

XML Type Checking for Macro Tree Transducers with Holes^{*}

Sebastian Maneth

National ICT Australia Ltd.
and University of New South Wales
sebastian.maneth@nicta.com.au

Keisuke Nakano

Department of Mathematical Informatics,
University of Tokyo
ksk@mist.i.u-tokyo.ac.jp

Abstract

Macro forest transducers (mfts) extend macro tree transducers (mtts) from ranked to unranked trees. Mfts are more powerful than mtts (operating on binary tree encodings) because they support sequence concatenation of output trees as build-in operation. Surprisingly, inverse type inference for mfts, for a fixed output type, can be done within the same complexity as for mtts. Inverse type inference is used in algorithms for exact type checking of XML transformations. The macro tree transducer with holes (hmtt) is a new concept that is introduced in this paper. It generalizes sequence concatenation of mfts to arbitrary tree concatenation. Hmtts are strictly more powerful than mfts, in a similar way as mfts are more powerful than mtts. Again, it comes as a surprise that inverse type inference remains within the same complexity bound as for mfts. Hmtts are a natural and robust extension of mtts: any hmtt can be simulated by an mtt, followed by a so called “YIELD-mapping”, and, conversely, any composition of an mtt with a YIELD-mapping can be simulated by an hmtt. This characterization implies that inverse type inference for two-fold compositions of total deterministic mtts can be done in 2-exponential time (a tower of exponents of height 2), while the previously best known algorithm takes 3-exponential time.

1. Introduction

The top-down tree transducer was invented in the 70s by Rounds and Thatcher [17, 20], inspired by research on compilers. Conventionally, such a transducer describes a translation, that is, a relation, over ranked trees, viz., the parse trees of a programming language that are translated during a phase of compilation. A top-down tree transducer can be seen as an attribute grammar [9] that has no inherited attributes, and which has trees as semantic domain and substitution of trees for leaves as only semantic function.

Since XML describes tree structured data, tree transducers serve as a natural model for XML transformations [21, 1, 14]. One technicality of modeling XML documents by trees, is that XML element nodes typically have arbitrary numbers of children; thus, XML is adequately modeled by *unranked* trees. A node of a ranked

tree, in contrast, has a fixed number of children that is determined by its label. In order to deal with this technicality, one can

- (i) code unranked trees by binary trees (and then use classical ranked tree transducers) or
- (ii) generalize the ranked tree transducer model to an unranked one.

As it turns out, these two ways often lead to formalisms with *different* expressive power. For instance, the unranked top-down tree transducers of [11] are strictly more expressive than classical ranked top-down tree transducers; or, the unranked macro tree transducers (called “macro forest transducers”) [16] are strictly more powerful than macro tree transducers [5] operating on binary tree encodings of forests. Only for one of the smallest class of tree translations: partial identities obtained by interpreting tree automata as tree transducers, (i) and (ii) of above yield classes with equal expressiveness. This comes from [18] where it is shown that the expressive power of the classical ranked finite tree automata (see, e.g., [3]) coincides the natural unranked model of finite tree automaton [2].

Consider now the problem of type checking a transformation T described by a tree transducer. Type checking means to decide, for given input and output types I and O , whether $T(t)$ is of type O for all possible input trees t of type I . In our setting, I and O are regular tree languages. In general, tree transducers do *not* preserve regular tree languages, i.e., $T(I)$ is not regular. This means that forward type inference cannot be exact, see, e.g., [19]. On the other hand, all classical models of tree transducers have the property that their inverses do effectively preserve regular tree languages, see, e.g., [5, 8]. Thus, $T^{-1}(O)$ is a regular tree language. Hence, type checking can be done by checking emptiness of $T^{-1}(\overline{O}) \cap I$, which is decidable because regular tree languages are closed under complement and intersection and have decidable emptiness. The bottleneck in this method of type checking is inverse type inference: it can be quite expensive to compute $T^{-1}(O)$. Even if T is only a top-down tree transducer (which has very limited expressive power), then type checking problem based on the computation of $T^{-1}(O)$ is already EXPTIME-complete [13]. For more powerful tree transducer models the problem becomes even harder. For instance, macro tree transducers extend top-down tree transducer by the use of context parameters. For compositions of macro tree transducers, the best known algorithm takes $O(2^W)$ where W is a tower of exponents of height proportional to the number of transducers in the composition, see, e.g., [10]. If a transformation T is given by a macro tree transducer with a fixed number of parameters, and O is a fixed output type, then computing $T^{-1}(O)$ can still be done in EXPTIME. This is nice, because macro tree transducers with a fixed number of parameters are strictly more powerful than top-down tree transducers.

An extension of macro tree transducers from ranked to unranked trees (forests) was introduced by Perst and Seidl [16]. Essentially,

[Copyright notice will appear here once ‘preprint’ option is removed.]

^{*} This work is partially supported by the *Comprehensive Development of e-Society Foundation Software* program of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

they add sequence concatenation of output trees to the macro tree transducer. Their main result is that (1) macro forest transducers are strictly more expressive than macro tree transducers that operate on binary tree encodings of unranked trees and (2) inverse type inference can be done within the same complexity as for macro forest transducers. In this paper we want to show that the result of Perst and Seidl can be seen as an instance of a more general result: macro tree transducers can be extended to “macro tree transducers with holes” by adding support for tree concatenation of output trees; then: (1) the resulting transducers (when operating on binary tree encodings of forests) are strictly more expressive than macro forest transducers and (2) inverse type inference can be done within the same complexity as for macro tree transducers. This is exciting because it brings us from the forest back into the pure tree world. Note that in terms of binary encodings of unranked trees, the sequence concatenation of Perst and Seidl’s model corresponds to adding a right subtree to a binary node. This also explains why their model is more expressive than macro tree transducers on binary encoded trees: their transducer can generate an output sequence of double exponential length. But a macro tree transducer can only generate trees of at most exponential height. The macro tree transducer with holes works on trees and generalizes the addition of a right-subtree (needed to simulate concatenation of Perst and Seidl’s transducers) to the arbitrary addition of subtrees. The resulting transducer can generate output trees of double exponential height, and therefore is strictly more powerful than macro tree and forest transducers.

Interestingly, inverse type inference for the new macro tree transducer with holes can be carried out within the same complexity bound as for macro tree transducers, namely, in exponential time if we keep fixed the output type, the number of holes, and the number of parameters. The proof is similar to the one for macro forest transducers [16]. Next, we provide characterizations of the class of tree translations realized by hmmts: it is equal to (i) compositions of one mtt followed by a “YIELD-mapping”, and, in the deterministic case to (ii) two-fold compositions of deterministic mts. The proof of these equivalences uses standard techniques from tree transducer theory: hole application is similar to tree substitution, and therefore can be simulated by “YIELD-mappings” which interpret trees as symbolic representation of iterated tree substitutions. We show that every macro tree transducer with holes can be simulated by an ordinary macro tree transducer followed by a YIELD mapping, and vice versa. This implies by known results that macro tree transducers with holes are as expressive as two-fold compositions of deterministic macro tree transducers (mtts). For the composition $M_1 \circ M_2$ of two deterministic mts, the best known algorithm for inverse type inference first infers an input type for M_2 . It then uses that type to infer the corresponding input type of M_1 . This gives a complexity of a tower of exponents of height 3 (“3-exponential”). Our results show that such a composition can be translated in polynomial time into an equivalent macro tree transducer with holes. We hence obtain a 2-exponential time algorithm for inverse type inference of two-fold compositions of deterministic macro tree transducers.

2. Preliminaries

We denote the set of non-negative integers including 0 by \mathbb{N} , and the sets $\{1, \dots, n\}$ by $[n]$ for $n \in \mathbb{N}$, in particular, $[0] = \emptyset$.

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 . The set of all ranked trees over Σ is denoted by \mathcal{T}_Σ . When all symbols in Σ have rank 2 except for the special symbol \perp with rank 0, the alphabet Σ and a tree in \mathcal{T}_Σ are called *binary alphabet* and *binary tree*, respectively.

We fix the sets of input variables $X = \{x_1, x_2, \dots\}$, context parameters $Y = \{y_1, y_2, \dots\}$, and holes $H = \{\square_1, \square_2, \dots\}$, and assume that any ranked alphabet Σ is always disjoint with X , Y , and H . In trees, the symbols of X , Y , and H always appear at leaves, i.e., they are all assumed to have rank 0. We denote by $\mathcal{T}_{\Sigma, k}$ ($k \in \mathbb{N}$) the set of k -hole trees $\mathcal{T}_{\Sigma \cup \{\square_l \mid l \in [k]\}}$. For trees $t, u_1, \dots, u_k \in \mathcal{T}_{\Sigma, k}$, the *hole application* $t \odot (u_1, \dots, u_k)$ is defined as follows: $\square_l \odot (u_1, \dots, u_k) = u_l$ for $l \in [k]$, $\square_l \odot (u_1, \dots, u_k) = \square_l$ for $l \notin [k]$ and $\sigma(t_1, \dots, t_n) \odot (u_1, \dots, u_k) = \sigma(t_1 \odot (u_1, \dots, u_k), \dots, t_n \odot (u_1, \dots, u_k))$ for $\sigma \in \Sigma$. The parentheses of the right operand of \odot may be omitted when it is a 1-tuple.

A Σ -forest is a sequence of unranked trees over an alphabet Σ ; it is recursively defined by $f ::= \perp \mid \sigma \langle f \rangle$ with $\sigma \in \Sigma$, where \perp denotes the empty forest. For a forest of the form $f \perp$ we simply write f . The set of all Σ -forests is denoted \mathcal{F}_Σ . A Σ -forest f is simulated by a 1-hole binary tree $\lceil f \rceil$ over Σ , where $\lceil f \rceil$ is defined by $\lceil \perp \rceil = \square_1$ and $\lceil \sigma \langle f_1 \rangle f_2 \rceil = \sigma(\lceil f_1 \rceil \odot \perp, \lceil f_2 \rceil)$. A binary tree simulating a forest has only one hole, at the rightmost leaf. The concatenation $f_1 f_2$ of two forests f_1 and f_2 is simulated by $\lceil f_1 \rceil \odot \lceil f_2 \rceil$. The binary tree representation of a forest f is given by $\lceil f \rceil \odot \perp$.

We denote the composition of tree transformations f and g by $f \circ g$, i.e., $(f \circ g)(t) = g(f(t))$. For two classes F and G of tree transformations, $F \circ G$ denotes the class of their composition, that is, $F \circ G = \{f \circ g \mid f \in F, g \in G\}$.

A (*deterministic*) *bottom-up tree automaton* 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, $\beta : \Sigma^{(n)} \times B^n \rightarrow B$, $n \geq 0$ is the transition function, and $B_f \subset B$ is the set of final states. We extend the transition function β to trees in \mathcal{T}_Σ by defining $\beta(\sigma(t_1, \dots, t_k)) = \beta(\sigma, \beta(t_1), \dots, \beta(t_k))$ for any tree $\sigma(t_1, \dots, t_k)$ with $\sigma \in \Sigma^{(k)}$, $k \geq 0$, and $t_1, \dots, t_k \in \mathcal{T}_\Sigma$. The language $L(A)$ recognized by A is then defined as $\{t \in \mathcal{T}_\Sigma \mid \beta(t) \in B_f\}$. We assume the reader to be familiar with the basic notions of tree automata [3].

3. Macro Tree Transducers with Holes

The macro tree transducer (mtt) [5] is a combination of the top-down tree transducer [17, 20] and the macro grammar [6]. It can be obtained by adding *context parameters* to the rules of a top-down tree transducer. Such context parameters are placeholders for output trees of the transducer, and are instantiated with actual output trees when a rule of the transducer is applied. Roughly speaking, the context parameters of a macro tree transducer correspond to the inherited attributes of an attribute grammar. While the macro tree transducer traverses its input tree, going top-down, it uses the context parameters to build up output trees in a bottom-up fashion. As an example, consider the rules

$$\begin{aligned} q(\gamma(x_1), y_1) &\rightarrow q(x_1, q(x_1, y_1)) \\ q(\alpha, y_1) &\rightarrow \sigma(y_1, y_1). \end{aligned}$$

For an input tree $\gamma(\dots \gamma(\alpha) \dots)$ consisting of n occurrences of γ , and for an initial value $y_1 = \alpha$ of y_1 , these rules generate as output the full binary tree over $\{\sigma^{(2)}, \alpha^{(0)}\}$ of height 2^n (and hence of size $2^{2^n - 1}$). Note that this particular translation cannot be realized by any attribute grammar, because tree translations of attribute grammars have linear height increase (see, e.g., Lemma 5.40 of [8]); on the other hand, it is well-known that the tree translation of any attribute grammar can be realized by a macro tree transducers (see, e.g., Lemma 6.1 of [8]). Before we formally define mts with holes (which are simply mts which can have holes and hole application in their output trees), we give another example of an mtt. The full description of an mtt M has an initial state “in” which may not have any parameters; this guarantees that output trees of M do not

contain parameters anymore. Consider the following rules:

$$\begin{aligned} in(succ(x_1)) &\rightarrow q(x_1, in(x_1)) \\ in(zero) &\rightarrow zero \\ q(succ(x_1), y_1) &\rightarrow succ(succ(q(x_1, y_1))) \\ q(zero, y_1) &\rightarrow succ(y_1). \end{aligned}$$

It should be clear that this macro tree transducer translates monadic trees $succ(\dots succ(zero)\dots)$ with n occurrences of $succ$ into such monadic trees, but with $n \cdot n$ occurrences of $succ$. Thus, the transducer computes the square of a number n that is represented as the monadic tree $succ^n(zero)$.

We now extend the macro tree transducer by allowing the use of holes and hole applications in the right-hand sides of rules.

DEFINITION 3.1. A macro tree transducer with k -holes (k -hmtt) is a tuple $M = (Q, \Sigma, \Delta, in, R)$, where

- Q is a set of ranked states;
- Σ and Δ are ranked alphabets with $Q \cap (\Sigma \cup \Delta) = \emptyset$, called the input alphabet and the output alphabet, respectively,
- $in \in Q$ is the initial state whose rank is less than or equal to k ,
- R is a set of rules such that $R = \bigcup_{q \in Q} R_q$ with finite sets R_q of q -rules of the form $q(\sigma(x_1, \dots, x_n), y_1, \dots, y_m) \rightarrow e$ with input variables x_i ($i \in [n]$) and formal parameters y_j ($j \in [m]$); the right hand side e is defined by the following grammar:

$$e ::= q'(x_i, e_1, \dots, e_{m'}) \mid y_j \mid \delta(e_1, \dots, e_r) \mid e \odot (e_1, \dots, e_{k'}) \mid \square_l$$

with $q' \in Q^{(m'+1)}$, $m' \geq 0$, $\delta \in \Delta^{(r)}$, $r \geq 0$, $k' \leq k$ and $l \in [k]$.

We will simply say hmtt for any k -hmtt. If there are several rules for the same state q and the same input symbol σ , we may write

$$q(\sigma(x_1, \dots, x_n), y_1, \dots, y_m) \rightarrow e_1 \mid \dots \mid e_r$$

for the rule to enumerate all right-hand side expressions e_1, \dots, e_r of the rules. An hmtt M is called deterministic if there exists at most one rule for every state $q \in Q$ and input symbol $\sigma \in \Sigma$. An hmtt M is called total if there exists at least one rule for every state $q \in Q$ and input symbol $\sigma \in \Sigma$.

Let q be a state of rank $m + 1$ of a k -hmtt. The semantics of q is a function which takes $m + 1$ arguments, an input tree plus the m actual values of q 's context parameters, and returns the corresponding output trees of q . Since we consider non-deterministic tree transducers, the arguments and output of the semantic function of q are sets of trees.

DEFINITION 3.2. Let $M = (Q, \Sigma, \Delta, in, R)$ be a k -hmtt and $t \in \mathcal{T}_\Sigma$. The semantics of the state $q \in Q$ of rank $m + 1$ is a function $\llbracket q \rrbracket : \mathcal{T}_\Sigma \times (2^{\mathcal{T}_{\Delta, k}})^m \rightarrow 2^{\mathcal{T}_{\Delta, k}}$. Let the q -rule be $q(\sigma(x_1, \dots, x_n), y_1, \dots, y_m) \rightarrow e_1 \mid \dots \mid e_r$. The function $\llbracket q \rrbracket$ is defined by $\llbracket q \rrbracket(\delta(t_1, \dots, t_n), S_1, \dots, S_m) = \llbracket e_1 \rrbracket_{\rho\theta} \cup \dots \cup \llbracket e_r \rrbracket_{\rho\theta}$ with $\rho(x_i) = t_i$ for $i \in [n]$ and $\theta(y_j) = S_j$ for $j \in [m]$ where $\llbracket _ \rrbracket_{\rho\theta}$ denotes the evaluation of a right-hand side expression with respect to the binding ρ and θ of the variables x_i and y_j , respectively, that is defined by

$$\begin{aligned} \llbracket q'(x_i, e'_1, \dots, e'_{m'}) \rrbracket_{\rho\theta} &= \\ &\llbracket q' \rrbracket(\rho(x_i), \llbracket e'_1 \rrbracket_{\rho\theta}, \dots, \llbracket e'_{m'} \rrbracket_{\rho\theta}), \\ \llbracket y_j \rrbracket_{\rho\theta} &= \theta(y_j), \\ \llbracket \delta(e'_1, \dots, e'_{n'}) \rrbracket_{\rho\theta} &= \\ &\{\delta(t_1, \dots, t_{n'}) \mid t_i \in \llbracket e'_i \rrbracket_{\rho\theta}, i \in [n']\}, \\ \llbracket e'_0 \odot (e'_1, \dots, e'_{k'}) \rrbracket_{\rho\theta} &= \\ &\{t_0 \odot (t_1, \dots, t_{k'}) \mid t_i \in \llbracket e'_i \rrbracket_{\rho\theta}, i \in \{0\} \cup [k']\}, \\ \llbracket \square_l \rrbracket_{\rho\theta} &= \{\square_l\}. \end{aligned}$$

The transformation of a k -hmtt is obtained by applying the semantic function of its initial state. In general, the resulting output trees may still contain up to k' parameters, if k' is the rank of the initial state. We replace every such parameter y_j by the j th hole \square_j . Thus, output trees may contain holes. Note that usually we do not use of this possibility, but choose the initial state to be of rank 1 (which means that output trees do not contain holes).

DEFINITION 3.3. The transformation induced by a k -hmtt $M = (Q, \Sigma, \Delta, in, R)$ is the function $\tau_M : \mathcal{T}_\Sigma \rightarrow 2^{\mathcal{T}_{\Delta, k'}}$ defined by $\tau_M(t) = \llbracket in \rrbracket(t, \{\square_1\}, \dots, \{\square_{k'}\})$ where $k' (\leq k)$ is the rank of in .

EXAMPLE 3.4. Let $M_{sq} = (Q, \Sigma, \Delta, in, R)$ be a 1-hmtt where $Q = \{in^{(1)}, f^{(2)}, g^{(1)}\}$, $\Sigma = \Delta = \{succ^{(1)}, zero^{(0)}\}$ and R consists of the following rules.

$$\begin{aligned} in(succ(x_1)) &\rightarrow f(x_1, g(x_1)) \\ in(zero) &\rightarrow zero \\ f(succ(x_1), y_1) &\rightarrow y_1 \odot f(x_1, y_1) \\ f(zero, y_1) &\rightarrow y_1 \odot zero \\ g(succ(x_1)) &\rightarrow succ(g(x_1)) \\ g(zero) &\rightarrow succ(\square_1) \end{aligned}$$

As an example, consider the input tree $s_3 = succ(succ(succ(zero)))$. We compute $\llbracket in \rrbracket(succ(succ(succ(zero)))) = \llbracket f(x_1, g(x_1)) \rrbracket_\rho$ with $\rho(x_1) = succ(succ(zero))$. This equals

$$\begin{aligned} &\llbracket f \rrbracket(succ(succ(zero)), \llbracket g \rrbracket(succ(succ(zero)))) \\ &= \llbracket f \rrbracket(succ(succ(zero)), succ(succ(succ(\square_1)))) \\ &= succ^3(\square_1) \odot \llbracket f \rrbracket(succ(zero), succ^3(\square_1)). \end{aligned}$$

Proceeding in a similar way we finally obtain the tree $succ^3(\square_1) \odot succ^3(\square_1) \odot succ^3(\square_1) \odot zero = succ^9(zero)$. Thus, this 1-hmtt realizes the same transformation as the example mtt that was given at the beginning of the section, i.e., it computes the square number of the input, with number n represented as $succ^n(zero)$.

An hmtt is an mtt if there is no rule whose right hand side contains either \odot or \square_l , in other words, an mtt is a 0-hmtt. The classes of all transformations that can be realized by hmtts, mttts, and total deterministic mttts are denoted by HMTT, MAC and MAC_{det,tot}, respectively. It is obvious that $MAC \subseteq HMTT$. We will show in the following section that this inclusion is strict.

4. Characterization

First we show that every macro forest transducer (for short, mft) can be simulated by an hmtt. Mftts extend mttts by providing a concatenation operation for output forests. As an example, consider the following mft, which translates input forests over the alphabet $\{a\}$ (with at least one a) into output forests over $\{b\}$.

$$\begin{aligned} q_0(a(x_1)x_2) &\rightarrow q(x_1, q(x_1, b(\perp))) \\ q(a(x_1)x_2, y_1) &\rightarrow q(x_1, q(x_1, y_1)) \\ q(\perp, y_1) &\rightarrow y_1 y_1 \end{aligned}$$

It should be intuitively clear that the above mft translates an input forest with n a -nodes into an output forests consisting of a sequence of 2^{2^n} one node trees (labeled b).

It is easy to see that the concatenation present in an mft can be simulated by a hole application; in fact, recall from the Preliminaries the definition of the 1-hole binary tree $\lceil f \rceil$ (and recall also that we write f instead of $f \perp$). Then, the forest $f_1 f_2$ is simulated by $\lceil f_1 \rceil \odot \lceil f_2 \rceil$. This means that the first rule of the above mft is simulated by this 1-hmtt rule

$$q_0(a(x_1, x_2)) \rightarrow q(x_1, q(x_1, b(\perp, \square_1)))$$

and the last rule is simulated by the rule with same left-hand side and with right-hand side $y_1 \odot y_1$. This shows that any mft can

easily be simulated (on binary trees) by simply applying $\lceil \cdot \rceil$ to each right-hand side of the mft's rule (and considering each input symbol as binary, besides the special symbol \perp of rank zero). Thus, $\text{FMAC} \subseteq \text{HMAC}$ holds where FMAC denotes the class of all transformation induced by mft's (seen as binary tree translations).

We now show the properness of this inclusion by using the notion of the *height* and *binary height* of forests. Let Σ be an alphabet and $f \in \mathcal{F}_\Sigma$ be a forest. For a binary tree representation $t = \lceil f \rceil \odot \perp$, the height $\text{ht}(t)$ and the binary height $\text{bht}(t)$ is inductively defined by

$$\begin{aligned} \text{ht}(\perp) &= \text{bht}(\perp) = 0 \\ \text{ht}(\sigma(x_1, x_2)) &= \max\{1 + \text{ht}(x_1), \text{ht}(x_2)\} \\ \text{bht}(\sigma(x_1, x_2)) &= 1 + \max\{\text{bht}(x_1), \text{bht}(x_2)\} \end{aligned}$$

where $\sigma \in \Sigma$. The height $\text{ht}(t)$ is the maximal nesting depth of elements in the corresponding forest. The binary height $\text{bht}(t)$ is the height of the binary tree.

LEMMA 4.1. *There exists an hmtt M such that, for infinitely many binary trees t , $\text{ht}(\tau_M(t)) \geq 2^{2^{\text{bht}(t)}}$.*

Proof. Let $M = (Q, \Sigma, \Delta, \text{in}, R)$ be a 1-hmtt with

$$\begin{aligned} Q &= \{\text{in}^{(1)}, q^{(2)}\}, \quad \Sigma = \{a^{(2)}, \perp^{(0)}\}, \quad \Delta = \{b^{(2)}, \perp^{(0)}\}, \\ R &= \{ \text{in}(a(x_1, x_2)) \rightarrow q(x_2, q(x_2, b(\square_1, \perp))) \odot \perp, \\ &\quad \text{in}(\perp) \rightarrow b(b(\perp, \perp), \perp), \\ &\quad q(a(x_1, x_2), y_1) \rightarrow q(x_2, q(x_2, y_1)), \\ &\quad q(\perp, y_1) \rightarrow y_1 \odot y_1 \}. \end{aligned}$$

Let $T = \{t_i \mid i \in \mathbb{N}\}$ and $U = \{u_i \mid i \in \mathbb{N}\}$ be sets of binary trees defined by $t_0 = \perp$, $t_{n+1} = a(\perp, t_n)$ for $n \in \mathbb{N}$, $u_0 = \square_1$ and $u_{n+1} = b(u_n, \perp)$ for $n \in \mathbb{N}$. The hmtt M performs a transformation from T to U . Consider a function h such that $h(z, r) = \text{ht}(\llbracket q \rrbracket(t_z, u_r))$ for $z, r \in \mathbb{N}$. Then we can show by induction on z that $h(z, r) = 2^{2^z} \cdot r$. Therefore we have

$$\begin{aligned} \text{ht}(\tau_M(t_z)) &= \text{ht}(\llbracket \text{in} \rrbracket(t_z)) \\ &= \text{ht}(\llbracket q \rrbracket(t_{z-1}, \llbracket q \rrbracket(t_{z-1}, u_1)) \odot \perp) \\ &= \text{ht}(\llbracket q \rrbracket(t_{z-1}, u_{2^{z-1}}) \odot \perp) \\ &= \text{ht}(u_{2^{2^{z-1}}} \odot \perp) \\ &= \text{ht}(u_{2^{2^z}} \odot \perp) = 2^{2^z} = 2^{2^{\text{bht}(t)}} \end{aligned}$$

for any $z \geq 1$. Hence the lemma is proved. \square

THEOREM 4.2. $\text{FMAC} \subseteq \text{HMAC}$.

Proof. The height property for FMAC says that for every mft $M \in \text{FMAC}$ there is a constant $c > 0$ such that for every $f \in \mathcal{F}_\Sigma$, $\text{ht}(\tau_M(f)) \leq 2^{c \cdot \text{bht}(f)}$ [16, Theorem 11]. Hence the translation realized by the hmtt M of LEMMA 4.1 cannot be realized by any mft. \square

From [16, Theorem 8] characterizing mft's, the properness of the inclusion relation $\text{MAC} \subset \text{HMAC}$ is immediately obtained.

COROLLARY 4.3. $\text{MAC} \subsetneq \text{HMAC}$.

In Theorem 9 of [16] it is shown that every mft can be simulated by the composition of two mfts (on binary tree representations of forests). In the following theorem we strengthen this result by showing that even every hmtt can be simulated by a particular composition of two mfts, namely, by one where the second mft is restricted to be total and deterministic. In fact, the second translation is of a special kind: it interprets the holes and hole application. The idea comes from a classical decomposition result for total deterministic mfts: every such transducer can be decomposed into a top-down tree transducer followed by a ‘‘YIELD’’ mapping. The lat-

ter interprets internal nodes as substitution, and leaf nodes as constant trees with parameters. Recall from the Preliminaries that $H = \{\square_1, \square_2, \dots\}$. Formally, let Σ and Δ be ranked alphabets and let f be a function from $\Sigma^{(0)}$ to trees over $\Delta \cup H$. Then f induces the YIELD-mapping $\text{YIELD}_f : \mathcal{T}_\Sigma \rightarrow \mathcal{T}_{\Delta \cup H}$ defined as $\text{YIELD}_f(\alpha) = f(\alpha)$ for $\alpha \in \Sigma^{(0)}$ and $\text{YIELD}_f(\sigma(s_0, \dots, s_k)) = \text{YIELD}_f(s_0) \odot (\text{YIELD}_f(s_1), \dots, \text{YIELD}_f(s_k))$. Note that the conventional definition (see, e.g., [5]) of YIELD uses Y instead of H , and is defined using tree substitution; in fact, let $s[y_1 \leftarrow t_1, \dots, y_m \leftarrow t_m]$ denote the result of substituting every occurrence of y_j in s by the tree t_j , for all $1 \leq j \leq m$. Then we have $\text{YIELD}_f(\sigma(s_0, \dots, s_k)) = \text{YIELD}_f(s_0)[y_1 \leftarrow \text{YIELD}_f(s_1), \dots, y_k \leftarrow \text{YIELD}_f(s_k)]$ for $\sigma \in \Sigma^{(k+1)}$, $k \geq 0$, and $s_1, \dots, s_k \in \mathcal{T}_\Sigma$. This obviously coincides with our above definition using H and hole application. Note further that this definition of YIELD is slightly more general than earlier definitions which use derived signatures, see [5] for a discussion. The following lemma says that an hmtt can be viewed as an mtt (producing holes and hole application as output symbols), followed by a YIELD mapping. This result is obvious from the definition, but depends on the particular definition of YIELD. Since we took the general definition of YIELD from [5], we need to define new nullary output symbols for the mtt together with a mapping f on those symbols which induces the YIELD-mapping YIELD_f .

LEMMA 4.4. $\text{HMAC} \subseteq \text{MAC} \circ \text{YIELD}$. *Totality, determinism, and the number of parameters are preserved when going from an hmtt to an mtt (and YIELD-mapping).*

Proof. Let $M = (Q, \Sigma, \Delta, \text{in}, R)$ be a k -hmtt, $k \geq 0$. Let $\delta \in \Delta^{(m)}$ with $m \geq 1$. We define new symbols $\hat{\delta}$ of rank zero and δ' of rank $(m+1)$. Furthermore, we define $f(\hat{\delta}) = \delta(\square_1, \dots, \square_m)$. The mtt M' is obtained from M by recursively replacing in the right-hand side of every rule of M : (i) any subtree of the form $\delta(s_1, \dots, s_m)$ by the tree $\delta'(\hat{\delta}, s_1, \dots, s_m)$ and (ii) any expression of the form $e \odot (e_1, \dots, e_{k'})$, $k' \leq k$, by the tree $\odot_{k'}(e, e_1, \dots, e_{k'})$ where $\odot_{k'}$ of rank $k'+1$ is a new output symbol of M' . It should be clear that $\tau_{M'} \circ \text{YIELD}_f = \tau_M$ and that M' is total and deterministic if M is. \square

We now show that also to converse of Lemma 4.4 holds, i.e., every composition of an mtt with a YIELD-mapping can be simulated by an hmtt.

LEMMA 4.5. $\text{MAC} \circ \text{YIELD} \subseteq \text{HMAC}$. *Totality, determinism, and the number of parameters are preserved when going from an mtt (and YIELD-mapping) to an hmtt.*

Proof. Let $M = (Q, \Sigma, \Delta, \text{in}, R)$ be an mtt where the maximum rank of states in Q is $k+1$ and let $\text{YIELD}_f : \mathcal{T}_\Delta \rightarrow \mathcal{T}_{\Gamma \cup H}$ be a YIELD-mapping with a function $f : \Delta^{(0)} \rightarrow \mathcal{T}_{\Gamma \cup H}$. In order to prove the statement, we construct a k -hmtt $M' = (Q', \Sigma, \Delta, \text{in}', R')$ from M such that $\tau_{M'} = \tau_M \circ \text{YIELD}_f$. The set Q' of states consists of all states with a hat in Q , i.e., $Q' = \{\hat{q}^{(n)} \mid q^{(n)} \in Q\}$. The initial state is given by $\text{in}' = \hat{\text{in}}$. For every rule $q(\sigma(x_1, \dots, x_n), y_1, \dots, y_m) \rightarrow e$ in R , the set R' has a rule:

$$\hat{q}(\sigma(x_1, \dots, x_n), y_1, \dots, y_m) \rightarrow I(e)$$

where I translates the right-hand side expressions in R to those in R' and is defined by

$$\begin{aligned} I(q'(x_i, e_1, \dots, e_{m'})) &= \hat{q}'(x_i, I(e_1), \dots, I(e_{m'})) \\ I(y_j) &= y_j \\ I(\delta(e_1, \dots, e_{n'})) &= I(e_1) \odot (I(e_2), \dots, I(e_{n'})) \quad (n' \geq 1) \\ I(\delta) &= f(\delta) \quad (\delta \in \Delta^{(0)}). \end{aligned}$$

This concludes the construction of M' . In order to show that $\tau_{M'}(t) = \text{YIELD}_f(\tau_M(t))$ for any input tree t , we prove, by induction on the structure of t , that $\text{YIELD}_f(\llbracket q \rrbracket(t, s_1, \dots, s_m)) = \llbracket \hat{q} \rrbracket(t, \text{YIELD}_f(s_1), \dots, \text{YIELD}_f(s_m))$ for any output trees s_1, \dots, s_m . Assume that $t = \sigma(t_1, \dots, t_n)$ and R contains a rule

$$q(\sigma(x_1, \dots, x_n), y_1, \dots, y_m) \rightarrow e_1 \mid \dots \mid e_r.$$

Then we have

$$\begin{aligned} & \text{YIELD}_f(\llbracket q \rrbracket(t, s_1, \dots, s_m)) \\ &= \text{YIELD}_f(\llbracket e_1 \rrbracket_{\rho\theta}) \cup \dots \cup \text{YIELD}_f(\llbracket e_r \rrbracket_{\rho\theta}) \end{aligned}$$

and

$$\begin{aligned} & \llbracket \hat{q} \rrbracket(t, \text{YIELD}_f(s_1), \dots, \text{YIELD}_f(s_m)) \\ &= \llbracket I(e_1) \rrbracket_{\rho\hat{\theta}} \cup \dots \cup \llbracket I(e_r) \rrbracket_{\rho\hat{\theta}} \end{aligned}$$

by the definition of $\llbracket q \rrbracket$ and $\llbracket \hat{q} \rrbracket$ where $\rho(x_i) = t_i$ for $i \in [n]$, $\theta(y_j) = s_j$ for $j \in [m]$ and $\hat{\theta} = \theta \circ \text{YIELD}_f$. These are equal if the following statement holds:

$$\text{YIELD}_f(\llbracket e \rrbracket_{\rho\theta}) = \llbracket I(e) \rrbracket_{\rho\hat{\theta}}$$

for any right-hand side expression e where the induction hypothesis holds for every t_i , that is, $\text{YIELD}_f(\llbracket q' \rrbracket(t_i, s'_1, \dots, s'_m)) = \llbracket \hat{q}' \rrbracket(t_i, \text{YIELD}_f(s'_1), \dots, \text{YIELD}_f(s'_m))$ holds for any q' and s'_1, \dots, s'_m . We can show the statement by a simple induction on the structure of e . \square

Lemmas 4.4 and 4.5 give us the following characterization of hmtts in terms of mtts and YIELD-mappings. In fact, let us now explicitly mention the number of parameters: for $m \geq 0$, denote by $m\text{-HMAC}$ and $m\text{-MAC}$ the classes of transformations realized by hmtts and mtts with at most m parameters.

THEOREM 4.6. *For $m \in \mathbb{N}$, $m\text{-HMAC}_{(\text{det}),(\text{tot})} = m\text{-MAC}_{(\text{det}),(\text{tot})} \circ \text{YIELD}$.*

Next we would like to show that the composition of two deterministic mtts is equal to one deterministic hmtt. By Theorem 6.18 of [5], $\text{MAC}_{\text{det}} = \text{FTA} \circ \text{MAC}_{\text{det,tot}}$, where FTA denotes the class of partial identity mappings, corresponding to finite tree automata. Note that in [5] the class MAC_{det} is denoted by MT_{OI} which refers to macro tree transducers with “outside-in (OI)” evaluation order; the latter means that in a sentential form with nested state calls, we always evaluate top-down (or outside-in, in terms of the bracketed expression) which corresponds to call-by-name or lazy evaluation in functional programming terminology. In our definition we did not specify any fixed evaluation order for nested state calls, but left it unrestricted. It is known that mtts (or deterministic ones) with unrestricted evaluation order have exactly the same expressiveness as mtts with outside-in evaluation order (see Corollary 3.13 of [5]). So we have $\text{MAC}_{\text{det}} \circ \text{MAC}_{\text{det}} = \text{MAC}_{\text{det}} \circ \text{FTA} \circ \text{MAC}_{\text{det,tot}}$. It is easy to see that $\text{MAC}_{\text{det}} \circ \text{FTA} = \text{MAC}_{\text{det}}$: given a deterministic mtt M and an FTA mapping defining the regular tree language L , we can restrict M to L by first constructing the regular input tree language $I = \tau_M^{-1}(L)$ using Theorem 7.4 of [5]. The restriction of M to inputs in I can easily be realized by regular look-ahead; since MAC_{det} is closed under regular look-ahead by Theorem 6.15 of [5], we obtain the desired result. Thus, $\text{MAC}_{\text{det}} \circ \text{MAC}_{\text{det}} = \text{MAC}_{\text{det}} \circ \text{MAC}_{\text{det,tot}}$. Next, we apply the decomposition $\text{MAC}_{\text{det,tot}} = \text{TOP}_{\text{det,tot}} \circ \text{YIELD}$ (Corollary 5.9 of [5]) to the second deterministic mtt, and then compose the top-down tree transducer with the first mtt through $\text{MAC}_{\text{det}} \circ \text{TOP}_{\text{det,tot}} = \text{MAC}_{\text{det}}$ (Theorem 7.6 of [5]). We obtain $\text{MAC}_{\text{det}} \circ \text{MAC}_{\text{det}} \subseteq \text{MAC}_{\text{det}} \circ \text{YIELD}$. The latter class equals DHMAC, the class of all translations realized by deterministic mtts with holes (this is the deterministic version of Theorem 4.6). Note that when

we do inverse type inference in Section 5 then we will only simulate two-fold compositions of *total* deterministic mtts by hmtts. The reason is that the constructions mentioned above can be problematic in the sense that the FTA mappings get too large (exponential in the number of parameters); see the Conclusions for a further discussion.

COROLLARY 4.7. $\text{MAC}_{\text{det}} \circ \text{MAC}_{\text{det}} = \text{DHMAC} \subseteq \text{HMAC}$.

Top-Down Translations

A k -hmtt is a *top-down tree transducer with k -holes* (k -htop) if all its states have rank 1. We simply say htop for any k -htop. The class of all transformations realized by htops is denoted HTOP. We now characterize htops in terms of mtts. In fact, the $m = 0$ case of Theorem 4.6 gives $\text{HTOP} = \text{TOP} \circ \text{YIELD}$ and $\text{DHTOP} = \text{DTOP} \circ \text{YIELD}$; the latter equals DMT_{IO} by Theorem 5.24 of [5], which is the class of transformations realized by deterministic mtts with inside-out (IO) evaluation order. This order corresponds to call-by-name or eager evaluation in functional programming jargon, i.e., it means that a state call in a sentential form may only be replaced if all its arguments trees do not contain state calls, i.e., are terminal trees in T_{Δ} . Interestingly, also in the nondeterministic case of the above equation there is a characterization by mtts with IO order: by Theorem 5.15 of [5], $\text{TOP} \circ \text{YIELD} = \text{LHOM} \circ \text{MT}_{\text{IO}}$, where MT_{IO} denotes the class of all transformations realized by (nondeterministic) mtts with IO evaluation order and LHOM denotes the class of “linear tree homomorphisms”, i.e., the class of translations realized by total, deterministic top-down tree transducers with one state only and with rules whose right-hand sides are linear in the input variables x_i .

COROLLARY 4.8. $\text{HTOP} = \text{TOP} \circ \text{YIELD} = \text{LHOM} \circ \text{MT}_{\text{IO}}$ and $\text{DHTOP} = \text{DTOP} \circ \text{YIELD} = \text{DMT}_{\text{IO}}$.

5. Inverse Type Inference

There are various type formalisms for XML, such as DTD, XML Schema or RELAX NG. In terms of their tree structures, all of these formalisms are subsumed by the regular tree languages [15]. We therefore consider the regular tree languages as input and output types of our XML transformations. Type checking of an XML transformation consists of deciding whether every tree that is valid for the input type is translated into a tree that is valid for the output type. One well known technique for type checking an XML transformation T is inverse type inference: since the inverse of T effectively preserves the regular tree languages, we can translate backwards the complement of the output type, and then intersect with the input type. The result is a regular tree language which can effectively be computed and which is empty if and only if T type checks with respect to the given input and output types. The most expensive step in this type checking procedure is inverse type inference, i.e., the computation of the type $\tau_T^{-1}(R)$ of input trees for which T generates outputs in R . Note that for deterministic top-down tree transducers (that is, macro hole tree transducer without holes and without parameters) the type checking problem is already EXPTIME-complete [13]. In practice, however, even macro tree translations can be type checked quite efficiently, as was recently shown by Frisch and Hosoya [7]. They use inverse type inference and represent input and output types by alternating tree automata. In [16], Perst and Seidl showed that macro forest transducers, which are strictly more expressive than macro tree transducers (operating on binary tree encodings of forests) can be type checked using inverse type inference, with the same complexity as ordinary macro tree transducers, if the output type is kept fixed. Here we go one step further and show that even for macro hole tree transducers, inverse type inference can be done with the same complexity as for

macro tree and forest transducers. At the same time, we know from the previous section that macro hole tree transducers (on binary tree encodings of forests) are strictly more expressive than macro forest transducers.

We now show in the next theorem how to achieve inverse type inference for macro hole tree transducers. Given an output type, represented by a tree automaton, and a macro hole transducer, we show how to construct a tree automaton which recognizes precisely the input trees which are translated by the transducer to tree in the output type.

THEOREM 5.1. *Let M be a k -hmtt and L a regular tree language. Then $\tau_M^{-1}(L) = \{s \mid \tau_M(s) \cap L \neq \emptyset\}$ is effectively regular.*

Proof. Let $M = (Q, \Sigma, \Delta, in, R)$ be a k -hmtt and $A_L = (B, \Delta, \beta, B_f)$ a deterministic bottom-up tree automaton which recognizes the trees in L , i.e., for which $L(A_L) = L$. We construct the bottom-up tree automaton $A = (D, \Sigma, \kappa, D_f)$ which accepts all input trees s such that $\tau_M(s) \in L$.

- The set of states $D = Dom \rightarrow 2^{B^{k+1}}$ where $Dom = \{(q^{(m)}, S_1, \dots, S_m) \mid q \in Q, S_1, \dots, S_m \in 2^{B^{k+1}}\}$.

- The transition $\kappa : \Sigma^{(n)} \times D^n \rightarrow D$ of states is defined as $\kappa(\sigma, d_1, \dots, d_n)(q, S_1, \dots, S_m) = F(e_1) \cup \dots \cup F(e_r)$ for the rule $q(\sigma(x_1, \dots, x_n), y_1, \dots, y_m) \rightarrow e_1 \mid \dots \mid e_r$ in R where $F(e)$ is given by

$$\begin{aligned} F(q'(x_i, e'_1, \dots, e'_{m'})) &= d_i(q', F(e'_1), \dots, F(e'_{m'})), \\ F(y_j) &= S_j, \\ F(\delta(e'_1, \dots, e'_{n'})) &= \\ &\{(b, b'_1, \dots, b'_k) \mid b = \beta(\delta, b_1, \dots, b_{n'}), \\ &\quad \forall i \in [n'] : (b_i, b'_1, \dots, b'_k) \in F(e_i)\}, \end{aligned}$$

$$F(\square_l) = \{(b_l, b_1, \dots, b_k) \mid b_1, \dots, b_k \in B\},$$

$$\begin{aligned} F(e'_o \odot (e'_1, \dots, e'_{k'})) &= \\ &\{(b_o, b'_1, \dots, b'_k) \mid (b_o, b_1, \dots, b_k) \in F(e'_o), \\ &\quad \forall l \in [k'] : (b_l, b'_1, \dots, b'_k) \in F(e'_l), \\ &\quad \forall l \notin [k'] : b_l = b'_l\}, \end{aligned}$$

- The set of final states D_f consists of all functions d such that the set $d(in, S_1, \dots, S_m)$ contains (b, b_1, \dots, b_k) with $b \in B_f$ and $b_1, \dots, b_k \in B$ where $S_l = \{(b_l, b'_1, \dots, b'_k) \mid b_l = b'_l, b'_1, \dots, b'_k \in B\}$ for every $l \in [k]$.

We give an intuitive explanation for the construction of the automaton A . As a state $b \in B$ corresponds to an output tree in T_Δ , a $(k+1)$ -tuple $(b_0, b_1, \dots, b_k) \in B^{k+1}$ in the definition of D and Dom corresponds to an output tree with k holes. The tree is accepted via the state b_0 when \square_l is filled by the tree accepted via the state b_l for every $l \in [k]$. A state $d \in D$ corresponds to all possible input trees t such that a set $\llbracket q \rrbracket(t, T_1, \dots, T_m)$ consists of output trees which corresponds to $d(q, S_1, \dots, S_m)$ where $S_j \in 2^{B^{k+1}}$ corresponds to T_j for each $j \in [m]$. The transition κ guarantees that the input symbol is exactly used with a rule in R when its subtrees corresponds to a state in D .

The sets S_1, \dots, S_m in the third item corresponds to a singleton set of a hole $\square_1, \dots, \square_m$, respectively. Hence an input tree accepted via the final states D_f must be consumed by in -rule with holes as parameters to transform into given output trees. \square

EXAMPLE 5.2. *Let $M_{sq} = (Q, \Sigma, \Delta, in, R)$ be the 1-hmtt given in EXAMPLE 3.4 and $A_L = (B, \Delta, \beta, B_f)$ the tree automaton where $B = \{b_e, b_o\}$, $\beta(succ, b_e) = b_o$, $\beta(succ, b_o) = b_e$, $\beta(zero) = b_e$ and $B_f = \{b_o\}$. Thus, B accepts all monadic*

trees over $\Delta = \Sigma = \{succ^{(1)}, zero\}$ which have an odd number of occurrences of the symbol $succ$. We now follow the proof of Theorem 5.1 and construct the tree automaton $A = (D, \Sigma, \kappa, D_f)$ which accept an input tree $s \in T_\Sigma$ if and only if A_L accepts $\tau_{M_{sq}}(s)$. Intuitively, A should accept precisely the same set of trees as A_L , because if n^2 is odd then n is odd too, and M_{sq} generates, for an input tree with n occurrences of $succ$, an output tree with n^2 -many $succs$.

A state in D is given by a function from Dom to $2^{B \times B}$ where $Dom = \{in, g\} \cup \{(f, S) \mid S \subseteq B \times B\}$. We now construct the transition function κ . For the symbol $zero$, we have $\kappa(zero) = d_{zero}$ where d_{zero} is a function in D . For the state in we compute $d_{zero}(in) = F(zero)$, because the right-hand side of M_{sq} 's rule for in and $zero$ is the tree $zero$. We follow the definition of F and, since $\beta(zero) = b_e$, we get $F(zero) = \{(b_e, b_o), (b_e, b_e)\}$. Next, we compute the value of $d_{zero}((f, S_1)) = F(y_1 \odot zero)$. By the definition of F , we have

$$\begin{aligned} F(y_1 \odot zero) &= \\ &= \{(b_0, b'_1) \mid (b_0, b_1) \in F(y_1), (b_1, b'_1) \in F(zero)\} \\ &= \{(b_0, b'_1) \mid (b_0, b_1) \in S_1, b_1 = b_e, b'_1 \in B\} \\ &= \{b_0 \mid (b_0, b_e) \in S_1\} \times B. \end{aligned}$$

The value of $d_{zero}(g) = F(succ(\square_1))$ is

$$\begin{aligned} F(succ(\square_1)) &= \\ &= \{(b, b'_1) \mid b = \beta(succ, b_1), (b_1, b'_1) \in F(\square_1)\} \\ &= \{(b, b_1) \mid b = \beta(succ, b_1)\} \\ &= \{(b_o, b_e), (b_e, b_o)\}. \end{aligned}$$

Similarly, we have $\kappa(succ, d) = d_{succ}$ with

$$\begin{aligned} d_{succ}(in) &= d(f, d(g)), \\ d_{succ}((f, S_1)) &= \{(b, b') \mid (b, b'') \in S_1, (b'', b') \in d((f, S_1))\}, \\ d_{succ}(g) &= \{(b_o, b) \mid (b_e, b) \in d(g) \\ &\quad \cup \{(b_e, b) \mid (b_o, b) \in d(g)\}. \end{aligned}$$

The set D_f of final states consists of all functions d such that $d(in) = \{(b_o, b_e), (b_o, b_o)\}$.

We check that the automaton A accepts all odd numbers as follows. It is shown by a simple induction on input numbers that an input $succ^n(zero)$ is accepted on a state d such that

$$d(in) = \begin{cases} \{(b_o, b_e), (b_o, b_e)\} & \text{if } n \text{ is odd} \\ \{(b_e, b_e), (b_e, b_o)\} & \text{if } n \text{ is even} \end{cases}$$

$$d((f, S)) = S_n$$

$$d(g) = \begin{cases} \{(b_e, b_e), (b_o, b_o)\} & \text{if } n \text{ is odd} \\ \{(b_o, b_e), (b_e, b_o)\} & \text{if } n \text{ is even} \end{cases}$$

where S_n is recursively defined by

$$S_0 = \{b \in B \mid (b, b_e) \in S\} \times B$$

$$S_{n+1} = \{(b, b') \mid (b, b'') \in S, (b'', b') \in S_n\}.$$

Hence A accepts all odd numbers. \square

Let us compute the cost of the type inference algorithm presented in Theorem 5.1. More precisely, let us compute the size of the constructed input automaton A , i.e., its number of states. The number of states of A is

$$2^{|B|^{k+1} \cdot |Dom|} \leq 2^{|B|^{k+1} \cdot |Q| \cdot 2^{|B|^{k+1} \cdot m}}$$

where $m+1$ is the maximum rank of all states in Q . This means that the input trees are recognizable by an automaton whose size is at most double-exponential in the size of the automaton A_L recognizing the output type, but still only single-exponential in the

size of the hmtt M where the number m and k of parameters and holes, respectively, are fixed. Thus, for fixed output type, number of parameters, and number of holes, we have a single-exponential inverse type inference algorithm.

Inverse Type Inference for Two-Fold Compositions of Total Deterministic Macro Tree Transducers

Let us roughly explain the complexity of type inference for compositions of two mttts that is obtained through a translation into a macro tree transducer with holes. According to Corollary 4.7 and to the explanation above that corollary, any two-fold composition of total deterministic macro tree transducers can be effectively transformed into an hmtt. This means that we can use Theorem 5.1 in order to perform inverse type inference for such compositions of mttts. The number of holes and parameters of the hmtt that is obtained according to the discussion before Corollary 4.7 can be calculated as follows. Let M_1, M_2 be total deterministic mttts with maximal number of parameters of their states m_1, m_2 , respectively, and with numbers of states p_1, p_2 , respectively. We can decompose $\tau_{M_1} \circ \tau_{M_2}$ into a translation in $\text{MAC}_{\text{det,tot}} \circ \text{TOP}_{\text{det,tot}} \circ \text{YIELD}$, following the results of [5], where the top-down tree transducer has the same number of states as p_2 . We can compose the mtt with the top-down tree transducer using the construction in the proof of Theorem 4.12 of [5]. According to the proof, the obtained mtt has as number of states $p_1 \cdot p_2$ and as maximal number of parameters $m_1 \cdot p_2$. Following Lemma 4.5 we construct a macro tree transducer with holes which has the same number of states and parameters and has m_2 -many holes. This means that for fixed output type and fixed numbers of parameters of M_1 and M_2 we obtain, by going through hmtts, an automaton A for the input type which has double-exponential (in p_2) many states (according to the formula shown below the proof of Theorem 5.1). In contrast, doing inverse type inference for M_2 and then using the obtained automaton to do inverse type inference for M_1 gives a number of states of the resulting input automaton that is 3-exponential (in p_2).

Note that for XML types, forward type inference fails because the image $T(R)$ of the input type R under T is in general *not* regular [19]. Only for very restricted kinds of translations which only translate each input nodes at most once, it holds that regular tree languages are preserved by the transformation (these are essentially the *linear* top-down and *linear* bottom-up tree transformations), see, e.g., [4]. However, there do exist forward inference procedures for certain classes of tree transformations which yield descriptions of possible output trees, beyond the regular tree languages. For instance, the image of an input type by a macro tree transducer that is linear in its input variables (but with no restriction on the use of parameters), is a context-free tree language and can effectively be inferred. This idea was used in [12] to give type checking algorithms for classes of macro tree transducers, based on forward inference.

6. Conclusion

We introduced macro tree transducer with holes (hmtt). This is an extension of macro tree transducers by holes and their application. The resulting class of translations is strictly larger than those of macro tree transducers and of macro forest transducers. Nevertheless, inverse type inference for an hmtt can still be done with the same complexity as for just one single mtt. We characterized hmtts by compositions of one ordinary mtt with a “YIELD-mapping” which carries out tree substitution. This allowed us to show that deterministic hmtts have the same expressiveness as two-fold compositions of deterministic macro tree transducers. Furthermore, we obtained a new inverse type inference algorithm for two-fold compositions of total deterministic mttts, which has double-exponential time complexity (if the output type and numbers of parameters of

the transducers are fixed) and improves the best known previous bound of triple-exponential (tower of exponents of height 3).

By Corollary 4.7 we know that even the composition of two (partial) deterministic mttts can be realized by one deterministic hmtt. However, as mentioned before that corollary, the size of the deterministic hmtt can be exponential in the numbers of parameters of the involved deterministic mttts. It remains therefore unclear whether our double-exponential inverse type inference procedure for two-fold compositions of total deterministic mttts can be extended to two-fold compositions of partial deterministic mttts.

Acknowledgments

We are thankful to Joost Engelfriet for helpful comments and for pointing out several flaws in an earlier version of this paper.

References

- [1] G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, 27:21–39, 2002.
- [2] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Technical Report Technical Report HKUSTTCSC-2001-0, The Hongkong University of Science and Technology, 2001.
- [3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: <http://www.grappa.univ-lille3.fr/tata>
- [4] J. Engelfriet. Bottom-up and top-down tree transformations — a comparison. *Math. Systems Theory*, 9:198–231, 1975.
- [5] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comp. Syst. Sci.*, 31:71–146, 1985.
- [6] M. Fischer. *Grammars with macro-like productions*. PhD thesis, Harvard University, Massachusetts, May 1968.
- [7] A. Frisch and H. Hosoya. Towards practical typechecking for macro tree transducers. Technical Report Research Report RR-6107, INRIA, 2007.
- [8] Z. Fülöp and H. Vogler. *Syntax-Directed Semantics – Formal Models based on Tree Transducers*. EATCS Monographs in Theoretical Computer Science (W. Brauer, G. Rozenberg, A. Salomaa, eds.). Springer-Verlag, 1998.
- [9] D. Knuth. Semantics of context-free languages. *Math. Systems Theory*, 2:127–145, 1968. (Corrections in *Math. Systems Theory*, 5:95–96, 1971).
- [10] S. Maneth, A. Berlea, T. Perst, and H. Seidl. XML type checking with macro tree transducers. In *Proc. PODS 2005*, pages 283–294. ACM Press, 2005.
- [11] S. Maneth and F. Neven. Recursive structured document transformations. In R. Connor and A. Mendelzon, editors, *Revised Papers DBPL’99*, volume 1949 of *LNCS*, pages 80–98. Springer-Verlag, 2000.
- [12] S. Maneth, T. Perst, and H. Seidl. Exact XML type checking in polynomial time. In *ICDT*, pages 254–268, 2007.
- [13] W. Martens and F. Neven. On the complexity of typechecking top-down XML transformations. *Theor. Comput. Sci.*, 336(1):153–180, 2005.
- [14] T. Milo, D. Suci, and V. Vianu. Typechecking for XML transformers. *J. Comp. Syst. Sci.*, 66:66–97, 2003.
- [15] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.*, 5(4):660–704, 2005.
- [16] T. Perst and H. Seidl. Macro forest transducers. *Inf. Process. Lett.*, 89:141–149, 2004.
- [17] W. Rounds. Mappings and grammars on trees. *Math. Systems Theory*, 4:257–287, 1970.

- [18] D. Suciu. Typechecking for semistructured data. In G. Ghelli and G. Grahne, editors, *Database Programming Languages, 8th International Workshop – DBPL'2001, Revised Papers*, volume 2397 of *LNCS*, pages 1–20. Springer-Verlag, 2002.
- [19] D. Suciu. The XML typechecking problem. *SIGMOD Record*, 31:89–96, 2002.
- [20] J. Thatcher. Generalized² sequential machine maps. *J. Comp. Syst. Sci.*, 4:339–367, 1970.
- [21] V. Vianu. A Web Odyssey: From Codd to XML. In *Proc. PODS'2001*, pages 1–15. ACM Press, 2001.