The Complexity of Tree Transducer Output Languages

Kazuhiro Inaba[1] and Sebastian Maneth[2,3]

[1] The University of Tokyo, kinaba@is.s.u-tokyo.ac.jp
[2] National ICT Australia, sebastian.maneth@nicta.com.au
[3] University of New South Wales, Sydney

Abstract. Two complexity results are shown for the output languages generated by compositions of macro tree transducers. They are in NSPACE(n) and hence are context-sensitive, and the class is NP-complete.

1 Introduction

Macro tree transducers (mtts) [12, 14] are a finite-state machine model of tree-to-tree translations. They are motivated by syntax-directed semantics of programming languages and recently have been applied to XML transformations and query languages [18, 21]. Mtts are a combination of top-down tree transducers [22, 24] and macro grammars [13]. They process the input tree top-down while accumulating several output trees using their context parameters. Sequential composition of mtts gives rise to a powerful hierarchy (the "mtt-hierarchy") of tree translations which contains most known classes of tree translations such as those realized by attribute grammars, by MSO-definable tree translations [5], or by pebble tree transducers [20]. Consider the range, or output language, of a tree translation; it is a set of trees. If we apply "yield" to these trees, i.e., concatenate their leaf symbols from left to right, we obtain a string language. The string languages obtained in this way from the mtt-hierarchy form a large class (containing for instance the IO- and OI-hierarchies [6]) with good properties, such as being a full AFL and having decidable membership, emptiness, and finiteness [7].

In this paper we study the complexity of the output (string or tree) languages of the mtt-hierarchy. Note that we do not explicitly distinguish between string or tree output languages here, because the translation "yield" which turns a tree into its frontier string (seen as a monadic tree) is a particular simple macro tree translation itself and hence the corresponding classes have the same complexity. Small subclasses of our class of languages considered here are the IO-macro languages (or, equivalently, the yields of context-free-tree languages under IO-derivation) and the string languages generated by attribute grammars. Both of these classes are LOG(CFL)-complete by [2] and [10], respectively. Another subclass of our class is that of OI-macro languages, which are equivalent to the indexed languages [1], by [13]. This class is known to be NP-complete [23]. Hence, our class is NP-hard too (even already at level 2). Our first main result is that output languages of the mtt-hierarchy are NP-complete; thus, the complexity remains in NP when going from indexed languages to the full mtt-hierarchy. In terms of space complexity, languages generated by compositions of top-down tree transducers (mtts without context parameters) are known to be in DSPACE(n) [3]. This result was generalized in [17] to compositions of *total deterministic* mtts. Our second main result is that output languages of the mtt-hierarchy (generated by compositions of *nondeterministic* mtts) with regular tree languages as inputs are in NSPACE(n) and thus are context-sensitive. The approach of our proof can be seen as a generalization of the proofs in [3] and [17]; moreover, we make essential use of the idea of compressed representation of backtracking information, used by Aho in [1] for showing that the indexed languages are in NSPACE(n).

We first solve the "translation membership" problem for a single mtt M. That is, we show that, given trees s and t, we can determine whether or not the pair (s, t) is in M's translation, in linear space and polynomial time with respect to $|s| + |t|$ on a nondeterministic Turing Machine ($|s|$ denotes the size of the tree s). The challenge here is the space complexity; we use a compressed representation of M's output trees for input s, inspired by [19], and then check if t is contained using a recursive procedure in which nodes needed for backtracking are compressed using a trie, similar to Aho's compression of index strings in [1]. Then, we generalize these results from one mtt to compositions of mtts. Here, the challenge is the existence of intermediate trees. Consider the composition τ of two translations realized by mtts: τ_1 followed by τ_2. To check $(s,t) \in \tau$, we nondeterministically guess an intermediate tree u, and check whether $(s,u) \in \tau_1$ and $(u,t) \in \tau_2$. From the complexity result of single mtts, we know that this can be done in $O(|s|+|u|+|t|)$ space. This can, however, be much larger than $O(|s|+|t|)$; the size $|u|$ of the intermediate tree u can actually be double-exponentially larger than $|s|$ and $|t|$. The basic idea to prove the linear size complexity for compositions of mtts is to bound the sizes of all such intermediate input trees. This is achieved by putting the mtts in certain normal forms such that they do not delete much of their input, in the sense that every output tree t has a corresponding input tree of size only linearly larger than $|t|$. In fact, also the initial input tree s can be changed into a smaller tree s' of size linear in $|t|$, for which $\tau(s') = \tau(s)$. Limiting the size of intermediate trees can also be used for showing the time complexity NP. We nondeterministically guess all linear size intermediate trees, and test the translation membership of each single mtt in polynomial time.

Although the idea of bounding the size of intermediate trees is quite similar to [3] and [17], the existence of context parameters and nondeterminism together adds some difficulty in every step of the proof. For example, consider the mtt M_{dexp} with the following three rules r_0, r_1, and r_2:

$$\langle q_0, \mathtt{a}(x)\rangle \to \langle q,x\rangle(\langle q,x\rangle(\mathtt{e})) \qquad (r_0) \qquad\qquad \langle q,\mathtt{e}\rangle(y) \to +(\mathtt{b}(y,y),\mathtt{c}(y,y)) \qquad (r_2)$$
$$\langle q, \mathtt{a}(x)\rangle(y) \to \langle q,x\rangle(\langle q,x\rangle(y)) \qquad (r_1)$$

Here, $+$ denotes a nondeterministic choice; e.g., when the state q reads an input node labeled $\mathtt{e}$, it generates an output node labeled either $\mathtt{b}$ or $\mathtt{c}$. This mtt takes a tree of form $\mathtt{a}(\mathtt{a}(\cdots \mathtt{a}(\mathtt{e})\cdots))$ as input (with n occurrences of $\mathtt{a}$) and generates a full binary tree of height 2^n (note that, without parameters, the height growth can only be linear) with each non-leaf node arbitrarily labeled either $\mathtt{b}$ or $\mathtt{c}$. Therefore, the size of the set of possible output trees is $2^{2^{2^n}}$. To decide whether $(s,t) \in \tau_{M_{\mathrm{dexp}}}$ for given trees s and t, we essentially have to find the correct choice among the triple exponentially many candidates. To address the issue, we (1) instead of solving the membership problem for all mtts, only deal with mtts in the above mentioned non-deleting normal form, and which are linear with respect to the input variables, and (2) exploit the compressed representation of outputs of mtts [19] for manipulating the output set.

Note that due to nondeterminism we cannot anymore obtain some of the useful normal forms (non-erasing, which is similar to having no ϵ-moves in a string transducer, and non-parameter-deleting) used in [17] for total deterministic mtts. We address this issue by introducing a slight extension of mtts, namely, *mtts with choice and failure*, which re-enable us to have the non-erasing normal form, and compensate the absence of the non-parameter-deleting normal form.

2 Preliminaries

We denote by ϵ the *empty list*, i.e., a list of length 0, and by $l_1.l_2$ the concatenation of two lists l_1 and l_2. A list l is said to be a *prefix* of a list l' if there is a list l'' such that $l.l'' = l'$, and to be a *proper prefix* if $l'' \neq \epsilon$. For a finite set A, we denote by $|A|$ the number of its elements. A finite set Σ with a mapping *rank* : $\Sigma \to \mathbb{N}$ is called a *ranked alphabet*. We often write $\sigma^{(k)}$ to indicate that *rank*$(\sigma) = k$ and write $\Sigma^{(k)}$ to denote the subset of Σ of rank-k symbols. The *product* of Σ and a set B is the ranked alphabet $\Sigma \times B = \{\langle\sigma,b\rangle^{(k)} \mid \sigma^{(k)} \in \Sigma, b \in B\}$. Throughout the paper, we fix the sets of input variables $X = \{x_1, x_2, \dots\}$ and parameters $Y = \{y_1, y_2, \dots\}$ which are all of rank 0, and the set of choice nodes $C = \{\theta^{(0)}, +^{(2)}\}$. We assume any other alphabet to be disjoint with X, Y, and C. The set X_i is defined as $\{x_1,\dots,x_i\}$, and Y_i is defined similarly.

The set T_Σ of *trees* t over a ranked alphabet Σ is defined by the BNF $t ::= \sigma(\overbrace{t,\dots,t}^{k})$ for $\sigma \in \Sigma^{(k)}$. Instead of $\sigma()$ we usually write σ. We recursively define the function *label*$_\Sigma$ from $T_\Sigma \times \mathbb{N}^*$ to

Σ as follows. For $t = \sigma(t_1,\ldots,t_k)$, $\sigma^{(k)} \in \Sigma$, $k \geq 0$, and $t_1,\ldots,t_k \in T_\Sigma$, $\mathit{label}_\Sigma(t,\epsilon) = \sigma$ and $\mathit{label}_\Sigma(t,i.\nu) = \mathit{label}_\Sigma(t_i,\nu)$. Thus, the empty list ϵ denotes the root node and $\nu.i$ denotes the i-th child of ν. We omit the subscript $_\Sigma$ if clear from the context. We define the set $\mathit{pos}(t) = \{\nu \in \mathbb{N}^* \mid \mathit{label}(t,\nu) \text{ is defined}\}$. For $p \in \mathit{pos}(t)$, $t|_\nu$ denotes the subtree of t rooted at the node ν. We write $|t|$ as a shorthand for $|\mathit{pos}(t)|$. A subset $L \subseteq T_\Sigma$ is called a *tree language*. By REGT, we denote the class of *regular tree languages* [15]. Let Σ and Δ be ranked alphabets. A relation $\tau \subseteq T_\Sigma \times T_\Delta$ is called a *tree translation* (over Σ and Δ) or simply a translation. For two translations τ_1 and τ_2, their sequential composition $\tau_1\,;\tau_2$ ("τ_1 followed by τ_2") is the translation $\{(x,z) \mid \exists y((x,y) \in \tau_1, (y,z) \in \tau_2)\}$. For two classes T_1 and T_2 of translations, we define $T_1\,;T_2 = \{\tau_1\,;\tau_2 \mid \tau_1 \in T_1, \tau_2 \in T_2\}$. The k-fold composition of the class T of translations is denoted by T^k. For a tree language L and a translation τ, $\tau(L) = \bigcup\{\tau(t) \mid t \in L\}$.For a class F of languages and a class T of translations, we denote by $T(F)$ the class of output languages $\{\tau(L) \mid \tau \in T, L \in F\}$.

A *macro tree transducer (mtt)* M is a tuple (Q,Σ,Δ,q_0,R), where Q is the ranked alphabet of *states*, Σ and Δ are the *input* and *output* alphabets, $q_0 \in Q^{(0)}$ is the *initial state*, and R is the finite set of *rules* of the form $\langle q,\sigma(x_1,\ldots,x_k)\rangle(y_1,\ldots,y_m) \to r$ where $q \in Q^{(m)}$, $\sigma \in \Sigma^{(k)}$, and r is a tree in $T_{\Delta\cup(Q\times X_k)\cup Y_m}$. Rules of such form are called $\langle q,\sigma\rangle$-rules, and the set of right-hand sides of all $\langle q,\sigma\rangle$-rules is denoted by $R_{q,\sigma}$. We always assume $\Sigma^{(0)}$ and $\Delta^{(0)}$ (and thus, T_Σ and T_Δ) are non-empty. The rules of M are used as term rewriting rules in the usual way. We denote by $\Rightarrow_M$ the derivation relation of M on $T_{(Q\times T_\Sigma)\cup\Delta}$, and by $u{\downarrow_M}$ the set $\{t \in T_\Delta \mid u \Rightarrow_M^* t\}$. Note that "state-calls" $\langle q,x_i\rangle$ can be nested and therefore different orders of evaluation yield different trees. Unless otherwise specified, we assume the *outside-in* (OI) derivation in which we always rewrite the outermost (= top-most) state calls. By Corollary 3.13 of [12], this order of evaluation yields the same set of output trees as the *unrestricted* order, i.e., the case where no restriction is imposed on the order of evaluation. The *translation realized by* M is the relation $\tau_M = \{(s,t) \in T_\Sigma \times T_\Delta \mid t \in \langle q_0,s\rangle{\downarrow_M}\}$. We omit the subscript $_M$ if it is clear from the context. We denote by MT the class of translations realized by mtts. An mtt is called a top-down tree transducer (tt) if all its states are of rank 0; the corresponding class of translations is denoted by T. We call an mtt *deterministic* (*total*, respectively) if for every $\langle q,\sigma\rangle \in Q \times \Sigma$, the number $|R_{q,\sigma}|$ of rules is at most (at least) one; the corresponding classes of translations are denoted by prefix D ($_\mathrm{t}$). An mtt is *linear* (denoted by prefix L) if in every right-hand side of its rules each input variable $x_i \in X$ occurs at most once. The same notation is used for tts; for instance, $\mathrm{D_tT}$ denotes the class of translations realized by total deterministic tts.

Macro Tree Transducers with Choice and Failure For a technical reason, we define a slight extension of mtts. An *mtt with choice and failure (mttcf)* M is a tuple (Q,Σ,Δ,q_0,R) defined as for normal mtts, except that the right-hand sides of rules are trees in $T_{\Delta\cup(Q\times X_k)\cup Y_m\cup C}$ (recall that $C = \{\theta^{(0)}, +^{(2)}\}$). The derivation relations ($\Rightarrow_M$ and $\downarrow_M$) and the realized translation (τ_M) are defined similarly as for mtts, with two additional rewrite rules: $+(t_1,t_2) \Rightarrow_M t_1$ and $+(t_1,t_2) \Rightarrow_M t_2$. Thus, $+$ denotes nondeterministic choice and θ denotes failure (because there is no rule for it). Again, we assume the outside-in evaluation order. For a right-hand side r of an mttcf, we say a position $\nu \in \mathit{pos}(r)$ is *top-level* if for all proper prefixes ν' of ν, $\mathit{label}(r,\nu') \in \Delta \cup C$. We say an mttcf is *canonical* if for every right-hand side r and for every top-level position $\nu \in \mathit{pos}(r)$, $\mathit{label}(r,\nu) \notin C$.

The idea of the choice and failure nodes comes from [12]; there they show that $\mathrm{MT} = \mathrm{D_tMT}\,;\mathrm{SET}$, where SET is the class of translations $\mathit{set}_\Delta : T_{\Delta\cup C} \to T_\Delta$ with $\mathit{set}_\Delta(\theta) = \emptyset$, $\mathit{set}_\Delta(+(c_1,c_2)) = \mathit{set}(c_1) \cup \mathit{set}(c_2)$, and $\mathit{set}_\Delta(\delta(c_1,\ldots,c_k)) = \{\delta(t_1,\ldots,t_k) \mid t_i \in \mathit{set}_\Delta(c_i)\}$ for $\delta \in \Delta$. Let us briefly summarize the proof. For any mttcf (or mtt) M, we can always construct a total deterministic mttcf M' that realizes the same translation, by taking the $\langle q,\sigma\rangle$-rule of M' as $\langle q,\sigma(\cdots)\rangle(\cdots) \to +(r_1,+(r_2,\ldots,+(r_n,\theta)\cdots))$ where $\{r_1,\ldots,r_n\} = R_{q,\sigma}$. Also note that the mttcf $M' = (Q,\Sigma,\Delta,$ $q_0,R')$ can be regarded as the mtt $M'' = (Q,\Sigma,\Delta\cup C,q_0,R')$, by merely interpreting the θ and $+$ nodes as output symbols. Each output of M'' is a "choice tree" denoting the set of possible output trees. Obviously, the translation *set*$_\Delta$ carries out this interpretation of choice trees, and thus we have $\tau_M = \tau_{M''} \,; \textit{set}_\Delta$.

The reason why we introduce mttcfs is twofold. One reason is the decomposition result explained above, through which we give the complexity bound of a single mtt translation in Section 3. The other reason is its more flexible use of nondeterminism and partiality. Suppose an mttcf rule $\langle q, \sigma(x_1)\rangle(y_1) \to \langle p, x_1\rangle(+(\delta_1, \delta_2))$. This could produce more output than an mtt with the pair of rules $\langle q, \sigma(x_1)\rangle(y_1) \to \langle p, x_1\rangle(\delta_1)$ and $\langle q, \sigma(x_1)\rangle(y_1) \to \langle p, x_1\rangle(\delta_2)$, because the state p may copy its parameter. Although in fact we can still emulate $+$ and θ by introducing auxiliary states, it seems much simpler to use $+$ and θ. For example, mttcfs have simple normal forms, such as non-erasure (Section 4.1), while the emulating mtts do not have such a normal form.

3 Complexity of a Single MTT

In this section we show that for any canonical mttcf M having properties called *path-linear* and *non-erasing*, there is a nondeterministic Turing Machine that decides whether a given pair (s, t) of trees is in τ_M in $O(|s| + |t|)$ space and in polynomial time with respect to $|s| + |t|$. Thus, this “translation membership” problem is in NSPACE(n) and NP. Two previous works on the same membership problem for restricted classes of macro tree transducers – for total deterministic mtts [17] and for nondeterministic mtts without parameters (top-down tree transducers) [3] – both give DSPACE(n) algorithms. First let us briefly explain where the difficulty arises in our case, i.e., with nondeterminism and parameters. For total deterministic mtts, the DSPACE(n) complexity is proved via a reduction to the case of linear total deterministic mtts, and then to attribute grammars (which are deterministic by default), whose output languages are LOG(CFL)-complete and therefore have DSPACE$(\log(n)^2)$ membership test [10]. For nondeterministic tts, the complexity is achieved by a straightforward backtracking-based algorithm; given the input tree s and the output tree t, it generates each possible output of s by simulating the recursive execution of state calls, while comparing with t. The following two facts imply the DSPACE(n) complexity: (1) the depth of the recursion is at most the height of s, and (2) to backtrack we only need to remember for each state call the rule that was applied (which requires constant space). Note that neither (1) nor (2) hold for mtts; the recursion depth can be exponential and the actual parameters passed to each state call must also be remembered for backtracking.

Here we concentrate on a restricted class of mttcfs, namely, *canonical*, *non-erasing*, and *path-linear* mttcfs, which is exactly the class of mttcfs needed later in Section 4, to obtain the complexity result for the output languages of the mtt-hierarchy. For a canonical mtt, we define a right-hand side of a rule to be *non-erasing* if it is *not* in Y. A canonical mttcf is *non-erasing* if the right-hand sides of all its rules are non-erasing. An mttcf is *path-linear* if a subtree of the form $\langle q, x_i\rangle(\cdots \langle p, x_j\rangle(\cdots)\cdots)$ in its rules implies $i \neq j$.

Making MTTCFs Total Deterministic Let M be a canonical, non-erasing, and path-linear mttcf. It is easy to see that As noted in Section 2, we can always construct a total deterministic mttcf M' equivalent to M by simply taking $\langle q, \sigma(\cdots)\rangle(\cdots) \to +(r_1, \cdots, +(r_n, \theta)\cdots)$ for $\{r_1, \ldots, r_n\} = R_{q,\sigma}$. Then, $M' = (Q, \Sigma, \Delta, q_0, R')$ can be seen as a total deterministic *mtt* $N = (Q, \Sigma, \Delta \cup C, q_0, R')$ whose outputs are the choice trees denoting sets of output trees of M. The canonicity and the non-erasure of M implies that in any right-hand side $r \in R'$ and every position $\nu \in \textit{pos}(r)$ with $\textit{label}(\nu) \in Y$, there exists a proper prefix ν' of ν with $\textit{label}(\nu') \neq +$. Path-linearity is preserved from M to M'.

Compressed Representation Our approach is to represent the output choice tree $\tau_N(s)$ in a compact (linear size) structure, and then compare it to the given output tree t. Given a total deterministic mtt N and an input tree $s \in T_\Sigma$, we can, in time $O(|s|)$, calculate a straight-line context-free tree grammar (or SLG, a context-free tree grammar that has no recursion and generates exactly one output) of size $O(|s|)$ that generates $\tau_N(s)$, using the idea of [19]. Rather than repeating the full construction of [19], we here give a direct representation of the nodes of $\tau_N(s)$.

Let N be a total, deterministic, non-erasing, and path-linear mtt with output alphabet $\Delta \cup C$ and let s be an input tree. Let $E = \{(r, \nu) \mid q \in Q, \sigma \in \Sigma, r \in R_{q,\sigma}, \nu \in \mathit{pos}(r)\}$. For a list $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ of elements of E, we define *orig*(e) (the *orig*in of e) as $\epsilon.i_0 \dots i_{k-1}$ where k is the smallest index satisfying *label*$(r_k, \nu_k) \notin Q \times X$ (or, let $k = n+1$ when all labels are in $Q \times X$) and i_j is the number such that $\langle q, x_{i_j} \rangle =$ *label*(r_j, ν_j) for some q. We call e *well-formed* if *label*$(r_i, \nu_i) \in Q \times X$ for every $i < n$, *label*$(r_n, \nu_n) \in \Delta \cup C$, and *orig*$(e) \in$ *pos*(s). Intuitively, e is a partial derivation or a "call stack" of the mtt N. Each node of $\tau_N(s)$ can be represented by a well-formed list, which can be stored in $O(|s|)$ space because its length is at most $1 +$ (height of s) and the size of each element depends only on the size of the fixed mtt, not on $|s|$. Note that e can represent many nodes in $\tau_N(s)$ if the mtt is non-linear in the parameters. For instance, for M_{dexp} from the Introduction and the input tree $s_3 = \mathtt{a}(\mathtt{a}(\mathtt{a}(\mathtt{e})))$, the list $(r_0, \epsilon.1)(r_1, \epsilon.1)(r_1, \epsilon.1)(r_2, \epsilon.1)$ represents all $\mathtt{b}$-nodes at depth 16 of the tree $\tau_{M_{\text{dexp}}}(s_3)$, of which there are 2^8 many. The label *c-label*(e) of the node represented by e is *label*(r_n, ν_n). The operation *c-child*(e, i) which calculates the representation of the i-th child of the node represented by e is defined in terms of the following three operations. For a well-formed list $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ with *rank*(*c-label*$(e)) = m$, we define *down*$_i(e)$ for $1 \leq i \leq m$ as $(r_0, \nu_0) \dots (r_n, \nu_n.i)$. For $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ such that *label*$(r_n, \nu_n) = y_i \in Y$, we define *pop*$(e) = (r_0, \nu_0) \dots (r_{n-1}, \nu_{n-1}.i)$. For a list $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ where *label*$(r_n, \nu_n) = \langle q, x_j \rangle \in Q \times X$, we define *expand*$(e) = (r_0, \nu_0) \dots (r_n, \nu_n)(r_{n+1}, \epsilon)$ where r_{n+1} is the right-hand side of the unique $\langle q, \mathit{label}(s, \mathit{orig}(e)) \rangle$-rule. Then, the operation *c-child*(e, i) is realized by the following algorithm: first apply *down*$_i$ to e, then repeatedly apply *pop* as long as possible, and then repeatedly apply *expand* as long as possible. The non-erasure of N ensures that this yields a well-formed list; in the last step, when *expand* cannot be applied to $e = \dots (r_n, \nu_n)$, *label*(r_n, ν_n) is obviously not in $Q \times X$ and by non-erasure is not in Y, hence it is in $\Delta \cup C$. Since the length of a well-formed list is bounded by $|s|$ and *pop* (and *expand*, respectively) always decreases (increases) the length of the list by one, each of them are executed at most $|s|$ times in the calculation of *c-child*. Hence, *c-child* runs in polynomial time with respect to $|s|$. Similarly, the representation of the root of $\tau_N(s)$ is obtained in polynomial time by repeatedly applying *expand* as long as possible to $e_0 = (r_0, \epsilon)$ where r_0 denotes the right-hand side of the unique $\langle q_0, \mathit{label}(s, \epsilon) \rangle$-rule. Note that a similar list representation is used in the proof of Theorem 3 in [4].

```
MATCH $(e, v)$
1:  while $label(e) = +$ do
2:      $e \leftarrow$ $c\text{-}child(e, k)$ where k = 1 or 2,
                        nondeterministically chosen
3:  if $c\text{-}label(e) \neq label(v)$ then
4:      return false
5:  else if $rank(label(v)) = 0$ then
6:      return true
7:  else
8:      for $i = 1$ to $rank(label(v))$ do
9:          if not MATCH$(c\text{-}child(e, i), child(v, i))$ then
10:             return false
11:     return true
```

Fig. 1. Matching Algorithm

Matching Algorithm with NP Time Complexity Let $t \in T_\Delta$. Figure 1 shows the nondeterministic algorithm MATCH that decides, given a well-formed list e and a node v of t, whether the set of trees represented by the choice tree at e contains the subtree of t rooted at v. The operations *c-label* and *c-child* are defined as above. The operations *label*, *rank*, and *child* are basic tree operations, assumed to run in polynomial time with respect to $|t|$. If we apply MATCH to the representations of the root nodes of $\tau_N(s)$ and $v = \epsilon$, we can decide whether $(s, t) \in \tau_M$. Since this is the standard top-down recursive comparison of two trees, the correctness of the algorithm should be clear.

In each nondeterministic computation, MATCH is called once for each node of t. In each call, the while-loop iterates at most $c|s|$ times for a constant c. This is due to non-erasure, i.e., for every Y-node in right-hand sides there exists a non-$+$ ancestor node. If we once *expand* a list for obtaining *c-child*, we never see Y-nodes in right-hand sides (thus never *pop*) before seeing some Δ-node. Thus, during the while-loop, the sequence of applied operations must be: first *pop*'s and *down*'s are applied, and then *expand* is applied (if any), and after that no *pop* is applied, i.e., the only operations applied are *expand* or *down*. In other words, it has to be in the regular set

$(\textit{pop}|\textit{down})^*(\textit{expand}|\textit{down})^*$. However, since the length of a well-formed list is at most $|s|$, we can continuously *pop* without *expand*ing at most $|s|$ times, and the same for *expand* without *pop*ping. Also, the numbers of continuous *down*'s are bounded by the height of the right-hand sides of the rules of N. Thus, the loop terminates after at most $2 \cdot (1 +$ the maximum height of right-hand sides of $N) \cdot |s|$ iterations. Altogether, the total running time is polynomial in $|s| + |t|$.

Linear Space Complexity The MATCH algorithm takes $O((|s| + \log |t|)|t|)$ space if naively implemented, because in the worst case the depth of recursion is $O(|t|)$ and we have to remember e (which costs $O(|s|)$ space) and v ($O(\log(|t|))$ space at least, depending on the tree node representation) in each step of the recursion. However, note that the lists of nodes share common prefixes! Suppose the root node is represented by $(r_0, \nu_0)(r_1, \nu_1)(r_2, \nu_2)(r_3, \nu_3)$ and its child node is obtained by applying *down*$_1$, *pop*, and *expand*. Then the child node is of the form $(r_0, \nu_0)(r_1, \nu_1)(r_2, \nu'_2)(r'_3, \nu'_3)$, which shares the first two elements with the root node representation. We show that if we store lists of nodes with common prefixes maximally shared, then, in the case of path-linear mtts, their space consumption becomes $O(|s|+|t|)$. The idea of sharing lists resembles the proof of context-sensitivity of indexed languages [1].

We encode a list of well-formed lists as a tree, written in parenthesized notation on the tape. For example, the list of three lists $[\rho_1\rho_2\rho_3, \rho_1\rho_2\rho_4, \rho_1\rho_5\rho_6]$ is encoded as $\rho_1(\rho_2(\rho_3, \rho_4), \rho_5(\rho_6))$. Since the number of parentheses is $\leq 2n$ and that of commas is $\leq n$ where n denotes the number of nodes, the size of this representation is $O(n)$. When the function MATCH is recursively called, we add the current e to the end of the list. The addition is represented as an addition to the rightmost path. As an example, let $e = \rho_1\rho_5\rho_7\rho_8$. The common prefix $\rho_1\rho_5$ with the current rightmost path $\rho_1\rho_5\rho_6$ is shared, and the suffix $\rho_7\rho_8$ is added as the rightmost child of the ρ_5-node. Then, we have a new tree $\rho_1(\rho_2(\rho_3, \rho_4), \rho_5(\rho_6, \rho_7(\rho_8)))$. Removal of the last list, which happens when MATCH returns, is the reverse operation of addition; the rightmost leaf and its ancestors that have only one descendant leaf are removed. Note that, since by definition a well-formed list cannot be a prefix of any other well-formed lists, each well-formed list always corresponds to a leaf node of the tree. It is straightforward to implement these two operations in linear space and in polynomial time.

Let us consider what happens if we apply this encoding to the output of a *path-linear* mtt. In the algorithm MATCH we only proceed downwards in the trees, i.e., the parameter e' to the recursive calls is always obtained by applying *c-child* several times to the previous parameter e. Thus, the lists $[e_0, e_1, \ldots, e_n]$ of node representations we have to store during the recursive computation always satisfy the relation $e_j \in \textit{c-child}^+(e_i)$ for every $i < j$. Let $e = (r_0, \nu_0) \ldots (r_m, \nu_m)$ and $e' = (r'_0, \nu'_0) \ldots (r'_m, \nu'_m)$ be proper prefixes of different elements in the same list satisfying the condition (here we assume that e is taken from the element preceding the one where e' is taken). Then, $\textit{orig}(e) = \textit{orig}(e')$ only if $e = e'$. This can be proved by contradiction. Suppose $\textit{orig}(e) = \textit{orig}(e')$ and $e \neq e'$, and the j-th elements are the first difference between e and e'. Recall that e' is a prefix of a well-formed list obtained by repeatedly applying *c-child* to another well-formed list, of which e is a prefix. Then it must be the case that $r_j = r'_j$ (by definition of *expand*, r_j and r'_j are uniquely determined from $(r_0, \nu_0) \ldots (r_{j-1}, \nu_{j-1})$ and $(r'_0, \nu'_0) \ldots (r'_{j-1}, \nu'_{j-1})$, which are equal) and ν_j is a proper prefix of ν'_j. However, due to path-linearity, the input variable at ν_j and ν'_j must be different, which contradicts $\textit{orig}(e) = \textit{orig}(e')$. Therefore, we can associate a unique node in $\textit{pos}(s)$ with each proper prefix of the lists, which means that the number of distinct proper prefixes is at most $|s|$. Similarly, it can be shown that adding only to the rightmost path is sufficient for maximally sharing all common prefixes. Suppose not, then there must be in the list three nodes of the forms $e_1 = e.(r, \nu).e'_1$, $e_2 = e.(r, \nu').e'_2$, and $e_3 = e.(r, \nu).e'_3$ with $\nu \neq \nu'$ in this order. Note that if this happened, then the prefix $e.(r, \nu)$ would not be shared by the rightmost addition. However, $e_2 \in \textit{c-child}^+(e_1)$ implies that ν is a proper prefix of ν', and by $e_3 \in \textit{c-child}^+(e_2)$, ν' is a proper prefix of ν, which is a contradiction. Hence, the number of nodes except leaves in the tree encoding equals the number of distinct proper prefixes, which is at most $|s|$. We can bound the number of leaves by $|t|$, the maximum depth of the recursion. So, the size of the tree encoding of a list of nodes is $O(|s| + |t|)$. We can easily remember the whole list of v's in $O(|t|)$

space. Since in the lists $[v_1, \ldots, v_n]$, v_{i+1} is always a child node of v_i, we only need to remember the child number for each node. For example, the list $[\epsilon, \epsilon.2, \epsilon.2.1]$ can be encoded as $[\epsilon, 2, 1]$. Thus, we only need $\leq$ *height*(t) many numbers, each of which is between 1 and the maximal rank of symbols in Δ, which is a constant.

Theorem 1. *Let M be a canonical, non-erasing, and path-linear mttcf. There effectively exists a non-deterministic Turing Machine which, given any s and t as input, determines whether or not $(s, t) \in \tau_M$ in $O(|s| + |t|)$ space and in polynomial time with respect to $|s| + |t|$.*

4 Bounding the Size of Intermediate Trees

As explained in the Introduction, the key idea for obtaining linear-size complexity for compositions of mtts is to bound the size of all intermediate input trees, and this is achieved by putting the mtts into "non-deleting" forms. In the same way as for total deterministic mtts [17], we classify the "deletion" in mtts into three categories – *erasing*, *input-deletion*, and *skipping* (a similar classification without erasing, which is a specific use of parameters, is also used in the case of nondeterministic tts [3]). The resolution of each kind of deletion, however, requires several new techniques and considerations compared to previous work, due to the interaction of nondeterminism and parameters. In the rest of this paper, we first explain how we eliminate each kind of deletion, and then show the main results.

4.1 Erasing

We first consider "erasing" rules – rules of the form $\langle q, \sigma(\cdots)\rangle(y_1, \ldots, y_m) \to y_i$, as defined in Section 3. An application of such a rule consumes one input σ-node without producing any new output symbols; hence it is deleting a part of the input. Note that if the rank of σ is non-zero, then a rule as above is at the same time also input-deleting, which is handled in Section 4.2. In the case of total deterministic mtts, "non-erasing" is a normal form, i.e., for every total deterministic mtt there is an equivalent one without erasing rules. Unfortunately, we could not find such a normal form for nondeterministic mtts with OI semantics. Note that for OI context-free tree grammars (essentially mtts without input: think of $\langle q, x_i\rangle$ as a nonterminal N_q, or equivalently, think of macro grammars [13] or indexed grammars [1], with trees instead of strings in right-hand sides), it has been shown [16] that there is *no* non-erasing normal form: erasing grammars are strictly more powerful than non-erasing ones. To see where the difficulty arises, let us consider the following example of a deterministic mtt and the input tree $\mathtt{a}(\mathtt{b}, \mathtt{b})$:

$$\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\langle q_3, x_2\rangle(\mathtt{B}, \mathtt{C})) \qquad \langle q_2, \mathtt{b}\rangle(y_1) \to \mathtt{A}(y_1, y_1) \qquad \langle q_3, \mathtt{b}\rangle(y_1, y_2) \to y_1.$$

The $\langle q_3, \mathtt{b}\rangle$-rule is erasing. The basic idea of obtaining the non-erasing normal form for total deterministic mtts is to apply all erasing rules inline where they are called in a right-hand side. That is, we remove the erasing rule and modify the q_1 rule to $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\mathtt{B})$ (plus a look-ahead check "$x_2 = \mathtt{b}$"). This approach does not work properly under OI-nondeterminism. Let us suppose the case when the $\langle q_3, \mathtt{b}\rangle$-rules are nondeterministic as follows:

$$\langle q_3, \mathtt{b}\rangle(y_1, y_2) \to y_1 \qquad \langle q_3, \mathtt{b}\rangle(y_1, y_2) \to y_2 \qquad \langle q_3, \mathtt{b}\rangle(y_1, y_2) \to \mathtt{A}(y_1, y_2).$$

Note that the q_2 rule duplicates its argument $\langle q_3, x_2\rangle(\mathtt{B}, \mathtt{C})$ *before* calling q_3, and evaluates the two copies independently. Thus, $\langle q_1, \mathtt{a}(\mathtt{b}, \mathtt{b})\rangle\!\downarrow$ contains all the nine trees $\mathtt{A}(t_1, t_2)$ with $t_1, t_2 \in \{\mathtt{B}, \mathtt{C}, \mathtt{A}(\mathtt{B}, \mathtt{C})\}$. However, the inline application of erasing rules now gives: $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\mathtt{B})$, $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\mathtt{C})$, and $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\mathtt{A}(\mathtt{B}, \mathtt{C}))$, which implies copying *after* evaluation of the q_3 call. So, in order to perform the expansion correctly, we need some way to preserve the nondeterministic choice after the expansion. For this purpose, we move from normal mtts to *mtts with choice and failure*. The example above can be represented by an mttcf rule $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(+(\mathtt{B}, +(\mathtt{C}, \mathtt{A}(\mathtt{B}, \mathtt{C}))))$, for instance. We will show that every mtt can be simulated by a non-erasing mttcf.

Lemma 2. *Let M be a mtt. There exists effectively a linear tt E and a canonical mttcf M' such that M' is non-erasing and $\tau_E\,;\tau_{M'} = \tau_M$. Path-linearity is preserved from M to M'.*

Proof. The idea is, we first predict all erasing beforehand and annotate each input node by the information of erasing, by using a preprocessing linear tt. Then we replace all erasing state calls (e.g., $\langle q, x_1\rangle(u_1)$ with the rule $\langle q, \ldots\rangle(y_1) \to y_1$) in the right-hand sides of rules with the result of the erasing call (e.g., u_1). Note that we have to deal with nondeterminism. Suppose we have two rules $\langle q, \sigma\rangle(y_1, y_2) \to y_1$ and $\langle q, \sigma\rangle(y_1, y_2) \to y_2$ and a state call $\langle q, x_1\rangle(u_1, u_2)$ in a right-hand side. In order to preserve the nondeterminism, we replace the state call by $+(u_1, u_2)$.

Let $M = (Q, \Sigma, \Delta, q_0, R)$. We define E to be a nondeterministic linear tt with the set of states $P = [Q \to 2^{\{1,\ldots,n\}}] \cup \{p_0\}$ (functions from Q to $2^{\{1,\ldots,n\}}$ where n is the maximum rank of the states of Q, and one distinct state p_0, which is the initial state), the input alphabet Σ, the output alphabet $\Sigma_p = \{(\sigma, p_1, \ldots, p_k)^{(k)} \mid \sigma^{(k)} \in \Sigma, p_i \in P\}$, and the following rules for every $\sigma^{(k)} \in \Sigma$ and $p_1, \ldots, p_k \in [Q \to 2^{\{1,\ldots,n\}}]$: $\langle p, \sigma(x_1, \ldots, x_k)\rangle \to (\sigma, p_1, \ldots, p_k)(\langle p_1, x_1\rangle, \ldots, \langle p_k, x_k\rangle)$ where $p \in \{p_0, (q \mapsto \bigcup\{f(r) \mid \langle q, \ldots\rangle(\ldots) \to r \in R\})\}$ with f recursively defined as follows: $f(y_i) = \{i\}$, $f(\delta(\ldots)) = \emptyset$, and $f(\langle q', x_j\rangle(r_1, \ldots, r_m)) = \bigcup\{f(r_i) \mid i \in p_j(q')\}$. The transducer E modifies the label $\sigma^{(k)}$ of each input node into the form $(\sigma^{(k)}, p_1, \ldots, p_k)$. The annotated information p_i intuitively means "if a state q of M is applied to the i-th child of the node, it will erase and return directly the e-th parameter for $e \in p_i(q)$". If $p_i(q) = \emptyset$ then no erasing will happen. The rule of E is naturally understood if it is read from right to left, as a bottom-up translation. Formally speaking, the following claim holds. It is easily proved by induction on the structure of s.

Claim. (1) For each $s \in T_\Sigma$ and $q \in Q^{(m)}$, there is a unique $p \in P \setminus \{p_0\}$ such that $\langle p, s\rangle{\downarrow_E} \neq \emptyset$, and $e \in p(q)$ if and only if $y_e \in \langle q, s\rangle(y_1, \ldots, y_m){\downarrow_M}$. (2) Let us denote by $[s]$ such p determined by s. The output $s' \in \tau_E(s)$ is unique. For $b \in \mathit{pos}(s) = \mathit{pos}(s')$, $\mathit{label}(s', b) = (\mathit{label}(s, b), [s|_{b.1}], \ldots, [s|_{b.k}])$.

We next define a non-erasing mttcf, using the annotation added by E. Let $M' = (Q, \Sigma_p, \Delta, q_0, R')$ with $R' = \{\langle q, (\sigma, p_1, \ldots, p_k)(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r' \mid r \in R_{q,\sigma}, r' \in \mathit{ne}(r), r' \notin Y\}$ where the set $\mathit{ne}(r)$ is defined inductively by

$$\begin{aligned}
\mathit{ne}(y_j) &= \{y_j\}\\
\mathit{ne}(\delta(r_1, \ldots, r_l)) &= \{\delta(r'_1, \ldots, r'_l) \mid r'_i \in \mathit{ne}(r_i)\}\\
\mathit{ne}(\langle q', x_j\rangle(r_1, \ldots, r_l)) &= \bigcup\{\mathit{ne}(r_i) \mid i \in p_j(q')\} \cup \{\langle q', x_j\rangle(\mathit{nep}(r_1), \ldots, \mathit{nep}(r_l))\},
\end{aligned}$$

and *nep* defined as follows: $\mathit{nep}(y_j) = y_j$, $\mathit{nep}(\delta(r_1, \ldots, r_l)) = \delta(\mathit{nep}(r_1), \ldots, \mathit{nep}(r_l))$, and $\mathit{nep}(\langle q', x_j\rangle(r_1, \ldots, r_l)) = +(u_1, +(u_2, \ldots, +(u_z, \theta)\cdots))$ where $\{u_1, \ldots, u_z\} = \mathit{ne}(\langle q', x_j\rangle(r_1, \ldots, r_l))$. Intuitively, *ne* adds rules by replacing each top-level state calls with its argument if the state call is erasing according to the annotation p_j's. The other function *nep* does essentially the same thing for non top-level positions, but by replacing erasing state calls with $+$ choices instead of adding rules to preserve the OI-nondeterminism. It should be clear from the definition that M' is canonical and non-erasing. Since *ne* only returns a set of subtrees of a right-hand side, *ne* and *nep* never add any new nesting among state calls, and thus M' is path-linear if M is.

The correctness of this construction is proved by induction on the structure of the input tree s, by showing that if $u_i{\downarrow_M} = u'_i{\downarrow_{M'}}$ then $\langle q, s\rangle(u_1, \ldots, u_k){\downarrow_M} = \langle q, \tau_E(s)\rangle(u'_1, \ldots, u'_k){\downarrow_{M'}} \cup \bigcup\{u_i{\downarrow_M} \mid y_i \in \langle q, s\rangle(y_1, \ldots, y_k){\downarrow_M}\}$. Applying this to the initial state q_0 proves the equation $\tau_M = \tau_E\,;\tau_{M'}$. □

4.2 Input-Deletion

The second kind of deletion we investigate is "input-deletion". For instance, if there is the rule $\langle q_0, \mathtt{a}(x_1, x_2)\rangle \to \mathtt{A}(\langle q_0, x_2\rangle)$ for the initial state q_0 and the input is of the form $\mathtt{a}(t_1, t_2)$, then the subtree t_1 is never used for the output calculation. Although total deterministic mtts can be made *nondeleting* (i.e., to always traverse all subtrees of *every* input tree) by preprocessing with a deleting linear tt [17], it becomes more difficult for nondeterministic mtts. This is because of the nondeterminism, which means that there can be more than one possible computation for a single input tree, and we cannot avoid the situation that one of the computations traverses all subtrees while others do not. Consider the input $\mathtt{a}(\mathtt{b}, \mathtt{c}(t_1, t_2))$ and the following set of rules:

$$\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\langle q_3, x_2\rangle) \qquad \langle q_3, \mathtt{c}(x_1, x_2)\rangle \to \langle q_4, x_1\rangle$$
$$\langle q_2, \mathtt{b}\rangle(y_1) \to \mathtt{A}(y_1, y_1) \qquad \langle q_3, \mathtt{c}(x_1, x_2)\rangle \to \langle q_4, x_2\rangle$$

Note that the state call for q_3 is duplicated by OI semantics; even though the mtt is linear. There are three possibilities with respect to input-deletion: either t_1 is deleted (the case all duplicated q_3 calls choose the second q_3-rule), t_2 is deleted (the case all choose the first rule), or no deletion occurs. We can still construct a linear tt that does preliminarily deletion, in such a way that it nondeterministically returns $\mathtt{a}(\mathtt{b}, \mathtt{c}_2(t_2))$, $\mathtt{a}(\mathtt{b}, \mathtt{c}_1(t_1))$, or $\mathtt{a}(\mathtt{b}, \mathtt{c}_{12}(t_1, t_2))$ (the subscript on $\mathtt{c}$ identifies the non-deleted children). We can also modify the mtt as follows

$$\langle q_3, \mathtt{c}_1(x_1)\rangle \to \langle q_4, x_1\rangle \qquad \langle q_3, \mathtt{c}_{12}(x_1, x_2)\rangle \to \langle q_4, x_1\rangle$$
$$\langle q_3, \mathtt{c}_2(x_1)\rangle \to \langle q_4, x_1\rangle \qquad \langle q_3, \mathtt{c}_{12}(x_1, x_2)\rangle \to \langle q_4, x_2\rangle$$

in which rules using the "deleted" input subtrees are removed. Then, for the former two "deleted" instances of the input trees, the mtt is successfully non-input-deleting. But sadly, this mtt still may delete for the last instance of the input tree, when all duplicated $\langle q_3, \mathtt{c}_{12}\rangle$ calls choose the same rule. The point is, under nondeterminism, we cannot argue the input-deleting property of each *transducer*. Rather, we can only argue whether each *computation* is input-deleting or not. This is a weaker version of the nondeletion condition used for total deterministic mtts, but it is sufficient for our purpose.

In order to speak more formally, here we define the notion of *computation tree* (following the method of [3], but extending it to deal with accumulating parameters). For any finite set P, we define the ranked alphabet $\underline{P} = \{\underline{p}^{(1)} \mid p \in P\}$. Let $M = (Q, \Sigma, \Delta, q_0, R)$ be an mttcf and $s \in T_\Sigma$. The set $\mathit{COMP}(M, s)$ is the set of trees $\mathit{comp}\langle q_0, \underline{\epsilon}\rangle{\downarrow} \subseteq T_{\Delta \cup \underline{\mathit{pos}(s)}}$ called *computation trees* (or sometimes, simply *computations*). The derivation $\mathit{comp}\langle q_0, \underline{\epsilon}\rangle{\downarrow}$ is carried out under the following set of rewriting rules with outside-in derivation: $+(u_1, u_2) \to u_1$, $+(u_1, u_2) \to u_2$, and $\mathit{comp}\langle q, \underline{\nu}\rangle(\vec{y}) \to f_\nu(r)$ for $q \in Q$, $\nu \in \mathit{pos}(s)$, $r \in R_{q,\mathit{label}(s,p)}$ where f_ν is inductively defined as follows:

$$\begin{aligned} f_\nu(y_i) &= y_i \\ f_\nu(\delta(r_1, \ldots, r_k)) &= \underline{\nu}(\delta(f_\nu(r_1), \cdots, f_\nu(r_k)) \\ f_\nu(\langle q', x_j\rangle(r_1, \ldots, r_k)) &= \mathit{comp}\langle q', \underline{\nu.j}\rangle(f_\nu(r_1), \cdots, f_\nu(r_k))). \end{aligned}$$

Intuitively, $\mathit{COMP}(M, s)$ is the set of trees $\langle q_0, s\rangle{\downarrow}$ where the parent of each Δ-node is a monadic node labeled by the position in the input tree s that generated the Δ-node. For example, the output tree $\underline{\epsilon}(\alpha(\underline{\epsilon.1}(\beta), \underline{\epsilon.2}(\gamma(\underline{\epsilon}(\delta)))))$ means that the α and δ nodes are generated at the root node of the input tree, and the β and γ nodes are generated at the first and the second child of the root node, respectively. Let *delpos* be the translation that removes all $\underline{\nu} \in \underline{\mathit{pos}(s)}$ nodes. It is easily proved by induction on the number of derivation steps that $\mathit{delpos}(\mathit{COMP}(M, s)) = \langle q_0, s\rangle{\downarrow}_M$, i.e., if we remove all $\underline{\mathit{pos}(s)}$ nodes from a computation tree, we obtain an output tree of the original mtt.

We say that a computation tree u is *non-input-deleting* if for every leaf position $\nu \in \mathit{pos}(s)$, there is at least one node in u labeled by $\underline{\nu}$. Note that the rewriting rules of *comp* corresponding to erasing rules do not generate any $\underline{\mathit{pos}(s)}$ node. Thus, non-input-deletion implies that not only some state is applied to every leaf, but also a *non-erasing* rule of some state must be applied.

Lemma 3. *Let M be a canonical non-erasing mttcf. There effectively exists a linear tt I and a canonical non-erasing mttcf M' such that $\tau_M = \tau_I\,;\tau_{M'}$, and for every input-output pair $(s,t) \in \tau_M$, there exists a tree s' and a computation tree $u \in \mathit{COMP}(M', s')$ such that $(s, s') \in \tau_I$, $t = \mathit{delpos}(u)$, and u is non-input-deleting. Also, M' is path-linear if M is.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, R)$. We define I as $(\{d\}, \Sigma, \Sigma', d, U)$ where $\Sigma' = \{(\sigma, i_1, \ldots, i_m)^{(k)} \mid \sigma^{(k)} \in \Sigma, 1 \leq i_1 < \cdots < i_m \leq k\}$ and

$$U = \{\langle d, \sigma(x_1, \ldots, x_k)\rangle \to (\sigma, i_1, \ldots, i_m)(\langle d, x_{i_1}\rangle, \ldots, \langle d, x_{i_m}\rangle) \mid (\sigma, i_1, \ldots, i_m) \in \Sigma'\}.$$

The transducer I reads the input tree and nondeterministically deletes subtrees while encoding the numbers of the undeleted subtrees in the current label. We define the mttcf M' as $(Q, \Sigma', \Delta, q_0, R')$ where

$$\begin{aligned} R' = \{&\langle q, (\sigma, i_1, \ldots, i_m)(x_1, \ldots, x_m)\rangle(\vec{y}) \to r' \\ &\mid r \in R_{q,\sigma} \text{ such that for all top-level calls } \langle q', x_p\rangle \text{ in } r, p \in \{i_1, \ldots, i_m\}, \text{ and } r' \text{ is obtained by} \\ &\quad \text{replacing } \langle q', x_{i_j}\rangle \text{ in } r \text{ with } \langle q', x_j\rangle \text{ and } \langle q', x_p\rangle \text{ with } \theta \text{ for } p \notin \{i_1, \ldots, i_m\}\}. \end{aligned}$$

The transducer M' has basically the same rules as M, except that state calls on 'deleted' children are replaced by θ (or, if it is at the top-level then the rule is removed, to preserve canonicity). It should be easy to see that M' is canonical and non-erasing, and preserves the path-linearity of M.

The correctness of this construction is proved as follows. Note that there is the natural one-to-one correspondence between the set of $\langle q, \sigma\rangle$-rules of M and the set of $\langle q, (\sigma, i_1, \ldots, i_m)\rangle$-rules of M'. First, we can prove (1) $\tau_{M'}(\tau_I(s)) \subseteq \tau_M(s)$ by induction on the number of derivation steps in M', showing that for each derivation step in M' we can always apply the corresponding rewriting rule in M and obtain the corresponding (i.e., differs only at state calls on 'deleted' nodes that are replaced with θ) sentential form, which proves that we can obtain the same final output as $\tau_I\,; \tau_{M'}$ by τ_M. Next, we show that (2) for any $u \in \mathit{COMP}(M, s)$, there exists an equivalent non-input-deleting computation. Let s' be the minimal substructure of s that contains all nodes and their ancestors contained in u. Then $s' \in \tau_I(s)$ assuming that s' is appropriately relabeled from Σ to Σ' as the transducer I does, and we can similarly prove that we can obtain a computation $u' \in \mathit{COMP}(M', s')$ corresponding to u by induction on the number of derivation steps (here, non-erasure assures that we can always choose the 'corresponding' rule; if a state call $\mathit{comp}\langle q, \underline{p}\rangle$ is rewritten in the derivation of M, then a node labeled $\underline{p'}$ must be generated for some descendant $\overline{p'}$ of p, and thus the node corresponding to p' and its ancestor $\overline{p}$ are kept not removed in s'). Then, by the construction we have $\mathit{delpos}(u) = \mathit{delpos}(u')$, and since s' is the minimal substructure of s that contains all nodes occurred in u, all leaf nodes of s' occur in u', which means that u' is non-input-deleting. Note that (2) implies $\tau_M(s) \subseteq \tau_{M'}(\tau_I(s))$. Therefore, together with (1), we have $\tau_I\,; \tau_{M'} = \tau_M$ as desired. □

4.3 Skipping

The third and last kind of deletion is "skipping". A computation tree u is *skipping* if there is a node $\nu \in \mathit{pos}(s)$ labeled by a rank-1 symbol such that no node in u is labeled $\underline{\nu}$. For a canonical, non-erasing, and path-linear mttcf, skipping is caused by either one of the following two forms of rules. One type is of the form $\langle q, \sigma(x_1)\rangle(y_1, \ldots, y_m) \to \langle q', x_1\rangle(u_1, \ldots, u_v)$ where $u_i \in T_{Y \cup C}$, and such rules are called *skipping*. The others are rules which are not skipping but are of the form $\langle q, \sigma(x_1)\rangle(y_1, \ldots, y_m) \to \langle q', x_1\rangle(u_1, \ldots, u_v)$ where $u_i \in T_{\Delta \cup Y \cup C}$, and such rules are called *quasi-skipping*. Note that, since the mttcf is path-linear, there are no nested state calls in right-hand sides of rules for input symbols of rank 1. Also note that if the root node of the right-hand side of a rule is not a state call, then it must be a Δ-node since the mttcf is canonical and non-erasing. So an application of such a rule generates a Δ-node and thus a $\underline{\nu} \in \overline{\mathit{pos}(s)}$ node for the current input node. Therefore, it is sufficient to consider only skipping and quasi-skipping rules.

Quasi-skipping rules may cause skipping computations due to parameter deletion: for example, consider the quasi-skipping rule $\langle q, \sigma(x_1)\rangle(y_1) \to \langle q', x_1\rangle(\delta(y_1))$; if there is a q'-rule with a right-hand side not using y_1, then the σ-node may be skipped. For total deterministic mtts [17], there is a "parameter non-deleting" normal form, i.e., every total deterministic mtt is equivalent to one that uses all parameters in the right-hand sides of its rules, and thus only skipping rules (without choice nodes) were considered there. Unfortunately, as for non-erasure, we could not find such a normal form for nondeterministic mtts. Instead, we add some auxiliary skipping rules to mttcfs, so that we only need to consider skipping rules. Note that quasi-skipping rules cause skipping computations only when parameters are deleted. The idea is, if a parameter in some rule is never used for a computation, then replacing the parameter by a failure symbol θ does not change the translation, and moreover, such replacement changes a quasi-skipping rule into a skipping rule. Thus we may assume that all skipping computations are caused by skipping rules, and hence we can straightforwardly extend the proofs for total deterministic mtts [17] and nondeterministic tts [3].

Lemma 4. *Let M be an canonical, non-erasing, and path-linear mttcf. There exists effectively a linear tt S and a canonical, non-erasing, and path-linear mttcf M' such that (1) $\tau_S\,;\tau_{M'} = \tau_M$ and (2) for every input tree s and non-input-deleting computation tree $u \in \mathit{COMP}(M, s)$, there exists a tree s' and a computation tree u' such that $s' \in \tau_S(s)$, $u' \in \mathit{COMP}(M', s')$, $\mathit{delpos}(u') = \mathit{delpos}(u)$, and u' is both non-input-deleting and non-skipping.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, R)$. We define $N = (Q, \Sigma, \Delta, q_0, R \cup \overline{R})$ with:

$$\begin{aligned}\overline{R} =&\{\langle q, \sigma(x_1)\rangle(\vec{y}) \to r' \mid q \in Q, \sigma \in \Sigma^{(1)}, r \in R_{q,\sigma}, r \text{ is quasi-skipping},\\ &\quad \text{and } r' \text{ is obtained by replacing all subtrees of } r \text{ of the form } \delta(\cdots), \delta \in \Delta \text{ by } \theta\}.\end{aligned}$$

Obviously $\tau_M \subseteq \tau_N$, and it should be also clear that $\tau_N \subseteq \tau_M$, because in each derivation of N, we can replace every application of $\overline{R}$ rules by the corresponding rules in R.

Furthermore, we can similarly prove that for any non-input-deleting computation u of $\mathit{COMP}(N, s)$ and U the set of rank-1 nodes of s that are skipped in u, there is a derivation that derives u and does not apply quasi-skipping rules to any $p \in U$. Suppose a quasi-skipping rule ρ is applied to a node $p \in U$ in a derivation of u. Then, since all Δ-nodes in ρ are skipped in u (that means, they never come to top-level position during the derivation), we can always replace the application with the corresponding skipping rule in $\overline{R}$ (recall that it is obtained by replacing all Δ-nodes by θ) without changing the final output computation u. Thus, for N, without loss of generality we may assume that all skipping computation are caused by skipping rules.

We define S as $(H, \Sigma, \Sigma \times H, h_0, U)$ where $H = \bigcup_m [Q^{(m)} \to \bigcup_n \mathcal{P}(Q^{(n)} \times \mathcal{P}(\{1, \ldots, m\})^n)]$ with $\mathcal{P}$ denoting power set, $h_0 = q \mapsto \{(q, \{1\}, \{2\}, \ldots, \{\mathit{rank}(q)\})\}$, and

$$\begin{aligned}U = &\{\langle h, \sigma(x_1, \ldots, x_k)\rangle \to (\sigma, h)(\langle h_0, x_1\rangle, \ldots, \langle h_0, x_k\rangle) \mid \sigma \in \Sigma, h \in H\}\\ &\cup \{\langle h, \sigma(x_1)\rangle \to \langle h'_\sigma, x_1\rangle \mid \sigma \in \Sigma^{(1)}, h \in H\}\}\end{aligned}$$

where h'_σ is the function

$$\begin{aligned}h'_\sigma = q \mapsto \{(q'', &f(r_1, \vec{\iota}), \ldots, f(r_l, \vec{\iota})\}) \mid\\ &(q', \iota_1, \ldots, \iota_n) \in h(q),\\ &\langle q', \sigma(x_1)\rangle(y_1, \ldots, y_n) \to \langle q'', x_1\rangle(r_1, \ldots, r_l) \in R \cup \overline{R}, r_i \in T_{Y \cup C}\}\end{aligned}$$

with f inductively defined as: $f(y_j, \iota_1, \ldots, \iota_m) = \iota_j$, $f(\theta, \iota_1, \ldots, \iota_m) = \emptyset$, and $f(+(r_1, r_2), \iota_1, \ldots, \iota_m)$ $= f(r_1, \iota_1, \ldots, \iota_m) \cup f(r_2, \iota_1, \ldots, \iota_m)$.

The transducer S reads the input tree and nondeterministically deletes sequences of rank-1 nodes, while encoding the possible way to pass parameters by skipping rules. Such information on parameter passing is encoded as a function $h \in H$. For example, one possible output of S from the input $\mathtt{a}(\mathtt{b}(\mathtt{c}(\cdots)))$ is $(\mathtt{c}, h)(\cdots)$, in which a and b nodes are deleted and the information is encoded in h. Intuitive meaning of each h is, "if $(q', \iota_1, \ldots, \iota_n) \in h(q)$, then when a state q were applied to the root of the deleted sequence of rank-1 nodes with parameters $t_1, \ldots, t_m$, then N would have skipped the sequence, and reached a state q' with parameters $t_{i_1}, \ldots t_{i_n}, i_j \in \iota_j$ for $1 \leq j \leq n$". The initial state h_0 means that "no node was skipped so far".

We then define M' as $(Q, \Sigma \times H, \Delta, q_0, R')$ where

$$R' = \{\langle q, (\sigma, h)(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r[y_1/iset(\iota_1), \cdots, y_n/iset(\iota_n)] \\ \mid (q', \iota_1, \ldots, \iota_n) \in h(q), r \in R_{q',\sigma}\}$$

with $iset(\{i_1, \ldots, i_p\}) = +(y_{i_1}, +(y_{i_2}, \cdots, +(y_{i_p}, \theta) \cdots))$. Since M is canonical, non-erasing, and path-linear, clearly so is M'.

It should be easy to verify that the definition of M' and S follow the above intuition. The composition $\tau_S\,;\tau_{M'} = \tau_M$ and the existence of a non-skipping computation can be proved similarly as for Lemma 3. Namely, to construct a non-skipping computation u' from a computation $u \in \mathit{COMP}(M, s)$, we take the intermediate tree s' as the tree obtained from s by deleting all rank-1 nodes not contained in u. Then by induction on the number of derivation steps, we can construct an equivalent non-skipping computation $u' \in \mathit{COMP}(M', s')$. □

4.4 Counting the Number

Lemma 5. *Let $M = (Q, \Sigma, \Delta, q_0, R)$ be an mttcf, s an input tree, and u a non-input-deleting, non-skipping computation tree in COMP(M, s) with delpos$(u) = t$. Then $|s| \leq 2|t|$.*

Proof. Since u is non-input-deleting and non-skipping, for all nodes $\nu \in pos(s)$ of rank zero or one, there exists a node labeled $\underline{\nu}$ in u, and by definition of computation trees, its child node is labeled by a symbol in Δ. Thus, *leaves*(s) + *rank1nodes*$(s) \leq |t|$ where *leaves*(s) is the number of leaf nodes of s and *rank1nodes*(s) is the number of nodes of s labeled by rank-1 symbols. Since $|s| \leq 2 \times$ *leaves*(s) + *rank1nodes*(s) (this holds for any tree s), we have $|s| \leq 2|t|$ as desired. □

5 Complexity of Compositions of MTTs

Lemma 6. *Let $\mathcal{K} \in \{\mathrm{NSPACE}(n), \mathrm{NP}\}$ and F a class of $\mathcal{K}$ languages effectively closed under* LT. *Then* LMT(F) *and* T(F) *are also in $\mathcal{K}$.*

Proof. Let M be a linear mtt or a tt. Note that in both cases, M is path-linear. First, we make it non-erasing; by Lemma 2, there exist a linear tt E and a canonical, non-erasing, and path-linear mttcf M_1 such that $\tau_E\,;\tau_{M_1} = \tau_M$. Next, we make each computation non-input-deleting; by Lemma 3, there exist a linear tt I and a canonical, non-erasing, and path-linear mttcf M_2 such that $\tau_I\,;\tau_{M_2} = \tau_{M_1}$. For every $(s_1, t) \in \tau_{M_1}$, there is an intermediate tree s_2 and a non-input-deleting computation $u \in$ *COMP*(M_2, s_2) such that $(s_1, s_2) \in \tau_I$ and *delpos*$(u) = t$. Then, we make each computation non-skipping; by Lemma 4, there exist a linear tt S and a canonical, non-erasing, and path-linear mttcf M_3 such that $\tau_S\,;\tau_{M_3} = \tau_{M_2}$. For every non-input-deleting computation $u \in$ *COMP*(M_2, s_2), there is an intermediate tree s_3 and a non-input-deleting, non-skipping computation $u' \in$ *COMP*(M_3, s_3) such that $(s_2, s_3) \in \tau_S$ and *delpos*(u') = *delpos*(u). Altogether, we have $\tau_E\,;\tau_I\,;\tau_S\,;\tau_{M_3} = \tau_M$, and for every $(s, t) \in \tau_M$ there exists a tree s_3 such that $(s, s_3) \in \tau_E\,;\tau_I\,;\tau_S$ and a non-input-deleting, non-skipping computation $u' \in$ *COMP*(M_3, s_3) such that *delpos*$(u') = t$. By Lemma 5, $|s_3| \leq 2|t|$.

Let L be a language in F. To check whether $t \in \tau_M(L)$, we nondeterministically generate every tree s' of size $|s'| \leq 2|t|$ and for each of them, test whether $(s', t) \in \tau_{M_3}$ and $s' \in (\tau_E \,;\tau_I \,;\tau_S)(L)$. By Theorem 1, the former test can be done nondeterministically in $O(|s'| + |t|) = O(|t|)$ space and polynomial time with respect to $|t|$. By the assumption that F is closed under LT, the language $(\tau_E \,;\tau_I \,;\tau_S)(L)$ is also in $\mathcal{K}$. Thus the latter test is in complexity $\mathcal{K}$ with respect to $|s'| = O(|t|)$. $\square$

Note that, for T, the same decomposition as shown in the first paragraph of the proof of Lemma 6 is already known (Lemma 1 of [3]). And since by Theorem 1 of [3] the translation membership of a single tt is in DSPACE(n), for T, the above result also holds for $\mathcal{K} =$ DSPACE(n).

Lemma 7. *Let $\mathcal{K} \in \{$NSPACE(n), NP$\}$ and F a class of $\mathcal{K}$ languages effectively closed under* LT. *Then* MT(F) *is also in $\mathcal{K}$ and effectively closed under* LT.

Proof. The closure under LT immediately follows from the following known results: MT $=$ D$_\mathrm{t}$MT ; T (Corollary 6.12 of [12]), T ; LT $=$ D$_\mathrm{t}$QREL ; T (Lemma 2.11 of [9]), and D$_\mathrm{t}$MT ; D$_\mathrm{t}$QREL $\subseteq$ D$_\mathrm{t}$MT (Lemma 11 of [11]). By Lemma 2.11 of [9] and Theorem 2.9 of [8], T ; LT $\subseteq$ LT ; T, which implies that T(F) is also closed under LT. By the decomposition MT $=$ D$_\mathrm{t}$T ; LMT (page 138 of [12]), MT$(F) \subseteq$ LMT(T(F)). By applying Lemma 6 twice, LMT(T(F)) is in $\mathcal{K}$. $\square$

Theorem 8. MT*(REGT) $\subseteq$ NSPACE$(n) \cap$ NP-complete.

Proof. The class REGT is closed under LT (Propositions 16.5 and 20.2 of [15]) and is in NSPACE$(n) \cap$ NP (see, e.g., [15]). By induction on $k \geq 1$ it follows from Lemma 7 that MT^k(REGT) is in NSPACE(n) and NP. As noted in the Introduction, NP-hardness follows from [23] and the fact that the indexed languages, which are equivalent to the yields of context-free-tree languages under OI-derivation, are in MT^2(REGT). $\square$

Although we only have considered outside-in evaluation order up to here, the previous result holds for compositions of mtts in *inside-out* evaluation order. This is because MT$^*_\mathrm{IO}$ $=$ MT* by Theorem 7.3 of [12], where MT$_\mathrm{IO}$ denotes the class of translations realized by mtts in inside-out evaluation order.

Corollary 9. MT$^*_\mathrm{IO}$(REGT) $\subseteq$ NSPACE$(n) \cap$ NP-complete.

The *yield* translation, which translates a tree into its string of leaf labels from left to right (seen as a monadic tree), is in D$_\mathrm{t}$MT. Therefore the output string languages *yield*(MT*(REGT)) of mtts are also in the same complexity class as Theorem 8. Especially, this class contains the IO- and OI- hierarchies [6]. Note that the IO-hierarchy is in D$_\mathrm{t}$MT*(REGT) and hence in DSPACE(n) by Corollary 17 of [17]. Since the first level of the OI-hierarchy are the indexed languages [13] which are NP-complete [23], we obtain the following.

Corollary 10. *The OI-hierarchy is in* NSPACE$(n) \cap$ NP-complete.

Acknowledgment This work was partly supported by the Japan Society for the Promotion of Science.

References

1. A. V. Aho. Indexed grammars—an extension of context-free grammars. *J. ACM*, 15:647–671, 1968.
2. P. R. J. Asveld. Time and space complexity of inside-out macro languages. *Int. J. Comp. Math.*, 10:3–14, 1981.
3. B. S. Baker. Generalized syntax directed translation, tree transducers, and linear space. *SIAM J. Comp.*, 7:376–391, 1978.
4. G. Busatto, M. Lohrey, and S. Maneth. Efficient memory representation of XML document trees. *Inf. Syst.*, 33:456–474, 2008.
5. B. Courcelle. Monadic second-order definable graph transductions: A survey. *Th. Comp. Sc.*, 126:53–75, 1994.
6. W. Damm. The IO- and OI-hierarchies. *Th. Comp. Sc.*, 20:95–207, 1982.
7. F. Drewes and J. Engelfriet. Decidability of the finiteness of ranges of tree transductions. *Inf. and Comp.*, 145:1–50, 1998.
8. J. Engelfriet. Bottom-up and top-down tree transformations – a comparison. *Math. Sys. Th.*, 9:198–231, 1975.

9. J. Engelfriet. Top-down tree transducers with regular look-ahead. *Math. Sys. Th.*, 10:289–303, 1977.
10. J. Engelfriet. The complexity of languages generated by attribute grammars. *SIAM J. Comp.*, 15:70–86, 1986.
11. J. Engelfriet and S. Maneth. Output string languages of compositions of deterministic macro tree transducers. *J. Comp. Sys. Sci.*, 64:350–395, 2002.
12. J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comp. Sys. Sci.*, 31:71–146, 1985.
13. M. J. Fischer. *Grammars with Macro-Like Productions*. PhD thesis, Harvard University, Cambridge, 1968.
14. Z. Fülöp and H. Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. Springer-Verlag, 1998.
15. F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol 3: Beyond Words*, pages 1–68. Springer-Verlag, 1997.
16. B. Leguy. Grammars without erasing rules. the OI case. In *Trees in Algebra and Programming*, 1981.
17. S. Maneth. The complexity of compositions of deterministic tree transducers. In *FSTTCS*, 2002.
18. S. Maneth, A. Berlea, T. Perst, and H. Seidl. XML type checking with macro tree transducers. In *PODS*, 2005.
19. S. Maneth and G. Busatto. Tree transducers and tree compressions. In *FoSSaCS*, 2004.
20. T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *PODS*, 2000.
21. T. Perst and H. Seidl. Macro forest transducers. *Information Processing Letters*, 89:141–149, 2004.
22. W. C. Rounds. Mappings and grammars on trees. *Math. Sys. Th.*, 4:257–287, 1970.
23. W. C. Rounds. Complexity of recognition in intermediate-level languages. In *FOCS*, 1973.
24. J. W. Thatcher. Generalized^2 sequential machine maps. *J. Comp. Sys. Sci.*, 4:339–367, 1970.