Multi-Return Macro Tree Transducers

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

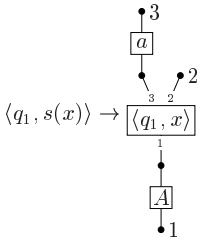
[1] University of Tokyo, {kinaba,hahosoya}@is.s.u-tokyo.ac.jp
[2] National ICT Australia, sebastian.maneth@nicta.com.au
[3] University of New South Wales, Sydney

Abstract. An extension of macro tree transducers is introduced with the capability of states to return multiple trees at the same time. Under call-by-value semantics, the new model is strictly more expressive than call-by-value macro tree transducers, and moreover, it has better closure properties under composition.

1 Introduction

Macro tree transducers (mtts) [1, 2] are a finite-state machine model for tree translation. They are motivated by compilers and syntax-directed semantics and more recently have been applied to XML transformations and query languages [3, 4]. An mtt processes the input tree top-down, starting in its initial state at the root node. Depending on its state and the label of the current input node, it produces an output subtree which possibly contains recursive state calls to children of the current node. State calls may appear at internal nodes of the output and can thus be nested. Technically speaking, this means that a (state, current label)-rule is parameterized by a sequence of arbitrary output trees. The number of such "accumulating parameters" is fixed for each state of the transducer. The initial state has zero parameters, because we are interested in tree-to-tree, not (tuple of trees)-to-tree translations. If every state has zero parameters, then we obtain an ordinary top-down tree transducer [5, 6], in which all state calls appear at leaves of output rule trees. It is well-known that accumulating parameters add power: mtts realize strictly more translations than top-down tree transducers (for instance, top-down tree transducers have at most exponential size increase while mtts can have double-exponential increase). However, mtts have the asymmetry that, while each state can propagate multiple output trees in a top-down manner in its accumulating parameters, it cannot do it in a bottom-up manner because it is still restricted to return only a single output tree and such a tree cannot be decomposed once created.

This paper introduces an extension of mtts called *multi-return macro tree transducer* (mr-mtt) that addresses this asymmetry. In an mr-mtt, states may return multiple trees (but a fixed number for each state, with the initial state returning exactly one tree). As an example, consider a nondeterministic translation $twist$ that takes as input monadic trees of the form $\mathtt{s}(\mathtt{s}(\ldots\mathtt{s}(\mathtt{z})\ldots))$ and produces output trees of the form $\mathtt{root}(t_1, t_2)$ where t_1 is a monadic tree over a's and b's (and a leaf e), and t_2 is a monadic tree over A's and B's such that t_2 is the reverse of t_1, and both have the same size as the input. For instance, $\mathtt{root}(\mathtt{a}(\mathtt{a}(\mathtt{b}(\mathtt{e}))), \mathtt{B}(\mathtt{A}(\mathtt{A}(\mathtt{E}))))$ is a possible output tree for the input $\mathtt{s}(\mathtt{s}(\mathtt{s}(\mathtt{z})))$. Such a translation can be realized by an mr-mtt with the rules of Fig. 1. The



$$\begin{aligned}
\langle q_0, \mathtt{s}(x)\rangle() &\to \text{ let } (z_1, z_2) = \langle q_1, x\rangle(\mathtt{A}(\mathtt{E})) \text{ in } \mathtt{root}(\mathtt{a}(z_1), z_2)\\
\langle q_0, \mathtt{s}(x)\rangle() &\to \text{ let } (z_1, z_2) = \langle q_1, x\rangle(\mathtt{B}(\mathtt{E})) \text{ in } \mathtt{root}(\mathtt{b}(z_1), z_2)\\
\langle q_0, \mathtt{z}\rangle() &\to \mathtt{root}(\mathtt{e}, \mathtt{E})\\
\langle q_1, \mathtt{s}(x)\rangle(y_1) &\to \text{ let } (z_1, z_2) = \langle q_1, x\rangle(\mathtt{A}(y_1)) \text{ in } (\mathtt{a}(z_1), z_2)\\
\langle q_1, \mathtt{s}(x)\rangle(y_1) &\to \text{ let } (z_1, z_2) = \langle q_1, x\rangle(\mathtt{B}(y_1)) \text{ in } (\mathtt{b}(z_1), z_2)\\
\langle q_1, \mathtt{z}\rangle(y_1) &\to (\mathtt{e}, y_1)
\end{aligned}$$

Fig. 1. Rules of the multi-return mtt realizing $twist$

state q_1 is "multi-return"; it generates pairs of trees. The first component is generated in a top-down manner: at each input s-node, an a-labeled output node is generated which has below it the first component (z_1) of the recursive q_1-call at the child of the current input node. This is the left branch of the whole output tree. The right branch is obtained by the second component and is generated in a bottom-up manner through the accumulating parameter of q_1. As we will show, the above translation *twist*, cannot be realized by any conventional mtt. The proof of inexpressibility is technically involved and uses special normal forms of $twist$ in order to derive a contradiction. In general it is very difficult to prove that a given translation cannot be realized by a tree transducer class, because hardly any tools exist for showing inexpressibility. Note that multi-return mtts have the same size increase as mtts.

In the case of deterministic and total deterministic transducers, mr-mtts are equally powerful as mtts. For the total deterministic case this already follows from the fact that tree generating top-down tree-to-graph transducers (trgen-tg) realize the same translations as total deterministic mtts [7]. Mr-mtts can be seen as particular trgen-tgs; e.g., the forth rule of $twist$ is depicted as trgen-tgs rule in the left of Fig. 1.

Besides an increase in expressive power, mr-mtts have better closure properties than mtts: they are closed under left *and* right composition with total deterministic top-down tree transducers (D_tTs). This is rather surprising, because ordinary call-by-value mtts are not closed under composition with D_tTs. The latter was already shown in [1] for the case of left-composition. The case of right-composition is proved in this paper (using *twist*). In fact, our proof can even be "twisted" to the call-by-name semantics of mtts to show that call-by-name mtts are also not closed under right-composition with D_tT. Thus, the two main classes of mtts, call-by-value and call-by-name are both not closed under right-composition with D_tT, while call-by-value multi-return mtts are closed.

2 Definitions

A set Σ with a mapping *rank* : $\Sigma \to \mathbb{N}$ is called a *ranked set*. We often write $\sigma^{(k)}$ to indicate that *rank*$(\sigma) = k$. The *product* of a ranked set A and a set B is the ranked set $A \times B = \{\langle a, b\rangle^{(k)} \mid a^{(k)} \in A, b \in B\}$. The set T_Σ of *trees* t over a ranked set Σ is defined by the BNF $t ::= \sigma(t_1, \ldots, t_k)$ for $\sigma^{(k)} \in \Sigma$. We often omit parentheses for rank-0 and rank-1 symbols and write them as strings. For example, we write abcd instead of a(b(c(d()))).

When $x_1^{(0)}, \ldots, x_m^{(0)} \in \Sigma$ and $t, t_1, \ldots, t_m \in T_\Sigma$, *(simultaneous) substitution* of $t_1, \ldots, t_m$ for $x_1, \ldots, x_m$ in t is written $t[x_1/t_1, \ldots, x_m/t_m]$ (or sometimes $t[\vec{x}/\vec{t}]$ for brevity) and defined to be a tree where every occurrence of x_i ($i = 1, \ldots, m$) in t is replaced by the corresponding t_i. For a ranked set Σ and rank-0 symbol $\Box \notin \Sigma$, a tree $\mathcal{C} \in T_{\Sigma \cup \{\Box\}}$ that contains exactly one occurrence of $\Box$ is called a *one-hole Σ-context*. We write $\mathcal{C}[t]$ as a shorthand for $\mathcal{C}[\Box/t]$.

Macro Tree Transducers Throughout the paper, we fix the sets of input variables $X = \{x_1, x_2, \ldots\}$ and accumulating parameters $Y = \{y_1, y_2, \ldots\}$ which are all of rank 0 and assume any other ranked set to be disjoint with X and Y. The set X_i is defined as $\{x_1, \ldots, x_i\}$, and Y_i is defined similarly.

A *macro tree transducer* (mtt) is specified as $M = (Q, q_0, \Sigma, \Delta, R)$, where Q, Σ, and Δ are finite ranked sets. We call Q the set of *states*, $q_0 \in Q$ the *initial state* of rank 0, Σ the *input alphabet*, Δ the *output alphabet*, and R the finite set of translation *rules* of the form $\langle q^{(m)}, \sigma^{(k)}(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r$ where the right-hand side r is a tree from $T_{\Delta \cup (Q \times X_k) \cup Y_m}$. Rules of this form are called $\langle q, \sigma\rangle$-rules. An mtt is *deterministic* (*total*, respectively) if there exists at most (at least) one $\langle q, \sigma\rangle$-rule for every $\langle q, \sigma\rangle \in Q \times \Sigma$. Also, an mtt is *linear* if the right-hand side of every rule in R contains at most one occurrence of x_i for each $x_i \in X$. We define the ranked set $\Lambda_M = \Delta \cup (Q \times T_\Sigma)$ and call trees in T_{Λ_M} the *sentential forms* of M. The translation realized by M is defined in terms of the rewrite relation $\Rightarrow_M$ over sentential forms. The "one-step derivation" relation that we use, is the *call-by-value* (also known as *IO-mode*) derivation relation. Let $u, u' \in T_{\Lambda_M}$. Then $u \Rightarrow_M u'$ if there is a $\langle q, \sigma\rangle$-rule in R with right-hand side r, a one-hole Λ_M-context $\mathcal{C}$, input trees $s_1, \ldots, s_k \in T_\Sigma$, and output trees $t_1, \ldots, t_m \in T_\Delta$, such that $u = \mathcal{C}[\langle q, \sigma(\vec{s})\rangle(\vec{t})]$ and $u' = \mathcal{C}[r[\vec{x}/\vec{s}, \vec{y}/\vec{t}]]$. We define $u{\downarrow_M} = \{t \in T_\Delta \mid u \Rightarrow_M^* t\}$ and the translation $\tau(M)$ realized by M as $\{\langle s, t\rangle \in T_\Sigma \times T_\Delta \mid t \in \langle q_0, s\rangle{\downarrow_M}\}$. The class of translations realized by mtts is denoted by *MT*. The restricted class of translations realized by deterministic (total, or linear) transducers is denoted by prefix D ($_t$, or L, respectively). An mtt with all its states of rank-0 (i.e., without accumulation parameters) is called *top-down tree transducer* and abbreviated as tt; the corresponding class of translations is denoted by T. As a special case, a linear total deterministic tt with one state only is also called *linear tree homomorphism* and the corresponding class of translations is denoted by *LHOM*.

The operator ; denotes sequential composition. That is, $\tau_1 ; \tau_2 = \{(s, t) \mid \exists w.(s, w) \in \tau_1, (w, t) \in \tau_2\}$ for two translations τ_1 and τ_2, and $A_1 ; A_2 = \{\tau_1 ; \tau_2 \mid \tau_1 \in A_1, \tau_2 \in A_2\}$ for two classes of translations A_1 and A_2.

Multi-Return Macro Tree Transducers The multi-return macro tree transducer extends mtt by construction and deconstruction (via let expressions) of tuples of return values. Each state now has a "dimension" which is the number of trees it returns. In addition to X and Y, we fix the set $Z = \{z_1, z_2, \ldots\}$ of let-variables, of rank 0, and assume it to be disjoint with any other ranked set.

Definition 1. A *multi-return macro tree transducer* (mr-mtt) of dimension $d \geq 1$ is a tuple $(Q, q_0, \Sigma, \Delta, R, D)$, where Q, q_0, Σ, and Δ are as for mtts, D is a mapping from Q to $\{1, \ldots, d\}$ such that $D(q_0) = 1$, and R is a set of rules of the form $\langle q^{(m)}, \sigma^{(k)}(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r$ where $r \in rhs^{D(q)}$ and, for $e \geq 1$ the set

rhs^e is defined as:

$$r ::= l_1 \dots l_n\ (u_1, \dots, u_e) \qquad (n \geq 0)$$
$$l ::= \text{let}\,(z_{j+1}, \dots, z_{j+D(q')}) = \langle q'^{(k)}, x_i \rangle (u_1, \dots, u_k)\ \text{in} \qquad (x_i \in X_k)$$

with $u_1, u_2, \dots \in T_{\Delta \cup Y_m \cup Z}$. 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 left-most 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 occurence. *Total*, *deterministic*, and *linear* mr-mtts are defined as for mtts.

As for mtts, the call-by-value semantics of mr-mtts is defined in terms of rewriting over sentential forms. Note that the target of rewriting is always the state call in the very first let expression of a given sentential form (this expression cannot have let-variables). Let $M = (Q, q_0, \Sigma, \Delta, R, D)$ be an mr-mtt. The set K_M of sentential forms κ of M is defined by the following BNF

$$\kappa ::= l_1 \dots l_n\ u_1$$
$$l ::= \text{let}\,(z_{j+1}, \dots, z_{j+D(q)}) = \langle q^{(k)}, s \rangle (u_1, \dots, u_k)\ \text{in}$$

where $s \in T_\Sigma$ and $u_1, u_2, \dots \in T_{\Delta \cup Z}$. Again, we require the sentential forms to be well-formed in the same sense as for the right-hand sides of rules. Let $\kappa_1, \kappa_2 \in K_M$. Then $\kappa_1 \Rightarrow_M \kappa_2$ if κ_1 has the form let $(z_{j+1}, \dots, z_{j+D(q)}) = \langle q^{(m)}, \sigma^{(k)}(s_1, \dots, s_k) \rangle$ $(t_1, \dots, t_m)$ in κ where $s_i \in T_\Sigma$ and $t_i \in T_\Delta$, and there is a $\langle q,\sigma \rangle$-rule in R with the right-hand side $l_1 \dots l_n\ (u_1, \dots, u_{D(q)})$ and κ_2 has the form $l'_1 \dots l'_n \kappa'$ where

$$l'_i = l_i[x_1/s_1, \dots, x_k/s_k, y_1/t_1, \dots, y_m/t_m] \qquad (i = 1, \dots, n)$$
$$u'_k = u_k[y_1/t_1, \dots, y_m/t_m] \qquad (k = 1, \dots, D(q))$$
$$\kappa' = \kappa[z_{j+1}/u'_1, \dots, z_{j+D(q)}/u'_{D(q)}].$$

Here, we adopt the standard convention that substitution automatically avoids inappropriate variable capture by silently renaming let-variables.

We define $\kappa{\downarrow_M} = \{t \in T_\Delta \mid \kappa \Rightarrow_M^* t\}$. The translation $\tau(M)$ realized by M is defined as $\{(s,t) \in T_\Sigma \times T_\Delta \mid t \in (\text{let}\, z = \langle q_0, s \rangle \text{ in } z){\downarrow_M}\}$. The class of translations realized by mr-mtts is denoted by *MM*. By *d-MM* with $d \geq 1$, we denote the class of translations realized by mr-mtts of dimension d. The prefixes D, $_t$, and L are used in the same way as for mtts. Note that *MT* $\subseteq$ *1-MM* (by replacing each nested state call of the mtt by a let-binding), with determinism and totality being preserved.

For technical convenience, we sometimes regard rhs^d as a subset of $T_{\Delta \cup (Q \times X_k) \cup Y_m \cup Z \cup L^d_{mr}}$ where $L^d_{mr} = \{\text{let}_1^{(4)}, \dots, \text{let}_d^{(3+d)}, \text{tup}_1^{(1)}, \dots, \text{tup}_d^{(d)}\}$, which should be understood as the abstract syntax tree of its textual representation.

3 Simulation of Multi-Return MTTs by MTTs

This section shows that any mr-mtt can be decomposed into a three-fold composition of simpler transducers, namely, a pre-processor for dealing with let-bindings, an mtt doing the essential translation, and a post-processor for dealing with tuples.

Tuple Return Values Following the well-known (Mezei-Wright-like [8]) tupling-selection technique, we use special symbols to represent tuples and selection. For $n \geq 2$, define $L^n_{tup} = \{\tau^{(2)}_2, \ldots, \tau^{(n)}_n, \pi^{(1)}_1, \ldots, \pi^{(1)}_n\}$. Intuitively, τ_i means "construct a tuple of i elements" and π_i means "select the i-th element of". We define the transducer $tups^n_\Delta$ whose purpose is to recursively convert subtrees of the form $\pi_i(\tau_k(t_1, \ldots, t_k))$ into t_i. The *tupling-and-selection transducer* tups^n_Δ is the linear deterministic total top-down tree transducer with input alphabet $\Delta \cup L^n_{tup}$, output alphabet Δ, set of states $\{q_1, \ldots, q_n\}$, initial state q_1, and the following rules for each q_i:

$$\begin{aligned}
\langle q_i, \pi_k(x_1)\rangle &\to \langle q_k, x_1\rangle \\
\langle q_i, \tau_k(x_1, \ldots, x_k)\rangle &\to \langle q_1, x_i\rangle \text{ if } 1 \leq i \leq k \\
&\to \langle q_1, x_1\rangle \text{ otherwise} \\
\langle q_i, \delta(x_1, \ldots, x_m)\rangle &\to \delta(\langle q_1, x_1\rangle, \ldots, \langle q_1, x_m\rangle) \text{ for } \delta^{(m)} \in \Delta, m \geq 0.
\end{aligned}$$

Lemma 2. *MM $\subseteq$ 1-MM$\,;LD_tT$. Totality, determinism, and numbers of rules and parameters are preserved.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, R, D)$ be an mr-mtt of dimension d. We define another mr-mtt $M' = (Q, \Sigma, \Delta \cup L^d_{tup}, q_0, R', D')$, where $D'(q) = 1$ for all $q \in Q$ and $R' = \{\langle q, \sigma(\vec{x})\rangle(\vec{y}) \to et(r) \mid \langle q, \sigma(\vec{x})\rangle(\vec{y}) \to r \in R\}$. The *explicit-tupling* function et is inductively defined as follows:

$$\begin{aligned}
et(u_1) &= u_1 \\
et((u_1, \ldots, u_e)) &= \tau_e(u_1, \ldots, u_e) && \text{if } e > 1 \\
et(\text{let } z_1 = \langle q, x\rangle(\vec{u}) \text{ in } r') &= \text{let } z_1 = \langle q, x\rangle(\vec{u}) \text{ in } et(r') \\
et(\text{let } (z_1, \ldots, z_m) = \langle q, x\rangle(\vec{u}) \text{ in } r') &= \\
\text{let } z_1 = \langle q, x\rangle(\vec{u}) \text{ in } &et(r'[z_1/\pi_1(z_1), \ldots, z_m/\pi_m(z_1)])) && \text{if } m > 1.
\end{aligned}$$

We also apply et to sentential forms in K_M, and tups^d_Δ to sentential forms in $K_{M'}$. Then for all $\kappa_1, \kappa_2 \in K_M$ and $\kappa'_1, \kappa'_2 \in K_{M'}$ such that $et(\kappa_1) = \tau(\mathit{tups}^d_\Delta)(\kappa'_1)$, $\kappa_1 \Rightarrow_M \kappa_2$, and $\kappa'_1 \Rightarrow_{M'} \kappa'_2$ assuming that the two derivations are done by corresponding rules, we have $et(\kappa_2) = \tau(\mathit{tups}^d_\Delta)(\kappa'_2)$. By induction on the number of derivation steps, we have that $t' \in \tau(M')(s)$ if and only if $\tau(tups^d_\Delta)(t') \in \tau(M)(s)$. Thus, $\tau(M) = \tau(M')\,;\tau(\mathit{tups}^d_\Delta)$ which proves the lemma. □

Let-Bindings Even without multiple return values, let-bindings still provide some additional power with respect to ordinary mtts. For example, the right-hand side of an mr-mtt rule let $z = \langle q, x\rangle$ in $\delta(z, z)$ is not necessarily equivalent to the mtt one $\delta(\langle q, x\rangle, \langle q, x\rangle)$. In the former rule, the two children of δ must be the same tree that is returned by a single state call $\langle q, x\rangle$. On the other hand, in the latter rule, two state calls $\langle q, x\rangle$ may return different trees due to nondeterminism. Thus, for simulating let-bindings we must first fully evaluate state calls to an output tree and *then* copy them if required. Basically, such order of evaluation can be simulated using accumulating parameters and state calls, since we adopt call-by-value semantics. For instance, the above example of mr-mtt rule is equivalent to the mtt rule $\langle p, x\rangle(\langle q, x\rangle)$ using an auxiliary state p and a set of auxiliary rules $\langle p, \sigma(\vec{x})\rangle(y) \to \delta(y, y)$ for every $\sigma \in \Sigma$.

However, this approach does not work for nested let-bindings. The problem is that the calls of auxiliary states to simulate copying must be applied to some child of the current node. Consider the following rule:

$$\begin{aligned}\langle q, \sigma(x_1, \ldots, x_n)\rangle \to \text{let } z_1 = \langle q_1, x_1\rangle \text{ in}\\ \text{let } z_2 = \langle q_2, x_2\rangle(z_1) \text{ in}\\ \vdots\\ \text{let } z_n = \langle q_n, x_n\rangle(z_1, \ldots, z_{n-1}) \text{ in } \delta(z_1, \ldots, z_n).\end{aligned}$$

To simulate the first let-binding, we need an auxiliary state call like $\langle p, x_i\rangle(\langle q_1, x_1\rangle)$, and we do the rest of the work in the $\langle p, \sigma\rangle$-rules. But this time, we have to generate other state calls such as $\langle q_2, x_2\rangle(z_1)$ in the auxiliary rule, which is impossible since in $\langle p, x_i\rangle$ we are only able to apply states to the *children* of x_i, while x_2 is a *sibling* of x_i.

One possible solution is to insert auxiliary nodes of rank 1 above each node of the input tree, similar as done for the removal of stay moves in [9]. We can then run the auxiliary states on the inserted nodes in order to simulate the let-bindings. For instance, the first two lets of the above rule can be simulated by

$$\begin{aligned}\langle q, \bar\sigma_1(x_1)\rangle &\to \langle p, x_1\rangle(\langle\langle q_1, 1\rangle, x_1\rangle, \alpha)\\ \langle p, \bar\sigma_2(x_1)\rangle(y_1, y_2) &\to \langle p, x_1\rangle(y_1, \langle\langle q_2, 2\rangle, x_1\rangle(y_1)),\end{aligned}$$

where α is an arbitrary output symbol of rank 0. The new auxiliary state $\langle q', i\rangle$ "skips" the next barred nodes and calls q' at the i-th child of the next σ-node. For $n \in \mathbb{N}$, we define mon^n_Σ ("*mon*adic insertion") as the linear tree homomorphism which, for each $\sigma^{(k)} \in \Sigma$, has the rule $\langle q, \sigma(x_1, \ldots, x_k)\rangle \to \bar\sigma_1(\bar\sigma_2(\cdots\bar\sigma_n(\sigma(\langle q, x_1\rangle, \ldots, \langle q, x_k\rangle))\cdots))$.

Lemma 3. *1-MM $\subseteq$ LHOM;MT. Totality and determinism are preserved. If the mr-mtt has n states of rank $\le k$, r rules, $\le l$ let-bindings per rule, and m input symbols of rank $\le b$, then the mtt has at most $n+r+nb$ states, $k+l$ parameters, and $(r+nb)(l+1)$ (or $(r+nb)(m+l+1)$ in the case of totality) rules.*

Proof. Let the mr-mtt be $(Q, \Sigma, \Delta, q_0, R, D)$. The state set of the simulating mtt is $Q \cup \{p_r^{(k+m)} \mid r \in R, m = \text{ number of let-bindings in } r, k = \text{ rank of the state of } r\} \cup (Q \times \{1, \ldots, b\})$. Suppose the mr-mtt has a rule $r \in R$ of the form (where $q^{(k)} \in Q$) $\langle q, \sigma(\vec x)\rangle(y_1, \ldots, y_k) \to$ let $z_1 = \langle q_1, x_{i_1}\rangle(\vec u_1)$ in $\ldots$ let $z_m = \langle q_m, x_{i_m}\rangle(\vec u_m)$ in u. Let ζ be the substitution $[z_1/y_{k+1}, \ldots, z_m/y_{k+m}]$. The simulating mtt has the following rules each corresponding to one let-binding:

$$\begin{aligned}\langle q, \bar\sigma_1(x_1)\rangle(y_1, \ldots, y_k) &\to \langle p_r, x\rangle(y_1, \ldots, y_k, \langle\langle q_1, i_1\rangle, x_1\rangle(\vec u_1), \alpha, \ldots, \alpha)\\ \langle p_r, \bar\sigma_2(x_1)\rangle(y_1, \ldots, y_{k+m}) &\to \langle p_r, x\rangle(y_1, \ldots, y_{k+1}, \langle\langle q_2, i_2\rangle, x_1\rangle(\vec u_2\zeta), \alpha, \ldots, \alpha)\\ &\vdots\\ \langle p_r, \bar\sigma_m(x_1)\rangle(y_1, \ldots, y_{k+m}) &\to \langle p_r, x\rangle(y_1, \ldots, y_{k+m-1}, \langle\langle q_m, i_m\rangle, x_1\rangle(\vec u_m\zeta))\\ \langle p_r, \bar\sigma_{m+1}(x_1)\rangle(y_1, \ldots, y_{k+m}) &\to \langle p_r, x\rangle(y_1, \ldots, y_{k+m})\\ &\vdots\\ \langle p_r, \bar\sigma_l(x_1)\rangle(y_1, \ldots, y_{k+m}) &\to \langle p_r, x\rangle(y_1, \ldots, y_{k+m})\\ \langle p_r, \sigma(\vec x)\rangle(y_1, \ldots, y_{k+m}) &\to u\zeta\end{aligned}$$

where α is an arbitrary rank-0 output symbol. These dummy arguments are passed just for supplying exactly m arguments and will never appear in output trees.

The states $\langle q, j\rangle \in Q \times \{1, \ldots, b\}$ are used to remember the correct child number j where to apply q. The rules for $\langle q, j\rangle$ are:

$$\langle\langle q, j\rangle, \bar{\sigma}_i(x_1)\rangle(\vec{y}) \to \langle\langle q, j\rangle, x_1\rangle(\vec{y}) \text{ for each } \sigma \in \Sigma \text{ and } 1 \le i \le l$$
$$\langle\langle q, j\rangle, \sigma(\vec{x})\rangle(\vec{y}) \to \langle q, x_j\rangle(\vec{y}) \text{ for each } \sigma \in \Sigma \text{ of rank } \ge j.$$

It should be clear that this mtt preceded by mon^l_Σ realizes the same translation as the original mr-mtt. Let us take a look at totality and determinism. The original state q remains total (or deterministic, respectively) for a symbol $\bar{\sigma}_1$ if and only if it is total (deterministic) for σ in the original rule set. Newly added states p_r are deterministic if the original state q was. Newly added states $\langle q, j\rangle$ are deterministic. For the remaining undefined part (q-rules for $\bar{\sigma}_2, \ldots, \bar{\sigma}_l$ and σ and $\langle q, j\rangle$-rules for σ with rank $< j$), we add dummy rules to regain totality if the original mr-mtt was total. □

By combining Lemmas 2 and 3, we obtain the main result of this section.

Lemma 4. *MM* $\subseteq$ *LHOM*;*MT*;*LD*$_t$*T*.

Since, by Theorem 7.6 of [1], *DMT* and *D*$_t$*MT* are both closed under left- and- right-composition with *D*$_t$*T*, we obtain the following corollary.

Corollary 5. *DMM* = *DMT* *and* *D*$_t$*MM* = *D*$_t$*MT*.

The right part of Corollary 5 follows also from the result of [7], that total deterministic tree generating top-down tree-to-graph transducers (trgen-tg) are equivalent to *D*$_t$*MT*, because as mentioned in the Introduction, mr-mtts are a special case of trgen-tgs.

4 Simulation of MTTs by Multi-Return MTTs

We now show that *MM* is closed under right-composition with *D*$_t$*T*. The idea is to construct the simulating mr-mtt by *running* the tt on the right-hand side of each rule of the original mr-mtt. Let $\{p_1, \ldots, p_n\}$ be the set of states of the tt. We construct the rules so that if a state q returns a tuple $(t_1, \ldots, t_d)$, then the corresponding state q' of the simulating mr-mtt returns $(\langle p_1, t_1\rangle\downarrow, \ldots, \langle p_1, t_d\rangle\downarrow, \ldots, \langle p_n, t_1\rangle\downarrow, \ldots, \langle p_n, t_d\rangle\downarrow)$.

Lemma 6. *MM*;*D*$_t$*T* $\subseteq$ *MM. Totality and determinism are preserved. The number of parameters and the dimension of the resulting mr-mtt are n times larger the original ones, where n is the number of states of the tt. The number of states increases by 1, and the number of rules is at most twice as that of the original one.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, R_M, D)$ be an mr-mtt and $N = (P, \Delta, \Gamma, p_1, R_N)$ be a D_tT with $P = \{p_1, \ldots, p_n\}$. We define the mr-mtt $M' = (Q', \Sigma, \Gamma, \hat{q}, R', D')$, where $Q' = \{q'^{(kn)} \mid q^{(k)} \in Q\} \cup \{\hat{q}^{(0)}\}$, $D'(q') = n \cdot D(q)$, $D'(\hat{q}) = 1$, and

$$R' = \{\langle q', \sigma(\vec{x})\rangle(y_1, \ldots, y_{kn}) \to \mathit{runN}(r) \mid \langle q, \sigma(\vec{x})\rangle(y_1, \ldots, y_k) \to r \in R_M\}$$
$$\cup\, \{\langle \hat{q}, \sigma(\vec{x})\rangle \to \mathit{runN}_0(r) \mid \langle q_0, \sigma(\vec{x})\rangle \to r \in R_M\}$$

where *runN* and *runN*$_0$ are defined inductively as follows. Recall from the Definitions the "tree view" of the right-hand side of our mr-mtt (with *tup* and *let* node).

$$\begin{aligned}
\mathit{runN}_0(\mathrm{tup}_1(u_1)) &= \mathrm{tup}_1(\langle p_1, u_1\rangle{\downarrow_{N'}}) \\
\mathit{runN}_0(\mathrm{let}_e(z_{s+1},\ldots,z_{s+e},\langle q,x\rangle(u_1,\ldots,u_k),\kappa) &= \mathrm{let}_{en}(\mathbf{z}_{s,e},\langle q,x\rangle(\mathbf{pu}_k),\mathit{runN}_0(\kappa)) \\
\mathit{runN}(\mathrm{tup}_e(u_1,\ldots,u_e)) &= \mathrm{tup}_{en}(\mathbf{pu}_e) \\
\mathit{runN}(\mathrm{let}_e(z_{s+1},\ldots,z_{s+e},\langle q,x\rangle(u_1,\ldots,u_k),\kappa) &= \mathrm{let}_{en}(\mathbf{z}_{s,e},\langle q,x\rangle(\mathbf{pu}_k),\mathit{runN}(\kappa))
\end{aligned}$$

where $\mathbf{z}_{s,e} = z_{sn+1},\ldots,z_{sn+en}$, $\mathbf{pu}_m = \langle p_1,u_1\rangle{\downarrow_{N'}},\ldots,\langle p_1,u_m\rangle{\downarrow_{N'}},\ldots,\langle p_n,u_1\rangle{\downarrow_{N'}},\ldots,\langle p_n,u_m\rangle{\downarrow_{N'}}$ and N' is N extended by the rules $\langle p_j,y_i\rangle \to y_{(i-1)n+j}$ and $\langle p_j,z_i\rangle \to z_{(i-1)n+j}$ for $1 \le i \le \mu$, $1 \le j \le n$, where μ is the maximum of the number of parameters and the number of let-bindings appearing in R_M. Then, by induction on the number of derivation steps, we can show that $\langle p_j,\langle q,t\rangle(\vec{u}){\downarrow_M}\rangle{\downarrow_{N'}}$ is equal to the corresponding subtuples of $\langle q',t\rangle(\mathbf{pu}_k){\downarrow_{M'}}$, which proves the lemma. □

Note that the proof of Lemma 6 relies on the *totality* of N. It simulates all p_i-translations, some of which may not contribute to the final output. If N is not total, this try-and-discard strategy does not work. Undefined calls that are to be discarded will stop the whole translation, since we are considering call-by-value evaluation. The proof relies also on the *determinism* of N. If p_j is nondeterministic, multiple calls of $p_j(y_i)$ may generate different outputs and thus replacing them by the same single variable $y_{(i-1)n+j}$ yields incorrect results.

Next, we investigate the case of left-composition. The idea is, again, to simulate the composition $D_tT;MT$ by constructing an mr-mtt by running the mtt on the rules of tt. Note that we crucially use let-bindings here for simulating parameter copying of the original mtt. Suppose we have a tt rule $\langle q,\mathtt{e}(x_1)\rangle \to \mathtt{a}(\mathtt{b},\langle q,x_1\rangle)$ and mtt rules:

$$\begin{aligned}
\langle p,\mathtt{a}(x_1,x_2)\rangle(y_1) &\to \langle p,x_1\rangle(\langle p,x_2\rangle(y_1)) \\
\langle p,\mathtt{b}\rangle(y_1) &\to \mathtt{d}(y_1,y_1).
\end{aligned}$$

Using a let-binding, we construct a rule of the simulating transducer $\langle\langle p,q\rangle,\mathtt{e}(x_1)\rangle(y_1) \to$ let $z = \langle\langle p,q\rangle,x_1\rangle(y_1)$ in $\mathtt{d}(z,z)$ which correctly preserves the original semantics that the left and right child of the `d` node are equal. Note that without let-bindings, we cannot avoid duplicating a state call; at best we will have the rule $\langle\langle p,q\rangle,\mathtt{e}(x_1)\rangle(y_1) \to \mathtt{d}(\langle\langle p,q\rangle,x_1\rangle(y_1),\langle\langle p,q\rangle,x_1\rangle(y_1))$, which is incorrect because the duplicated state calls may nondeterministically yield different outputs, which is not originally intended.

Lemma 7. *$D_tT;MT \subseteq$ 1-MM. Totality and determinism are preserved. The number of states is n times larger, where n is the number of states of the D_tT. The number of parameters remains the same. The number of rules may be double exponential with respect to the depth of right-hand sides of the D_tT.*

Proof. Let $M_1 = (Q,\Sigma,\Gamma,q_0,R_1)$ be a D_tT and $M_2 = (P,\Gamma,\Delta,p_0,R_2)$ an mtt. Define $M_\times = (P\times Q,\Sigma,\Delta,\langle p_0,q_0\rangle,R,D)$ with $R = \{\langle\langle p,q\rangle,\sigma(\vec{x})\rangle(\vec{y}) \to \kappa \mid \kappa \in f_z(\langle p,r\rangle(\vec{y}),z), r$ is the right-hand side of the unique $\langle q,\sigma\rangle$-rule of $R_1\}$. Intuitively, a state $\langle p,q\rangle$ denotes the translation by q followed by p. The relation f_z is very similar to the derivation relation of M_2 (thus, $f_z(\langle p,r\rangle(\vec{y}),z)$ should be intuitively read as

$\langle p, r\rangle(\vec{y})\downarrow_{M_2}$). However, to "factor out" let-bindings for avoiding incorrect duplication of state calls, we define it slightly differently. For the sake of simplicity, we define f_z as a nondeterministic function as follows:

$$
\begin{aligned}
f_z(y, u) &= u[z/y] && y \in Y \cup Z \\
f_z(\delta(t_1, \ldots, t_k), u) &= f_{z_1}(t_1, \ldots f_{z_k}(t_k, u[z/\delta(z_1, \ldots, z_k)]) \cdots) && \delta \in \Delta \\
f_z(\langle p, \langle q, x_i\rangle\rangle(t_1, \ldots, t_k), u) &= f_{z_1}(t_1, \ldots f_{z_k}(t_k, \\
&\qquad \text{let } z = \langle\langle p, q\rangle, x_i\rangle(z_1, \ldots, z_k) \text{ in } u) \ldots) \\
f_z(\langle p, \gamma(\vec{s})\rangle(t_1, \ldots, t_k), u) &= f_{z_1}(t_1, \ldots f_{z_k}(t_k, f_z(\kappa[\vec{x}/\vec{s}, \vec{y}/\vec{z}], u) \cdots) \\
&\qquad \text{for every right-hand side } \kappa \text{ of any } \langle p, \gamma\rangle\text{-rule, } \gamma \in \Gamma.
\end{aligned}
$$

The last argument u of f_z denotes a context where the translated right-hand side of the rule should be placed. By induction on the structure of the input tree s, we can prove $\langle p, \langle q, s\rangle\downarrow_{M_1}\rangle(\vec{t})\downarrow_{M_2} = \langle\langle p, q\rangle, s\rangle(\vec{t})\downarrow_{M_\times}$ for $p \in P$, $q^{(k)} \in Q$, $\vec{t} \in T_\Delta^k$, and $s \in T_\Sigma$, which proves the lemma. □

We can now generalize the lemma in two directions: the second translation from *MT* to *MM*, and the first translation from total to partial.

Lemma 8. *$D_tT\,;MM \subseteq MM$. Totality and determinism are preserved.*

Proof. By Lemma 4, $D_tT\,;MM \subseteq D_tT\,;LHOM\,;MT\,;LD_tT$. By Lemma 6.9 of [6], which says that D_tT is closed under composition, the latter is in $D_tT\,;MT\,;LD_tT$. By Lemma 7 this is included in $MM\,;LD_tT$, which is in *MM* by Lemma 6. □

Lemma 9. *$DT\,;MM \subseteq MM$.*

Proof. We have $DT \subseteq$ *DT-FTA*$\,;D_tT$ (Lemma 5.22 of [1]) where *DT-FTA* is the class of partial identity translations recognized by deterministic top-down tree automata, Lemma 8, and *DT-FTA*$\,;MM \subseteq MM$ (can be proved by the same construction as for Lemma 5.21 of [1]; for every rule of the initial state, we add one let-binding that carries out the run of the automaton). These three lemmas prove $DT\,;MM \subseteq MM$. □

Using the lemmas proved up to here, we obtain the two main theorems: the characterization of mr-mtts in terms of mtts and its closure properties.

Theorem 10. *$MM = LHOM\,;MT\,;LD_tT$. Determinism and totality are preserved.*

Theorem 11. *$DT\,;MM \subseteq MM$ and $MM\,;D_tT \subseteq MM$.*

5 Expressiveness

First we show that mr-mtts of dimension 1 are already more powerful than normal mtts even without tuple-returning capability. On page 123 of [1], a counterexample to show $MT \subsetneq LHOM\,;MT$ is given without proof. (The difficult part of their counterexample to be realized in *MT* is the generation of two *identical* pairs of a nondeterministic relabeling of the input, which is similar to our *twist* translation that generates *mutually reverse* pair of nondeterministic relabelings.) By this example and Lemma 7, we have the following proposition, which shows that binding intermediate trees by let-expressions itself adds expressiveness.

Proposition 12. *MT* $\subsetneq$ *1-MM.*

Moreover, mr-mtts that return pairs of trees are strictly more powerful than single return ones. Here we only give a sketch of the proof. For more detail, see [10] (which proves the unrealizability in *MT*, but it also works for *1-MM* and call-by-name mtts).

Theorem 13. *1-MM* $\subsetneq$ *2-MM.*

Proof (Sketch). A translation realized in *2-MM*, namely, the *twist* translation of the Introduction is shown not to be realizable by any mr-mtt with dimension 1. The proof is by contradiction. Note that the number of outputs of $twist$ is exponential with respect to the size of the input, that is, $|twist(\mathtt{s}^n\mathtt{z})| = 2^n$. We first assume an mr-mtt M of dimension 1 to realize $twist$, and then by giving two normal forms of sentential forms (a weak normal form that contains no let-variables under output symbols, and a strong normal form that is the weak normal form with at most one let-binding), we can show that $|\tau(M)(\mathtt{s}^n\mathtt{z})| = O(n^2)$, which is a contradiction. □

Note that the composition $MT\,;D_tT$ can realize $twist$. We can construct an mtt (both in call-by-value and call-by-name semantics) that nondeterministically translates the input $\mathtt{s}^n\mathtt{z}$ into all monadic trees of the form $(\mathtt{a}|\mathtt{b})^n(\mathtt{A}|\mathtt{B})^n\mathtt{E}$ such that the lower-part is the reverse of the upper-part. Then we split such monadic trees to lower- and upper-parts by a D_tT transducer, so that the composition of these two translations realizes $twist$. Thus, together with the proof of Theorem 13, we have the following theorem.

Theorem 14. *$MT\,;D_tT \not\subseteq MT$ and $MT_{OI}\,;D_tT \not\subseteq MT_{OI}$, where MT_{OI} denotes the class of translations realized by call-by-name mtts.*

Acknowledgements We like to thank Joost Engelfriet for his comment that mr-mtts can be simulated by simpler transducers, which led us to the results in Sections 3 and 4. This work was partly supported by Japan Society for the Promotion of Science.

References

1. Engelfriet, J., Vogler, H.: Macro tree transducers. JCSS **31** (1985) 71–146
2. Fülöp, Z., Vogler, H.: Syntax-Directed Semantics: Formal Models Based on Tree Transducers. Springer-Verlag (1998)
3. Maneth, S., Berlea, A., Perst, T., Seidl, H.: XML type checking with macro tree transducers. In: Principles of Database Systems (PODS). (2005) 283–294
4. Perst, T., Seidl, H.: Macro forest transducers. IPL **89** (2004) 141–149
5. Rounds, W.C.: Mappings and grammars on trees. MST **4** (1970) 257–287
6. Thatcher, J.W.: Generalized sequential machine maps. JCSS **4** (1970) 339–367
7. Engelfriet, J., Vogler, H.: The translation power of top-down tree-to-graph transducers. JCSS **49** (1994) 258–305
8. Mezei, J., Wright, J.B.: Algebraic automata and context-free sets. Inf. Contr. **11** (1967) 3–29
9. Engelfriet, J., Maneth, S.: A comparison of pebble tree transducers with macro tree transducers. Acta Informatica **39** (2003) 613–698
10. Inaba, K., Hosoya, H.: Multi-return macro tree transducers. In: PLAN-X. (2008)