The Complexity of Translation Membership for Macro Tree Transducers

Kazuhiro Inaba
The University of Tokyo
kinaba@is.s.u-tokyo.ac.jp

Sebastian Maneth
NICTA and University of New South Wales
sebastian.maneth@nicta.com.au

ABSTRACT

Macro tree transducers (mtts) are a useful formal model for XML query and transformation languages. In this paper one of the fundamental decision problems on translations, namely the "translation membership problem" is studied for mtts. For a fixed translation, the translation membership problem asks whether a given input/output pair is element of the translation. For call-by-name mtts this problem is shown to be NP-complete. The main result is that translation membership for call-by-value mtts is in polynomial time. For several extensions, such as addition of regular look-ahead or the generalization to multi-return mtts, it is shown that translation membership still remains in PTIME.

1. INTRODUCTION

Macro tree transducers (mtts) [6] are a popular formal model for XML query and transformation languages (cf., e.g., [4, 13, 15]. They are powerful enough to represent a wide range of practical transformations, and they subsume various well-known models of tree translations such as attribute grammars, MSO-definable tree translations [2], or pebble tree transducers [16]. Yet, mtts have many decidable properties such as exact typechecking or emptiness and finiteness and membership of their domains and ranges. These make mtts a useful device for static verification of XML translation programs.

In the algorithms that decide such properties, we sometimes encounter as a sub-problem the "translation membership problem" [11]. For a fixed translation, the translation membership problem asks whether a given input/output pair is element of the translation. Although the problem itself seems simple, it is far beyond trivial to solve the problem efficiently, in particular if we consider nondeterministic mtts. Nondeterminism is useful when using the mtt to approximate the behavior of a "real" (Turing-complete) programming language (viz. a complicated if-then-else expression; it is translated into an mtt that nondeterministically chooses one of the conditional branches). Depending on the order of evaluation, there are two different models of nondeterministic mtts, namely, call-by-value (also called inside-out or IO for short) and call-by-name (outside-in or OI). Note that in the limit, to one given input tree of size n an mtt can associate at most $2^{2^{2^n}}$ -many different output trees, if the mtt operates in OI mode. In contrast, the limit for mtts in IO mode is at most 2^{2^n} different output trees for a given input tree of size n. Consider the following four rules of an mtt.

$$\begin{array}{lcl} \mathrm{start}(\mathtt{a}(x_1)) & \to & \mathrm{double}(x_1, \mathrm{double}(x_1, \mathtt{e})) \\ \mathrm{double}(\mathtt{a}(x_1), y_1) & \to & \mathrm{double}(x_1, \mathrm{double}(x_1, y_1)) \\ \mathrm{double}(\mathtt{e}, y_1) & \to & \mathtt{f}(y_1, y_1) \mid \mathtt{g}(y_1, y_1). \end{array}$$

For an input tree of the form $s_n = \mathtt{a}(\mathtt{a}(\cdots \mathtt{a}(\mathtt{e}) \cdots))$ with n $\mathtt{a}$-nodes, this mtt generates a full binary tree of height 2^n (and thus of size 2^{2^n}). If the mtt operates in OI derivation mode, then each node of the binary output tree is nondeterministically labeled either $\mathtt{f}$ or $\mathtt{g}$; thus, there are $2^{2^{2^n}}$ -many output trees associated to the input tree s_n. If, however, the mtt with the same rules operates in IO derivation mode, then for input s_n it generates only 2^{2^n} many different output trees (the nodes on one level of an output tree all have the same label). Thus, mtts in OI derivation mode (call-by-name) have "much more" nondeterminism than mtts in IO derivation mode (call-by-value). This difference suggests that translation membership is computationally harder for OI-mtts than for IO-mtts.

In this paper, we first show that for OI-mtts, translation membership is NP-complete, and so is for compositions of multiple IO-mtts (Section 3). We then present our main result: translation membership for IO-mtts is solvable in polynomial time (Section 4). Our algorithm for IO translation membership is based on a technique called inverse type inference. For an mtt M and a given output type, i.e., a regular tree language L of output trees, inverse type inference constructs a description of the corresponding input type, i.e., of the regular tree language $M^{-1}(L)$. Note that, inverse type inference basically takes exponential time, because the size of the inverse-type automaton itself can be that large [16, 15, 17]. To avoid this, we construct the automaton on-the-fly and obtain the PTIME efficiency. Our technique is then generalized to several extension of IO-mtts, such as addition of regular look-ahead or the generalization to multi-return mtts. In fact, we even consider a more powerful look-ahead mechanism that is based on tree automata with equality and disequality constraints between siblings [1].

Note that, for total deterministic mtts: OI equals IO, and

International Workshop on Programming Language Techniques for XML (PLAN-X 2009),
January 24, 2009, Savannah, Georgia.

by Theorem 15 of [12], given an input tree s, the output tree $\tau(s)$ can be computed in time $O(|s| + |t|)$, even for an n-fold composition of total deterministic mtts. Hence, by simply computing the output, translation membership can be solved in linear time for this class of translations. The result can easily be extended to deterministic but partial mtts (in either IO or OI derivation mode), as mentioned at the end of Section 4.

2. DEFINITIONS

For a finite set A, we denote by $|A|$ the number of its elements. A finite set Σ with a mapping $\mathit{rank} : \Sigma \to \mathbb{N}$ is called a *ranked alphabet*. We often write $\sigma^{(k)}$ to indicate that $\mathit{rank}(\sigma) = k$ and write $\Sigma^{(k)}$ to denote the subset of Σ of rank-k symbols. The *product* of Σ and a set B is the ranked alphabet $\Sigma \times B = \{\langle \sigma, b\rangle^{(k)} \mid \sigma^{(k)} \in \Sigma, b \in B\}$. Throughout the paper, we fix the sets of input variables $X = \{x_1, x_2, \dots\}$, parameters $Y = \{y_1, y_2, \dots\}$, and let-variables $Z = \{z_1, z_2, \dots\}$, which are all of rank 0. We assume any other alphabet to be disjoint with X, Y, and Z. The set X_i is defined as $\{x_1, \dots, x_i\}$, and Y_i and Z_i are defined similarly.

The set T_Σ of *trees* t over a ranked alphabet Σ is defined by the BNF $t ::= \sigma(\overbrace{t, \dots, t}^{k})$ for $\sigma \in \Sigma^{(k)}$. We often omit parentheses for rank-0 and rank-1 symbols. We recursively define the function *label* from $T_\Sigma \times \mathbb{N}^*$ to Σ as follows. For $t = \sigma(t_1, \dots, t_k)$, $\sigma^{(k)} \in \Sigma$, $k \geq 0$, and $t_1, \dots, t_k \in T_\Sigma$, $\mathit{label}(t, \epsilon) = \sigma$ and $\mathit{label}(t, i.\nu) = \mathit{label}(t_i, \nu)$. Thus, the empty list ϵ denotes the root node and $\nu.i$ denotes the i-th child of ν. We define the set $\mathit{pos}(t) = \{\nu \in \mathbb{N}^* \mid \mathit{label}(t, \nu) \text{ is defined}\}$. We denote by $|t|$ the number of nodes in the tree t. For a node v of t, $t|_v$ denotes the subtree of t rooted at the node v. For trees $t, t_1, \dots, t_n \in T_\Sigma$ and $\sigma_1, \dots, \sigma_n \in \Sigma^{(0)}$, we denote by $t\,[\sigma_1/t_1, \dots, \sigma_n/t_n]$ the simultaneous substitution of the σ_i by the t_i.

Let Σ and Δ be ranked alphabets. A relation $\tau \subseteq T_\Sigma \times T_\Delta$ is called a *tree translation* (over Σ and Δ) or simply a translation. We define $\mathit{range}(\tau) = \{b \mid \exists a : (a, b) \in \tau\}$. For two translations τ_1 and τ_2, their sequential composition $\tau_1\,;\tau_2$ ("τ_1 followed by τ_2") is the translation $\{(a, c) \mid \exists b : ((a, b) \in \tau_1, (b, c) \in \tau_2)\}$. For two classes T_1 and T_2 of translations, we define $T_1\,;T_2 = \{\tau_1\,;\tau_2 \mid \tau_1 \in T_1, \tau_2 \in T_2\}$. The k-fold composition of the class T of translations is denoted by T^k.

Definition 1. A *macro tree transducer (mtt)* M is a tuple $(Q, \Sigma, \Delta, q_0, R)$, where Q is the ranked alphabet of *states*, Σ and Δ are the *input* and *output* alphabets, $q_0 \in Q^{(0)}$ is the *initial state*, and R is the finite set of *rules* of the form

$$\langle q, \sigma(x_1, \dots, x_k)\rangle(y_1, \dots, y_m) \to r$$

where $q \in Q^{(m)}$, $\sigma \in \Sigma^{(k)}$, and r is a tree in $T_{\Delta \cup (Q \times X_k) \cup Y_m}$. Rules of such form are called $\langle q, \sigma\rangle$-rules, and the set of right-hand sides of all $\langle q, \sigma\rangle$-rules is denoted by $R_{q,\sigma}$. We define the size of the mtt by $|M| = \sum\{|r| \mid r \in R_{q,\sigma}, q \in Q, \sigma \in \Sigma\}$.

For the remainder of this section, let M be an mtt as in Definition 1. A state q of a macro tree transducer can be regarded as a (nondeterministic) *function* in functional programming languages. Depending on the order of evaluation, two different semantics can be considered: call-by-value (or inside-out, IO) and call-by-name (or, outside-in, OI). Let $\mu \in \{\mathrm{IO}, \mathrm{OI}\}$. For the tree $u \in T_{\Delta \cup (Q \times T_\Sigma) \cup Y}$, its meaning with respect to M $[\![u]\!]^M_\mu \subseteq T_{\Delta \cup Y}$ is inductively defined as follows

$$\begin{aligned}
[\![y_i]\!]^M_\mu &= \{y_i\} \\
[\![\delta(u_1, \dots, u_n)]\!]^M_\mu &= \{\delta(t_1, \dots, t_n) \mid \\
&\qquad t_i \in [\![u_i]\!]^M_\mu \text{ for all } i\} \\
[\![\langle q, \sigma(s_1, \dots, s_k)\rangle(u_1, \dots, u_m)]\!]^M_\mu &= \\
\bigcup_{r \in R_{q,\sigma}} \Big([\![r[x_1/s_1, \dots, x_k/s_k]]\!]^M_\mu &\underset{\mu}{\leftarrow} ([\![u_1]\!]^M_\mu, \dots, [\![u_m]\!]^M_\mu)\Big)
\end{aligned}$$

where $\underset{\mathrm{IO}}{\leftarrow}$ and $\underset{\mathrm{OI}}{\leftarrow}$ denote IO- and OI-substitution, respectively, and are defined as follows for $L, L_1, \dots, L_n \subseteq T_{\Delta \cup Y}$.

$$\begin{aligned}
L \underset{\mu}{\leftarrow} (L_1, \dots, L_n) &= \bigcup_{t \in L} \Big(t \underset{\mu}{\leftarrow} (L_1, \dots, L_n)\Big) \\
t \underset{\mathrm{IO}}{\leftarrow} (L_1, \dots, L_n) &= \{t[y_1/t_1, \dots, y_n/t_n] \mid \\
&\qquad t_i \in L_i \text{ for all } i\} \\
y_i \underset{\mathrm{OI}}{\leftarrow} (L_1, \dots, L_n) &= L_i \\
\delta(t_1, \dots, t_m) \underset{\mathrm{OI}}{\leftarrow} (L_1, \dots, L_n) &= \\
\{\delta(t'_1, \dots, t'_m) \mid t'_i \in \Big(t_i \underset{\mathrm{OI}}{\leftarrow} (L_1, \dots, L_n)\Big) &\text{ for all } i\}.
\end{aligned}$$

The difference of IO- and OI- semantics lies in the interpretation of state calls. In IO-semantics we use IO-substitution for parameters; each parameter y_i is bound to some fixed (but nondeterministically chosen) tree in $[\![u_i]\!]^M_{\mathrm{IO}}$, and every occurrence of y_i is replaced with the same single tree. On the other hand, in OI-semantics, each parameter is bound to the set of trees $[\![u_i]\!]^M_{\mathrm{OI}}$, and at every occurrence of y_i we nondeterministically choose some tree in $[\![u_i]\!]^M_{\mathrm{OI}}$, independent from the choices made at other occurrences of y_i.

As an example of the definition of $[\![u]\!]_\mu$, consider the example from the Introduction. Note that there we used slightly different notation: the right-hand side $\mathrm{double}(x, \mathrm{double}(x, \mathtt{e}))$ is now written as $\langle \mathrm{double}, x_1\rangle(\langle \mathrm{double}, x_1\rangle(\mathtt{e}))$, i.e., we distinguish the first parameter—which is the special parameter that is bound to an input tree in T_Σ—from others bound to output trees in T_Δ, by enclosing it with angle brackets. Now, let us compute $[\![\langle \mathrm{start}, \mathtt{a}(\mathtt{a}(\mathtt{e}))\rangle]\!]_\mu$.

$$\begin{aligned}
&[\![\langle \mathrm{start}, \mathtt{a}(\mathtt{a}(\mathtt{e}))\rangle]\!]_\mu \\
&= [\![\langle \mathrm{double}, \mathtt{a}(\mathtt{e})\rangle(\langle \mathrm{double}, \mathtt{a}(\mathtt{e})\rangle(\mathtt{e}))]\!]_\mu \\
&= [\![\langle \mathrm{double}, e\rangle(\langle \mathrm{double}, \mathtt{e}\rangle(y_1))]\!]_\mu \underset{\mu}{\leftarrow} [\![\langle \mathrm{double}, \mathtt{a}(\mathtt{e})\rangle(\mathtt{e})]\!]_\mu \\
&= \Big(\{\mathtt{f}(y_1, y_1), \mathtt{g}(y_1, y_1)\} \underset{\mu}{\leftarrow} [\![\langle \mathrm{double}, \mathtt{e}\rangle(\mathtt{e})]\!]_\mu\Big) \\
&\qquad\qquad \underset{\mu}{\leftarrow} [\![\langle \mathrm{double}, \mathtt{a}(\mathtt{e})\rangle(\mathtt{e})]\!]_\mu \\
&= \Big(\{\mathtt{f}(y_1, y_1), \mathtt{g}(y_1, y_1)\} \underset{\mu}{\leftarrow} \{\mathtt{f}(y_1, y_1), \mathtt{g}(y_1, y_1)\}\Big) \\
&\qquad\qquad \underset{\mu}{\leftarrow} [\![\langle \mathrm{double}, \mathtt{a}(\mathtt{e})\rangle(\mathtt{e})]\!]_\mu
\end{aligned}$$

Here, we encountered the μ-substitution $L \underset{\mu}{\leftarrow} L$ for $L = \{\mathtt{f}(y_1, y_1), \mathtt{g}(y_1, y_1)\}$. Now, if $\mu = \mathrm{IO}$ then $L \underset{\mu}{\leftarrow} L = \{\mathtt{f}(\mathtt{f}(y_1, y_1), \mathtt{f}(y_1, y_1)),\ \mathtt{g}(\mathtt{f}(y_1, y_1), \mathtt{f}(y_1, y_1)),\ \mathtt{f}(\mathtt{g}(y_1, y_1), \mathtt{g}(y_1, y_1)),$

$\mathtt{g}(\mathtt{g}(y_1,y_1),\mathtt{g}(y_1,y_1))\}$; the size of the set is $2\times 2 = 4$. On the other hand, if $\mu = \mathrm{OI}$ then we obtain $L \underset{\mu}{\leftarrow} L = \{\mathtt{f}(\mathtt{f}(y_1,y_1),\mathtt{f}(y_1,y_1)),\ \mathtt{f}(\mathtt{f}(y_1,y_1),\mathtt{g}(y_1,y_1)),\ \mathtt{f}(\mathtt{g}(y_1,y_1),\mathtt{f}(y_1,y_1)),\ \mathtt{f}(\mathtt{g}(y_1,y_1),\mathtt{g}(y_1,y_1)),\ \mathtt{g}(\mathtt{f}(y_1,y_1),\mathtt{f}(y_1,y_1)),\ \mathtt{g}(\mathtt{f}(y_1,y_1),\mathtt{g}(y_1,y_1)),\ \mathtt{g}(\mathtt{g}(y_1,y_1),\mathtt{f}(y_1,y_1)),\ \mathtt{g}(\mathtt{g}(y_1,y_1),\mathtt{g}(y_1,y_1))\}$; the size is $2\times 2^2 = 8$ where the exponent 2 comes from the number of occurrences of the parameter y_1 in each target term of the substitution.

We define the *translation realized by* M in μ-mode by the relation $\tau_{\mu,M} = \{(s,t) \in T_\Sigma \times T_\Delta \mid t \in [\![\langle q_0, s\rangle]\!]_\mu\}$. The class of all translations realized by all mtts in μ-mode is denoted by MTT_μ. An mtt is called *deterministic* (respectively, *total*) if for every q,σ, the number of rules $|R_{q,\sigma}|$ is at most (at least) 1; the corresponding classes of translations are denoted by prefix D (t). An mtt is called *linear (in the input variables)* if in every right-hand side of the rules, each input variable x_i appears at most once; the corresponding class of translation is denoted by prefix L. For example, the class of translations realized by linear, deterministic, and total mtts in OI mode is denoted by $\mathrm{LDtMTT}_{\mathrm{OI}}$.

For a translation $\tau \subseteq T_\Sigma \times T_\Delta$, the *translation membership problem* for τ is a decision problem that determines, given a tree $s \in T_\Sigma$ and a tree $t \in T_\Delta$, whether $(s,t) \in \tau$. In the rest of the paper, we focus on the *data complexity* of this problem. That is, we measure the complexity in terms of $|s| + |t|$, regarding the translation τ to be fixed. We will always assume that the input and output tree that are inputs to the problem are denoted by "s" and "t".

3. NP-COMPLETE CLASSES

The first result is that translation membership for OI-mtts is NP-hard, even for linear mtts. The proof is based on the reduction to 3-SAT, which resembles [18] which shows NP-completeness of the membership problem for indexed languages. In fact, the indexed languages can be obtained as yields (strings of leaves from left to right) of output languages of linear mtts (by the fact that each indexed language is the yield of some OI context-free tree language [7] and each OI context-free tree language is equivalent to the range *range*(τ) of some $\tau \in \mathrm{LMTT}_{\mathrm{OI}}$ by Corollary 6.13 in [6]). However, given a word w as input for the membership problem of an indexed language L, it is not clear how to construct a pair (s,t) such that $(s,t) \in \tau_{\mathrm{OI},M}$ for some linear mtt if and only if w is in L. We can choose $s = a^n$ with $n =$ *length*(w) and an LMTT which produces trees t which have as yield the word w. But how to select such a tree t as input for the translation membership problem? Note that it is easy to construct from w an input for translation membership for a two-fold composition of mtts: the second transducer realizes "yield", i.e., it turns a tree t into a monadic tree that represents t's yield (such a transducer is even total deterministic). Thus, it follows that translation membership for two-fold compositions of mtts is NP-hard. This was mentioned already in [11]. The next lemma shows that even translation membership for a single linear mtt is NP-hard.

Lemma 1 Translation membership for $\mathrm{LMTT}_{\mathrm{OI}}$ (and hence $\mathrm{MTT}_{\mathrm{OI}}$) is NP-hard.

PROOF. We construct an mtt $M = (Q, q_0, \Sigma, \Delta, R)$ so that it generates the parse-trees of all satisfiable boolean formulas in 3-conjunctive normal form, given the number of variables n and clauses m as the inputs. We slightly abuse our notation and write y_v, y_t, y_f in place of y_1, y_2, y_3, respectively. Let $Q = \{q_0^{(0)}, q_c^{(2)}, q^{(3)}\}$, $\Sigma = \{\mathtt{a}^{(1)}, \mathtt{b}^{(3)}, \mathtt{c}^{(1)}, \mathtt{d}^{(0)}\}$, $\Delta = \{\wedge^{(2)}, \vee^{(3)}, \neg^{(1)}, \mathtt{v}^{(1)}, \mathtt{e}^{(0)}\}$, and R the following set of rules:

$$\begin{aligned}
\langle q_0, \mathtt{a}(x_1)\rangle &\to \langle q, x_1\rangle(\mathtt{v}(\mathtt{e}), \mathtt{e}, \neg(\mathtt{e}))\\
\langle q_0, \mathtt{a}(x_1)\rangle &\to \langle q, x_1\rangle(\mathtt{v}(\mathtt{e}), \neg(\mathtt{e}), \mathtt{e})\\
\langle q, \mathtt{b}(x_1,x_2,x_3)\rangle(y_v,y_t,y_f) &\to\\
&\langle q, x_1\rangle(\mathtt{v}(y_v), \langle q_c, x_2\rangle(y_t, y_v), \langle q_c, x_3\rangle(y_f, \neg(y_v)))\\
\langle q, \mathtt{b}(x_1,x_2,x_3)\rangle(y_v,y_t,y_f) &\to\\
&\langle q, x_1\rangle(\mathtt{v}(y_v), \langle q_c, x_2\rangle(y_t, \neg(y_v)), \langle q_c, x_3\rangle(y_f, y_v))\\
\langle q_c, \mathtt{d}\rangle(y_1,y_2) &\to y_1\\
\langle q_c, \mathtt{d}\rangle(y_1,y_2) &\to y_2
\end{aligned}$$

$$\begin{aligned}
\langle q, \mathtt{c}(x_1)\rangle(y_v,y_t,y_f) &\to \wedge(\vee(y_t,y_t,y_t), \langle q, x_1\rangle(y_v,y_t,y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v,y_t,y_f) &\to \wedge(\vee(y_t,y_t,y_f), \langle q, x_1\rangle(y_v,y_t,y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v,y_t,y_f) &\to \wedge(\vee(y_t,y_f,y_t), \langle q, x_1\rangle(y_v,y_t,y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v,y_t,y_f) &\to \wedge(\vee(y_f,y_t,y_t), \langle q, x_1\rangle(y_v,y_t,y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v,y_t,y_f) &\to \wedge(\vee(y_t,y_f,y_f), \langle q, x_1\rangle(y_v,y_t,y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v,y_t,y_f) &\to \wedge(\vee(y_f,y_f,y_t), \langle q, x_1\rangle(y_v,y_t,y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v,y_t,y_f) &\to \wedge(\vee(y_f,y_t,y_f), \langle q, x_1\rangle(y_v,y_t,y_f))\\
\langle q, \mathtt{d}\rangle(y_v,y_t,y_f) &\to \vee(y_t,y_t,y_f)\\
&\vdots \quad \text{(same as the } \vee(\cdots) \text{ part of } \langle q, c\rangle\text{-rules)}\\
\langle q, \mathtt{d}\rangle(y_v,y_t,y_f) &\to \vee(y_f,y_t,y_f).
\end{aligned}$$

From an input tree $\mathtt{a}(\overbrace{\mathtt{b}(\mathtt{b}(\cdots\mathtt{b}(}^{n}\mathtt{c}^m\mathtt{d},\mathtt{d},\mathtt{d})\cdots),\mathtt{d},\mathtt{d}))$ of size $3n+m+2$, it generates all satisfiable boolean formulas in 3-conjunctive normal form with n variables and m conjuncts. The output language encodes boolean formulas as follows: a boolean variable p_i for $0 \le i < n$ is represented as $\mathtt{v}^i\mathtt{e}$, and three boolean operations $\neg$, $\wedge$, and $\vee$ are represented as they are. For example, the formula $(p_0 \vee \neg p_1 \vee p_2) \wedge (\neg p_0 \vee p_1 \vee p_2)$ is encoded as $\wedge(\vee(\mathtt{e}, \neg\mathtt{ve}, \mathtt{vve}), \vee(\neg\mathtt{e}, \mathtt{ve}, \mathtt{vve}))$.

Intuitively, when the mtt reads the root node of the input, it nondeterministically assigns a truth-value to the first variable p_0. The first $\langle q_0, \mathtt{a}\rangle$-rule is the case when it assigned 'true' and the other rule is for 'false'. Three parameters are passed to the state q. Intuitively, the first parameter y_v denotes the name of the next variable to be assigned a truth-value. The second (and the third, respectively) parameter y_t (y_f) denotes the set of 'true' ('false') literals (namely, variables or negated variables) that have been constructed up to now. While reading $\mathtt{b}$ nodes in the state q, the mtt nondeterministically assigns a truth-value to each variable p_1 to p_{n-1}, similarly to p_0. Here, OI-nondeterminism is crucially used to represent arbitrary choice of positive and negative literals; each time y_t and y_f are copied to the output, they contain unevaluated "combs" of q_c-calls (on $\mathtt{d}$-nodes). Each such comb represents the nondeterministic choice of any of the positive (y_t) or negative (y_f) literals that have been generated so far. The state q_c means a union of two sets, by taking two parameters and nondeterministically returns

either one of them. The parameter y_t is assigned an unevaluated expression, e.g., like $\langle q_c, d\rangle(\langle q_c, d\rangle(\neg p_0, p_1), p_2)$, and each time the value of y_t is needed, it is nondeterministically evaluated to either $\neg p_0$, p_1, or p_2. Then, while reading $\mathtt{c}$ nodes in the input, the transducer generates m conjunctions of 'true' clauses. Since we generate 3-CNF formulas, each clause consists of a disjunction of exactly three literals. There are seven possibilities (all combinations of y_t and y_f, except $\vee(y_f, y_f, y_f)$), which are generated by the $\langle q, \mathtt{c}\rangle$-rules of the transducer.

It should be clear for the reader that this mtt generates all (and only) satisfiable 3-CNF formulas; it nondeterministically constructs any of the 2^n possible assignments to the variables $p_0, \ldots, p_{n-1}$, and under each assignment, generates any of the possible 7^m types of 'true' formulas. The point is, the choices at $\langle q_c, \mathtt{d}\rangle$ for enumerating all possible literals are nondeterministically evaluated each time generating a disjunct, while the choices at $\langle q_0, \mathtt{a}\rangle$ and $\langle q, \mathtt{b}\rangle$ for enumerating all possible truth-value assignments are evaluated and uniformly determined prior to the generation of all conjuncts.

It is also obvious that, given any 3-CNF formula, we can in polynomial time encode the formula to the above explained encoding to obtain t, and count the number of variables and clauses to obtain s. Then, $(s,t) \in \tau_M$ if and only if the original formula is satisfiable. It is well known that the satisfiability of 3-CNF is NP-complete (see, e.g., [8]). □

In [11], we have proved two closely related results; one is that the above NP-hard lowerbound is tight, i.e., the translation membership for $\mathrm{LMTT}_{\mathrm{OI}}$ can be determined in NP time complexity. The other is that the complexity of membership problem of the *output language* is in NP, even for finitely many compositions of $\mathrm{MTT}_{\mathrm{OI}}$'s. Altogether, we have the following theorem.

Theorem 2 Translation membership for $\mathrm{MTT}^n_{\mathrm{OI}}$ for $n \geq 1$ is NP-complete.

Proof. NP-hardness follows from the preceding lemma. Let $\tau \in \mathrm{MTT}^n_{\mathrm{OI}}$. We can easily construct a translation $\tau' = \{(s, \pi(s,t)) \mid (s,t) \in \tau\}$ in $\mathrm{MTT}^n_{\mathrm{OI}}$ where π is a new binary symbol. This is done by changing the first mtt M_1 (with input alphabet Σ and initial state q_0) of the composition as follows. Replace for $\sigma \in \Sigma^{(k)}$ every $\langle q_0, \sigma\rangle$-rule with right-hand side t by the new rule $\langle q_0, \sigma(x_1, \ldots, x_k)\rangle \to \pi(\sigma(\langle q_{id}, x_1\rangle, \ldots, \langle q_{id}, x_k\rangle), t)$ and introduce $\langle q_{id}, \sigma(x_1, \ldots, x_k)\rangle \to \sigma(\langle q_{id}, x_1\rangle, \ldots, \langle q_{id}, x_k\rangle)$ for the new state q_{id} of rank 0. Then, the subsequent mtts M_i $(2 \leq i \leq n)$ are augmented by the new rule $\langle q_0, \pi(x_1, x_2)\rangle \to \pi(\langle q_{id}, x_1\rangle, \langle q_0, x_2\rangle)$ and q_{id} rules as for M_1. Note that $(s,t) \in \tau$ if and only if $\pi(s,t) \in \mathit{range}(\tau')$. Since by Theorem 8 of [11] the complexity of the membership test of $\mathit{range}(\tau')$ is in NP, we can also check $(s,t) \in \tau$ in NP. □

Note that compositions of two $\mathrm{MTT}_{\mathrm{IO}}$'s can simulate all $\mathrm{MTT}_{\mathrm{OI}}$ translations (Theorem 6.10 of [6]), and conversely, compositions of $\mathrm{MTT}_{\mathrm{IO}}$'s can be simulated by compositions $\mathrm{MTT}_{\mathrm{OI}}$'s (Theorem 7.8 of [6]). Therefore, we now have the NP-completeness for compositions of $\mathrm{MTT}_{\mathrm{IO}}$'s.

Corollary 3 Translation membership for $\mathrm{MTT}^n_{\mathrm{IO}}$ for $n \geq 2$ is NP-complete.

4. TRACTABLE CLASSES

In this section, we first prove that IO-mtts have polynomial-time translation membership, contrary to OI-mtts. Then we extend the result to several other extensions of IO-mtts, and to some restricted subclasses of OI-mtts.

The idea of the proof is based on inverse type inference for mtts M (Theorem 7.4 of [6]); given a finite tree automaton $\mathcal{B}$ (accepting output trees), we can effectively construct a finite tree automaton that recognizes the corresponding input trees $\tau_M^{-1}(L(\mathcal{B}))$. Given an output tree t, by constructing its minimal dag representation (i.e., the pointer representation of t such that all isomorphic subtrees are shared), we can simply consider it as the trivial deterministic automaton $\mathcal{B}_t$ with at most $|t|$-many states which recognizes $\{t\}$. Once we have constructed the automaton $\mathcal{A}$ for $\tau_M^{-1}(L(\mathcal{B}_t))$, we merely need to check whether $s \in L(\mathcal{A})$, in order to solve translation membership for (s,t). However, the automaton $\mathcal{A}$ can be very large: its worst case number of states is exponential in $|\mathcal{B}_t|$. Thus, we must avoid to fully construct $\mathcal{A}$ in order to obtain PTIME complexity. Our idea is to construct $\mathcal{A}$ on demand, while running it on the tree s. Note that inverse type inference of an IO-mtt constructs an input type automaton which has states that are functions p from Q to $(V^m \to 2^V)$ where V is the set of states of $\mathcal{B}_t$, Q is the set of states of M, and m is the maximum rank of states in Q. Such a state p tells us for each $q \in Q$, which state of $\mathcal{B}_t$ is obtained if we apply the state q to an input tree. That is, if $\mathcal{A}$ reaches the state p after reading a tree s, it means that running $\mathcal{B}_t$ on output trees in $\langle q, s\rangle(t|_{v_1}, \ldots, t|_{v_m})$ obtains the states $(p(q))(v_1, \ldots, v_m)$.

Theorem 4 Let M be an mtt. Translation membership for $\tau_{\mathrm{IO},M}$ can be determined in time $O(|s| \cdot |t|^{2m+2} \cdot |M|)$ where m is the maximum rank of M's states.

Proof. Let t_{dag} be the minimal dag representing t. It is folklore that t_{dag} can be computed in amortized linear time in $|t|$, using hashing, and even in linear time using pseudo radix sorting, see [3]. Let V_t be the set of nodes of t_{dag}. We define $\mathit{label}(v)$ to denote the label in Σ of the node $v \in V_t$, and $\mathit{child}(v,i)$ to denote the i-th child node of v. Assuming a standard pointer structure representing dags, we regard each execution of *label* and *child* takes $O(1)$ time.

Let $\perp$ be an element distinct from V_t. Let $V = V_t \cup \{\perp\}$ and $\mathit{label}(\perp)$ to be undefined. Let $\mathit{run} : T_\Sigma \to A$ with $A = 2^{\bigcup_i Q^{(i)} \times V^i \times V}$ be the function defined inductively as follows

$$\mathit{run}(\sigma(s_1, \ldots, s_k)) = \mathit{tr}(\sigma, \mathit{run}(s_1), \ldots, \mathit{run}(s_k))$$

where tr is defined below. The set A contains the states of the deterministic bottom-up automaton of $\tau^{-1}(t)$, tr is the transition function, and run computes the run of the automaton. The intuition of the set of states A is, that "$(q, \vec{v}, v') \in \mathit{run}(s')$" means that "if q is applied to the input subtree s' with output subtrees rooted at $\vec{v}$ as parameters, then it may generate an output subtree rooted at v'". The special value $\perp \in V$ is used to denote a tree that is not

a subtree of t. That is, for example, "$(q, \vec{v}, \bot) \in run(s')$" means that an application of q to s' with parameters $\vec{v}$ may yield a tree that is not a subtree of t.

The transition function $tr : (\bigcup_i \Sigma^{(i)} \times A^i) \to A$ is defined as follows

$$tr(\sigma, \vec{a}) = \Big\{(q, \vec{v}, v') \in \bigcup_i Q^{(i)} \times V^i \times V \;\Big|\; \exists r \in R_{q,\sigma} : f_{\vec{v},\vec{a}}(r, v')\Big\}$$

where $f_{\vec{v},\vec{a}} : T_{\Delta\cup(Q\times X)\cup Y} \times V \to \{true, false\}$ is defined inductively on right-hand sides of the rules:

$$f_{\vec{v},\vec{a}}(y_i, v') = true \quad \text{if } v' = v_i$$
$$f_{\vec{v},\vec{a}}(y_i, v') = false \quad \text{if } v' \neq v_i$$
$$f_{\vec{v},\vec{a}}(\delta(r_1, \ldots, r_n), v') =$$
$$label(v') = \delta \wedge \bigwedge_{1\le i\le n} f_{\vec{v},\vec{a}}(r_i, child(v', i)) \quad \text{if } v' \in V_t$$
$$f_{\vec{v},\vec{a}}(\delta(r_1, \ldots, r_n), \bot) = \big(\exists \vec{u} \in V^n : \bigwedge_{1\le i\le n} f_{\vec{v},\vec{a}}(r_i, u_i)\big) \wedge$$
$$\big(\forall u' \in V_t : \neg\big(label(u') = \delta \wedge \bigwedge_{1\le i\le n} child(u', i) \neq u_i\big)\big)$$
$$f_{\vec{v},\vec{a}}(\langle q', x_j\rangle(r_1, \ldots, r_n), v') =$$
$$\exists \vec{u} \in V^n : \Big((q', \vec{u}, v') \in a_j \wedge \bigwedge_{1\le i\le n} f_{\vec{v},\vec{a}}(r_i, u_i)\Big).$$

The relation $f_{\vec{v},\vec{a}}(r, v')$ should be understood as: "evaluation of r will yield the output subtree at v', under the assumption that the parameters $\vec{y}$ are bound to $\vec{v}$ and the effects of application of a state to each child is as described by $\vec{a}$".

For a tree $t' \in T_\Delta$, let $\rho(t')$ be $v \in V_t$ if $t' = t|_v$, and $\rho(t') = \bot$ otherwise. We also define $\rho(T)$ for $T \subseteq T_\Delta$ as $\{\rho(t) \mid t \in T\}$. The correctness of the above construction is verified by the following claim. Note that the claim is just rephrasing the intuition of the set of states A explained above, in a formal way.

Claim For every input tree s', we have the following equation for all $q \in Q$, $r_i \in T_{\Delta\cup(Q\times T_\Sigma)}$, and an environment Γ: $\rho\Big([\![\langle q, s'\rangle(r_1, \ldots, r_n)]\!]^M_{\mathrm{IO}}\Big) = \Big\{v' \;\Big|\; (q, (v_1, \ldots, v_n), v') \in run(s'), v_i \in \rho([\![r_i]\!]^M_{\mathrm{IO}}) \text{ for all } i\Big\}$

By applying the claim for $q = q_0$ and $s' = s$, we know that $t \in [\![\langle q, s\rangle]\!]^M_{\mathrm{IO}}$ is equal to $(q_0, (), v_\epsilon) \in run(s)$ where v_ϵ is the root node of t_{dag}. Hence, the translation membership can be determined by computing the set $run(s)$.

The proof of the claim is by nested induction first on structure of s', and then on the structure of right-hand sides of the rules. Let $s' = \sigma(s_1, \ldots, s_k)$ (the base case is the case $k = 0$). By definition of the IO-semantics we have

$$\rho\Big([\![\langle q, s'\rangle(r_1, \ldots, r_n)]\!]^M_{\mathrm{IO}}\Big) = \bigcup_{r\in R_{q,\sigma}} \Big\{\rho(t'[y_1/t_1, \ldots, y_n/t_n]) \;\Big|\; t' \in [\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO}}, t_i \in [\![r_i]\!]^M_{\mathrm{IO}} \text{ for all } i\Big\}$$

and by definition of run, we have

$$\Big\{v' \;\Big|\; (q, \vec{v}, v') \in run(s'), v_i \in \rho([\![r_i]\!]^M_{\mathrm{IO}})\Big\} = \bigcup_{r\in R_{q,\sigma}} \Big\{v' \;\Big|\; f_{\vec{v},\vec{a}}(r, v'), v_i \in \rho([\![r_i]\!]^M_{\mathrm{IO}})\Big\}$$

where $\vec{a} = (run(s_1), \ldots, run(s_k))$. To show these two sets are equal, it is sufficient to prove the the following statement: if $\rho(t_i) = v_i$ then $\{\rho(t'[\vec{y}/\vec{t}]) \mid t' \in [\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO}}\} = \{v' \mid f_{\vec{v},\vec{a}}(r, v')\}$. The proof is by nested induction on the structure of r. For example, if $r = \langle q', x_i\rangle(r_1, \ldots, r_n)$, we have $\{v' \mid f_{\vec{v},\vec{a}}(\langle q', x_i\rangle(r_1, \ldots, r_n), v')\} = \{v' \mid (q', \vec{u}, v') \in a_i, f_{\vec{v},\vec{a}}(r_i, u_i) \text{ for all } i\}$, which is by inner induction hypothesis equal to $\{v' \,|\, (q', \vec{u}, v') \in a_i, u_i \in \rho([\![r_i[\vec{x}/\vec{s}, \vec{y}/\vec{t}]]\!]^M_{\mathrm{IO}}) \text{ for all } i\}$, and then by outer induction hypothesis it is equal to $\rho([\![\langle q', s_i\rangle(r_1[\vec{x}/\vec{s}, \vec{y}/\vec{t}], \ldots, r_n[\vec{x}/\vec{s}, \vec{y}/\vec{t}])]\!]^M_{\mathrm{IO}}) = \{\rho(t'[\vec{y}/\vec{t}]) \mid t' \in [\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO}}\}$. The other cases are proved similarly.

The time complexity for testing $(q_0, (), v_\epsilon) \in run(s)$ is computed as follows. The value $run(s)$ for the whole input tree s can be computed by executing the tr function on each node of s. The computation is done in bottom-up fashion as bottom-up tree automata does, so that the states in $\vec{a}$ are already constructed. The number of execution of the tr function is $|s|$. The set $tr(\sigma, \vec{a})$ can be constructed by simply testing all combinations of $(q, \vec{v}, v') \in \bigcup_i Q^{(i)} \times V^i \times V$ (which is of size $\le |Q| \cdot |V|^{m+1}$) and $r \in R_{q,\sigma}$ by $f_{\vec{v},\vec{a}}$. Note that $f_{\vec{v},\vec{a}}$ may receive $|r| \cdot |V|$ different pairs of arguments, and the computation of each value $f_{\vec{v},\vec{a}}(r', v')$ takes $O(|V|^m)$ time in the worst case (the $f_{\vec{v},\vec{a}}(\langle q', x_j\rangle(\cdots))$ case) assuming the values of $f_{\vec{v},\vec{a}}$ are already computed for all subexpressions of r'. Hence, $O(|r| \cdot |V|^{m+1})$ time is sufficient here. Note that the $f_{\vec{v},\vec{a}}(\delta(\cdots), \bot)$ case can be computed efficiently in $O(|V|)$ time by remembering the number $|\{v \mid f_{\vec{v},\vec{a}}(r', v)\}|$ for each sub-expression r': the existence of $\vec{u}$ can be checked by verifying the number is non-zero, and the check $child(u', i) \neq u_i$ is replaced with "either not $f_{\vec{v},\vec{a}}(r', child(u', i))$ or the number is more than one". Since it is only required to compute the $f_{\vec{v},\vec{a}}(\delta(\cdots), \bot)$ cases at most $|r|$ times, the time complexity for the cases is $O(|r| \cdot |V|)$, which is subsumed by $O(|r| \cdot |V|^{m+1})$. Altogether, multiplying all of them yields the desired complexity bound $O(|s| \cdot |t|^{2m+2} \cdot |M|)$. Note that we have $|V| \le |t| + 1$ by definition, and that the parameter $|M|$ subsumes $\Sigma_{q\in Q, r\in R_{q,\sigma}} |r|$. □

The reader may wonder why the same approach does not work for OI-mtts, whose inverses also preserve the regular tree languages. The problem is, for OI, the states of the inferred automata are in $A = 2^{\bigcup_i Q^{(i)} \times (2^V)^i \times V}$ instead of $A = 2^{\bigcup_i Q^{(i)} \times V^i \times V}$. The difference is intuitively explained as follows: in IO-mtts, every copy of a same parameter is an identical output tree and thus corresponds to a single node in V, while in OI-mtts, each copy is evaluated independently and thus may correspond to different output nodes. To capture this phenomenon in the inverse type inference, each parameter must be represented by a *set of* nodes rather than a *single* output node. The additional exponential implies that a single state in A (a subset of $\bigcup_i Q^{(i)} \times (2^V)^i \times V$) can already be exponentially large. Therefore, on-the-fly construction does not help to obtain a PTIME algorithm. Of course, Lemma 1 implies that there is no PTIME algorithm for translation membership for OI-mtts (unless NP=P).

Nevertheless, some subclasses of OI-mtts still admit PTIME translation membership. Note that the essential difficulty of OI-translation membership comes from the copying of parameters. Consider, for example, an OI-mtt that is linear in the parameters (i.e., in every right-hand side each parameter y_i occurs at most once); then each parameter is either used once or is never used. In this case, it can be represented in the inverse-type automaton by a set of size ≤ 1. More generally, if an OI-mtt is *finite copying* in the parameter, its translation membership can be tested in polynomial time. An mtt is finite copying in the parameter if there exists a constant c such that for any q, s, and $u \in [\![\langle q, s\rangle(y_1, \ldots, y_k)]\!]$, the number of occurrences of y_i in u is no more than c; the number c is called a (parameter) *copying bound* by M. Note that "linear-in-parameter" mtts are a special case of finite copying mtts; they are not only finite copying with copying bound 1, but also the finiteness can be known by simply counting the number of syntactic occurrences of each variable in the rules, while finite copying in general is a semantic property of mtts. Also note that finite copying is a decidable property, and the copying bound can be effectively obtained. (See Lemma 4.10 of [5]. Although it is proved only for total deterministic mtts, the same technique also works for IO- and OI- nondeterministic mtts.)

Theorem 5 Let M be an mtt that is finite copying in the parameters with copying bound c. Then, translation membership for $\tau_{\mathrm{OI},M}$ can be determined in time $O(|s| \cdot |t|^{c(2m+2)} \cdot c \cdot |M|)$ where m is the maximum rank of M's states.

Proof. Let t_{dag} be the minimal dag representing t. Let V be the set of nodes of t_{dag}. We define $label(v)$ to denote the label in Σ of the node $v \in V$, and $child(v, i)$ to denote the i-th child node of v.

Let $A = 2^{\bigcup_i Q^{(i)} \times \mathcal{P}_c(V)^i \times V}$ where $\mathcal{P}_c(V) = \{S \subseteq V \mid |S| \leq c\}$ and the function run be defined as follows:

$$run(\sigma(s_1, \ldots, s_k)) = tr(\sigma, run(s_1), \ldots, run(s_k)).$$

The transition function $tr : (\bigcup_i \Sigma^{(i)} \times A^i) \to A$ is defined as follows

$$tr(\sigma, \vec{a}) = \Big\{(q, \vec{\beta}, v') \in \bigcup_i Q^{(i)} \times \mathcal{P}_c(V)^i \times V \;\Big|\; \exists r \in R_{q,\sigma} : f_{\vec{\beta},\vec{a}}(r, v')\Big\}$$

where $f_{\vec{\beta},\vec{a}} : T_{\Delta \cup (Q \times X) \cup Y} \times V \to \{true, false\}$ defined as follows:

$$f_{\vec{\beta},\vec{a}}(y_i, v') = true \qquad \text{if } v' \in \beta_i$$
$$f_{\vec{\beta},\vec{a}}(y_i, v') = false \qquad \text{if } v' \notin \beta_i$$
$$f_{\vec{\beta},\vec{a}}(\delta(r_1, \ldots, r_n), v') = \bigwedge_{1 \leq i \leq n} f_{\vec{\beta},\vec{a}}(r_i, child(v', i)) \qquad \text{if } label(v') = \delta$$
$$f_{\vec{\beta},\vec{a}}(\delta(r_1, \ldots, r_n), v') = false \qquad \text{if } label(v') \neq \delta$$
$$f_{\vec{\beta},\vec{a}}(\langle q', x_j\rangle(r_1, \ldots, r_n), v') =$$
$$\exists \vec{\gamma} : ((q', \vec{\gamma}, v') \in a_j \text{ and for all } i \text{ and } u \in \gamma_i : f_{\vec{\beta},\vec{a}}(r_i, u)).$$

Note that we do not have the $\perp$ element in V this time. Instead, the empty set $\emptyset$ plays the same role. The complexity of this algorithm is computed similarly to the case of IO-mtts: we need to test by $f_{\vec{\beta},\vec{a}}$ all combinations of $a \in \bigcup_i Q^{(i)} \times \mathcal{P}_c(V)^i \times V$ (which is of size $O(|Q| \cdot |V|^{cm+1})$ this time) and $r \in R_{q,\sigma}$, then $f_{\vec{\beta},\vec{a}}$ receives $|r| \cdot |V|$ different pairs of arguments, and finally the computation of $f_{\vec{v},\vec{a}}(\langle q', x_j\rangle(\cdots))$ takes $O(|V|^{cm} \cdot c)$ time where $|V|^{cm}$ comes from the part "$\exists\vec{\gamma}$" and c comes from the part "$u \in \gamma_i$". The correctness is shown by proving the following claim.

Claim For every input tree s', $t' \in [\![\langle q, s'\rangle(u_1, \ldots, u_n)]\!]^M_{\mathrm{OI}}$ if and only if there exist subtrees $t_{1,1}, \ldots, t_{1,l_1}, \ldots, t_{n,1}, \ldots, t_{n,l_n}$ of t such that $\{t_{i,1}, \ldots, t_{i,n_i}\} \subseteq [\![u_i]\!]^M_{\mathrm{OI}}$ with $l_i \leq c$ and $(q, (\{\rho(t_{1,1}), \ldots, \rho(t_{1,l_1})\}, \ldots, \{\rho(t_{n,1}), \ldots, \rho(t_{n,l_n})\}), \rho(t')) \in run(s')$, where ρ is defined as in the proof of Theorem 4.

The proof is by induction, too. The finite-copying property ensures that in the semantics of the mtt, OI-substitution is done only on parameters y_i whose number of occurrence is less than or equal to c. It justifies that our algorithm only considers sets of size $\leq c$ as parameter representation. □

On the other hand, the PTIME result for IO-mtts can be generalized to a more powerful extension of IO-mtts. One popular way to extend mtts is by *regular look-ahead*. Mtts with regular look-ahead are equipped with one deterministic bottom-up tree automaton and are allowed to select a rule with respect to the state of the tree automaton, in addition to the current state and the label of the current node. Since any $\mathrm{MTT}_{\mathrm{IO}}$'s with regular look-ahead can be simulated by a normal $\mathrm{MTT}_{\mathrm{IO}}$ (Theorem 5.19 of [6]), the translation membership for $\mathrm{MTT}_{\mathrm{IO}}$ with regular look-ahead is also in PTIME. In fact, we can further extend the model to use a more expressive model of look-ahead, namely, tree automata with equality and disequality constraints [1], while still preserving the PTIME translation membership.

Definition 2. A *bottom-up tree automaton with equality and disequality constraints* (TAC) is a tuple $B = (P, \Sigma, \delta)$, where P is the set of states, Σ the input alphabet, and δ is a set of transitions of the form $(\sigma^{(m)}, p_1, \ldots, p_m, E, D, p)$ where $E, D \subseteq \{1, \ldots, m\}^2$ are the sets of equality and disequality constraints, respectively. A list of trees $t_1, \ldots, t_m$ is said to satisfy the constraints if $\forall (i, j) \in E : t_i = t_j$ and $\forall (i, j) \in D : t_i \neq t_j$. We define $\tilde{\delta}$ inductively as follows:

$$\tilde{\delta}(\sigma(t_1, \ldots, t_m)) = \{p \in P \mid \exists(\sigma, p_1, \ldots, p_m, E, D, p) \in \delta : p_i \in \tilde{\delta}(t_i) \text{ for all } i \text{ and } t_1, \ldots, t_m \text{ satisfy } E \text{ and } D\}.$$

A TAC is total and deterministic if for any $\sigma \in \Sigma$, $p_1, \ldots, p_m \in P$, and $t_1, \ldots, t_m \in T_\Sigma$, there exists one unique transition $(\sigma^{(m)}, p_1, \ldots, p_m, E, D, p) \in \delta$ such that $t_1, \ldots, t_m$ satisfies the constraints E and D. For a total deterministic TAC, we abuse the notation and denote by $\tilde{\delta}(t)$ the unique element of itself.

Note that, as well as a normal bottom-up tree automaton, we can run a TAC on a tree in (amortized) linear time, by first computing the minimal dag representation of the input tree; due to its minimality, the equality (or disequality) test of two subtrees can be carried out in constant time, by a single pointer comparison. Also note that total deterministic

TACs are equally expressive as its nondeterministic version (as shown in Proposition 4.2 of [1] by a variant of usual powerset construction). Hence, we adopt total deterministic TACs as our look-ahead model for mtts, without sacrificing the expressiveness.

Definition 3. An *mtt with TAC look-ahead* is a tuple $M = (Q, q_0, \Sigma, \Delta, R, B)$ where $B = (P, \Sigma, \delta)$ is a total and deterministic TAC, and all other components are defined as for mtts, except that the form of rules are as follows:

$$\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r \qquad (p_1, \ldots, p_k, E, D).$$

The set of right-hand side of all rules of such form is denoted by $R_{q,\sigma,p_1,\ldots,p_k,E,D}$. The size $|M|$ is defined as for normal mtts.

The semantics of mtts with TAC look-ahead differs from normal mtts only in the side-condition of state application, which is defined as follows:

$$[\![\langle q, \sigma(s_1, \ldots, s_k)\rangle(u_1, \ldots, u_m)]\!]^M_\mu = \bigcup_{r \in R'} \Big([\![r[x_1/s_1, \ldots, x_k/s_k]]\!]^M_\mu \underset{\mu}{\leftarrow} ([\![u_1]\!]^M_\mu, \ldots, [\![u_m]\!]^M_\mu) \Big)$$

where $R' = R_{q,\sigma,\tilde{\delta}(s_1),\ldots,\tilde{\delta}(s_k),E,D}$ such that $s_1, \ldots, s_k$ satisfies E and D.

In a word, rules in $R_{q,\sigma,p_1,\ldots,p_k,E,D}$ are used when the state q is applied to a node satisfying all the following three conditions: (1) labeled σ, (2) the child subtrees $s_1, \ldots, s_k$ of the node satisfy the constraints E and D, and (3) $\tilde{\delta}(s_i) = p_i$ for all i.

Mtts with TAC look-ahead are strictly more expressive than normal mtts. For example, the translation $\{(\pi(s, s), e) \mid s \in T_\Sigma\}$ where π is a symbol of rank 2 and e is of rank 0, can be done by a transducer with TAC look-ahead. But no mtt-composition can realize this translation because the domain is not regular (by Corollary 5.6 of [6], the domain of any mtt must be a regular tree language). Nevertheless, the PTIME translation membership for $\mathrm{MTT}_{\mathrm{IO}}$ can be extended to mtts with TAC look-ahead.

Theorem 6 Let M be an mtt with TAC look-ahead. Translation membership for $\tau_{\mathrm{IO},M}$ can be determined in time $O(|s| \cdot |t|^{2m+2} \cdot |M|)$ where m is the maximum rank of M's states.

Proof. The basic idea is again the on-the-fly construction of the inverse-type automaton, but this time, to deal with the look-ahead, we run parallely the look-ahead automaton.

Let s_{dag} be the minimal dag representation of s, which can be computed in $O(|s|)$ time. As explained before, the equality (or disequality) test of two subtrees of s_{dag} can be carried out in constant time. Let V_s be the set of nodes of s_{dag}. Let V_t be the set of nodes of t_{dag} and $V = V_t \cup \{\bot\}$. The functions $\mathit{label}(v)$, $\mathit{child}(v, i)$, and $\rho(t)$ are defined as in the proof of Theorem 4.

Let $A = 2^{\bigcup_i Q^{(i)} \times V^i \times V}$ and $\mathit{run} : T_\Sigma \to V_s \times P \times A$ (note the difference of the return value of *run*, compared to that in Theorem 4) be the function defined as follows

$$\mathit{run}(s') = \mathit{tr}(s', \sigma, \mathit{run}(s_1), \ldots, \mathit{run}(s_k)) \quad \text{with } s' = \sigma(s_1, \ldots, s_k)$$

where the function tr is:

$$\mathit{tr}(s', \sigma, (s_1, p_1, a_1), \ldots, (s_k, p_k, a_k)) = \Big(s', \tilde{\delta}(s'), \big\{(q, \vec{v}, v') \in \bigcup_i Q^{(i)} \times V^i \times V \;\big|\; \exists r \in R_{q,\sigma,p_1,\ldots,p_k,E,D} : (s_1, \ldots, s_k) \text{ satisfies } E, D \text{ and } f_{\vec{v},\vec{a}}(r, v')\big\}\Big).$$

The definition of $f_{\vec{v},\vec{a}}$ remains the same as in Theorem 4.

The look-ahead state $\tilde{\delta}(s')$ can be computed from σ, $p_1, \ldots, p_k$, and $s_1, \ldots, s_k$ in constant time. By the same argument as the case of normal mtts, we obtain the $O(|s| \cdot |t|^{2m+2} \cdot |M|)$ time complexity. The correctness of the construction is proved also in the same way as for normal mtts. That is, we can prove the following claim by nested induction on structure of s', and then on the structure of right-hand sides of the rules.

Claim For every input tree s', we have the following equation for all $q \in Q$, $r_i \in T_{\Delta \cup (Q \times T_\Sigma) \cup Y}$, and an environment Γ: $\rho\Big([\![\langle q, s'\rangle(r_1, \ldots, r_n)]\!]^M_{\mathrm{IO}}\Big) = \Big\{v' \;\Big|\; (q, (v_1, \ldots, v_n), v') \in \mathit{run}(s'), v_i \in \rho([\![r_i]\!]^M_{\mathrm{IO}}) \text{ for all } i\Big\}$

Again, applying the claim to $\rho([\![\langle q_0, s\rangle]\!]^M_{\mathrm{IO}})$, we know that the translation membership is equivalent to $(q_0, (), v_\epsilon) \in \mathit{run}(s)$ where v_ϵ is the root node of t_{dag}. Hence, the translation membership can be determined by computing the set $\mathit{run}(s)$. □

Another extension of mtts that admits a polynomial time translation membership is *multi-return mtts (mr-mtts)* [9, 10]. In an mr-mtt, states may return multiple trees (with the initial state returning exactly one tree). Mr-mtts are strictly more expressive than normal mtts, and furthermore, have better closure properties under composition with top-down tree transducers [10].

Definition 4. A *multi-return macro tree transducer (mtt)* M is a tuple $(Q, \Sigma, \Delta, q_0, R, D)$, where Q, Σ, Δ, and q_0 are defined as for mtts, $D : Q \to \mathbb{N}$ is the *dimension* such that $D(q_0) = 1$, and R is the finite set of *rules* of the form

$$\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r$$

where $q \in Q^{(m)}$, $\sigma \in \Sigma^{(k)}$, and $r \in \mathrm{rhs}^{D(q)}_{X_k}$ where for $e \geq 1$ and a set Q, the set rhs^e_W is defined as:

$$r ::= l_1 \ldots l_n\ (u_1, \ldots, u_e) \qquad (n \geq 0)$$

$$l ::= \text{let } (z_{j+1}, \ldots, z_{j+D(q')}) = \langle q', x_i\rangle(u_1, \ldots, u_n) \text{ in} \qquad (j \in \mathbb{N}, q' \in Q^{(n)}, x_i \in W)$$

with $u_1, u_2, \ldots \in T_{\Delta \cup Y_m \cup Z}$. We usually omit parentheses around tuples of size one, i.e., write like let $z_j = \cdots$ in u_1.

We require any rule to be well-formed, that is, the leftmost occurrence of any variable z_i must appear at a "binding" position (between 'let' and '='), and the next occurrence (if any) must appear after the 'in' corresponding to the binding occurrence. The set of right-hand sides of such rules is denoted by $R_{q,\sigma}$. The size $|M|$ of the mr-mtt is defined to be the sum of the size of right-hand sides, i.e., the number of δ, Y, Z, and $Q \times X$ nodes.

The IO-semantics of mr-mtts is inductively defined as follows. For $u \in T_{\Delta \cup Y_m \cup Z}$, $[\![u]\!]^M_{\mathrm{IO}} \subseteq T_{\Delta \cup Y \cup Z}$ is

$$[\![\delta(u_1, \ldots, u_k)]\!]^M_{\mathrm{IO}} = \{\delta(t_1, \ldots, t_e) \mid t_i \in [\![u_i]\!]^M_{\mathrm{IO}} \text{ for all } i\}$$
$$[\![y_i]\!]^M_{\mathrm{IO}} = \{y_i\}$$
$$[\![z_i]\!]^M_{\mathrm{IO}} = \{z_i\}$$

and for $\kappa \in \mathrm{rhs}^e_{T_\Sigma}$, $[\![\kappa]\!]^M_{\mathrm{IO}} \subseteq T^e_{\Delta \cup Y \cup Z}$ is

$$[\![(u_1, \ldots, u_e)]\!]^M_{\mathrm{IO}} = \{(t_1, \ldots, t_e) \mid t_i \in [\![u_i]\!]^M_{\mathrm{IO}} \text{ for all } i\}$$
$$[\![\text{let } (z_1, \ldots, z_d) = \langle q, \sigma(s_1, \ldots, s_m)\rangle(u_1, \ldots, u_k) \text{ in } \kappa']\!]^M_{\mathrm{IO}}$$
$$= \Big\{\xi[z_1/t_1, \ldots, z_d/t_d] \mid \xi \in [\![\kappa']\!]^M_{\mathrm{IO}}, r \in R_{q,\sigma}, (t_1, \ldots, t_d) \in \Big([\![r[x_1/s_1, \ldots, x_m/s_m]]\!]^M_{\mathrm{IO}} \underset{\mathrm{IO}}{\longleftarrow} ([\![u_1]\!]^M_{\mathrm{IO}}, \ldots, [\![u_k]\!]^M_{\mathrm{IO}})\Big)\Big\}.$$

The translation $\tau_{\mathrm{IO},M} \subseteq T_\Sigma \times T_\Delta$ realized by M is the set $\{(s,t) \mid t \in [\![\text{let } z = \langle q_0, s\rangle \text{ in } z]\!]^M_{\mathrm{IO}}\}$.

Here is an example of an mr-mtt, which is used in [9] as a counterexample that cannot be realized in normal mtts:

$$\langle q_0, \mathtt{s}(x)\rangle() \to \text{let } (z_1, z_2) = \langle q_1, x\rangle(\mathtt{A}(\mathtt{E})) \text{ in } \mathtt{r}(\mathtt{a}(z_1), z_2)$$
$$\langle q_0, \mathtt{s}(x)\rangle() \to \text{let } (z_1, z_2) = \langle q_1, x\rangle(\mathtt{B}(\mathtt{E})) \text{ in } \mathtt{r}(\mathtt{b}(z_1), z_2)$$
$$\langle q_0, \mathtt{z}\rangle() \to \mathtt{r}(\mathtt{e}, \mathtt{E})$$
$$\langle q_1, \mathtt{s}(x)\rangle(y_2) \to \text{let } (z_1, z_2) = \langle q_1, x\rangle(\mathtt{A}(y_2)) \text{ in } (\mathtt{a}(z_1), z_2)$$
$$\langle q_1, \mathtt{s}(x)\rangle(y_2) \to \text{let } (z_1, z_2) = \langle q_1, x\rangle(\mathtt{B}(y_2)) \text{ in } (\mathtt{b}(z_1), z_2)$$
$$\langle q_1, \mathtt{z}\rangle(y_2) \to (\mathtt{e}, y_2)$$

This nondeterministic translation takes as input monadic trees of the form $\mathtt{s}(\mathtt{s}(\cdots \mathtt{s}(\mathtt{z}) \cdots))$ and produces output trees of the form $\mathtt{r}(t_1, t_2)$ where t_1 is a monadic tree over $\mathtt{a}$'s and $\mathtt{b}$'s (and a leaf $\mathtt{e}$), and t_2 is a monadic tree over $\mathtt{A}$'s and $\mathtt{B}$'s such that t_2 is the reverse of t_1, and both have the same size as the input. For instance, $\mathtt{r}(\mathtt{a}(\mathtt{a}(\mathtt{b}(\mathtt{e}))), \mathtt{B}(\mathtt{A}(\mathtt{A}(\mathtt{E}))))$ is a possible output tree for the input $\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{z})))$. Consider the return value of the state call $[\![\langle q_1, \mathtt{s}(\mathtt{z})\rangle(\mathtt{E})]\!]$: it is the set $\{(\mathtt{a}(\mathtt{E}), \mathtt{A}(\mathtt{E})), (\mathtt{b}(\mathtt{E}), \mathtt{B}(\mathtt{E}))\}$ of *pairs of* trees. In a word, the state q_1 returns only *mutually reverse* pairs of monadic trees. This is impossible in normal mtts, in which we must carry out two state calls in order to obtain two output trees; two nondeterministic state calls are evaluated independently, and cannot avoid generating unrelated pairs of trees.

Despite their expressive power over normal mtts, mr-mtts still have a similar complexity for inverse type inference. Therefore the translation membership remains in PTIME.

Theorem 7 Let M be an mr-mtt. Translation membership for $\tau_{\mathrm{IO},M}$ can be determined in time $O(|s| \cdot |t|^{2m+2d} \cdot |M|)$ where m is the maximum rank of the states and d is the maximum dimension.

PROOF. For mr-mtts, we take the set A of inverse-type automaton as $A = 2^{\bigcup_{i,j} Q^{(i,j)} \times V^i \times V^j}$ where $Q^{(i,j)}$ is the set of states q of $\mathit{rank}(q) = i$ and $D(q) = j$. The intuition of the set of states A is similar to the case of normal mtts. That is, "$(q, \vec{v}, \vec{w}) \in \mathit{run}(s')$" means that "if q is applied to the input subtree s' with output subtrees rooted at $\vec{v}$ as parameters, then it may return a tuple of output subtrees $\vec{w}$". The construction is quite similar to that of the proof of Theorem 4. □

As a final remark we would like to mention the complexity of translation membership for deterministic mtts; it can be determined in linear time. Since domains of compositions of mtts are regular, we can factor out the partiality and have the following decomposition: for $\mu \in \{\mathrm{IO}, \mathrm{OI}\}$, $\mathrm{DMTT}^n_\mu \subseteq \mathrm{FTA}\,;\mathrm{DtMTT}^n$ where FTA is the class of partial identities whose domain is regular (analogous to Theorem 6.18 of [6]). Therefore, to compute the translation membership for a composition of deterministic mtts, we first check in $O(|s|)$ time whether the given input s is contained in the domain of the translation, and then check the translation membership for composition of deterministic *and total* mtts. Here, by Theorem 15 of [12], for a translation $\tau \in \mathrm{DtMTT}^n$ we can compute the unique output tree $t' \in \tau(s)$ from the input s in time $O(|s| + |t'|)$, and during the computation, the size of every intermediate tree is less than or equal to $2^n \cdot |t'|$. Hence, for testing $(s,t) \in \tau$, we simply compute $\tau(s)$; if the size of any intermediate tree exceeds $2^n \cdot |t|$ then (s,t) cannot be an element of τ, and otherwise, we compare the computed tree $\tau(s)$ with t. The time complexity of the above procedure is $O(|s| + 2^n \cdot |t|)$.

Theorem 8 Let $\mu \in \{\mathrm{IO}, \mathrm{OI}\}$ and $n \geq 1$. Translation membership for DMTT^n_μ is in $O(|s| + 2^n|t|)$.

5. FUTURE WORK

The complexity of the translation membership problem remains open for several interesting subclasses and extensions of mtts. One example is the mtt with holes [14] in IO mode. Note that, similar to Theorem 4.6 of [14], hole-mtts in IO mode are equal to $\mathrm{MTT}_{\mathrm{IO}}\,;\mathrm{YIELD}$, which is included in $\mathrm{MTT}_{\mathrm{IO}}\,;\mathrm{LDtMTT}$. An algorithm based on inverse type inference does not work, because the parameter part of the states of the inverse-type automaton is a set of functions $[V \to V]$, which is exponential in size with respect to the output tree $|t|$. On the other hand, it is not clear either whether it is NP-hard. Note that mtts with holes in OI mode can simulate all OI-mtts, and therefore their translation membership is NP-complete.

Another interesting class is that of *1-parameter* mtts in OI mode. Our encoding of 3-SAT used three parameters. In fact, the number of parameters can be reduced to two by embedding the encodings of boolean variables in the input tree s. Can we encode 3-SAT into a 1-parameter mtt? Or, do 1-parameter mtts actually have PTIME translation membership? (Again, the inverse-type automaton technique used in this paper for IO-mtts does not seem to work in this case, because the automaton gets too large.)

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

6. REFERENCES

[1] B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, 1992.

[2] B. Courcelle. Monadic second-order definable graph transductions: A survey. *Theoretical Computer Science*, 126:53–75, 1994.

[3] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27:758–771, 1980.

[4] J. Engelfriet and S. Maneth. A comparison of pebble tree transducers with macro tree transducers. *Acta Informatica*, 39:613–698, 2003.

[5] J. Engelfriet and S. Maneth. Macro tree translations of linear size increase are mso definable. *SIAM Journal on Computing*, 32:950–1006, 2003.

[6] J. Engelfriet and H. Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31:71–146, 1985.

[7] M. J. Fischer. *Grammars with Macro-Like Productions*. PhD thesis, Harvard University, Cambridge, 1968.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[9] K. Inaba and H. Hosoya. XML transformation language based on monadic second order logic. In *Programming Language Technologies for XML (PLAN-X)*, pages 49–60, 2007.

[10] K. Inaba, H. Hosoya, and S. Maneth. Multi-return macro tree transducers. In *Conference on Implementation and Application of Automata (CIAA)*, 2008.

[11] K. Inaba and S. Maneth. The complexity of tree transducer output languages. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2008 (Available at http://arbre.is.s.u-tokyo.ac.jp/~kinaba/fst.pdf).

[12] S. Maneth. The complexity of compositions of deterministic tree transducers. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2002.

[13] S. Maneth, A. Berlea, T. Perst, and H. Seidl. XML type checking with macro tree transducers. In *Principles of Database Systems (PODS)*, 2005.

[14] S. Maneth and K. Nakano. XML type checking for macro tree transducers with holes. In *Programming Language Technologies for XML (PLAN-X)*, 2008.

[15] S. Maneth, T. Perst, and H. Seidl. Exact XML type checking in polynomial time. In *International Conference on Database Theory (ICDT)*, 2007.

[16] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66:66–97, 2003.

[17] T. Perst and H. Seidl. Macro forest transducers. *Information Processing Letters*, 89:141–149, 2004.

[18] W. C. Rounds. Complexity of recognition in intermediate-level languages. In *Foundations of Computer Science (FOCS)*, 1973.