Macro Forest Transducers

Thomas Perst
Helmut Seidl
Institut für Informatik, Technische Universität München, 85748 Garching, Germany, {seidl,perst}@in.tum.de

Abstract XML documents conceptually are not trees, but forests. Therefore, we extend the concept of macro tree transducers (mtt’s) to a transformation formalism of forests, macro forest transducers (mft’s). We show that mft’s form a strict superset of mtt’s operating on forests (represented as binary trees). On the other hand, the transformation of every mft can be simulated by the composition of two mtt’s. Although macro forest transducers are more powerful, the complexity of inverse type inference, i.e., computing the pre-image of a recognizable language, is almost the same as for tree transducers.

Keywords: XML; Macro tree transducers; inverse type inference.

1 Introduction

XML is a markup language for storing structured data in a sequential format. It is originally designed for the use in document processing [22] and it is believed to become the de facto data exchange standard for the Internet [15, 17]. Data in XML are grouped into elements delimited by tags and elements can be nested [22, 17].

A key idea of XML is that documents can be type checked. The type is given by a document type definition (DTD) [22] or, more generally, a schema [23, 2] describing a recognizable language. Usually, type checking is done dynamically during parsing time. A validating parser checks if the current document conforms to the specified schema, i.e., the parser decides whether the document is valid or not. In many cases, documents are automatically generated, e.g., as the result of a transformation induced by stylesheet processors [1, 21, 8]. In order to guarantee type safety in such an environment, we clearly could repeat the type checking procedure after every transformation phase. The obvious drawback of such dynamic type checking is, however, that it is time consuming and detects errors at transformation time only.

Therefore, we search for methods to statically guarantee type safety. Here, we are given two types $R_{in}$ and $R_{out}$, the input and output type, respectively. For a given transformer $P$ we want to determine, if $P$ produces for all inputs of $R_{in}$ only documents that are valid with respect to $R_{out}$. A naive solution to this type checking problem is to infer the set $R'$ of outputs produced by $P$ for inputs of type $R_{in}$. Having inferred this set, the type checking problem for $P$ is reduced to the test, whether $R'$ is included in $R_{out}$. The applicability of this approach, however, is quite limited. As observed by several authors, even for simple transformations $R'$ is no longer recognizable [19, 12, 17, 18].

Sometimes it turns out, though, that the inverse problem is easier tractable, that is: Given a program $P$ and an output type $R_{out}$, to determine the exact set $R'$ of all inputs $d$, for which $P(d)$ is contained in $R_{out}$. Computing $R'$ is called the inverse type inference problem [12, 19, 4]. Then the type checking problem is answered by the inclusion test $R_{in} \subseteq R'$.

Various abstract models have been proposed to describe transformations of XML. Tozawa uses finite tree automata as document types [19] and infers the inverse type on the basis of the operational semantics of XSLT expressions, which describe a subset of XSLT (Extensible Stylesheet Language Transformation). More or less, these transformations correspond to top-down finite state transductions. Milo, Suciu and Vianu use k-pebble tree transducers to model the transformations [12]. These transducers are allowed to arbitrarily traverse their input and use pebbles to mark already visited nodes. At the leaves of the output new com-
putations can be spawned.

Recently, also macro tree transducers \[3, 5, 7\] have been considered as a model for XML transductions \[20, 9, 4\]. Macro tree transducers (mtt’s) originate from macro grammars \[6\] and are top-down finite state tree transducers which additionally are equipped with parameters to accumulate auxiliary results. At each computation step the values of the parameters can be used to build up the output. Similar to \(k\)-pebble transducers, they transform forests by referring to their representations as trees. The advantage of macro tree transducers is that they are well-understood and have a rich theory \[7\]. In particular, they are closed under regular look-ahead \[5\]. The major deficiency, though, is that, due to the encoding as trees, concatenation of intermediate forests is not supported.

In this paper we therefore enhance macro tree transducers by adding concatenation. Because the resulting model works on forests rather than on trees, we call it macro forest transducers (mft’s). We formally prove that the computational power is indeed increased. The resulting transformations can, however, be realized by the composition of two mtt’s. Thus, by the results of \[9\], the translations of deterministic mft’s can be computed in time linear in the sum of the sizes of the corresponding input and output forests. For the new intermediate class of transformations, we furthermore observe that inverse type inference (for a fixed recognizable output specification and a bounded number of parameters) is \textsc{dextime}\nobreakdash-complete — thus, meeting the corresponding complexity bounds for mtt’s and even ordinary top-down transductions \[11\].

The paper is organized as follows: In the next section we introduce basic notations concerning trees and forests. Then we define the new notion of macro forest transducers. Then we illustrate the computational power of our transducers by comparing them with macro tree transducers and relate both to the corresponding transducer classes without parameters. The last section is concerned with inverse type inference.

\section{Preliminaries}

Because XML transformation languages are defined on forests, rather than on trees, we are interested in forests \(f\) over an alphabet \(\Sigma\). The children of a node are a sequence of arbitrarily many trees, that is, a forest.

\begin{definition}
Let \(\Sigma\) be a finite alphabet. Then the set \(\mathcal{F}_\Sigma\) of forests \(f\) over \(\Sigma\) is recursively defined as:
\[
  t \ ::= \ a(f), \ a \in \Sigma \\
  f \ ::= \ \epsilon \mid tf,
\]
where \(\epsilon\) denotes the empty forest. For simplicity, we also write \(a\) for \(a\). \Halmos
\end{definition}

Sometimes forests are also called \(\Sigma\)-hedges, e.g., by M. Murata \[14\] and A. Tozawa \[19\].

On the other hand, macro tree transducers work on ranked trees. This means that the number of the children of a node is determined by the rank of the symbol at that node. We assume here that \(\epsilon\) is a unique symbol of rank 0 while every symbol in \(\Sigma\) has rank 2.

\begin{definition}
The set \(\mathcal{B}_\Sigma\) of binary trees over the alphabet \(\Sigma\) is defined as:
\[
  t \ ::= \ \epsilon \mid a(t_1, t_2), \ a \in \Sigma \text{ and } t_1, t_2 \in \mathcal{B}_\Sigma,
\]
where the symbol \(\epsilon\) identifies the leaves. \Halmos
\end{definition}

In fact, binary trees are in one-to-one correspondence to forests: For a tree \(a(t_1, t_2)\) we understand the root label \(a\) together with the left successor \(t_1\) as the representation of an (unranked) tree with a root named \(a\) while the right successor \(t_2\) represents the right context of this tree. Figure 1 shows this correspondence for the tree:

\[
a(a(d(\epsilon, e(\epsilon, e)), b(\epsilon, c(b(\epsilon, c(\epsilon, e)), e)), \epsilon))\)
\]

In order to keep the binary tree readable, we omitted all occurrences of the leaf symbol \(\epsilon\). The encoding of forests by binary trees, however, is too inflexible: in the XML view of documents, the content of an element is a \textit{sequence} of elements. Accordingly, any reasonable document transformer supports concatenation of arbitrary forests. Therefore, we prefer to work with forests directly and generalize the notion of macro tree transducers appropriately.

\section{Preliminaries}

Because XML transformation languages are defined on forests, rather than on trees, we are interested in forests \(f\) over an alphabet \(\Sigma\). The children of a
Definition 3 A macro forest transducer is a tuple $M = (Q, \Sigma, \Delta, q_0, R)$, where

- $Q$ is the set of states (or procedures);
- $\Sigma$ and $\Delta$ are finite alphabets with $Q \cap (\Sigma \cup \Delta) = \emptyset$, called the input and the output alphabet, respectively;
- $q_0 \in Q$ is the initial state;
- $R$ is a finite set of rules of the form:

$$
p(x_1, y_1, \ldots, y_n) \rightarrow f, \text{ or } \quad p(x_1) x_2, y_1, \ldots, y_n \rightarrow f,
$$

with $a \in \Sigma$ and $p \in Q$. The right-hand sides are of the following form $f$:

$$
t := b(f) \quad f := q(x_1, f_1, \ldots, f_m) \quad | \quad \epsilon \quad | \quad y_j \quad | \quad t f \quad | \quad f_1 f_2
$$

where $q \in Q$, $b \in \Delta$ and $i \in \{1, 2\}$, $j = 1, \ldots, n$, $n, m \in \mathbb{N}$, $n, m \geq 0$. Moreover, the right-hand sides for empty input forests $\epsilon$ must not contain occurrences of the variables $x_1$ or $x_2$.

In case of several rules for the same $p$ and the same symbol $a$ (or $e$) we also write: $f_1 | \ldots | f_k$ to enumerate all occurring right-hand sides. \(\triangleright\)

Note that states/procedures $q$ may differ in their ranks, i.e., the number of their accumulating parameters. We write $Q^k$ for the procedures with $k$ such parameters.

The transducer works as follows: Each state can be considered as a procedure which recurses over the first argument while the additional arguments serve as accumulating parameters. Instead of defining the translation by means of a derivation relation, we here prefer to take a denotational point of view and interpret each procedure by a function mapping input forests (together with tuples of actual parameters) to sets of output forests. These functions are mutually recursive and defined by structural induction on input forests.

Definition 4 Let $M = (Q, \Sigma, \Delta, q_0, R)$ be an arbitrary, but fixed, macro forest transducer. The meaning of a state $q \in Q$ having $k \geq 0$ accumulating parameters is a function:

$$[[q]] : \mathcal{F}_\Sigma \times (2^{2^\Delta})^k \rightarrow 2^{2^\Delta}$$

These functions are inductively defined by:

$$[[q]] (a(s_1) s_2, S_1, \ldots, S_k) = \left[ [f_1] \right] \sigma \rho \lor \ldots \lor \left[ [f_m] \right] \sigma \rho$$

for $a \in \Sigma$ where $\sigma(x_i) = s_i$, $i = 1, 2$, $\rho(y_j) = S_j$ for $j = 1, \ldots, k$, and

$$q(a(x_1) x_2, y_1, \ldots, y_k) \rightarrow f_1 | \ldots | f_m \in R$$

and

$$[[q]] (\epsilon, S_1, \ldots, S_k) = \left[ [f_1] \right] \emptyset \lor \ldots \lor \left[ [f_m] \right] \emptyset \rho$$

where $\emptyset$ is the empty assignment, $\rho(y_j) = S_j$ for $j = 1, \ldots, k$, and

$$q(\epsilon, y_1, \ldots, y_k) \rightarrow f_1 | \ldots | f_m \in R$$

Here, $[[\cdot]] \sigma \rho$ denotes the evaluation of a right-hand side expression w.r.t. the bindings $\sigma, \rho$ of the formal parameters $x_i$ and $y_j$, respectively. Thus,

$$[[\epsilon]] \sigma \rho = \{ \epsilon \} \quad [y_j] \sigma \rho = \rho(y_j) \quad [b(f_1) f_2] \sigma \rho = \{ b(s_1) s_2 | s_1 \in [f_1] \sigma \rho \} \quad [f_1 f_2] \sigma \rho = \{ s_1 s_2 | s_1 \in [f_1] \sigma \rho \} \quad [q(x_1, f_1, \ldots, f_k)] \sigma \rho = \left[ [q] \right] (\sigma(x_1), [f_1] \sigma \rho, \ldots, [f_k] \sigma \rho)$$

\(\triangleright\)

The transformation induced by the transducer $M = (Q, \Sigma, \Delta, q_0, R)$ is the function $t_M : \mathcal{F}_\Sigma \rightarrow 2^{2^\Delta}$ induced by the start procedure $q_0$ on input.
forests and empty forests in the accumulating parameters:
\[ \tau_M(f) = [q_0](f, \{\epsilon\}, \ldots, \{\epsilon\}) \]
Analogously to Theorem 3.17 of [5], our denotational semantics here is equivalent to an operational semantics based on rewriting with a call-by-name mode for parameter passing. Note that, since procedure calls may be nested, the order in which they are unfolded indeed matters. Call-by-name evaluation is achieved by fixing the order of rule application to be outside-in. This means that a rule is only applied to a procedure call if it occurs outside, i.e., it has no ancestor labeled by a procedure call.

The class of all macro forest transformations is denoted by \textit{FMAC}.

The mtt’s as invented by Engelfriet and Vogler [3, 5] differ from our forest transducers only in that they are defined on trees. Technically, this means that right-hand sides of rules may not contain subexpressions \( f_1 f_2 \) where \( f_1 \) is not of the form \( b(f') \). Let \textit{MAC} denote the class of all transformations that can be performed by macro tree transducers.

**Example 5** Let the macro forest transducer \( M_r = (Q, \Sigma, \Delta, q_0, R) \) be defined by
- \( Q = \{q_0\} \)
- \( \Sigma = \Delta \) finite alphabets
- \( R \) consists of the rules:
  1. \( q_0(\epsilon, y_1) \rightarrow y_1 \)
  2. \( q_0(a(x_1) x_2, y_1) \rightarrow q_0(x_2, a(q_0(x_1, \epsilon)) y_1), \) for all \( a \in \Sigma \).

This transducer takes a forest \( f = t_1 t_2 \ldots t_n \) over \( \Sigma \) as input and produces as output a forest \( f' = t'_n \ldots t'_2 t'_1 \) where each \( t'_i \) is the mirror image of the corresponding \( t_i \).

The rules from the third item take the first tree of the input forest and concatenate its mirror image to the front of the accumulating parameter \( y_1 \). Applying these rules recursively, reverses the order on the current level.

\[ \text{Lemma 7} \quad \text{There is a macro forest transducer } M \text{ such that, for infinitely many } f \in \mathcal{F}_\Sigma, \text{ bht}(s) \geq 2^{3^{|s|}} \text{ for every } s \in \tau_M(f). \]

\[ \text{Proof.} \quad \text{Consider the total and deterministic mft } M = (Q, \Sigma, \Delta, q, R) \text{ where } Q = \{q_0, q\}, \Sigma = \{a\}, \Delta = \{b\}, \text{ and } R \text{ consists of the rules:} \]
\[
\begin{align*}
q_0(a(x_1) x_2) & \rightarrow q(x_2, q(x_2, b)) \\
q_0(\epsilon) & \rightarrow b b \\
q(a(x_1) x_2, y_1) & \rightarrow q(x_2, q(x_2, y_1)) \\
q(\epsilon, y_1) & \rightarrow y_1 y_1
\end{align*}
\]

The last rule concatenates the parameter \( y_1 \) to itself, thus doubling the binary height of the accumulated output forest. Let \( f_k = a^k, k \geq 0 \). Assume that for \( r, m \in \mathbb{N}, \{a^k, \{b^m\}\} = \{b^m\} \). We claim:
\[ m = 2^{k \cdot r} \]
Since $\text{bht}(f_a) = \text{bht}(a^n) = n$ and $\text{bht}(b^m) = m$ the assertion of the lemma follows. In order to prove our claim, we proceed by induction on $k$. For $k = 0$,

$$[q](\epsilon, \{b', b''\}) = \{b' b''\} = \{b^{2r}\}$$

and the assertion follows since $2^{2^0} = 2^1 = 2$.

For $k > 1$,

$$[q](a a^{k-1}, b') = [q](a^{k-1}, [q](a^{k-1}, \{b'\}))$$

which, by induction hypothesis applied to both procedure calls, results in a singleton set $\{b^m\}$ with

$$m = 2^{2^{k-1}} \cdot 2^{2^{k-1}} \cdot r = 2^{2^k} \cdot r$$

This proves the claim.

**Theorem 8** \ MAC $\subseteq$ \ FMAC

**Proof.** The height property for \ MAC says that for every mtt $M \in$ \ MAC there exists $c > 0$ such that for all $f \in \mathcal{F}_C$ and all $s \in \tau_M(f)$, $\text{bht}(s) \leq c^{\text{bht}(f)}$ [7]. Hence, the translation of the mtt $M$ from the last lemma cannot be expressed by any mtt.

Similarly, every non-empty set of images produced by a $k$-pebble transducer for an input $f$ contains some output of binary height polynomial in the size of $f$ [13]. Intuitively, the reason is that these transducers build up their outputs node by node and neither support parameters nor concatenation. We conclude that $k$-pebble transducers cannot simulate the mtt $M$ either.

On the other hand, we show that every mtt can be simulated by the composition of two mtt’s. Let $M$ denote an mtt. The first macro tree transducer $M_1$ essentially executes the transitions of $M$ — with the only difference that each concatenation $f_1 f_2$ in right-hand sides of $M$ is replaced by an application of a concatenation symbol “@”. Formally, if $f$ is a right-hand side of $M$, then the corresponding right-hand side of $M_1$ is obtained as $A[f]$ where:

$$A[\epsilon] = \epsilon$$

$$A[b(f_1) f_2] = b(A[f_1]) A[f_2]$$

$$A[q(x_1, f_1, \ldots, f_m)] = \tilde{q}(x_1, A[f_1], \ldots, A[f_m])$$

$$A[y_j] = y_j$$

$$A[f_1 f_2] = @A[f_1] A[f_2]$$

where every state $q$ of $M$ is simulated by the corresponding state $\tilde{q}$ of $M_1$.

Let $\text{eval}(f)$ denote the transformation which evaluates the concatenation symbols “@”. Thus,

$$\text{eval}(\epsilon) = \epsilon$$

$$\text{eval}(@f_1 f_2) = \text{eval}(f_1) \text{eval}(f_2)$$

Moreover, we extend the transformation $\text{eval}(\cdot)$ to sets of forests by:

$$\text{eval}(S) = \{\text{eval}(s) \mid s \in S\}$$

By structural induction on input forests, one verifies for all states $q$ of $M$ that:

$$[q](f, S_1, \ldots, S_k) = \text{eval}(\{q\}(f, S_1, \ldots, S_k))$$

where $S_i = \text{eval}(S_i), i = 1, \ldots, k$. This assertion follows from proving for every sub-forest $s$ occurring in a right-hand side of a rule of $M$ that:

$$[s] \sigma \rho' = \text{eval}([A[s]] \sigma \rho)$$

where $\rho'(y_j) = \text{eval}(\rho(y_j))$ for $j = 1, \ldots, k$. Hence,

$$[\langle y_0 \rangle f, \langle \epsilon \rangle, \ldots, \langle \epsilon \rangle] = \text{eval}(\{\langle y_0 \rangle f, \langle \epsilon \rangle, \ldots, \langle \epsilon \rangle\})$$

Therefore, $\tau_M(f) = \text{eval}(\tau_{M_1}(f))$ for every $f \in \mathcal{F}_C$.

In the second step, we show that the transformation $\text{eval}(\cdot)$ itself can be implemented by another mtt $M_\oplus$. $M_\oplus \in$ \ MAC has just one state $q$ and the following rules:

1. $q(\epsilon, y_1) \rightarrow y_1$;
2. $q(b(x_1, x_2, y_1) \rightarrow q(x_1, \epsilon)) q(x_2, y_1)$ for all output symbols $b$ of $M$;
3. $q(@x_1, x_2, y_1) \rightarrow q(x_1, q(x_2, y_1))$.

By definition, $M_\oplus$ is total and deterministic. By induction on the structure of input forests $f$, we verify that for all forests $s$:

$$[q](f, \{s\}) = \{\text{eval}(f) s\}$$

We conclude that $\tau_{M_\oplus}(f) = \{\text{eval}(f)\}$ which we wanted to prove.

Let $\text{MAC}^2$ denote the class of all transductions that can be performed by the composition of two macro tree transducers. Thus we have proved:
Theorem 9  \( \text{FMAC} \subseteq \text{MAC}^2 \)

As a corollary we obtain a height property for FMAC: For every mft \( M \) there is a constant \( c > 0 \) such that for every \( f \in \mathcal{F}_\Sigma \) and \( s \in \tau_M(f) \),
\[
\text{bht}(s) \leq 2^{c \cdot \text{bht}(f)}
\]

The decomposition of an mft into two mtt’s is not arbitrary — but (for a given \( \Delta \)) always uses the same mtt \( M_\Delta \) to evaluate the auxiliary concatenation symbols. For every such transducer \( M_\Delta \), we observe:

Lemma 10  For every forest \( f \in \mathcal{F}_\Delta \cup \{\epsilon\} \) and \( \{s\} = \tau_{M_\Delta}(f) \),
\[
\text{ht}(s) \leq \text{ht}(f)
\]

Proof. Recall that the transducer \( M_\Delta \) is total and deterministic. Assume \( \{s\} = [q](\{f, s'\}) \) where \( f \in \mathcal{F}_\Delta \cup \{\epsilon\} \). We claim:
\[
\text{ht}(s) \leq \max\{\text{ht}(f), \text{ht}(s')\}
\]

We proceed by structural induction on \( f \). If \( f = \epsilon \), then \([q](f, \{s'\}) = \{s'\}\). Thus, \( s = s' \) and the claim follows. If \( f = @ (f_1) f_2 \), then \([q](f, \{s'\}) = [q](f_1, [q](f_2, \{s'\}))\). By induction hypothesis for \( f_2 \), \( \text{ht}(s_2) \leq \max\{\text{ht}(f_2), \text{ht}(s'_2)\}\) for \( \{s_2\} = [q](f_2, \{s'\})\). By applying the induction hypothesis now to \( f_1 \) we obtain:
\[
\text{ht}(s) \leq \max\{\text{ht}(f_1), s_2\}
\]

which we wanted to prove. Finally, assume that \( f = a(f_1) f_2 \) for some \( a \in \Delta \). Then
\[
[q](f, \{s'\}) = \{a(s_1), s_2\} \quad \text{where}
\{s_1\} = [q](f_1, \{\epsilon\}) \quad \text{and}
\{s_2\} = [q](f_2, \{s'\})
\]

and the claim follows from the induction hypothesis applied to \( f_1 \) and \( f_2 \).

Lemma 10 gives us a single-exponential upper bound on the height of output forests of mft’s:

Theorem 11  For every mft \( M \) there is a constant \( c > 0 \) such that for every \( f \in \mathcal{F}_\Sigma \) and \( s \in \tau_M(f) \),
\[
\text{ht}(s) \leq 2^{c \cdot \text{bht}(f)}
\]

In [7], an example mtt is provided which translates monadic trees of height \( n \) into monadic trees of height \( 2^n \). Iterating this mtt, a translation in \( \text{MAC}^2 \) is obtained which, by Theorem 11 cannot be realized through one mtt. Thus, we conclude:

Corollary 12  \( \text{FMAC} \subseteq \text{MAC}^2 \).

In 2001, Tozawa has considered a transformational formalism [19] which essentially equals our notion of macro forest transformations: without, however, accumulating parameters. Let us call such transformations \textit{top-down forest transformations} or \textit{top-down tree transformations} — depending on whether the right-hand sides make use of concatenation or not. Let \( \text{FTOP} \) and \( \text{TOP} \) denote the respective classes. The transducers from \( \text{FTOP} \) are also closely related to the top-down transducers of Maneth and Neven [10] (but slightly more general). The next corollary summarizes the inclusions among these classes.

Theorem 13  \( \text{TOP} \subseteq \text{FTOP} \subseteq \text{MAC} \)

Proof. (1) While the binary height of the output of a transducer \( T_1 \in \text{TOP} \) is linear bounded by the binary height of the input [7], the binary height of an output forest of \( T_2 \in \text{FTOP} \) can be exponentially large in the binary height of the input [16].

(2) Let \( T \in \text{FTOP} \). We define an mtt \( M \in \text{MAC} \) that simulates the transducer \( T \):

- a rule \( q(a(x_1) x_2) \rightarrow f \) of \( T \) is simulated by \( \hat{q}(a(x_1) x_2, y_1) \rightarrow C[f] y_1 \)
- and a rule \( q(\epsilon) \rightarrow f \) is simulated by \( \hat{q}(\epsilon, y_1) \rightarrow C[f] y_1 \)

where \( C[\cdot] \cdot \) is defined by:
\[
C[\cdot] c = c
\]
\[
C[a(f_1) f_2] c = a(C[f_1] c) (C[f_2] c)
\]
\[
C[q(x_1)] c = \hat{q}(x_1, c)
\]
\[
C[f_1 f_2] c = C[f_1] (C[f_2] c)
\]

By structural induction on input forests \( f \in \mathcal{F}_\Delta \), one proves that for all states \( q \) of \( T \),
\[
([q](f)) \cdot S = [q](f, S)
\]

for all \( S \subseteq \mathcal{F}_\Delta \). Here, we use the operator “\( \cdot \)” to denote element-wise concatenation of sets of forests. Then in particular,
\[
[q](f) = [q](f, \{\epsilon\})
\]
The claim follows by verifying that for every subforest $s$ occurring in a right-hand side of $T$ that

$$ ([s] \sigma \emptyset) \cdot S = [C[s] y_1] \sigma \rho $$

where $\rho(y_1) = S$.

In order to prove the inclusion of $\text{FTOP}$ in $\text{MAC}$ strict, we recall from Theorem 9 that every translation in $\text{FTOP}$ can be decomposed into an ordinary top-down transduction from $\text{TOP}$ followed by the mtt $M_\alpha$. Thus, we deduce from lemma 10 that for every transducer $M \in \text{FTOP}$ there is a constant $c > 0$ such that for every input forest $f$ and $s \in \tau_M(f)$,

$$ \text{ht}(s) \leq c \cdot \text{ht}(f) $$

Using the already mentioned lower-bound example from [7], we conclude that not every macro tree transducer can be simulated by a top-down forest transducer from $\text{FTOP}$. 

5 Inverse Type Inference

Often XML documents are transformed by stylesheet processors. XML transformers may, e.g., create a database view or convert an XML document into HTML. Well-formed views or valid HTML output are given by recognizable types $R_{\text{out}}$. An important question is whether the computation of the view or the converter program work correctly. Given that the input documents are of type $R_{\text{in}}$, we want to know whether for a (possibly non-deterministic) transformer $M$, $M(s) \subseteq R_{\text{out}}$ for all $s \in R_{\text{in}}$. In the introduction, this has been called the type checking problem for the transformer $M$.

Here, we are interested in the inverse type inference problem. Given a transformer $M$, we want to determine the set $P$ of all input forests which can be processed and always result in legal output, i.e.,

$$ P = \{ f \in \mathcal{F}_\Sigma | \tau_M(f) \cap R_{\text{out}} = \emptyset \}, $$

where $R_{\text{out}}$ denotes the complement of the set of legal outputs $R_{\text{out}}$. This set $P$ is the complement of the pre-image $\tau_M^{-1}(R_{\text{out}})$ of $R_{\text{out}}$ w.r.t. $M$.

The main result of [12] is that the inverse type inference problem is solvable for pebble tree transducers. It is also well known in tree transducer theory that the pre-image of a recognizable tree language w.r.t. a macro tree transducer is again recognizable and can be effectively computed [5]. Thus, the inverse type inference problem is solvable for all transformations from $\text{MAC}$. Because mft’s can be simulated by the two-fold composition of mtt’s, we conclude that the pre-image $\tau_M^{-1}(R_{\text{out}})$ of a mft $M$ is recognizable as well and can be effectively constructed. The proof for the next Theorem proceeds along the lines of the one given at the end of [4]. We give the explicit construction to illustrate how the extra feature of concatenation can be treated directly, and to exhibit that the complexity meets the complexity of the corresponding construction for mtt’s.

**Theorem 14** Let $M \in \text{FMAC}$ be a mft of bounded rank and $R'$ be a recognizable set. Then,

1. The pre-image $\tau_M^{-1}(R')$ is recognizable.
2. The computation of the pre-image for a fixed $R'$ can be performed in deterministic exponential time.
3. Deciding emptiness of the pre-image of recognizable languages w.r.t. mft’s is dexpTime-hard.

**Proof.** Let $M = (Q, \Sigma, \Delta, q_0, R)$ be a mft. Let $A' = (B, \Delta, \beta, b_0, F_B)$ be an ordinary deterministic right-to-left bottom-up forest automaton on the binary encodings with the set $F_B$ of final states and $\beta : \Delta \times B \times B \rightarrow B$, such that $L(A') = R'$. A right-to-left automaton works as follows: the result states are computed from right to left. Computation always starts with an initial state $b_0$ for the empty forest. The state for the forest $f_i f_1$ is obtained from the states $b_i$ for the forests $f_i$ by evaluating $\beta(a, b_1, b_2)$.

From the automaton $A'$ and the mft $M$, we construct a deterministic automaton $A = (D, \Sigma, \delta, d_0, F_D)$ recognizing $\tau_M^{-1}(R')$. The key issue of $A$ is to simulate the accepting computations of $A'$ on the right-hand sides of the transformation rules of $M$.

Let $\text{Dom}$ denote the set of all states $q \in Q$ of the mft $M$ combined with all suitable tuples of relations:

$$ \text{Dom} = \{(q, S_1, \ldots, S_n) | q \in Q^{(n)}, S_i \subseteq B \times B \}, $$

where $n$ is the maximal rank of states. Then, the set $D$ of states of $A$ is given by:

$$ D = \text{Dom} \rightarrow 2^{B \times B} $$
Intuitively, a relation $S \subseteq B \times B$ describes the set of all possible state transitions of the automaton for $R'$ on the forests from a given set of outputs. A state $d \in D$ assigned to an input forest $f$ records for every procedure $q$ of the mft and every possible sequence $S_1, \ldots, S_n$ of such effect descriptions for the accumulating parameters the set of all state transitions which can be obtained by calling $q$ on $f$ and actual parameters satisfying the $S_i$.

Given this set of states $D$, we define:

- $d_0(q, S_1 \ldots, S_n) = \bigcup_{i=1}^k \overline{[f_i]^{\rho}}$ if $q(a_1 x_2, y_2, \ldots, y_n) \rightarrow f_1 | \ldots | f_k$ in $M$
- $\rho(y_i) = S_i, 1 \leq i \leq n$;

- $\delta(d, d_1, d_2) = d$ where $d(q, S_1, \ldots, S_n) = \bigcup_{i=1}^k \overline{[f_i]^{\sigma \rho}}$

if $q(a(x_1 x_2, y_2, \ldots, y_n) \rightarrow f_1 | \ldots | f_k$ in $M$;
- $\sigma(x_i) = d_i, i = 1, 2$, and $\rho(y_i) = S_i, 1 \leq i \leq n$;

- $D_F$ consists of all mappings $d$ such that the relation $d(y_0, 1, \ldots, 1)$ contains a pair $(b, b_0)$ with $b \in F_B$. Here, 1 denotes the relation

$$\{(b, b) \mid b \in B\}$$

The mapping $\overline{[\cdot]^{\cdot}}$ which we have used to define $d_0$ and $\delta$ simulates the computation of $A'$ on the right-hand sides of the transformation rules:

$$\overline{[x_1]^{\cdot}} = 1 \subseteq B \times B$$
$$\overline{[y_1]^{\cdot}} - \rho(y_1)$$
$$\overline{[f_1 f_2]^{\cdot \sigma \rho}} = ([f_1]^{\cdot \sigma \rho} \circ ([f_2]^{\cdot \sigma \rho})$$
$$\overline{\{a(b_1 b_2) \mid \exists s_1, s_2 \in B : (s_1, b_0) \in S_1, (s_2, b_2) \in S_2, b_1 = \beta(a, s_1, s_2)\},}$$

where $S_1 = \overline{[f_1]^{\cdot \sigma \rho}}$ and $S_2 = \overline{[f_2]^{\cdot \sigma \rho}}$ and \(\circ\) denotes the composition of relations.

$$\overline{p(x_j, f_1, \ldots, f_m)^{\cdot \sigma \rho}} = (\sigma(x_j)) \overline{p([f_1]^{\cdot \sigma \rho}, \ldots, [f_m]^{\cdot \sigma \rho})}$$

The number of states of $A$ is at most:

$$2^{|B|^2 |\text{Dom}|} \leq 2^{|B|^2 |Q| 2^{|B|^2}}.$$

Therefore, the pre-image is double exponential in the size of the automaton $A'$ representing the output type — but only single-exponential in the size of the transducer $M$. Since the construction can be performed in time polynomial in the size of the computed automaton $A$, the upper complexity bound follows (given that $n$ is bounded by some constant). For the lower bound, we recall that the lower bound holds already for TOP [11] and thus also for FMAC.

6 Conclusion

We extended the concept of macro tree transducers to a transformation formalism for forests. These macro forest transducers form a strict superclass of mtt’s. Although macro forest transducers are in some sense more powerful, the decidability and complexity results for inverse type checking, i.e., computing the pre-image of a recognizable language, are almost the same as for mtt’s.

Macro forest transducers support concatenation of forests as basic operation and thus are closely related to XML’s data model. Future work may extend macro forest transducers to a formal model for style sheet processors such as XSLT or fxt [1]. A drawback of the algorithm for the inverse type inference is it’s complexity. In [16], we proposed a typed macro language for XML which is interpreted during parsing time of a document. At the expense of rejecting harmless inputs, a document type for safe input documents is inferred. The type-safeness of macro expansion can thus be guaranteed by validating input documents with unexpanded macro calls. It is an interesting question whether such approximative type inference can also be designed for more powerful Xmt-transducers.

Acknowledgment We thank Sebastian Maneth for his comments on mtt’s and the composition results. We are also grateful to the anonymous referee for his extremely precise and valuable comments and his suggestion to distinguish two notions of height.

References


