT}^{n+1}(\mathrm{REGT})$, then $L \in y\mathrm{DMTT}^n(\mathrm{REGT})$.
(b) If $L' \in y\mathrm{DMTT}(\mathrm{REGT})$, then $L \in y\mathrm{DT}(\mathrm{REGT})$.

The next theorem (Theorem 41, which is the main result of this section) proves that there is an n-dptt that generates an output string language which is not in $y\mathrm{DMTT}^n(\mathrm{REGT})$. Recall from Definition 1 that an n-ptt is monadic if its input and output alphabets Σ and Δ are monadic, and that the corresponding string-to-string translations are those realized by two-way n-pebble string transducers. The first part of the proof of Theorem 41 was already presented in (the proof of) Theorem 5 of [EM02b]: with one pebble more, there is a monadic $(n+1)$-dptt that generates an output language which is not in $y\mathrm{DMTT}^n(\mathrm{REGT})$ when viewed as a string language (through paths). Together with the fact that the output languages of monadic n-dptts (viewed as string languages) are output string languages of n-fold compositions of total deterministic mtts (Theorem 4 of [EM02b]) this proves that the output tree languages of monadic n-dptts form a proper hierarchy with respect to the number n of pebbles: the "pebble string transducer hierarchy" (Theorem 5 of [EM02b]). The second part of the proof of Theorem 41 shows that without the monadic restriction the extra pebble is not needed.

Note that, in terms of the translations, this result implies immediately that n-DPTT $\not\subseteq \mathrm{DMTT}^n$, which cannot be proved using size-height properties of translations of dmtts. Thus, the inclusion n-DPTT $\subseteq \mathrm{DMTT}^{n+1}$ of Theorem 35 is optimal.

Theorem 41. For every $n \geq 1$, $y(n\text{-DPTT}(\mathrm{REGT})) - y\mathrm{DMTT}^n(\mathrm{REGT}) \neq \varnothing$.

Proof. The inequality will be proved using the 'bridge theorem' Lemma 40. First, let $n = 1$ and let $\Sigma = \{a^{(1)}, e^{(0)}\}$. It is well known that the language $K = \{(a^m b)^m \mid m \geq 0\}$ is not in $y\mathrm{DT}(\mathrm{REGT})$ (see Theorem 3.16 of [Eng82]). This means, by Lemma 40(b), that K can be used in order to construct languages K' not in $y\mathrm{DMTT}(\mathrm{REGT})$.

Before it is shown how to obtain a 1-dptt such that the yield of its output language is such a K', we show how to construct a monadic 1-dptt M_K with $p\tau_{M_K}(T_\Sigma) = \{(a^m b)^m \mid m \geq 0\} = K$ (recall the definition of p from Subsection 2.1, e.g., for the tree $s = a(a(b(e)))$, ps is the string aab). The idea of M_K is straightforward: M_K uses the pebble as a counter to make m copies of the input tree $a^m(e)$. On input tree $a^m e$, it drops the pebble at the root node, copies the input tree top-down, replacing the e by b, thus generating $a^m b$ as output. Then it searches the pebble, moves it one node down, and then again generates another copy of $a^m b$. This is repeated until the pebble has reached the leaf of the input tree, thus generating the monadic tree $(a^m b)^m e$. It should be obvious how to define the rules of M_K.

Given a monadic n-dptt M (with arbitrary input and output alphabets Σ and Δ, respectively) we will construct below the

– n-dptt split(M) and the
– monadic $(n+1)$-dptt conf(M)

(with the same input alphabet Σ) such that for $L = p\tau_M(T_\Sigma)$ and $A = \Delta^{(1)}$:

– $L' = y\tau_{\text{split}(M)}(T_\Sigma)$ is δ-complete for L and
– $L' = p\tau_{\text{conf}(M)}(T_\Sigma)$ is δ-complete for L,

and $\text{res}_A(L') = L$ in both cases, which will be proved in Claims 2 and 1, respectively.

Let now, again, $n = 1$ and $\Sigma = \{a^{(1)}, e^{(0)}\}$. Consider the 1-dptt split(M_K) obtained from the monadic 1-dptt M_K of above. Then $K' = y\tau_{\text{split}(M_K)}(T_\Sigma)$ is δ-complete for $p\tau_{M_K}(T_\Sigma) = K$ and $\text{res}_A(K') = K$, where $A = \Delta^{(1)}$ and Δ is the output alphabet of M_K. Since $K \notin y\text{DT}(\text{REGT})$ we apply Lemma 40(b) in order to "bridge" K' out of the class $y\text{DMTT}(\text{REGT})$; we obtain that $K' \notin y\text{DMTT}(\text{REGT})$ which proves the theorem for $n = 1$ (because $T_\Sigma \in \text{REGT}$).

Now let $n > 1$. Define inductively the monadic n-dptt $N_n = \text{conf}(N_{n-1})$ and $N_1 = M_K$. We will prove by induction on n that

$$p\tau_{N_n}(T_\Sigma) \notin y\text{DMTT}^{n-1}(\text{REGT}).$$

As stated above, $L' = p\tau_{N_n}(T_\Sigma)$ is δ-complete for $L = p\tau_{N_{n-1}}(T_\Sigma)$ and $\text{res}_A(L') = L$, where $A = \Delta^{(1)}$ and Δ is the output alphabet of N_{n-1}.

For $n = 2$, $L = K \notin y\text{DT}(\text{REGT})$ which implies by Lemma 40(b) that $L' \notin y\text{DMTT}(\text{REGT})$.

For $n > 2$ assume that $L = p\tau_{N_{n-1}}(T_\Sigma) \notin y\text{DMTT}^{n-2}(\text{REGT})$. Then $L' = p\tau_{N_n}(T_\Sigma)$ is not in $y\text{DMTT}^{n-1}(\text{REGT})$ by Lemma 40(a).

Note that, since monadic n-ptts are the same as n-pebble string transducers, $p\tau_{N_n}(T_\Sigma)$ is an output language of an n-pebble string transducer; as mentioned above the theorem, the fact that $p\tau_{N_n}(T_\Sigma) \notin y\text{DMTT}^{n-1}(\text{REGT})$ was used in Theorem 5 of [EM02b] to prove the properness of the pebble string transducer hierarchy.

We now apply the construction split to N_n, to obtain the n-dptt split(N_n). Take $L = p\tau_{N_n}(T_\Sigma)$ and $L' = y\tau_{\text{split}(N_n)}(T_\Sigma)$. Then, by the above, L' is δ-complete for L and $\text{res}_A(L') = L$, where $A = \Delta^{(1)}$ and Δ is the output alphabet of N_n. Hence, since $L \notin y\text{DMTT}^{n-1}(\text{REGT})$ by the inductive proof above, we obtain from Lemma 40(a) that $L' \notin y\text{DMTT}^n(\text{REGT})$. Since $L' \in y(n\text{-DPTT}(\text{REGT}))$, this proves the theorem.

Let $M = (\Sigma, \Delta, Q, q_0, R)$ be an arbitrary monadic n-dptt, $n \geq 1$. In what follows, we construct the monadic $(n+1)$-dptt conf(M) and the n-dptt split(M), and prove in Claims 1 and 2 that their corresponding output languages are δ-complete for $p\tau_M(T_\Sigma)$. First we define the monadic $(n+1)$-dptt conf(M): The construction of conf(M) is similar to the construction of M_{n+1} in Example 5. The idea is that conf(M) simulates M, and additionally inserts above each unary symbol of the output tree of M a coding w_c of the current configuration $c \in C_{M,s}$. This coding is obtained as follows. If the current configuration is $c = \langle q, (u, \pi)\rangle$, then conf$(M)$ first moves from u up to the root (in state q_{up}). From there (in state q_{down}) it moves to the leaf of the input tree s, outputting at each node v the symbol (q, σ, b), where σ is the label of v and b is the information on the pebbles at v. After this, conf(M) needs to move back to the node u to continue the computation of M. This is done by dropping, before the coding is generated, an extra pebble at u. After the coding is generated, conf(M) changes into the state q_{find} and moves to the node with the most recently dropped pebble, i.e., to u. Note that the symbol (q, σ, b) generated by conf(M) at v also contains the information about the position of the reading head: if b indicates that the most recently dropped pebble is present, then the reading head is at v (i.e., $v = u$). It should now be clear that w_c is indeed a coding of c, i.e., $w_c \neq w_{c'}$ for $c \neq c'$.

Define conf$(M) = (\Sigma, \Gamma, Q', q_0, R')$ with

– $\Gamma = \Delta \cup \{(q, \sigma, b)^{(1)} \mid q \in Q, \sigma \in \Sigma, b \in \{0,1\}^{\leq n+1}\}$
– $Q' = Q \cup \{q_c \mid q \in Q, c \in \{\text{up}, \text{down}, \text{find}, \text{back}\}\}$
– For every rule $r = (\langle q, \sigma, b, j\rangle \to \zeta)$ in R: if $\zeta \in \langle Q, I_{\sigma,b,j}\rangle$ or $\zeta = e$, then let r be in R'; if $\zeta = a(q')$ with $a \in \Delta^{(1)}$ and $q' \in Q$, then let the rules

$$\begin{aligned}\langle q, \sigma, b, j\rangle &\to \langle q_{\text{up}}, \text{drop}\rangle\\ \langle q_{\text{back}}, \sigma, b, j\rangle &\to a(q')\end{aligned}$$

be in R'. For every $q \in Q$, $b \in \{0,1\}^{\leq n+1}$, and $b' \in \{0,1\}^{\leq n}$ let the following rules be in R':

$$\begin{array}{lll}\langle q_{\text{up}}, \sigma, b, 1\rangle & \to \langle q_{\text{up}}, \text{up}\rangle & \text{for } \sigma \in \Sigma\\ \langle q_{\text{up}}, \sigma, b, 0\rangle & \to \langle q_{\text{down}}, \text{stay}\rangle & \text{for } \sigma \in \Sigma\\ \langle q_{\text{down}}, \sigma, b, j\rangle & \to (q, \sigma, b)(\langle q_{\text{down}}, \text{down}_1\rangle) & \text{for } \sigma \in \Sigma^{(1)}, j \in \{0,1\}\\ \langle q_{\text{down}}, e, b, j\rangle & \to (q, e, b)(\langle q_{\text{find}}, \text{stay}\rangle) & \text{for } j \in \{0,1\}\\ \langle q_{\text{find}}, \sigma, b'0, 1\rangle & \to \langle q_{\text{find}}, \text{up}\rangle & \text{for } \sigma \in \Sigma\\ \langle q_{\text{find}}, \sigma, b'1, j\rangle & \to \langle q_{\text{back}}, \text{lift}\rangle & \text{for } \sigma \in \Sigma, j \in \{0,1\}.\end{array}$$

Note that Γ should be defined in such a way that the set $\{(q, \sigma, b)^{(1)} \mid q \in Q, \sigma \in \Sigma, b \in \{0,1\}^{\leq n+1}\}$ is disjoint with Δ. In that way we will be able to apply Lemma 40 for $A = \Delta^{(1)}$ and $B = \Gamma - \Delta$.

Clearly, $\tau_{\text{conf}(M)} \in (n+1)$-DPTT and $\text{res}_{\Delta^{(1)}}(p\tau_{\text{conf}(M)}(T_\Sigma)) = p\tau_M(T_\Sigma)$.

Claim 1: Let M be a monadic n-dptt with input ranked alphabet Σ. Then $p\tau_{\text{conf}(M)}(T_\Sigma)$ is δ-complete for $p\tau_M(T_\Sigma)$.

Let M' denote conf(M). Since both M and M' are monadic, we will drop the parentheses and the symbol e when we show computations. It has to be shown that for every $w \in L = p\tau_M(T_\Sigma)$ there is a $w' \in L' = p\tau_{M'}(T_\Sigma)$ such that w' is a δ-string for w. Let $s \in T_\Sigma$. If $w = p\tau_M(s)$ is defined, then there is a computation

$$\begin{aligned}\langle q_0, h_0\rangle = c_0 \Rightarrow^*_{M,s} d_0 \Rightarrow_{M,s} a_0c_1 \Rightarrow^*_{M,s} a_0d_1 \Rightarrow_{M,s} a_0a_1c_2 \Rightarrow^*_{M,s} \cdots\\ \Rightarrow^*_{M,s} a_0 \cdots a_{m-1}d_m \Rightarrow_{M,s} a_0 \cdots a_mc_{m+1} \Rightarrow^*_{M,s} a_0 \cdots a_m = w,\end{aligned}$$

where $a_0, \ldots, a_m \in \Delta^{(1)}$ and $c_0, d_0, \ldots, c_m, d_m, c_{m+1} \in C_{M,s}$. Then, all configurations $d_0, \ldots, d_m$ are pairwise different because M is deterministic. Take $w' = \tau_{M'}(s)$. Now, if $c \Rightarrow_{M,s} d$ then $c \Rightarrow_{M',s} d$, because M' has the same rules as M for right-hand sides that do not contain an output symbol. If $d \Rightarrow_{M,s} ac$, then $d \Rightarrow^*_{M',s} w_dac$ where w_d is the coding of d discussed above. Applied to the computation of w by $\Rightarrow_{M,s}$, we obtain

$$\begin{aligned}c_0 \Rightarrow^*_{M',s} d_0 \Rightarrow^*_{M',s} w_{d_0}a_0c_1 \Rightarrow^*_{M',s} w_{d_0}a_0d_1 \Rightarrow^*_{M',s} w_{d_0}a_0w_{d_1}a_1c_2 \Rightarrow^*_{M',s} \cdots\\ \Rightarrow^*_{M',s} w_{d_0}a_0w_{d_1}a_1 \cdots w_{d_m}a_mc_{m+1} \Rightarrow^*_{M',s} w_{d_0}a_0w_{d_1}a_1 \cdots w_{d_m}a_m = w'.\end{aligned}$$

All the strings w_{d_i} are pairwise different because the d_i are. This implies that w' is a δ-string for w, which ends the proof of Claim 1.

Second, we construct the n-dptt split(M). The idea of split(M) is as follows: just as conf(M), split(M) simulates M and additionally outputs before each unary output symbol generated by M a coding of the current configuration that M is in. However, this time we do not want to use an extra pebble to do this, but rather generate the corresponding string as yield and use split(M)'s ability to generate non-monadic output in order to 'split' the computation, i.e., to initiate parallel computations (which start with the same input configurations) in order to insert the current configuration of M before an output symbol. More precisely, if M, in configuration c, outputs a unary symbol a and changes into configuration c', then split(M), in c, computes a tree with yield $w_c\, a\, c'$, where w_c is a coding of the configuration c. The coding is similar to the one of conf(M): split(M) moves from the current node u up to the root of the input tree s and then down to the leaf of s, outputting at every node v a symbol (q, σ, b), where $c = \langle q, h\rangle$ for some h, σ is the label of v, and b is the information of the pebbles at v. It should be clear that in this way the current configuration is coded in a unique way. Note that split(M) also produces output when moving up; this takes care of the coding of the position u of the reading head. Thus w_c is a coding of c, i.e., $w_c \neq w_{c'}$ for $c \neq c'$.

Define split$(M) = (\Sigma, \Gamma, Q', q_0, R')$ with

- $\Gamma = \{\phi^{(3)}, \psi^{(2)}, e^{(0)}\} \cup \{a^{(0)} \mid a \in \Delta^{(1)}\} \cup \{(q, \sigma, b)^{(0)} \mid q \in Q, \sigma \in \Sigma, b \in \{0,1\}^{\leq n}\}$;
- $Q' = Q \cup \{q_{\text{up}} \mid q \in Q\} \cup \{q_{\text{down}} \mid q \in Q\}$
- For every rule $r = (\langle q, \sigma, b, j\rangle \to \zeta)$ in R: if $\zeta \in \langle Q, I_{\sigma,b,j}\rangle$ or $\zeta = e$, then let r be in R'; if $\zeta = a(q')$ with $a \in \Delta^{(1)}$ and $q' \in Q$ then let the rule

  $$\langle q, \sigma, b, j\rangle \to \phi(\langle q_{\text{up}}, \text{stay}\rangle, a, \langle q', \text{stay}\rangle)$$

  be in R'.

  For every $q \in Q$ and $b \in \{0,1\}^{\leq n}$ let the following rules be in R':

  $$\begin{array}{lll} \langle q_{\text{up}}, \sigma, b, 1\rangle & \to \psi((q,\sigma,b), \langle q_{\text{up}}, \text{up}\rangle) & \text{for } \sigma \in \Sigma \\ \langle q_{\text{up}}, \sigma, b, 0\rangle & \to \psi((q,\sigma,b), \langle q_{\text{down}}, \text{stay}\rangle) & \text{for } \sigma \in \Sigma \\ \langle q_{\text{down}}, \sigma, b, j\rangle & \to \psi((q,\sigma,b), \langle q_{\text{down}}, \text{down}_1\rangle) & \text{for } \sigma \in \Sigma^{(1)}, j \in \{0,1\} \\ \langle q_{\text{down}}, e, b, j\rangle & \to (q, e, b) & \text{for } j \in \{0,1\}. \end{array}$$

As before for conf(M), Γ should be defined in such a way that $A = \Delta^{(1)}$ is disjoint with $B = \Gamma - \Delta$.

Clearly, $\tau_{\text{split}(M)} \in n$-DPTT and $\text{res}_{\Delta^{(1)}}(y\tau_{\text{split}(M)}(T_\Sigma)) = p\tau_M(T_\Sigma)$.

Claim 2: Let M be a monadic n-dptt with input alphabet Σ. Then $y\tau_{\text{split}(M)}(T_\Sigma)$ is δ-complete for $p\tau_M(T_\Sigma)$.

Let M' denote split(M) and let $s \in T_\Sigma$. If $w = p\tau_M(s)$ is defined, then there is a computation by $\Rightarrow_{M,s}$ as displayed in the proof of Claim 1. Now, if $c \Rightarrow_{M,s} d$ then $c \Rightarrow_{M',s} d$, because M' has the same rules as M for right-hand sides that do not contain an output symbol. If $d \Rightarrow_{M,s} ac$, then there is a computation (showing only the yields of the respective sentential forms) $d \Rightarrow^*_{M',s} w_d ac$ where w_d is the coding of d described above. This means that there is the computation by $\Rightarrow_{M',s}$ displayed in the proof of Claim 1, generating as yield the string w' which is δ-complete for w. This proves Claim 2. □

It follows immediately from Theorem 41 and the inclusion n-DPTT $\subseteq$ DMTT^{n+1} in Theorem 35, that the dptt-hierarchy is proper: our fourth main result.

Theorem 42. The dptt-hierarchy is proper, i.e., for $n \geq 0$,

$$y(n\text{-DPTT(REGT)}) \subsetneq y((n+1)\text{-DPTT(REGT)}).$$

The fact that $y(n\text{-DPTT(REGT)}) - y\text{DMTT}^n\text{(REGT)} \neq \varnothing$ (Theorem 41) means that counter examples of the dmtt-hierarchy can already be found in the dptt-hierarchy. Thus, if we additionally knew that $y\text{DMTT(REGT)} - y\text{DPTT(REGT)} \neq \varnothing$, then the inclusion diagram in Figure 6 would be a Hasse diagram. We conjecture

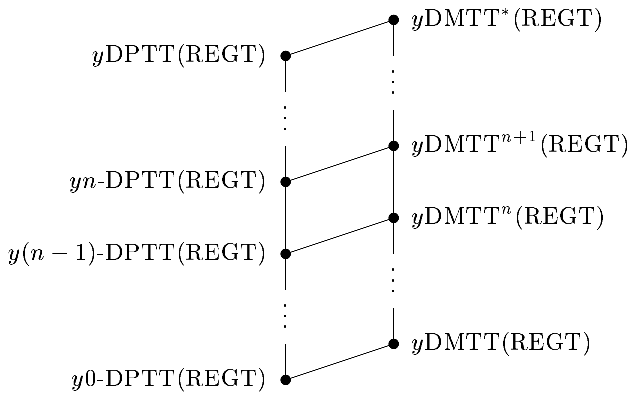

Fig. 6. Inclusion diagram relating the dptt-hierarchy to the dmtt-hierarchy

that this is the case.

Note that, with respect to the corresponding classes of translations the figure is a Hasse diagram because, as shown in Example 13, DMTT $-$ DPTT $\neq \varnothing$ (M of Example 13 is a dmtt).

In the case of nondeterministic n-pebble tree transducers (and also for compositions of nondeterministic mtts) it is open whether there is a proper hierarchy of output languages. If we compare the output languages of nondeterministic ptts with those of deterministic ones, then it can be shown that even at the lowest level (i.e., without pebbles), nondeterminism is more powerful than determinism: There is a language generated by a nondeterministic 0-ptt, which is not in DPTT*(REGT), and hence

$$\text{DPTT(REGT)} \subsetneq \text{PTT(REGT)}.$$

In terms of databases this means that, for queries realized by pebble transducers, nondeterminism gives strictly more views than determinism. It follows from the fact that there is a language L generated by a (nondeterministic) top-down tree transducer, i.e., which is in yT(REGT), and which cannot be generated by compositions of deterministic mtts, i.e., which is not in yDMTT*(REGT). This was proved in Theorem 25 of [EM02a], as another application of the bridge theorem (Lemma 40). Since T $\subseteq$ 0-PTT by Lemma 23, and DPTT* = DMTT* by Theorem 38, we obtain that $L \in y(\text{0-PTT(REGT)}) - y\text{DPTT}^*\text{(REGT)}$.

7 Type Checking

As mentioned in the Introduction, the application of a query q to a database D (a set of inputs) defines a derived version of the database: the *view of D under q*. Now if q is computed by the tree transducer M, and D is represented by the regular tree language R, then the view of D under q is equal to the output language $\tau_M(R)$. An important issue in XML-based query languages is *type checking of views* (see, e.g., [MSV00,PV00,Via01,AMN$^+$01a,AMN$^+$01b,Toz01,Suc02]). The main result of [MSV00] is that type checking is decidable for pebble tree transducers. For a class X of tree translations, the *type checking problem* (*for X*) is defined in Figure 7.

input: types $R_{\text{in}}, R_{\text{out}} \in \text{REGT}$, translation $\tau \in X$

output: $\begin{cases} \text{“yes”} & \text{if } \tau(R_{\text{in}}) \subseteq R_{\text{out}} \\ \text{“no”} & \text{otherwise.} \end{cases}$

Fig. 7. Type checking for translations in X.

If the output of type checking is “yes”, i.e., the view $\tau(R_{\text{in}})$ is included in R_{out}, then we say that τ *type checks for* $(R_{\text{in}}, R_{\text{out}})$.

Intuitively, type checking means to verify whether or not all documents in a view conform to a certain type. As a typical scenario of type checking, imagine that τ translates XML documents into HTML documents. Thus, for a set R of XML documents $\tau(R)$ is an “HTML-view” of the documents in R. Now, a particular user might be interested only in very particular XML documents, for instance, documents that have no nested lists, represented by the type $\text{XML}_{\text{no-nest}}$. Since XML documents are unranked trees, this type corresponds to a string, or, rather, a forest $a_1, \ldots, a_k$ of one node trees a_i; using the usual encoding of unranked trees by binary trees, this corresponds to ‘combs’ of the form $\sigma(a_1, \sigma(a_2, \cdots \sigma(a_k, e) \cdots))$, where σ has rank 2. Obviously, this is a regular tree language. Then, the user wants to verify that also the corresponding HTML documents $\tau(\text{XML}_{\text{no-nest}})$ do not have nested lists, i.e, are of type $\text{HTML}_{\text{no-nest}}$. Thus, he wants to know whether or not τ type checks for $(\text{XML}_{\text{no-nest}}, \text{HTML}_{\text{no-nest}})$. As mentioned above, this problem is decidable if τ is defined by a ptt.

It is well known in tree transducer theory that type checking is decidable for translations in MTT^*, i.e., it is decidable for an output language in $\text{MTT}^*(\text{REGT})$ whether or not it is included in a given regular tree language.

Proposition 43. Type checking of compositions of macro tree transducers is decidable.

This can be seen as follows. The translation $\tau \in \text{MTT}^*$ type checks for $(R_{\text{in}}, R_{\text{out}})$ iff $K = \tau(R_{\text{in}}) \cap R^c_{\text{out}}$ is empty, where R^c_{out} denotes the complement of R_{out}. Since REGT is effectively closed under complement and $\text{MTT}^*(\text{REGT})$ is effectively closed under intersection with regular tree languages, the tree language K is in $\text{MTT}^*(\text{REGT})$. This implies, by Fact 25(i), that K's emptiness is decidable which gives Proposition 43. Note that it is obvious that $\text{MTT}^*(\text{REGT})$ is closed under intersection with a regular tree language R, because that is the same as applying the partial identity for R, i.e., a mapping in FTA (cf. the discussion in the beginning of Section 6), which is a top-down tree translation and hence is in MTT.

Together with Theorem 30, Proposition 43 implies that even for compositions of stay-mtts, type checking is decidable.

Corollary 44. Type checking of compositions of stay-mtts is decidable.

Since $\text{PTT}^*(\text{REGT}) = \text{MTT}^*(\text{REGT})$ by Corollary 39, Proposition 43 gives an alternative proof of the main result of [MSV00]. We can now strengthen this result, based on the fact that the finiteness of languages in $\text{MTT}^*(\text{REGT})$ is decidable by Fact 25(ii). More precisely, we can solve *almost always type checking*, which is a weaker variation of type checking, defined in Figure 8. Intuitively, almost always type checking means to check whether or not all output documents in the view $\tau(R_{\text{in}})$, except finitely many exceptions, satisfy the output type R_{out}. Moreover, if the answer is yes, the list of exceptions is produced.

input: types $R_{\text{in}}, R_{\text{out}} \in \text{REGT}$, translation $\tau \in X$

output: $\begin{cases} \text{“yes”}, \tau(R_{\text{in}}) - R_{\text{out}} & \text{if } \tau(R_{\text{in}}) - R_{\text{out}} \text{ is finite} \\ \text{“no”} & \text{otherwise.} \end{cases}$

Fig. 8. Almost always type checking for translations in X.

Since $K = \tau(R_{\text{in}}) - R_{\text{out}}$ is in $\text{MTT}^*(\text{REGT})$, as shown above, Fact 25(ii) implies that its finiteness is decidable, and if so, that the finitely many exceptions can be computed. By Corollary 39, this proves the next theorem.

Theorem 45. Almost always type checking of compositions of pebble tree transducers is solvable.

Note that in the affirmative case, τ type checks for $(R_{\text{in}}, R_{\text{out}} \cup F)$ where $F = \tau(R_{\text{in}}) - R_{\text{out}}$ is the finite set of exceptions. The new output type $R_{\text{out}} \cup F$ is indeed a regular tree language and can be determined effectively.

In the remainder of this section we present a straightforward type checking algorithm for translations realized by (compositions of) deterministic mtts. As shown below Proposition 43, type checking for compositions of dmtts can be solved using Fact 25(i), and as discussed below Fact 25, the proof of Fact 25(i) uses *inverse type inference (for τ and R_{out})*; this means to determine the set of input trees of τ which generate output in R_{out}, i.e., to determine the regular tree language $\tau^{-1}(R_{\text{out}})$. Recall the example XML to HTML translation τ of before. Now imagine that the generated HTML documents should conform to a certain type R_{out}, and one wants to know which XML documents are admissible as input of τ, in order to generate documents of the required type R_{out}: just do inverse type inference for τ and R_{out}.

Clearly, for a function τ

$$\tau \text{ typechecks for } (R_{\text{in}}, R_{\text{out}}) \quad \text{iff} \quad R_{\text{in}} \subseteq \tau^{-1}(R_{\text{out}}).$$

Since checking the inclusion of two regular tree languages is well known, we concentrate on the inverse type inference problem. Note that also in [MSV00] type checking is solved by inverse type inference (using MSO logic to represent types).

If τ is a composition $\tau_1 \circ \tau_2 \circ \cdots \circ \tau_n$ of translations, then

$$\tau^{-1}(R_{\text{out}}) = \tau_1^{-1}(\tau_2^{-1}(\cdots \tau_n^{-1}(R_{\text{out}}))).$$

Thus, to solve the type inference problem for X^* (where X is a class of translations) it suffices to solve it for X.

We now discuss an algorithm that performs inverse type inference for τ_M and R_{out}, for a deterministic mtt M and an output type R_{out}. Hence, the algorithm constructs a description of the regular tree language $\tau_M^{-1}(R_{\text{out}})$. Note that the existence of such an algorithm follows from the facts that $\text{DMTT} \subseteq (\text{DT} \cup \text{YIELD})^3$, see Subsection 5.5, and that the inverses of DT and YIELD both (effectively) preserve the regular tree languages (cf. the proof of Fact 24 in Theorem 7.4 of [EV85]). From the proofs of these results in the literature it is straightforward, but rather awkward, to extract the algorithm. Since, in fact, a direct algorithm is quite easy to understand, we present it here. As description for a regular tree language we use the deterministic bottom-up finite state tree automaton, defined next.

A *deterministic bottom-up finite state tree automaton* (for short, dbfta) is a tuple $B = (P, P_{\text{fin}}, \Sigma, \delta)$ where P is a finite set of states, $P_{\text{fin}} \subseteq P$ is the set of final states, Σ is a ranked alphabet, and δ is the collection $(\delta_\sigma)_{\sigma \in \Sigma}$ of transition functions such that for every $\sigma \in \Sigma^{(k)}$, $k \geq 0$, δ_σ is a mapping from P^k to P. The tree language $L(B) \subseteq T_\Sigma$ recognized by B is $\{s \in T_\Sigma \mid \delta(s) \in P_{\text{fin}}\}$ where δ is the extension of δ_σ to trees in T_Σ which is recursively defined as follows. For every $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k \in T_\Sigma$, $\delta(\sigma(s_1, \ldots, s_k)) = \delta_\sigma(\delta(s_1), \ldots, \delta(s_k))$.

Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a deterministic mtt. For technical reasons we assume M to be total. Clearly, this is not a restriction: just add a new "undefined" symbol $\perp$ and for each left-hand side that has no rule, add a rule with right-hand side $\perp$. Moreover, we assume that the $\langle q, \sigma, \lambda, j\rangle$-rules of M disregard j, i.e., all $\langle q, \sigma, \lambda, j\rangle$-rules for $j \in [0, J]$ have the same right-hand side. (Obviously this is not a restriction, because the j can be incorporated into the states; cf. the discussion below Lemma 23.)

We are now ready to construct the dbfta A with $L(A) = \tau_M^{-1}(R_{\text{out}})$. Let $B = (P, P_{\text{fin}}, \Delta, \beta)$ be a deterministic bottom-up finite state tree automaton with $L(B) = R_{\text{out}}$. Define $A = (D, D_{\text{fin}}, \Sigma, \delta)$ where D consists of all mappings d such that for every $q \in Q^{(m)}$ and $m \geq 0$, $d(q)$ is a mapping from P^m to P, and D_{fin} consists of all $d \in D$ such that $d(q_0)() \in P_{\text{fin}}$.

For every $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $d_1, \ldots, d_k \in D$, let $\delta_\sigma(d_1, \ldots, d_k) = d$ where d is defined as follows. For every $q \in Q^{(m)}$, $m \geq 0$, $p_1, \ldots, p_m \in P$, and rule $\langle q, \sigma, \lambda, j\rangle(y_1, \ldots, y_m) \to \zeta$ in R, let

$$d(q)(p_1, \ldots, p_m) = \beta'(\zeta[y_j \leftarrow p_j \mid j \in [m]]),$$

where β' is the following extension of β to trees over $\langle Q, \{\text{down}_i \mid i \in [k]\}\rangle \cup \Delta \cup \{p^{(0)} \mid p \in P\}$. For every $\langle q', \text{down}_i\rangle \in \langle Q, \{\text{down}_i \mid i \in [k]\}\rangle^{(m')}$, $m' \geq 0$, and $p'_1, \ldots, p'_{m'}$, let

$$\beta'_{\langle q', \text{down}_i\rangle}(p'_1, \ldots, p'_m) = d_i(q')(p'_1, \ldots, p'_{m'}),$$

and let $\beta'_p() = p$ for $p \in P$. This ends the construction of A.

Intuitively, the idea of A is to run the dbfta B on the right-hand sides of the rules of M. In order to do this, B has to be extended appropriately, because the right-hand side ζ of a q-rule of M might contain parameters y_j or instructions of the form $\langle q, \text{down}_i\rangle$. Since the state p_j in which B arrives after processing the actual parameter tree t_j of y_j is not determined, a state d of A will contain all possible choices of states of B for the parameters, i.e., $d(q)$ is a function from P^m to P and $d(q)(p_1, \ldots, p_m) = p$ means that, assuming state p_μ for t_μ, $\mu \in [m]$, B will arrive in p after processing ζ. The $\langle q', \text{down}_i\rangle$ in ζ are handled by applying $d_i(q')$, where d_i is the state in which A arrives at the ith input subtree. In fact, for $s \in T_\Sigma$, $\delta(s)(q)(p_1, \ldots, p_m)$ is the state in P in which B arrives on the output tree generated by q on input s assuming that it arrives in p_μ for the parameter y_μ. More precisely, if $\langle q, h_0\rangle(y_1, \ldots, y_m) \Rightarrow^*_{M,s} t \in T_\Delta(Y_m)$ then $\delta(s)(q)(p_1, \ldots, p_m) = \beta(t[y_\mu \leftarrow t_\mu])$ where, for $\mu \in [m]$, t_μ is an arbitrary tree in T_Δ with $\beta(t_\mu) = p_\mu$. From this it should be clear that indeed

$$L(A) = \{s \in T_\Sigma \mid \tau_M(s) \cap R_{\text{out}} \neq \varnothing\} = \tau_M^{-1}(R_{\text{out}}).$$

This concludes the construction of the dbfta A and our inverse type inference algorithm.

8 Conclusions and Problems

In this paper we have shown that n-ptts can be decomposed into compositions of 0-ptts and compositions of mtts, respectively: (1) $n\text{-PTT} \subseteq 0\text{-PTT}^{n+1}$ and $n\text{-DPTT} \subseteq 0\text{-DPTT}^{n+1}$ and (2) $n\text{-PTT} \subseteq \text{sMTT}^{n+1}$ and $n\text{-DPTT} \subseteq \text{DMTT}^{n+1}$. It was shown that (3) $\text{PTT}^* = 0\text{-PTT}^* = \text{sMTT}^*$ and $\text{DPTT}^* = 0\text{-DPTT}^* = \text{DMTT}^*$, i.e., as query languages all three models, n-ptt, mtt, and 0-ptt, have the same expressiveness. The output languages of dptts form a proper hierarchy with respect to the number of pebbles: (4) n-DPTT(REGT) $\subsetneq$ $(n+1)$-DPTT(REGT) which even holds for the yields of these tree languages. Finally, (5) almost always type checking for n-ptts is decidable.

We now discuss some topics for further research. It was shown in Subsection 3.2 that (deterministic) zero-pebble tree transducers are, essentially, attribute grammars. This implies that the implementation techniques known for attribute grammars (see, e.g., [DJL88,AM91,Paa95]) carry over to zero-pebble tree transducers. The question arises, whether and how these techniques can be generalized to the n-pebble case.

In Section 6 it was proved that the output languages of deterministic n-ptts form a proper hierarchy with respect to n, see (4) above. The proof is similar to (and uses parts of) the proof in [EM02a] of the fact that the output languages of n-fold compositions of deterministic macro tree transducers give rise to a proper hierarchy, with respect to n. As observed in Section 6, the exact Hasse diagram for these hierarchies (see Fig. 6) has not yet been determined. It would also be interesting to know whether or not the hierarchy of output languages of nondeterministic n-pebble tree transducers is proper. Note that also for output languages of macro tree transducers the properness of the nondeterministic hierarchy is an open problem (stated in [EM02a]).

In Section 5 the n-pebble macro tree transducer was defined, but not investigated. It is straightforward to extend the decomposition result of Section 4 to the macro case, in the following way:

$$n\text{-PMTT} \subseteq 0\text{-PTT}^n \circ 0\text{-PMTT}.$$

For the deterministic case a similar result can be proved. Now note that the translation τ_M of the 0-dpmtt M of Example 13 is equal to the composition $\tau_{M_1} \circ \tau_{M_2}$ of the two deterministic 0-ptts M_1 and M_2 of Example 4. We suspect that every (deterministic) 0-pebble macro tree transducer can be realized by the composition of two (deterministic) 0-ptts. In fact, by Subsection 3.2, 0-ptts are essentially attributed tree transducers; the addition of parameters to the attributes of the attributed tree transducer gives the macro attributed tree transducer of [KV94] which can be simulated by the composition of two attributed tree transducers (to be precise, the class of translations realized by macro attributed tree transducers equals the class of two-fold compositions of attributed tree transducers; cf. Corollary 7.30 of [FV98]). For the pebble formalism this suggests that 0-PMTT $\subseteq$ 0-PTT^2 and 0-DPMTT $\subseteq$ 0-DPTT^2 ; does this actually hold? As a special case of Corollary 3.27 of [EV86] (viz. the case that S = Tree-walk) we obtain that 0-PMTT $\subseteq$ 0-PTT $\circ$ MTT and hence n-PMTT $\subseteq$ sMTT^{n+2} and PMTT* = PTT*. Is it true that 0-DPMTT $\subseteq$ DMTT^2? If so, then we would obtain that n-DPMTT $\subseteq$ DMTT^{n+2} and DPMTT* = DPTT*. Using the results of [EV86] it can be shown that the total functions in 0-DPMTT are also in DMTT^2.

Furthermore, it can probably be shown that n-PTT $\subseteq$ $(n-1)$-PMTT, i.e., a pebble can be avoided by the addition of parameters, in a similar way as the removal of the tree-walk facility of the reading-head (which can be seen as a pebble), in the proof of Lemma 34.

Last but not least, we conjecture that the class DPTT of deterministic pebble tree translations can be characterized inside the class DPTT* as those translations for which the number of different subtrees in the output tree is polynomial in the size of the input tree (cf. [EM01], where the MSO definable tree translations are characterized inside the class DMTT as those translations for which the size of the output tree is linear in the size of the input tree).

Acknowledgment

We are grateful to Frank Neven for drawing our attention to the pebble tree transducer model of [MSV00,Via01], and for asking us about its relationship to the macro tree transducer.

References

[AM91] H. Alblas and B. Melichar, editors. *International Summer School on Attribute grammars, applications and systems*, volume 545 of *LNCS*. Springer-Verlag, 1991.

[AMN$^+$01a] N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. Typechecking XML views of relational databases. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science – LICS'2001*. IEEE, 2001.

[AMN$^+$01b] N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with data values: Typechecking revisited. In *Proceedings of the 20th ACM Symposium on Principles of Database Systems – PODS'2001*, pages 138 – 149. ACM Press, 2001.

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

[BMN02] G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, 27:21–39, 2002.

[CF82] B. Courcelle and P. Franchi-Zannettacci. Attribute grammars and recursive program schemes. *Theoret. Comput. Sci.*, 17:163–191 and 235–257, 1982.

[CJ77] M. P. Chytil and V. Jákl. Serial composition of 2-way finite-state transducers and simple programs on strings. In A. Salomaa and M. Steinby, editors, *Proceedings of the 15th International Colloquium on Automata, Languages and Programming – ICALP'77*, volume 52 of *LNCS*, pages 135–147. Springer-Verlag, 1977.

[Cou83] B. Courcelle. Fundamental properties of infinite trees. *Theoret. Comput. Sci.*, 25:95–169, 1983.

[DE98] F. Drewes and J. Engelfriet. Decidability of finiteness of ranges of tree transductions. *Inform. and Comput.*, 145:1–50, 1998.

[DJ90] N. Dershowitz and J.P. Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, chapter 6, pages 243–320. Elsevier, 1990.

[DJL88] P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars, Definitions and Bibliography*, volume 323 of *LNCS*. Springer-Verlag, 1988.

[EF81] J. Engelfriet and G. Filè. The formal power of one-visit attribute grammars. *Acta Informatica*, 16:275–302, 1981.

[EH99] J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In J. Karhumäki, H. Maurer, Gh. Păun, and G. Rozenberg, editors, *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa*, volume 1644 of *LNCS*, pages 72–83. Springer-Verlag, 1999.

[EM99] J. Engelfriet and S. Maneth. Macro tree transducers, attribute grammars, and MSO definable tree translations. *Inform. and Comput.*, 154:34–91, 1999.

[EM01] J. Engelfriet and S. Maneth. Macro tree translations of linear size increase are MSO definable. Technical Report 01-08, Leiden University, 2001. To appear in SIAM J. on Computing.

[EM02a] J. Engelfriet and S. Maneth. Output string languages of compositions of deterministic macro tree transducers. *J. of Comp. Syst. Sci.*, 64:350–395, 2002.

[EM02b] J. Engelfriet and S. Maneth. Two-way finite state transducers with nested pebbles. In K. Diks and W. Rytter, editors, *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science – MFCS'2002*, volume 2430 of *LNCS*, pages 234–244. Springer-Verlag, 2002.

[Eng80] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory; perspectives and open problems*. New York, Academic Press, 1980.

[Eng81] J. Engelfriet. Tree transducers and syntax-directed semantics. Technical Report Memorandum 363, Technische Hogeschool Twente, 1981. Also in: Proceedings of the 7th Colloquium on Trees in Algebra and Programming – CAAP'82, Lille, France, 1982.

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

[Eng86] J. Engelfriet. Context-free grammars with storage. Technical Report 86-11, University of Leiden, 1986.

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

[ES77] J. Engelfriet and E.M. Schmidt. IO and OI, Part I. *J. of Comp. Syst. Sci.*, 15:328–353, 1977. And Part II, *J. of Comp. Syst. Sci.*, 16: 67–99 (1978).

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *J. of Comp. Syst. Sci.*, 31:71–146, 1985.

[EV86] J. Engelfriet and H. Vogler. Pushdown machines for the macro tree transducer. *Theoret. Comput. Sci.*, 42:251–368, 1986.

[EV88] J. Engelfriet and H. Vogler. High level tree transducers and iterated pushdown tree transducers. *Acta Informatica*, 26:131–192, 1988.

[Fis68] M.J. Fischer. *Grammars with macro-like productions.* PhD thesis, Harvard University, Massachusetts, 1968.

[FM00] Z. Fülöp and S. Maneth. Domains of partial attributed tree transducers. *Inform. Proc. Letters*, 73:175–180, 2000.

[Fra82] P. Franchi-Zannettacci. *Attributes semantiques et schemas de programmes.* PhD thesis, Université de Bordeaux I, 1982. Thèse d'Etat.

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

[FV98] Z. Fülöp and H. Vogler. *Syntax-Directed Semantics – Formal Models based on Tree Transducers.* EATCS Monographs on Theoretical Computer Science (W. Brauer, G. Rozenberg, A. Salomaa, eds.). Springer-Verlag, 1998.

[FV99] Z. Fülöp and H. Vogler. A characterization of attributed tree transformations by a subclass of macro tree transducers. *Theory Comput. Systems*, 32:649–676, 1999.

[GH96] N. Globerman and D. Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoret. Comput. Sci.*, 169:161–184, 1996.

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

[GS84] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[GS97] F. Gécseg and M. Steinby. Tree automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Volume 3*, chapter 1. Springer-Verlag, 1997.

[GTWW77] J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24:68–95, 1977.

[Iro61] E.T. Irons. A syntax directed compiler for ALGOL 60. *Commun. ACM*, 4:51–55, 1961.

[Kam83] T. Kamimura. Tree automata and attribute grammars. *Inform. and Control*, 57:1–20, 1983.

[Klo92] J. W. Klop. Term rewrite systems. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 2*, pages 1–116. Oxford Science Publications, 1992.

[KMMM] H.-P. Kolb, J. Michaelis, U. Mönnich, and F. Morawietz. An operational and denotational approach to non-context-freeness. To appear in *Theoret. Comput. Sci.*

[Knu68] D.E. Knuth. Semantics of context-free languages. *Math. Systems Theory*, 2:127–145, 1968. (Corrections in *Math. Systems Theory*, 5:95-96, 1971).

[Küh98] A. Kühnemann. Benefits of tree transducers for optimizing functional programs. In V. Arvind and R. Ramanujam, editors, *Proceedings of the 18th Conference on Foundations of Software Technology and Theoretical Computer Science – FST&TCS'98*, volume 1530 of *LNCS*, pages 146–157. Springer-Verlag, 1998.

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

[KV97] A. Kühnemann and H. Vogler. *Attributgrammatiken*. Vieweg-Verlag, 1997.

[Man02] S. Maneth. The complexity of compositions of deterministic tree transducers. In M. Agrawal and A. Seth, editors, *Proceedings of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science – FSTTCS'2002*, volume 2556 of *LNCS*, pages 265–276. Springer-Verlag, 2002.

[MMM01] J. Michaelis, U. Mönnich, and F. Morawietz. On minimalist attribute grammars and macro tree transducers. In C. Rohrer, A. Rossdeutscher, and H. Kamp, editors, *Linguistic Form and its Computation*, pages 287–326. CSLI Publications, Stanford, 2001.

[MN00] S. Maneth and F. Neven. Recursive structured document transformations. In R. Conner and A. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming – Revised Papers DBPL'99*, volume 1949 of *LNCS*, pages 80–98. Springer-Verlag, 2000.

[MSV] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. To appear in *J. of Comp. Syst. Sci.*

[MSV00] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *Proceedings of the 19th ACM Symposium on Principles of Database Systems – PODS'2000*, pages 11–22. ACM Press, 2000.

[NSV01] F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science – MFCS'2001*, volume 2136 of *LNCS*. Springer-Verlag, 2001.

[Paa95] J. Paakki. Attribute grammar paradigms – a high-level methodology in language implementation. *ACM Computing Surveys*, 27:196–255, 1995.

[PV00] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proceedings of the 19th ACM Symposium on Principles of Database Systems – PODS'2000*, pages 35–46. ACM Press, 2000.

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

[Suc02] D. Suciu. The XML typechecking problem. *SIGMOD Record*, 31:89 – 96, 2002.

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

[Toz01] A. Tozawa. Towards static type checking for XSLT. In *Proceedings of the ACM Symposium on Document Engineering*, pages 18 – 27. ACM Press, 2001.

[Via01] V. Vianu. A Web Odyssey: From Codd to XML. In *Proceedings of the 20th ACM Symposium on Principles of Database Systems – PODS'2001*, pages 1–15. ACM Press, 2001.

[Vog91] H. Vogler. Functional description of the contextual analysis in block-structured programming languages: a case study of tree transducers. *Science of Computer Programming*, 16:251–275, 1991.

[Voi02] J. Voigtländer. Conditions for efficiency improvement by tree transducer composition. In *Proceedings of the 13th International Conference on Rewriting Techniques and Applications – RTA'2002*, volume 2378 of *LNCS*, pages 222–236. Springer-Verlag, 2002.

[WM95] R. Wilhelm and D. Maurer. *Compiler Design*. Addison-Wesley, 1995.