Acta Informatica
https://doi.org/10.1007/s00236-019-00360-8

ORIGINAL ARTICLE

Linear-bounded composition of tree-walking tree transducers: linear size increase and complexity

Joost Engelfriet[1] · Kazuhiro Inaba[2] · Sebastian Maneth[3]

Received: 19 April 2019 / Accepted: 27 November 2019

Abstract

Compositions of tree-walking tree transducers form a hierarchy with respect to the number of transducers in the composition. As main technical result it is proved that any such composition can be realized as a linear-bounded composition, which means that the sizes of the intermediate results can be chosen to be at most linear in the size of the output tree. This has consequences for the expressiveness and complexity of the translations in the hierarchy. First, if the computed translation is a function of linear size increase, i.e., the size of the output tree is at most linear in the size of the input tree, then it can be realized by just one, deterministic, tree-walking tree transducer. For compositions of deterministic transducers it is decidable whether or not the translation is of linear size increase. Second, every composition of deterministic transducers can be computed in deterministic linear time on a RAM and in deterministic linear space on a Turing machine, measured in the sum of the sizes of the input and output tree. Similarly, every composition of nondeterministic transducers can be computed in simultaneous polynomial time and linear space on a nondeterministic Turing machine. Their output tree languages are deterministic context-sensitive, i.e., can be recognized in deterministic linear space on a Turing machine. The membership problem for compositions of nondeterministic translations is nondeterministic polynomial time and deterministic linear space. All the above results also hold for compositions of macro tree transducers. The membership problem for the composition of a nondeterministic and a deterministic tree-walking tree translation (for a nondeterministic IO macro tree translation) is log-space reducible to a context-free language, whereas the membership problem for the composition of a deterministic and a nondeterministic tree-walking tree translation (for a nondeterministic OI macro tree translation) is possibly NP-complete.

✉ Sebastian Maneth
maneth@uni-bremen.de

Joost Engelfriet
j.engelfriet@liacs.leidenuniv.nl

Kazuhiro Inaba
kinaba@google.com

1 LIACS, Leiden University, Leiden, The Netherlands

2 Google Japan G.K., Tokyo, Japan

3 Department of Mathematics and Informatics, Universität Bremen, P.O. Box 330 440, 28334 Bremen, Germany

Contents

1 Introduction

Tree transducers are used, e.g., in compiler theory or, more generally, the theory of syntax-directed semantics of context-free languages [39], and in the theory of XML queries and XML document transformation [47,69]. One of the most basic types of tree transducer is the top-down tree transducer (in short $\mathrm{TT}_\downarrow$). It is a finite-state device that walks top-down on the input tree, from parent to child, possibly branching into parallel copies of itself at each step (thus allowing the transducer to visit all children of the parent). During this process, the output tree is generated top-down. The $\mathrm{TT}_\downarrow$ has been generalized in two different ways. By allowing it to walk also bottom-up, from child to parent, still possibly branching at every step and still generating the output tree top-down, one obtains the tree-walking tree transducer (in short, TT).[1] On the other hand, restricting its walk to be top-down but allowing its states to have parameters of type output tree, one obtains the macro tree transducer (in short, MT). In general we consider nondeterministic transducers, with deterministic transducers as an important special case (abbreviated as $\mathrm{dTT}_\downarrow$, dTT, and dMT).

To turn the $\mathrm{TT}_\downarrow$ into a more flexible model of tree transformation, it was enhanced with the feature of regular look-ahead, which means that it can test whether or not the subtree at the current node of the input tree belongs to a given regular tree language. The MT already has the ability to implement regular look-ahead. Since both the enhanced $\mathrm{TT}_\downarrow$ and the MT process the input tree top-down, they can also implement "regular look-around", which means that they can test arbitrary regular properties of the current node of the input tree. More precisely, they can test whether the input tree, in which the current node is marked, belongs to a given regular tree language. Such regular look-around tests are also called MSO tests, because they can be expressed by formulas of monadic second-order logic with one free node variable. The TT, as defined in [63], does not have regular look-ahead or look-around. One of the drawbacks of this is that the TT cannot recognize all regular tree languages without branching [9]. Hence, from now on, we assume that the TT (and the $\mathrm{TT}_\downarrow$) is enhanced with regular look-around,

[1] The name "tree-walking tree transducer" was introduced in [26]. The adjective "tree-walking" stands for the fact that the transducer walks on the input tree (just as the tree-walking automaton of [2]). The TT is the generalization to trees of the two-way finite-state string transducer, which walks on its input string in both directions and produces the output string one-way from left to right. Note that "tree-walking" and "two-way" alliterate.

i.e., with regular tests of the current input node. The resulting TT formalism is a quite robust, flexible, and intuitive model of tree transformation.

The TT and MT, generalizations of the $\mathrm{TT}_{\downarrow}$, are closely related, in particular in the deterministic case. In fact, every dTT can be simulated by a dMT, whereas every dMT can be simulated by a composition of two dTT's. Thus, every composition of dTT's can be realized by a composition of dMT's, and vice versa. Compositions of dTT's form a proper hierarchy, in an obvious way. A single dTT is at most of exponential size increase, which means that the size of the output tree is at most exponential in the size of the input tree. However, a composition of two dTT's can be of double exponential size increase. In general, compositions of k dTT's are at most, and can be, of k-fold exponential size increase. Compositions of dMT's form a proper hierarchy by a similar argument. For nondeterministic TT's and MT's the situation is similar but more complicated. Every MT can be simulated by a composition of two TT's. However, as opposed to TT's, MT's are always finitary, which means that for every given input tree an MT computes finitely many output trees.

In this paper we investigate compositions of TT's (and hence of MT's) with respect to their expressivity and their complexity. Our main technical result is that every composition of TT's can be realized by a *linear-bounded composition* of TT's, which means that, when computing an output tree from an input tree, the intermediate results can be chosen in such a way that their sizes are at most linear in the size of the output tree. More precisely, a composition of two transducers (for simplicity) is linear-bounded if there is a constant c such that for every pair (t, s) of an input tree t and output tree s in the composed translation there is an intermediate tree r (meaning that (t, r) and (r, s) are in the first and second translation, respectively) such that the size of r is at most c times the size of s. Intuitively, to compute s from t there is no need to consider intermediate results that are much larger than s. If both transducers are deterministic it means that for every input tree t in the domain of the composed translation the size of the unique intermediate tree r is at most linear in the size of the unique output tree s.

To prove that every composition of two TT's can be realized by a linear-bounded composition of two TT's, we first show that every TT can be decomposed into a $\mathrm{TT}_{\downarrow}$ that "prunes" the input tree, followed by a TT that is "productive" on at least one of the intermediate trees generated by the $\mathrm{TT}_{\downarrow}$, which means that it uses each leaf and each monadic node of that intermediate tree in order to generate the output tree. Productivity guarantees that the composition of these two transducers is linear-bounded. We also prove that the composition of an arbitrary TT with a "pruning" top-down TT can be realized by one TT. Thus, when two TT's are composed, the second TT can split off the pruning $\mathrm{TT}_{\downarrow}$ (to the left), which can be absorbed (to the right) by the first TT. The composition of the resulting two TT's is then linear-bounded. This also holds for deterministic transducers, in which case the pruning TT is also deterministic. Similar results were presented for macro tree transducers in [58, Section 3] and [51, Section 4].

Thus, roughly speaking, our main technical result provides a method to implement compositions of TT's in such a way that the generation of superfluous nodes, i.e., nodes on which a TT just walks around without producing any output, is avoided by pruning those superfluous parts from the intermediate trees. As such it can be viewed as a static garbage collection procedure, and leads, in principle, to algorithms for automatic compiler and XML query optimization. Since TT's are essentially finite-state automata walking on trees, it is not really surprising that only a linearly bounded amount of intermediate information is useful to the final output. However, proving this rigorously requires quite some effort. In particular, the subcomputations of the TT during which it does not produce output will be determined by regular look-around.

The above method can be used to obtain results on both the expressivity and the complexity of compositions of TT's, as discussed in the next paragraphs.

Expressivity We have seen above that compositions of TT's can be of k-fold exponential size increase. However, many real world tree transformations are of *linear size increase*. We prove that the hierarchy of compositions of deterministic TT's collapses when restricted to translations of linear size increase: every composition of dTT's that is of linear size increase can be realized by just one dTT. We also show that it is decidable whether or not a composition of dTT's is of linear size increase. This means that a compiler or XML query, no matter how inefficiently programmed in several phases, can be realized in one efficient phase, provided it is of linear size increase. In fact, as we will see below, that single phase can be executed in linear time. More theoretically, we additionally prove that a function that can be realized by a composition of nondeterministic TT's, can also be realized by a composition of deterministic TT's, and hence by one deterministic TT if that function is of linear size increase. Thus, the only (functional) tree transformations that can be realized by a composition of TT's but not by a single TT, are tree transformations of superlinear size increase.

The proof of the collapse of the hierarchy of compositions of dTT's is based on the known fact that every dTT of linear size increase can be realized by a dTT that is "single-use", which means that it never visits a node of the input tree twice in the same state. In fact, it is proved in [29,32] that even dMT's of linear size increase can be realized by single-use dTT's. Vice versa, it is obvious that every single-use dTT is of linear size increase. In [7] it is shown that single-use dTT's have the same power as deterministic MSO tree transducers, which use formulas of monadic second-order logic to define the output tree in terms of the input tree (see [13,14]).

By our main technical result, we may always assume that a composition of two dTT's is linear-bounded. If the composition is of linear size increase, then the first dTT is obviously also of linear size increase, and can therefore be realized by a single-use dTT. We also prove that the composition of a single-use dTT with an arbitrary dTT can be realized by one dTT. Thus, altogether, if the composition of two dTT's is of linear size increase, then it can be realized by a single-use dTT. This argument can easily be turned into an inductive proof for a composition of any number of dTT's.

Complexity We first consider deterministic TT's. The translation realized by a deterministic TT can be computed on a RAM in linear time, in the sum of the sizes of the input and output tree. With respect to space, we prove that it can be computed on a deterministic Turing machine in linear space (again, in the sum of the sizes of the input and output tree). Since we may assume by our main technical result that the sizes of the intermediate results are at most linear in the size of the output tree, it should be clear that these facts also hold for compositions of dTT's. We also consider output tree languages, i.e., the images of a regular tree language under a composition of dTT's. Since the regular tree languages are closed under prunings, our technical decomposition result now implies that these output languages are in $\mathsf{DSPACE}(n)$, i.e., can be recognized by a Turing machine in deterministic linear space (or, in other words, are deterministic context-sensitive). Since the yield of a tree can be computed by a dTT (representing it by a monadic tree), the output string languages, which are the yields of the output tree languages, are also in $\mathsf{DSPACE}(n)$. The languages in the well-known IO-hierarchy are examples of such output languages. For compositions of top-down tree transducers (even nondeterministic ones) this result on output languages was proved in [4], using a technical result very similar to ours.

Our results on nondeterministic TT's (and their proofs) are very similar to those for dTT's. The translation realized by a composition of TT's can be computed by a nondeterministic Turing machine in simultaneous polynomial time and linear space (in the sum of the sizes of the input and output tree). The corresponding output languages can be recognized by such a Turing machine and hence are in NPTIME. Using the results on the membership problem for compositions of TT's discussed in the next paragraph, we generalize the result of [4] and prove that these output languages are even in DSPACE(n), which means that they are deterministic context-sensitive. The languages in the well-known OI-hierarchy are examples of such output languages.

Finally, we consider the membership problem for compositions of TT's, which asks whether or not a given pair (t, s) of input tree t and output tree s belongs to the composed translation. It follows easily from the above complexity results that for (non)deterministic TT's the problem is decidable in (non)deterministic polynomial time and in (non)deterministic linear space. For the special case of the composition of a nondeterministic TT with a deterministic TT we prove that the problem is even in LOGCFL, i.e., log-space reducible to a context-free language, and hence in PTIME and DSPACE($\log^2 n$). From this we conclude that for nondeterministic TT's the problem is even decidable in deterministic linear space. However, for the special case of the composition of a deterministic TT with a nondeterministic one, the problem can be NP-complete. From the two special cases we obtain that the membership problem for a (single) nondeterministic macro tree transducer is in LOGCFL for IO macro tree transducers (strengthening the result in [52] where it was shown to be in PTIME), whereas it can be NP-complete for OI macro tree transducers.

Structure of the paper The reader is assumed to be familiar with the basics of formal language theory, in particular tree language theory, and complexity theory. The only formalisms used are tree-walking tree transducers (TT's, of course), top-down tree transducers ($\text{TT}_\downarrow$'s, as a special case of TT's), context-free grammars, regular tree grammars, and finite-state tree automata. Results on macro tree transducers are taken from the literature.

The main results are proved in Sects. 8, 9, 10, 11 and 12. Section 2 contains a number of preliminary notions, in particular linear-bounded composition, linear size increase, and regular look-around. In Sect. 3 we define the tree-walking tree transducer (with regular look-around), together with some of its special cases such as top-down and single-use. A TT that does not use regular look-around tests is called “local”. A “pruning” TT is a $\text{TT}_\downarrow$ that, roughly speaking, removes or relabels each node of the input tree and possibly deletes several of its children (together with their descendants). After giving two examples we present the composition hierarchy of dTT's and end the section with some elementary syntactic properties of TT's. In Sect. 4 it is shown how to separate the regular look-around from a TT and incorporate it into another TT. For instance, every TT can be decomposed into a deterministic pruning $\text{TT}_\downarrow$ that just relabels the nodes of the input tree (and hence does not really “prune”), followed by a local TT. We also state the fact that the domain of a TT is a regular tree language. Consequently, it is possible to define the regular tests of a TT as domains of other TT's, which is a convenient technical tool. Section 5 contains three composition results. We prove that the composition of a TT with a pruning $\text{TT}_\downarrow$ can be realized by a TT (such that determinism is preserved). Together with the above-mentioned decomposition, this implies for instance that in a composition of two TT's, the second TT can always be assumed to be local: the second TT splits off a pruning $\text{TT}_\downarrow$ that is absorbed by the first TT. In the deterministic case, we even prove that the composition of a dTT with an arbitrary $\text{dTT}_\downarrow$ can be realized by a dTT, and we also prove that the composition of a single-use dTT with a dTT can be realized by a dTT. Section 6 presents the known fact that every dTT of linear size increase can be realized by a single-use dTT, and discusses the relationship between TT's, macro tree transducers, and MSO tree transducers. In Sect. 7 we show that a (partial) function that can be realized by a composition of nondeterministic TT's, can also be realized by a composition of deterministic TT's. To prove this we first prove a lemma: for every $\mathrm{TT}_{\downarrow}$ there is a deterministic $\mathrm{TT}_{\downarrow}$ that realizes a "uniformizer" of the translation realized by the given $\mathrm{TT}_{\downarrow}$, i.e., a function that is a subset of that translation, with the same domain. Section 8 contains our main technical result: every TT can be decomposed into a pruning TT and another TT such that the composition is linear-bounded. It implies (by splitting and absorbing) that a composition of TT's can always be assumed to be linear-bounded. The "uniformizer" lemma of the previous section is applied to the pruning $\mathrm{TT}_{\downarrow}$, proving the same result for deterministic TT's. Section 9 presents the main results on linear size increase, and Sects. 10 and 11 present the main results on the complexity of compositions of deterministic and nondeterministic TT's, respectively. In Sect. 12 we prove the main results on the complexity of the membership problem for the composition of two TT's. Finally, in Sect. 13 we show (in a straightforward way) that all main results also hold for transducers that transform unranked trees, or forests, which are a natural model of XML documents.

The reader who is interested only in complexity can disregard all results on single-use TT's, and skip Sects. 6.2 and 9. The reader who is interested only in expressivity can just skip Sects. 10, 11 and 12.

Remarks on the literature Top-down tree transducers were introduced in [66,72]; regular look-ahead was added in [20]. Macro tree transducers were introduced in [15,34]. Tree-walking tree transducers were introduced in [63] (where they are called 0-pebble tree transducers), and studied in, e.g., [26,31,62]. They were already mentioned in [24, Section 3(7)] (where they are called RT(Tree-walk) transducers). Regular look-around was added to TT's in [14, Section 8.2] (where they are called MS tree-walking transducers); for tree-walking automata that was already done in [6]. However, formal models similar to the TT were introduced and studied before. The tree-walking automaton of [2] translates trees into strings. As explained in [24, Section 3(7)] and [31, Section 3.2], the TT is closely related to the attribute grammar [53], which is a well-known model of syntax-directed semantics (and a compiler construction tool). An attribute grammar translates derivation trees of an underlying context-free grammar into arbitrary values. Tree-valued attribute grammars were considered, e.g., in [27]. The attributed tree transducer, introduced in [38], is an operational version of the tree-valued attribute grammar, without underlying context-free grammar. Regular look-around was added to the attributed tree transducer in [7] (where it is called look-ahead). Attributed tree transducers are a special type of TT's, of which the states are viewed as attributes of the nodes of the input tree. By definition a deterministic attributed tree transducer (like an attribute grammar) has to be noncircular, which means that it should generate an output tree whenever it is started in any state on any node of an input tree. Thus, it is total in a strong sense. This is natural from the point of view of syntax-directed semantics, but quite restrictive and inconvenient from the operational point of view of tree transformation. Several of the auxiliary results in Sects. 3, 4 and 5 are closely related to (and generalizations of) well-known results on attributed tree transducers (see, e.g., [39]). As an example, it is proved in [38, Theorem 4.3] that, for deterministic transducers, the composition of an attributed tree transducer with a top-down tree transducer can be realized by an attributed tree transducer. That does not immediately imply that the same is true for a dTT and a $\mathrm{dTT}_{\downarrow}$, which we show in Sect. 5, because dTT's are not necessarily total and they have regular look-around. Moreover, we wanted such results also to be understandable for readers unfamiliar with attribute grammars and attributed tree transducers.

The main results of this paper were first presented at FSTTCS '02 [58] (on the complexity of compositions of deterministic MT's), at FSTTCS '03 [59] (on compositions of MT's that realize functions of linear size increase), at FSTTCS '08 [51] (on the complexity of compositions of nondeterministic MT's), at PLAN-X '09 [52] (on the complexity of the membership problem for MT's), and in the Ph.D. Thesis of the second author [48] (on the last two subjects).

2 Preliminaries

Convention All results stated and/or proved in this paper are effective.

Sets, strings, and relations The set of natural numbers is $\mathbb{N} = \{0, 1, 2, \dots\}$. For $m, n \in \mathbb{N}$, we denote the interval $\{k \in \mathbb{N} \mid m \leq k \leq n\}$ by $[m, n]$. The cardinality or size of a set A is denoted by $\#(A)$. The set of strings over A is denoted by A^*. It consists of all sequences $w = a_1 \cdots a_m$ with $m \in \mathbb{N}$ and $a_i \in A$ for every $i \in [1, m]$. The length m of w is denoted by $|w|$. The empty string (of length 0) is denoted by ε. The concatenation of two strings v and w is denoted by $v \cdot w$ or just vw. Moreover, $w^0 = \varepsilon$ and $w^{k+1} = w \cdot w^k$ for $k \in \mathbb{N}$.

The domain and range of a binary relation $R \subseteq A \times B$ are denoted by $\mathrm{dom}(R)$ and $\mathrm{ran}(R)$, respectively. For $A' \subseteq A$, $R(A') = \{b \in B \mid (a, b) \in R \text{ for some } a \in A'\}$. The composition of R with a binary relation $S \subseteq B \times C$ is $R \circ S = \{(a, c) \mid \exists\, b \in B : (a, b) \in R,\ (b, c) \in S\}$. The inverse of R is $R^{-1} = \{(b, a) \mid (a, b) \in R\}$. Note that $\mathrm{dom}(R \circ S) = R^{-1}(\mathrm{dom}(S))$ and $\mathrm{ran}(R \circ S) = S(\mathrm{ran}(R))$. If $A = B$ then the transitive-reflexive closure of R is $R^* = \bigcup_{k \in \mathbb{N}} R^k$ where $R^0 = \{(a, a) \mid a \in A\}$ and $R^{k+1} = R \circ R^k$. The composition of two classes of binary relations $\mathcal{R}$ and $\mathcal{S}$ is $\mathcal{R} \circ \mathcal{S} = \{R \circ S \mid R \in \mathcal{R},\ S \in \mathcal{S}\}$. Moreover, $\mathcal{R}^1 = \mathcal{R}$ and $\mathcal{R}^{k+1} = \mathcal{R} \circ \mathcal{R}^k$ for $k \geq 1$. The relation R is *finitary* if $R(a)$ is finite for every $a \in A$, where $R(a)$ denotes $R(\{a\})$. It is a (partial) function from A to B if $R(a)$ is empty or a singleton for every $a \in A$, and it is a total function if, moreover, $\mathrm{dom}(R) = A$.

Trees An alphabet is a finite set of symbols. A *ranked* alphabet Σ is an alphabet together with a mapping $\mathrm{rank}_\Sigma : \Sigma \to \mathbb{N}$ (of which the subscript Σ will be dropped when it is clear from the context). The maximal rank of elements of Σ is denoted mx_Σ. For every $m \in \mathbb{N}$ we denote by $\Sigma^{(m)}$ the elements of Σ that have rank m.

Trees over Σ are recursively defined to be strings over Σ, as follows. For every $m \in \mathbb{N}$, if $\sigma \in \Sigma^{(m)}$ and $t_1, \dots, t_m$ are trees over Σ, then $\sigma\, t_1 \cdots t_m$ is a tree over Σ. For readability we also write the tree $\sigma\, t_1 \cdots t_m$ as the term $\sigma(t_1, \dots, t_m)$. The set of all trees over Σ is denoted T_Σ; thus $T_\Sigma \subseteq \Sigma^*$. For an arbitrary finite set A, disjoint with Σ, we denote by $T_\Sigma(A)$ the set $T_{\Sigma \cup A}$, where each element of A has rank 0.

As usual trees are viewed as directed labeled graphs. The nodes of a tree t are indicated by Dewey notation, i.e., by elements of $\mathbb{N}^*$, which are strings of natural numbers. The root of t is indicated by the empty string ε, but will also be denoted by root_t for readability. The i-th child of a node u of t is indicated by ui, and there is a directed edge from the parent u to the child ui. Formally, the set $\mathcal{N}(t)$ of nodes of a tree $t = \sigma\, t_1 \cdots t_m$ over Σ can be defined recursively by $\mathcal{N}(t) = \{\varepsilon\} \cup \{iu \mid i \in [1, m],\ u \in \mathcal{N}(t_i)\}$. Thus, $\mathcal{N}(t) \subseteq [1, mx_\Sigma]^*$. The root of $t = \sigma t_1 \cdots t_m$ has label σ, and the node iu of t has the same label as the node u of t_i. The rank of node u is the rank of its label, i.e., the number of its children. A leaf is a node of rank 0, and a monadic node is a node of rank 1. Every node of t has a child number: each node ui has child number i, and the root ε is given child number 0 for technical convenience. For a node u of t the subtree of t with root u is denoted $t|_u$; thus, $t|_\varepsilon = t$ and $t|_{iu} = t_i|_u$. A node v of t is a descendant of a node u of t, and u is an ancestor of v, if there exists $w \in \mathbb{N}^*$

such that $w \neq \varepsilon$ and $v = uw$ (thus, u is *not* a descendant/ancestor of itself). The size of a tree t is $|t|$, i.e., its length as a string. Note that $|t| = \#(\mathcal{N}(t))$ because the nodes of t correspond one-to-one to the positions in the string t, i.e., for every $\sigma \in \Sigma$, each occurrence of σ in t corresponds to a node of t with label σ. The left-to-right linear order on $\mathcal{N}(t)$ according to this correspondence is called the pre-order of the nodes of t. The yield of t is the string of labels of its leaves, in pre-order. The height of t is the number of edges of a longest directed path from the root of t to a leaf; thus, it is the maximal length of its nodes (which are strings over $\mathbb{N}$).

A *tree language* L is a set of trees over Σ, for some ranked alphabet Σ, i.e., $L \subseteq T_\Sigma$. A *tree translation* τ is a binary relation between trees over Σ and trees over Δ, for some ranked alphabets Σ and Δ, i.e., $\tau \subseteq T_\Sigma \times T_\Delta$.

Linear-bounded composition Let Σ, Δ, and Γ be ranked alphabets. For tree translations $\tau_1 \subseteq T_\Sigma \times T_\Delta$ and $\tau_2 \subseteq T_\Delta \times T_\Gamma$, we say that the pair (τ_1, τ_2) is *linear-bounded* if there is a constant $c \in \mathbb{N}$ such that for every $(t, s) \in \tau_1 \circ \tau_2$ there exists $r \in T_\Delta$ such that $(t, r) \in \tau_1$, $(r, s) \in \tau_2$, and $|r| \leq c \cdot |s|$. Thus, the intermediate result r can be chosen such that its size is linear in the size of the output s. Note that if τ_1 and τ_2 are functions, this means that $|r| \leq c \cdot |\tau_2(r)|$ for every $r \in \mathrm{ran}(\tau_1) \cap \mathrm{dom}(\tau_2)$.

For classes $\mathcal{T}_1$ and $\mathcal{T}_2$ of tree translations, we define $\mathcal{T}_1 * \mathcal{T}_2$ to consist of all translations $\tau_1 \circ \tau_2$ such that $\tau_1 \in \mathcal{T}_1$, $\tau_2 \in \mathcal{T}_2$, and (τ_1, τ_2) is linear-bounded.

Lemma 1 *Let $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ be classes of tree translations. Then*

$$\mathcal{T}_1 \circ (\mathcal{T}_2 * \mathcal{T}_3) \subseteq (\mathcal{T}_1 \circ \mathcal{T}_2) * \mathcal{T}_3 \quad \textit{and} \quad (\mathcal{T}_1 * \mathcal{T}_2) * \mathcal{T}_3 \subseteq \mathcal{T}_1 * (\mathcal{T}_2 \circ \mathcal{T}_3).$$

Proof Let $\tau_i \in \mathcal{T}_i$ for $i \in \{1, 2, 3\}$. If the pair (τ_2, τ_3) is linear-bounded then so is the pair $(\tau_1 \circ \tau_2, \tau_3)$, with the same constant c. If (τ_1, τ_2) and $(\tau_1 \circ \tau_2, \tau_3)$ are linear-bounded with constant c_1 and c_2, respectively, then $(\tau_1, \tau_2 \circ \tau_3)$ is linear-bounded with constant $c_1 \cdot c_2$. □

A function $\tau : T_\Sigma \to T_\Delta$ is *of linear size increase* if there is a constant $c \in \mathbb{N}$ such that $|\tau(t)| \leq c \cdot |t|$ for every $t \in \mathrm{dom}(\tau)$. The class of functions of linear size increase will be denoted by LSIF.

Lemma 2 *Let $\tau_1 : T_\Sigma \to T_\Gamma$ and $\tau_2 : T_\Gamma \to T_\Delta$ be functions such that* $\mathrm{ran}(\tau_1) \subseteq \mathrm{dom}(\tau_2)$. *If $\tau_1 \circ \tau_2 \in$ LSIF and (τ_1, τ_2) is linear-bounded, then $\tau_1 \in$ LSIF.*

Proof It follows from $\mathrm{ran}(\tau_1) \subseteq \mathrm{dom}(\tau_2)$ that $\mathrm{dom}(\tau_1 \circ \tau_2) = \mathrm{dom}(\tau_1)$. Since (τ_1, τ_2) is linear-bounded, there is a c such that $|\tau_1(t)| \leq c \cdot |\tau_2(\tau_1(t))|$ for every $t \in \mathrm{dom}(\tau_1)$. Since $\tau_1 \circ \tau_2 \in$ LSIF, there is a c' such that $|\tau_2(\tau_1(t))| \leq c' \cdot |t|$ for every $t \in \mathrm{dom}(\tau_1)$. Hence $|\tau_1(t)| \leq c \cdot c' \cdot |t|$ for every $t \in \mathrm{dom}(\tau_1)$, which means that $\tau_1 \in$ LSIF. □

Grammars and automata Context-free grammars and, in particular, regular tree grammars will be used to define the computations of tree-walking tree transducers, and to define the "regular look-around" used by these transducers. A context-free grammar is specified as a tuple $G = (N, T, \mathcal{S}, R)$, where N is the nonterminal alphabet, T the terminal alphabet (disjoint with N), $\mathcal{S} \subseteq N$ the set of initial nonterminals, and R the finite set of rules, where each rule is of the form $X \to \zeta$ with $X \in N$ and $\zeta \in (N \cup T)^*$. A sentential form of G is a string $v \in (N \cup T)^*$ such that $S \Rightarrow_G^* v$ for some $S \in \mathcal{S}$, where $\Rightarrow_G$ is the usual derivation relation of G: if $X \to \zeta$ is in R, then $v_1 X v_2 \Rightarrow_G v_1 \zeta v_2$ for all $v_1, v_2 \in (N \cup T)^*$. The language $L(G)$ generated by G is the set of all terminal sentential forms, i.e., $L(G) = \{w \in T^* \mid \exists\, S \in \mathcal{S} : S \Rightarrow_G^* w\}$. To formally define the derivation trees of G as ranked trees, we need to subscript its nonterminals with ranks because G can have rules $X \to \zeta_1$ and $X \to \zeta_2$ with $|\zeta_1| \neq |\zeta_2|$. Let $\overline{N}$ be the ranked alphabet consisting of all symbols X_m, of rank m, such that G has a rule $X \to \zeta$ with $|\zeta| = m$. The terminal symbols in T are given rank 0. Then the derivation trees of G are generated by the context-free grammar $G^{\mathrm{der}} = (N', \overline{N} \cup T, \mathcal{S}', R^{\mathrm{der}})$ such that $N' = \{X' \mid X \in N\}$, $\mathcal{S}' = \{S' \mid S \in \mathcal{S}\}$, and if R contains a rule $X \to \zeta$, then R^{der} contains the rule $X' \to X_m\zeta'$ where $m = |\zeta|$ and ζ' is obtained from ζ by changing every nonterminal Y into Y'. Note that we only consider derivation trees that correspond to derivations $S \Rightarrow_G^* w$ with $S \in \mathcal{S}$ and $w \in T^*$. Such a derivation tree has yield w, because when taking the yield of a derivation tree we skip the leaves with label X_0. Moreover, when considering a derivation tree of G, we will disregard the subscripts of the nonterminals and we will say that a node has label X rather than X_m. As an example, if G has the rules $S \to aXYb$, $X \to aY$, $Y \to ba$, and $Y \to \varepsilon$, then G^{der} has the rules $S' \to S_4aX'Y'b$, $X' \to X_2aY'$, $Y' \to Y_2ba$, and $Y' \to Y_0$. The string $aabab$ is generated by G, and the derivation tree $S_4aX_2aY_0Y_2bab = S_4(a, X_2(a, Y_0), Y_2(ba), b)$ is generated by G^{der}; the nodes of this tree are labeled by S, X, Y, a, and b, and its yield is $aabab$.

A context-free grammar is *ε-free* if it does not have ε-rules, i.e., rules $X \to \varepsilon$. We will mainly deal with ε-free context-free grammars.

A context-free grammar G is *finitary* if $L(G)$ is finite. We need the following elementary lemma on finitary context-free grammars.

Lemma 3 *Let $G = (N, T, \mathcal{S}, R)$ be a finitary context-free grammar. For every string $w \in L(G)$ there exists a derivation tree $d \in L(G^{\mathrm{der}})$ such that the yield of d is w and the height of d is at most $\#(N)$.*

Proof Let d be a derivation tree with yield w and suppose that a node u of d and a descendant v of u have the same nonterminal label (disregarding the ranking subscripts). Then the tree d can be pumped in the usual way. But since $L(G)$ is finite, the yield of the pumped tree remains the same. Hence we can remove the pumped part from d. Repeating this, we obtain a derivation tree as required. □

A context-free grammar $G = (N, T, \mathcal{S}, R)$ is *forward deterministic* if $\mathcal{S}$ is a singleton and distinct rules have distinct left-hand sides.[2] Such a grammar generates at most one string in T^* and has at most one derivation tree. If $L(G^{\mathrm{der}}) = \{d\}$, then the height of d is at most $\#(N)$ by Lemma 3.

A *regular tree grammar* is a context-free grammar $G = (N, \Sigma, \mathcal{S}, R)$ such that Σ is a ranked alphabet, and $\zeta \in T_\Sigma(N)$ for every rule $X \to \zeta$ in R. A regular tree grammar generates trees over Σ, i.e., $L(G) \subseteq T_\Sigma$. Note that every regular tree grammar is ε-free. Note also that for every context-free grammar G, the grammar G^{der} is a regular tree grammar. If, in particular, G is itself a regular tree grammar, as above, then it should be noted that the elements of Σ all have rank 0 in G^{der}. As an example, if G has the rules $S \to \sigma(X, Y)$, $X \to \tau(Y)$, $Y \to \tau(a)$, and $Y \to a$, where σ, τ, and a have ranks 2, 1 and 0, respectively, then G^{der} has the rules $S' \to S_3(\sigma, X', Y')$, $X' \to X_2(\tau, Y')$, $Y' \to Y_2(\tau, a)$, and $Y' \to Y_1(a)$. The tree $\sigma(\tau(\tau(a)), a)$ is generated by G, and the derivation tree $S_3(\sigma, X_2(\tau, Y_2(\tau, a)), Y_1(a))$ by G^{der}.

2 That is as opposed to a "backward deterministic" context-free grammar in which distinct rules have distinct right-hand sides, see, e.g., [26]. A forward deterministic context-free grammar that generates a string is also called a "straight-line" context-free grammar.

A (total deterministic) *bottom-up finite-state tree automaton* is specified as a tuple $A = (\Sigma, P, F, \delta)$ where Σ is a ranked alphabet, P is a finite set of states, $F \subseteq P$ is the set of final states, and δ is the state transition function such that $\delta(\sigma, p_1, \ldots, p_m) \in P$ for every $\sigma \in \Sigma$ and $p_1, \ldots, p_m \in P$, where m is the rank of σ. For every $t \in T_\Sigma$, we define the state $\delta(t)$ in which A arrives at the root of t recursively by $\delta(\sigma\, t_1 \cdots t_m) = \delta(\sigma, \delta(t_1), \ldots, \delta(t_m))$. The tree language recognized by A is $L(A) = \{t \in T_\Sigma \mid \delta(t) \in F\}$.

A *regular tree language* is a set of trees that can be generated by a regular tree grammar, or equivalently, recognized by a bottom-up finite-state tree automaton. The class of regular tree languages will be denoted by REGT. The basic properties of regular tree languages can be found in, e.g., [11,19,43,44].

Regular look-around Let Σ be a ranked alphabet. A *node test over* Σ is a set of trees over Σ with a distinguished node, i.e., it is a subset of the set

$$T_\Sigma^\bullet = \{(t, u) \mid t \in T_\Sigma,\ u \in \mathcal{N}(t)\}.$$

Intuitively it is a property of a node of a tree.

We introduce a new ranked alphabet $\Sigma \times \{0, 1\}$, such that the rank of (σ, b) equals that of σ in Σ. For a tree t over Σ and a node u of t we define $\mathrm{mark}(t, u)$ to be the tree over $\Sigma \times \{0, 1\}$ that is obtained from t by changing the label σ of u into $(\sigma, 1)$ and changing the label σ of every other node into $(\sigma, 0)$. Thus, $\mathrm{mark}(t, u)$ is t with one "marked" node u. A *regular (node) test over* Σ is a node test $T \subseteq T_\Sigma^\bullet$ such that its marked representation is a regular tree language, i.e., $\mathrm{mark}(T) \in \mathsf{REGT}$. Note that $\varnothing$ and $T_\Sigma^\bullet$ are regular tests, and that the class of regular tests over Σ is closed under the boolean operations complement, intersection, and union, because REGT is closed under those operations. Hence every boolean combination of regular tests is again a regular test.

For a tree language $L \subseteq T_\Sigma$ we define the node test

$$T(L) = \{(t, u) \in T_\Sigma^\bullet \mid t|_u \in L\}$$

over Σ. Intuitively it is a property of the distinguished node that only depends on the subtree at that node. Clearly, if L is regular then $T(L)$ is regular. A regular test of the form $T(L)$ with $L \in \mathsf{REGT}$ will be called a *regular sub-test*. Note that $T(T_\Sigma) = T_\Sigma^\bullet$ and $T(\varnothing) = \varnothing$. Note also that for regular tree languages L and L' over Σ, $T(L) \cap T(L') = T(L \cap L')$ and $T_\Sigma^\bullet \setminus T(L) = T(T_\Sigma \setminus L)$. This shows that the class of regular sub-tests over Σ is also closed under the boolean operations complement, intersection, and union.

For a given node test T over Σ, we also wish to be able to apply T to a node v of a tree $\mathrm{mark}(t, u)$, where v need not be equal to u. Thus, we define the node test $\mu(T)$ over $\Sigma \times \{0, 1\}$ to consist of all $(\mathrm{mark}(t, u), v)$ such that $(t, v) \in T$ and $u \in \mathcal{N}(t)$. The test $\mu(T)$ just disregards the marking of t. It is easy to see that if T is regular, then so is $\mu(T)$.

The reader familiar with monadic second-order logic (abbreviated MSO logic) should realize that it easily follows from the result of Doner, Thatcher and Wright [18,73] that a node test is regular if and only if it is MSO definable (see [6, Lemma 7]). A node test T over Σ is MSO definable if there is an MSO formula $\varphi(x)$ over Σ, with one free variable x, such that $T = \{(t, u) \mid t \models \varphi(u)\}$, where $t \models \varphi(u)$ means that the formula $\varphi(x)$ holds in t for the node u as value of x. The formulas of MSO logic on trees over Σ use the atomic formulas $\mathrm{lab}_\sigma(x)$ and $\mathrm{down}_i(x, y)$, for every $i \in [1, mx_\Sigma]$, meaning that node x has label $\sigma \in \Sigma$, and that y is the i-th child of x, respectively. In the literature, regular tests are also called MSO tests.

3 Tree-walking tree transducers

In this section we define tree-walking tree transducers, with and without regular look-around, and discuss some of their properties.

A *tree-walking tree transducer* (with regular look-around), in short TT, is a finite state device with one reading head that walks from node to node over its input tree following the edges in either direction. In addition to testing the label and child number of the current node, it can even test any regular property of that node. The output tree is produced recursively, in a top-down fashion. When the transducer produces a node of the output tree, labeled by an output symbol of rank k, it branches into k copies of itself, which then proceed independently, in parallel, to produce the subtrees rooted at the children of that output node.

The TT is specified as a tuple $M = (\Sigma, \Delta, Q, Q_0, R)$, where Σ and Δ are ranked alphabets of input and output symbols, Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, and R is a finite set of rules. The rules are divided into *move rules* and *output rules*. Each move rule is of the form $\langle q, \sigma, j, T\rangle \to \langle q', \alpha\rangle$ such that $q, q' \in Q$, $\sigma \in \Sigma$, $j \in [0, mx_\Sigma]$, T is a regular test over Σ (specified in some effective way), and α is one of the following *instructions*:

$$\begin{array}{ll} \text{stay}, & \\ \text{up} & \text{provided } j \neq 0, \text{ and} \\ \text{down}_i & \text{with } 1 \leq i \leq \text{rank}_\Sigma(\sigma). \end{array}$$

Each output rule is of the form $\langle q, \sigma, j, T\rangle \to \delta(\langle q_1, \alpha_1\rangle, \dots, \langle q_k, \alpha_k\rangle)$ such that the left-hand side is as above, $\delta \in \Delta^{(k)}$, $q_1, \dots, q_k \in Q$, and $\alpha_1, \dots, \alpha_k$ are instructions as above. A rule $\langle q, \sigma, j, T\rangle \to \zeta$ with $T = T_\Sigma^\bullet$ will be written $\langle q, \sigma, j\rangle \to \zeta$. The TT M is *deterministic*, in short a dTT, if Q_0 is a singleton, and $T \cap T' = \varnothing$ for every two distinct rules $\langle q, \sigma, j, T\rangle \to \zeta$ and $\langle q, \sigma, j, T'\rangle \to \zeta'$ in R. A dTT with initial state q_0 will be specified as $M = (\Sigma, \Delta, Q, q_0, R)$.

A *configuration* $\langle q, u\rangle$ of the TT M on a tree t over Σ is given by the current state q of M and the current position u of the head of M on t. Formally, $q \in Q$ and $u \in \mathcal{N}(t)$. The set of all configurations of M on t is denoted $\text{Con}(t)$, i.e., $\text{Con}(t) = Q \times \mathcal{N}(t)$. A rule $\langle q, \sigma, j, T\rangle \to \zeta$ is *applicable* to a configuration $\langle q', u\rangle$ of M on t if $q' = q$ and u satisfies the tests σ, j, and T, i.e., σ and j are the label and child number of u, and $(t, u) \in T$. For a node u of t and an instruction α we define the node $\alpha(u)$ of t as follows: if α is stay, up, or down$_i$, then $\alpha(u)$ equals u, is the parent of u, or is the i-th child of u, respectively.

For every input tree $t \in T_\Sigma$ we define the regular tree grammar $G_{M,t} = (N, \Delta, \mathcal{S}, R_{M,t})$ where $N = \text{Con}(t)$, $\mathcal{S} = \{\langle q_0, \text{root}_t\rangle \mid q_0 \in Q_0\}$ and $R_{M,t}$ is defined as follows. Let $\langle q, u\rangle$ be a configuration of M on t and let $\langle q, \sigma, j, T\rangle \to \zeta$ be a rule of M that is applicable to $\langle q, u\rangle$. If $\zeta = \langle q', \alpha\rangle$ then $R_{M,t}$ contains the rule $\langle q, u\rangle \to \langle q', \alpha(u)\rangle$, and if $\zeta = \delta(\langle q_1, \alpha_1\rangle, \dots, \langle q_k, \alpha_k\rangle)$ then $R_{M,t}$ contains the rule $\langle q, u\rangle \to \delta(\langle q_1, \alpha_1(u)\rangle, \dots, \langle q_k, \alpha_k(u)\rangle)$. The derivation relation $\Rightarrow_{G_{M,t}}$ will be written as $\Rightarrow_{M,t}$. The *translation realized by* M, denoted τ_M, is defined as $\tau_M = \{(t, s) \in T_\Sigma \times T_\Delta \mid s \in L(G_{M,t})\}$. In other words, $\tau_M = \{(t, s) \in T_\Sigma \times T_\Delta \mid \exists\, q_0 \in Q_0 : \langle q_0, \text{root}_t\rangle \Rightarrow^*_{M,t} s\}$. Two TT's M and N are *equivalent* if $\tau_M = \tau_N$.

The *domain of* M, denoted by $\text{dom}(M)$, is defined to be the domain of the translation τ_M, i.e., $\text{dom}(M) = \text{dom}(\tau_M) = \{t \in T_\Sigma \mid \exists\, s \in T_\Delta : (t, s) \in \tau_M\}$. The TT M is *total* if $\text{dom}(M) = T_\Sigma$.

The TT M is *finitary* if τ_M is finitary, which means that $\tau_M(t)$ is finite (or equivalently, that $G_{M,t}$ is finitary) for every input tree $t \in T_\Sigma$. All classical top-down tree transducers (with or without regular look-ahead) and all macro tree transducers are finitary.

If M is deterministic, then at most one rule of M is applicable to a given configuration. Hence $G_{M,t}$ is forward deterministic and $L(G_{M,t})$ is either empty or a singleton. Thus, τ_M is a partial function from T_Σ to T_Δ (and a total function if M is total). For every $(t, s) \in \tau_M$ the context-free grammar $G_{M,t}$ has exactly one derivation tree, with root label $\langle q_0, \mathrm{root}_t\rangle$ and yield s.

Intuitively, the derivation relation $\Rightarrow_{M,t}$ of the grammar $G_{M,t}$ formalizes the computation steps of the TT M on the input tree t, the derivations of $G_{M,t}$ are the sequential computations of M on t, and the derivation trees of $G_{M,t}$, generated by the regular tree grammar $G^{\mathrm{der}}_{M,t}$, model the parallel computations of the independent copies of M on t. If M is deterministic and $t \in \mathrm{dom}(M)$, then M has exactly one parallel computation on t.

A sentential form of $G_{M,t}$ will be called an *output form* of M on t. It is a tree $s \in T_\Delta(\mathrm{Con}(t))$ such that $\langle q_0, \mathrm{root}_t\rangle \Rightarrow^*_{M,t} s$ for some $q_0 \in Q_0$. Intuitively, such an output form s consists on the one hand of Δ-labeled nodes that were produced by M previously in the computation, using output rules, and on the other hand of leaves that represent the independent copies of M into which the computation has branched previously, due to those output rules, where each leaf is labeled by the current configuration of that copy. An output form is *initial* if it is the configuration $\langle q_0, \mathrm{root}_t\rangle$ for some $q_0 \in Q_0$, where root_t is the root of t, and it is *final* if it is in T_Δ, which means that all copies of M have disappeared.

Intuitively, the computation steps of M lead from one output form to another, as follows. Let s be an output form and let v be a leaf of s with label $\langle q, u\rangle \in \mathrm{Con}(t)$. If $\langle q, u\rangle \to \langle q', \alpha(u)\rangle$ is a rule of $G_{M,t}$, resulting from a move rule $\langle q, \sigma, j, T\rangle \to \langle q', \alpha\rangle$ of M that is applicable to configuration $\langle q, u\rangle$, as defined above, then $s \Rightarrow_{M,t} s'$ where s' is obtained from s by changing the label of v into $\langle q', \alpha(u)\rangle$. Thus, this copy of M just changes its configuration. Moreover, if $\langle q, u\rangle \to \delta(\langle q_1, \alpha_1(u)\rangle, \ldots, \langle q_k, \alpha_k(u)\rangle)$ is a rule of $G_{M,t}$, resulting from an output rule $\langle q, \sigma, j, T\rangle \to \delta(\langle q_1, \alpha_1\rangle, \ldots, \langle q_k, \alpha_k\rangle)$ of M, as defined above, then $s \Rightarrow_{t,M} s'$ where s' is obtained from s by changing the label of v into δ and adding children $v1, \ldots, vm$ with labels $\langle q_1, \alpha_1(u)\rangle, \ldots, \langle q_m, \alpha_m(u)\rangle$, respectively. Thus, M outputs δ, and for each child vi it branches into a new process, a copy of itself started in state q_i at the node $\alpha_i(u)$. In the particular case that $k = 0$, s' is obtained from s by changing the label of v into δ; thus, the copy of M corresponding to the node v of s disappears. The translation τ_M realized by M consists of all pairs of trees t over Σ and s over Δ such that M has a sequential computation on t that starts with an initial output form and ends with the final output form s.

Before giving an example of a tree-walking tree transducer, we define six properties of TT's that will be used throughout this paper.

The TT M is *sub-testing*, abbreviated TT^{s}, if the regular tests used by M are regular subtests, i.e., only test the subtree at the current node. Formally, for every rule $\langle q, \sigma, j, T\rangle \to \zeta$ there is a regular tree language L over Σ such that $T = T(L)$. Recall that $T(L) = \{(t, u) \mid t|_u \in L\}$. Thus, informally, M is sub-testing if it uses regular look-ahead rather than the more general regular look-around.

The TT M is *local*, abbreviated TT^{ℓ}, if it does not use regular tests, i.e., $T = T^\bullet_\Sigma$ $(= \{(t, u) \mid t \in T_\Sigma, u \in \mathcal{N}(t)\})$ for every rule $\langle q, \sigma, j, T\rangle \to \zeta$. So all its rules are written $\langle q, \sigma, j\rangle \to \zeta$. Recall that $T^\bullet_\Sigma = T(T_\Sigma)$; thus, every local TT is sub-testing. Note that in the formalism of the (non-local) TT the tests on σ and j could be dropped from a rule $\langle q, \sigma, j, T\rangle \to \zeta$, because they can be incorporated in the regular test T.

The TT M is *top-down*, abbreviated $\mathrm{TT}_\downarrow$, if it does not use the up-instruction in the right-hand sides of its rules. Due to the use of stay-instructions, a $\mathrm{TT}_\downarrow$ need not be finitary. It is straightforward to show that the finitary (deterministic) $\mathrm{TT}^{\mathrm{s}}_\downarrow$ and $\mathrm{TT}^{\ell}_\downarrow$ are equivalent to the classical nondeterministic (deterministic) top-down tree transducer, with and without regular look-ahead, respectively; see the end of this section. Note that in the rules of a $\text{TT}^{\text{s}}_{\downarrow}$ or $\text{TT}^{\ell}_{\downarrow}$ the test on the child number j could be dropped, because j can be stored in the finite state if necessary.

The TT M is *single-use*, abbreviated TT_{su}, if it is deterministic and never visits a node of the input tree twice in the same state. Formally, it should satisfy the following property: for every $t \in T_\Sigma$, $s' \in T_\Delta(\text{Con}(t))$, $s \in T_\Delta$, and $\langle q, u\rangle \in \text{Con}(t)$, if $\langle q_0, \text{root}_t\rangle \Rightarrow^*_{M,t} s' \Rightarrow^*_{M,t} s$ then $\langle q, u\rangle$ occurs at most once in s'. In other words, for every $t \in \text{dom}(M)$, no nonterminal occurs twice in the (unique) derivation tree d of the context-free grammar $G = G_{M,t}$. Note that, as discussed in the proof of Lemma 3 (and the paragraph following it), the configuration $\langle q, u\rangle$ cannot occur at two distinct nodes on a path from the root of d to a leaf. The single-use property also forbids $\langle q, u\rangle$ to occur at two independent nodes of d. It was introduced for attribute grammars in [40,41,45].

The TT M is *pruning*, abbreviated TT_{pru}, if it is a top-down TT of which each move rule is of the form $\langle q, \sigma, j, T\rangle \to \langle q', \text{down}_i\rangle$, and each output rule is of the form $\langle q, \sigma, j, T\rangle \to \delta(\langle q_1, \text{down}_{i_1}\rangle, \dots, \langle q_k, \text{down}_{i_k}\rangle)$ such that $1 \le i_1 < \cdots < i_k \le \text{rank}(\sigma)$. Intuitively, a pruning TT is a $\text{TT}_\downarrow$ without stay-instructions that, when arriving at an input node u, either removes u and all its children except one (together with the descendants of those children), or relabels u and possibly removes some of its children (together with their descendants). Since a TT_{pru} does not use the stay-instruction, it is finitary (and single-use if it is deterministic). Every $\text{TT}^{\text{s}}_{\text{pru}}$ and $\text{TT}^{\ell}_{\text{pru}}$ is equivalent to a classical *linear* top-down tree transducer, with and without regular look-ahead, but not vice versa because the latter transducer can generate an arbitrary finite number of output nodes at each computation step, rather than zero or one.

The TT M is *relabeling*, abbreviated TT_{rel}, if every rule of M is an output rule of the form $\langle q, \sigma, j, T\rangle \to \delta(\langle q_1, \text{down}_1\rangle, \dots, \langle q_m, \text{down}_m\rangle)$ where $m = \text{rank}_\Sigma(\sigma) = \text{rank}_\Delta(\delta)$. Thus, the label σ is replaced by the label δ. Obviously, every relabeling TT is pruning.

We use the notation TT for the class of translations realized by tree-walking tree transducers, and fTT and dTT for the subclasses realized by finitary and deterministic TT's, respectively. Thus, $\mathsf{dTT} \subseteq \mathsf{fTT} \subseteq \mathsf{TT}$. The subclasses of TT, fTT, and dTT realized by TT's with the above six properties (and their combinations) are indicated by the superscripts 's' and 'ℓ', and the subscripts '$\downarrow$', 'su', 'pru', and 'rel', as above. For instance, $\mathsf{dTT}^{\text{s}}_{\downarrow}$ denotes the class of translations realized by deterministic tree-walking tree transducers that are both sub-testing and top-down. Note that TT^{ℓ} is a proper subclass of TT^{s}, because a local TT of which all output symbols have rank 0 can be viewed as a tree-walking automaton, which cannot recognize all regular tree languages by the result of [9].

By [14, Section 8.4], the TT is equivalent to the MS tree-walking transducer of [14, Section 8.2]. As discussed in the Introduction, the TT^{ℓ} generalizes the attributed tree transducer of [38], which is required to be noncircular and hence finitary; the deterministic attributed tree transducer is also required to be total.[3] In the same way the deterministic TT generalizes the (deterministic) attributed tree transducer with look-ahead of [7]. In [26] all tree-walking tree transducers are local.

Example 4 Let $\Sigma = \{\sigma, e\}$ with $\text{rank}_\Sigma(\sigma) = 2$ and $\text{rank}_\Sigma(e) = 0$, and let $\Delta = \{\sigma, e\} \cup [1, mx_\Sigma]$ with $\text{rank}_\Delta(\sigma) = 2$, $\text{rank}_\Delta(e) = 0$, and $\text{rank}_\Delta(j) = 0$ for every $j \in [1, mx_\Sigma] = \{1, 2\}$. Moreover, let T be an arbitrary regular node test over Σ. For simplicity we assume that

[3] The TT M is *circular* if there exist $t \in T_\Sigma$, $u \in \mathcal{N}(t)$, $q \in Q$, and $s \in T_\Delta(\text{Con}(t))$ such that $\langle q, u\rangle \Rightarrow^*_{M,t} s$ and $\langle q, u\rangle$ occurs in s. Thus, M is noncircular if and only if $G_{M,t}$ is nonrecursive for every $t \in T_\Sigma$, which implies that $L(G_{M,t})$ is finite. Note that a total deterministic TT is noncircular if and only if for every $t \in T_\Sigma$, $u \in \mathcal{N}(t)$, and $q \in Q$ there exists $s \in T_\Delta$ such that $\langle q, u\rangle \Rightarrow^*_{M,t} s$. It can be shown that for every finitary TT there is an equivalent noncircular TT, but that will not be needed in this paper.

T is not satisfied at the leaves of t, i.e., if $(t,u) \in T$ then u is not a leaf of t. For instance, T consists of all $(t,u) \in T_\Sigma^\bullet$ such that u has at least one ancestor that has exactly one child that is a leaf, and at least one descendant with that same property. We consider a total deterministic TT $M = (\Sigma, \Delta, Q, q_0, R)$ that performs T as a query, i.e., for every input tree t it outputs all nodes of t that satisfy T, in pre-order. More precisely, if $u_1, \ldots, u_n$ are the nodes u of t such that $(t,u) \in T$, in pre-order, then M outputs the tree $s = \sigma(s_1, \sigma(s_2, \ldots \sigma(s_n, e)\cdots))$ where $s_i = \sigma(\cdots\sigma(\sigma(e, j_1), j_2)\ldots, j_k)$ if $u_i = j_1 j_2 \cdots j_k$ with $j_1, j_2, \ldots, j_k \in [1, mx_\Sigma]$. Note that the yield of s is $eu_1eu_2\cdots eu_ne$. The transducer M performs a left-to-right depth-first traversal of the input tree t and applies the test T to every node of t, in pre-order. Whenever M finds a node u_i that satisfies the test, it branches into two copies. The first copy outputs the tree s_i with yield eu_i, walking from u to the root, and the second copy continues the traversal.

Formally, M has the set of states $Q = \{d, u_1, u_2, p, p'\}$ and initial state $q_0 = d$. Intuitively, d stands for 'down', u_j for 'up from the j-th child', and p for 'print'. It has the following rules, where $j' \in [0, mx_\Sigma]$, $j \in [1, mx_\Sigma]$, $T^c = T_\Sigma^\bullet \setminus T$, and $\tau \in \Sigma$:

$$\begin{array}{llll}
\langle d, \sigma, j', T^c\rangle \to \langle d, \mathrm{down}_1\rangle & & \langle d, e, j\rangle \to \langle u_j, \mathrm{up}\rangle \\
\langle d, \sigma, j', T\rangle \to \sigma(\langle p, \mathrm{stay}\rangle, \langle d, \mathrm{down}_1\rangle) & & \langle d, e, 0\rangle \to e \\
\langle u_1, \sigma, j'\rangle \to \langle d, \mathrm{down}_2\rangle & & \langle p, \tau, j\rangle \to \sigma(\langle p, \mathrm{up}\rangle, j) \\
\langle u_2, \sigma, j\rangle \to \langle u_j, \mathrm{up}\rangle & & \langle p, \tau, 0\rangle \to e \\
\langle u_2, \sigma, 0\rangle \to e & &
\end{array}$$

where the rule $\langle p, \tau, j\rangle \to \sigma(\langle p, \mathrm{up}\rangle, j)$ abbreviates the two rules

$$\langle p, \tau, j\rangle \to \sigma(\langle p, \mathrm{up}\rangle, \langle p', \mathrm{stay}\rangle) \quad \text{and} \quad \langle p', \tau, j\rangle \to j.$$

The TT M does not have any of the six properties defined above. Note that M is not single-use because it pays n visits to the root of t in state p. For the example test T it is not clear whether there is a local TT equivalent to M, but that does not seem likely. □

Example 5 Let $\Sigma = \{\sigma, e\}$ as in Example 4. We consider a total deterministic local TT M_{exp} that translates each tree t with n leaves into the full binary tree of height n with 2^n leaves. As in Example 4, it performs a depth-first left-to-right traversal of t, and branches into two copies whenever it visits a leaf of t. Formally, $M_{\mathrm{exp}} = (\Sigma, \Sigma, Q, q_0, R)$ with $Q = \{d, u_1, u_2, q\}$ and $q_0 = d$. Its rules are similar to those of M in Example 4. In particular, the three rules for states u_1 and u_2 are the same. The rules for state d are the following, with $j' \in [0, mx_\Sigma]$ and $j \in [1, mx_\Sigma]$:

$$\begin{array}{l}
\langle d, \sigma, j'\rangle \to \langle d, \mathrm{down}_1\rangle \\
\langle d, e, j\rangle \to \sigma(\langle u_j, \mathrm{up}\rangle, \langle u_j, \mathrm{up}\rangle) \\
\langle d, e, 0\rangle \to \sigma(e, e)
\end{array}$$

where the last rule abbreviates the two rules $\langle d, e, 0\rangle \to \sigma(\langle q, \mathrm{stay}\rangle, \langle q, \mathrm{stay}\rangle)$ and $\langle q, e, 0\rangle \to e$. □

An elementary property of the translation realized by a deterministic TT is that it is of "linear size-height increase", as stated in the next lemma. Since the size of a tree is at most exponential in its height, this implies that it is of exponential size increase. This is well known for attributed tree transducers [38, Lemma 4.1] (see also [39, Lemma 5.40]) and for local TT's [31, Lemma 7], and obviously also holds for TT's. If, moreover, the TT is single-use, then it is of linear size increase.

Lemma 6 *For every $\tau \in \mathsf{dTT}$ there is a constant c such that for every $(t, s) \in \tau$ the height of s is at most $c \cdot |t|$. Moreover,* $\mathsf{dTT}_{\mathrm{su}} \subseteq \mathsf{LSIF}$.

Proof Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a dTT and let $(t, s) \in \tau_M$. Let d be the unique derivation tree generated by $G^{\mathrm{der}}_{M,t}$. Clearly, since each rule of M outputs at most one node of s, the height of s is at most the height of d. By Lemma 3 the height of d is at most $\#(\mathrm{Con}(t))$, which equals $\#(Q) \cdot |t|$. Thus, we can take $c = \#(Q)$.

It should also be clear that the size of s is at most the number of nodes of d that are labeled by a configuration. If M is single-use, then no configuration occurs twice in d. Hence $|s| \leq \#(Q) \cdot |t|$, i.e., the function τ_M is of linear size increase. □

Example 5 and Lemma 6 imply that compositions of deterministic TT's form a proper hierarchy. This was proved for attributed tree transducers in [38, Corollary 4.1] (see also [39, Theorem 5.45]), and the proof for TT's is exactly the same.

Proposition 7 *For every $k \geq 1$,* $\mathsf{dTT}^k \subsetneq \mathsf{dTT}^{k+1}$.

Proof Let τ_{exp} be the translation realized by the dTT M_{exp} of Example 5. Then $\tau_{\mathrm{exp}} \circ \tau_{\mathrm{exp}}$ translates each tree t with n leaves into the full binary tree of height 2^n with 2^{2^n} leaves. Since $|t| = 2n - 1$, it follows from Lemma 6 that $\tau_{\mathrm{exp}} \circ \tau_{\mathrm{exp}}$ is not in dTT. Hence $\mathsf{dTT} \subsetneq \mathsf{dTT}^2$. In a similar way it can be shown that $\tau_{\mathrm{exp}}^{k+1}$ is not in dTT^k. Since the size of a tree is at most exponential in its height, it follows from Lemma 6 that for every $\tau \in \mathsf{dTT}^2$ there is a constant c such that for every $(t, s) \in \tau$ the height of s is at most $2^{c \cdot |t|}$. Similarly for $\tau \in \mathsf{dTT}^k$, the height of s is at most $(k-1)$-fold exponential in $|t|$. □

Thus, in terms of size increase, a composition of k dTT's can create at most a k-fold exponentially large output tree, whereas a composition of $k + 1$ dTT's can naturally create an output tree of $(k + 1)$-fold exponential size. In Sect. 7 we will prove that compositions of nondeterministic TT's also form a hierarchy, with the same counter-examples. One of our aims is to show that these hierarchies collapse for functions of linear size increase, i.e., that $\mathsf{TT}^k \cap \mathsf{LSIF} \subseteq \mathsf{dTT}$ for every $k \geq 1$.

We end this section by discussing some syntactic properties of TT's. First, for an arbitrary TT it may always be assumed that its output rules only use the stay-instruction: an output rule $\langle q, \sigma, j, T\rangle \to \delta(\langle q_1, \alpha_1\rangle, \ldots, \langle q_k, \alpha_k\rangle)$ can be replaced by the output rule $\langle q, \sigma, j, T\rangle \to \delta(\langle p_1, \mathrm{stay}\rangle, \ldots, \langle p_k, \mathrm{stay}\rangle)$ and the move rules $\langle p_i, \sigma, j, T\rangle \to \langle q_i, \alpha_i\rangle$ for every $i \in [1, k]$, where $p_1, \ldots, p_k$ are new states. This replacement preserves determinism and the sub-testing, local, top-down, and single-use properties (but not pruning or relabeling).

Second, we may always assume that the regular tests of a TT are disjoint. For a TT M, let $\mathcal{T}_M$ be the set of regular tests in the left-hand sides of the rules of M.

Lemma 8 *For every* TT *M there is an equivalent* TT *M' such that the tests in $\mathcal{T}_{M'}$ are mutually disjoint. The construction preserves determinism and the sub-testing, local, top-down, single-use, pruning, and relabeling properties.*

Proof If $T, T' \in \mathcal{T}_M$ and $T \cap T' \neq \varnothing$, then every rule $\langle q, \sigma, j, T\rangle \to \zeta$ can be replaced by the two rules $\langle q, \sigma, j, T \cap T'\rangle \to \zeta$ and $\langle q, \sigma, j, T \setminus T'\rangle \to \zeta$. The transducer M' is obtained by repeating this procedure. □

Third, we can extend the definition of a TT $M = (\Sigma, \Delta, Q, q_0, R)$ by allowing "general rules", which can generate any finite number of output nodes, cf. [31, Lemma 2]. Simple examples of general rules are $\langle p, \tau, j\rangle \to \sigma(\langle p, \text{up}\rangle, j)$ in Example 4 and $\langle d, e, 0\rangle \to \sigma(e, e)$ in Example 5. Formally, a *general rule* is of the form $\langle q, \sigma, j, T\rangle \to \zeta$ such that ζ is a tree in $T_\Delta(Q \times I_{\sigma,j})$, where $I_{\sigma,j}$ is the usual set of instructions: stay, up (provided $j \neq 0$), and down$_i$ with $i \in [1, \text{rank}(\sigma)]$. If this rule is applicable to a configuration $\langle q, u\rangle$ of M on $t \in T_\Sigma$, then $G_{M,t}$ has the rule $\langle q, u\rangle \to \zeta_u$, where ζ_u is obtained from ζ by changing every label $\langle q', \alpha\rangle$ into $\langle q', \alpha(u)\rangle$. It is easy to see that a general rule can be replaced by the set of ordinary rules defined as follows. Let p_u be a new state for every $u \in \mathcal{N}(\zeta)$. Then the rules are $\langle q, \sigma, j, T\rangle \to \langle p_\varepsilon, \text{stay}\rangle$, where ε is the root of ζ, and all rules $\langle p_u, \sigma, j, T\rangle \to \lambda(\langle p_{u1}, \text{stay}\rangle, \dots, \langle p_{uk}, \text{stay}\rangle)$ where λ is the label of u in ζ and k is its rank. The first rule is a move rule that just changes state, and the latter rules output the Δ-labeled nodes of ζ one by one ($\lambda \in \Delta$), and then make the required moves ($\lambda \in Q \times I_{\sigma,j}$). This construction preserves determinism and the sub-testing, local, top-down, and single-use properties. Note that the classical top-down tree transducer has general rules.

If we allow general rules, then the stay-instruction is not needed any more in finitary TT's. Let us say that a TT is *stay-free* if it does not use the stay-instruction in its rules. For every TT M (with general rules) we can construct an equivalent stay-free TT M_{sf} with general rules, *with possibly infinitely many rules* but such that the right-hand sides of rules with the same left-hand side form a regular tree language. If M is finitary, then we can transform M_{sf} into an equivalent stay-free TT with finitely many rules. The construction is as follows, where we may assume that the node tests in $\mathcal{T}_M$ are mutually disjoint, by (the proof of) Lemma 8.

For every left-hand side $\langle q, \sigma, j, T\rangle$ of a rule of $M = (\Sigma, \Delta, Q, Q_0, R)$ we define a regular tree grammar $G_{q,\sigma,j,T}$ that simulates the computations of M, starting in a configuration $\langle q, u\rangle$ to which $\langle q, \sigma, j, T\rangle$ is applicable, without leaving the current node u, i.e., executing stay-instructions only. Its set of nonterminals is $\{\langle q', \text{stay}\rangle \mid q' \in Q\}$ with initial nonterminal $\langle q, \text{stay}\rangle$. Its set of terminals is $\Delta \cup D_{\sigma,j}$, where $D_{\sigma,j} = Q \times (I_{\sigma,j} \setminus \{\text{stay}\})$ each element of which has rank 0. Finally, if $\langle q', \sigma, j, T\rangle \to \zeta$ is a rule of M (with $q' \in Q$ and the same σ, j, and T), then $G_{q,\sigma,j,T}$ has the rule $\langle q', \text{stay}\rangle \to \zeta$.

We now define $M_{\text{sf}} = (\Sigma, \Delta, Q, Q_0, R_{\text{sf}})$ where R_{sf} consists of all general rules $\langle q, \sigma, j, T\rangle \to \zeta$ such that $\zeta \in L(G_{q,\sigma,j,T})$, for every left-hand side $\langle q, \sigma, j, T\rangle$ of a rule of M. Even if M_{sf} has infinitely many rules, it should be clear that (with all the definitions as in the finite case) M_{sf} is equivalent to M.

Note that if M is deterministic, then so is M_{sf}, because $G_{q,\sigma,j,T}$ is forward deterministic and hence $L(G_{q,\sigma,j,T})$ is empty or a singleton. Thus, M_{sf} has finitely many rules.

Assume now that M, and hence M_{sf}, is finitary. Let $\langle q, \sigma, j, T\rangle$ be the left-hand side of a rule of M, and let $D \subseteq D_{\sigma,j}$. If M_{sf} has infinitely many rules $\langle q, \sigma, j, T\rangle \to \zeta$ with $\zeta \in T_\Delta(D)$, then we remove those rules from R_{sf}. In fact, if M_{sf} would have a computation $\langle q_0, \text{root}_t\rangle \Rightarrow^*_{M_{\text{sf}},t} s$ with $q_0 \in Q_0$ in which one of those rules is applied, then it would have a similar computation (with the same q_0 and t, but, in general, another s) in which any other of those rules is applied. Since s contains at least as many occurrences of symbols in Δ as ζ, that would contradict the finitariness of M_{sf}. Removing all these rules, for every $D \subseteq D_{\sigma,j}$, we are left with an equivalent version of M_{sf} with finitely many rules. The construction is effective because $L(G_{q,\sigma,j,T}) \cap T_\Delta(D)$ is a regular tree language and hence its finiteness can be decided.

The above constructions also preserve the sub-testing, local, top-down, and single-use properties. Note that if M is a finitary $\text{TT}^{\text{s}}_{\downarrow}$ or $\text{TT}^{\ell}_{\downarrow}$, then M_{sf} is a classical top-down tree transducer (after incorporating the child number in its finite state), with or without regular look-ahead, respectively.

4 Regular look-around

In this section we discuss some basic properties of TT's with respect to the feature of regular look-around. We start with the simple fact that the domain of a TT can always be restricted to a regular tree language, except when the TT is local.

Lemma 9 *For every* TT M *and every* $L \in$ REGT *there is a* TT M' *such that* $\tau_{M'} = \{(t, s) \in \tau_M \mid t \in L\}$*. The construction preserves determinism and the sub-testing, top-down, single-use, pruning, and relabeling properties.*

Proof The TT M' simulates M, but additionally verifies that the input tree t is in L, by using the regular sub-test $T(L)$ at the root of t. Formally, M' is obtained from M by changing every rule $\langle q_0, \sigma, 0, T\rangle \to \zeta$ into $\langle q_0, \sigma, 0, T \cap T(L)\rangle \to \zeta$, for every initial state q_0. □

In the remainder of this section we show how to separate the regular look-around from a TT, by incorporating it into another TT. We first prove that every TT M can be decomposed into a deterministic relabeling TT N and a local TT M'. The relabeling TT N preprocesses the input tree t by adding to the label of each node u of t the truth values of the regular tests of M at that node. This allows M', during its simulation of M, to inspect the new label of u instead of testing u. The idea is similar to that of removing regular look-ahead in [20, Theorem 2.6]. The translation realized by N is called an MSO relabeling in [7,14] and [29, Section 4].

Lemma 10 $\mathrm{TT} \subseteq \mathrm{dTT}_{\mathrm{rel}} \circ \mathrm{TT}^{\ell}$*, i.e., for every* TT M *there are a deterministic relabeling* TT N *and a local* TT M' *such that* $\tau_N \circ \tau_{M'} = \tau_M$*. The construction preserves determinism, the top-down property, and the pruning property.*

Proof Let $M = (\Sigma, \Delta, Q, Q_0, R)$ be a TT, and let $\mathcal{T}$ be the set of regular tests in the left-hand sides of the rules in R. By Lemma 8 we may assume that the tests in $\mathcal{T}$ are mutually disjoint. Now let $\mathcal{T}_\perp = \mathcal{T} \cup \{\perp\}$ where $\perp$ is the intersection of the complements of the tests in $\mathcal{T}$. Thus, for every $t \in T_\Sigma$ and $u \in \mathcal{N}(t)$, (t, u) belongs to a unique node test in $\mathcal{T}_\perp$. Let $\Sigma \times \mathcal{T}_\perp$ be the ranked alphabet such that $\langle \sigma, T\rangle$ has the same rank as σ.

We define the relabeling TT $N = (\Sigma, \Sigma \times \mathcal{T}_\perp, \{p\}, p, R_N)$ such that for every $\sigma \in \Sigma$, $j \in [0, mx_\Sigma]$, and $T \in \mathcal{T}_\perp$, the output rule

$$\langle p, \sigma, j, T\rangle \to \langle \sigma, T\rangle(\langle p, \mathrm{down}_1\rangle, \ldots, \langle p, \mathrm{down}_m\rangle)$$

is in R_N, where m is the rank of σ. Additionally we define the local TT $M' = (\Sigma \times \mathcal{T}_\perp, \Delta, Q, Q_0, R')$ with the following rules. If $\langle q, \sigma, j, T\rangle \to \zeta$ is a rule in R, then R' contains the rule $\langle q, \langle \sigma, T\rangle, j\rangle \to \zeta$. Note that N is total and deterministic. Also, if M is deterministic, then so is M'. It should be clear that $\tau_{M'}(\tau_N(t)) = \tau_M(t)$ for every $t \in T_\Sigma$, i.e., $\tau_N \circ \tau_{M'} = \tau_M$. □

We will also need a variant of this lemma, for nondeterministic TT's only.

Lemma 11 $\mathrm{TT}^{\mathrm{s}} \subseteq \mathrm{TT}^{\ell}_{\mathrm{rel}} \circ \mathrm{TT}^{\ell}$ *and* $\mathrm{TT}^{\mathrm{s}}_{\mathrm{pru}} \subseteq \mathrm{TT}^{\ell}_{\mathrm{rel}} \circ \mathrm{TT}^{\ell}_{\mathrm{pru}}$.

Proof Let $M = (\Sigma, \Delta, Q, Q_0, R)$ be a sub-testing TT, and let $\mathcal{T}$ be the set of regular tests in the left-hand sides of the rules in R. As in the proof of Lemma 10 we may assume that the tests in $\mathcal{T}$ are mutually disjoint (by Lemma 8), and we define $\mathcal{T}_\perp = \mathcal{T} \cup \{\perp\}$ as in that proof. Let $\mathcal{T}_\perp = \{T(L_1), \ldots, T(L_n)\}$ where $L_1, \ldots, L_n$ are regular tree languages. Clearly, there is a bottom-up finite-state tree automaton $A = (\Sigma, P, F, \delta)$ (where F is irrelevant) and a partition $\{F_1, \ldots, F_n\}$ of P such that for every $t \in T_\Sigma$ and $i \in [1, n]$, $t \in L_i$ if and only if $\delta(t) \in F_i$. We define the local relabeling TT $N = (\Sigma, \Sigma \times \mathcal{T}_\perp, P, P, R_N)$ such that it nondeterministically simulates A top-down. For every $\sigma \in \Sigma$ of rank m, every sequence of states $p_1, \dots, p_m \in P$, and every $j \in [0, mx_\Sigma]$, if $\delta(\sigma, p_1, \dots, p_m) = p \in F_i$, then R_N contains the rule $\langle p, \sigma, j\rangle \to \langle \sigma, T(L_i)\rangle(\langle p_1, \mathrm{down}_1\rangle, \dots, \langle p_m, \mathrm{down}_m\rangle)$. The local TT M' is defined as in the proof of Lemma 10. $\square$

The next lemma is based on the folklore technique of computing the states of a bottom-up finite-state tree automaton that are "successful" at the current node (see, e.g., the proofs of [7, Theorem 10] and [6, Theorem 8]). The lemma shows that every top-down TT is equivalent to one that is sub-testing, and hence to a classical top-down tree transducer with regular look-ahead if it is finitary. It is a slight generalization of the fact that every MSO relabeling can be computed by a top-down tree transducer with regular look-ahead, as shown in [7, Theorem 10] and [31, Theorem 4.4].

Lemma 12 $\mathsf{TT}_\downarrow = \mathsf{TT}^{\mathrm{s}}_\downarrow$. *The construction preserves determinism, pruning, and relabeling.*

Proof Let $M = (\Sigma, \Delta, Q, Q_0, R)$ be a $\mathrm{TT}_\downarrow$ that uses a regular test T over Σ in its rules. For simplicity we first assume that M uses T in each of its rules. Let $A = (\Sigma \times \{0, 1\}, P, F, \delta)$ be a bottom-up finite-state tree automaton that recognizes $\mathrm{mark}(T)$. We identify the symbols $(\sigma, 0)$ and σ; thus, A can also handle trees over Σ. For every tree $t \in T_\Sigma$ and every node $u \in \mathcal{N}(t)$, we define the set $\mathrm{succ}_t(u)$ of *successful states* of A at u to consist of all states $p \in P$ such that A recognizes t when started at u in state p. To be precise, $\mathrm{succ}_t(\mathrm{root}_t) = F$ and if u has label $\sigma \in \Sigma^{(m)}$ and $i \in [1, m]$, then $\mathrm{succ}_t(ui)$ is the set of all states $p \in P$ such that $\delta(\sigma, p_1, \dots, p_{i-1}, p, p_{i+1}, \dots, p_m) \in \mathrm{succ}_t(u)$, where $p_j = \delta(t|_{uj})$, i.e., p_j is the state in which $\mathcal{A}$ arrives at the j-th child of u, for every $j \in [1, m] \setminus \{i\}$. Obviously, $\mathrm{mark}(t, u)$ is recognized by A if and only if $\delta((\sigma, 1), \delta(t|_{u1}), \dots, \delta(t|_{um})) \in \mathrm{succ}_t(u)$.

For every $\sigma \in \Sigma^{(m)}$ and every sequence of states $p_1, \dots, p_m \in P$ let $L_{\sigma, p_1, \dots, p_m}$ be the regular tree language consisting of all trees $\sigma(t_1, \dots, t_m) \in T_\Sigma$ such that $\delta(t_i) = p_i$ for every $i \in [1, m]$. Thus, the regular sub-test $T(L_{\sigma, p_1, \dots, p_m})$ verifies that A arrives at the i-th child of the current node in state p_i for every $i \in [1, m]$.

We construct a sub-testing $\mathrm{TT}_\downarrow$ $M' = (\Sigma, \Delta, Q', Q'_0, R')$ that is equivalent to M. It keeps track of $\mathrm{succ}_t(u)$ in its finite state. Its set of states is $Q' = Q \times \{S \mid S \subseteq P\}$ with set of initial states $Q'_0 = \{(q_0, F) \mid q_0 \in Q_0\}$. The set of rules R' is defined as follows. Let $\langle q, \sigma, j, T\rangle \to \zeta$ be a rule in R, let $S \subseteq P$, and let $p_1, \dots, p_m \in P$ such that $\delta((\sigma, 1), p_1, \dots, p_m) \in S$ where $m = \mathrm{rank}_\Sigma(\sigma)$. Then R' contains the rule $\langle (q, S), \sigma, j, T(L_{\sigma, p_1, \dots, p_m})\rangle \to \zeta'$ where ζ' is obtained from ζ by changing every $\langle q', \mathrm{stay}\rangle$ into $\langle (q', S), \mathrm{stay}\rangle$ and every $\langle q', \mathrm{down}_i\rangle$ into $\langle (q', S_i), \mathrm{down}_i\rangle$ with $S_i = \{p \in P \mid \delta(\sigma, p_1, \dots, p_{i-1}, p, p_{i+1}, \dots, p_m) \in S\}$.

In the general case where M uses regular tests $T_1, \dots, T_n$, the transducer M' must keep track of $\mathrm{succ}_t(u)$ for each of the corresponding bottom-up finite-state tree automata $A_1, \dots, A_n$. $\square$

The proof of Lemma 12 also shows that in a rule $\langle q, \sigma, j, T(L)\rangle \to \zeta$ of a sub-testing $\mathrm{TT}_\downarrow$ we may assume that L is of the form $L = \sigma(L_1, \dots, L_m) = \{\sigma(t_1, \dots, t_m) \mid t_1 \in L_1, \dots, t_m \in L_m\}$ for regular tree languages $L_1, \dots, L_m$ (where $m = \mathrm{rank}(\sigma)$). This is how regular look-ahead is usually defined for classical top-down tree transducers.

By Lemmas 10 and 12, $\mathsf{dTT} \subseteq \mathsf{dTT}^{\mathrm{s}}_\downarrow \circ \mathsf{dTT}^\ell$. It is proved in [28, Lemmas 49 and 50] that even $\mathsf{dTT} \subseteq \mathsf{dTT}^\ell_\downarrow \circ \mathsf{dTT}^\ell$, but this will not be needed in what follows.[4] Using Lemmas 10 and 12 we can now prove three essential properties of TT's, based on well-known results from the literature.

[4] In [28], dTT and dTT^ℓ are denoted by $\mathsf{dTT}^{\mathrm{MSO}}$ and dTT, respectively.

Lemma 13 *The regular tree languages are closed under inverses of* TT *translations, i.e., if* $L \in \mathrm{REGT}$ *and* $\tau \in \mathrm{TT}$, *then* $\tau^{-1}(L) \in \mathrm{REGT}$.

Proof Since the inverse of a composition is the composition of the inverses, it suffices to show this for $\mathrm{dTT}^{\mathrm{s}}_{\mathrm{rel}}$ and TT^{ℓ} by Lemmas 10 and 12. For $\mathrm{dTT}^{\mathrm{s}}_{\mathrm{rel}}$ it follows from [20, Theorem 2.6 and Lemma 1.2], and for TT^{ℓ} it is proved in [26, Lemma 3].[5] □

Corollary 14 *The domain of a* TT M *is regular, i.e.,* $\mathrm{dom}(M) \in \mathrm{REGT}$. *More generally, for every* $k \geq 1$, *if* $\tau \in \mathrm{TT}^k$ *then* $\mathrm{dom}(\tau) \in \mathrm{REGT}$.

Corollary 14 was proved for (nondeterministic) attributed tree transducers in [5], from which it is easy to conclude that Lemma 13 holds for attributed tree transducers, as explained in [26, Lemma 3].

Lemma 15 *The regular tree languages are closed under pruning* TT *translations, i.e., if* $L \in \mathrm{REGT}$ *and* $\tau \in \mathrm{TT}_{\mathrm{pru}}$, *then* $\tau(L) \in \mathrm{REGT}$.

Proof By Lemma 12, $\mathrm{TT}_{\mathrm{pru}} = \mathrm{TT}^{\mathrm{s}}_{\mathrm{pru}}$. As observed before, every $\tau \in \mathrm{TT}^{\mathrm{s}}_{\mathrm{pru}}$ can be realized by a classical linear top-down tree transducer with regular look-ahead. It is well known that, due to linearity, REGT is closed under such translations, see, e.g., [43, Corollary IV.6.7]. □

Lemma 13, Corollary 14 and Lemma 15 are powerful technical tools because they allow us to show that certain node tests of a TT M are regular by defining them in terms of, e.g., the domains of other TT's or of variants of M itself. In other words, a TT can use TT's "to look around". For instance, Lemma 13 is used for this purpose in the proof of Lemma 16 below, where we show the following.

In a composition of a dTT with a sub-testing TT the second transducer can even be assumed to be local, because the first transducer can determine the truth values of the regular sub-tests of the output tree by executing appropriate regular tests on its input tree.

Lemma 16 $\mathrm{dTT} \circ \mathrm{TT}^{\mathrm{s}} \subseteq \mathrm{dTT} \circ \mathrm{TT}^{\ell}$. *The construction preserves determinism (of the second transducer) and the top-down, single-use, pruning, and relabeling properties of both transducers.*

Proof Let $M_1 = (\Sigma, \Delta, Q, q_0, R)$ be a dTT and let M_2 be a sub-testing TT with input alphabet Δ. We will construct a dTT M_1' and a local TT M_2' that simulate the composition of M_1 and M_2. The construction preserves the top-down, single-use, pruning, and relabeling property of each transducer, i.e., if M_1 has one of these properties, then so has M_1', and similarly for M_2 and M_2'. Moreover, if M_2 is deterministic, then so is M_2'.

Let $(t, s) \in \tau_{M_1}$. The dTT M_1' simulates M_1 on the input tree t. Simultaneously it executes the sub-tests of M_2 at every node v of the output tree s and preprocesses s by adding to the label of v the truth values of these sub-tests at v, cf. the text before Lemma 10. This allows M_2', during its simulation of M_2 on s, to inspect the new label of v instead of sub-testing v.

Every node of s is produced by an output rule of M_1 during its computation on t. Let $\bar{s}$ be an output form of M_1 on t, and let v be a leaf of $\bar{s}$ with label $\langle q, u\rangle$. It should be clear that $\langle q, u\rangle \Rightarrow^*_{M_1,t} s|_v$. Now let L be a regular tree language over Δ such that M_2 uses the sub-test $T' = T(L)$. We claim that, in configuration $\langle q, u\rangle$, M_1' can test whether $(s, v) \in T'$

[5] We note that an alternative proof is by Lemma 26 (in Sect. 6) and [34, Theorem 7.4] (see also [65, Section 5]). For the reader familiar with MSO translations, see [14], we note that it is proved in [29, Section 4] that $\mathrm{dTT}^{\mathrm{s}}_{\mathrm{rel}}$ is the class of MSO (tree) relabelings, and that REGT, which is the class of MSO definable tree languages, is closed under inverse MSO (tree) transductions by [14, Corollary 7.12].

by a regular test $\text{inv}_q(T')$. Note that $(s, v) \in T(L)$ if and only if $s|_v \in L$. Thus, $\text{inv}_q(T')$ should test whether the output tree generated by the configuration $\langle q, u\rangle$ is in L. To prove that $\text{mark}(\text{inv}_q(T'))$ is regular, we define a dTT N_q such that $\text{mark}(\text{inv}_q(T')) = \tau_{N_q}^{-1}(L)$ and we use Lemma 13. The transducer N_q first uses a regular test at the root to verify that the input tree is of the form $\text{mark}(t, u)$.[6] After that it walks to the (unique) marked node u, using move rules to execute a depth-first search of the input tree, and then simulates M_1 starting in state q at u, producing the output tree $s|_v$. During that simulation it treats each symbol $(\sigma, 0)$ or $(\sigma, 1)$ as σ, and for each regular test T of M_1 it instead uses the test $\mu(T)$, which is the set of all $(\text{mark}(t, u), v)$ such that $(t, v) \in T$ and $u \in \mathcal{N}(t)$, see Sect. 2.

The construction of M_1' and M_2' is similar to the construction of N and M' in the proof of Lemma 10. Let $\mathcal{T}$ be the set of regular tests in the left-hand sides of the rules of M_2. As in the proof of Lemma 10 we may assume that the tests in $\mathcal{T}$ are mutually disjoint (by Lemma 8), and we define $\mathcal{T}_\perp = \mathcal{T} \cup \{\perp\}$ as in that proof. Note that the elements of $\mathcal{T}_\perp$ are still regular sub-tests. Note also that for every $q \in Q$, $t \in \text{dom}(M_1)$ and $u \in \mathcal{N}(t)$, (t, u) belongs to a unique regular test in $\{\text{inv}_q(T') \mid T' \in \mathcal{T}_\perp\}$.

We define the dTT $M_1' = (\Sigma, \Delta \times \mathcal{T}_\perp, Q, q_0, R')$ such that R' contains all move rules in R, and moreover, if $\langle q, \sigma, j, T\rangle \to \delta(\langle q_1, \alpha_1\rangle, \dots, \langle q_k, \alpha_k\rangle)$ is an output rule in R, then R' contains the rule

$$\langle q, \sigma, j, T \cap \text{inv}_q(T')\rangle \to \langle \delta, T'\rangle(\langle q_1, \alpha_1\rangle, \dots, \langle q_k, \alpha_k\rangle)$$

for every $T' \in \mathcal{T}_\perp$. We define the local TT M_2' with input alphabet $\Delta \times \mathcal{T}_\perp$ and the following rules. If $\langle q, \delta, j, T'\rangle \to \zeta$ is a rule of M_2, then M_2' has the rule $\langle q, \langle \delta, T'\rangle, j\rangle \to \zeta$. It should now be clear that $\tau_{M_2'}(\tau_{M_1'}(t)) = \tau_{M_2}(\tau_{M_1}(t))$ for every $t \in T_\Sigma$, i.e., $\tau_{M_1'} \circ \tau_{M_2'} = \tau_{M_1} \circ \tau_{M_2}$. If M_1 is single-use, then M_1' is also single-use, because M_1' visits the nodes of the input tree in the same states as M_1; the same is true for M_2 and M_2'. Preservation of the other properties easily follows from the construction of M_1' and M_2'. □

5 Composition

In this section we prove three composition results for TT's. Our first aim is to prove that dTT's are closed under right-composition with top-down dTT's, and hence in particular with pruning dTT's. As already mentioned at the end of the Introduction, this generalizes the result of [38, Theorem 4.3] for attributed tree transducers, because dTT's need not be total and they have regular look-around. By Lemma 12 we may assume that the top-down TT is sub-testing. It may even be assumed to be local by Lemma 16.

Lemma 17 $\text{dTT} \circ \text{dTT}_\downarrow^\ell \subseteq \text{dTT}$. *In particular*

$$\text{dTT}_\downarrow \circ \text{dTT}_\downarrow^\ell \subseteq \text{dTT}_\downarrow \quad \textit{and} \quad \text{dTT}_{\text{pru}} \circ \text{dTT}_{\text{pru}}^\ell \subseteq \text{dTT}_{\text{pru}}.$$

Proof Since the domain of a TT can always be restricted to $\text{dom}(M_1)$ by Lemma 9 and Corollary 14, it suffices to show that for every dTT M_1 and every local top-down dTT M_2, a dTT M can be constructed such that $\tau_M(t) = \tau_{M_2}(\tau_{M_1}(t))$ for every input tree $t \in \text{dom}(M_1)$. For the case where M_1 is also local this construction was presented in the proof of [28, Theorem 55], which can easily be adapted to the general case. We repeat it here for completeness sake, and because the proofs of the other two composition closure results will be based on it.

[6] To be precise, the regular sub-test $T(\text{mark}(T_\Sigma^\bullet))$.

The transducer M is obtained by a straightforward product construction. For every $(t, s) \in \tau_{M_1}$, M simulates M_1 on the input tree t until M_1 uses an output rule that generates a node v of s. Then M switches to the simulation of M_2 on v, as long as M_2 executes stay-instructions. When M_2 executes a down_i-instruction, M switches again to the simulation of M_1 in order to generate the i-th child of v.

Formally, let $M_1 = (\Sigma, \Delta, P, p_0, R_1)$ and $M_2 = (\Delta, \Gamma, Q, q_0, R_2)$. To simplify the construction of M we assume that M_1 keeps track in its finite state of the child number of the output node to be generated. To be precise, we assume that there is a mapping $\chi : P \to [0, mx_\Delta]$ such that for every output form s' and every leaf v of s' that is labeled by a configuration $\langle p, u\rangle$, the child number of v in s' is $\chi(p)$. That is possible because the output tree is generated top-down. If M_1 does not satisfy this assumption, then we change M_1 as follows. The new set of states is $P \times [0, mx_\Delta]$, and we define $\chi(p, i) = i$. The new initial state is $(p_0, 0)$, because M_1 starts by generating the root of the output tree. Each move rule $\langle p, \sigma, j, T\rangle \to \langle p', \alpha\rangle$ of M_1 is changed into the rules $\langle (p, i), \sigma, j, T\rangle \to \langle (p', i), \alpha\rangle$ and each output rule $\langle p, \sigma, j, T\rangle \to \delta(\langle p_1, \alpha_1\rangle, \ldots, \langle p_k, \alpha_k\rangle)$ into $\langle (p, i), \sigma, j, T\rangle \to \delta(\langle (p_1, 1), \alpha_1\rangle, \ldots, \langle (p_k, k), \alpha_k\rangle)$, for every $i \in [0, mx_\Delta]$. For the sake of the proof of Lemma 22 we note that this transformation of M_1 preserves the single-use property, because we have only added information to the states of M_1.

The dTT M has input alphabet Σ and output alphabet Γ. Its states are of the form (p, q) or (ρ, q), where $p \in P$, $q \in Q$, and ρ is an output rule of M_1, i.e., a rule of the form $\langle p, \sigma, j, T\rangle \to \delta(\langle p_1, \alpha_1\rangle, \ldots, \langle p_k, \alpha_k\rangle)$. Its initial state is (p_0, q_0). A state (p, q) is used by M to simulate the computation of M_1 that generates the next current node of M_2 when M_2 moves down (keeping the state q of M_2 in memory). Initially M simulates the computation of M_1 that generates the root of the output tree. A state (ρ, q) is used by M to simulate the computation of M_2 on the node that M_1 has generated with rule ρ. The rules of M are defined as follows.

First, rules that simulate M_1. Let $\rho : \langle p, \sigma, j, T\rangle \to \zeta$ be a rule in R_1. If $\zeta = \langle p', \alpha\rangle$, then M has the rules $\langle (p, q), \sigma, j, T\rangle \to \langle (p', q), \alpha\rangle$ for every $q \in Q$. If ρ is an output rule, then M has the rules $\langle (p, q), \sigma, j, T\rangle \to \langle (\rho, q), \text{stay}\rangle$ for every $q \in Q$.

Second, rules that simulate M_2. Let $\langle q, \delta, i\rangle \to \zeta$ be a rule in R_2 and let $\rho\colon \langle p, \sigma, j, T\rangle \to \delta(\langle p_1, \alpha_1\rangle, \ldots, \langle p_k, \alpha_k\rangle)$ be an output rule in R_1, with the same δ and with $\chi(p) = i$. Then M has the rule $\langle (\rho, q), \sigma, j, T\rangle \to \zeta'$ where ζ' is obtained from ζ by changing every $\langle q', \text{stay}\rangle$ into $\langle (\rho, q'), \text{stay}\rangle$, and every $\langle q', \text{down}_\ell\rangle$ into $\langle (p_\ell, q'), \alpha_\ell\rangle$. Note that the test on σ, j, and T is actually superfluous, because that was already tested when M included ρ in its state.

It is easy to see that $\tau_M(t) = \tau_{M_2}(\tau_{M_1}(t))$ for every input tree $t \in \text{dom}(M_1)$. If the rules of M_2 do not contain stay-instructions, then M does not need the states (ρ, q). Its rules can then be simplified as follows. Let $\langle p, \sigma, j, T\rangle \to \zeta$ be a rule in R_1. As above, if $\zeta = \langle p', \alpha\rangle$, then M has the rules $\langle (p, q), \sigma, j, T\rangle \to \langle (p', q), \alpha\rangle$ for every $q \in Q$. If $\zeta = \delta(\langle p_1, \alpha_1\rangle, \ldots, \langle p_k, \alpha_k\rangle)$ and $\langle q, \delta, i\rangle \to \zeta'$ is a rule in R_2, with the same δ and with $\chi(p) = i$, then M has the rule $\langle (p, q), \sigma, j, T\rangle \to \zeta''$ where ζ'' is obtained from ζ' by changing every $\langle q', \text{down}_\ell\rangle$ into $\langle (p_\ell, q'), \alpha_\ell\rangle$. This shows that if both M_1 and M_2 are pruning, then M is pruning too. □

We obtain our first composition closure result from Lemmas 12, 16 and 17. Note that the closure under composition of $\mathsf{dTT}_\downarrow$ already follows from Lemma 12 and [20, Theorem 2.11(2)].

Theorem 18 $\mathsf{dTT} \circ \mathsf{dTT}_\downarrow \subseteq \mathsf{dTT}$. *In particular,* $\mathsf{dTT}_\downarrow$ *and* $\mathsf{dTT}_{\text{pru}}$ *are closed under composition.*

Theorem 18 can be used to show that in a composition of two dTT's we may always assume that the second one is local (thus strengthening Lemma 16): by Lemma 10 the second TT can be decomposed into a top-down TT and a local TT, and then (by Theorem 18), the top-down one can be absorbed by the first TT. Hence $\mathsf{dTT} \circ \mathsf{dTT} \subseteq \mathsf{dTT} \circ \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}^{\ell} \subseteq \mathsf{dTT} \circ \mathsf{dTT}^{\ell}$. This was already proved in [28, Theorem 53] by means of pebble tree transducers.

Our second composition result generalizes Theorem 18 to nondeterministic TT's, restricted to right-composition with pruning TT's. The proof of the next lemma is similar to that of Lemma 17.

Lemma 19 $\mathsf{TT} \circ \mathsf{TT}^{\ell}_{\mathrm{pru}} \subseteq \mathsf{TT}$. *In particular*

$$\mathsf{TT}_{\downarrow} \circ \mathsf{TT}^{\ell}_{\mathrm{pru}} \subseteq \mathsf{TT}_{\downarrow} \quad \textit{and} \quad \mathsf{TT}_{\mathrm{pru}} \circ \mathsf{TT}^{\ell}_{\mathrm{pru}} \subseteq \mathsf{TT}_{\mathrm{pru}}.$$

Proof Let $M_1 = (\Sigma, \Delta, P, P_0, R_1)$ be a TT and $M_2 = (\Delta, \Gamma, Q, Q_0, R_2)$ a local pruning TT. The construction of the transducer M such that $\tau_M = \tau_{M_1} \circ \tau_{M_2}$ is a straightforward variant of the one in the last paragraph of the proof of Lemma 17. This time, we do not verify at the start that the input tree is in the domain of M_1, because it has to be checked at each step of M that M_1 can produce an output tree, in particular when M_2 deletes part of that output tree (cf. the proof of [20, Lemma 2.9]).

We define $M = (\Sigma, \Gamma, P \times Q, P_0 \times Q_0, R)$ as follows. As in the proof of Lemma 17 we assume that M_1 keeps track in its finite state of the child number of the output node to be generated, through a mapping $\chi : P \to [0, mx_\Sigma]$. Let $\langle p, \sigma, j, T\rangle \to \zeta$ be a rule in R_1. As before, if $\zeta = \langle p', \alpha\rangle$, then M has the rules $\langle (p,q), \sigma, j, T\rangle \to \langle (p',q), \alpha\rangle$ for every $q \in Q$. If $\zeta = \delta(\langle p_1, \alpha_1\rangle, \dots, \langle p_k, \alpha_k\rangle)$ and $\langle q, \delta, i\rangle \to \zeta'$ is a rule in R_2, with the same δ and with $\chi(p) = i$, then M has the rule $\langle (p,q), \sigma, j, T \cap T'\rangle \to \zeta''$ where ζ'' is obtained (as before) from ζ' by changing every $\langle q', \mathrm{down}_\ell\rangle$ into $\langle (p_\ell, q'), \alpha_\ell\rangle$, and the node test T' consists of all (t,u) such that for every $\ell \in [1,k]$ there exists a computation $\langle p_\ell, \alpha_\ell(u)\rangle \Rightarrow^*_{M_1,t} s_\ell$ for some $s_\ell \in T_\Delta$. Thus, the only difference with the proof of Lemma 17 is the additional test T'. In fact, it suffices that T' tests every $\ell \in [1,k]$ for which down_ℓ does not occur in ζ'. That guarantees the existence of an output tree of M_1 on which M_2 is simulated by M. It should be clear that T' is regular by Corollary 14: it can be written as $\bigcap_{\ell \in [1,k]} T'_\ell$ where $\mathrm{mark}(T'_\ell)$ is the domain of a TT that walks to node $\alpha_\ell(u)$ and then simulates M_1 starting in state p_ℓ.

We note that this construction does not work for an arbitrary top-down M_2 without stay-instructions. If some down_ℓ occurs twice in ζ', then there are two occurrences $\langle (p_\ell, q'), \alpha_\ell\rangle$ and $\langle (p_\ell, q''), \alpha_\ell\rangle$ in ζ'' and it is not guaranteed (as it should) that from both occurrences the same output subtree of M_1 is generated by M. We finally note that, as in the proof of Lemma 17, if both M_1 and M_2 are pruning, then so is M. □

We obtain our second composition result from Lemma 12, the second inclusion of Lemma 11, and two applications of Lemma 19 (taking into account that $\mathsf{TT}^{\ell}_{\mathrm{rel}} \subseteq \mathsf{TT}^{\ell}_{\mathrm{pru}}$).

Theorem 20 $\mathsf{TT} \circ \mathsf{TT}_{\mathrm{pru}} \subseteq \mathsf{TT}$. *In particular* $\mathsf{TT}_{\downarrow} \circ \mathsf{TT}_{\mathrm{pru}} \subseteq \mathsf{TT}_{\downarrow}$, *and* $\mathsf{TT}_{\mathrm{pru}}$ *is closed under composition.*

Hence, also in a composition of two nondeterministic TT's we may always assume that the second one is local: $\mathsf{TT} \circ \mathsf{TT} \subseteq \mathsf{TT} \circ \mathsf{dTT}_{\mathrm{rel}} \circ \mathsf{TT}^{\ell} \subseteq \mathsf{TT} \circ \mathsf{TT}^{\ell}$ by Lemma 10 and Theorem 20, respectively.

The range of a deterministic TT M can be restricted to a regular tree language L by restricting its domain to $\tau_M^{-1}(L)$, using Lemmas 9 and 13. For a nondeterministic TT we can use the next corollary.

Corollary 21 *The translation* $\tau' = \{(t,s) \in \tau \mid s \in L\}$ *is in* TT *for every* $\tau \in \mathsf{TT}$ *and* $L \in \mathsf{REGT}$. *If* τ *is in* $\mathsf{TT}_{\downarrow}$ *or* TT_{pru}, *then so is* τ'.

Proof Let Σ be the output alphabet of τ and let $A = (\Sigma, P, F, \delta)$ be a bottom-up finite-state tree automaton such that $L(A) = L$. Obviously $\tau' = \tau \circ \tau_L$ where τ_L is the identity on L, and obviously $\tau_L \in \mathsf{TT}^{\ell}_{\text{rel}}$: it is realized by the local relabeling TT $(\Sigma, \Sigma, P, F, R)$ where R consists of all rules

$$\langle p, \sigma, j\rangle \to \sigma(\langle p_1, \text{down}_1\rangle, \dots, \langle p_m, \text{down}_m\rangle)$$

such that $\delta(\sigma, p_1, \dots, p_m) = p$. By Theorem 20, τ' satisfies the requirements. □

Our third composition result is that deterministic TT's are closed under left-composition with (deterministic) single-use TT's. This is a variant of one of the main results of [40,41,45] for (a variant of) attribute grammars, cf. the last paragraph of [7]. It is proved for attributed tree transducers in [56, Theorem 3] (see also [55, Satz 6.5]).

Lemma 22 $\mathsf{dTT}_{\text{su}} \circ \mathsf{dTT}^{\ell} \subseteq \mathsf{dTT}$.

Proof Let $M_1 = (\Sigma, \Delta, P, p_0, R_1)$ and $M_2 = (\Delta, \Gamma, Q, q_0, R_2)$ be a single-use dTT and a local dTT, respectively. We extend the proof of Lemma 17 to the case that M_2 is an arbitrary local dTT. Thus, we have to deal with the fact that now M_2 can also move up on the output tree of M_1. Let $(t,s) \in \tau_{M_1}$, and let d be the derivation tree of the computation $\langle p_0, \text{root}_t\rangle \Rightarrow^*_{M_1,t} s$. Since M_1 is single-use, we can identify each node of d that is labeled by a configuration with that configuration, because a configuration $\langle p, u\rangle$ of M_1 occurs at most once in d. Suppose that M_1, in configuration $\langle p, u\rangle$ on t, has generated a node v of s. When M_2 executes an up-instruction at node v, the new transducer M has to backtrack on the computation of M_1, back to the moment that the parent of v in s was generated by M_1. Thus, starting with the configuration $\langle p, u\rangle$ of M_1, M has to determine the ancestors of $\langle p, u\rangle$ in d, and stop at the first ancestor that is a configuration generating an output node. Since M_1 is single-use, each configuration $\langle p, u\rangle$ has a unique parent configuration $\langle p', u'\rangle$ in d. That allows us to find $\langle p', u'\rangle$ by a regular test, as follows.

For every $p, p' \in P$ and every instruction α of M_1, we will define a regular test $T_{p,p',\alpha}$ such that for every $t \in \text{dom}(M_1)$ and $u \in \mathcal{N}(t)$, $(t,u) \in T_{p,p',\alpha}$ if and only if $\langle p', \alpha(u)\rangle$ is the parent of $\langle p, u\rangle$ in the derivation tree of the computation $\langle p_0, \text{root}_t\rangle \Rightarrow^*_{M_1,t} \tau_{M_1}(t)$.[7] We will construct a TT N and define $T_{p,p',\alpha} = \{(t,u) \mid \text{mark}(t,u) \in \text{dom}(N)\}$. Then $T_{p,p',\alpha}$ is regular by Corollary 14. To be able to describe N, we change notation and consider the node test $T_{\bar{p},\bar{p}',\bar{\alpha}}$ for $\bar{p}, \bar{p}' \in P$ and instruction $\bar{\alpha}$.

Let $M_1' = (\Sigma, \varnothing, P, \{p_0\}, R_1')$ be the nondeterministic TT obtained from M_1 by changing every output rule $\langle p, \sigma, j, T\rangle \to \delta(\langle p_1, \alpha_1\rangle, \dots, \langle p_k, \alpha_k\rangle)$ into the move rules $\langle p, \sigma, j, T\rangle \to \langle p_i, \alpha_i\rangle$ for every $i \in [1,k]$. Intuitively, for an input tree $t \in \text{dom}(M_1)$, the tree-walking automaton M_1' follows an arbitrary path in the unique derivation tree $d \in L(G^{\text{der}}_{M_1,t})$, from the root of d down to the leaves. Whenever M_1 branches, M_1' nondeterministically follows one of those branches. The transducer N, which is a variant of M_1', has states (p, p', α) with p, p', α as above. The initial state is $(p_0, -, -)$, with the second and third component fixed, but irrelevant (e.g., (p_0, p_0, stay)). On a tree $\text{mark}(t,u)$, N uses the state (p, p', α) to simulate the computations of M_1' in state p on t, but additionally keeps the previous configuration of M_1' in its finite state, as the pair (p', α). When it arrives at the marked node u in state $(\bar{p}, \bar{p}', \bar{\alpha})$, it outputs a symbol of rank 0. Formally, let

[7] For the definition of $\alpha(u)$ see Sect. 3.

$\langle p, \sigma, j, T\rangle \to \zeta$ be a rule in R'_1, let $p' \in P$, let α be an instruction, and let $b \in \{0, 1\}$. Then N has the rule $\langle(p, p', \alpha), (\sigma, b), j, \mu(T)\rangle \to \zeta'$ where $\langle\tilde{p}, \text{down}_i\rangle' = \langle(\tilde{p}, p, \text{up}), \text{down}_i\rangle$, $\langle\tilde{p}, \text{up}\rangle' = \langle(\tilde{p}, p, \text{down}_j), \text{up}\rangle$, and $\langle\tilde{p}, \text{stay}\rangle' = \langle(\tilde{p}, p, \text{stay}), \text{stay}\rangle$ for every $\tilde{p} \in P$ and $i \in [1, \text{rank}(\sigma)]$. Additionally, N has the rule $\langle(\bar{p}, \bar{p}', \bar{\alpha}), (\sigma, 1), j, \mu(T)\rangle \to \top$, where $\top$ is its unique output symbol, of rank 0. Thus, if the tree-walking automaton N arrives in state $(\bar{p}, \bar{p}', \bar{\alpha})$ at the marked node u, it can accept $\text{mark}(t, u)$. Hence, for every $t \in \text{dom}(M_1)$, N accepts $\text{mark}(t, u)$ if and only if $\langle\bar{p}', \bar{\alpha}(u)\rangle$ is the parent of $\langle\bar{p}, u\rangle$ in the derivation tree of the computation $\langle p_0, \text{root}_t\rangle \Rightarrow^*_{M_1,t} \tau_{M_1}(t)$.

The transducer M is an extension of the one in the proof of Lemma 17. It additionally has states $\text{back}^1_{p,q}$ and $\text{back}^*_{p,q}$ to simulate the first and the following backward steps of the computation of M_1. Its rules are obtained as follows. First, it has the same rules that simulate (the forward computation of) M_1. Second, the rules of M that simulate M_2 are extended in such a way that, to obtain ζ' from ζ, one has to change additionally every $\langle q', \text{up}\rangle$ into $\langle\text{back}^1_{p,q'}, \text{stay}\rangle$. Third, M additionally has rules that simulate the backward computation of M_1. For each state $\text{back}^1_{p,q}$ it has all rules $\langle\text{back}^1_{p,q}, \sigma, j, T_{p,p',\alpha}\rangle \to \langle\text{back}^*_{p',q}, \alpha\rangle$ (where the tests on σ and j are irrelevant, because M arrived in state $\text{back}^1_{p,q}$ by a stay-instruction). For each state $\text{back}^*_{p,q}$ it has the following rules. Let $\rho : \langle p, \sigma, j, T\rangle \to \zeta$ be a rule of M_1. If ρ is a move rule, then M has all rules $\langle\text{back}^*_{p,q}, \sigma, j, T \cap T_{p,p',\alpha}\rangle \to \langle\text{back}^*_{p',q}, \alpha\rangle$. If ρ is an output rule, then M has the rule $\langle\text{back}^*_{p,q}, \sigma, j, T\rangle \to \langle(\rho, q), \text{stay}\rangle$. □

Theorem 23 $\mathsf{dTT}_{\text{su}} \circ \mathsf{dTT} \subseteq \mathsf{dTT}$.

Proof It follows from Lemmas 10, 12 and 16 that

$$\mathsf{dTT}_{\text{su}} \circ \mathsf{dTT} \subseteq \mathsf{dTT}_{\text{su}} \circ \mathsf{dTT}^{\ell}_{\text{rel}} \circ \mathsf{dTT}^{\ell}.$$

Thus, by Lemma 22, it suffices to show that $\mathsf{dTT}_{\text{su}} \circ \mathsf{dTT}^{\ell}_{\text{rel}} \subseteq \mathsf{dTT}_{\text{su}}$. For a single-use dTT M_1 and a local relabeling dTT M_2, consider the construction of the dTT M in the last paragraph of the proof of Lemma 17. It should be clear that M is single-use: if M_1 visits an input node in state p, then M visits that node in state (p, q) for some q. □

It can be proved that dTT_{su} is closed under composition, which also follows from Proposition 29 in the next section. The inclusion $\mathsf{dTT}_{\text{su}} \circ \mathsf{dTT}^{\ell}_{\text{rel}} \subseteq \mathsf{dTT}_{\text{su}}$ in the previous proof is a special case of that.

6 Macro and MSO

In this section we collect some results on the connection between TT's, macro tree transducers (in short MT's) and MSO tree transducers. They are taken from the literature or can easily be proved using results from the literature. This section can be skipped on first reading, except that the reader interested in linear size increase should glance at Corollaries 32 and 33.

6.1 Macro tree transducers

Let MT denote the class of translations realized by MT's, with unrestricted or outside-in (OI) derivation mode, let dMT denote the subclass realized by deterministic MT's, and let $\mathsf{d_tMT}$ denote the class of *total* translations in dMT (see [34] where they are denoted by MT_{OI}, DMT_{OI}, and $\mathsf{D_tMT}$, respectively). We first consider the relationship between deterministic TT's and MT's.

It is proved in [28, Lemma 49 and Corollary 51] that $\mathsf{dTT} \subseteq \mathsf{dMT}$, and in [14, Theorem 8.22] (see also [28, Corollary 51]) that $\mathsf{dMT} = \mathsf{dTT}_{\downarrow}^{\ell} \circ \mathsf{dTT}^{\ell}$. Here we prove the following variant.

Lemma 24 $\mathsf{dTT} \subseteq \mathsf{dMT} = \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}$.

Proof We first show that $\mathsf{dTT}_{\downarrow} \circ \mathsf{dMT} \subseteq \mathsf{dMT}$. By Lemma 12 it suffices to show that $\mathsf{dTT}_{\downarrow}^{\mathrm{s}} \circ \mathsf{dMT} \subseteq \mathsf{dMT}$. The inclusion $\mathsf{dTT}_{\downarrow}^{\ell} \circ \mathsf{dMT} \subseteq \mathsf{dMT}$ is proved in [34, Theorem 7.6(3)]. As also argued before [32, Theorem 7.5], this implies the inclusion $\mathsf{dTT}_{\downarrow}^{\mathrm{s}} \circ \mathsf{dMT} \subseteq \mathsf{dMT}$ as follows. By [20, Theorem 2.6] $\mathsf{dTT}_{\downarrow}^{\mathrm{s}} \subseteq \mathsf{DBQREL} \circ \mathsf{dTT}_{\downarrow}^{\ell}$, where DBQREL is the class of deterministic bottom-up finite-state relabelings. Hence $\mathsf{dTT}_{\downarrow}^{\mathrm{s}} \circ \mathsf{dMT} \subseteq \mathsf{DBQREL} \circ \mathsf{dMT}$. Since dMT is closed under regular look-ahead by [34, Theorem 6.15], it is straightforward to prove that $\mathsf{DBQREL} \circ \mathsf{dMT} \subseteq \mathsf{dMT}$, similar to the proof of [34, Lemma 6.17].

By Lemma 10, $\mathsf{dTT} \subseteq \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}^{\ell}$. It is proved in [31, Theorem 35 for $n = 0$] that $\mathsf{dTT}^{\ell} \subseteq \mathsf{dMT}$.[8] Hence $\mathsf{dTT} \subseteq \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}^{\ell} \subseteq \mathsf{dTT}_{\downarrow} \circ \mathsf{dMT} \subseteq \mathsf{dMT}$, which implies that $\mathsf{dTT}_{\downarrow} \circ \mathsf{dTT} \subseteq \mathsf{dTT}_{\downarrow} \circ \mathsf{dMT} \subseteq \mathsf{dMT}$. It now remains to show that $\mathsf{dMT} \subseteq \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}$. It is proved in [31, Section 5.5] that $\mathsf{d_tMT} \subseteq \mathsf{dTT}_{\downarrow}^{\ell} \circ \mathsf{dTT}^{\ell}$. As shown in [34, Theorem 6.18], every translation $\tau \in \mathsf{dMT}$ is the restriction to a regular tree language L of a translation $\tau' \in \mathsf{d_tMT}$. Hence $\tau' \in \mathsf{dTT}_{\downarrow}^{\ell} \circ \mathsf{dTT}^{\ell}$ and so $\tau \in \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}^{\ell}$, because the first TT can start by verifying that the input tree is in L with a regular test at the root of t, by Lemma 9. □

From Lemma 24, together with Theorem 18, we obtain the following corollary on compositions.

Corollary 25 *For every* $k \geq 1$, $\mathsf{dTT}^{k} \subseteq \mathsf{dMT}^{k} = \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}^{k} \subseteq \mathsf{dTT}^{k+1}$.

The above two inclusions are proper, cf. [39, Lemma 6.54] and [34, Theorem 4.16]. In fact, the macro tree transducer is, and can be, of exponential height increase [34, Theorem 3.24]. Hence $\tau_{\mathrm{exp}}^{k+1}$ is not in dMT^{k}, cf. the proof of Proposition 7. Also, τ_{M}^{k} is not in dTT^{k} where M is an MT that translates $\tau^{n}a$ into $\tau^{2^n}a$ (with τ of rank 1 and a of rank 0).

The relationship between nondeterministic TT's and MT's is less straightforward. On the one hand, even $\mathsf{TT}_{\downarrow}$ is not included in MT because all macro tree translations are finitary. But we can express every TT as a composition of two top-down TT's and an MT.

Lemma 26 $\mathsf{TT} \subseteq \mathsf{TT}_{\downarrow} \circ \mathsf{TT}_{\downarrow} \circ \mathsf{MT}$.

Proof By Lemma 10, $\mathsf{TT} \subseteq \mathsf{TT}_{\downarrow} \circ \mathsf{TT}^{\ell}$. It follows from [31, Lemmas 34 and 27] that $\mathsf{TT}^{\ell} \subseteq \mathsf{MON} \circ \mathsf{MT}$, where MON is a specific simple subclass of $\mathsf{TT}_{\downarrow}^{\ell}$ defined before [31, Lemma 27].

We note that by Lemma 10, $\mathsf{TT} \subseteq \mathsf{dTT}_{\mathrm{rel}} \circ \mathsf{TT}^{\ell}$ and that it is easy to prove that $\mathsf{dTT}_{\mathrm{rel}} \circ \mathsf{TT}_{\downarrow}^{\ell} \subseteq \mathsf{TT}_{\downarrow}$. Hence we even obtain that $\mathsf{TT} \subseteq \mathsf{TT}_{\downarrow} \circ \mathsf{MT}$. □

On the other hand, every MT can still be realized by a composition of two (finitary) TT's.

Lemma 27 $\mathsf{MT} \subseteq \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT} \circ \mathsf{TT}_{\mathrm{pru}} \subseteq \mathsf{dTT}_{\downarrow} \circ \mathsf{fTT}$.

Proof By [34, Theorem 6.10], $\mathsf{MT} = \mathsf{d_tMT} \circ \mathsf{SET}$, and by the proof of [34, Theorem 6.10], $\mathsf{SET} \subseteq \mathsf{TT}_{\mathrm{pru}}^{\ell}$. Hence $\mathsf{MT} \subseteq \mathsf{d_tMT} \circ \mathsf{TT}_{\mathrm{pru}}^{\ell} \subseteq \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT} \circ \mathsf{TT}_{\mathrm{pru}}$ by Lemma 24. That is included in $\mathsf{dTT}_{\downarrow} \circ \mathsf{fTT}$ by Theorem 20. □

[8] By mistake, [31, Theorem 35] is stated for $n \geq 1$ only. It also holds for $n = 0$ by [31, Lemma 34 and Theorem 31].

It can be shown that $\mathsf{fTT} \subseteq \mathsf{MT} = \mathsf{dTT}_{\downarrow} \circ \mathsf{fTT}$, thus generalizing Lemma 24 to the finitary case, but that will not be needed in what follows.

Finally, let $\mathsf{MT}_{\mathrm{IO}}$ denote the class of translations realized by MT's with inside-out (IO) derivation mode (see [34]), and let $\mathsf{mrMT}_{\mathrm{IO}}$ denote the class of translations realized by the multi-return macro tree transducers of [49,50], which generalize IO macro tree transducers.

Lemma 28 $\mathsf{MT}_{\mathrm{IO}} \subseteq \mathsf{mrMT}_{\mathrm{IO}} \subseteq \mathsf{fTT}_{\downarrow} \circ \mathsf{dTT}$.

Proof It is shown in [34, Lemma 5.5] that $\mathsf{MT}_{\mathrm{IO}} \subseteq \mathsf{fTT}^{\ell}_{\downarrow} \circ \mathsf{YIELD}$, and in [31, Lemma 36] that $\mathsf{YIELD} \subseteq \mathsf{dTT}^{\ell}$, and so $\mathsf{MT}_{\mathrm{IO}} \subseteq \mathsf{fTT}^{\ell}_{\downarrow} \circ \mathsf{dTT}$. It follows from [50, Lemma 4] that $\mathsf{mrMT}_{\mathrm{IO}} \subseteq \mathsf{dTT}^{\ell}_{\downarrow} \circ \mathsf{MT}_{\mathrm{IO}} \circ \mathsf{dTT}^{\ell}_{\downarrow}$. Hence

$$\mathsf{mrMT}_{\mathrm{IO}} \subseteq \mathsf{dTT}^{\ell}_{\downarrow} \circ \mathsf{fTT}^{\ell}_{\downarrow} \circ \mathsf{dTT} \circ \mathsf{dTT}_{\downarrow}$$

which is included in $\mathsf{fTT}^{\mathrm{s}}_{\downarrow} \circ \mathsf{dTT}$ by [20, Theorem 2.11(2)] and Theorem 18. □

6.2 MSO tree transducers

Let dMSOT denote the class of deterministic MSO tree translations (see [14, Chapter 8], where it is denoted DMSOT, and where MSO tree translations are called MS-transductions of terms). The next result is a variant of the main result of [7], which concerns attributed tree transducers with look-ahead instead of TT's. In its present form it is proved in [14, Theorems 8.6 and 8.7].

Proposition 29 $\mathsf{dMSOT} = \mathsf{dTT}_{\mathrm{su}}$.

The next proposition is the main result of [32].

Proposition 30 $\mathsf{d_tMT} \cap \mathsf{LSIF} \subseteq \mathsf{dMSOT}$.

This can be extended to arbitrary deterministic OI macro tree translations as follows.

Lemma 31 $\mathsf{dMT} \cap \mathsf{LSIF} \subseteq \mathsf{dMSOT}$.

Proof Since the domain L of any MT M is regular ([34, Theorem 7.4]), and dMT is closed under regular look-ahead ([34, Theorem 6.15]), there is a total MT M' that extends M by the identity on the complement of L. Clearly, $\tau_{M'}$ is of linear size increase if and only if τ_M is. Hence, by Propositions 29 and 30, if τ_M is of linear size increase, then $\tau_{M'}$ is in $\mathsf{dTT}_{\mathrm{su}}$. And so τ_M, which is the restriction of $\tau_{M'}$ to the regular tree language L, is also in $\mathsf{dTT}_{\mathrm{su}}$ by Lemma 9. □

From Lemma 24, Lemma 31, Proposition 29, and Lemma 6 we obtain the following corollary.

Corollary 32 $\mathsf{dTT} \cap \mathsf{LSIF} = \mathsf{dTT}_{\mathrm{su}}$.

It is also shown in [32] that it is decidable for a total deterministic MT whether or not it is of linear size increase. That also holds for arbitrary deterministic MT's by the proof of Lemma 31, and hence also for dTT's by Lemma 24.

Corollary 33 *It is decidable for a deterministic* TT *whether or not it is of linear size increase.*

Note that since Corollary 32 is effective, if the dTT is indeed of linear size increase, then an equivalent $\mathrm{TT}_{\mathrm{su}}$ can be constructed. One of our aims is to extend Corollaries 32 and 33 to arbitrary compositions of dTT's.

7 Functional nondeterminism

In this section we prove that for every nondeterministic top-down TT M a deterministic top-down TT M' can be constructed that realizes a "uniformizer" of τ_M, i.e., a subset of τ_M with the same domain. This is a generalization of [21, Lemma], where it is proved for classical nondeterministic top-down tree transducers. Note that, as opposed to the deterministic case, the nondeterministic top-down TT is more powerful than the classical nondeterministic top-down tree transducer with regular look-ahead, because, due to the stay-instructions, it may not be finitary, i.e., it possibly translates one input tree into infinitely many output trees.

A *uniformizer* of a tree translation τ is a function f such that $f \subseteq \tau$ and $\mathrm{dom}(f) = \mathrm{dom}(\tau)$. Intuitively, f selects for every input tree $t \in \mathrm{dom}(\tau)$ one of the elements of $\tau(t)$.

Lemma 34 *Every* $\tau \in \mathsf{TT}_{\downarrow}$ *has a uniformizer* $\tau' \in \mathsf{dTT}_{\downarrow}$. *If* $\tau \in \mathsf{TT}_{\mathrm{pru}}$, *then* $\tau' \in \mathsf{dTT}_{\mathrm{pru}}$.

Proof Let $M = (\Sigma, \Delta, Q, Q_0, R)$ be a nondeterministic TT$_{\downarrow}$. Without loss of generality we assume that M has exactly one initial state q_0, i.e., $Q_0 = \{q_0\}$. We have to construct a deterministic TT$_{\downarrow}$ M' that computes one possible output tree in $\tau_M(t)$ for every $t \in \mathrm{dom}(M)$. The idea of the proof of [21, Lemma] is to pick, at the current node of t, one of the rules that lead to the generation of an output tree (which can be checked by a regular test). However, that idea does not work here, because M may have an infinite computation on t (see [24, New Observation 5.10]). Thus, we have to be more careful. Note that an infinite computation is entirely due to the stay-instructions in the rules of M.

The stay-instructions can be removed from M by constructing the equivalent stay-free TT $M_{\mathrm{sf}} = (\Sigma, \Delta, Q, \{q_0\}, R_{\mathrm{sf}})$, with general rules, as we did at the end of Sect. 3. Recall that we assume that the regular tests in $\mathcal{T}_M$ are mutually disjoint, and that the set R_{sf} consists of all general rules $\langle q, \sigma, j, T\rangle \to \zeta$ such that $\zeta \in L(G_{q,\sigma,j,T})$, for every left-hand side $\langle q, \sigma, j, T\rangle$ of a rule of M. In this case M_{sf} is a top-down TT, with possibly infinitely many rules. Since its rules do not contain stay-instructions any more, it does not have infinite computations on the trees in its domain. Thus, the idea above can be applied to M_{sf}, which means that for every q, σ, j, and T we have to pick one general rule $\langle q, \sigma, j, T\rangle \to \zeta$ from R_{sf}, under the condition that its application leads to the generation of an output tree. This condition can be checked by a regular sub-test, as follows. Note that $\zeta \in T_\Delta(D_\sigma)$ where $D_\sigma = \{\langle q', \mathrm{down}_i\rangle \mid q' \in Q,\ i \in [1, \mathrm{rank}_\Sigma(\sigma)]\}$.

For every $\sigma \in \Sigma$, $q' \in Q$, and $i \in [1, \mathrm{rank}(\sigma)]$, let $T_{\sigma,q',i}$ be the node test over Σ consisting of all (t,u) such that u has label σ in t and there is a computation $\langle q', ui\rangle \Rightarrow^*_{M,t} s$ for some $s \in T_\Delta$. This node test is regular by Corollary 14 because $\mathrm{mark}(T_{\sigma,q',i})$ is the domain of a TT $M_{q',i}$ that on input $\mathrm{mark}(t,u)$ walks to the marked node u, checks that its label is σ, moves to the i-th child of u, and then simulates M on t, starting in state q. For every $\sigma \in \Sigma$ and $D \subseteq D_\sigma$, let $T_{\sigma,D}$ be the regular node test that is the intersection of all $T_{\sigma,q',i}$ such that $\langle q', \mathrm{down}_i\rangle \in D$ and all $T^\bullet_\Sigma \setminus T_{\sigma,q',i}$ such that $\langle q', \mathrm{down}_i\rangle \notin D$. Obviously the node tests $T_{\sigma,D}$ are mutually disjoint.

We now define the deterministic TT$_{\downarrow}$ $M' = (\Sigma, \Delta, Q, q_0, R')$, where R' consists of the following general rules. For every left-hand side $\langle q, \sigma, j, T\rangle$ of a rule of M and every $D \subseteq D_\sigma$, if $L(G_{q,\sigma,j,T}) \cap T_\Delta(D) \neq \varnothing$, then R' contains the general rule $\langle q, \sigma, j, T \cap T_{\sigma,D}\rangle \to \zeta$ where ζ is a fixed element of $L(G_{q,\sigma,j,T}) \cap T_\Delta(D)$.

It should be clear that M' satisfies the requirements, i.e., it has the same domain as M_{sf} and it realizes a subset of $\tau_{M_{\mathrm{sf}}}$. Note that M' can be constructed effectively, because $L(G_{q,\sigma,j,T}) \cap T_\Delta(D)$ is a regular tree language, and hence its nonemptiness can be decided and, if so, an element can be computed. Finally, the general rules of M' can be replaced by ordinary rules, as discussed after Lemma 8. □

At the end of this section we prove that any function that is realized by a composition of nondeterministic TT's can also be realized by a composition of deterministic TT's. That will (only) be used to show that the results of Sect. 9 also hold for nondeterministic TT's and MT's. Let $\mathcal{F}$ be the class of all partial functions from trees to trees.

Theorem 35 *For every $k \geq 1$, $(\mathsf{TT}_{\downarrow} \circ \mathsf{TT}^k) \cap \mathcal{F} \subseteq \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}^k$.*

Proof By Lemmas 26 and 27, $\mathsf{TT} \subseteq \mathsf{TT}_{\downarrow} \circ \mathsf{TT}_{\downarrow} \circ \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT} \circ \mathsf{TT}_{\downarrow}$. Now let $\tau \in (\mathsf{TT}_{\downarrow} \circ \mathsf{TT}^k) \cap \mathcal{F}$. Then $\tau = \tau_1 \circ \cdots \circ \tau_m$ where $m = 5k+1$, $\tau_{5j} \in \mathsf{dTT}$ for every $j \in [1,k]$, and $\tau_i \in \mathsf{TT}_{\downarrow}$ for every $i \in [1,m] \setminus \{5j \mid j \in [1,k]\}$. By Corollary 14, the domain of a translation in TT is regular. Hence, we may assume that $\mathrm{ran}(\tau_i) \subseteq \mathrm{dom}(\tau_{i+1})$ for every $i \in [1, m-1]$. If not, then we change τ_i into $\bar{\tau}_i$ for $i = m, \ldots, 1$ inductively as follows. First, $\bar{\tau}_m = \tau_m$. Second, for $i < m$ we obtain $\bar{\tau}_i$ from τ_i by restricting its range to $\mathrm{dom}(\bar{\tau}_{i+1})$, see Corollary 21 and the paragraph preceding it.

Since τ is a function, it should be clear that $\tau = \tau'_1 \circ \cdots \circ \tau'_m$ where $\tau'_i \in \mathsf{dTT}_{\downarrow}$ is the uniformizer of τ_i that exists by Lemma 34 if $\tau_i \in \mathsf{TT}_{\downarrow}$, and $\tau'_i = \tau_i$ if $\tau_i \in \mathsf{dTT}$. Thus, $\tau \in \mathsf{dTT}_{\downarrow} \circ (\mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT} \circ \mathsf{dTT}_{\downarrow})^k$ and so, by Theorem 18, $\tau \in \mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}^k$. □

Corollary 36 *For every $k \geq 1$, $\mathsf{MT}^k \cap \mathcal{F} \subseteq \mathsf{dMT}^k$.*

Proof By the same argument as in the proof of Theorem 35, using Lemma 27 only, we obtain that $\mathsf{MT}^k \cap \mathcal{F} \subseteq (\mathsf{dTT}_{\downarrow} \circ \mathsf{dTT} \circ \mathsf{dTT}_{\downarrow})^k$. By Theorem 18 that is included in $\mathsf{dTT}_{\downarrow} \circ \mathsf{dTT}^k$, which equals dMT^k by Corollary 25. □

Since the inclusions in Corollary 25 are proper, as discussed after that corollary, Theorem 35 and Corollary 36 imply that TT^k and MT^k are also proper hierarchies, i.e., $\mathsf{TT}^k \subsetneq \mathsf{TT}^{k+1}$ and $\mathsf{MT}^k \subsetneq \mathsf{MT}^{k+1}$ for every $k \geq 1$.

8 Productivity

In this section we prove that every TT can be decomposed into a pruning TT and another TT such that the composition is linear-bounded. It implies that we may always assume that a composition of two TT's is linear-bounded. Recall from Sect. 2 that the composition of tree translations $\tau_1 \subseteq T_\Sigma \times T_\Delta$ and $\tau_2 \subseteq T_\Delta \times T_\Gamma$ is linear-bounded if there is a constant $c \in \mathbb{N}$ such that for every $(t,s) \in \tau_1 \circ \tau_2$ there exists $r \in T_\Delta$ such that $(t,r) \in \tau_1$, $(r,s) \in \tau_2$, and $|r| \leq c \cdot |s|$. Formally we say that the pair (τ_1, τ_2) is linear-bounded. Recall also that for classes $\mathcal{T}_1$ and $\mathcal{T}_2$ of tree translations, the class $\mathcal{T}_1 * \mathcal{T}_2$ consists of all translations $\tau_1 \circ \tau_2$ such that $\tau_1 \in \mathcal{T}_1$, $\tau_2 \in \mathcal{T}_2$, and (τ_1, τ_2) is linear-bounded. Two elementary properties of this class operation were stated in Lemma 1. We will prove the following theorem.

Theorem 37 $\mathsf{TT} \subseteq \mathsf{TT}_{\mathrm{pru}} * \mathsf{TT}$ *and* $\mathsf{dTT} \subseteq \mathsf{dTT}_{\mathrm{pru}} * \mathsf{dTT}$.

Since pruning TT's can be absorbed to the right by arbitrary TT's (by Theorems 20 and 18), Theorem 37 can be generalized to compositions of TT's. It implies that we may always assume that a composition of a TT with any number of TT's is linear-bounded.

Corollary 38 *Let $k \geq 1$.*

(1) $\mathsf{TT}^k \subseteq \mathsf{TT}_{\mathrm{pru}} * \mathsf{TT}^k$ *and* $\mathsf{TT} \circ \mathsf{TT}^k = \mathsf{TT} * \mathsf{TT}^k$, *and*
(2) $\mathsf{dTT}^k \subseteq \mathsf{dTT}_{\mathrm{pru}} * \mathsf{dTT}^k$ *and* $\mathsf{dTT} \circ \mathsf{dTT}^k = \mathsf{dTT} * \mathsf{dTT}^k$.

Proof (1) The proof of the inclusion is by induction on k. For $k = 1$ it is Theorem 37. The induction step is proved as follows:

$$\begin{aligned} \mathsf{TT} \circ \mathsf{TT}^k &\subseteq \mathsf{TT} \circ (\mathsf{TT}_{\text{pru}} * \mathsf{TT}^k) \\ &\subseteq (\mathsf{TT} \circ \mathsf{TT}_{\text{pru}}) * \mathsf{TT}^k \\ &\subseteq \mathsf{TT} * \mathsf{TT}^k \\ &\subseteq (\mathsf{TT}_{\text{pru}} * \mathsf{TT}) * \mathsf{TT}^k \\ &\subseteq \mathsf{TT}_{\text{pru}} * (\mathsf{TT} \circ \mathsf{TT}^k) \end{aligned}$$

where the first inclusion is by the induction hypothesis and the remaining inclusions are by Lemma 1, Theorem 20 (which says that $\mathsf{TT} \circ \mathsf{TT}_{\text{pru}} \subseteq \mathsf{TT}$), Theorem 37, and Lemma 1 again. The equation now follows from the inclusions above.

(2) The proof is exactly the same as in (1), using Theorem 18 instead of Theorem 20. □

The remainder of this section is devoted to the proof of Theorem 37. It is essentially a variant of the proof of [25, Lemma 4.1], which is the key lemma of [25] and concerns the removal of “superfluous computations” in attribute grammars. In its turn, that proof generalized the proof of [4, Lemma 1] where this was done for top-down tree transducers (and strangely enough, the author of [25] did not mention that).

To prove Theorem 37 it suffices, by Lemma 10, Lemma 1, and Theorems 20 and 18, to consider local TT’s, i.e., to prove that $\mathsf{TT}^\ell \subseteq \mathsf{TT}_{\text{pru}} * \mathsf{TT}$ and that $\mathsf{dTT}^\ell \subseteq \mathsf{dTT}_{\text{pru}} * \mathsf{dTT}$. We prove the first and second inclusion in a first and second subsection, respectively. In the first subsection we additionally take care that the construction preserves the determinism of the given TT.

8.1 Nondeterministic productivity

Let $M = (\Sigma, \Delta, Q, Q_0, R)$ be a TT. For a pair $(t, s) \in \tau_M$ and a computation $\langle q_0, \text{root}_t\rangle \Rightarrow^*_{M,t} s$ with $q_0 \in Q_0$, we say that a node u of t is *productive* (in that computation) if there is a $q \in Q$ such that an output rule is applied to the configuration $\langle q, u\rangle$ in the computation. Obviously, the size of s is at least the number of productive nodes of t. For $i \in \{0, 1\}$ we define the computation to be *i-productive* if all nodes of t of rank i are productive.[9] Moreover, the computation is *productive* if it is both 0-productive and 1-productive, i.e., all leaves and monadic nodes of t are productive. Finally, we define τ^0_M to consist of all $(t, s) \in \tau_M$ for which there is a 0-productive computation $\langle q_0, \text{root}_t\rangle \Rightarrow^*_{M,t} s$ for some $q_0 \in Q_0$, and we define τ^{01}_M to consist of all $(t, s) \in \tau_M$ for which there is a productive computation of that form. Since the size of t is at most twice the number of leaves plus the number of monadic nodes of t,[10] it follows that $|t| \leq 2 \cdot |s|$ for every $(t, s) \in \tau^{01}_M$.

To prove that $\mathsf{TT}^\ell \subseteq \mathsf{TT}_{\text{pru}} * \mathsf{TT}$, our goal is to construct, for a given TT^ℓ M, a pruning TT N and a TT^ℓ M' in such a way that $\tau_N \circ \tau_{M'} \subseteq \tau_M$ and $\tau_M \subseteq \tau_N \circ \tau^{01}_{M'}$. This obviously implies that $\tau_N \circ \tau_{M'} = \tau_M$. The second inclusion says that for every $(t, s) \in \tau_M$ there exists a tree t' such that $(t, t') \in \tau_N$ and $(t', s) \in \tau^{01}_{M'}$. Thus, as observed above, $|t'| \leq 2 \cdot |s|$, and hence $(\tau_N, \tau_{M'})$ is linear-bounded (for the constant $c = 2$).

[9] Recall from Sect. 2 that the rank of a node is the rank of its label, i.e., the number of its children.

[10] To be precise, $|t| \leq (2 \cdot |t|_0 - 1) + |t|_1$ where $|t|_0$ and $|t|_1$ are the number of leaves and monadic nodes of t, respectively.

To this aim, N will remove sufficiently many unproductive nodes from the input tree, and add state transition information of M to the labels of the remaining nodes, thus allowing M' to simulate M without having to visit those unproductive nodes. Since productivity of a node of the input tree t depends on the computation of M on t, N nondeterministically guesses which nodes to remove, and uses its regular tests to determine the possible behaviour of M on the remaining nodes. To reduce the technical complexity of the proof, the construction of N and M' will be done in two steps, removing unproductive leaves and monadic nodes in the first and second step, respectively.

Lemma 39 *For every* TT^{ℓ} M *there are a* $\mathrm{TT}_{\mathrm{pru}}$ N *and a* TT^{ℓ} M' *such that*

$$\tau_N \circ \tau_{M'} \subseteq \tau_M \subseteq \tau_N \circ \tau^0_{M'}.$$

If M is deterministic, then so is M'.

Lemma 40 *For every* TT^{ℓ} M *there are a* $\mathrm{TT}_{\mathrm{pru}}$ N *and a* TT^{ℓ} M' *such that*

$$\tau_N \circ \tau_{M'} \subseteq \tau_M \quad \textit{and} \quad \tau^0_M \subseteq \tau_N \circ \tau^{01}_{M'}.$$

If M is deterministic, then so is M'.

It is easy to see that applying these lemmas one after the other, we have obtained the goal above; note that pruning TT's are closed under composition by Theorem 20. It remains to prove the two lemmas. The constructions in their proofs are similar to the removal of ε-rules and chain rules from a context-free grammar, respectively. As is well known, one should not remove these rules in the reverse order, because the removal of ε-rules can create new chain rules. Similarly in our case, we should remove unproductive leaves and monadic nodes in that order, because the removal of unproductive leaves can create new unproductive monadic nodes. Note also that removing ε-rules and chain rules in one construction is technically more complex.

Proof of Lemma 39 Let $M = (\Sigma, \Delta, Q, Q_0, R)$ be a TT^{ℓ}. As discussed in the second paragraph after Proposition 7 (in Sect. 3), we may assume that the output rules of M only use the stay-instruction. Let us consider $(t, s) \in \tau_M$ and a computation $\langle q_0, \mathrm{root}_t\rangle \Rightarrow^*_{M,t} s$ with $q_0 \in Q_0$. The idea of the construction of the $\mathrm{TT}_{\mathrm{pru}}$ N and TT^{ℓ} M' is that N (nondeterministically) preprocesses t by removing the maximal subtrees of t that consist of unproductive nodes only, and that M' simulates M on the rest of t. Let us say that a node u of t is *superfluous* (in this computation) if it is unproductive and all its descendants are unproductive. Note that the root of t is *not* superfluous. Thus, N changes t into t' by pruning all superfluous nodes of t. Moreover, it adds state transition information of M to the labels of the remaining nodes to allow M' on t' to simulate the above computation of M on t. In the resulting computation of M' on t', the input tree t' of M' has no superfluous nodes, which means in particular that all its leaves are productive. Note that, due to the removal of the superfluous nodes, each remaining node loses its superfluous children. Since the pruning TT N does not know which nodes are going to be superfluous in M's computation, it just nondeterministically removes subtrees of the input tree t and adds to the label of each remaining node all possible state transitions of M in computations on the removed subtrees that use move rules only. Whereas N just guesses the superfluous nodes, it uses its regular tests to determine the state transitions of M on those nodes.

As intermediate alphabet we use the ranked alphabet Γ consisting of all symbols $\langle \sigma, (i_1, \ldots, i_n), \gamma\rangle$ such that $\sigma \in \Sigma$, $n \in [0, \mathrm{rank}(\sigma)]$, $1 \leq i_1 < i_2 < \cdots < i_n \leq \mathrm{rank}(\sigma)$, and $\gamma \subseteq Q \times Q$. The rank of $\langle \sigma, (i_1, \dots, i_n), \gamma \rangle$ is n. In the case where M is deterministic we require γ to be a partial function from Q to Q. Intuitively, a node u of t with label σ that is not removed by N, will be relabeled by $\langle \sigma, (i_1, \dots, i_n), \gamma \rangle$ such that the subtrees at its children ui with $i \notin \{i_1, \dots, i_n\}$ are removed by N and γ is the set of all $(q, \bar{q})$ such that M has a computation from $\langle q, u \rangle$ to $\langle \bar{q}, u \rangle$ (using move rules only) that visits one of the removed subtrees.

Formally, we define $N = (\Sigma, \Gamma, \{p\}, \{p\}, R_N)$ with one state p. For every symbol $\langle \sigma, (i_1, \dots, i_n), \gamma \rangle$ in Γ and every $j \in [0, mx_\Sigma]$, it has the rule

$$\langle p, \sigma, j, T \rangle \to \langle \sigma, (i_1, \dots, i_n), \gamma \rangle (\langle p, \text{down}_{i_1} \rangle, \dots, \langle p, \text{down}_{i_n} \rangle)$$

where T is defined as follows. Let $t \in T_\Sigma$ and let $u \in \mathcal{N}(t)$. The state transition relation γ is uniquely determined by $(i_1, \dots, i_n)$, and is expressed by T. Let us say that a node $v \in \mathcal{N}(t)$ is a *ghost* if $v = uiw$ for some $i \notin \{i_1, \dots, i_n\}$ and $w \in \mathbb{N}^*$. Moreover, let us say that a computation

$$\langle q_1, u_1 \rangle \Rightarrow_{M,t} \langle q_2, u_2 \rangle \Rightarrow_{M,t} \cdots \Rightarrow_{M,t} \langle q_m, u_m \rangle,$$

$m \geq 3$, is a *ghost computation* from $\langle q_1, u_1 \rangle$ to $\langle q_m, u_m \rangle$ if u_j is a ghost for every $j \in [2, m-1]$. Note that such a computation is due to move rules only, that it visits at least one ghost, and that the ghosts $u_2, \dots, u_{m-1}$ all belong to a subtree at the same child ui. Finally, for states $q, \bar{q} \in Q$ we will write $q \hookrightarrow \bar{q}$ if there is a ghost computation from $\langle q, u \rangle$ to $\langle \bar{q}, u \rangle$. We now define T to consist of all (t, u) such that $\gamma = \{(q, \bar{q}) \in Q \times Q \mid q \hookrightarrow \bar{q}\}$. Note that γ is indeed a partial function if M is deterministic. The test T is regular because it is a boolean combination of tests $T_{q,\bar{q}} = \{(t, u) \mid q \hookrightarrow \bar{q}\}$, which are regular because the tree language $\{\text{mark}(t, u) \mid q \hookrightarrow \bar{q}\}$ is regular for every $(q, \bar{q}) \in Q \times Q$ by Corollary 14: it is the domain of a TT that first walks to u, then simulates a ghost computation of M on t from $\langle q, u \rangle$ to $\langle \bar{q}, u \rangle$, and finally outputs a symbol of rank 0.

We define $M' = (\Gamma, \Delta, Q, Q_0, R')$ with the following rules. Let $\rho : \langle q, \sigma, j \rangle \to \zeta$ be a rule in R, and let $\langle \sigma, (i_1, \dots, i_n), \gamma \rangle$ be an element of Γ (with the same σ). If ρ is an output rule or $\zeta = \langle q', \alpha \rangle$ with $\alpha \in \{\text{up}, \text{stay}\}$, then R' contains the rule $\langle q, \langle \sigma, (i_1, \dots, i_n), \gamma \rangle, j \rangle \to \zeta$. If $\zeta = \langle q', \text{down}_{i_k} \rangle$ with $k \in [1, n]$, then R' contains the rule $\langle q, \langle \sigma, (i_1, \dots, i_n), \gamma \rangle, j \rangle \to \langle q', \text{down}_k \rangle$. Otherwise (i.e., $\zeta = \langle q', \text{down}_i \rangle$ with $i \notin \{i_1, \dots, i_n\}$), R' contains the rule $\langle q, \langle \sigma, (i_1, \dots, i_n), \gamma \rangle, j \rangle \to \langle \bar{q}, \text{stay} \rangle$ for every $(q, \bar{q}) \in \gamma$. Note that if M is deterministic, then so is M'.

It should be clear that $\tau_N \circ \tau_{M'} \subseteq \tau_M$, because for every $t' \in \tau_N(t)$ the computations of M' on t' simulate computations of M on t.

To understand that $\tau_M \subseteq \tau_N \circ \tau^0_{M'}$, consider a computation $\langle q_0, \text{root}_t \rangle \Rightarrow^*_{M,t} s$ with $q_0 \in Q_0$, and let $t' \in \tau_N(t)$ be such that all superfluous nodes of t (in this computation) are removed. Then it should be clear that the computation of M on t can be simulated by a computation $\langle q_0, \text{root}_{t'} \rangle \Rightarrow^*_{M',t'} s$ of M' on t'. In fact, if M visits a superfluous child of the current (non-superfluous) node u of t, then M' just stays in the node v corresponding to u in t' and changes its state to the one in which M returns to u. For a completely formal correctness proof one would have to formalize the obvious bijective correspondence f between the non-superfluous nodes of t and the nodes of t'. In fact, $f(\varepsilon) = \varepsilon$, and if u is non-superfluous and $ui_1, \dots, ui_n$ are all the non-superfluous children of u, then $f(ui_k) = f(u)k$ for every $k \in [1, n]$. Note that u and $f(u)$ have the same child number. However, the correctness of the construction should be clear without such a proof. The configurations $\langle q, u \rangle$ of M on t, for every non-superfluous node u, are simulated by the configurations $\langle q, f(u) \rangle$ of M' on t'. Finally, the above computation of M' on t' is 0-productive, because each leaf $f(u)$ of t'

corresponds to a non-superfluous node u of t of which all descendants are superfluous, i.e., to a productive node. Since M' simulates M, it follows that $f(u)$ is a productive node of t'. This ends the proof of Lemma 39. □

Proof of Lemma 40 This proof is similar to the previous one. Let $M = (\Sigma, \Delta, Q, Q_0, R)$ be a TT^{ℓ}. Again, we assume that the output rules of M only use the stay-instruction. And again, let us consider $(t, s) \in \tau_M$ and a computation $\langle q_0, \text{root}_t\rangle \Rightarrow^*_{M,t} s$ with $q_0 \in Q_0$. This time we define a node of t to be *superfluous* if it is unproductive (in this computation) and has rank 1. As before, N changes t into t' by pruning all superfluous nodes of t, and adds information to the labels of the remaining nodes to allow M' on t' to simulate the above computation of M on t. Whereas in the previous case, M' had to shortcut the subcomputations of M on maximal subtrees of superfluous nodes, in the present case M' has to shortcut the subcomputations of M on maximal sequences $u_1, \ldots, u_n$ of superfluous nodes ($n \geq 1$), where u_{i+1} is the unique child of u_i for every $i \in [1, n-1]$. For such a sequence, the unique child u_{n+1} of u_n is non-superfluous, and either u_1 is the root of t, or the parent u_0 of u_1 is non-superfluous. In the second case, a subcomputation of M on $u_1, \ldots, u_n$ is as follows. When it moves from u_0 down to u_1, it either returns to u_0, or it walks to u_{n+1}. And when it moves from u_{n+1} up to u_n, it either returns to u_{n+1}, or it walks to u_0. In the first case, M can only move from u_{n+1} up to u_n and return to u_{n+1}. Thus, to the label of every non-superfluous node u of t we have to add information both on trips to superfluous nodes above u and trips to superfluous nodes below u. In the first case, u_{n+1} will be the root of t'. In the second case, u_{n+1} will be the i-th child of u_0 in t', where i is the child number of u_1 in t. Thus, the child number of u_{n+1} changes from 1 to 0, or from 1 to i, respectively.

As in the previous proof, the pruning TT N does not know in advance which nodes are going to be superfluous in M's computation. Thus, it just nondeterministically removes monadic nodes of the input tree t and adds to the label of each remaining node all possible state transitions of M in subcomputations on the removed nodes that use move rules only. Rather than constructing N directly, it is more convenient to realize this pruning of t by two consecutive pruning TT's N_1 and N_2, and use Theorem 20. The local relabeling TT N_1 nondeterministically marks monadic nodes of t, by possibly changing the label σ of a monadic node into $\widehat{\sigma}$. The (deterministic) TT N_2 then removes the marked nodes, and relabels the unmarked nodes, adding the appropriate state transitions of M (determined by regular tests). Since it is easy to construct N_1, we only discuss N_2.

The intermediate alphabet Γ now consists of all symbols $\langle \sigma, j, U, \gamma\rangle$ such that $\sigma \in \Sigma$, $j \in [0, mx_\Sigma)]$, $U \subseteq \{\text{up}\} \cup \{\text{down}_i \mid i \in [1, \text{rank}(\sigma)]\}$, and $\gamma \subseteq Q \times (Q \times I)$, where I is the set of all possible instructions. The rank of $\langle \sigma, j, U, \gamma\rangle$ is the rank of σ. As before, in the case where M is deterministic we require γ to be a partial function from Q to $Q \times I$. Intuitively, a node u of t with label σ that is not marked by N_1, will be relabeled by $\langle \sigma, j, U, \gamma\rangle$ such that j is its child number in t, $\alpha \in U$ if and only if $\alpha(u)$ is marked by N_1, and γ is the set of all $(q, \langle \bar{q}, \beta\rangle)$ such that the following holds: M has a computation from $\langle q, u\rangle$ to $\langle \bar{q}, \bar{u}\rangle$ (using move rules only) that visits a maximal sequence of marked nodes, for some unmarked node $\bar{u}$ such that $\beta(v) = \bar{v}$, where v and $\bar{v}$ are the nodes corresponding to u and $\bar{u}$ in the tree t'.

We define $N_2 = (\Sigma \cup \widehat{\Sigma}, \Gamma, P, p_0, R_2)$, where $\widehat{\Sigma} = \{\widehat{\sigma} \mid \sigma \in \Sigma^{(1)}\}$, $P = \{p_j \mid j \in [0, mx_\Sigma]\}$, and R_2 is defined as follows. For every $\sigma \in \Sigma^{(1)}$ and $j, j' \in [0, mx_\Sigma]$ the transducer N_2 has the rule $\langle p_j, \widehat{\sigma}, j'\rangle \to \langle p_j, \text{down}_1\rangle$. Moreover, for every $\langle \sigma, j, U, \gamma\rangle \in \Gamma$ and $j' \in [0, mx_\Sigma]$ it has the rule

$$\langle p_j, \sigma, j', T\rangle \to \langle \sigma, j, U, \gamma\rangle(\langle p_1, \text{down}_1\rangle, \ldots, \langle p_m, \text{down}_m\rangle)$$

where $m = \text{rank}(\sigma)$ and T is defined as follows. Let $\hat{t}$ be a tree over $\Sigma \cup \widehat{\Sigma}$ and let $u \in \mathcal{N}(\hat{t})$. We define $\pi(\hat{t})$ to be the tree over Σ that is obtained from $\hat{t}$ by changing every label $\widehat{\sigma}$ into σ. Both U and γ are uniquely determined, and they are expressed by T. Let us say that a node $v \in \mathcal{N}(\hat{t})$ is a *ghost* if its label is in $\widehat{\Sigma}$. A *ghost computation* is defined as in the previous proof, for $t = \pi(\hat{t})$; note that $\mathcal{N}(t) = \mathcal{N}(\hat{t})$. And let us write $\langle q, u\rangle \hookrightarrow \langle \bar{q}, \bar{u}\rangle$ if there is a ghost computation from $\langle q, u\rangle$ to $\langle \bar{q}, \bar{u}\rangle$. We now define T to consist of all $(\hat{t}, u)$ such that

- up $\in U$ if and only u has a parent and that parent is a ghost,
- $\text{down}_i \in U$ if and only if ui is a ghost,
- $(q, \langle \bar{q}, \text{stay}\rangle) \in \gamma$ if and only if $\langle q, u\rangle \hookrightarrow \langle \bar{q}, u\rangle$,
- $(q, \langle \bar{q}, \text{up}\rangle) \in \gamma$ if and only if $\langle q, u\rangle \hookrightarrow \langle \bar{q}, \bar{u}\rangle$ for some ancestor $\bar{u}$ of u,
- $(q, \langle \bar{q}, \text{down}_i\rangle) \in \gamma$ if and only if $\langle q, u\rangle \hookrightarrow \langle \bar{q}, \bar{u}\rangle$ for some descendant $\bar{u}$ of ui.

As before, if M is deterministic, then γ is indeed a partial function. It is straightforward to prove, using Corollary 14, that T is regular; we leave that to the reader.

We define $M' = (\Gamma, \Delta, Q, Q_0, R')$ with the following rules. Let $\rho : \langle q, \sigma, j\rangle \to \zeta$ be a rule of M, and let $\langle \sigma, j, U, \gamma\rangle$ be in Γ (with the same σ and j). If ρ is an output rule or $\zeta = \langle q', \alpha\rangle$ with $\alpha \notin U$, then R' contains the rule $\langle q, \langle \sigma, j, U, \gamma\rangle, j'\rangle \to \zeta$ for every $j' \in [0, mx_\Sigma]$ (except $j' = 0$ when $\alpha = \text{up}$). If $\zeta = \langle q', \alpha\rangle$ with $\alpha \in U$, then R' contains the rule $\langle q, \langle \sigma, j, U, \gamma\rangle, j'\rangle \to \langle \bar{q}, \beta\rangle$ for every $(q, \langle \bar{q}, \beta\rangle) \in \gamma$ and every $j' \in [0, mx_\Sigma]$ (except $j' = 0$ when $\beta = \text{up}$).

Let $\tau = \tau_{N_1} \circ \tau_{N_2}$. It should be clear that $\tau \circ \tau_{M'} \subseteq \tau_M$, as in the previous proof. To understand that $\tau_M^0 \subseteq \tau \circ \tau_{M'}^{01}$, consider a 0-productive computation $\langle q_0, \text{root}_t\rangle \Rightarrow_{M,t}^* s$ with $q_0 \in Q_0$, and let $t' \in \tau(t)$ be obtained from t by removing all superfluous nodes of t. As in the previous proof, there is an obvious bijective correspondence f between the non-superfluous nodes of t and the nodes of t'. For a node u of t we define $g(u) = u$ if u is non-superfluous, and $g(u)$ is the first (i.e., shortest) non-superfluous descendant of u otherwise. Then $f(g(\varepsilon)) = \varepsilon$, and if u is non-superfluous and ui is a child of u, then $f(g(ui)) = f(u)i$. And as before, there is a computation $\langle q_0, \text{root}_{t'}\rangle \Rightarrow_{M',t'}^* s$ of M' on t' that simulates the computation of M on t, such that the configurations $\langle q, u\rangle$ of M, for every non-superfluous node u of t, are simulated by the configurations $\langle q, f(u)\rangle$ of M'. Since τ does not remove leaves of t, the computation of M' is still 0-productive. Moreover, it is also 1-productive because all unproductive monadic nodes were removed by τ. This ends the proof of Lemma 40. □

Remark 41 In the Introduction we observed that our main technical result can be viewed as a static garbage collection procedure, which leads, in principle, to algorithms for automatic compiler and XML query optimization. For practical applicability our proof of this result is, however, of restricted value because the sizes of the involved transducers are blown up exponentially. This is due to the fact that, in the proof of Lemmas 39 and 40, the pruning TT N uses regular tests to determine the relevant state transition information $\gamma \subseteq Q \times Q$ (or $\gamma \subseteq Q \times (Q \times I)$) of the given TT M, due to its ghost computations. These regular tests are constructed through Corollary 14, applied to variants of M. Naturally, the number of states of the finite-state tree automaton recognizing the domain of such a variant is exponential in the number $\#(Q)$ of states of M, cf. the proof of [26, Lemma 1]. If one now considers the proof of $\text{TT} \circ \text{TT} \subseteq \text{TT} * \text{TT}$ in Corollary 38 (in which the pruning TT N for the second TT M is incorporated in the first TT by Theorem 20), it can be seen that the number of states of the first constructed TT is 2-fold exponential in the number of states of M. The additional exponential jump is due to Lemma 12, which turns the pruning TT N into one that is sub-testing. This implies that in the construction for the inclusion $\text{TT} \circ \text{TT}^k \subseteq \text{TT} * \text{TT}^k$ of Corollary 38, the size of the first constructed TT can be $2(k-1)$-fold exponential in the size of the last given TT. This will also hold for the deterministic version. □

8.2 Deterministic productivity

Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a deterministic TT. For $t \in \mathrm{dom}(M)$ we say that a node u of t is productive if it is productive in the computation $\langle q_0, \mathrm{root}_t\rangle \Rightarrow^*_{M,t} \tau_M(t)$, and we say that t is productive (for M) if that computation is productive, i.e., if all leaves and monadic nodes of t are productive.[11] We define $L_{M,\mathrm{prod}}$ to be the set of all productive trees $t \in \mathrm{dom}(M)$. Note that τ^{01}_M is the restriction of τ_M to $L_{M,\mathrm{prod}}$. The next lemma shows that the set of productive input trees is a regular tree language.

Lemma 42 *Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a deterministic* TT.

(1) *There is a regular test $T_{M,\mathrm{prod}}$ over Σ such that for every $t \in \mathrm{dom}(M)$ and $u \in \mathcal{N}(t)$, $(t, u) \in T_{M,\mathrm{prod}}$ if and only if u is productive.*
(2) *$L_{M,prod}$ is a regular tree language over Σ.*

Proof (1) Let $M' = (\Sigma \times \{0, 1\}, \{\top\}, Q, \{q_0\}, R')$ be the nondeterministic TT such that $\top$ has rank 0, and R' is defined as follows. If $\langle q, \sigma, j, T\rangle \to \langle q', \alpha\rangle$ is a move rule in R, then $\langle q, (\sigma, b), j, \mu(T)\rangle \to \langle q', \alpha\rangle$ is a rule in R' for every $b \in \{0, 1\}$. If $\langle q, \sigma, j, T\rangle \to \delta(\langle q_1, \alpha_1\rangle, \ldots, \langle q_k, \alpha_k\rangle)$ is an output rule in R, then R' contains the rules $\langle q, (\sigma, 0), j, \mu(T)\rangle \to \langle q_i, \alpha_i\rangle$ for every $i \in [k]$ and it also contains the rule $\langle q, (\sigma, 1), j, \mu(T)\rangle \to \top$. Intuitively, for an input tree $\mathrm{mark}(t, u)$ with $t \in \mathrm{dom}(M)$, the tree-walking automaton M' follows an arbitrary path in the unique derivation tree $d \in L(G^{\mathrm{der}}_{M,t})$, from the root of d down to the leaves (cf. M'_1 and N in the proof of Lemma 22). Whenever M branches at an unmarked node, M' nondeterministically follows one of those branches. It accepts $\mathrm{mark}(t, u)$ when an output rule is applied to the marked node u. It should be clear that $T_{M,\mathrm{prod}} = \mathrm{mark}^{-1}(\mathrm{dom}(M'))$ satisfies the requirements. It is regular by Corollary 14.

(2) Let M'' be a dTT that performs a depth-first left-to-right traversal of the input tree $t \in T_\Sigma$ and verifies that $(t, u) \in T_{M,\mathrm{prod}}$ for every leaf and monadic node u of t. Then $L_{M,\mathrm{prod}} = \mathrm{dom}(M) \cap \mathrm{dom}(M'')$, which is regular by Corollary 14. □

For a given deterministic TT M there are a nondeterministic pruning TT N and a deterministic TT$^\ell$ M' such that $\tau_N \circ \tau_{M'} = \tau_M$ and $\tau_M \subseteq \tau_N \circ \tau^{01}_{M'}$, by Lemmas 39 and 40. Our aim is to transform N and M' in such a way that N becomes deterministic. We basically do this by applying Lemma 34 to τ_N, replacing it by one of its uniformizers. But to preserve the above two properties we first restrict the domain of M' to productive input trees and then restrict the range of N to the new domain, as follows.

By Lemma 42, the tree language $L_{M',\mathrm{prod}}$ is regular. Let M'' be the dTT that is obtained from M' by restricting its domain to $L_{M',\mathrm{prod}}$, see Lemma 9. Hence, $\tau_{M''} = \tau^{01}_{M'}$ and so $\tau_N \circ \tau_{M''} = \tau_M$. Since M'' behaves in the same way as M', every tree $t' \in \mathrm{dom}(M'')$ is productive (for M''). Next, we change N into the nondeterministic pruning TT N' by restricting its range to $\mathrm{dom}(M'')$, by Corollary 21. Now $\tau_{N'} \circ \tau_{M''} = \tau_M$ and $\mathrm{ran}(\tau_{N'}) \subseteq \mathrm{dom}(\tau_{M''})$. Finally, we define $\tau \in \mathsf{dTT}_{\mathrm{pru}}$ to be the uniformizer of $\tau_{N'}$ according to Lemma 34. Then $\tau \circ \tau_{M''} = \tau_M$. Now consider $(t, s) \in \tau_M$. Then $s = \tau_{M''}(r)$ for $r = \tau(t)$. Since r is productive for M'', it follows that $|r| \leq 2 \cdot |s|$ as observed at the end of the second paragraph of Sect. 8.1. Hence $(\tau, \tau_{M''})$ is linear-bounded, which shows that $\tau_M \in \mathsf{dTT}_{\mathrm{pru}} * \mathsf{dTT}$.

[11] There are several such computations, but they all have the same unique derivation tree in $L(G^{\mathrm{der}}_{M,t})$. The definition of productivity clearly does not depend on the particular choice of the derivation.

9 Linear size increase

In this section we show our first main result: the hierarchy of TT's collapses for functions of linear size increase.

Theorem 43 *For every $k \geq 1$, $\mathsf{dTT}^k \cap \mathsf{LSIF} = \mathsf{dTT}_{\mathrm{su}}$.*

Proof The proof is by induction on k. For $k = 1$ it is Corollary 32. To prove that $\mathsf{dTT}^{k+1} \cap \mathsf{LSIF} \subseteq \mathsf{dTT}_{\mathrm{su}}$, let $\tau \in \mathsf{dTT}^k$ and let M be a dTT such that $\tau_M \circ \tau \in \mathsf{LSIF}$. By Corollary 38(2) we may assume that (τ_M, τ) is linear-bounded. Moreover, by restricting the domain of M to $\mathrm{dom}(\tau_M \circ \tau)$ we may assume that $\mathrm{ran}(\tau_M) \subseteq \mathrm{dom}(\tau)$, see Lemma 9 and Corollary 14. Hence $\tau_M \in \mathsf{LSIF}$ by Lemma 2 and so $\tau_M \in \mathsf{dTT}_{\mathrm{su}}$ by Corollary 32. Then $\tau_M \circ \tau \in \mathsf{dTT}^k$ by Theorem 23. Hence $\tau_M \circ \tau \in \mathsf{dTT}_{\mathrm{su}}$ by induction. □

Theorem 44 *It is decidable for a composition of deterministic* TT*'s whether or not it is of linear size increase.*

Proof The proof is, again, by induction on k, the number of dTT's in the composition. It goes along the lines of the proof of Theorem 43, using Corollary 33 instead of Corollary 32 for the case $k = 1$. Assuming that we have an algorithm $\mathcal{A}_k$ for a composition of k dTT's, we construct $\mathcal{A}_{k+1}$ as follows. Let $M, M_1, \ldots, M_k$ be dTT's, $k \geq 1$, and let $\tau = \tau_{M_1} \circ \cdots \circ \tau_{M_k}$. Since all our results are effective, we may assume as in the proof of Theorem 43 that (τ_M, τ) is linear-bounded and $\mathrm{ran}(\tau_M) \subseteq \mathrm{dom}(\tau)$. To decide whether or not $\tau_M \circ \tau$ is of linear size increase, we first decide whether or not τ_M is of linear size increase by Corollary 33. If not, then $\tau_M \circ \tau$ is not of linear size increase, by Lemma 2. If so, then a dTT M_1' that realizes $\tau_M \circ \tau_{M_1}$ can be constructed by Corollary 32 and Theorem 23, and we apply $\mathcal{A}_k$ to $M_1', M_2, \ldots, M_k$. □

Together with Lemma 24 and Proposition 29 in Sect. 6, Theorems 43 and 44 imply the following two corollaries on macro tree transducers.

Corollary 45 *For every $k \geq 1$, $\mathsf{dMT}^k \cap \mathsf{LSIF} = \mathsf{dMSOT} = \mathsf{dTT}_{\mathrm{su}} \subseteq \mathsf{dMT}$.*

Corollary 46 *It is decidable for a composition of deterministic* MT*'s whether or not it is of linear size increase.*

For the class $\mathsf{dMT}_{\mathrm{IO}}$ of translations realized by deterministic macro tree transducers with inside-out (IO) derivation mode, we obtain that $\mathsf{dMT}_{\mathrm{IO}}^k \cap \mathsf{LSIF} \subseteq \mathsf{dTT}_{\mathrm{su}}$ for every $k \geq 1$, for the simple reason that $\mathsf{dMT}_{\mathrm{IO}}$ is a (proper) subclass of dMT by [34, Theorem 7.1(1)]. For the same reason Corollary 46 is also valid for those transducers. However, $\mathsf{dTT}_{\mathrm{su}}$ is not included in $\mathsf{dMT}_{\mathrm{IO}}$, because not every regular tree language is the domain of a deterministic IO macro tree transducer (see [34, Corollary 5.6]).

Since $\mathsf{LSIF} \subseteq \mathcal{F}$, it follows from Theorems 43 and 35 that Theorem 43 also holds for nondeterministic TT's, i.e., $\mathsf{TT}^k \cap \mathsf{LSIF} = \mathsf{dTT}_{\mathrm{su}}$ for every $k \geq 1$.[12] Similarly, it follows from Corollaries 36 and 45 that Corollary 45 also holds for nondeterministic MT's, i.e., $\mathsf{MT}^k \cap \mathsf{LSIF} = \mathsf{dMSOT} = \mathsf{dTT}_{\mathrm{su}} \subseteq \mathsf{dMT}$ for every $k \geq 1$. This even holds for the so-called stay-macro tree transducers that can use stay-instructions, introduced in [31, Section 5.3], because it is shown in [31, Lemma 37] that the stay-macro tree translations are in TT^4. For the class

[12] We do not know whether Theorem 44 holds for nondeterministic TT's, i.e., whether it is decidable for a composition of nondeterministic TT's whether or not it realizes a translation in LSIF.

$\mathsf{MT}_{\mathrm{IO}}$ of nondeterministic IO macro tree translations we also obtain that $\mathsf{MT}_{\mathrm{IO}}^k \cap \mathsf{LSIF} \subseteq \mathsf{dTT}_{\mathrm{su}}$ for every $k \geq 1$, because $\mathsf{MT}_{\mathrm{IO}} \subseteq \mathsf{TT}^2$ by Lemma 28; the same is true for multi-return macro tree transducers.

The *k-pebble tree transducer* was introduced in [63] as a model of XML document transformation. It is a TT that additionally can use k distinct pebbles to drop on, and lift from, the nodes of the input tree. The life times of these pebbles must be nested. The TT is the 0-pebble tree transducer. It is shown in [31, Theorem 10] that every (deterministic) k-pebble tree translation can be realized by a composition of (deterministic) $k+1$ TT's. Hence Theorems 43 and 44 also hold for deterministic k-pebble tree transducers, while Theorem 43 additionally holds for the nondeterministic case. In [28, Theorems 5 and 55] this is extended to k-pebble tree transducers that, in addition to the k distinct "visible" pebbles, can use an arbitrary number of "invisible" pebbles, still with nested life times: they can be realized by a composition of $k+2$ TT's. Thus, Theorems 43 and 44 also hold for such transducers, cf. [28, Theorem 57].[13]

The *high-level tree transducer* was introduced in [35] as a generalization of both the top-down tree transducer and the macro tree transducer. It is proved in [35, Theorem 8.1(b)] that nondeterministic high-level tree transducers can be simulated by compositions of nondeterministic MT's. Since every deterministic high-level tree transducer realizes a partial function (as should be clear from the proof of [35, Lemma 5.7]), it follows from Corollary 36 that, similarly, deterministic high-level tree transducers can be simulated by compositions of deterministic MT's. Consequently, Corollaries 45 and 46 also hold for deterministic high-level tree transducers, and Corollary 45 additionally for the nondeterministic case.

10 Deterministic complexity

Our first main complexity result says that a composition of deterministic TT's can be computed by a RAM program in linear time, more precisely in time $O(n)$ where n is the sum of the sizes of the input and the output tree.

Theorem 47 *For every $k \geq 1$ and every $\tau \in \mathsf{dTT}^k$ there is an algorithm that computes, given an input t, the output $s = \tau(t)$ in time $O(|t| + |s|)$.*

Proof The proof is by induction on k. We first prove the case $k = 1$, which is a slight generalization of the well-known fact for attribute grammars that the attribute evaluation of an input tree takes linear time (see, e.g., [17,23]). Let $\tau \in \mathsf{dTT}$ and let t be an input tree of τ. By Corollary 14, $\mathrm{dom}(\tau)$ is regular and hence can be recognized by a bottom-up finite-state tree automaton. Thus, we can decide whether or not $t \in \mathrm{dom}(\tau)$ in time $O(|t|)$ by running that automaton on t. By Lemmas 10 and 12, $\tau = \tau_1 \circ \tau_2$ with $\tau_1 \in \mathsf{dTT}_{\mathrm{rel}}^{\mathrm{s}}$ and $\tau_2 \in \mathsf{dTT}^{\ell}$. As observed in Sect. 3, τ_1 can be realized by a classical linear deterministic top-down tree transducer with regular look-ahead. Thus, by (the proof of) [20, Theorem 2.6], it can be realized by a deterministic bottom-up finite-state relabeling (DBQREL) and a local relabeling TT. To run these two relabelings on $t \in \mathrm{dom}(\tau)$ obviously takes time $O(|t|)$. Thus, it remains to consider the case that $\tau \in \mathsf{dTT}^{\ell}$. Let M be a local dTT that realizes τ. To compute $\tau_M(t)$, we first construct the regular tree grammar $G_{M,t}$ in time $O(|t|)$, the number of configurations of M on t. Then we remove the chain rules from the context-free grammar

[13] A "visible" pebble can be observed by the transducer during its entire life time (as usual for pebbles), whereas an "invisible" pebble p cannot be observed during the life time of a pebble p' of which the life time is nested within the one of p; thus, such a pebble p' "hides" the pebble p.

$G_{M,t}$, i.e., the rules $\langle q,u\rangle \to \langle q',u'\rangle$ resulting from the move rules of M. Since $G_{M,t}$ is forward deterministic, this can also be done in time $O(|t|)$, as follows. Viewing the chain rules as edges of a directed graph with configurations as nodes, we compute an evaluation order of the graph by topological sorting, in time $O(|t|)$. Then we compute the new rules by traversing this order from right to left, again in time $O(|t|)$. For an edge $\langle q,u\rangle \to \langle q',u'\rangle$, if the (old or new) rule for $\langle q',u'\rangle$ is $\langle q',u'\rangle \to \delta(\langle q_1,u_1\rangle,\ldots,\langle q_k,u_k\rangle)$, then the new rule for $\langle q,u\rangle$ is $\langle q,u\rangle \to \delta(\langle q_1,u_1\rangle,\ldots,\langle q_k,u_k\rangle)$. Finally, we use this new regular tree grammar, equivalent to $G_{M,t}$, to generate $s = \tau_M(t)$, which takes time $O(|s|)$ because each rule generates a node of s.

Now let $\tau = \tau_1 \circ \tau_2$ such that $\tau_1 \in \mathsf{dTT}$ and $\tau_2 \in \mathsf{dTT}^k$, $k \geq 1$. By Corollary 38(2) we may assume that (τ_1,τ_2) is linear-bounded. Let t be an input tree of τ. Since $\mathrm{dom}(\tau)$ is regular by Corollary 14, we can check that $t \in \mathrm{dom}(\tau)$ in linear time, as above. By the case $k=1$, the intermediate tree $r = \tau_1(t)$ can be computed in time $O(|t|+|r|)$, and by induction the output tree $s = \tau(t) = \tau_2(r)$ can be computed in time $O(|r|+|s|)$. Since (τ_1,τ_2) is linear-bounded, there is a constant $c \in \mathbb{N}$ such that $|r| \leq c\cdot|s|$, i.e., $|r| = O(|s|)$. Hence the total time is $O(|t|+|r|)+O(|r|+|s|) = O(|t|+|s|)$. □

It should be noted that the constant in the time complexity $O(|t|+|s|)$ can be large in terms of the size of the given transducers due to the use of linear-boundedness, cf. Remark 41.

Since deterministic macro tree transducers, pebble tree transducers, and high-level tree transducers can be realized as compositions of deterministic TT's (see Sect. 9), Theorem 47 also holds for such transducers. For k-pebble tree transducers this improves the result of [63, Proposition 3.5], where the time bound is $O(|t|^k + |s|)$.

Before we proceed, we need an elementary lemma on leftmost derivations of context-free grammars. For a context-free grammar $G = (N,T,\mathcal{S},R)$, a leftmost sentential form is a string $v \in (N\cup T)^*$ such that $S \Rightarrow^*_{G,\mathrm{lm}} v$ for some $S \in \mathcal{S}$, where $\Rightarrow_{G,\mathrm{lm}}$ is the usual leftmost derivation relation of G: if $X \to \zeta$ is in R, then $v_1 X v_2 \Rightarrow_{G,\mathrm{lm}} v_1\zeta v_2$ for all $v_1 \in T^*$ and $v_2 \in (N\cup T)^*$.

Lemma 48 *Let $G = (N,T,\mathcal{S},R)$ be an ε-free context-free grammar, and let $G' = (N',T,\mathcal{S},R')$ be the equivalent context-free grammar such that $N' = N \cup \{Z\}$ and $R' = \{X \to \zeta Z \mid X \to \zeta \in R\} \cup \{Z \to \varepsilon\}$, where Z is a new nonterminal. Let v be a leftmost sentential form of G', and let $S \Rightarrow^*_{G',\mathrm{lm}} v \Rightarrow^*_{G',\mathrm{lm}} w$ be a leftmost derivation of G' with $S \in \mathcal{S}$ and $w \in L(G)$. Moreover, let d be the derivation tree corresponding to that derivation. Then the number of occurrences of Z in v is at most the height of d.*[14]

Proof Each occurrence of a nonterminal $Y \in N'$ in v corresponds to a node of d with label Y in a well-known way. Let u be the node of d corresponding to the leftmost occurrence of Z in v. Clearly the number of occurrences of Z in v is equal to the number of edges on the path from u to the root of d. □

By [64, Theorem 2.5] it follows from Theorem 47 that a composition of deterministic TT's can be computed by a deterministic Turing machine in cubic time, more precisely in time $O(n^3)$ where n is the sum of the sizes of the input and the output tree. Our second complexity result says that a composition of deterministic TT's can be computed by a deterministic multi-tape Turing machine N in linear space (in the sum of the sizes of the input and output tree). On a work tape of N we will represent the input tree t over Σ by the string $\varphi(t)$ over

[14] Note that there is a straightforward one-to-one correspondence between the leftmost derivations of G and G', and between their derivation trees. Since G is ε-free, the derivation trees have the same height.

$\Sigma \cup \{(,)\}$, where $\{(,)\}$ is the set consisting of the left- and right-parenthesis, defined such that if $\varphi(t_1) = t'_1, \ldots, \varphi(t_m) = t'_m$ then $\varphi(\sigma t_1 \cdots t_m) = \sigma(t'_1 \cdots t'_m)$. In other words, we formally insert the parentheses (but not the commas) that are always used informally to denote trees. The parentheses allow N to walk on the tree t, from node to node, because it can recognize a subtree of t by checking that the numbers of left- and right-parentheses in the corresponding substring of $\varphi(t)$ are equal. In particular, it can determine the child number of a node of t by counting the number of its younger siblings. Obviously, the mapping φ is injective, and can be computed in linear space (simulating a one-way push-down transducer). In what follows we identify t and $\varphi(t)$.

Theorem 49 *For every $k \geq 1$ and every $\tau \in \mathsf{dTT}^k$ there is a deterministic Turing machine that computes, given an input t, the output $s = \tau(t)$ in space $O(|t| + |s|)$.*

Proof Again, we first show this for $k = 1$. Let $M = (\Sigma, \Delta, Q, q_0, R)$ be a dTT, and let $t \in T_\Sigma$ be an input tree. As usual we assume that the output rules of M only contain stay-instructions. We describe a deterministic multi-tape Turing machine N that computes τ_M in linear space. By Corollary 14, $\mathrm{dom}(M)$ is a regular tree language and hence a context-free language, which can be recognized in deterministic linear space. Thus, N starts by deciding whether or not $t \in \mathrm{dom}(M)$. Now assume that $t \in \mathrm{dom}(M)$. To compute $s = \tau_M(t)$, the machine N simulates the (unique) leftmost derivation of the forward deterministic context-free grammar $G_{M,t}$. Every leftmost sentential form of $G_{M,t}$ is of the form $w\langle q_1, u_1\rangle \cdots \langle q_n, u_n\rangle$ with $w \in \Delta^*$ and $\langle q_i, u_i\rangle \in \mathrm{Con}(t)$. If one views the states of M as recursive procedures with one parameter of type 'node of t', then $\langle q_1, u_1\rangle \cdots \langle q_n, u_n\rangle$ corresponds to the contents of the stack in the usual implementation of recursive procedures: each configuration $\langle q_i, u_i\rangle$ is a call of procedure q_i with actual parameter u_i. The machine N uses a one-way output tape on which it prints w (which will finally be s), a work tape with the input tree t (or rather $\varphi(t)$), and a work tape that contains a stack representing $\langle q_1, u_1\rangle \cdots \langle q_n, u_n\rangle$, with the top of the stack to the left. At each moment of time, a reading head of N is at node u_1 of t, and another reading head is at the top of the stack. Note that $n \leq |s|$ because every configuration $\langle q_i, u_i\rangle$ will generate at least one symbol of s. If N would represent the parameters $u_2, \ldots, u_n$ by their Dewey notation, the size of the stack could be $|s| \cdot |t|$, which is too much. Thus, we need a more compact representation of the nodes $u_2, \ldots, u_n$. In a rule of $G_{M,t}$ with left-hand side $\langle q, u\rangle$, every node u' in the right-hand side is a neighbour of u, or u itself, and so, the "difference" between u and u' can be expressed by an instruction in $I = \{\mathrm{up}, \mathrm{stay}\} \cup \{\mathrm{down}_i \mid i \in [1, mx_\Sigma]\}$. This allows us to represent $\langle q_1, u_1\rangle \cdots \langle q_n, u_n\rangle$ by the node u_1 and a stack of the form $q_1\gamma_1 q_2\gamma_2 \cdots q_n\gamma_n$ where $\gamma_i \in I^*$ is a sequence of instructions that leads from u_i to u_{i+1} (with $u_{n+1} = \mathrm{root}_t$). Let us now consider in detail how N simulates the leftmost derivation of $G_{M,t}$.

At each moment of time, the current node of t and the current contents of the output tape and the stack tape represent a leftmost sentential form of $G_{M,t}$, which is an element of $\Delta^* \cdot \mathrm{Con}(t)^*$. The stack tape contains a string in $(Q \cup I)^*\bot$, where $\bot$ is the bottom stack symbol and I is as above. The current node u of t and the current contents $w \in \Delta^*$ and $\xi \in (Q \cup I)^*\bot$ of the output tape and stack tape, respectively, represent the leftmost sentential form $w \cdot \mu(u, \xi)$, where the string $\mu(u, \xi) \in \mathrm{Con}(t)^*$ is defined as follows (for every $q \in Q$ and $\beta \in I$): $\mu(u, q\xi) = \langle q, u\rangle \cdot \mu(u, \xi)$, $\mu(u, \beta\xi) = \mu(\beta(u), \xi)$, and $\mu(\bot) = \varepsilon$. Initially, N starts at the root of t, with empty output tape and with stack tape $q_0\bot$, representing the initial output form $\langle q_0, \mathrm{root}_t\rangle$. If the top symbol of the stack is $\bot$, then N halts. Otherwise, to compute the next leftmost sentential form, N first pops the top symbol off the stack. If that symbol was $q \in Q$, and the current node u of t has label σ and child number j, then N selects the unique rule $\langle q, \sigma, j, T\rangle \to \zeta$ that is applicable to $\langle q, u\rangle$. Note that it can test in linear space whether or not $(t, u) \in T$, because mark(T) is a context-free language. If $\zeta = \langle q', \alpha\rangle$, then N moves to node $\alpha(u)$ of t and pushes the string $q'\beta$ on the stack where β is defined as follows: if α is up, stay, or down$_i$, then β is down$_j$, stay, or up, respectively. If $\zeta = \delta(\langle q_1, \text{stay}\rangle, \dots, \langle q_k, \text{stay}\rangle)$, then N outputs δ, and pushes $q_1 \cdots q_k$ on the stack (if $k > 0$). It is easy to check that in both these cases the resulting configuration of N represents the next leftmost sentential form of $G_{M,t}$. If the top symbol of the stack was $\beta \in I$, the machine N moves to node $\beta(u)$ of t. This does not change the represented leftmost sentential form. Thus, after applying a rule $\langle q, \sigma, j, T\rangle \to \delta$ (with δ of rank 0), N removes instructions from the stack (and moves its reading head on t accordingly) until the top of the stack is a state again. When N halts, the output tape contains s.

It remains to show that the length of the stack is linear in $|t|+|s|$. As mentioned above, since every configuration $\langle q, u\rangle$ will generate at least one symbol of s, the number of occurrences of states in the stack is at most $|s|$. To estimate the number of occurrences of instructions in the stack, we use Lemma 48. In the above case where q is the top stack symbol and $\langle q, \sigma, j, T\rangle \to \langle q', \alpha\rangle$ is the rule applicable to $\langle q, u\rangle$, the machine N does not apply the rule $\langle q, u\rangle \to \langle q', \alpha(u)\rangle$ of $G_{M,t}$, but rather the rule $\langle q, u\rangle \to \langle q', \alpha(u)\rangle\beta$ where β is defined as above. Moreover, when β is the top stack symbol, N applies the rule $\beta \to \varepsilon$. From this it should be clear that, by Lemma 48 and footnote 14, the number of occurrences of instructions in the stack is at most the height of the derivation tree corresponding to the derivation $\langle q_0, \text{root}_t\rangle \Rightarrow^*_{M,t} s$ of $G_{M,t}$. As observed in Sect. 2 after Lemma 3, that height is at most $\#(\text{Con}(t))$, i.e., $\#(Q) \cdot |t|$. Thus, the length of the stack is indeed $O(|s| + |t|)$.

The induction step can be proved in exactly the same way as in the proof of Theorem 47, with 'time' replaced by 'space'. ◻

For a class $\mathcal{T}$ of tree translations and a class $\mathcal{L}$ of tree languages, we denote by $\mathcal{T}(\mathcal{L})$ the class of tree languages $\tau(L)$ with $\tau \in \mathcal{T}$ and $L \in \mathcal{L}$. The elements of $\mathcal{T}(\mathsf{REGT})$ are called the output tree languages (or surface languages) of $\mathcal{T}$. Since $\mathsf{dTT} \subseteq \mathsf{dMT}$ by Lemma 24, it follows from the proof of [34, Theorem 7.5] that the output tree languages of dTT^k are recursive. From Theorem 49 we now obtain that they are in $\mathsf{DSPACE}(n)$, i.e., can be recognized by a Turing machine in deterministic linear space. This was shown for classical top-down tree transducers in [4].

Theorem 50 *For every $k \geq 1$, $\mathsf{dTT}^k(\mathsf{REGT}) \subseteq \mathsf{DSPACE}(n)$.*

Proof Let $L \in \mathsf{REGT}$ and $\tau \in \mathsf{dTT}^k$. By Corollary 38(2), $\tau = \tau_1 \circ \tau_2$ such that $\tau_1 \in \mathsf{dTT}_{\text{pru}}$, $\tau_2 \in \mathsf{dTT}^k$, and (τ_1, τ_2) is linear-bounded for some constant c. Let $L' = \tau_1(L)$, and note that $\tau(L) = \tau_2(L')$ and that $L' \in \mathsf{REGT}$ by Lemma 15. It is straightforward to show that for every $s \in \tau(L)$ there exists $t \in L'$ such that $(t, s) \in \tau_2$ and $|t| \leq c \cdot |s|$. To check whether a given tree s is in $\tau(L)$, a deterministic Turing machine systematically enumerates all input trees t (of τ_2) such that $|t| \leq c \cdot |s|$. For each such t it first checks that $t \in L'$ in space $O(|t|)$. Then it uses the algorithm of Theorem 49 to compute $\tau_2(t)$ in space $c' \cdot (|t| + |\tau_2(t)|)$, but rejects t as soon as the computation takes more than space $c' \cdot (|t| + |s|)$; thus, the space used is $O(|t| + |s|) = O(|s|)$. Clearly, $s \in \tau(L)$ if and only if $\tau_2(t) = s$ for some such t. ◻

For a tree t we denote its yield by yt, for a tree language L we define $yL = \{yt \mid t \in L\}$, and for a class $\mathcal{L}$ of tree languages we define $y\mathcal{L} = \{yL \mid L \in \mathcal{L}\}$. For a class $\mathcal{T}$ of tree translations, the languages in $y\mathcal{T}(\mathsf{REGT})$ are called the output string languages (or target languages) of $\mathcal{T}$.

Corollary 51 *For every $k \geq 1$, $y\mathsf{dTT}^k(\mathsf{REGT}) \subseteq \mathsf{DSPACE}(n)$.*

Proof For an alphabet Δ, let $\Gamma = \Delta \cup \{e\}$ be the ranked alphabet such that e has rank 0 and every element of Δ has rank 1. For a string w over Δ we define $\text{mon}(w) = we \in T_\Gamma$. It is easy to see that for every ranked alphabet Σ there is a dTT^ℓ M such that $\tau_M(t) = \text{mon}(yt)$. From this and Theorem 50 the result follows. □

We observe here, for $k = 1$, that $\mathsf{dTT}(\mathsf{REGT})$ and $y\mathsf{dTT}(\mathsf{REGT})$ are included in LOGCFL, the class of languages that are log-space reducible to a context-free language. This will be proved in Corollaries 64 and 65. Note that $\mathsf{LOGCFL} \subseteq \mathsf{DSPACE}(\log^2 n)$.

We also observe that Theorem 50 and Corollary 51 also hold for nondeterministic TT's, as will be proved in Theorem 67 (and was proved for classical top-down tree transducers in [4]).

As before, Theorems 49 and 50 and Corollary 51 also hold for deterministic macro tree transducers, pebble tree transducers, and high-level tree transducers. It is proved in [30, Theorem 23] that composition of deterministic MT's yields a proper hierarchy of output string languages (called the $y\mathsf{dMT}$-hierarchy), i.e., that $y\mathsf{dMT}^k(\mathsf{REGT}) \subsetneq y\mathsf{dMT}^{k+1}(\mathsf{REGT})$ for every $k \geq 1$. The IO-*hierarchy* consists of the classes of string languages $\mathsf{IO}(k)$ generated by level-k grammars, with the inside-out (IO) derivation mode (see, e.g., [16]). By [33, Theorem 7.5] the IO-hierarchy can be defined as output string languages of tree transformations: $\mathsf{IO}(k) = y\mathsf{YIELD}^k(\mathsf{REGT})$. Since $\mathsf{YIELD} \subseteq \mathsf{dTT}$ by [31, Lemma 36], we obtain that $\mathsf{IO}(k) \subseteq y\mathsf{dTT}^k(\mathsf{REGT})$. Thus, the next corollary is immediate from Corollary 51. Note that it was already proved in [37, Theorem 3.3.8] that the IO languages (i.e., the languages in $\mathsf{IO}(1)$) are in $\mathsf{NSPACE}(n)$; in [3] this was improved to LOGCFL. It was proved in [16, Corollary 8.12] that the languages in the IO-hierarchy are recursive.

Corollary 52 *For every $k \geq 1$, $\mathsf{IO}(k) \subseteq \mathsf{DSPACE}(n)$.*

Note that by [30, Theorem 36] the EDTOL control hierarchy is included in the IO-hierarchy.

By Corollary 25, $y\mathsf{dTT}^k(\mathsf{REGT}) \subseteq y\mathsf{dMT}^k(\mathsf{REGT}) \subseteq y\mathsf{dTT}^{k+1}(\mathsf{REGT})$. It is proved in [30, Theorem 32] that there exists a language in $\mathsf{IO}(k+1)$ that is not in $y\mathsf{dMT}^k(\mathsf{REGT})$. Since $\mathsf{IO}(k+1) \subseteq y\mathsf{dTT}^{k+1}(\mathsf{REGT})$, that implies the following stronger version of Proposition 7.

Corollary 53 *For every $k \geq 1$, $y\mathsf{dTT}^k(\mathsf{REGT}) \subsetneq y\mathsf{dTT}^{k+1}(\mathsf{REGT})$.*

11 Nondeterministic complexity

We now turn to the complexity of compositions of nondeterministic TT's. We first consider the case where all the transducers in the composition are finitary. The next lemma shows that Theorem 37 and Corollary 38 also hold for fTT.

Lemma 54 $\mathsf{fTT}^k \subseteq \mathsf{TT}_{\text{pru}} * \mathsf{fTT}^k$ *and* $\mathsf{fTT} \circ \mathsf{fTT}^k = \mathsf{fTT} * \mathsf{fTT}^k$ *for every $k \geq 1$.*

Proof To show that $\mathsf{fTT} \subseteq \mathsf{TT}_{\text{pru}} * \mathsf{fTT}$, let $\tau \in \mathsf{fTT}$. By Theorem 37, $\tau = \tau_1 \circ \tau_2$ such that $\tau_1 \in \mathsf{TT}_{\text{pru}}$, $\tau_2 \in \mathsf{TT}$, and (τ_1, τ_2) is linear-bounded. Since $\text{ran}(\tau_1) \in \mathsf{REGT}$ by Lemma 15, we may assume that $\text{dom}(\tau_2) \subseteq \text{ran}(\tau_1)$ by Lemma 9. Then τ_2 is finitary too.

Theorem 20 implies that $\mathsf{fTT} \circ \mathsf{TT}_{\text{pru}} \subseteq \mathsf{fTT}$, because the composition of two finitary translations is finitary. The remainder of the proof is now entirely similar to the one of Corollary 38. □

We will prove that a composition of TT's can be computed by a nondeterministic Turing machine in linear space and polynomial time (in the sum of the sizes of the input and output tree), which generalizes Theorem 49. In the next lemma we consider the case where all TT's are finitary.

Lemma 55 *For every $k \geq 1$ and every $\tau \in \mathsf{fTT}^k$ there is a nondeterministic Turing machine that computes, given an input t, any output $s \in \tau(t)$ in space $O(|t| + |s|)$ and in time polynomial in $|t| + |s|$.*

Proof For the case $k = 1$ the proof is exactly the same as that of Theorem 49 except, of course, that the Turing machine N nondeterministically simulates any leftmost derivation of $G_{M,t}$, selecting nondeterministically a rule of M to compute a next leftmost sentential form. It follows from Lemmas 3 and 48 that the number n of occurrences of instruction symbols in the stack is $O(|t|)$. In fact, since M is finitary, it suffices by Lemma 3 to simulate leftmost derivations of $G_{M,t}$ for which the corresponding derivation tree in $L(G^{\text{der}}_{M,t})$ has height at most $\#(Q) \cdot |t|$. As in the proof of Theorem 49, Lemma 48 implies that n is at most that height, i.e., at most $\#(Q) \cdot |t|$. Thus, N works in space $O(|t| + |s|)$. Moreover, it works in time $O(|t|^2 \cdot |s|)$, because the size of such a derivation tree (and hence the length of the leftmost derivation) is at most $\#(Q) \cdot |t| \cdot |s|$, and each step in the leftmost derivation takes time $O(|t|)$. Note that regular tree languages (which are context-free languages) can be recognized in nondeterministic linear time.

Now let $\tau = \tau_1 \circ \tau_2$ such that $\tau_1 \in \mathsf{fTT}$ and $\tau_2 \in \mathsf{fTT}^k$, $k \geq 1$. We may assume by Lemma 54 that (τ_1, τ_2) is linear-bounded. So, there is a constant $c \in \mathbb{N}$ such that for every $(t, s) \in \tau$ there exists a tree r such that $(t, r) \in \tau_1$, $(r, s) \in \tau_2$, and $|r| \leq c \cdot |s|$. By the case $k = 1$, the intermediate tree r can be computed from t in nondeterministic space $O(|t| + |r|)$, and by induction, the output tree s can be computed from r in nondeterministic space $O(|r| + |s|)$. Hence, since $|r| = O(|s|)$, s can be computed from t in nondeterministic space $O(|t| + |s|)$. The time is polynomial in $|t| + |r|$ and $|r| + |s|$, and hence polynomial in $|t| + |s|$. □

By Lemma 27, $\mathsf{MT} \subseteq \mathsf{fTT}^2$. Consequently Lemma 55 also holds for every $\tau \in \mathsf{MT}^k$.

We now turn to the output languages of fTT^k. By $\mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$ we will denote the class of languages that can be recognized by a nondeterministic Turing machine in simultaneous linear space and polynomial time. Trivially, $\mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$ is included in both $\mathsf{NSPACE}(n)$ and NPTIME.

Lemma 56 *For every $k \geq 1$, $\mathsf{fTT}^k(\mathsf{REGT}) \subseteq \mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$.*

Proof The proof is similar to the one of Theorem 50. Let $L \in \mathsf{REGT}$ and $\tau \in \mathsf{fTT}^k$. By Lemma 54, $\tau = \tau_1 \circ \tau_2$ where $\tau_1 \in \mathsf{TT}_{\text{pru}}$, $\tau_2 \in \mathsf{fTT}^k$, and (τ_1, τ_2) is linear-bounded for some constant c. Let $L' = \tau_1(L)$. Then $s \in \tau(L)$ if and only if there exists $t \in L'$ such that $(t, s) \in \tau_2$ and $|t| \leq c \cdot |s|$. To check whether a given tree s is in $\tau(L)$, a nondeterministic Turing machine guesses an input tree t such that $|t| \leq c \cdot |s|$, it checks that $t \in L'$ in time and space $O(|t|)$ (because L' is a context-free language), and then computes any $s' \in \tau(t)$ with $|s'| \leq |s|$ in space $O(|t| + |s'|)$ and time polynomial in $|t| + |s'|$, by Lemma 55. Finally it checks that $s' = s$ in time and space $O(|s|)$. Thus the space used is $O(|t| + |s|) = O(|s|)$, and the time is polynomial in $|s|$. □

Although MT's are finitary, whereas TT's need not be finitary, it is proved in [31, Theorem 38 and Corollary 39] that compositions of MT's have the same output languages as compositions of (local) TT's. This implies that Lemmas 55 and 56 also hold for TT^k.

Theorem 57 *For every $k \geq 1$, $\mathsf{TT}^k(\mathsf{REGT}) \subseteq \mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$, and moreover, $\mathsf{MT}^k(\mathsf{REGT}) \subseteq \mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$.*

Proof By Lemma 27, $\mathsf{MT} \subseteq \mathsf{fTT}^2$. Thus, by Lemma 56, $\mathsf{MT}^k(\mathsf{REGT}) \subseteq \mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$. From Lemma 10 and Theorem 20 it follows (by induction on k) that $\mathsf{TT}^k \subseteq \mathsf{dTT}_{\mathrm{rel}} \circ (\mathsf{TT}^\ell)^k$ and hence $\mathsf{TT}^k(\mathsf{REGT}) \subseteq (\mathsf{TT}^\ell)^k(\mathsf{REGT})$ by Lemma 15. Finally, by [31, Theorem 38 and Corollary 39], $(\mathsf{TT}^\ell)^k(\mathsf{REGT}) \subseteq \mathsf{MT}^m(\mathsf{REGT})$ for some $m \geq 1$. Hence $\mathsf{TT}^k(\mathsf{REGT}) \subseteq \mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$, by the above. □

As observed already after Corollary 51, the space part of Theorem 57 will be strengthened to $\mathsf{DSPACE}(n)$ in Theorem 67.

Theorem 58 *For every $k \geq 1$ and every $\tau \in \mathsf{TT}^k$ there is a nondeterministic Turing machine that computes, given an input t, any output $s \in \tau(t)$ in space $O(|t| + |s|)$ and in time polynomial in $|t| + |s|$. The same holds for $\tau \in \mathsf{MT}^k$.*

Proof For $\tau \in \mathsf{MT}^k$ this was already observed after Lemma 55. Now let $\tau \in \mathsf{TT}^k$ with input alphabet Σ. Let $\bar{\Sigma} = \{\bar{\sigma} \mid \sigma \in \Sigma\}$ with $\mathrm{rank}(\bar{\sigma}) = \mathrm{rank}(\sigma)$ be a set of new symbols, and let $\bar{t} \in T_{\bar{\Sigma}}$ be obtained from $t \in T_\Sigma$ by changing each label σ into $\bar{\sigma}$. Finally, let # be a new symbol of rank 2. It is easy to show that the tree language $L_\tau = \{\#(\bar{t}, s) \mid (t, s) \in \tau\}$ is in $\mathsf{TT}^k(\mathsf{REGT})$: the first transducer additionally copies the input to the output (with bars), and each other transducer copies the first subtree of the input to the output. By Theorem 57, there is a nondeterministic Turing machine N that recognizes L_τ in linear space and polynomial time. We construct the nondeterministic Turing machine N' that, on input t, guesses a possible output tree s, writing $\#(\bar{t}, s)$ on a worktape, uses N as a subroutine to verify that $(t, s) \in \tau$, and outputs s. Clearly, N' satisfies the requirements. □

Since IO (multi-return) macro tree translations, pebble tree translations, and high-level tree translations can be realized by compositions of TT's (see Sect. 9), Theorems 57 and 58 also hold for those translations.

By the proof of Corollary 51, we additionally obtain from Theorem 57 that $y\mathsf{TT}^k(\mathsf{REGT}) \subseteq \mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$ for every $k \geq 1$, and the same is true for $y\mathsf{MT}^k(\mathsf{REGT})$. The OI-*hierarchy* consists of the classes of string languages $\mathsf{OI}(k)$ generated by level-k grammars, with the outside-in (OI) derivation mode (see, e.g., [16,33]). It was shown in [37, Theorem 4.2.8] that $\mathsf{OI}(1)$ equals the class of indexed languages of [1], and hence that $\mathsf{OI}(1) \subseteq \mathsf{NSPACE}(n)$ by [1, Theorem 5.1]. Moreover, it was shown in [67, Proposition 2] that $\mathsf{OI}(1) \subseteq \mathsf{NPTIME}$. In [16, Corollary 7.26] it was proved that the languages in the OI-hierarchy are recursive. As observed in the last paragraph of [35], $\mathsf{OI}(k)$ is included in $y\mathsf{MT}^m(\mathsf{REGT})$ for some m.

Corollary 59 *For every $k \geq 1$, $\mathsf{OI}(k) \subseteq \mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$.*

It is shown in [67] that there is an NP-complete language in both $\mathsf{OI}(1)$ and $y\mathsf{fTT}_\downarrow(\mathsf{REGT})$, and it is shown in [74] that there even is one in the class ETOL, which is a subclass of both $\mathsf{OI}(1)$ and $y\mathsf{fTT}_\downarrow(\mathsf{REGT})$. Note that by [75, Theorem 14] the ETOL control hierarchy is included in the OI-hierarchy.

It will be shown in Corollary 68 that $\mathsf{OI}(k) \subseteq \mathsf{DSPACE}(n)$.

12 Translation complexity

In this section we study the time and space complexity of the membership problem of the tree translations in TT^k, i.e., for a fixed tree translation $\tau \subseteq T_\Sigma \times T_\Delta$ we want to know, for given trees $t \in T_\Sigma$ and $s \in T_\Delta$, how hard it is to decide whether or not $(t, s) \in \tau$. To formalize this, we denote by L_τ the string language $\{\#ts \mid (t,s) \in \tau\}$, where # is a new symbol. For simplicity, and without loss of generality, we assume that $\Sigma \cap \Delta = \varnothing$. Otherwise, we replace Σ by $\bar{\Sigma} = \{\bar{\sigma} \mid \sigma \in \Sigma\}$ as in the proof of Theorem 58. So, L_τ is a tree language over $\Sigma \cup \Delta \cup \{\#\}$, where # has rank 2. For a class $\mathcal{T}$ of tree translations and a complexity class $\mathcal{C}$, we will write $\mathcal{T} \subseteq \mathcal{C}$ to mean that $L_\tau \in \mathcal{C}$ for every $\tau \in \mathcal{T}$. As usual, we denote the class of languages that are accepted by a deterministic Turing machine in polynomial time by PTIME, and the class of languages that are log-space reducible to a context-free language by LOGCFL. Note that every regular tree language is a context-free language and hence is in LOGCFL. Note also that LOGCFL $\subseteq$ PTIME (see [68]) and LOGCFL $\subseteq$ DSPACE$(\log^2 n)$ (see [57,68] and [46, Theorem 12.7.4]).

If $\tau \in \mathsf{dTT}^k$ then, on input $\#(t,s)$, we can compute $\tau(t)$ according to Theorems 47 and 49 (rejecting the input when the computation takes more than time or space $c \cdot (|t| + |s|)$ for the given constant c) and then verify that $\tau(t) = s$, cf. the proof of Theorem 50. Thus, L_τ can be accepted by a RAM program in linear time and by a deterministic Turing machine in linear space. This means that $\mathsf{dTT}^k \subseteq$ PTIME and $\mathsf{dTT}^k \subseteq$ DSPACE(n). If $\tau \in \mathsf{TT}^k$, then, as mentioned in the proof of Theorem 58, the tree language L_τ is in the class of output languages TT^k(REGT), and hence in NSPACE$(n) \wedge$ NPTIME by Theorem 57. This means that $\mathsf{TT}^k \subseteq$ NPTIME and $\mathsf{TT}^k \subseteq$ NSPACE(n). Due to the presence of both the input tree and the output tree in L_τ, one would expect that better upper bounds can be shown. Indeed, we will prove that $\mathsf{TT}^k \subseteq$ DSPACE(n).

Our main aim in this section is to prove that $\mathsf{TT} \circ \mathsf{dTT} \subseteq$ LOGCFL. We follow the approach of [25], using multi-head automata.

A *multi-head tree-walking tree transducer* $M = (\Sigma, \Delta, Q, Q_0, R)$ (in short, mhTT) is defined in the same way as a TT, but has an arbitrary, fixed number of reading heads. Each of these heads can walk on the input tree, independent of the other heads. It can test the label and child number of the node that it is currently reading, and additionally apply a regular test to that node. Moreover, we assume that the heads are "sensing", which means that M can test which heads are currently scanning the same node. Thus, if M has ℓ heads, then its move rules are of the form

$$\langle q, \sigma_1, j_1, T_1, \ldots, \sigma_\ell, j_\ell, T_\ell, E\rangle \to \langle q', \alpha_1, \ldots, \alpha_\ell\rangle$$

where $E \subseteq [1,\ell] \times [1,\ell]$ is an equivalence relation. A configuration of M on input tree t is of the form $\langle q, u_1, \ldots, u_\ell\rangle$, to which the rule is applicable if M is in state q, each u_i satisfies the tests σ_i, j_i, and T_i, and $u_i = u_j$ for every $(i,j) \in E$. After application the new configuration is $\langle q', \alpha_1(u_1), \ldots, \alpha_\ell(u_\ell)\rangle$. The output rules are defined in a similar way. Initially all reading heads are at the root of the input tree. This is all similar to how multi-head automata on strings are defined.

We will use the mhTT M as an acceptor of its domain. We will say that it accepts dom(M) in polynomial time if there is a polynomial $p(n)$ such that for every $t \in \mathrm{dom}(M)$ there is a computation $\langle q_0, \mathrm{root}_t\rangle \Rightarrow^*_{M,t} s$ of length at most $p(|t|)$ for some $q_0 \in Q_0$ and $s \in T_\Delta$. Note that we consider nondeterministic mhTT's only.

Lemma 60 *For every multi-head* TT *M,* dom$(M) \in$ PTIME. *Moreover, if M accepts* dom(M) *in polynomial time, then* dom$(M) \in$ LOGCFL.

Proof After this paragraph we will show that the domain of a multi-head TT can be accepted by an alternating multi-head finite automaton (in short, AMFA), in a straightforward way. Moreover, we will show that if the mhTT accepts in polynomial time, then the corresponding AMFA accepts in polynomial tree-size. That proves the lemma because PTIME is the class of languages accepted by AMFA's (see [10,12]) and LOGCFL is the class of languages accepted by AMFA's in polynomial tree-size (see [68,71]).

It is well known that the domain of a classical local TT can be accepted by an alternating (one-head) tree-walking automaton, see, e.g., [70], [24, Section 4], and [63, Section 4], and the same is true for the multi-head case. Let $M = (\Sigma, \Delta, Q, Q_0, R)$ be an mhTT. The AMFA M' that accepts $\mathrm{dom}(M)$ simulates M on the input $t \in T_\Sigma$, without producing output. The reading heads of M are simulated by reading heads of M' in the obvious way. Every (initial) state q of M is simulated by the existential (initial) state q of M', and a move rule of M is simulated by a transition of M' in an obvious way. If M applies an output rule in state q, then M' first goes into a universal state q' and then branches in the same way as M, going into existential states. A regular test T of M is simulated by M' in a side branch, using an AMFA subroutine that accepts the context-free language $\mathrm{mark}(T)$, with additional reading heads. Note that since the heads are sensing, the node to be tested is "marked" by a reading head. Similarly, to move a head h from a parent u to its i-th child ui, M' first moves an auxiliary head h' nondeterministically to a position to the right of u, then checks in a side branch that the string between h and h' belongs to the context-free language T_Σ^{i-1}, and finally moves h to h'. In a similar way M' can move from ui to u, and can determine the child number of u.

If M accepts t in time m, then the size of the corresponding computation tree of M' is polynomial in m, because each computation step of M takes polynomial tree-size. Thus, if M accepts in polynomial time, then M' accepts in polynomial tree-size.

Note that if we assume that the simulation of a step of M takes constant tree-size, and we assume moreover that M only uses output rules (by eventually replacing the right-hand side ζ of each move rule by $\delta(\zeta)$, where δ has rank 1), then the output tree of M can be viewed both as the derivation tree of the computation of M and as the computation tree of M', roughly speaking. $\square$

Thus, to prove that $\mathsf{TT} \circ \mathsf{dTT} \subseteq \mathsf{LOGCFL}$ it suffices to show, for every $\tau = \tau_1 \circ \tau_2$ with $\tau_1 \in \mathsf{TT}$ and $\tau_2 \in \mathsf{dTT}$, that L_τ can be accepted by a multi-head TT M in polynomial time. Let M_1 and M_2 be TT's that realize τ_1 and τ_2. For an input tree t and an output tree s of τ, M will simulate M_1 on t, generating an intermediate tree r, and verify that M_2 translates r into s. Since M cannot store its output tree r, it must verify the translation of r into s on the fly, i.e., while generating r. That can be done because the context-free grammar $G_{M_2,r}$ is forward deterministic, and hence its reduced version has a unique fixed point: during the generation of the nodes v of r, M can guess the values of the nonterminals $\langle q, v\rangle$ of $G_{M_2,r}$ (which are subtrees of s) and check the fixed point equations for them. However, since $G_{M_2,r}$ need not be reduced, we have to be more careful.

Let $G = (N, \Delta, \{S\}, R)$ be a forward deterministic context-free grammar, and let # be a symbol not in $N \cup \Delta$ (which stands for 'undefined'). A string homomorphism $h : N \to \Delta^* \cup \{\#\}$ is a *fixed point* of G if (1) $h(S) \neq \#$, (2) $h(X)$ is a substring of $h(S)$ for every $X \in N$ such that $h(X) \neq \#$, and (3) $h(X) = h(\zeta)$ for every rule $X \to \zeta$ in R such that $h(X) \neq \#$, where h is extended to Δ by defining $h(a) = a$ for every $a \in \Delta$. In the special case that G is a regular tree grammar, a *tree fixed point* of G is a fixed point h of G such that $h(X) \in T_\Delta \cup \{\#\}$ for every $X \in N$ and $h(X)$ is a subtree of $h(S)$ for every $X \in N$ such that $h(X) \neq \#$.

Lemma 61 *Let $G = (N, \Delta, \{S\}, R)$ be a forward deterministic context-free grammar such that $L(G) \neq \varnothing$. For every $w \in \Delta^*$, $L(G) = \{w\}$ if and only if there is a fixed point h of G such that $h(S) = w$. If G is a regular tree grammar, then the same statement holds for $w \in T_\Delta$ and h a tree fixed point.*

Proof Let $L(G) = \{w\}$, and define $h_G(X)$ to be the unique string generated by X, if that exists and is a substring of w, and otherwise $h_G(X) = \#$. It is easy to see that $h = h_G$ satisfies the requirements.

Let h be a fixed point of G such that $h(S) = w$. Then $h(v) = w$ for every sentential form v of G. Since $L(G) \neq \varnothing$, this shows that $L(G) = \{w\}$. □

Theorem 62 $\mathsf{TT} \circ \mathsf{dTT} \subseteq \mathsf{LOGCFL}$.

Proof Let $M_1 = (\Sigma, \Omega, P, P_0, R_1)$ be a TT, and let $M_2 = (\Omega, \Delta, Q, q_0, R_2)$ be a dTT. We will denote τ_{M_1} and τ_{M_2} by τ_1 and τ_2, respectively. Since it is easy to prove (as in the proof of Corollary 38) that $\mathsf{TT} \circ \mathsf{dTT} = \mathsf{TT} * \mathsf{dTT}$, we may assume that (τ_1, τ_2) is linear-bounded. We may also assume, by Lemma 10 and Theorem 20, that M_2 is local. That does not change the linear-boundedness of the composition: if $(\tau_1, \tau_2' \circ \tau_2'')$ is linear-bounded and $\tau_2' \in \mathsf{TT}_{\mathrm{rel}}$, then $(\tau_1 \circ \tau_2', \tau_2'')$ is linear-bounded because τ_2' is size-preserving. Similarly, we may assume that $\mathrm{ran}(\tau_1) \subseteq \mathrm{dom}(\tau_2)$ by Corollaries 14 and 21. Finally we assume (as in the proofs of Lemmas 17 and 19) that M_1 keeps track in its finite state of the child number of the output node to be generated, through a mapping $\chi : P \to [0, mx_\Sigma]$.

On the basis of Lemma 60, we will describe a multi-head TT M that accepts L_τ in polynomial time, where $\tau = \tau_1 \circ \tau_2$. Initially M verifies by a regular test that the input tree is of the form $\#(t, s)$ with $t \in T_\Sigma$ and $s \in T_\Delta$. We will denote the root of $\#(t, s)$ by its label #. As mentioned before, on input $\#(t, s)$ the transducer M simulates M_1 on t generating an output tree r of M_1, which is in the domain of M_2 because $\mathrm{ran}(\tau_1) \subseteq \mathrm{dom}(\tau_2)$. It keeps the state p of M_1 in its finite state, uses one of its heads to point at a node of t (which it initially moves to the root of t), and instead of a regular test T applies the regular test $\{(\#(t, s), 1u) \mid (t, u) \in T\}$.[15] While generating r it guesses a tree fixed point $h : \mathrm{Con}(r) \to T_\Delta \cup \{\#\}$ of the regular tree grammar $G_{M_2,r}$ such that $h(\langle q_0, \mathrm{root}_r\rangle) = s$. If that fixed point can be guessed, then $\tau_2(r) = s$ by Lemma 61, and hence $(t, s) \in \tau$.

Initially, M guesses the values under h of the configurations in $\mathrm{Con}(r)$ that contain the root of r, in linear time. For each $q \in Q$ the value of $\langle q, \mathrm{root}_r\rangle$ is guessed by nondeterministically moving a reading head named (q, stay) to a node x of s, i.e., node $2x$ of $\#(t, s)$, meaning that $h(\langle q, \mathrm{root}_r\rangle) = s|_x$, or to node #, meaning that $h(\langle q, \mathrm{root}_r\rangle) = \#$ (i.e., that $h(\langle q, \mathrm{root}_r\rangle)$ is "undefined"). In particular, the head (q_0, stay) is moved to the root of s, thus guessing that $\tau_2(r) = s$.

Suppose that M is going to produce a node v of r with label ω, by simulating an output rule $\langle p, \sigma, j, T\rangle \to \omega(\langle p_1, \alpha_1\rangle, \ldots, \langle p_k, \alpha_k\rangle)$ of M_1. In such a situation, M has already guessed the values under h of the configurations in $\mathrm{Con}(r)$ that contain v, and also of those that contain the parent v' of v (if it has one). For each $q \in Q$ the value of $\langle q, v\rangle$ is stored using the reading head named (q, stay), as explained above for $v = \mathrm{root}_r$, and the value of $\langle q, v'\rangle$ is stored in a similar way using a reading head named (q, up). Now M guesses the values of the configurations that contain the children of v, in linear time. For every $q \in Q$ and $i \in [1, k]$, the value $h(\langle q, vi\rangle)$ is guessed by nondeterministically moving a reading head named (q, down_i) to some node of s or to #. Then M checks that these values satisfy requirement (3) of a fixed point of $G_{M_2,r}$ as follows, in linear time. If $\langle q, \omega, \chi(p)\rangle \to \langle q', \alpha\rangle$ is a move rule of M_2 such that head (q, stay) does not point to #, then M checks that the heads (q, stay) and (q', α) point to nodes with the same subtree. It can do this using two auxiliary heads that simultaneously perform a depth-first left-to-right traversal of those subtrees. Similarly, if

[15] Note that a node of t has the same label and child number in t and $\#(t, s)$, except when it has child number 1 in $\#(t, s)$ in which case it has child number 0 or 1 in t, depending on whether or not its parent in $\#(t, s)$ has label #.

$\langle q, \omega, \chi(p)\rangle \to \delta(\langle q_1, \alpha_1\rangle, \dots, \langle q_m, \alpha_m\rangle)$ is an output rule of M_2 such that head (q, stay) does not point to #, then M checks that it points to a node with label δ and that the subtree at the i-th child of that node equals the subtree at the head (q_i, α_i), for every $i \in [1, m]$. After checking the fixed point requirement (3), M outputs the node v and branches in the same way as M_1. In the i-th branch (apart from simulating M_1's rule in the obvious way) it moves head (q, up) to the position of head (q, stay) and then moves head (q, stay) to the position of head (q, down_i), for every $q \in Q$, in linear time.

This ends the description of M. It should be clear that τ_M is the set of all pairs $(\#(t, s), r)$ such that $(t, r) \in \tau_1$ (because M simulates M_1) and $\tau_2(r) = s$ (because M computes a tree fixed point h of $G_{M_2,r}$ such that $h(\langle q_0, \text{root}_r\rangle) = s$). Hence $\text{dom}(M) = \{\#(t, s) \mid \exists r : (t, r) \in \tau_1,\ \tau_2(r) = s\} = L_\tau$. It remains to show that M accepts L_τ in polynomial time.

There is a computation of M_1 of length at most $\#(P) \cdot |t| \cdot |r|$ that translates t into r, because if the number of move rules applied between two output rules is more than the number of configurations of M_1 on t, then there is a loop in the computation that can be removed. Since (τ_1, τ_2) is linear-bounded, we may assume that the size of r is at most linear in the size of s. Hence the length of that computation is polynomial in $|t|$ and $|s|$, and hence in $|\#(t, s)|$. Since M simulates M_1, and each simulated computation step takes linear time (as shown above), M accepts $\#(t, s)$ in polynomial time. □

From Theorem 62 and Lemma 28, which says that $\mathsf{mrMT_{IO}} \subseteq \mathsf{fTT_{\downarrow}} \circ \mathsf{dTT}$, we obtain the following corollary. Note that $\mathsf{TT} \circ \mathsf{dTT}$ is larger than $\mathsf{fTT_{\downarrow}} \circ \mathsf{dTT}$ in two respects. First, it contains non-finitary translations. Second, it contains total functions for which the height of the output tree can be double exponential in the height of the input tree, viz. τ^2_{exp} in the proof of Proposition 7, whereas that is at most exponential for total functions in $\mathsf{TT_{\downarrow}} \circ \mathsf{dTT}$ by Theorem 35, Lemma 24, and the paragraph after Corollary 25.

Corollary 63 $\mathsf{MT_{IO}} \subseteq \mathsf{mrMT_{IO}} \subseteq \mathsf{LOGCFL}$.

As another corollary we even obtain an upper bound on the complexity of the output languages of dTT that improves the one of Theorem 50. It was proved for attribute grammars in [25].

Corollary 64 $\mathsf{dTT(REGT)} \subseteq \mathsf{LOGCFL}$.

Proof Let L be a regular tree language over Ω and let $\tau_2 \subseteq T_\Omega \times T_\Delta$ be in dTT. Let $\Sigma = \{e\}$ with $\text{rank}(e) = 0$, and let $\tau_1 = \{(e, r) \mid r \in L\}$. The one-state TT^ℓ with rules $\langle p, e, 0\rangle \to \omega(\langle p, \text{stay}\rangle, \dots, \langle p, \text{stay}\rangle)$ for every $\omega \in \Omega$ realizes the translation $\{(e, r) \mid r \in T_\Omega\}$, and hence $\tau_1 \in \mathsf{TT}$ by Corollary 21. Let $\tau = \tau_1 \circ \tau_2$. Then $L_\tau = \{\#(e, s) \mid \exists r : r \in L,\ \tau_2(r) = s\} = \{\#(e, s) \mid s \in \tau_2(L)\}$. By Theorem 62 $L_\tau \in \mathsf{LOGCFL}$, and hence $\tau_2(L) \in \mathsf{LOGCFL}$ because $\tau_2(L)$ is log-space reducible to L_τ. □

Theorem 62 and Corollary 64 can be extended to deal with the yields of the output trees, as also proved in [25] for attribute grammars (generalizing the proof in [3] of $\mathsf{IO}(1) \subseteq \mathsf{LOGCFL}$). For a ranked alphabet Σ we define the mapping $y_\Sigma : T_\Sigma \to (\Sigma^{(0)})^*$ such that $y_\Sigma(t) = yt$, the yield of t. Let yield be the class of all such mappings y_Σ. In what follows we will identify each string w with the monadic tree $\text{mon}(w)$ as defined in the proof of Corollary 51. Hence, as mentioned in that proof, $\mathsf{yield} \subseteq \mathsf{dTT}^\ell$. This even holds if we assume the existence of special symbols in $\Sigma^{(0)}$ that are skipped when taking the yield of t (such as the symbols X_0 in the derivation trees of context-free grammars with ε-rules, cf. Sect. 2).

Corollary 65 $\mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{yield} \subseteq \mathsf{LOGCFL}$ *and* $y\mathsf{dTT(REGT)} \subseteq \mathsf{LOGCFL}$.

Proof It is straightforward to show that $\mathsf{yield} \subseteq \mathsf{dTT}_{\text{pru}} * \mathsf{yield}$. In fact, the deterministic pruning TT removes all nodes of rank 1 and, using regular look-ahead, all subtrees of which the yield is the empty string ε (due to the special symbols mentioned above). Consequently, as in the proof of Corollary 38, $\mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{yield} = \mathsf{TT} * (\mathsf{dTT} \circ \mathsf{yield})$. This allows us to repeat the proof of Theorem 62, this time with respect to the forward deterministic context-free grammar $G'_{M_2,r}$ that generates the yields of the trees generated by $G_{M_2,r}$: if $X \to \zeta$ is a rule of $G_{M_2,r}$, then $X \to y\zeta$ is a rule of $G'_{M_2,r}$. Thus, this time the mhTT M guesses a fixed point h of $G'_{M_2,r}$, rather than a tree fixed point. To do this it uses two heads $\langle q, \text{stay}, \text{left}\rangle$ and $\langle q, \text{stay}, \text{right}\rangle$ instead of the one head $\langle q, \text{stay}\rangle$, to guess the left- and right-end of the substring generated by the configuration $\langle q, v\rangle$, and similarly for up and down$_i$. It should be clear that the fixed point requirement (3) can easily be checked, showing that one such substring equals another one or is the concatenation of several other ones. □

The inclusion $\mathsf{TT} \circ \mathsf{dTT} \subseteq \mathsf{LOGCFL}$ of Theorem 62 has consequences for both space and time complexity. We first consider space complexity.

Since $\mathsf{LOGCFL} \subseteq \mathsf{DSPACE}(n)$, we obtain that $\mathsf{TT} \subseteq \mathsf{DSPACE}(n)$ from Theorem 62. This can easily be generalized to arbitrary compositions of TT's.

Theorem 66 *For every $k \geq 1$, $\mathsf{TT}^k \subseteq \mathsf{DSPACE}(n)$.*

Proof The proof is by induction on k, with an induction step similar to the one in the proof of Theorem 47.

Let $\tau = \tau_1 \circ \tau_2$ such that $\tau_1 \in \mathsf{TT}$ and $\tau_2 \in \mathsf{TT}^k$, $k \geq 1$. For a given input string $\#ts$ it has to be checked whether $(t, s) \in \tau$. By Corollary 38(1) we may assume that (τ_1, τ_2) is linear-bounded. Hence there is a constant $c \in \mathbb{N}$ such that for every $(t, s) \in \tau$ there is an intermediate tree r such that $|r| \leq c \cdot |s|$. To check whether $(t, s) \in \tau$ a deterministic Turing machine systematically enumerates all trees r such that $|r| \leq c \cdot |s|$ (cf. the proof of Theorem 50). For each such r it can check in linear space whether $(t, r) \in \tau_1$ by the case $k = 1$. Moreover, by induction it can check in linear space whether $(r, s) \in \tau_2$. Thus it uses space $O(|t| + |r|) + O(|r| + |s|) = O(|t| + |s|)$. □

This result allows us to prove one of our main results, viz. that the output languages of TT^k are in $\mathsf{DSPACE}(n)$, originally proved in [48]. It generalizes the main result of [4] from classical top-down tree transducers to tree-walking tree transducers and macro tree transducers.

Theorem 67 *For every $k \geq 1$,*

$$\mathsf{TT}^k(\mathsf{REGT}) \subseteq \mathsf{DSPACE}(n) \quad \textit{and} \quad \mathsf{MT}^k(\mathsf{REGT}) \subseteq \mathsf{DSPACE}(n).$$

Proof The proof is similar to the one of Theorem 50. Let $L \in \mathsf{REGT}$ and $\tau \in \mathsf{TT}^k$. By Corollary 38(1), $\tau = \tau_1 \circ \tau_2$ such that $\tau_1 \in \mathsf{TT}_{\text{pru}}$, $\tau_2 \in \mathsf{TT}^k$, and (τ_1, τ_2) is linear-bounded for some constant c. Let $L' = \tau_1(L)$, and note that $\tau(L) = \tau_2(L')$ and that $L' \in \mathsf{REGT}$ by Lemma 15. It is straightforward to show that for every $s \in \tau(L)$ there exists $t \in L'$ such that $(t, s) \in \tau_2$ and $|t| \leq c \cdot |s|$. To check whether a given tree s is in $\tau(L)$, a deterministic Turing machine enumerates all input trees t (of τ_2) such that $|t| \leq c \cdot |s|$. For each such t it first checks that $t \in L'$ in space $O(|t|) = O(|s|)$. Then it uses the algorithm of Theorem 66 to check that $(t, s) \in \tau_2$ in space $O(|t| + |s|) = O(|s|)$.

The inclusion for MT^k is now immediate from Lemma 27. □

As before, Theorems 66 and 67 also hold for IO (multi-return) macro tree translations, pebble tree translations, and high-level tree translations, which can be realized by compositions of TT's (see Sect. 9).

By the proof of Corollary 51, Theorem 67 implies that

$$y\mathsf{TT}^k(\mathsf{REGT}) \subseteq \mathsf{DSPACE}(n) \quad \text{and} \quad y\mathsf{MT}^k(\mathsf{REGT}) \subseteq \mathsf{DSPACE}(n)$$

for every $k \geq 1$. Hence the OI-hierarchy is also contained in $\mathsf{DSPACE}(n)$, cf. Corollaries 52 and 59.

Corollary 68 *For every $k \geq 1$, $\mathsf{OI}(k) \subseteq \mathsf{DSPACE}(n)$.*

Next we consider time complexity. Since $\mathsf{LOGCFL} \subseteq \mathsf{PTIME}$, it follows from Theorem 62 that $\mathsf{TT} \circ \mathsf{dTT} \subseteq \mathsf{PTIME}$. This result can be generalized as follows.

One way to increase the power of the TT is to give it a more powerful feature of look-around. For a class $\mathcal{L}$ of tree or string languages, we define the TT *with $\mathcal{L}$ look-around* by allowing the TT to use node tests T such that $\mathrm{mark}(T) \in \mathcal{L}$. Similarly we obtain the mhTT with $\mathcal{L}$ look-around. We now consider in particular the case where $\mathcal{L} = \mathsf{PTIME}$. Obviously, (the proof of) the first sentence of Lemma 60 is still valid for a multi-head TT M with PTIME look-around. Thus, the domain of an mhTT with PTIME look-around is in PTIME, and hence, in particular, the domain of a TT with PTIME look-around is in PTIME. This implies that Lemma 19, and hence Theorem 20, also holds if the first transducer has PTIME look-around. From the proof of Theorem 62 it now easily follows that $\mathsf{TT}^{\mathsf{P}} \circ \mathsf{dTT} \subseteq \mathsf{PTIME}$, where the feature of PTIME look-around is indicated by a superscript P. This, in its turn, implies the following variant of Corollary 63 for (multi-return) IO macro tree transducers with PTIME look-around (appropriately defined): $\mathsf{MT}^{\mathsf{P}}_{\mathrm{IO}} \subseteq \mathsf{mrMT}^{\mathsf{P}}_{\mathrm{IO}} \subseteq \mathsf{PTIME}$. Examples of tree languages in PTIME that can be used as look-around are those in $\mathsf{dTT}(\mathsf{REGT})$, by Corollary 64, and the tree languages defined by bottom-up tree automata with equality and disequality constraints ([8]), which can obviously be accepted by a multi-head TT.

In the remainder of this section we show that there are translations in $\mathsf{dTT} \circ \mathsf{TT}$, even in $\mathsf{dTT}_{\downarrow} \circ \mathsf{TT}$, for which the membership problem is NP-complete. We will use a reduction of SAT, the satisfiability problem of boolean formulas (see, e.g., [42]), to such a membership problem.

Let $\Delta = \{\vee, \wedge, \neg, \mathsf{v}, \mathsf{e}\}$ with $\Delta^{(2)} = \{\vee, \wedge\}$, $\Delta^{(1)} = \{\neg, \mathsf{v}\}$, and $\Delta^{(0)} = \{\mathsf{e}\}$. Let $\mathcal{B}$ be the set of all trees over Δ generated by the regular tree grammar with nonterminals F and V, initial nonterminal F, and rules $F \to \vee(F, F)$, $F \to \wedge(F, F)$, $F \to \neg(F)$, $F \to V$, $V \to \mathsf{v}(V)$, and $V \to \mathsf{v}(\mathsf{e})$. Thus, $\mathcal{B}$ is the set of all boolean formulas that use boolean variables of the form $\mathsf{v}^\ell\mathsf{e}$ for $\ell \geq 1$. For a boolean formula φ we define $\nu(\varphi)$ to be the nesting-depth of its boolean operators, i.e., $\nu(\varphi) = 0$ if φ is a variable, $\nu(\vee(\varphi_1, \varphi_2)) = \nu(\wedge(\varphi_1, \varphi_2)) = \max\{\nu(\varphi_1), \nu(\varphi_2)\} + 1$, and $\nu(\neg(\varphi)) = \nu(\varphi) + 1$. For every $m \geq 0$ and $n \geq 1$, let $\mathcal{B}(m, n)$ be the set of all formulas $\varphi \in \mathcal{B}$ such that $\nu(\varphi) \leq m$, and $\ell \in [1, n]$ for every $\mathsf{v}^\ell\mathsf{e}$ that occurs in φ. Thus, the formulas in $\mathcal{B}(m, n)$ have nesting-depth at most m and use at most the variables $\mathsf{ve}, \mathsf{vve}, \ldots, \mathsf{v}^n\mathsf{e}$.

The proof of the next lemma is essentially a variant of the one of [74, Theorem 3.1]. Let $\Sigma = \{c, d, 0, 1, a\}$ with $\Sigma^{(1)} = \{c, d, 0, 1\}$ and $\Sigma^{(0)} = \{a\}$.

Lemma 69 *There is a translation $\tau \in \mathsf{fTT}^{\ell}_{\downarrow}$ such that, for every $m \geq 0$ and every string $w \in \{0, 1\}^*$ of length $n \geq 1$, the set $\tau(d^m cwa)$ consists of all boolean formulas $\varphi \in \mathcal{B}(m, n)$ such that φ is true when the value of $\mathsf{v}^\ell\mathsf{e}$ is the ℓ-th symbol of w for every $\ell \in [1, n]$.*

Proof We construct the top-down local TT $M = (\Sigma, \Delta, \{q_0, q_1\}, \{q_1\}, R)$. Note that the initial state is q_1. The boolean operations $i \vee j$, $i \wedge j$, and $\neg i$ on $\{0, 1\}$ are defined as usual, where 0 stands for 'false' and 1 for 'true'. Since the child numbers of the nodes of the input tree will be irrelevant, we omit them from the left-hand sides of the rules of M. The only instruction used in the right-hand sides of the rules is $\alpha = \mathrm{down}_1$. The rules are the following, for every $i, j \in \{0,1\}$.

$$\begin{array}{llll}
\langle q_{i\vee j}, d\rangle \to \vee(\langle q_i,\alpha\rangle, \langle q_j,\alpha\rangle) & & \langle q_i, c\rangle \to \mathsf{v}(\langle q_i,\alpha\rangle) \\
\langle q_{i\wedge j}, d\rangle \to \wedge(\langle q_i,\alpha\rangle, \langle q_j,\alpha\rangle) & & \langle q_i, j\rangle \to \mathsf{v}(\langle q_i,\alpha\rangle) \\
\langle q_{\neg i}, d\rangle \to \neg(\langle q_i,\alpha\rangle) & & \langle q_i, i\rangle \to \mathsf{e} \\
\langle q_i, d\rangle \to \langle q_i,\alpha\rangle & &
\end{array}$$

Let u be the node of the input tree $t = d^m cwa$ with label c. After consuming d^m, the TT M has nondeterministically generated any output form that is a boolean formula φ of nesting-depth at most m and with the two configurations $\langle q_i, u\rangle$ as variables, such that φ is true when the value of $\langle q_i, u\rangle$ is i. For instance, in the first step of that computation M consumes d and changes the initial output form $\langle q_1, \mathrm{root}_t\rangle$ into one of the output forms $\vee(\langle q_1,x\rangle, \langle q_0,x\rangle)$, $\vee(\langle q_0,x\rangle, \langle q_1,x\rangle)$, $\vee(\langle q_1,x\rangle, \langle q_1,x\rangle)$, $\wedge(\langle q_1,x\rangle, \langle q_1,x\rangle)$, $\neg(\langle q_0,x\rangle)$, or $\langle q_1,x\rangle$, where x is the child of root_t. After that, each $\langle q_i, u\rangle$ generates any variable $\mathsf{v}^\ell\mathsf{e}$ such that the ℓ-th symbol of w is i. Note that since i and j are not necessarily distinct, M has in particular the rule $\langle q_i, i\rangle \to \mathsf{v}(\langle q_i,\alpha\rangle)$ for every $i \in \{0,1\}$. Thus, q_i can nondeterministically choose any occurrence of i in w to output e and end the computation. □

Applying the translation τ of Lemma 69 to the regular tree language L consisting of all trees $d^m cwa$ such that $m \geq 0$ and w is a nonempty string over $\{0,1\}$, produces the set $\tau(L)$ of all satisfiable formulas in $\mathcal{B}$. Thus, since the membership problem for that set is NP-complete, we obtain the following corollary that was proved in [67], as already mentioned after Corollary 59. Note that it is easy to prove that $\mathsf{fTT}_\downarrow(\mathsf{REGT}) \subseteq y\mathsf{fTT}_\downarrow(\mathsf{REGT})$: just change every output rule $\langle q,\sigma,j,T\rangle \to \delta(\langle q_1,\alpha_1\rangle,\dots,\langle q_k,\alpha_k\rangle)$ into the (general) rule $\langle q,\sigma,j,T\rangle \to \omega_{k+1}(\delta, \langle q_1,\alpha_1\rangle,\dots,\langle q_k,\alpha_k\rangle)$ where ω_{k+1} has rank $k+1$ (and δ now has rank 0).

Corollary 70 *There is an* NP*-complete language in* $\mathsf{fTT}_\downarrow(\mathsf{REGT})$*, and hence there is one in* $y\mathsf{fTT}_\downarrow(\mathsf{REGT})$.

We now prove the existence of a translation in $\mathsf{dTT}_\downarrow \circ \mathsf{TT}$ for which the membership problem is NP-complete. Recall that, for a tree translation τ, we denote by L_τ the tree language $\{\#(t,s) \mid (t,s) \in \tau\}$.

Theorem 71 *There is a translation* $\tau \in \mathsf{dTT}^\ell_\downarrow \circ \mathsf{dTT}^\ell \circ \mathsf{TT}^\ell_{\mathrm{pru}} \subseteq \mathsf{dTT}_\downarrow \circ \mathsf{fTT}$ *such that* L_τ *is* NP*-complete.*

Proof The inclusion $\mathsf{dTT}^\ell \circ \mathsf{TT}^\ell_{\mathrm{pru}} \subseteq \mathsf{fTT}$ is immediate from Lemma 19. We first describe a translation $\tau \in \mathsf{dTT}^\ell_\downarrow \circ \mathsf{fTT}^\ell$ such that L_τ is NP-complete. Let $\Gamma = \{a,b,c,d,e\}$ with $\Gamma^{(1)} = \{a,b,c,d\}$ and $\Gamma^{(0)} = \{e\}$. The translation $\tau \subseteq T_\Gamma \times T_\Delta$ transforms each tree $t = ab^n cd^m e$ into all satisfiable boolean formulas in $\mathcal{B}(m,n)$. This will be realized by the composition of two TT's M_1 and M_2 such that the deterministic TT M_1 transforms t into a tree s of which the path language[16] consists of all strings $awcd^m e$ with $w \in \{0,1\}^*$ of length n, and M_2 nondeterministically chooses a leaf of s and then walks back to the root of s while simulating the transducer M of (the proof of) Lemma 69 on the tree $d^m cwa \in T_\Sigma$. Thus, M_1 provides all possible valuations of the variables $\mathsf{ve}, \mathsf{vve}, \dots, \mathsf{v}^n\mathsf{e}$ and M_2 chooses one such valuation and produces all formulas in $\mathcal{B}(m,n)$ that are true for that valuation.

[16] The path language of a tree $s \in T_\Omega$ consists of all strings in Ω^* that are obtained by walking along a path from the root of s to one of its leaves, writing down the labels of the nodes of that path from left to right.

Let $\Omega = \{a, 0, 1, c, d, e\}$ with $\Omega^{(2)} = \{a, 0, 1\}$, $\Omega^{(1)} = \{c, d\}$, and $\Omega^{(0)} = \{e\}$. We define $\tau = \tau_{M_1} \circ \tau_{M_2} \subseteq T_\Gamma \times T_\Delta$ where M_1 and M_2 are the following TT's. The deterministic $\mathrm{TT}^\ell_\downarrow$ $M_1 = (\Gamma, \Omega, \{q, q_0, q_1, p\}, \{q\}, R_1)$ has the following rules, for $i \in \{0, 1\}$ and $\alpha = \mathrm{down}_1$.

$$\begin{array}{llll} \langle q, a, 0\rangle & \to a(\langle q_0, \alpha\rangle, \langle q_1, \alpha\rangle) & \langle p, d, 1\rangle & \to d(\langle p, \alpha\rangle) \\ \langle q_i, b, 1\rangle & \to i(\langle q_0, \alpha\rangle, \langle q_1, \alpha\rangle) & \langle p, e, 1\rangle & \to e \\ \langle q_i, c, 1\rangle & \to c(\langle p, \alpha\rangle) & & \end{array}$$

It should be clear that for an input tree $ab^n cd^m e$, with $m \geq 0$ and $n \geq 1$, the path language of the tree $\tau_{M_1}(ab^n cd^m e)$ consists of all strings $awcd^m e$ with $w \in \{0, 1\}^*$ of length n.

The TT^ℓ $M_2 = (\Omega, \Delta, Q, Q_0, R_2)$ has states $Q = \{q_2, q_0, q_1\}$ and $Q_0 = \{q_2\}$. On an input tree $\tau_{M_1}(ab^n cd^m e)$, it walks nondeterministically in state q_2 from the root to some leaf (without producing output), moves to the parent of that leaf, and then simulates the transducer M of Lemma 69 on the tree $d^m cwa \in T_\Sigma$ while walking back to the root. It starts that simulation in the state q_1 of M, and then uses the rules of M with $\alpha = \mathrm{up}$.

With this definition of M_1 and M_2, it follows from Lemma 69 that the set $\tau(ab^n cd^m e)$ consists of all boolean formulas $\varphi \in \mathcal{B}(m, n)$ such that φ is satisfiable. Thus, for a formula $\varphi \in \mathcal{B}(m, n)$, φ is satisfiable if and only if $\#(ab^n cd^m e, \varphi)$ is in L_τ. This shows that satisfiability is reducible to membership in L_τ, because the nesting-depth m of φ and the number n of variables it uses, can easily be computed from any $\varphi \in \mathcal{B}$ in polynomial time.

We finally show that $\tau_{M_2} \in \mathrm{dTT}^\ell \circ \mathrm{TT}^\ell_{\mathrm{pru}}$, by a standard technique (see, e.g., [34, Section 6.1]). In fact, we will show that $\tau_{M_2} \in \mathrm{dTT}^\ell \circ \mathrm{SET}$, cf. the proof of Lemma 27. Let $+$ be a new symbol of rank 2, and θ a new symbol of rank 0. Let M_2' be the deterministic TT^ℓ with output alphabet $\Delta \cup \{+, \theta\}$ that is obtained from M_2 as follows. For every triple $\langle q, \omega, j\rangle$ such that $q \in Q$, $\omega \in \Omega$, and $j \in [0, mx_\Omega]$, if $\langle q, \omega, j\rangle \to \zeta_1, \ldots, \langle q, \omega, j\rangle \to \zeta_r$ are all the rules of M_2 with left-hand side $\langle q, \omega, j\rangle$, then M_2' has the rule $\langle q, \omega, j\rangle \to +(\zeta_1, +(\zeta_2, \zeta_3))$ if $r = 3$, the rule $\langle q, \omega, j\rangle \to +(\zeta_1, \zeta_2)$ if $r = 2$, the rule $\langle q, \omega, j\rangle \to \zeta_1$ if $r = 1$, and the rule $\langle q, \omega, j\rangle \to \theta$ if $r = 0$. Let M_3 be the pruning TT with one state p and rules $\langle p, \delta, j\rangle \to \delta(\langle p, \mathrm{down}_1\rangle, \ldots, \langle p, \mathrm{down}_k\rangle)$ for every $\delta \in \Delta^{(k)}$, plus the rules $\langle p, +, j\rangle \to \langle p, \mathrm{down}_1\rangle$ and $\langle p, +, j\rangle \to \langle p, \mathrm{down}_2\rangle$ (for every child number j). Since M_2 first moves from the root to a leaf, and then moves back to the root, it does not have infinite computations. From that it should be clear that $\tau_{M_2} = \tau_{M_2'} \circ \tau_{M_3}$. □

Corollary 72 *There is a translation $\tau \in \mathrm{MT}$ such that L_τ is* NP-*complete.*

Proof By Lemma 24, $\mathrm{dTT}^\ell_\downarrow \circ \mathrm{dTT}^\ell \subseteq \mathrm{dMT}$. Moreover, by [34, Theorem 7.6(3)], $\mathrm{dMT} \circ \mathrm{TT}^\ell_{\mathrm{pru}} \subseteq \mathrm{MT}$. Hence the translation τ of Theorem 71 is in MT. □

Since $\mathrm{MT} \subseteq \mathrm{MT}^2_{\mathrm{IO}}$ by [34, Theorem 6.10], this also shows that there is a translation $\tau \in \mathrm{MT}^2_{\mathrm{IO}}$ such that L_τ is NP-complete, cf. Corollary 63.

13 Forest transducers

Whereas we have considered ranked trees until now, i.e., trees over a ranked alphabet, XML documents naturally correspond to unranked trees or *forests*, over an ordinary unranked alphabet. For that reason we now consider transducers that transform forests into forests. Rather than generalizing the TT to a "forest-walking forest transducer", we take the equivalent, natural approach of letting the TT transform representations of forests by (ranked) trees, cf. [63] and [28, Section 11].

For an ordinary (unranked) alphabet Σ the set F_Σ of forests over Σ is the language generated by the context-free grammar with nonterminals F and T, initial nonterminal F, set of terminals $\Sigma \cup \{[\,,]\}$, where $\{[\,,]\}$ is the set consisting of the left and right square bracket, and rules $F \to \varepsilon$, $F \to TF$, and $T \to \sigma[F]$ for every $\sigma \in \Sigma$. Thus, intuitively, a forest is a sequence of unranked trees, and an unranked tree is of the form $\sigma[t_1 \cdots t_n]$ where each t_i is an unranked tree. Note that every forest $f \in F_\Sigma$ can be uniquely written as $f = \sigma[f_1]f_2$ with $\sigma \in \Sigma$ and $f_1, f_2 \in F_\Sigma$.

As usual, forests can be encoded as binary trees. With Σ we associate the ranked alphabet $\Sigma_e = \Sigma \cup \{e\}$ where e has rank 0 and every $\sigma \in \Sigma$ has rank 2. The mapping $\mathrm{enc}_\Sigma : F_\Sigma \to T_{\Sigma_e}$ is defined as follows. The encoding of the empty forest is $\mathrm{enc}_\Sigma(\varepsilon) = e$, and recursively, the encoding of a forest $f = \sigma[f_1]f_2$ is $\mathrm{enc}_\Sigma(f) = \sigma(\mathrm{enc}_\Sigma(f_1), \mathrm{enc}_\Sigma(f_2))$. The mapping enc_Σ is a bijection, and the inverse decoding is denoted by dec_Σ. Let enc and dec denote the classes of encodings enc_Σ and decodings dec_Σ, respectively, for all alphabets Σ. We define $\mathsf{FT} = \mathsf{enc} \circ \mathsf{TT} \circ \mathsf{dec}$ to be the class of TT *forest translations*. Thus, a TT forest translation is of the form $\tau = \mathrm{enc}_\Sigma \circ \tau_M \circ \mathrm{dec}_\Delta$ where Σ and Δ are alphabets and M is a TT with input alphabet Σ_e and output alphabet Δ_e, which in this context can be called a TT forest transducer. We first restrict attention to deterministic TT forest transducers, i.e., to the class $\mathsf{dFT} = \mathsf{enc} \circ \mathsf{dTT} \circ \mathsf{dec}$.

The next simple lemma shows that the encodings of compositions are the compositions of encodings (of deterministic TT's).

Lemma 73 *For every $k \geq 1$, $\mathsf{dFT}^k = \mathsf{enc} \circ \mathsf{dTT}^k \circ \mathsf{dec}$.*

Proof The inclusion $\mathsf{dFT}^k \subseteq \mathsf{enc} \circ \mathsf{dTT}^k \circ \mathsf{dec}$ is obvious, because $\mathrm{dec}_\Delta \circ \mathrm{enc}_\Delta$ is the identity on T_{Δ_e} for every (unranked) alphabet Δ. To show that $\mathsf{enc} \circ \mathsf{dTT}^k \circ \mathsf{dec} \subseteq \mathsf{dFT}^k$, it suffices to prove that $\mathsf{dTT} \circ \mathsf{dTT} \subseteq \mathsf{dTT} \circ \mathsf{dec} \circ \mathsf{enc} \circ \mathsf{dTT}$. Let Γ be the (ranked) output alphabet of a first transducer, which is also the input alphabet of the second, and let id_Γ be the identity on T_Γ. By the composition results of Theorems 18 and 23, it now suffices to show that $\mathrm{id}_\Gamma \in \mathsf{dTT}^\ell_\downarrow \circ \mathsf{dec} \circ \mathsf{enc} \circ \mathsf{dTT}^\ell_{\mathrm{su}}$. We do this by encoding the trees over Γ as binary trees, similar to the transformation of the derivation trees of a context-free grammar into those of its Chomsky Normal Form. Let ω be a new symbol, and let Δ be the unranked alphabet $\Gamma \cup \{\omega\}$. We encode the trees over Γ as trees over the ranked alphabet Δ_e, which are the usual encodings of forests over Δ. The encoding $h : T_\Gamma \to T_{\Delta_e}$ is defined as follows: for every $\gamma \in \Gamma^{(k)}$, if $h(t_i) = t'_i$ for every $i \in [1,k]$, then $h(\gamma(t_1, t_2, \ldots, t_k)) = \gamma(e, \omega(t'_1, \omega(t'_2, \ldots \omega(t'_k, e) \cdots)))$. It should be clear that h is an injection. It should also be clear that $h \in \mathsf{dTT}^\ell_\downarrow$ (in fact, h is a tree homomorphism, which can be realized by a classical top-down tree transducer). Finally, it is also easy to construct a local top-down single-use TT M such that $\tau_M(h(t)) = t$ for every $t \in T_\Gamma$. It has the set of states $Q = \{q_i \mid i \in [0, mx_\Gamma]\}$ with initial state q_0, and the following rules (where $\gamma \in \Gamma^{(k)}$, $j \in [0,2]$, and $q_i \in Q$, $i \neq 1$):

$$\begin{aligned}
\langle q_0, \gamma, j\rangle &\to \gamma(\langle q_1, \mathrm{down}_2\rangle, \ldots, \langle q_k, \mathrm{down}_2\rangle)\\
\langle q_1, \omega, 2\rangle &\to \langle q_0, \mathrm{down}_1\rangle\\
\langle q_i, \omega, 2\rangle &\to \langle q_{i-1}, \mathrm{down}_2\rangle
\end{aligned}$$

Note that γ and ω have rank 2 in Δ_e. □

We now wish to show that our main results also hold for deterministic TT forest translations. Let us first consider the complexity results of Sect. 10. It is easy to see that for every alphabet Σ, the mappings enc_Σ and dec_Σ can be computed by a deterministic Turing machine in linear time and space, simulating a one-way pushdown transducer.[17] This implies, by Lemma 73, that Theorems 47 and 49 also hold for dFT^k. We define a set of forests $L \subseteq F_\Sigma$ to be a *regular forest language* if $\mathrm{enc}_\Sigma(L) \in \mathsf{REGT}$, and we denote the class of regular forest languages by REGF. Then, for every $k \geq 1$, the class $\mathsf{dFT}^k(\mathsf{REGF})$ of output forest languages is included in the class $\mathsf{dec}(\mathsf{dTT}^k(\mathsf{REGT}))$ by Lemma 73. Let $L \in \mathsf{REGT}$ and $\tau \in \mathsf{dTT}^k$ with output alphabet Δ_e. Then a forest f over Δ is in $\mathrm{dec}_\Delta(\tau(L))$ if and only if $\mathrm{enc}_\Delta(f)$ is in $\tau(L)$. That implies that Theorem 50 also holds for dFT^k, in the sense that $\mathsf{dFT}^k(\mathsf{REGF}) \subseteq \mathsf{DSPACE}(n)$.

Next we consider the results of Sect. 9, and extend the class LSIF in the obvious way to forest translations. Since it is easy to show that for every forest $f \in F_\Sigma$, we have $|\mathrm{enc}_\Sigma(f)| = \frac{2}{3}|f| + 1$ (see footnote 17), a translation $\tau' = \mathrm{enc}_\Sigma \circ \tau \circ \mathrm{dec}_\Delta$ is of linear size increase if and only if τ is of linear size increase. Thus, since $\mathsf{dFT}^k = \mathsf{enc} \circ \mathsf{dTT}^k \circ \mathsf{dec}$ by Lemma 73, it is decidable for a given composition of deterministic TT forest transducers whether or not it is of linear size increase. And if so, an equivalent deterministic TT forest transducer can be constructed: $\mathsf{dFT}^k \cap \mathsf{LSIF} = \mathsf{enc} \circ (\mathsf{dTT}^k \cap \mathsf{LSIF}) \circ \mathsf{dec} = \mathsf{enc} \circ \mathsf{dTT}_{\mathrm{su}} \circ \mathsf{dec} \subseteq \mathsf{dFT}$. Intuitively, $\mathsf{enc} \circ \mathsf{dTT}_{\mathrm{su}} \circ \mathsf{dec}$ is the class of translations realized by "single-use forest-walking forest transducers". Since $\mathsf{dTT}_{\mathrm{su}} = \mathsf{dMSOT}$ by Proposition 29, it is also the class $\mathsf{enc} \circ \mathsf{dMSOT} \circ \mathsf{dec}$. Viewing forests as graphs, and hence as logical structures, in the obvious way (just as trees), every encoding enc_Σ and every decoding dec_Σ is a deterministic (i.e., parameterless) MSO translation, as defined in [14, Chapter 7]. Hence, by the closure of MSO translations under composition [14, Theorem 7.14], $\mathsf{enc} \circ \mathsf{dMSOT} \circ \mathsf{dec}$ equals the (natural) class of deterministic MSO translations from forests to forests.

As observed in [65] for macro tree transducers, whereas the encoding of forests as binary trees is quite natural for the input forest of a TT, for the output forest it is less natural, because it forces the TT to generate the output forest f in its unique form $f = \sigma[f_1]f_2$. It is more natural to additionally allow the TT to generate f as a concatenation $f_1 f_2$ of two forests f_1 and f_2. To formalize this, as in [26, Section 7] and in accordance with [65], we associate with an alphabet Δ the ranked alphabet $\Delta_@ = \Delta \cup \{@, e\}$ where @ has rank 2, e has rank 0, and every $\delta \in \Delta$ has rank 1. The mapping $\mathrm{flat}_\Delta : T_{\Delta_@} \to F_\Delta$ is a "flattening" defined as follows (for $t_1, t_2 \in T_{\Delta_@}$ and $\delta \in \Delta$): $\mathrm{flat}_\Delta(e) = \varepsilon$, $\mathrm{flat}_\Delta(@(t_1, t_2)) = \mathrm{flat}_\Delta(t_1)\mathrm{flat}_\Delta(t_2)$, the concatenation of $\mathrm{flat}_\Delta(t_1)$ and $\mathrm{flat}_\Delta(t_2)$, and $\mathrm{flat}_\Delta(\delta(t_1)) = \delta[\mathrm{flat}_\Delta(t_1)]$. The mapping flat_Δ is surjective but, in general, not injective. Let flat denote the class of flattenings flat_Δ, for all alphabets Δ. We define $\mathsf{FT}_@ = \mathsf{enc} \circ \mathsf{TT} \circ \mathsf{flat}$ to be the class of *extended* TT *forest translations*. An extended TT forest tree transducer is a TT with input alphabet Σ_e and output alphabet $\Delta_@$. Again, we first restrict attention to deterministic transducers, i.e., to the class $\mathsf{dFT}_@ = \mathsf{enc} \circ \mathsf{dTT} \circ \mathsf{flat}$.

Let us show that there is an extended TT forest translation in $\mathsf{dFT}_@$ that is not in dFT. That was shown for macro tree transducers in [65, Theorem 8] by a similar argument. Let $\Gamma = \{\sigma\}$ and $\Omega = \{\delta\}$ be alphabets, and let us identify the forest $\sigma[\,]$ with the symbol σ, and similarly $\delta[\,]$ with δ. Then $\Gamma^* \subseteq F_\Gamma$ and $\Omega^* \subseteq F_\Omega$. There is a deterministic extended TT forest transducer that translates the string σ^n into the string $\delta^{2^{n+1}}$ for every $n \in \mathbb{N}$. In fact, let M be the dTT (with general rules) that is obtained from the dTT M_{exp} of Example 5 by changing its output alphabet into $\Omega_@ = \{@, \delta, e\}$, and changing σ into @ and e into $\delta(e)$ in the right-hand sides of its rules. Note that the input alphabet Σ of M_{exp} and M equals Γ_e. The input tree $t_n = \mathrm{enc}_\Gamma(\sigma^n) = \sigma(e, \sigma(e, \ldots \sigma(e, e) \cdots))$ is translated by M_{exp} into the full binary tree s_n over Σ with 2^{n+1} leaves. Clearly, M translates t_n into the tree s'_n that is obtained from s_n by changing every σ into @ and every e into $\delta(e)$. Thus, $\mathrm{flat}_\Omega(s'_n) = \delta^{2^{n+1}}$.

[17] In fact, enc_Σ can even be computed without pushdown: for every forest $f \in F_\Sigma$, $\mathrm{enc}_\Sigma(f)$ can be obtained from f by removing all left-brackets, changing each right-bracket into e, and adding one e at the end.

This forest translation is not in dFT, because $|\mathrm{enc}_\Gamma(\sigma^n)| = |t_n| = 2n+1$ but the height of $s''_n = \mathrm{enc}_\Omega(\delta^{2^{n+1}})$ is 2^{n+1}, and so, by Lemma 6, there is no dTT that translates t_n into s''_n.

We will show that $\mathsf{dFT} \subseteq \mathsf{dFT}_@ \subseteq \mathsf{dFT}^2$. A similar result was proved for macro tree transducers in [65, Theorem 8 and Corollary 12]. To compare dFT and $\mathsf{dFT}_@$, and their compositions, we establish two relationships between dec and flat in the next lemma.

Lemma 74 $\mathsf{dec} \subseteq \mathsf{dTT}^\ell_\downarrow \circ \mathsf{flat}$ *and* $\mathsf{flat} \subseteq \mathsf{dTT}^\ell_{\mathrm{su}} \circ \mathsf{dec}$.

Proof To show the first inclusion, let Δ be an alphabet and define the mapping $h\colon T_{\Delta_e} \to T_{\Delta_@}$ such that $h(e) = e$ and if $h(t_1) = t'_1$ and $h(t_2) = t'_2$, then $h(\delta(t_1, t_2)) = @(\delta(t'_1), t'_2)$. It is straightforward to prove that $h \circ \mathrm{flat}_\Delta = \mathrm{dec}_\Delta$. It is also easy to show that $h \in \mathsf{dTT}^\ell_\downarrow$ (as in the proof of Lemma 73, h is a tree homomorphism, which can be realized by a classical top-down tree transducer). Hence $\mathrm{dec}_\Delta \in \mathsf{dTT}^\ell_\downarrow \circ \mathsf{flat}$.

For the second inclusion, let Δ be an alphabet. The mapping $\mathrm{flat}_\Delta \circ \mathrm{enc}_\Delta$ can be realized by a local single-use dTT $M = (\Delta_@, \Delta_e, Q, q_0, R)$ that performs a depth-first left-to-right tree traversal in a special way. Rather than performing this traversal in one branch, it does so in all its branches together, each branch performing a separate piece of the traversal. When M arrives from above at a node u with label $\delta \in \Delta$, it outputs δ and splits into two branches. The first branch traverses the subtree at u, and the second branch continues the traversal after that subtree. Each branch outputs e when arriving from below at a Δ-labeled node (or at the root, at the end of the traversal). Formally, M has the state set $Q = \{d, u_1, u_2\}$ with initial state $q_0 = d$, cf. Examples 4 and 5. It has the following (general) rules, where $j' \in [0, mx_\Sigma]$, $j \in [1, mx_\Sigma]$, and $\delta \in \Delta$:

$$\begin{array}{llll}
\langle d, @, j'\rangle \to \langle d, \mathrm{down}_1\rangle & & \langle d, e, j\rangle \to \langle u_j, \mathrm{up}\rangle \\
\langle d, \delta, j\rangle \to \delta(\langle d, \mathrm{down}_1\rangle, \langle u_j, \mathrm{up}\rangle) & & \langle d, e, 0\rangle \to e \\
\langle d, \delta, 0\rangle \to \delta(\langle d, \mathrm{down}_1\rangle, e) & & \\
\langle u_1, @, j'\rangle \to \langle d, \mathrm{down}_2\rangle & & \langle u_1, \delta, j'\rangle \to e \\
\langle u_2, @, j\rangle \to \langle u_j, \mathrm{up}\rangle & & \\
\langle u_2, @, 0\rangle \to e & &
\end{array}$$

Thus, since $\tau_M = \mathrm{flat}_\Delta \circ \mathrm{enc}_\Delta$, it follows that $\mathrm{flat}_\Delta = \tau_M \circ \mathrm{dec}_\Delta \in \mathsf{dTT}^\ell_{\mathrm{su}} \circ \mathsf{dec}$.

We note that the mapping $\mathrm{flat}_\Delta \circ \mathrm{enc}_\Delta$ is denoted 'eval' in [65, Section 4], 'APP' in [61], and 'app' in [26, Section 7]. For the reader familiar with MSO translations we observe that it is also easy to show that both flat_Δ and enc_Δ are deterministic MSO translations, and hence their composition is one. The second inclusion then follows from Proposition 29. □

It follows from the first inclusion of Lemma 74 that $\mathsf{dFT} \subseteq \mathsf{dFT}_@$. In fact, $\mathsf{enc} \circ \mathsf{dTT} \circ \mathsf{dec} \subseteq \mathsf{enc} \circ \mathsf{dTT} \circ \mathsf{dTT}^\ell_\downarrow \circ \mathsf{flat}$, which is included in $\mathsf{enc} \circ \mathsf{dTT} \circ \mathsf{flat}$ by Theorem 18. It follows from the second inclusion that $\mathsf{dFT}^k_@ \subseteq \mathsf{dFT}^{k+1}$ for every $k \geq 1$. In fact, $\mathsf{dFT}^k_@ = (\mathsf{enc} \circ \mathsf{dTT} \circ \mathsf{flat})^k \subseteq (\mathsf{enc} \circ \mathsf{dTT} \circ \mathsf{dTT}^\ell_{\mathrm{su}} \circ \mathsf{dec})^k \subseteq \mathsf{enc} \circ (\mathsf{dTT} \circ \mathsf{dTT}^\ell_{\mathrm{su}})^k \circ \mathsf{dec}$, which is included in $\mathsf{enc} \circ \mathsf{dTT}^k \circ \mathsf{dTT}^\ell_{\mathrm{su}} \circ \mathsf{dec}$ by Theorem 23 and hence in $\mathsf{enc} \circ \mathsf{dTT}^{k+1} \circ \mathsf{dec}$, which equals dFT^{k+1} by Lemma 73.

Corollary 75 $\mathsf{dFT}^k \subseteq \mathsf{dFT}^k_@ \subseteq \mathsf{dFT}^{k+1}$ *for every* $k \geq 1$.

From the second inclusion we obtain that our main results also hold for deterministic extended TT forest transducers. It is decidable whether or not a composition of such transducers is of linear size increase, and

$$\mathsf{dFT}^k_@ \cap \mathsf{LSIF} = \mathsf{enc} \circ \mathsf{dTT}_{\mathrm{su}} \circ \mathsf{dec} \subseteq \mathsf{dFT} \subseteq \mathsf{dFT}_@.$$

The complexity results of Theorems 47, 49, and 50 also hold for $\mathsf{dFT}_@^k$.

The class of deterministic macro forest translations of [65] can be defined as $\mathsf{dMFT}_@ = \mathsf{enc} \circ \mathsf{dMT} \circ \mathsf{flat}$. Since $\mathsf{dTT} \subseteq \mathsf{dMT} \subseteq \mathsf{dTT}^2$ by Lemma 24, we conclude by similar arguments as for $\mathsf{dFT}_@$ that $\mathsf{dMFT}_@^k \subseteq \mathsf{dFT}^{2k+1}$ and hence our main results also hold for deterministic macro forest transducers. It is decidable whether or not a composition of such transducers is of linear size increase, and

$$\mathsf{dMFT}_@^k \cap \mathsf{LSIF} = \mathsf{enc} \circ \mathsf{dTT}_{\mathrm{su}} \circ \mathsf{dec} \subseteq \mathsf{dFT} \subseteq \mathsf{dFT}_@ \subseteq \mathsf{dMFT}_@ .$$

The complexity results of Theorems 47, 49, and 50 also hold for $\mathsf{dMFT}_@^k$.

The main results of Sects. 9 and 11 also hold for nondeterministic forest transducers. Instead of Lemma 73 we use the obvious fact that $\mathsf{TT} \circ \mathsf{dec} \circ \mathsf{enc} \circ \mathsf{TT} \subseteq \mathsf{TT} \circ \mathsf{TT}$.[18] This implies, together with Lemma 74, that it suffices to prove that the results for TT^k also hold for the class $\mathsf{enc} \circ \mathsf{TT}^k \circ \mathsf{dec}$. For the nondeterministic version of Theorem 43 in Sect. 9, we note that a translation $\mathrm{enc}_\Sigma \circ \tau \circ \mathrm{dec}_\Delta$ is a function if and only if τ is a function. Consequently, $(\mathsf{enc} \circ \mathsf{TT}^k \circ \mathsf{dec}) \cap \mathcal{F} \subseteq \mathsf{enc} \circ (\mathsf{TT}^k \cap \mathcal{F}) \circ \mathsf{dec} \subseteq \mathsf{enc} \circ \mathsf{dTT}^{k+1} \circ \mathsf{dec} = \mathsf{dFT}^{k+1}$ by Theorem 35 and Lemma 73. Hence $(\mathsf{enc} \circ \mathsf{TT}^k \circ \mathsf{dec}) \cap \mathsf{LSIF} = \mathsf{enc} \circ \mathsf{dTT}_{\mathrm{su}} \circ \mathsf{dec}$. Obviously, the complexity results of Theorems 57 and 58 in Sect. 11 hold for $\mathsf{enc} \circ \mathsf{TT}^k \circ \mathsf{dec}$, with the same proof as in the deterministic case. The class of nondeterministic macro forest translations of [65] can be defined as $\mathsf{MFT}_@ = \mathsf{enc} \circ \mathsf{MT} \circ \mathsf{flat}$. From Lemmas 27 and 74 we obtain that $\mathsf{MFT}_@^k \subseteq \mathsf{enc} \circ \mathsf{TT}^{3k} \circ \mathsf{dec}$, and hence all these results also hold for macro forest transducers.

We finally show that the results of Sect. 12 also hold for nondeterministic forest transducers. We first consider $\mathsf{enc} \circ \mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{dec}$ and $\mathsf{enc} \circ \mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{flat}$. For a forest translation τ we define the forest language $L_\tau = \{\#[fg] \mid (f, g) \in \tau\}$. If $\tau = \mathrm{enc}_\Sigma \circ \tau' \circ \mathrm{dec}_\Delta$ with $\tau' \in \mathsf{TT} \circ \mathsf{dTT}$, then $\#[fg] \in L_\tau$ if and only if $\#(\mathrm{enc}_\Sigma(f), \mathrm{enc}_\Delta(g)) \in L_{\tau'}$. Since $\mathrm{enc}_\Sigma(f)$ can be computed by a deterministic finite-state transducer (see footnote 17), and similarly for $\mathrm{enc}_\Delta(g)$, L_τ is log-space reducible to $L_{\tau'}$. Hence $\mathsf{enc} \circ \mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{dec} \subseteq \mathsf{LOGCFL}$ by Theorem 62. Similarly if $\tau' \in \mathsf{dTT}$, then $g \in \tau(L)$ if and only if $\mathrm{enc}_\Delta(g) \in \tau'(\mathrm{enc}_\Sigma(L))$ for every $L \in \mathsf{REGF}$, and hence $\mathsf{dFT}(\mathsf{REGF}) \subseteq \mathsf{LOGCFL}$ by Corollary 64. To show the same results for flat instead of dec, we need the following small lemma.

Lemma 76 $\mathsf{flat} \subseteq \mathsf{dTT}_\downarrow \circ \mathsf{yield}$.

Proof For an alphabet Δ, let Ω be the ranked alphabet $\Delta \cup \{[\,,]\} \cup \{\lambda, @, \omega\}$ such that $\Omega^{(0)} = \Delta \cup \{[\,,], \lambda\}$, $\Omega^{(2)} = \{@\}$, and $\Omega^{(4)} = \{\omega\}$. We define the deterministic $\mathrm{TT}_\downarrow^\ell$ $N = (\Delta_@, \Omega, \{p\}, p, R)$ with the following (general) rules.

$$\begin{aligned}
\langle p, j, @\rangle &\to @(\langle p, \mathrm{down}_1\rangle, \langle p, \mathrm{down}_2\rangle)\\
\langle p, j, e\rangle &\to \lambda\\
\langle p, j, \delta\rangle &\to \omega(\delta, [\,, \langle p, \mathrm{down}_1\rangle,]\,)
\end{aligned}$$

for every $\delta \in \Delta$. Assuming that the symbol λ is skipped when taking yields (cf. the sentence before Corollary 65), it should be clear that $\mathrm{flat}_\Delta(t)$ is the yield of $\tau_N(t)$ for every $t \in T_{\Delta_@}$. □

It follows from Lemma 76 and Theorem 18 that $\mathsf{enc} \circ \mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{flat} \subseteq \mathsf{enc} \circ \mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{yield}$ and $\mathsf{dFT}_@ = \mathsf{enc} \circ \mathsf{dTT} \circ \mathsf{flat} \subseteq \mathsf{enc} \circ \mathsf{dTT} \circ \mathsf{yield}$. If τ is a forest translation such that $\tau = \mathrm{enc}_\Sigma \circ \tau'$ with $\tau' \in \mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{yield}$, then $\#[fg] \in L_\tau$ if and only if $\#(\mathrm{enc}_\Sigma(f), g) \in L_{\tau'}$. Hence $\mathsf{enc} \circ \mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{flat} \subseteq \mathsf{LOGCFL}$ by Corollary 65. Similarly if $\tau' \in \mathsf{dTT} \circ \mathsf{yield}$, then

[18] It can be shown that the nondeterministic version of Lemma 73 also holds, but we will not do that here.

$\tau(L) = \tau'(\mathrm{enc}_\Sigma(L))$ for every $L \in \mathsf{REGF}$, and so $\mathsf{dFT}_@(\mathsf{REGF}) \subseteq \mathsf{LOGCFL}$ by Corollary 65. If we define the class of IO macro forest translations to be $\mathsf{enc} \circ \mathsf{MT}_{\mathrm{IO}} \circ \mathsf{flat}$, then that class is included in $\mathsf{enc} \circ \mathsf{TT} \circ \mathsf{dTT} \circ \mathsf{flat}$ by Lemma 28 and hence in LOGCFL by the above. Thus, Corollary 63 also holds for macro forest transducers.

For a forest translation $\tau = \mathrm{enc}_\Sigma \circ \tau' \circ \mathrm{dec}_\Delta$ with $\tau' \in \mathsf{TT}^k$ it is easy to prove that $L_\tau \in \mathsf{DSPACE}(n)$ and that $\tau(L) \in \mathsf{DSPACE}(n)$ for every $L \in \mathsf{REGF}$, as we did above for $\tau' \in \mathsf{TT} \circ \mathsf{dTT}$ and $\tau' \in \mathsf{dTT}$, respectively, thus generalizing Theorems 66 and 67. That also holds for flat_Δ instead of dec_Δ, because $\mathsf{enc} \circ \mathsf{TT}^k \circ \mathsf{flat} \subseteq \mathsf{enc} \circ \mathsf{TT}^{k+1} \circ \mathsf{dec}$ by Lemma 74.

The NP-completeness results of Sect. 12 also hold for extended forest translations. The translation τ of Theorem 71 can be changed into a translation in $\mathsf{enc} \circ \mathsf{dTT}_\downarrow \circ \mathsf{fTT} \circ \mathsf{flat}$ as follows. First, change M_1 in the proof of Theorem 71 such that it obtains as input the encodings of the strings ab^ncd^m (viewed as forests). Second, change M_2 such that it outputs trees over $\Delta_@$ rather than Δ (by changing the rule $\langle q_{i\vee j}, d\rangle \to \vee(\langle q_i, \alpha\rangle, \langle q_j, \alpha\rangle)$ of M in the proof of Lemma 69 into the general rule $\langle q_{i\vee j}, d\rangle \to \vee(@(\langle q_i, \alpha\rangle, \langle q_j, \alpha\rangle))$, and similarly for $\wedge$). As a result τ outputs boolean expressions as forests rather than ranked trees. Thus we obtain an NP-complete extended forest translation in $\mathsf{enc} \circ \mathsf{dTT}_\downarrow \circ \mathsf{fTT} \circ \mathsf{flat}$, and hence one in $\mathsf{MFT}_@$. In a similar way we also obtain an NP-complete forest language in $\mathsf{FT}_@(\mathsf{REGF})$. The details are left to the reader. It is not clear whether these results hold for dec instead of flat.

14 Conclusion

Our main technical result transforms a composition of k TT's into a linear-bounded composition of k TT's, cf. Corollary 38. As observed in Remark 41, our proof of this result can involve a $2(k-1)$-fold blow-up of the sizes of the transducers, which also influences the constants of their time and space complexities, cf. the sentence after Theorem 47. We do not know whether this transformation can be realized in a more efficient way.

Our main result on expressivity is that $\mathsf{dTT}^k \cap \mathsf{LSIF} \subseteq \mathsf{dTT}$ for every $k \geq 1$, i.e., that every composition of dTT's that is of linear size increase can be realized by one dTT. Moreover, it is decidable whether or not such a composition is of linear size increase. Do similar results hold for polynomial size increase? For instance, does there exist $m \geq 1$ such that every translation in $\bigcup_{k\geq 1} \mathsf{dTT}^k$ of quadratic size increase is in dTT^m? The same question can be asked for ℓ-fold exponential size increase, for each fixed $\ell \in \mathbb{N}$.

We have shown in Sect. 7 that even $\mathsf{TT}^k \cap \mathsf{LSIF} \subseteq \mathsf{dTT}$ for every $k \geq 1$, generalizing Theorem 43. Although this result is effective, we do not know whether Theorem 44 can also be generalized, i.e., whether it is decidable for a nondeterministic TT M whether or not τ_M is a function of linear size increase. This would be solved if it was decidable whether or not τ_M is a function. But that is also unknown, whereas it has been proved for classical top-down tree transducers (with regular look-ahead) in [36, Theorem 8]. Note that deciding functionality of τ_M also solves the equivalence problem for dTT's, which is already a long standing open problem (cf. [22,60]); in fact, $\tau_1, \tau_2 \in \mathsf{dTT}$ are the same if and only if they have the same domain and $\tau_1 \cup \tau_2$ is functional.

Another open question for nondeterministic TT's is whether or not there exists $m \geq 1$ such that the inclusion $\mathsf{TT}^k \cap \mathsf{LSIR} \subseteq \mathsf{TT}^m$ holds for every $k \geq 1$, where LSIR consists of all relations $\tau \subseteq T_\Sigma \times T_\Delta$ of linear size increase, which means that there is a constant $c \in \mathbb{N}$ such that $|s| \leq c \cdot |t|$ for every $(t, s) \in \tau$. It follows from (the proof of) [48, Theorem 3.21] (see also [49,50]) that $\mathsf{TT}^2 \cap \mathsf{LSIR}$ is not included in MT, and hence not in TT by the remark following Lemma 27.

Similar questions can be asked for macro tree transducers, i.e., for the classes dMT and MT.

We have shown in Lemma 12 that $\mathsf{dTT}_{\downarrow} = \mathsf{dTT}^{\mathrm{s}}_{\downarrow}$, but we do not know whether or not $\mathsf{dTT} = \mathsf{dTT}^{\mathrm{s}}$. In other words, we do not know whether for every TT there is an equivalent sub-testing TT, in which the regular test of a rule only inspects the subtree of the current node. Or even more informally, can regular look-around be simulated by regular look-ahead?

We have shown in Corollary 59 that the string languages in the OI-hierarchy, which are generated by high-level grammars, are in $\mathsf{NSPACE}(n) \wedge \mathsf{NPTIME}$, and in Corollary 68 that they are in $\mathsf{DSPACE}(n)$. However, the languages of the OI-hierarchy are generated by so-called "safe" high-level grammars, and it is not known whether the same results hold for unsafe high-level grammars. It is proved in [54] that the languages generated by unsafe level-2 grammars, the unsafe version of $\mathsf{OI}(2)$, are in $\mathsf{NSPACE}(n)$.

In Sect. 12 we have shown that $\mathsf{dTT}^k \subseteq \mathsf{PTIME}$, that $\mathsf{TT} \circ \mathsf{dTT} \subseteq \mathsf{LOGCFL} \subseteq \mathsf{PTIME}$, and that $\mathsf{dTT} \circ \mathsf{TT}$ contains an NP-complete translation. It remains to find out for $k \geq 2$ whether $\mathsf{TT} \circ \mathsf{dTT}^k \subseteq \mathsf{PTIME}$ or whether it contains an NP-complete translation.

Acknowledgements We are grateful to the reviewers for their constructive comments.

References

1. Aho, A.V.: Indexed grammars—an extension of context-free grammars. J. ACM **15**, 647–671 (1968)
2. Aho, A.V., Ullman, J.D.: Translations on a context-free grammar. Inf. Control **19**, 439–475 (1971)
3. Asveld, P.R.J.: Time and space complexity of inside-out macro languages. Int. J. Comput. Math. **10**, 3–14 (1981)
4. Baker, B.S.: Generalized syntax-directed translation, tree transducers, and linear space. SIAM J. Comput. **7**, 376–391 (1978)
5. Bartha, M.: An algebraic definition of attributed transformations. Acta Cybern. **5**, 409–421 (1982)
6. Bloem, R., Engelfriet, J.: Monadic second order logic and node relations on graphs and trees. In: Mycielski, J., Rozenberg, G., Salomaa, A. (eds.) Structures in Logic and Computer Science. Lecture Notes in Computer Science, vol. 1261, pp. 144–161. Springer, Berlin. A corrected version is available at https://www.researchgate.net/publication/221350026 (1997)
7. Bloem, R., Engelfriet, J.: A comparison of tree translations defined by monadic second order logic and by attribute grammars. J. Comput. Syst. Sci. **61**, 1–50 (2000)
8. Bogaert, B., Tison, S.: Equality and disequality constraints on direct subterms in tree automata. In: Finkel, A., Jantzen, M. (eds.) Proceedings of STACS'92. Lecture Notes in Computer Science, vol. 577, pp. 161–171. Springer, Berlin (1992)
9. Bojańczyk, M., Colcombet, T.: Tree-walking automata do not recognize all regular languages. SIAM J. Comput. **38**, 658–701 (2008)
10. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM **28**, 114–133 (1981)
11. Comon, H. et al.: Tree Automata Techniques and Applications. http://tata.gforge.inria.fr/. Accessed 15 Mar 2019
12. Cook, S.A.: Characterizations of pushdown machines in terms of time-bounded computers. J. ACM **18**, 4–18 (1971)
13. Courcelle, B.: Monadic second-order definable graph translations: a survey. Theor. Comput. Sci. **126**, 53–75 (1994)
14. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic. Cambridge University Press, Cambridge (2012)
15. Courcelle, B., Franchi-Zannettacci, P.: Attribute grammars and recursive program schemes I, II. Theor. Comput. Sci. **17**(163–191), 235–257 (1982)
16. Damm, W.: The IO- and OI-hierarchies. Theor. Comput. Sci. **20**, 95–207 (1982)
17. Deransart, P., Jourdan, M., Lorho, B.: Attribute Grammars—Definitions, Systems and Bibliography. Lecture Notes in Computer Science, vol. 323. Springer, Berlin (1988)
18. Doner, J.: Tree acceptors and some of their applications. J. Comput. Syst. Sci. **4**, 406–451 (1970)

19. Engelfriet, J.: Tree automata and tree grammars. DAIMI FN-10 Lecture Notes, Aarhus University. A slightly revised version is available at arXiv:1510.02036 (1975)
20. Engelfriet, J.: Top-down tree transducers with regular look-ahead. Math. Syst. Theory **10**, 289–303 (1977)
21. Engelfriet, J.: On tree transducers for partial functions. Inf. Process. Lett. **7**, 170–172 (1978)
22. Engelfriet, J.: Some open questions and recent results on tree transducers and tree languages. In: Book, R.V. (ed.) Formal Language Theory—Perspectives and Open Problems, pp. 241–286. Academic Press, London (1980)
23. Engelfriet, J.: Attribute grammars: attribute evaluation methods. In: Lorho, B. (ed.) Methods and Tools for Compiler Construction, pp. 103–138. Cambridge University Press, Cambridge (1984)
24. Engelfriet, J.: Context-free grammars with storage. Technical Report 86-11, University of Leiden. A slightly revised version is available at arXiv:1408.0683 (1986)
25. Engelfriet, J.: The complexity of languages generated by attribute grammars. SIAM J. Comput. **15**, 70–86 (1986)
26. Engelfriet, J.: The time complexity of typechecking tree-walking tree transducers. Acta Inform. **46**, 139–154 (2009)
27. Engelfriet, J., Filé, G.: The formal power of one-visit attribute grammars. Acta Inform. **16**, 275–302 (1981)
28. Engelfriet, J., Hoogeboom, H.J., Samwel, B.: XML navigation and transformation by tree-walking automata and transducers with visible and invisible pebbles. Technical Report. arXiv:1809.05730 (2018)
29. Engelfriet, J., Maneth, S.: Macro tree transducers, attribute grammars, and MSO definable tree translations. Inf. Comput. **154**, 34–91 (1999)
30. Engelfriet, J., Maneth, S.: Output string languages of compositions of deterministic macro tree transducers. J. Comput. Syst. Sci. **64**, 350–395 (2002)
31. Engelfriet, J., Maneth, S.: A comparison of pebble tree transducers with macro tree transducers. Acta Inform. **39**, 613–698 (2003)
32. Engelfriet, J., Maneth, S.: Macro tree translations of linear size increase are MSO definable. SIAM J. Comput. **32**, 950–1006 (2003)
33. Engelfriet, J., Schmidt, E.M.: IO and OI, Part II. J. Comput. Syst. Sci. **16**, 67–99 (1978)
34. Engelfriet, J., Vogler, H.: Macro tree transducers. J. Comput. Syst. Sci. **31**, 71–146 (1985)
35. Engelfriet, J., Vogler, H.: High level tree transducers and iterated pushdown tree transducers. Acta Inform. **26**, 131–192 (1988)
36. Ésik, Z.: Decidability results concerning tree transducers I. Acta Cybern. **5**, 1–20 (1980)
37. Fischer, M.J.: Grammars with Macro-Like Productions. Ph.D. thesis, Harvard University (1968)
38. Fülöp, Z.: On attributed tree transducers. Acta Cybern. **5**, 261–279 (1981)
39. Fülöp, Z., Vogler, H.: Syntax-Directed Semantics—Formal Models Based on Tree Transducers. Springer, Berlin (1998)
40. Ganzinger, H.: Increasing modularity and language-independency in automatically generated compilers. Sci. Comput. Program. **3**, 223–278 (1983)
41. Ganzinger, H., Giegerich, R.: Attribute coupled grammars. In: Proceedings of SIGPLAN'84. SIGPLAN Notices, vol. 19, pp. 157–170 (1984)
42. Garey, M.R., Johnson, D.S.: Computers and Intractability—A Guide to the Theory of NP-Completeness. W. H. Freeman and Co, New York (1979)
43. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984). A re-edition is available at arXiv:1509.06233
44. Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3 (Chapter 1). Springer, Berlin (1997)
45. Giegerich, R.: Composition and evaluation of attribute coupled grammars. Acta Inform. **25**, 355–423 (1988)
46. Harrison, M.A.: Introduction to Formal Language Theory. Addison-Wesley, Boston (1978)
47. Hosoya, H.: Foundations of XML Processing—The Tree-Automata Approach. Cambridge University Press, Cambridge (2011)
48. Inaba, K.: Complexity and Expressiveness of Models of XML Transformations. Ph.D. thesis, The University of Tokyo. http://www.kmonos.net/pub/files/phd.pdf (2009). Accessed 15 Mar 2019
49. Inaba, K., Hosoya, H.: Multi-return macro tree transducers. In: Proceedings of PLAN-X 2008. http://www.kmonos.net/pub/files/mrmtt08.pdf (2008)
50. Inaba, K., Hosoya, H., Maneth, S.: Multi-return macro tree transducers. In: Ibarra, O.H., Ravikumar, B. (eds.) Proceedings of CIAA'08. Lecture Notes in Computer Science, vol. 5148, pp. 102–111. Springer, Berlin (2008)

51. Inaba, K., Maneth, S.: The complexity of tree transducer output languages. In: Hariharan, R., Mukund, M., Vinay, V. (eds) Proceedings of FSTTCS'08, pp. 244–255. http://drops.dagstuhl.de/opus/volltexte/2008/1757 (2008). Accessed 15 Mar 2019
52. Inaba, K., Maneth S.: The complexity of translation membership for macro tree transducers. In: Proceedings of PLAN-X'09. arXiv:0910.2315 (2009)
53. Knuth, D.E.: Semantics of context-free languages. Math. Syst. Theory **2**, 127–145 (1968)
54. Kobayashi, N., Inaba, K., Tsukada, T.: Unsafe order-2 tree languages are context-sensitive. In: Muscholl, A. (ed.) Proceedings of FOSSACS'14. Lecture Notes in Computer Science, vol. 8412, pp. 149–163. Springer, Berlin (2014)
55. Kühnemann, A.: Berechnungsstärken von Teilklassen primitiv-rekursiver Programmschemata. Ph.D. thesis, Technical University of Dresden, Shaker Verlag (1997)
56. Kühnemann, A.: Benefits of tree transducers for optimizing functional programs. In: Arvind, V., Ramanujam, R. (eds) Proceedings of FSTTCS'98. Lecture Notes in Computer Science, vol. 1530, pp. 146–158. Springer, Berlin (1998)
57. Lewis, P.M., Stearns, R.E., Hartmanis, J.: Memory bounds for the recognition of context-free and context-sensitive languages. In: Proceedings of 6th Annual IEEE Symposium on Switching Circuit Theory and Logical Design, pp. 191–212 (1965)
58. Maneth, S.: The complexity of compositions of deterministic tree transducers. In: Agrawal, M., Seth, A. (eds) Proceedings of FSTTCS'02. Lecture Notes in Computer Science, vol. 2556, pp. 265–276. Springer, Berlin (2002)
59. Maneth, S.: The macro tree transducer hierarchy collapses for functions of linear size increase. In: Pandya, P.K., Radhakrishnan, J. (eds) Proceedings of FSTTCS'03. Lecture Notes in Computer Science, vol. 2914, pp. 326–337. Springer, Berlin (2003)
60. Maneth, S.: A survey on decidable equivalence problems for tree transducers. Int. J. Found. Comput. Sci. **26**, 1069–1100 (2015)
61. Maneth, S., Berlea, A., Perst, T., Seidl, H.: XML type checking with macro tree transducers. In: Proceedings of PODS'05, pp. 283–294. ACM Press (2005). Technical Report TUM-I0407 of the Technische Universität München (2004) is available at https://www.researchgate.net/publication/221559877
62. Maneth, S., Friese, S., Seidl, H.: Type checking of tree walking transducers. In: D'Souza, D., Shankar, P. (eds.) Modern Applications of Automata Theory. IISc Research Monographs Series 2, pp. 325–372. World Scientific, Singapore (2012)
63. Milo, T., Suciu, D., Vianu, D.: Typechecking for XML transformers. J. Comput. Syst. Sci. **66**, 66–97 (2003)
64. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Boston (1994)
65. Perst, T., Seidl, H.: Macro forest transducers. Inf. Process. Lett. **89**, 141–149 (2004)
66. Rounds, W.C.: Mappings and grammars on trees. Math. Syst. Theory **4**, 257–287 (1970)
67. Rounds, W.C.: Complexity of recognition in intermediate-level languages. In: Proceedings of 14th Annual Symposium on Switching and Automata Theory, pp. 145–158 (1973)
68. Ruzzo, W.L.: Tree-size bounded alternation. J. Comput. Syst. Sci. **21**, 218–235 (1980)
69. Schwentick, T.: Automata for XML—a survey. J. Comput. Syst. Sci. **73**, 289–315 (2007)
70. Slutzki, G.: Alternating tree automata. Theor. Comput. Sci. **41**, 305–318 (1985)
71. Sudborough, I.H.: On the tape complexity of deterministic context-free languages. J. ACM **25**, 405–414 (1978)
72. Thatcher, J.W.: Generalized^2 sequential machine maps. J. Comput. Syst. Sci. **4**, 339–367 (1970)
73. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory **2**, 57–81 (1968)
74. Van Leeuwen, J.: The membership question for ETOL-languages is polynomially complete. Information Processing Letters **3**, 138–143 (1975)
75. Vogler, H.: The OI-hierarchy is closed under control. Information and Computation **78**, 187–204 (1988)