IPSJ Transactions on Programming Vol. 2 No. 4 53–63 (Aug. 2009)

Regular Paper

Computing the Cost of Typechecking of Composition of Macro Tree Transducers

Keisuke Nakano[†1] and Sebastian Maneth[†2,†3]

Macro tree transducers are a classical formal model for structural-recursive tree transformation with accumulative parameters. They have recently been applied to model XML transformations and queries. Typechecking a tree transformation means checking whether all valid input trees are transformed into valid output trees, for the given regular tree languages of input and output trees. Typechecking macro tree transducers is generally based on inverse type inference, because of the advantageous property that inverse transformations effectively preserve regular tree languages. It is known that the time complexity of typechecking an n-fold composition of macro tree transducers is non-elementary. The cost of typechecking can be reduced if transducers in the composition have special properties, such as being deterministic or total, or having no accumulative parameters. In this paper, the impact of such properties on the cost of typechecking is investigated. Reductions in cost are achieved by applying composition and decomposition constructions to tree transducers. Even though these constructions are well-known, they have not yet been analyzed with respect to the precise sizes of the transducers involved. The results can directly be applied to typechecking XML transformations, because type formalisms for XML are captured by regular tree languages.

1. Introduction

Top-down tree transducers are a classical formal model of structural recursive tree transformation invented by Rounds[18] and Thatcher[20]. Originally, they were introduced to model syntax-directed compilation. Engelfriet and Vogler[5] extended them with accumulative parameters by means of the concept of macro grammars[8]. Macro tree transducers have recently attracted considerable attention as a fundamental model of XML transformations. Most models of XML transformations are realized by sequential composition of macro tree transducers[4),14),16)].

†1 The University of Electro-Communications
†2 National ICT Australia Ltd.
†3 University of New South Wales

Typechecking macro tree transducers has become an increasingly hot topic because it can directly be applied to typechecking XML transformations. Typechecking tree transducers means checking whether all valid input trees are transformed into valid output trees with respect to their types, which are given by the regular tree languages of input and output trees. Let L_{in} and L_{out} be input and output regular tree languages, respectively, and T a tree transformation. Typechecking T with respect to L_{in} and L_{out} ensures that $T(L_{in}) \subseteq L_{out}$. It is well known that $T(L_{in})$ is generally not captured by regular tree languages; hence, we cannot exactly infer the output type[19]. Therefore, we employ *inverse type inference* based on the facts that the statement for typechecking is equivalent to $T^{-1}({L_{out}}^c) \cap L_{in} = \emptyset$ with the complement language ${L_{out}}^c$ of L_{out}, and that the inverse transformation of macro tree transducers effectively preserves regular tree languages. Since regular tree languages are closed under complementation and intersection, and the emptiness of regular tree languages is decidable, the typechecking problem is decidable. The bottleneck in this method is to compute $T^{-1}(L)$ for a given regular tree language L. The size of a tree automaton for the inverse image is exponentially large, which would affect the time for computing the intersection and deciding emptiness. In particular, sequential composition of tree transformations will be harmful because we need to construct such exponentially large tree automata for each transformation. It is well known that the time complexity of typechecking an n-fold composition of macro tree transducers is non-elementary[4),16)] in the size of given tree automata.

The cost of inverse typechecking is characterized by the size of the constructed tree automaton for the inverse image. The cost of typechecking can be reduced according to the special properties of macro tree transducers in composition, such as being top-down tree transducers, deterministic or total, or taking OI- (call-by-name) or IO- (call-by-value) semantics. In this paper, we first give a general algorithm for the inverse type inference of a single macro tree transducer (in Section 3) to compute the upper bound for complexity and then investigate the impact of such properties of mtts on the cost of typechecking in Section 4. We have to be careful to look for pretexts that are often caused by composition laws on tree transducers. For example, we have two facts:

- We can always effectively construct a deterministic macro tree transducer equivalent to the composition of a deterministic macro tree transducer and a total deterministic top-down tree transducer.
- We can typecheck all deterministic mtts in exponential time to the size of output tree automaton.

These facts do not imply that the composition law that derives a single tree transducer from two tree transducers reduces the cost of typechecking. The constructed macro tree transducer is exponentially large. Therefore, the cost of typechecking gets much worse before the composition law is applied. In Section 4, we explore what kind of composition can improve the cost of typechecking by precisely computing the size of tree automaton. We also demonstrate that the *top-yield decomposition* 5) of macro tree transducers can reduce the cost of typechecking. Combining composition and decomposition leads to reducing the cost of typechecking for the composition of macro tree transducers.

2. Preliminaries

We denote the set of non-negative integers including 0 by $\mathbb{N}$, and sets $\{1,\ldots,n\}$ by $[n]$ for $n \in \mathbb{N}$, in particular, $[0] = \emptyset$. The power set of a set S is denoted by 2^S. The Cartesian product of two sets S and T, denoted by $S \times T$, is given by a set $\{\langle s,t\rangle \mid s \in S, t \in T\}$. The set of functions from set S to set T is denoted by $S \to T$.

Let Σ be a ranked alphabet, i.e., a finite set Σ together with a mapping that associates a natural number, the *rank*, to each $\sigma \in \Sigma$. We write $\Sigma^{(n)}$ to denote the set of symbols in Σ that have rank n, and write $\sigma^{(n)}$ to mean that σ's rank equals n. We denote the rank of a symbol σ by $\mathit{rank}(\sigma)$. For ranked alphabet Σ, the maximum rank of symbols in Σ is denoted by $\mathit{maxr}(\Sigma)$, i.e., $\mathit{maxr}(\Sigma) = \max\{\mathit{rank}(\sigma) \mid \sigma \in \Sigma\}$. The set of all ranked trees over Σ is denoted by $\mathcal{T}_\Sigma$. We fix the sets of input variables $X = \{x_1, x_2, \ldots\}$ and context parameters $Y = \{y_1, y_2, \ldots\}$, and assume that any ranked alphabet Σ is always disjoint with X and Y. In trees, the symbols of X and Y always appear at the leaves, i.e., they are all assumed to have rank 0. The set of ranked trees over Σ with a set V of variables is denoted by $\mathcal{T}_\Sigma(V)(= \mathcal{T}_{\Sigma\cup V})$. We denote by $t[v := s]$, the *substitution* of all occurrences of variable v in t by s.

A tree transformation from $\mathcal{T}_\Sigma$ to $\mathcal{T}_\Delta$ is represented by a function $\tau : \mathcal{T}_\Sigma \to 2^{\mathcal{T}_\Delta}$. Function τ is naturally extended to $\tau : 2^{\mathcal{T}_\Sigma} \to 2^{\mathcal{T}_\Delta}$ as $\tau(S) = \cup_{t\in S}\tau(t)$. We define the inverse function $\tau^{-1} : 2^{\mathcal{T}_\Delta} \to 2^{\mathcal{T}_\Sigma}$ by $\tau^{-1}(S) = \{t \in \mathcal{T}_\Sigma \mid \tau(t) \cap S \neq \emptyset\}$. We denote the composition of tree transformations f and g by $f \fatsemi g$, i.e., $(f \fatsemi g)(t) = g(f(t))$ for every tree t. For two classes F and G of tree transformations, $F \fatsemi G$ denotes the class of their composition, i.e., $F \fatsemi G = \{f \fatsemi g \mid f \in F,\ g \in G\}$.

A *deterministic bottom-up tree automaton* (dbta) is specified as $A = (B, \Sigma, \beta, B_f)$, where B is a finite set of states, Σ is a ranked alphabet of input symbols, $\beta : \Sigma^{(n)} \times B^n \to B$, $n \geq 0$ is the transition function, and $B_f \subset B$ is the set of final states. We extend transition function β to trees in $\mathcal{T}_\Sigma$ by recursively defining $\beta(\sigma(t_1,\ldots,t_k)) = \beta(\sigma, \beta(t_1),\ldots,\beta(t_k))$ for any tree $\sigma(t_1,\ldots,t_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $t_1,\ldots,t_k \in \mathcal{T}_\Sigma$. The language $L(A)$ recognized by A is then defined as $\{t \in \mathcal{T}_\Sigma \mid \beta(t) \in B_f\}$. We assume that readers are familiar with the basic notions of tree automata 2).

3. Typechecking Macro Tree Transducers

This section introduces macro tree transducers and explains their classification categorized by their special properties through examples. We also review results on the cost of typechecking macro tree transducers.

3.1 Macro Tree Transducers

Macro tree transducers 5) are a formal model of tree transformation with context parameters. They can be seen as functional programs based on structural recursion over input trees with accumulative parameters. A function can store arbitrary output trees in accumulative parameters, which may be used as part of the final output trees. For example, a macro tree transducer M_{exp} has the rules

$$\begin{array}{lcl} \mathit{main}(x) & \to & \mathit{exp}(x, \mathsf{Zero}), \\ \mathit{exp}(\mathsf{Succ}(x), y) & \to & \mathit{exp}(x, \mathit{exp}(x, y)), \text{ and} \\ \mathit{exp}(\mathsf{Zero}, y) & \to & \mathsf{Succ}(y), \end{array}$$

which give a definition of the exponentiation function with base 2, i.e., $\mathit{main}(\mathsf{Succ}^n(\mathsf{Zero})) = \mathsf{Succ}^{2^n}(\mathsf{Zero})$. These rules define a transformation from $\mathcal{T}_\Sigma$ to $\mathcal{T}_\Sigma$ with $\Sigma = \{\mathsf{Succ}^{(1)}, \mathsf{Zero}^{(0)}\}$. Function symbols are called *states* in the terminology of tree transducers. An expression on the right-hand side of the rule of $\mathit{main}(x)$ is an *axiom tree*, which represents an entry point in the functional program. The accumulative parameter y is called a *context parameter*. A macro tree transducer is called a *top-down tree transducer* [18),20)] if there are no context parameters.

There may exist more than one applicable rule, i.e., the tree transformation is *nondeterministic*. For example, the rules

$$\begin{array}{lll} \mathit{main}(x) & \rightarrow & \mathit{ndet}(x), \\ \mathit{ndet}(\mathsf{Succ}(x)) & \rightarrow & \mathsf{A}(\mathit{ndet}(x)), \\ \mathit{ndet}(\mathsf{Succ}(x)) & \rightarrow & \mathsf{B}(\mathit{ndet}(x)), \text{ and} \\ \mathit{ndet}(\mathsf{Zero}) & \rightarrow & \mathsf{E} \end{array}$$

specify a transformation from arbitrary natural numbers (represented by trees) to monadic trees of the same length whose nodes are either A or B. For example, there are four possible output trees for $\mathit{main}(\mathsf{Succ}(\mathsf{Succ}(\mathsf{Zero})))$, i.e., $\mathsf{A}(\mathsf{A}(\mathsf{E}))$, $\mathsf{A}(\mathsf{B}(\mathsf{E}))$, $\mathsf{B}(\mathsf{A}(\mathsf{E}))$, and $\mathsf{B}(\mathsf{B}(\mathsf{E}))$.

In contrast, no applicable rules may exist i.e., the tree transformation is *partial*. For example, the rules

$$\begin{array}{lll} \mathit{main}(x) & \rightarrow & \mathit{half}(x), \\ \mathit{half}(\mathsf{Zero}) & \rightarrow & \mathsf{Zero}, \\ \mathit{half}(\mathsf{Succ}(x)) & \rightarrow & \mathsf{Succ}(\mathit{aux}(x)), \text{ and} \\ \mathit{aux}(\mathsf{Succ}(x)) & \rightarrow & \mathit{half}(x) \end{array}$$

specify a partial function that outputs half of a number only if the input is even, i.e., for $n \geq 0$, $\mathit{main}(\mathsf{Succ}^{2n}(\mathsf{Zero}))$ is $\mathsf{Succ}^{n}(\mathsf{Zero})$ and $\mathit{main}(\mathsf{Succ}^{2n+1}(\mathsf{Zero}))$ is undefined.

Definition 3.1 A *macro tree transducer* (mtt) is a tuple $(Q, \Sigma, \Delta, E, R)$, where

- Q is a finite set of ranked states whose rank is more than 0,
- Σ and Δ are ranked alphabets with $Q \cap (\Sigma \cup \Delta) = \emptyset$, called the *input* and *output alphabets*, respectively,
- $E \subset \mathcal{T}_{Q\cup\Delta}(\{x\})$ is a set of *axiom trees* in which the first child of $q \in Q$ is x; for all $e \in E$, we write $\mathit{main}(x) \rightarrow e$ for readability,
- R is a set of rules such that $R = \cup_{q\in Q,\sigma\in\Sigma} R_{q,\sigma}$ with finite sets $R_{q,\sigma}$ of (q,σ)*-rules* of the form $q(\sigma(x_1,\ldots,x_n), y_1,\ldots,y_m) \rightarrow e$ with $q \in Q^{(m+1)}$, $\sigma \in \Sigma^{(n)}$, *input variables* $x_i (i \in [n])$, and *context parameters* $y_j (j \in [m])$; the right hand side expression e ranges over the following syntax:

  $e ::= q'(x_i, e_1, \ldots, e_{m'}) \mid y_j \mid \delta(e_1, \ldots, e_{n'})$

  with $q' \in Q^{(m'+1)}$, $\delta \in \Delta^{(n')}$, $i \in [n]$, and $j \in [m]$. We write $\mathit{rhs}_M(q,\sigma)$ for the set of right-hand side expressions of all (q,σ)-rules in M.

An mtt M is a *top-down tree transducer* (tdtt) if $Q = Q^{(1)}$ is a ranked alphabet of states, i.e., there are no context parameters. An mtt M is *deterministic* if E is a singleton and there exists at most one (q,σ)-rule for every $q \in Q$ and $\sigma \in \Sigma$. Otherwise, M is called *nondeterministic*. An mtt M is *total* if there exists at least one (q,σ)-rule for every $q \in Q$ and $\sigma \in \Sigma$. Otherwise, M is called *partial*. An mtt M is *linear* if the right-hand side of every rule is linear in the input variables X. An mtt M is *(input) non-deleting* if the right-hand side of every rule contains all input variables in X that occur on the left-hand side. □

Let us consider two different evaluation orders for mtts, outside-in (OI) and inside-out (IO). Transformation in OI-semantics corresponds to call-by-name evaluation, while transformation in IO-semantics corresponds to call-by-value evaluation. OI- and IO-semantics generally give different results (they coincide only if the mtt is total and deterministic as shown by Theorem 4.1 of Ref. 5)). For example, consider a nondeterministic mtt whose rules are

$$\begin{array}{lll} \mathit{main}(x) & \rightarrow & \mathit{dup}(x, \mathit{ndet}(x)), \\ \mathit{ndet}(\mathsf{Zero}) & \rightarrow & \mathsf{A}, \\ \mathit{ndet}(\mathsf{Zero}) & \rightarrow & \mathsf{B}, \text{ and} \\ \mathit{dup}(\mathsf{Zero}, y) & \rightarrow & \mathsf{C}(y, y). \end{array}$$

The computation of $\mathit{main}(\mathsf{Zero}) = \mathit{dup}(\mathsf{Zero}, \mathit{ndet}(\mathsf{Zero}))$ depends on the evaluation order. In OI-semantics, $\mathit{dup}(\ldots)$ is evaluated first. We obtain four results, $\mathsf{C}(\mathsf{A},\mathsf{A})$, $\mathsf{C}(\mathsf{A},\mathsf{B})$, $\mathsf{C}(\mathsf{B},\mathsf{A})$, and $\mathsf{C}(\mathsf{B},\mathsf{B})$, through $\mathsf{C}(\mathit{ndet}(\mathsf{Zero}), \mathit{ndet}(\mathsf{Zero}))$. In IO-semantics, $\mathit{ndet}(\ldots)$ is evaluated first, which gives two output trees $\mathsf{C}(\mathsf{A},\mathsf{A})$ and $\mathsf{C}(\mathsf{B},\mathsf{B})$ through $\mathit{dup}(\mathsf{Zero},\mathsf{A})$ and $\mathit{dup}(\mathsf{Zero},\mathsf{B})$, respectively.

The next example demonstrates that differences between OI and IO can be found even for deterministic mtts. Consider a deterministic mtt whose rules are

$$\begin{array}{lll} \mathit{main}(x) & \rightarrow & \mathit{const}(x, \mathit{part}(x)), \\ \mathit{const}(\mathsf{Zero}, y) & \rightarrow & \mathsf{A}, \text{ and} \\ \mathit{part}(\mathsf{Succ}(x)) & \rightarrow & \mathsf{B}. \end{array}$$

For $\mathit{main}(\mathsf{Zero}) = \mathit{const}(\mathsf{Zero}, \mathit{part}(\mathsf{Zero}))$, OI-semantics gives output A, while IO-semantics gives no result.

In the following formal semantics of mtts, we use power sets of trees to represent nondeterminism. In IO-semantics, the domains of context parameters do not have to be power sets so that two occurrences of the same context parameter should be synchronized.

Definition 3.2 Let $M = (Q, \Sigma, \Delta, E, R)$ be an mtt. The *OI-semantics* of a state $q^{(m+1)} \in Q$ is a function $[\![q]\!]_{\text{OI}} : \mathcal{T}_\Sigma \times (2^{\mathcal{T}_\Delta})^m \to 2^{\mathcal{T}_\Delta}$ defined by

$$[\![q]\!]_{\text{OI}}(\sigma(t_1, \dots, t_n), S_1, \dots, S_m) = \cup_{e \in rhs_M(q,\sigma)} \langle\!| e |\!\rangle_\rho$$

for $\sigma^{(n)} \in \Sigma$, where $\langle\!| \_ |\!\rangle_\rho$ denotes the evaluation of a right-hand side expression with respect to mapping $\rho = [x_i \mapsto t_i]_{i \in [n]} \cup [y_j \mapsto S_j]_{j \in [m]}$, which is defined by

$$\begin{aligned} \langle\!| q'(x_i, e_1, \dots, e_{m'}) |\!\rangle_\rho &= [\![q']\!]_{\text{OI}}(\rho(x_i), \langle\!| e_1 |\!\rangle_\rho, \dots, \langle\!| e_{m'} |\!\rangle_\rho), \\ \langle\!| y_j |\!\rangle_\rho &= \rho(y_j), \text{ and} \\ \langle\!| \delta(e_1, \dots, e_{n'}) |\!\rangle_\rho &= \{\delta(u_1, \dots, u_{n'}) \mid \forall i \in [n'] : u_i \in \langle\!| e_i |\!\rangle_\rho\}. \end{aligned}$$

The *OI-transformation induced by* M is the function $\tau_M : \mathcal{T}_\Sigma \to 2^{\mathcal{T}_\Delta}$ defined by $\tau_M(t) = \cup_{e \in E} \langle\!| e |\!\rangle_{[x \mapsto t]}$.

In contrast, the *IO-semantics* of a state $q^{(m+1)} \in Q$ is a function $[\![q]\!]_{\text{IO}} : \mathcal{T}_\Sigma \times (\mathcal{T}_\Delta)^m \to 2^{\mathcal{T}_\Delta}$ defined by

$$[\![q]\!]_{\text{IO}}(\sigma(t_1, \dots, t_n), s_1, \dots, s_m) = \cup_{e \in rhs_M(q,\sigma)} \langle\!| e |\!\rangle_\rho$$

for $\sigma^{(n)} \in \Sigma$, where $\langle\!| \_ |\!\rangle_\rho$ denotes the evaluation of a right-hand side expression with respect to mapping $\rho = [x_i \mapsto t_i]_{i \in [n]} \cup [y_j \mapsto s_j]_{j \in [m]}$, that is defined by

$$\begin{aligned} \langle\!| q'(x_i, e_1, \dots, e_{m'}) |\!\rangle_\rho &= \cup_{k \in [m'], u_k \in \langle\!| e_k |\!\rangle_\rho} [\![q']\!]_{\text{IO}}(\rho(x_i), u_1, \dots, u_{m'}), \\ \langle\!| y_j |\!\rangle_\rho &= \{\rho(y_j)\}, \text{ and} \\ \langle\!| \delta(e_1, \dots, e_{n'}) |\!\rangle_\rho &= \{\delta(u_1, \dots, u_{n'}) \mid \forall i \in [n'] : u_i \in \langle\!| e_i |\!\rangle_\rho, \}. \end{aligned}$$

The *IO-transformation induced by* M is the function $\tau_M : \mathcal{T}_\Sigma \to 2^{\mathcal{T}_\Delta}$ defined by $\tau_M(t) = \cup_{e \in E} \langle\!| e |\!\rangle_{[x \mapsto t]}$. □

The classes of all transformations realized by mtts using OI- and IO-transformations are denoted by MAC_{OI} and MAC_{IO}, respectively. The class of all transformations realized by tdtts is simply denoted by TOP because OI- and IO-semantics obviously coincide for tdtts. We use D, $_t$, D_t, and L as prefixes of classes for deterministic, total, total deterministic, and linear tree transducers, respectively. We use $_{\text{NONDEL}}$ as a postfix of classes for non-deleting tree transducers. For instance, $D\text{MAC}_{\text{OI}}$, $_t\text{MAC}_{\text{IO}}$, and $D_t\text{LTOP}_{\text{NONDEL}}$ correspond to the classes of partial deterministic mtts with OI-transformation, total nondeterministic mtts with IO-transformation, and total deterministic linear non-deleting tdtts, respectively. Since $D_t\text{MAC}_{\text{OI}} = D_t\text{MAC}_{\text{IO}}$ holds [5], we simply denote the class by $D_t\text{MAC}$.

3.2 Inverse Typechecking of Mtts

Typechecking an mtt M means verifying that $\tau_M(L_{in}) \subseteq L_{out}$ for two given regular tree languages L_{in} and L_{out} of input and output trees, respectively. We usually verify the equivalent statement $\tau_M^{-1}({L_{out}}^c) \cap L_{in} = \emptyset$ where L^c is the complement of L since $\tau_M(L_{in})$ generally exceeds regular tree languages (even context-free tree languages) while $\tau_M^{-1}({L_{out}}^c)$ is regular [5]. This is decidable because regular tree languages are closed under complementation and intersection, and their emptiness is decidable [2]. If two languages L_{in} and L_{out} are given as dbtas, the complexity of typechecking mainly depends on the complexity of computing $\tau_M^{-1}(L)$ for a regular tree language L; the latter is often called *inverse type inference*.

First, let us consider the OI-transformation of an mtt $M = (Q, \Sigma, \Delta, E, R)$. Let $A_{out^c} = (B, \Delta, \beta, B_f)$ be an output dbta for the complement of an output tree language. We construct a dbta A which exactly accepts all input trees t such that $\tau_M(t) \cap L(A_{out^c}) \neq \emptyset$, following the construction by Perst and Seidl [17]. The main idea is to assign the possible transition of states of A_{out^c} to every state in A according to the rules of M. Formally, dbta A is given by (D, Σ, κ, D_f), where

- Set D of states consists of all functions of type $Q^{(m+1)} \to (2^B)^m \to 2^B$; set D is finite because Q and B are finite.
- Transition $\kappa : \Sigma^{(n)} \times D^n \to D$ of states is defined as $\kappa(\sigma, d_1, \dots, d_n) = d$ with
$$d(q^{(m+1)})(B_1, \dots, B_m) = \cup_{e \in rhs_M(q,\sigma)} \langle\!| e |\!\rangle_\rho,$$
where $\langle\!| \_ |\!\rangle_\rho$ with respect to mapping $\rho = [x_i \mapsto d_i]_{i \in [n]} \cup [y_j \mapsto B_j]_{j \in [m]}$ is given by
$$\begin{aligned} \langle\!| q'(x_i, e_1, \dots, e_{m'}) |\!\rangle_\rho &= \rho(x_i)(q')(\langle\!| e_1 |\!\rangle_\rho, \dots, \langle\!| e_{m'} |\!\rangle_\rho), \\ \langle\!| y_j |\!\rangle_\rho &= \rho(y_j), \text{ and} \\ \langle\!| \delta(e_1, \dots, e_{n'}) |\!\rangle_\rho &= \{\beta(\delta, b_1, \dots, b_{n'}) \mid \forall i \in [n'] : b_i \in \langle\!| e_i |\!\rangle_\rho\}. \end{aligned}$$
- Set D_f of final states is given as $\{d \mid \langle\!| e |\!\rangle_{[x \mapsto d]} \cap B_f \neq \emptyset, e \in E\}$.

It can be shown in a similar way to that by Perst and Seidl [17] that $\kappa(t) \in D_f$ if and only if $\beta(\tau_M(t)) \in B_f$ for $t \in \mathcal{T}_\Sigma$. This implies that $\tau_M(L(A)) = L(A_{out^c})$.

The construction works for the inverse type inference of not only nondeterministic mtts with OI-transformation, i.e., MAC_{OI}, but also the other classes of mtts

$D = Q \to (2^B)^m \to 2^B$	for $\mathrm{MAC}_{\mathrm{OI}}$
$D = Q \to B^m \to 2^B$	for $\mathrm{MAC}_{\mathrm{IO}}$
$D = Q \to (B \uplus \{\bot\})^m \to (B \uplus \{\bot\})$	for $D\mathrm{MAC}_{\mathrm{OI}}$
$D = Q \to ((B^m \to B) \uplus \{\bot\})$	for $D\mathrm{MAC}_{\mathrm{IO}}$
$D = Q \to B^m \to B$	for $D_t\mathrm{MAC}$
$D = Q \to 2^B$	for TOP
$D = Q \to (B \uplus \{\bot\})$	for $D\mathrm{TOP}$
$D = Q \to B$	for $D_t\mathrm{TOP}$

Fig. 1 Domain of states of inferred input dbtas.

$2^{pN \cdot 2^{mN}}$	for $\mathrm{MAC}_{\mathrm{OI}}[p, m]$
$2^{pN^{m+1}}$	for $\mathrm{MAC}_{\mathrm{IO}}[p, m]$
$(N+1)^{p(N+1)^m}$	for $D\mathrm{MAC}_{\mathrm{OI}}[p, m]$
$(N^{N^m} + 1)^p$	for $D\mathrm{MAC}_{\mathrm{IO}}[p, m]$
N^{pN^m}	for $D_t\mathrm{MAC}[p, m]$
2^{pN}	for $\mathrm{TOP}[p]$
$(N+1)^p$	for $D\mathrm{TOP}[p]$
N^p	for $D_t\mathrm{TOP}[p]$

Fig. 2 Cost of inverse type inference of single mtt.

by replacing set D of states as shown in **Fig. 1**, where we fix the ranks of states of mtts with the maximum rank $m + 1 = maxr(Q)$, i.e., $D = Q \to (2^B)^m \to B$ for $\mathrm{MAC}_{\mathrm{OI}}$, to estimate the upper bound of the cost of type inference. The designated symbol $\bot$ stands for undefined outputs for partial functions. In order to apply the construction, set B may be used as a set of singleton sets of elements in B, i.e., is $\{\{b\} \mid b \in B\}$.

We give a rough explanation on the domain of states of inferred input dbtas in Fig. 1. For nondeterministic mtts with IO-transformation (in $\mathrm{MAC}_{\mathrm{IO}}$), context parameters should be bound to the same output tree as indicated by IO-semantics in Definition 3.2. Thus, we need singleton sets of output states for context parameters in D. For deterministic (partial) mtts with OI-transformation (in $D\mathrm{MAC}_{\mathrm{OI}}$), we do not consider power sets for outputs since all rules are deterministic. Instead, we have to consider the case where output is undefined because transformation is partial. In OI-semantics, context parameters can be undefined even if the output is defined. Thus, we need to consider $\bot$ for not only outputs but also all context parameters. For deterministic (partial) mtts with IO-transformation (in $D\mathrm{MAC}_{\mathrm{IO}}$), we do not have to consider each case where some context parameters are undefined. In IO-semantics, output is always undefined if one of the context parameters is undefined. Thus, all cases where output is undefined are treated as $\bot$ as well as the case with the absence of applicable rules. For total deterministic mtts (in $D_t\mathrm{MAC}$), we simply consider the case where all outputs and context parameters are singleton sets. For tdtts, the domain of states of inferred input dbtas are obtained as the special case of $m = 0$ in corresponding mtts. We consider either $\mathrm{MAC}_{\mathrm{OI}}$ or $\mathrm{MAC}_{\mathrm{IO}}$ for TOP, either $D\mathrm{MAC}_{\mathrm{OI}}$ or $D\mathrm{MAC}_{\mathrm{IO}}$ for $D\mathrm{TOP}$, and $D_t\mathrm{MAC}$ for $D_t\mathrm{TOP}$, where we obtain the same domain no matter which transformation is used, IO or OI.

3.3 Cost of Typechecking Mtts

We count the size of an inferred input dbta against a given output dbta to compute the cost of typechecking since size is the main factor for checking the emptiness of the intersection of the inferred dbta and the given input dbta. Usually, the size of a tree automaton is defined as its number of transitions. It suffices to count the number of states for dbta, which is cardinality of D in Fig. 1, because the constructed tree automaton is deterministic, i.e., all transitions are deterministic.

Let N be the number of states of a given output dbta, p the number of states of the given mtt, and $m + 1$ their maximum rank. **Figure 2** shows the cost of inverse type inference, where we use postfix $[p, m]$ to denote the class of tree transducers with the number p of states and the maximum number m of context parameters. The second number will be omitted for classes of tdtts because it is always equal to 0, e.g., $D_t\mathrm{TOP}[p]$. These costs are given as the cardinality of sets D in Fig. 1. Recall that linearity, the non-deleting property, and totality (with nondeterministic) do not affect the construction of an input dbta of mtts. We can ignore these restrictions under the context of the type inference of mtts. Therefore, the cost of type inference for $\mathrm{LTOP}_{\mathrm{NONDEL}}$ and ${}_t\mathrm{MAC}_{\mathrm{OI}}$ is the same as that for TOP and $\mathrm{MAC}_{\mathrm{OI}}$, respectively, as long as we employ the construction of the input dbta shown in Section 3.2. We do not discuss other constructions in this paper even though they may reduce the cost.

Inverse type inference for a trivial automaton that accepts all trees in $\mathcal{T}_\Delta$ computes the domain of the transducer. Thus, an emptiness test of its domain automaton solves emptiness of the transducer. The costs of the complexity of the emptiness test are computed by applying $N = 1$ to the formulae in Fig. 2.

4. Cost of Typechecking Composition of Mtts

It is well-known that mtts have at most an exponential height increase (Theorem 3.2[5]). This implies that the composition of two mtts is strictly more expressive than an mtt. For example, $M_{exp} \mathbin{;} M_{exp}$ has a *double* exponential height increase, where M_{exp} is the mtt presented in Section 3.1. If one or both of the mtts in the composition are restricted to particular classes, then the composition can be expressed by a single mtt. In this section, we review the results of the composition of two mtts and compare the cost of type inference between equivalent classes of transformations. Composition improves the cost in some cases, while decomposition does in others.

4.1 On Composition of Two Tdtts

Nondeterministic tdtts are not closed under composition[3),20)], while total deterministic tdtts are[18),20)]. Baker[1)] investigated what kind of restrictions are required of two tdtts so that their composition can be realized by a single tdtt. **Figure 3** shows her results on the composition of two tdtts except for compositions (6) and (7), which are immediately derived from (5). All constructions of a single tdtt from two tdtts are done by the coupling states of two tdtts ("product construction"). Therefore, the number of states of the synthesized tdtt is obtained by multiplying the numbers of states of two tdtts. Baker presented more results on the composition of tdtts that were obtained by combining results where the composition of linear tree transducers was linear or where the composition of one state tree transducers was one state.

Let us compare the cost of type inference for transformations on both sides of each composition shown in Fig. 3. We compute the cost of inverse type inference for a given number N of states of the output dbta. Consider composition (1). The input dbta of the second tdtt on the left-hand side, which can also be the output dbta for the first tdtt, has N^{p_2}-many states because the second tdtt is

$$D_t\mathrm{TOP}[p_1] \mathbin{;} D_t\mathrm{TOP}[p_2] \subseteq D_t\mathrm{TOP}[p_1p_2] \tag{1}$$
$$D_t\mathrm{TOP}[p_1] \mathbin{;} \mathrm{TOP}[p_2] \subseteq \mathrm{TOP}[p_1p_2] \tag{2}$$
$$_t\mathrm{TOP}[p_1] \mathbin{;} \mathrm{LTOP}[p_2] \subseteq \mathrm{TOP}[p_1p_2] \tag{3}$$
$$D\mathrm{TOP}[p_1] \mathbin{;} \mathrm{TOP}_{\mathrm{NONDEL}}[p_2] \subseteq \mathrm{TOP}[p_1p_2] \tag{4}$$
$$\mathrm{TOP}[p_1] \mathbin{;} \mathrm{LTOP}_{\mathrm{NONDEL}}[p_2] \subseteq \mathrm{TOP}[p_1p_2] \tag{5}$$
$$\mathrm{TOP}[p_1] \mathbin{;} D\mathrm{LTOP}_{\mathrm{NONDEL}}[p_2] \subset \mathrm{TOP}[p_1p_2] \tag{6}$$
$$\mathrm{TOP}[p_1] \mathbin{;} D_t\mathrm{LTOP}_{\mathrm{NONDEL}}[p_2] \subset \mathrm{TOP}[p_1p_2] \tag{7}$$

Fig. 3 Composition of two tdtts.

total deterministic. Thus, the number of states of the inferred input dbta is $(N^{p_2})^{p_1} = N^{p_1p_2}$. This coincides with the number of states of the input dbta inferred for the right-hand side transformation of the composition. Therefore, composition (1) does not change the cost of type inference by composition. We find that composition (2) similarly also does not change the cost of type inference.

Consider composition (3). The number of states of the input dbta is $2^{p_1N'}$ on the left-hand side, where $N' = 2^{p_2N}$ is the number of states of the input dbta for the second tdtt. Hence, the number is $2^{p_1 \cdot 2^{p_2N}}$. On the other hand, the cost of type inference of transformation in $\mathrm{TOP}[p_1p_2]$ is $2^{p_1p_2N}$. Note that $p_1, p_2, N > 0$. This composition decreases the cost with a reduction in its complexity. Compositions (5), (6), and (7) also reduce the cost of type inference in a similar way.

For composition (4), the costs of type inference for the left- and right-hand sides are $(2^{p_2N} + 1)^{p_1}$ and $2^{p_1p_2N}$, respectively. Hence, this composition reduces the cost.

In summary, compositions (3), (4), (5), (6), and (7) reduce the cost of type inference. They even reduce the exponential height of the cost except for composition (4).

4.2 On Composition of Mtt and Tdtt

Engelfriet and Vogler[5)] investigated many combinations of two classes of mtts whose composition belonged to a single class of mtts, in particular where either one of them was a tdtt. To precisely compute the cost of type inference, we have to focus on the proofs of their results because we cannot apply the same construction, which is different from Baker's results on the composition of two tdtts.

According to Corollary 4.10 and Theorem 4.12 in Ref. 5), the composition of a total deterministic mtt and a total deterministic tdtt (in any order) can be realized by a single (total deterministic) mtt. We have

$$D_t\mathrm{MAC}[p_1, m_1] \mathbin{;} D_t\mathrm{TOP}[p_2] \subseteq D_t\mathrm{MAC}[p_1p_2, m_1p_2] \text{ and} \quad (8)$$

$$D_t\mathrm{TOP}[p_1] \mathbin{;} D_t\mathrm{MAC}[p_2, m_2] \subseteq D_t\mathrm{MAC}[p_1p_2, m_2]. \quad (9)$$

For composition (8), the cost of type inference on the left-hand side is $N'^{p_1 N'^{m_1}}$ with $N' = N^{p_2}$; hence, $N^{p_1p_2N^{m_1p_2}}$. Therefore, this composition does not change the cost of type inference. We easily see that composition (9) also does not change the cost because we obtain $N^{p_1p_2N^{m_2}}$ for both sides.

Engelfriet and Vogler presented three cases where the first mtt was total deterministic other than composition (8). Since we can use the same construction for these three, we have

$$D_t\mathrm{MAC}[p_1, m_1] \mathbin{;} D\mathrm{TOP}[p_2] \subseteq D\mathrm{MAC}_{\mathrm{OI}}[p_1p_2, m_1p_2], \quad (10)$$

$$D_t\mathrm{MAC}[p_1, m_1] \mathbin{;} {}_t\mathrm{TOP}[p_2] \subseteq {}_t\mathrm{MAC}_{\mathrm{OI}}[p_1p_2, m_1p_2], \text{ and} \quad (11)$$

$$D_t\mathrm{MAC}[p_1, m_1] \mathbin{;} \mathrm{TOP}[p_2] \subseteq \mathrm{MAC}_{\mathrm{OI}}[p_1p_2, m_1p_2]. \quad (12)$$

It is easy to check that these compositions do not change the cost of type inference. The costs are $(N+1)^{p_1p_2(N+1)^{m_1p_2}}$ for composition (10) and $2^{p_1p_2N\cdot 2^{m_1p_2N}}$ for compositions (11) and (12).

For the case where the first mtt of the composition is partial deterministic in either $D\mathrm{MAC}_{\mathrm{IO}}$ or $D\mathrm{TOP}$, Engelfriet and Vogler presented several results. Consider one of these, $D\mathrm{TOP} \mathbin{;} D_t\mathrm{MAC} = D\mathrm{MAC}_{\mathrm{IO}}$. It is not simple to compute the cost for this composition because we cannot directly use the same construction for mtts as in composition (9). Their proof of this composition was based on the decomposition $D\mathrm{TOP}[p] \subseteq \mathrm{DTFTA}[2^p] \mathbin{;} D_t\mathrm{TOP}[p]$ of Theorem 3.1 in Ref. 6), where $\mathrm{DTFTA}[p']$ denotes the class of deterministic top-down tree automata with p'-many states. A tree automaton is used to make the partial tdtt total in order to apply the same construction of mtts as in composition (9). Therefore,

$$\begin{aligned}
&D\mathrm{TOP}[p_1] \mathbin{;} D_t\mathrm{MAC}[p_2, m_2] \\
&\subseteq \mathrm{DTFTA}[2^{p_1}] \mathbin{;} D_t\mathrm{TOP}[p_1] \mathbin{;} D_t\mathrm{MAC}[p_2, m_2] \\
&\subseteq \mathrm{DTFTA}[2^{p_1}] \mathbin{;} D_t\mathrm{MAC}[p_1p_2, m_2] \\
&\subseteq D\mathrm{MAC}_{\mathrm{IO}}[p_1p_2 + 2^{p_1} + r, \max\{m_2, s+1\}] \quad (13)
\end{aligned}$$

with the number r of rules in $D_t\mathrm{MAC}[p_1p_2, m_2]$ in the second last line and the cardinality s of the input alphabet in the next line, where we apply Theorem 5.21 from Ref. 5) in the last step, which demonstrates the closure property of mtts on regular look-ahead (see Theorem 5.19 in Ref. 5)). It is obvious that the composition worsens the cost of type inference. Similarly, we find that all the following compositions, in which the first mtt is in either $D\mathrm{MAC}_{\mathrm{IO}}$ or $D\mathrm{TOP}$, increase the cost of type inference because they always require their domain to be extracted using DTFTA (with an exponential number of states) to obtain a single mtt:

$$D\mathrm{MAC}_{\mathrm{IO}} \mathbin{;} D_t\mathrm{TOP} = D\mathrm{MAC}_{\mathrm{IO}}, \quad (14)$$

$$D\mathrm{MAC}_{\mathrm{IO}} \mathbin{;} D\mathrm{TOP} = D\mathrm{MAC}_{\mathrm{OI}}, \quad (15)$$

$$D\mathrm{MAC}_{\mathrm{IO}} \mathbin{;} \mathrm{TOP} = \mathrm{MAC}_{\mathrm{OI}}, \quad (16)$$

$$D\mathrm{TOP} \mathbin{;} D\mathrm{MAC}_{\mathrm{IO}} = D\mathrm{MAC}_{\mathrm{IO}}, \quad (17)$$

$$D\mathrm{TOP} \mathbin{;} D\mathrm{MAC}_{\mathrm{OI}} = D\mathrm{MAC}_{\mathrm{OI}}, \text{ and} \quad (18)$$

$$D\mathrm{TOP} \mathbin{;} \mathrm{MAC}_{\mathrm{OI}} = \mathrm{MAC}_{\mathrm{OI}}. \quad (19)$$

For the case where the first mtt is in $D\mathrm{MAC}_{\mathrm{OI}}$, Engelfriet and Vogler presented two results, $D\mathrm{MAC}_{\mathrm{OI}} \mathbin{;} D\mathrm{TOP} = D\mathrm{MAC}_{\mathrm{OI}}$ and $D\mathrm{MAC}_{\mathrm{OI}} \mathbin{;} \mathrm{TOP} = \mathrm{MAC}_{\mathrm{OI}}$. We have to extract the domain of $D\mathrm{MAC}_{\mathrm{OI}}$ to make it total in a similar way to $D\mathrm{TOP}$. The domain is realized by a nondeterministic finite tree automaton whose class is denoted by $\mathrm{FTA}[p]$ with the number p of its states. According to the results by Theorems 5.22 and 6.18 of Ref. 5) and Theorem 3.1 of Ref. 6). we have $D\mathrm{MAC}_{\mathrm{OI}}[p, m] \subseteq \mathrm{FTA}[2^{p\cdot 2^m}] \mathbin{;} D_t\mathrm{MAC}[p, m]$ (the number of states of FTA can also be derived from the cost for $D\mathrm{MAC}_{\mathrm{OI}}$ in Fig. 2 by fixing $N = 1$ to obtain a domain automaton).

Therefore,

$$\begin{aligned}
&D\mathrm{MAC}_{\mathrm{OI}}[p_1, m_1] \mathbin{;} D\mathrm{TOP}[p_2] \\
&\subseteq \mathrm{FTA}[2^{p_1\cdot 2^{m_1}}] \mathbin{;} D_t\mathrm{MAC}[p_1, m_1] \mathbin{;} D\mathrm{TOP}[p_2] \\
&\subseteq D_t\mathrm{MAC}[p_1 + 2^{p_1\cdot 2^{m_1}}, \max\{m_1, 2\}] \mathbin{;} D\mathrm{TOP}[p_2] \\
&\subseteq D\mathrm{MAC}_{\mathrm{OI}}[p_2p_1 + p_2 \cdot 2^{p_1\cdot 2^{m_1}}, \max\{m_1p_2, 2p_2\}],
\end{aligned}$$

where we apply Theorem 4.21 of Ref. 5) in the second last step.

In summary, the composition laws (8), (9), (10), (11), and (12) do not change the cost of type inference, while the other laws for mtts and tdtts increase the cost.

4.3 On Composition of Two Mtts

Voigtländer and Kühnemann[22] presented results on the composition of two restricted mtts based on the composition of attributed tree transducers[11)–13)]. According to Construction 5.1 in their paper, we have

$$D_t\mathrm{MAC}_{\mathrm{NC}}[p_1, m_1] \mathbin{;} D_t\mathrm{MAC}_{\mathrm{WSU}}[p_2, m_2] = D_t\mathrm{MAC}[p_1p_2(m_1m_2+1), p_2(m_1+m_2)],$$

where NC and WSU correspond to restrictions of mtts, *non-copying* and *weakly single-use*, respectively. The non-copying mtt does not have more than one occurrence of the same context parameter on the right-hand side of the rules. The weakly single-use mtt $M = (Q, \Sigma, \Delta, E, R)$ does not have more than one occurrence of $q'(x_i, \dots)$ with the same state $q' \in Q$ and the same input variable x_i on all right-hand sides of (q, σ)-rules with $q \in Q$ for every $\sigma \in \Sigma$.

The cost of type inference for the left-hand side is $N^{p_1p_2N^{m_2+m_1p_2N^{m_2}}}$ using the cost in Fig. 2 as two mtts in $D_t\mathrm{MAC}$. On the other hand, the cost for the right-hand side is $N^{p_1p_2(m_1m_2+1)N^{p_2(m_1+m_2)}}$. It is easy to see that this composition may reduce complexity with respect to N.

4.4 Decomposition of Mtts

It is known that a single mtt is decomposed into a tdtt and *yield* mapping[5),10)]. We call this *top-yield decomposition*. Yield mapping, a total deterministic linear mtt with a single state, transforms a tree, in which the operation of tree substitution is expressed symbolically, into the tree it denotes. We extend the result so that the tdtt will be non-deleting because the cost of type inference may be improved by using compositions (4), (5), (6), and (7) when the tdtt follows another tdtt.

Theorem 4.1 For a total deterministic mtt M, there exist a total deterministic non-deleting tdtt M_1 and a total deterministic linear mtt M_2 with a single state such that $\tau_M = \tau_{M_1} \mathbin{;} \tau_{M_2}$.

Proof. Let $M = (Q, \Sigma, \Delta, E, R)$ be a total deterministic mtt. We construct a total deterministic non-deleting tdtt $M_1 = (P, \Sigma, \Gamma, E', R')$ where

- $P = \{p_q^{(1)} \mid q \in Q\}$,
- $\Gamma = \{\gamma_\delta^{(0)} \mid \delta \in \Delta\} \cup \{c_k^{(k)} \mid k \in [\mathit{maxr}(Q \cup \Sigma) + 1]\} \cup \{\pi_j \mid j \in [\mathit{maxr}(Q) - 1]\} \cup \{\mathit{del}^{(2)}\}$, and
- $E' = \{\langle\!\lbrace e \rbrace\!\rangle \mid e \in E\}$ and $R' = \{q(\sigma(x_1, \dots, x_n)) \to \phi_{Z,q}(\langle\!\lbrace e \rbrace\!\rangle) \mid e \in \mathit{rhs}_M(q, \sigma).Z$ is the set of deleted input variables in $e.\}$ with auxiliary functions $\langle\!\lbrace \_ \rbrace\!\rangle$ and $\phi_{Z,q}$ defined by

$$\begin{array}{lcl}
\phi_{\{z_1,\dots,z_k\},q}(e) & = & \mathit{del}(p_q(z_1), \dots, \mathit{del}(p_q(z_k), e) \cdots), \\
\langle\!\lbrace q'(x_i, e_1, \dots, e_{m'}) \rbrace\!\rangle & = & c_{m'+1}(p_{q'}(x_i), \langle\!\lbrace e_1 \rbrace\!\rangle, \dots, \langle\!\lbrace e_{m'} \rbrace\!\rangle), \\
\langle\!\lbrace y_j \rbrace\!\rangle & = & \pi_j, \text{ and} \\
\langle\!\lbrace \delta(e_1, \dots, e_n) \rbrace\!\rangle & = & c_{n+1}(\gamma_\delta, \langle\!\lbrace e_1 \rbrace\!\rangle, \dots, \langle\!\lbrace e_n \rbrace\!\rangle).
\end{array}$$

Next, we construct a total deterministic mtt $M_2 = (\{q^{(m+1)}\}, \Gamma, \Delta, \{q(x, \delta', \dots, \delta')\}, R'')$ where $m = \mathit{maxr}(Q \cup \Delta)$ and R'' consists of the following rules using some $\delta' \in \Delta^{(0)}$ as a dummy symbol.

- For $\delta \in \Delta^{(n)}$, R'' contains $\mathit{in}(\gamma_\delta) \to \delta'$ and $q(\gamma_\delta, y_1, \dots, y_m) \to \delta(y_1, \dots, y_n)$; note that $n = \mathit{rank}(\delta) \le m$,
- For c_k with $k \in [m+1]$, R'' contains
  $\mathit{in}(c_k(x_1, \dots, x_k)) \to q(x_1, \mathit{in}(x_2), \dots, \mathit{in}(x_k), \delta', \dots, \delta')$
  and
  $q(c_k(x_1, \dots, x_k), y_1, \dots, y_m) \to$
  $q(x_1, q(x_2, y_1, \dots, y_m), \dots, q(x_k, y_1, \dots, y_m), \delta', \dots, \delta')$.
- For π_j with $j \in [\mathit{maxr}(Q) - 1]$, R'' contains $\mathit{in}(\pi_j) \to \delta'$ and $q(\pi_j, y_1, \dots, y_m) \to y_j$.
- For $\mathit{del}^{(2)}$, R'' contains $\mathit{in}(\mathit{del}(x_1, x_2)) \to \mathit{in}(x_2)$ and $q(\mathit{del}(x_1, x_2), y_1, \dots, y_m) \to q(x_2, y_1, \dots, y_m)$.

Since all symbols *del* introduced by M_1 are appropriately eliminated by M_2, we can claim $\tau_{M_1} \mathbin{;} \tau_{M_2} = \tau_M$ by following the proof of the original decomposition theorem of Corollary 5.9 of Ref. 5) and Theorem 4.37 of Ref. 10). □

Let us call M_2 in the proof above $M(\Gamma)$. We denote the class of all yield transformations by YIELD, which is given by $\{\tau_{M(\Gamma)} \mid \Gamma = \mathcal{A}(\Delta, m, n)$ with ranked alphabetΔ and $m, n \in \mathbb{N}\}$, where $\mathcal{A}(\Delta, m, n) = \Delta \cup \{c_k^{(k)} \mid k \in [m+1]\} \cup \{\pi_j \mid j \in [n]\} \cup \{\mathit{del}^{(2)}\}$. We write YIELD$[m]$ for the subset of YIELD obtained by fixing the second parameter of $\mathcal{A}$ to m. The cost of type inference for the transformations in YIELD$[m]$ is N^{N^m} because YIELD$[m] \subset D_t\mathrm{MAC}[1, m]$. According to Engelfriet and Vogler[5] and Theorem 4.1, we have

$$D_t\mathrm{MAC}[p, m] \subseteq D_t\mathrm{TOP}_{\mathrm{NONDEL}}[p] \mathbin{;} \mathrm{YIELD}[m], \tag{20}$$

$$D\text{MAC}_{\text{OI}}[p,m] \subseteq D_t\text{TOP}_{\text{NONDEL}}[p] \mathbin{;} \text{YIELD}[m] \mathbin{;} \text{EMPTY}, \quad (21)$$
$$D\text{MAC}_{\text{IO}}[p,m] \subseteq D\text{TOP}_{\text{NONDEL}}[p+1] \mathbin{;} \text{YIELD}[m], \quad (22)$$
$$\text{MAC}_{\text{OI}}[p,m] \subseteq D_t\text{TOP}_{\text{NONDEL}}[p] \mathbin{;} \text{YIELD}[m] \mathbin{;} \text{SET}, \text{ and} \quad (23)$$
$$\text{MAC}_{\text{IO}}[p,m] \subseteq \text{TOP}_{\text{RES}}[p+\#(\Delta)] \mathbin{;} \text{YIELD}[m], \quad (24)$$

where SET and EMPTY are special kinds of tdtts introduced by Engelfriet and Vogler[5]. Here, we assume that number m is greater than the maximum rank of input alphabet Σ for the sake of simplicity. We may have to write $D_t\text{MAC}[p,m] = D_t\text{TOP}_{\text{NONDEL}}[p] \mathbin{;} \text{YIELD}[\max\{m, maxr(\Sigma)\}]$ for decomposition (20) if we want to precisely compute the cost. We do not show concrete definitions of SET and EMPTY. What we have to know to compute the cost of type inference is the cost for each transformation, i.e.,

$$2^N \quad \text{for SET, and}$$
$$N+1 \quad \text{for EMPTY,}$$

which are obtained from the facts of $\text{SET} \subset \text{TOP}[1]$ and $\text{EMPTY} \subset D\text{TOP}[1]$, respectively. Note that the tdtt has $(p+1)$-many states for decomposition (22). The tdtt may have to have one more (total) state because of the partiality of state p_q in the definition of ϕ. Since in the decomposition in the proof of Theorem 4.1 we have assumed the totality of the original mtt, there are no problems with the construction of the tdtt. If the original mtt is partial, there may be no rule for p_q, which makes outputs undefined. Hence, we should use a "total" state for p_q. The identity state p_{id} would be reasonable for an additional state, which has rules with the form

$$p_{id}(\delta(x_1,\ldots,x_n)) \to \delta(p_{id}(x_1),\ldots,p_{id}(x_1))$$

for every output symbol δ.

For decomposition (24), the number of states of the tdtt depends on the number $\#(\Delta)$ of output symbols of the original mtt, following the proof of Theorem 5.12 of Ref. 5). TOP_{RES} stands for the class of *restricted* tdtts, in which the right-hand side of every rule is either an output tree (with no state call $q(x_i)$) or a tree whose subtree with children should have a state call as its first child.

Linearity is always preserved for all of these decompositions, e.g.,

$$D_t\text{LMAC}[p,m] \subseteq D_t\text{LTOP}_{\text{NONDEL}}[p] \mathbin{;} \text{YIELD}[m]$$

for decomposition (20).

Now, let us investigate the cost reduction for each decomposition. For decomposition (20), the cost of type inference for the right-hand side is N'^p with $N' = N^{N^m}$, i.e., N^{pN^m}. This is the same cost as that for the left-hand side. Therefore, this decomposition does not change the cost of type inference. It is also easy to find that neither decompositions (21) nor (23) change the cost. Decomposition (22) slightly increases the cost.

For decomposition (24), the cost of type inference for the left-hand side is $2^{pN^{m+1}}$. On the other hand, the cost for the right-hand side is $2^{pN^{N^m}}$. Therefore, this decomposition increases the cost even its exponential height. This is not a bad result when we use the opposite direction of decomposition (24). According to Lemmas 5.4 and 5.8 of 5), the composition of a tdtt and yield mapping is realized by a single mtt in MAC_{IO} if every subtree $\delta(t_1,t_2,\ldots,t_n)$ on the right-hand side of the rules in the tdtt satisfies both of the statements (†):

- All states in $t_2,\ldots,t_n$ are deterministic or π_i occurs at most once in $\tau_{M_2}(t_1)$ and
- All states in $t_2,\ldots,t_n$ are total or π_i occurs at least once in $\tau_{M_2}(t_1)$,

where M_2 is a yield mapping mtt in the proof of Theorem 4.1. As presented in Theorem 5.12[5], a tdtt in TOP_{RES} satisfies the condition (†). An mtt in MAC_{IO} can be constructed by applying yield mapping to the right-hand side of all rules of the tdtt as presented in Lemma 5.8[5]. We modify the axiom tree instead of adding a dummy initial state so as not to change the cost. Thus, we have

$$\text{TOP}_{\text{RES}}[p] \mathbin{;} \text{YIELD}[m] \subseteq \text{MAC}_{\text{IO}}[p,m] \text{ and} \quad (25)$$
$$\text{TOP}_{\dagger}[p] \mathbin{;} \text{YIELD}[m] \subseteq \text{MAC}_{\text{IO}}[p,m], \quad (26)$$

where $\text{TOP}_\dagger$ stands for the class of tdtts satisfying the condition (†).

Since tdtts in $\text{SET} \subset \text{TOP}[1]$ and $\text{EMPTY} \subset D\text{TOP}[1]$ are just identity transformations except for specified input symbols, the condition (†) is preserved on the composition, i.e., we have

$$\text{SET} \mathbin{;} \text{LTOP}_{\dagger\text{NONDEL}}[p] \subseteq \text{TOP}_\dagger[p] \text{ and} \quad (27)$$
$$\text{EMPTY} \mathbin{;} \text{TOP}_{\dagger\text{NONDEL}}[p] \subseteq \text{TOP}_\dagger[p] \quad (28)$$

using compositions (5) and (4), respectively. As their corollaries, we have

$$\text{SET} \mathbin{;} D\text{LTOP}_{\dagger\text{NONDEL}}[p] \subseteq \text{TOP}_\dagger[p], \quad (29)$$
$$\text{EMPTY} \mathbin{;} D\text{TOP}_{\dagger\text{NONDEL}}[p] \subseteq D\text{TOP}_\dagger[p], \quad (30)$$
$$\text{SET} \mathbin{;} D_t\text{LTOP}_{\dagger\text{NONDEL}}[p] \subseteq \text{TOP}_\dagger[p], \text{ and} \quad (31)$$
$$\text{EMPTY} \mathbin{;} D_t\text{TOP}_{\dagger\text{NONDEL}}[p] \subseteq D\text{TOP}_\dagger[p], \quad (32)$$

which still reduce the cost except for composition (32). These compositions are useful when we combine them with other decompositions.

Engelfriet and Vogler[5] also showed a decomposition result for tdtts in Lemma 5.14,

$$\mathrm{TOP}[p] \subseteq D_t\mathrm{TOP}[1] \mathbin{⨾} \mathrm{TOP}_{\mathrm{RES}}[ps], \tag{33}$$

where s is the maximum size of the right-hand side expressions of the rules of the original tdtt. Although it is obvious that the cost of type inference gets worse due to this decomposition, we can reduce the cost by combining it with composition (25) as follows.

$$\begin{aligned}\mathrm{TOP}[p] \mathbin{⨾} \mathrm{YIELD}[m] &\subseteq D_t\mathrm{TOP}[1] \mathbin{⨾} \mathrm{TOP}_{\mathrm{RES}}[ps] \mathbin{⨾} \mathrm{YIELD}[m]\\ &\subseteq D_t\mathrm{TOP}[1] \mathbin{⨾} \mathrm{MAC}_{\mathrm{IO}}[ps, m]. \end{aligned}\tag{34}$$

The cost of type inference is $2^{pN^{N^m}}$ on the left-hand side, while the cost is $2^{psN^{m+1}}$ on the right-hand side. The cost is reduced if $s < N^{N^m-m-1}$. Since the condition holds in many cases, the transformation in law (34) may be used as one of the cost-improvement laws.

4.5 Cost Improvement of Composition of Mtts

The cost of type inference for the sequential composition of many mtts may be reduced if we combine it with the other decompositions (20), (21), and (23). The strategy for the cost improvement consists of two steps:

- Decompose as many mtts as possible by laws (20), (21), and (23).
- Compose as many mtts as possible by laws (3), (4), (5), (6), (7), (27), (28), (29), (30), and (31).

Since no decomposition laws change the cost and all composition laws reduce the cost, this strategy always reduces the cost as long as we can apply composition laws.

It is better to take care of the condition of mtts in the decomposition step so that we can compose many combinations of mtts. For example, consider $\mathrm{MAC}_{\mathrm{OI}} \mathbin{⨾} D_t\mathrm{L}_{io}\mathrm{MAC}$, where L_{io} represents linearity for not only input variables but also context parameters, called *strong linearity*. It is easy to show that

$$D_t\mathrm{L}_{io}\mathrm{MAC}[p, m] \subseteq D_t\mathrm{LTOP}_{\dagger\mathrm{NONDEL}}[p] \mathbin{⨾} \mathrm{YIELD}[m], \tag{35}$$

based on the decomposition law (20). The cost of type inference can be improved as shown in **Fig. 4**, where we explicitly write the number of states and the maximum rank of states. We usually do not have to take these numbers into

$$\begin{aligned}&\mathrm{MAC}_{\mathrm{OI}}[p_1, m_1] \mathbin{⨾} D_t\mathrm{L}_{io}\mathrm{MAC}[p_2, m_2]\\ &\subseteq D_t\mathrm{TOP}_{\mathrm{NONDEL}}[p_1] \mathbin{⨾} \mathrm{YIELD}[m_1] \mathbin{⨾} \mathrm{SET} \mathbin{⨾} D_t\mathrm{LTOP}_{\dagger\mathrm{NONDEL}}[p_2] \mathbin{⨾} \mathrm{YIELD}[m_2]\\ &\hspace{20em}\text{by (23) and (35)}\\ &\subset D_t\mathrm{TOP}_{\mathrm{NONDEL}}[p_1] \mathbin{⨾} \mathrm{YIELD}[m_2] \mathbin{⨾} \mathrm{TOP}_{\dagger}[p_2] \mathbin{⨾} \mathrm{YIELD}[m_2] \qquad \text{by (31)}\\ &\subseteq D_t\mathrm{TOP}_{\mathrm{NONDEL}}[p_1] \mathbin{⨾} \mathrm{YIELD}[m_2] \mathbin{⨾} \mathrm{MAC}_{\mathrm{IO}}[p_2, m_2] \qquad \text{by (26)}\end{aligned}$$

Fig. 4 Example of improvement of sequential composition of mtts.

consideration since the total cost is reduced as long as we only employ composition and decomposition, which do not change the cost, plus at least one of them that reduces the cost.

Actually, this example gives another proof for the cost improvement of type inference for macro forest transducers[17] and macro tree transducers with holes[15] because these classes of transducers are represented by a composition of mtts and forest concatenations (or hole applications) which can be specified by a total deterministic strongly-linear mtt. Our approach can achieve exactly the same cost improvement as their methods.

5. Conclusion

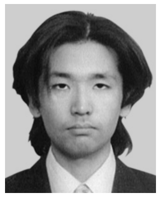
We have shown that the cost of typechecking for sequential compositions of macro tree transducers (mtts) can be improved by applying composition and decomposition constructions to mtts according to their properties, such as being deterministic or total, having no accumulative parameters, or taking OI- or IO-semantics. The results can be used to reduce of the cost of typechecking for macro forest transducers and macro tree transducers with holes.

We have only counted the number of states and the maximum rank to compute the cost of typechecking in this paper. For more precise cost estimation and cost improvement, we may have to count the number of input and output symbols and the size of the right-hand side expressions of rules as required in the transformation law (34). Additionally, we have only considered the construction of deterministic tree automata for inputs. The cost of typechecking may be reduced if we consider the construction of nondeterministic tree automata as Engelfriet did for the composition of tree-walking tree transducers[7]. In this case, we need to count the number of transitions instead of that of states as the size of tree automata. For practical typechecking, it is nicer to consider alternating tree automata as Tozawa [21], Frisch and Hosoya [9] did. A similar technique may improve the cost of typechecking combined with our composition and decomposition.

References

1) Baker, B.S.: Composition of Top-Down and Bottom-Up Tree Transductions, *Inf. Control*, Vol.41, No.2, pp.186–213 (1979).
2) Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S. and Tommasi, M.: Tree Automata Techniques and Applications, Available on: http://www.grappa.univ-lille3.fr/tata.
3) Engelfriet, J.: Bottom-up and top-down tree transformations — A comparison, *Mathematical Systems Theory*, Vol.9, pp.198–231 (1975).
4) Engelfriet, J. and Maneth, S.: A Comparison of Pebble Tree Transducers with Macro Tree Transducers, *Acta Inf.*, Vol.39, pp.613–698 (2003).
5) Engelfriet, J. and Vogler, H.: Macro tree transducers, *J. Comput. Syst. Sci.*, Vol.31, pp.71–146 (1985).
6) Engelfriet, J.: Top-down Tree Transducers with Regular Look-ahead, *Mathematical Systems Theory*, Vol.10, pp.289–303 (1977).
7) Engelfriet, J.: The time complexity of typechecking tree-walking tree transducers, Technical Report Technical Report 2008-01, Leiden Institute of Advanced Computer Science, Leiden University (2008).
8) Fischer, M.: Grammars with macro-like productions, PhD Thesis, Harvard University, Massachusetts (1968).
9) Frisch, A. and Hosoya, H.: Towards Practical Typechecking for Macro Tree Transducers, *Revised Selected Papers in 11th International Symposium on Database Programming Languages*, pp.246–260 (2007).
10) Fülöp, Z. and Vogler, H.: *Syntax-Directed Semantics — Formal Models based on Tree Transducers*, Springer-Verlag (1998).
11) Ganzinger, H. and Giegerich, R.: Attribute Coupled Grammars, *Proc. ACM SIGPLAN Symposium on Compiler Construction*, *SIGPLAN Notices*, Vol.19, No.6, pp.157–170 (1984).
12) Giegerich, R.: Composition and Evaluation of Attribute Coupled Grammars, *Acta Inf.*, Vol.25, No.4, pp.355–423 (1988).
13) Kühnemann, A.: Berechnungsstärken von Teilklassen primitiv-rekursiver Programmschemata, PhD Thesis, Technical University of Dresden (1997).
14) Maneth, S., Berlea, A., Perst, T. and Seidl, H.: XML Type Checking with Macro Tree Transducers, *Proc. 24th ACM SIGMOD Principles of Database Systems*, pp.283–294, ACM Press (2005).
15) Maneth, S. and Nakano, K.: XML Type Checking for Macro Tree Transducers with Holes, *Proc. ACM SIGPLAN Workshop on Programming Language Technologies for XML* (2008).
16) Milo, T., Suciu, D. and Vianu, V.: Typechecking for XML Transformers, *J. Comput. Syst. Sci.*, Vol.66, pp.66–97 (2003).
17) Perst, T. and Seidl, H.: Macro forest transducers, *Inf. Process. Lett.*, Vol.89, pp.141–149 (2004).
18) Rounds, W.: Mappings and grammars on trees, *Mathematical Systems Theory*, Vol.4, pp.257–287 (1970).
19) Suciu, D.: The XML Typechecking Problem, *SIGMOD Record*, Vol.31, pp.89–96 (2002).
20) Thatcher, J.: Generalized^2 sequential machine maps, *J. Comput. Syst. Sci.*, Vol.4, pp.339–367 (1970).
21) Tozawa, A.: Towards static type checking for XSLT, *Proc. ACM Symposium on Document Engineering*, pp.18–27 (2001).
22) Voigtländer, J. and Kühnemann, A.: Composition of Functions with Accumulating Parameters, *Journal of Functional Programming*, Vol.14, pp.317–363 (2004).

(Received February 16, 2009)
(Accepted April 10, 2009)



Keisuke Nakano received his Ph.D. from Kyoto University in 2006. He worked as a researcher at the University of Tokyo from 2003 to 2008. Since March 2008, he has been an assistant professor at the University of Electro-Communications. His research interests include formal language theory, programming language theory, and functional programming. He is a member of JSSST.

Sebastian Maneth is a senior researcher at NICTA, Australias's ICT research center of excellence, and a conjoint associate professor at the University of New South Wales (Dr.). His work has been focused on formal models of tree processing and currently deals with applying those models to build efficient XML query engines.