Multi-Return Macro Tree Transducers

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

Haruo Hosoya
The University of Tokyo
hahosoya@arbre.is.s.u-tokyo.ac.jp

Abstract

Macro tree transducers are a simple yet expressive formal model for XML transformation languages. The power of this model comes from its accumulating parameters, which allow to carry around several output tree fragments in addition to the input tree. However, while each procedure is enabled by this facility to propagate intermediate results in a top-down direction, it still cannot do it in a bottom-up direction since it is restricted to return only a single tree and such tree cannot be decomposed once created. In this paper, we introduce *multi-return* macro tree transducers as a mild extension of macro tree transducers with the capability of each procedure to return more than one tree at the same time, thus attaining symmetry between top-down and bottom-up propagation of information. We illustrate the usefulness of this capability for writing practically meaningful transformations. Our main technical contributions consists of two formal comparisons of the expressivenesses of macro tree transducers and its multi-return extension: (1) in the deterministic case, the expressive powers of these two coincide (2) in the nondeterministic case (with the call-by-value evaluation strategy) multi-return macro tree transducers are *strictly* more expressive.

1. Introduction

The emergence of XML has invoked an active body of research of the formal study of tree transformation models. Among these, the tree transducer family has drawn much attention since it is expressive enough to represent a wide range of practical transformations, yet simple enough to ensure various desirable properties such as exact typechecking, evaluation complexity bounds, composability, streaming, and so on [2, 4, 7, 12, 15, 3, 14, 10, 13, 16, 11, 6]. A particularly important model is the *macro tree transducer* (MTT) [4, 7]. MTTs are a finite-state machine model to transform a tree where each state (or a procedure in programming language terminology) is allowed to pass several extra arguments (called *accumulating parameters*) in addition to the input tree. This facility to carry around multiple intermediate results is proved to be useful for various purposes and indeed increase the expressive power compare to top-down tree transducers lacking this facility [4]. However, MTTs have the asymmetry that, while each state can propagate multiple trees in a top-down manner, it cannot do it in a bottom-up manner since it is still restricted to return only a single tree and such tree cannot be decomposed once created.

Proceedings of the 6th ACM SIGPLAN Workshop on Programming Language Technologies for XML (PLAN-X 2008).
January 9, 2008, San Francisco, California.

This paper introduces a mild extension of MTTs called *multi-return macro tree transducers* (mr-MTT) where each state is endowed the capability of returning more than one tree. We will see below that this addition is indeed useful in writing concise programs for practically meaningful transformations that need a longer and more cumbersome description in conventional MTTs. However, beyond this informal argument, one would naturally ask: how are the expressivenesses different between MTTs and mr-MTTs in principle? Our main contributions in this paper give affirmative answers to this question in two cases:

1. In the deterministic case, these two are equally expressive.
2. In the nondeterministic case with the call-by-value semantics, mr-MTTs are strictly more expressive than MTTs.

Multi-return macro tree transducers Let us informally introduce MTTs and mr-MTTs to illustrate how the multi-return capability is useful. First, MTTs can be seen as a simple functional language where the procedures can be defined only by induction on the first argument. The following shows an MTT expressing a transformation that takes a tree consisting of binary (`a` and `b`) and leaf (`e`) nodes and returns an `r` node holding two lists. The first list consists of `item` nodes each of which contains the first subtree of an `a` node appearing in the input tree; the second list is similar but collects the first subtrees of the `b` nodes.

$$
\begin{aligned}
\langle \mathsf{main}, x\rangle() &\to \mathtt{r}(\langle \mathsf{geta}, x\rangle(\mathtt{e}), \langle \mathsf{getb}, x\rangle(\mathtt{e}))\\
\langle \mathsf{geta}, \mathtt{a}(x_1, x_2)\rangle(y) &\to \mathtt{item}(\langle \mathsf{copy}, x_1\rangle(),\\
&\qquad \langle \mathsf{geta}, x_1\rangle(\langle \mathsf{geta}, x_2\rangle(y)))\\
\langle \mathsf{geta}, \mathtt{b}(x_1, x_2)\rangle(y) &\to \langle \mathsf{geta}, x_1\rangle(\langle \mathsf{geta}, x_2\rangle(y))\\
\langle \mathsf{geta}, \mathtt{e}\rangle(y) &\to y\\
\langle \mathsf{getb}, \mathtt{a}(x_1, x_2)\rangle(y) &\to \langle \mathsf{getb}, x_1\rangle(\langle \mathsf{getb}, x_2\rangle(y))\\
\langle \mathsf{getb}, \mathtt{b}(x_1, x_2)\rangle(y) &\to \mathtt{item}(\langle \mathsf{copy}, x_1\rangle(),\\
&\qquad \langle \mathsf{getb}, x_1\rangle(\langle \mathsf{getb}, x_2\rangle(y)))\\
\langle \mathsf{getb}, \mathtt{e}\rangle(y) &\to y\\
\langle \mathsf{copy}, \mathtt{a}(x_1, x_2)\rangle() &\to \mathtt{a}(\langle \mathsf{copy}, x_1\rangle(), \langle \mathsf{copy}, x_2\rangle())\\
\langle \mathsf{copy}, \mathtt{b}(x_1, x_2)\rangle() &\to \mathtt{b}(\langle \mathsf{copy}, x_1\rangle(), \langle \mathsf{copy}, x_2\rangle())\\
\langle \mathsf{copy}, \mathtt{e}\rangle() &\to \mathtt{e}
\end{aligned}
$$

We use the form $\langle f, t\rangle(t_1, \ldots, t_n)$ for procedure definitions and calls where t is the first argument (the recursion argument, i.e., the input tree) and $t_1, \ldots, t_n$ the remainder (i.e., the accumulating parameters). We adopt this notation to emphasize the special role of the first argument. The procedure `geta` is intended to collect all the first subtrees of `a` nodes in the input tree. This procedure takes an extra parameter for holding the first subtrees of `a` nodes that have been collected so far. Each rule works as follows. If the input tree is an `a` node, we recursively call `geta` on the subtrees x_2 and x_1 to yield the result from these subtrees combined with the collections of the passed parameter y; we then augment the collection, in an

`item` node, with the first subtree x_1 (more precisely, its copy via the `copy` procedure defined in the bottom; MTTs have no way to directly return the input tree) to yield an answer to the original call. The second rule for `geta` on a `b` node is similar except that we do not construct an `item` node. In the case that the input is a leaf, we directly return the passed accumulating parameter. The procedure `getb` is analogous.

The above MTT traverses the input tree twice, once for collecting the first subtrees of `a`'s and another for collecting those of `b`'s. It is not possible to combine the traversals to a single one since, as stated above, procedures in MTTs cannot return multiple separate trees simultaneously. We may attempt to combine the two lists of `a`'s and `b`'s in a temporary node and return it, as in the usual practice of functional programming. But then, we cannot add later more `a`'s or `b`'s to the appropriate list since we cannot decompose the temporary node back to the two list (otherwise the procedures would not be defined by induction on the first argument). This restriction is rather awkward. For example, if the two traversals have a common complex condition, then separating the procedures for these traversals may require us to duplicate the same code for the condition.

It is thus natural to extend MTTs with the capability to return multiple return values. More precisely, our extension is to allow a procedure to return a tuple of trees and the elements of such a tuple to be bound to variables by a let construct (immediately after the return). The following expresses the same transformation as above by an mr-MTT (eliding the `copy` procedure for brevity).

$$
\begin{aligned}
\langle \mathsf{main}, x\rangle() &\to \langle \mathsf{get}, x\rangle(\mathtt{e}, \mathtt{e})\\
\langle \mathsf{get}, \mathtt{a}(x_1, x_2)\rangle(y_1, y_2) &\to \text{let } (z_1, z_2) = \langle \mathsf{get}, x_2\rangle(y_1, y_2) \text{ in}\\
&\qquad \text{let } (z_3, z_4) = \langle \mathsf{get}, x_1\rangle(z_1, z_2) \text{ in}\\
&\qquad\quad (\mathtt{item}(\langle \mathsf{copy}, x_1\rangle(), z_3), z_4)\\
\langle \mathsf{get}, \mathtt{b}(x_1, x_2)\rangle(y_1, y_2) &\to \text{let } (z_1, z_2) = \langle \mathsf{get}, x_2\rangle(y_1, y_2) \text{ in}\\
&\qquad \text{let } (z_3, z_4) = \langle \mathsf{get}, x_1\rangle(z_1, z_2) \text{ in}\\
&\qquad\quad (z_3, \mathtt{item}(\langle \mathsf{copy}, x_1\rangle(), z_4))\\
\langle \mathsf{get}, \mathtt{e}\rangle(y_1, y_2) &\to (y_1, y_2)
\end{aligned}
$$

This time, we make only a single traversal on the input by carrying around two lists at the same time, using two accumulating parameters and two return values. The rules work in the natural way that, if the input tree is an `a` node, then we augment the first list with an `item` node, and so on.

Expressiveness A natural question that arises here is: what is the exact relationship between MTTs and mr-MTTs in terms of expressiveness? In the previous section, we have shown an MTT and an mr-MTT that perform exactly the same transformation. These examples would suggest that MTTs and mr-MTTs might have equal expressiveness. As stated already, it is indeed true in the deterministic case. The formal proof will be shown in Section 4.1, but the intuition is that each procedure in an mr-MTT returning a k-tuple of trees can be split to k procedures each returning a single tree (generalizing our example above).

We did not stop our investigation at the deterministic case, but further proceeded to the nondeterministic case. One might ask why we should care about the nondeterministic case from the first place, given the situation where many practical tree transformations seem to be deterministic. Nondeterminism is in fact quite useful or even required in some areas, namely,

- to encode transformations or queries that are nondeterministic by design; for example, the XDuce language for XML processing has regular expression pattern matching whose semantics on choices is nondeterministic [9];
- to approximately represent transformations in some higher-level language (e.g., XSLT [1]) that allow complicated forms in conditional expressions; such approximation may be useful for performing typechecking, cf. [10, 11, 6];
- to encode queries that return a *set* of trees.

As already mentioned, in the nondeterministic case with call-by-value semantics, mr-MTTs are strictly more expressive than MTTs. Intuitively, consider an mr-MTT that is similar to the one above, but has nondeterministic choices at each node and, depending on which choice to take, we put a different thing in the lists. We cannot simply split the procedure to two since we would not necessarily take the same choice at a given node in the two separate traversals. In the formal proof given in Section 4.2, we will exhibit a counterexample inspired by this observation and show that this can be expressed by an mr-MTT but not an MTT. The proof of the inexpressibility is a rather long and involved one, which is a natural consequence of the current situation where there is no standard proof technique for tree transducers analogous to Pumping Lemma in automata theory [8] (except for the height property, i.e., the maximal increase of tree heights from input to output, cf. [4], which fails in our case since MTTs and mr-MTTs obviously have the same height property).

Structure The rest of the paper is organized as follows. We first define MTTs, giving their known properties, in Section 2, and then define mr-MTTs in Section 3. In Section 4, we formally compare the expressivenesses of MTTs and mr-MTTs in the deterministic and the nondeterministic cases. Section 5 touches upon possible future directions.

2. Preliminaries

This section gives the definition of (single-return) MTTs and their known properties.

A set Σ with a mapping $rank : \Sigma \to \mathbb{N}$ is called a *ranked set*. We sometimes annotate each element $\sigma \in \Sigma$ with its rank by a superscript like σ^k to indicate that $rank(\sigma) = k$. The *product* of a ranked set A and a set B is a ranked set $A \times B = \{(a, b)^k \mid a^k \in A, b \in B\}$. Note that the rank of first component is inherited. The set T_Σ of *trees* t over a ranked set Σ is defined by the following BNF:

$$t ::= \sigma^k(t_1, \ldots, t_k) \quad (\sigma^k \in \Sigma)$$

In the rest of this paper, we sometimes omit parentheses for rank-0 and rank-1 symbols and write them like strings. For example, we write `abcd` instead of $\mathtt{a}(\mathtt{b}(\mathtt{c}(\mathtt{d}())))$.

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 written $t[\vec{x}/\vec{t}]$ 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.

Definition 1. Assume a set $X = \{x_1, x_2, \ldots\}$ of input variables and a set $Y = \{y_1, y_2, \ldots\}$ of (accumulating) parameters both of rank-0. A macro tree transducer (MTT) is a tuple $(Q, q_0, \Sigma, \Delta, R)$, where Q, Σ, and Δ are finite ranked sets. We call Q a set of states, $q_0 \in Q$ an initial state of rank 0, Σ an input alphabet, Δ an output alphabet, and R a finite set of translation rules of the form

$$\langle q^k, \sigma^m(x_1, \ldots, x_m)\rangle(y_1, \ldots, y_k) \to r$$

where a right hand side r is a tree from $T_{\Delta \cup (Q \times X) \cup Y}$, that is, either a node construction $\sigma^k(r_1, \ldots, r_k)$, a procedure call $\langle q^k, x\rangle(r_1, \ldots, r_k)$, or an accumulating parameter y where $r_1, \ldots, r_k$ themselves are from $T_{\Delta \cup (Q \times X) \cup Y}$. We call the left hand side of a rule a *rule head*. We require that any MTT is well-formed,

that is, all free variables in the right hand side of each rule are bound by the rule head. Given an MTT $(Q, q_0, \Sigma, \Delta, R)$, a state $q \in Q$ is *deterministic* if there exists at most one rule of the form $\langle q, \sigma(\ldots)\rangle(\ldots) \to r$ in R for each $\sigma \in \Sigma$ and the whole MTT is deterministic if all its states are deterministic.

Below, we will define the semantics of MTTs by a rewriting relation over intermediate states called sentential forms. Intuitively, a sentential form is an output tree that contains procedure call forms $\langle q^k, \sigma^m(t_1, \ldots, t_m)\rangle(s_1, \ldots, s_k)$, i.e., application of q^k to the input tree $\sigma^m(t_1, \ldots, t_m) \in T_\Sigma$ with accumulating parameters $s_1, \ldots, s_k \in T_\Delta$. In each rewriting step, we find a context that holds such a procedure call and expand the call to the right hand side of a matching rule with appropriate substitution.

Definition 2. Let $M = (Q, q_0, \Sigma, \Delta, R)$ be an MTT. We define a set $\Lambda_M = \Delta \cup (Q \times T_\Sigma)$ and call trees in T_{Λ_M} *sentential forms* of M. (We omit the subscript M when it is clear.)

Definition 3. For a ranked set Σ and a rank-0 symbol $\Box \notin \Sigma$, a tree $\mathcal{C} \in T_{\Sigma \cup \{\Box\}}$ that contains exactly one occurrence of $\Box$ is called *one-hole Σ-context*. We write $\mathcal{C}[s]$ as a shorthand for $\mathcal{C}[\Box/s]$.

We give a *call-by-value* (also known as *IO-mode*) semantics of MTTs. The definition is done by interpreting the translation rules of MTTs as rewriting rules over sentential forms.

Definition 4. Let $M = (Q, q_0, \Sigma, \Delta, R)$ be an MTT, and $u, u' \in T_\Lambda$. The binary "small-step evaluation" relation $u \Rightarrow u'$ is defined when there is a rule in R of the form

$$\langle q^k, \sigma^m(x_1, \ldots, x_m)\rangle(y_1, \ldots, y_k) \to r$$

and there are a one-hole Λ-context $\mathcal{C}$, input trees $s_1, \ldots, s_m \in T_\Sigma$, and output trees $t_1, \ldots, t_m \in T_\Delta$, such that

$$u = \mathcal{C}[\langle q, \sigma(t_1, \ldots, t_m)\rangle(s_1, \ldots, s_k)]$$

and

$$u' = \mathcal{C}[r[x_1/t_1, \ldots, x_m/t_m, y_1/s_1, \ldots, y_k/s_k]].$$

That is, when u has a context in which a procedure call appears, this call can be expanded to the right hand side of a matching rule with the subtrees $t_1, \ldots, t_m$ substituted for the input variables and the arguments $s_1, \ldots, s_k$ for the accumulating parameters. As usual, the derivation relation $\Rightarrow^*$ is defined as a reflexive, transitive closure of $\Rightarrow$. We say that v is derivable from u when $u \Rightarrow^* v$ and define $u\downarrow = \{s \in T_\Delta \mid u \Rightarrow^* s\}$. Two sentential forms u and v are equivalent, written $u \equiv v$, when $u\downarrow = v\downarrow$.

Definition 5. Let $M = (Q, q_0, \Sigma, \Delta, R)$ be an MTT, the translation $\tau(M)$ of M is defined as $\{(t, s) \in T_\Sigma \times T_\Delta \mid s \in \langle q_0, t\rangle()\downarrow\}$. When a relation f is equal to $\tau(M)$, we say that f is *realized* by M.

In the subsequent proofs, it is actually convenient to extend the above-defined derivation relation from sentential forms to contexts. Furthermore, it is also convenient to generalize contexts so as to contain multiple occurrences of several kinds of holes.

Definition 6. For a ranked set Σ and rank-0 symbols $\Box_1, \ldots, \Box_n \notin \Sigma$, a tree $\mathcal{C} \in T_{\Sigma \cup \{\Box_1, \ldots, \Box_n\}}$ is called *Σ-n-context*. We write $\mathcal{C}[s_1, \ldots, s_n]$ as a shorthand for $\mathcal{C}[\Box_1/s_1, \ldots, \Box_n/s_n]$.

Thus, a one-hole Σ-context is a Σ-1-context $\mathcal{C}$ that contains exactly one occurrence of $\Box_1$.

Definition 7. Let $M = (Q, q_0, \Sigma, \Delta, R)$ be an MTT and rank-0 symbols $\Box_1, \ldots, \Box_n \notin \Lambda$. Then, for $\mathcal{C}, \mathcal{C}' \in T_{\Lambda \cup \{\Box_1, \ldots, \Box_n\}}$, we write $\mathcal{C} \Rightarrow \mathcal{C}'$ when the relation holds in the extended MTT $M' = (Q, q_0, \Sigma, \Delta \cup \{\Box_1, \ldots, \Box_n\}, R)$. The relations $\mathcal{C} \Rightarrow^* \mathcal{C}'$ and $\mathcal{C}\downarrow$ are extended accordingly.

By using the derivation relation over contexts, the following known propositions hold. Intuitively, when a sentential form can be split to a context and subtrees in T_Δ, performing derivation from the whole sentential form is equivalent to performing first derivation from the context and then substitution of the subtrees. We will repeatedly use these propositions in the sequel.

Prop 1 (Lemma 3.19 of [4])**.** *Let $\mathcal{C}$ be a Λ-n-context and $s_1, \ldots, s_n \in T_\Delta$. Then we have $\mathcal{C}[s_1, \ldots, s_n]\downarrow = \{\mathcal{D}[s_1, \ldots, s_n] \mid \mathcal{D} \in \mathcal{C}\downarrow\}$*

Prop 2 (Lemma 5.2 of [4])**.** *Let $\mathcal{C}$ be a Λ-n-context and $r_1, \ldots, r_n \in T_\Lambda$. If (1) $\mathcal{C}$ contains exactly one occurrence for each of $\Box_1, \ldots, \Box_n$, or (2) $\mathcal{C}$ contains at least one occurrence for each of $\Box_1, \ldots, \Box_n$ and the MTT is deterministic, then we have $\mathcal{C}[r_1, \ldots, r_n]\downarrow = \{\mathcal{D}[s_1, \ldots, s_n] \mid \mathcal{D} \in \mathcal{C}\downarrow, s_i \in r_i\downarrow\}$*

3. Multi-return macro tree transducers

In this section, we formally define multi-return macro tree transducers and translations realized by them.

The essential addition to mr-MTTs with respect to MTTs is construction and deconstruction (via let expressions) of tuples of return values. However, for simplicity, we adopt a style similar to an A-Normal Form [5] in λ-calculus so that we disallow nesting of procedure calls, that is, every procedure call must appear immediately as the bound expression of a let expression. In examples, though, we sometimes use nested procedure calls, which should be read as a short-hand for nested let forms. For instance,

$$\begin{aligned}&\text{let } (z_1, z_2) = \langle q_1, x\rangle(\langle q_2, x\rangle()) \text{ in}\\ &\quad(\langle q_3, x\rangle(z_1), \langle q_4, x\rangle(z_2))\end{aligned}$$

is a short-hand for the following:

$$\begin{aligned}&\text{let } z_2' = \langle q_2, x\rangle() \text{ in}\\ &\text{let } (z_1, z_2) = \langle q_1, x\rangle(z_2') \text{ in}\\ &\text{let } z_3' = \langle q_3, x\rangle(z_1) \text{ in}\\ &\text{let } z_4' = \langle q_4, x\rangle(z_2) \text{ in}\\ &\quad(z_3', z_4')\end{aligned}$$

Definition 8. Assume a set Z of rank-0 temporary variables. A multi-return macro tree transducer (mr-MTT) is a tuple of six components $(Q, q_0, \Sigma, \Delta, D, R)$, where Q, Σ, and Δ are ranked sets and D is a mapping of $Q \to \mathbb{N}$. As before, we call Q a set of states, $q_0 \in Q$ an initial state of rank 0, Σ an input alphabet, and Δ an output alphabet. Since each state in an mr-MTT may return multiple values, we represent the number of such return values, called *dimension*, by using D. Then, R is a finite set of translation rules of the form:

$$\langle q^k, \sigma^m(x_1, \ldots, x_m)\rangle(y_1, \ldots, y_k) \to r$$

where $r \in rhs^{D(q)}$. Rules like this form are called q, σ-rules. Here, the set rhs^d of right-hand-sides r in dimension $d \in \mathbb{N}$ is defined by the following

$$\begin{aligned}&r ::= \beta_1 \ldots \beta_n\ (u_1, \ldots, u_d) && (n \geq 0)\\ &\beta ::= \text{let } (z_1, \ldots, z_{D(q)}) = \langle q'^k, x\rangle(u_1, \ldots, u_k) \text{ in}\end{aligned}$$

where $u_1, u_2, \cdots \in T_{\Delta \cup Y \cup Z}$, $z_1, z_2, \cdots \in Z$ and $x \in X$. We require that any mr-MTT is well-formed, that is, all free variables in any subpart of the right hand side of each rule must be bound in the surrounding expression (either in the rule head or in a let binding). An mr-MTT is called *deterministic* if for every pair of $q \in Q$ and $\sigma \in \Sigma$, there exists at most one q, σ-rule in R.

Below, we give a call-by-value semantics to mr-MTTs in a similar way to the case of MTTs in the last section. That is, we first define sentential forms of mr-MTTs and then translations realized

by mr-MTTs in terms of rewriting over sentential forms. Though, we do not define contexts this time since the target of rewriting is always the procedure call that appears in the very first let expression of a given sentential form.

Definition 9. Let $M = (Q, q_0, \Sigma, \Delta, R, D)$ be an mr-MTT. The set K_M of sentential forms κ of M in dimension d are defined by the following

$$\kappa ::= l_1 \dots l_n\ (u_1, \dots, u_d)$$
$$l ::= \text{let}\ (z_1, \dots, z_{D(q)}) = \langle q^k, t\rangle(u_1, \dots, u_k)\ \text{in}$$

where $t \in T_\Sigma$ and $u_1, u_2, \dots \in T_{\Delta \cup Z}$. (The subscript M is omitted if it is clear.) Analogous to the definition of right-hand sides of rules, we require that any sentential form is well-formed. That is, any variable contained in a sentential form must be in scope from a surrounding let bindings.

Definition 10. Let $M = (Q, q_0, \Sigma, \Delta, R, D)$ be an mr-MTT and $t, t' \in K$. The binary relation $t \Rightarrow t'$ holds if t has the form

$$\text{let}\ (z_1, \dots, z_d) = \langle q^k, \sigma^m(t_1, \dots, t_m)\rangle(s_1, \dots, s_k)\ \text{in}\ \kappa$$

and there is a rule in R of the form

$$\langle q^k, \sigma^m(x_1, \dots, x_m)\rangle(y_1, \dots, y_k) \to l_1 \cdots l_n\ (u_1, \dots, u_d)$$

where $d = D(q)$ and t' has the form

$$l'_1 \cdots l'_n \kappa'$$

where

$$l'_i = l_i[x_1/t_1, \dots, x_m/t_m, y_1/s_1, \dots, y_k/s_k] \quad (i = 1, \dots, n)$$
$$u'_j = u_j[y_1/s_1, \dots, y_k/s_k] \quad (j = 1, \dots, d)$$
$$\kappa' = \kappa[z_1/u'_1, \dots, z_d/u'_d].$$

Here, we adopt the standard convention that substitution automatically avoids inappropriate variable capture by silently performing α-conversion. As before, we define the derivation relation $\Rightarrow^*$ as a reflexive, transitive closure of $\Rightarrow$ and say that s is derivable from t when $t \Rightarrow^* s$. We then define $t{\downarrow} = \{s \in T^d_\Delta \mid t \Rightarrow^* s\}$ where d is the dimension of t (note that the final results can be tuples of trees). Technically, the procedure call form $\langle q^k, t\rangle(s_1, \dots, s_k)$ is not a sentential form in an mr-MTT, but for convenience we define $\langle q^k, t\rangle(s_1, \dots, s_k){\downarrow}$ as the set $(\text{let}\ (z_1, \dots, z_{D(q)}) = \langle q^k, t\rangle(s_1, \dots, s_k)\ \text{in}\ (z_1, \dots, z_{D(q)})){\downarrow}$.

Definition 11. Let $M = (Q, q_0, \Sigma, \Delta, R, D)$ be an mr-MTT, A relation $\tau(M)$ is defined as $\{(t, s) \in T_\Sigma \times T^{D(q_0)}_\Delta \mid s \in \langle q_0, t\rangle(){\downarrow}\}$. When a relation f is equal to $\tau(M)$, f is said to be *realized* by M.

4. Expressiveness comparison

When the dimension of its initial state is 1, an mr-MTT realizes a translation of single trees to single trees. In this section, we compare the expressive power of such mr-MTTs to normal MTTs. Since mr-MTTs are obviously at least as expressive as MTTs, we concentrate on whether the converse holds.

In the deterministic case, it is shown that returning multiple trees will not change the expressiveness. In the nondeterministic case, we prove that the multi-return capability does increase the expressive power, by exhibiting an example of transformation that can be written by an mr-MTT but not by an MTT.

4.1 The deterministic case

In order to show that a deterministic MTT can always be constructed from any deterministic mr-MTT, we define two functions: (1) a split function that divides a state q returning d-tuples of trees into d states returning single trees, and (2) a build function that turns an mr-MTT with all states returning single trees into an MTT. Then, we formally prove that these two operations do not change the translation realized.

Before that, however, we first need the technical lemma below. In the definition of derivations for mr-MTTs in the previous section, a let-bound procedure call in a sentential form is replaced by the right hand side of one of its defining rules (with appropriate substitution), and then this right hand side is subject to derivation. The lemma below states that the order between such replacement and such derivation is interchangeable. That is, first fully evaluating the procedure call alone and then replacing the let-bound variables with the resulting tuples will derive exactly the same outputs. Note that this lemma corresponds to Prop 2 (1) for normal MTTs. Although we prove and use the lemma only for deterministic mr-MTTs in this paper, it can be shown that it also holds for nondeterministic case.

Lemma 1. *Let $M = (Q, q_0, \Sigma, \Delta, R, D)$ be a deterministic mr-MTT and κ be a sentential form. Then the following equation holds.*

$$(\textit{let}\ (z_1, \dots, z_d) = \langle q, t\rangle(s_1, \dots, s_k)\ \textit{in}\ \kappa){\downarrow} =$$
$$\bigcup\{\kappa[z_1/u_1, \dots, z_d/u_d]{\downarrow} \mid (u_1, \dots, u_d) \in \langle q, t\rangle(s_1, \dots, s_k){\downarrow}\}$$

Proof. The proof is done by induction on the structure of input trees t. We first consider the case when t is a leaf node $\sigma()$. If there is no rule q, σ-rule in R, then the both sides are empty and thus equal. Otherwise, since M is deterministic, there is a unique q, σ-rule. Here, since t is a leaf node, the rule cannot contain procedure calls. Thus the rule must have a form $(r_1, \dots, r_d)$. So we have:

$$(\text{let}\ (z_1, \dots, z_d) = \langle q, t\rangle(s_1, \dots, s_k)\ \text{in}\ \kappa){\downarrow}$$
$$= (\kappa[z_1/r'_1, \dots, z_d/r'_d]){\downarrow}$$
$$\text{where}\ r'_i = r_i[y_1/s_1, \dots, y_d/s_d]$$

But here, since we have $\{(r'_1, \dots, r'_i)\} = \langle q, t\rangle(s_1, \dots, s_k){\downarrow}$, we can conclude that this is equal to:

$$\bigcup\{\kappa[z_1/u_1, \dots, z_d/u_d]{\downarrow} \mid (u_1, \dots, u_d) \in \langle q, t\rangle(s_1, \dots, s_k){\downarrow}\}$$

We next consider the case when t is a branch node $\sigma(t_1, \dots, t_m)$. The case when no q, σ-rule is in R is similar to the case of leaf node. Both sides become empty and thus equal. Otherwise, there is a unique q, σ-rule of the form $\beta_1 \cdots \beta_n(r_1, \dots, r_d)$ Then

$$(\text{let}\ (z_1, \dots, z_d) = \langle q, t\rangle(s_1, \dots, s_k)\ \text{in}\ \kappa){\downarrow}$$
$$= (\beta'_1 \cdots \beta'_n \kappa'){\downarrow}$$

where $\beta'_i = \beta_i[x_1/t_1, \dots, x_m/t_m, y_1/s_1, \dots, y_k/s_k]$, $\kappa' = \kappa[z_1/r'_1, \dots, z_d/r'_d]$, and $r'_i = r_i[y_1/s_1, \dots, y_k/s_k]$. Let β_1 be let $z_{\vec{\beta 1}} = \langle q_1, t_{i_1}\rangle(\cdots)$ in. By induction hypothesis, we can first evaluate the procedure call in β_1 first, and replace the variables $z_1, \dots, z_{d1}$ occurring in the rest by the result. Thus we have

$$(\beta'_1 \cdots \beta'_n \kappa'){\downarrow}$$
$$= \bigcup\{(\beta'_2 \cdots \beta'_n \kappa')[z_{\vec{\beta 1}}/u_{\vec{\beta 1}}]{\downarrow} \mid u_{\vec{\beta 1}} \in \langle q_1, t_{i1}\rangle(\cdots){\downarrow}\}$$

and, by expanding the remaining calls $\beta'_2, \dots, \beta'_n$ similarly, we have:

$$= \bigcup\{\kappa'[z_{\vec{\beta 1}}/u_{\vec{\beta 1}}] \cdots [z_{\vec{\beta n}}/u_{\vec{\beta n}}]{\downarrow} \mid$$
$$u_{\vec{\beta 1}} \in \langle q_1, t_{i_1}\rangle(\cdots){\downarrow},$$
$$u_{\vec{\beta 2}} \in \langle q_2, t_{i_2}\rangle(\cdots)[z_{\vec{\beta 1}}/u_{\vec{\beta 1}}]{\downarrow},$$
$$\vdots$$

$$u_{\vec{\beta}n} \in \langle q_n, t_{i_n}\rangle(\cdots)[z_{\vec{\beta}1}/u_{\vec{\beta}1}]\cdots[z_{\beta\vec{n-1}}/u_{\beta\vec{n-1}}]\downarrow\}$$

On the other hand, a similar reasoning leads us to:

$$\begin{aligned}
&\langle q,t\rangle(s_1,\ldots,s_k)\downarrow= \\
&\quad\{(r'_1,\ldots,r'_d)[z_{\vec{\beta}1}/u_{\vec{\beta}1}]\cdots[z_{\vec{\beta}n}/u_{\vec{\beta}n}] \mid \\
&\qquad u_{\vec{\beta}1} \in \langle q_1,t_1\rangle(\cdots)\downarrow, \\
&\qquad u_{\vec{\beta}2} \in \langle q_2,t_2\rangle(\cdots)[z_{\vec{\beta}1}/u_{\vec{\beta}1}]\downarrow, \\
&\qquad \vdots \\
&\qquad u_{\vec{\beta}n} \in \langle q_n,t_n\rangle(\cdots)[z_{\vec{\beta}1}/u_{\vec{\beta}1}]\cdots[z_{\beta\vec{n-1}}/u_{\beta\vec{n-1}}]\downarrow\}
\end{aligned}$$

So by rewriting $\langle q,t\rangle(s_1,\ldots,s_k)\downarrow$ in the right-hand side of the lemma statement according to the above equation, the question boils down to the following equation

$$\kappa[\vec{z}/\vec{r'}][z_{\vec{\beta}}/u_{\vec{\beta}}] = \kappa[\vec{z}/(\vec{r'}[z_{\vec{\beta}}/u_{\vec{\beta}}])]$$

which holds by the associativity of substitutions. □

Now we define the split operation on mr-MTTs and prove that it preserves their meanings.

Definition 12. Let $M = (Q, q_0, \Sigma, \Delta, R, D)$ be an mr-MTT. We define a new mr-MTT $\mathsf{split}(M) = (Q', q'_0, \Sigma, \Delta, R', D')$ as follows.

$$\begin{aligned}
Q' &= \bigcup\{\{q_{[1]},\ldots,q_{[D(q)]}\} \mid q \in Q\} \\
q'_0 &= q_{0[1]} \\
R' &= \{\langle q_{[i]}, \sigma(x_1,\ldots,x_m)\rangle(y_1,\ldots,y_k) \to \mathsf{split}_i(r) \mid \\
&\quad (\langle q, \sigma(x_1,\ldots,x_m)\rangle(y_1,\ldots,y_k) \to r) \in R, 1 \le i \le D(q)\} \\
D' &= q \mapsto 1 \quad \text{(constant function returning 1)}
\end{aligned}$$

where split_i is defined as follows:

$$\begin{aligned}
&\mathsf{split}_i((u_1,\ldots,u_d)) = u_i \\
&\mathsf{split}_i(\text{let } (z_1,\ldots,z_n) = \langle q,x\rangle(u_1,\ldots,u_k) \text{ in } \kappa) = \\
&\qquad \text{let } z_1 = \langle q_{[1]},x\rangle(u_1,\ldots,u_k) \text{ in} \\
&\qquad\quad \vdots \\
&\qquad \text{let } z_n = \langle q_{[n]},x\rangle(u_1,\ldots,u_k) \text{ in } \mathsf{split}_i(\kappa)
\end{aligned}$$

Note that the split operation preserves the determinism of the input transducer M.

Lemma 2. *Let $M = (Q, q_0, \Sigma, \Delta, R, D)$ be a deterministic mr-MTT. Then we have $\tau(M) = \tau(\mathsf{split}(M))$.*

Proof. We prove that for each state $q \in Q$ and any trees $t \in T_\Sigma$, $s_1,\ldots,s_k \in T_\Delta$, the following equation holds, applying which to q_0 yields $\tau(M) = \tau(\mathsf{split}(M))$.

$$\begin{aligned}
&\langle q,t\rangle(s_1,\ldots,s_k)\downarrow= \\
&\qquad \langle q_{[1]},t\rangle(s_1,\ldots,s_k)\downarrow \times\cdots\times \langle q_{[D(q)]},t\rangle(s_1,\ldots,s_k)\downarrow
\end{aligned}$$

The proof is done by induction on input trees t. We first consider the case when t is a leaf node $\sigma()$. If there is no q,σ-rule in M, there is no $q_{[i]},\sigma$-rule for every i, too. Hence, both sides of the equation are $\emptyset$ and thus equal. Otherwise, since M is deterministic, there is a unique q,σ-rule. Since σ is a leaf, the rule cannot contain procedure calls. So the rule has the form $(r_1,\ldots,r_{D(q)})$ for some $r_1,\ldots,r_{D(q)} \in T_{\Delta\cup Y}$. The corresponding $q_{[i]},\sigma$-rule in $\mathsf{split}(M)$ is r_i. Thus, both sides of the equation denote the equivalent set $\{(r'_1,\ldots,r'_{D(q)})\}$ where $r'_i = r_i[y_1/s_1,\ldots,y_k/s_k]$. Note that the determinacy is critically used here. That is, all $q_{[i]},\sigma$-rules selected correspond to the same q,σ-rule in the original mr-MTT, since such rule is unique.

We next consider the case when t is a branch node $\sigma(t_1,\ldots,t_m)$. Again, if there is no q,σ-rule, both sides of the equation are empty and thus equal. Otherwise, there is a unique q,σ-rule κ, and what we have to show is: $\kappa[\vec{x}/\vec{t},\vec{y}/\vec{s}]\downarrow= \mathsf{split}_1(\kappa)[\vec{x}/\vec{t},\vec{y}/\vec{s}]\downarrow \times\cdots\times \mathsf{split}_{D(q)}(\kappa)[\vec{x}/\vec{t},\vec{y}/\vec{s}]\downarrow$. We prove this by an inner induction on the structure of r, showing that the following equation holds for any $\vec{x},\vec{t},\vec{y},\vec{s},\vec{z}$ and $\vec{u}$.

$$\begin{aligned}
&\kappa[\vec{x}/\vec{t},\vec{y}/\vec{s},\vec{z}/\vec{u}]\downarrow= \\
&\quad \mathsf{split}_1(\kappa)[\vec{x}/\vec{t},\vec{y}/\vec{s},\vec{z}/\vec{u}]\downarrow \times\cdots\times \mathsf{split}_{D(q)}(\kappa)[\vec{x}/\vec{t},\vec{y}/\vec{s},\vec{z}/\vec{u}]\downarrow
\end{aligned}$$

The base case, which is the case when κ has no let bindings, is proved in the same way as the case of a leaf node. The case when it contains let bindings, i.e., has the following form

$$\text{let } (z_1,\ldots,z_d) = \langle p,x_j\rangle(r_1,\ldots,r_l) \text{ in } \kappa'$$

where $d = D(p)$, the left hand side of the equation becomes

$$(\text{ let } (z_1,\ldots,z_d) = \langle p,t_j\rangle(r'_1,\ldots,r'_l) \text{ in } \kappa'')\downarrow$$

where $r'_i = r_i[y_1/s_1,\ldots y_k/s_k]$ for each i and $\kappa'' = \kappa'[x_1/t_1,\ldots, x_m/t_m, y_1/s_1,\ldots y_k/s_k]$. This is, by Lemma 1, equivalent to:

$$\begin{aligned}
&\bigcup\{\kappa''[z_1/u_1,\ldots,z_d/u_d]\downarrow \mid \\
&\qquad (u_1,\ldots,u_d) \in \langle p,t_j\rangle(r'_1,\ldots,r'_l)\downarrow\}
\end{aligned}$$

By outer induction hypothesis, this is equivalent to:

$$\begin{aligned}
&\bigcup\{\kappa''[z_1/u_1,\ldots,z_d/u_d]\downarrow \mid \\
&\qquad u_1 \in \langle p_{[1]},t_j\rangle(r'_1,\ldots,r'_l)\downarrow,\cdots, \\
&\qquad u_d \in \langle p_{[D(p)]},t_j\rangle(r'_1,\ldots,r'_l)\downarrow\}
\end{aligned}$$

Here, each u_i and $\langle p_{[i]},t_j\rangle(r'_1,\ldots,r'_l)$ do not contain variables. Thus, the substitution applied to κ' can be distributed, and we have an equivalent set

$$\begin{aligned}
&\bigcup\{\kappa''[z_1/u_1,\ldots,z_d/u_d]\downarrow \mid \\
&\qquad u_1 \in \langle p_{[1]},t_j\rangle(r'_1,\ldots,r'_l)\downarrow, \\
&\qquad u_2 \in \langle p_{[2]},t_j\rangle(r'_1,\ldots,r'_l)[z_1/u_1]\downarrow,\cdots, \\
&\qquad u_d \in \langle p_{[d]},t_j\rangle(r'_1,\ldots,r'_l)[z_1/u_1]\cdots[z_{d-1}/u_{d-1}]\downarrow\}
\end{aligned}$$

which is, by inner induction hypothesis, equivalent to

$$\begin{aligned}
&\bigcup\{\mathsf{split}_1(\kappa'')[z_1/u_1,\ldots,z_d/u_d]\downarrow \\
&\qquad \times\cdots\times \mathsf{split}_{D(q)}(\kappa'')[z_1/u_1,\ldots,z_d/u_d]\downarrow \mid \\
&\qquad u_1 \in \langle p_{[1]},t_j\rangle(r'_1,\ldots,r'_l)\downarrow, \\
&\qquad u_2 \in \langle p_{[2]},t_j\rangle(r'_1,\ldots,r'_l)[z_1/u_1]\downarrow,\cdots, \\
&\qquad u_d \in \langle p_{[d]},t_j\rangle(r'_1,\ldots,r'_l)[z_1/u_1]\cdots[z_{d-1}/u_{d-1}]\downarrow\}
\end{aligned}$$

Since $\mathsf{split}(M)$ is deterministic, we have an equivalent set

$$\begin{aligned}
&\bigcup\{\mathsf{split}_1(\kappa'')[z_1/u_1,\ldots,z_d/u_d]\downarrow \mid \\
&\qquad u_1 \in \langle p_{[1]},t_j\rangle(r'_1,\ldots,r'_l)\downarrow, \\
&\qquad u_2 \in \langle p_{[2]},t_j\rangle(r'_1,\ldots,r'_l)[z_1/u_1]\downarrow,\cdots, \\
&\qquad u_d \in \langle p_{[d]},t_j\rangle(r'_1,\ldots,r'_l)[z_1/u_1]\cdots[z_{d-1}/u_{d-1}]\downarrow\} \\
&\times\cdots\times \\
&\bigcup\{\mathsf{split}_{D(q)}(\kappa'')[z_1/u_1,\ldots,z_d/u_d]\downarrow \mid \\
&\qquad u_1 \in \langle p_{[1]},t_j\rangle(r'_1,\ldots,r'_l)\downarrow, \\
&\qquad u_2 \in \langle p_{[2]},t_j\rangle(r'_1,\ldots,r'_l)[z_1/u_1]\downarrow,\cdots, \\
&\qquad u_d \in \langle p_{[d]},t_j\rangle(r'_1,\ldots,r'_l)[z_1/u_1]\cdots[z_{d-1}/u_{d-1}]\downarrow\}
\end{aligned}$$

This is, by Lemma 1 and the definition of split_i, equivalent to

$$\mathsf{split}_1(\kappa)[\vec{x}/\vec{t},\vec{y}/\vec{s},\vec{z}/\vec{u}]{\downarrow} \times \cdots \times \mathsf{split}_{D(q)}(\kappa)[\vec{x}/\vec{t},\vec{y}/\vec{s},\vec{z}/\vec{u}]{\downarrow}$$

as desired. □

As a remark, we mention here without proof, that for a nondeterministic mr-MTT M, we have $\tau(M) \subseteq \tau(\mathsf{split}(M))$.

Next, we define the build operation that converts an mr-MTT to an MTT. Basically, it simply rewrites let $z = e_1$ in e_2 by $e_2[z/e_1]$ and the determinacy assures the equivalence. However, there is a subtle difference in the semantics. Namely, since a deterministic mr-MTT could in fact be non-total (i.e., it may have no q,σ-rule for some q and σ), if it binds a variable z to a procedure call that yields no result and the variable is not mentioned after the binding, then a wrongly built MTT might eliminate the call and yield some result. To deal with this subtlety, we use an extra trick to ensure such elimination never to happen.

Definition 13. Let $M = (Q, q_0, \Sigma, \Delta, R, D)$ be an mr-MTT such that D is a constant function returning 1. We define an MTT $\mathsf{build}(M) = (Q', q_0, \Sigma, \Delta, R')$ as follows.

$$\begin{aligned} Q' &= Q \cup \{q_{id}^2\} \\ R' &= \{\langle q,\sigma(x_1,\ldots,x_m)\rangle(y_1,\ldots,x_k) \to \mathsf{build}(r,\emptyset) \mid \\ &\qquad (\langle q,\sigma(x_1,\ldots,x_m)\rangle(y_1,\ldots,x_k) \to r) \in R\} \\ &\quad \cup \{\langle q_{id},\sigma^m(x_1,\ldots,x_m)\rangle(y_1,y_2) \to y_2 \mid \sigma^m \in \Sigma\} \end{aligned}$$

where

$$\begin{aligned} &\mathsf{build}(t,\{z_1,\ldots,z_n\}) \\ &\qquad = \langle q_{id},x_1\rangle(z_1,\langle q_{id},x_1\rangle(\cdots,\langle q_{id},x_1\rangle(z_n,t)\cdots)) \\ &\mathsf{build}(\text{let } z = \langle q,x\rangle(s_1,\ldots,s_k) \text{ in } r, V) \\ &\qquad = \mathsf{build}(r, V\cup\{z\})[z/\langle q,x\rangle(s_1,\ldots,s_k)] \end{aligned}$$

Note that the build operation preserves the determinism of the input transducer M.

The extra state q_{id} ignores the input and the first parameter and simply returns the second parameter; this state is used in the internally defined build function. The internal build function takes a right hand side of an mr-MTT and its possible free variables, and returns a right hand side of an MTT. In the first rule of build, for each variable z_i, we call q_{id} with the first argument z_i. Since each such variable will be replaced with the corresponding procedure call by the second rule, this treatment makes sure that all procedure calls that are made in the original mr-MTT are indeed made in the converted MTT.

Lemma 3. *Let $M = (Q, q_0, \Sigma, \Delta, R, D)$ be a deterministic mr-MTT such that D is a constant function returning* 1*. Then the MTT* $\mathsf{build}(M)$ *realizes the same translation as M.*

Proof. We prove the following for any $q \in Q$, $t \in T_\Sigma$, and $s_1,\ldots,s_k \in T_\Delta$ by induction on the structure on input trees t:

$$\langle q,t\rangle(s_1,\ldots,s_k){\downarrow_M} = \langle q,t\rangle(s_1,\ldots,s_k){\downarrow_{\mathsf{build}(M)}}$$

If t is a leaf $\sigma()$, then a q,σ-rule (1) does not exist for either side, or (2) uniquely exists for both sides in exactly the same form. In either case, obviously both sides become equal.

If t is a branch $\sigma(t_1,\ldots,t_m)$, it is either that (1) a q,σ-rule does not exist for either side, or (2) uniquely exists for both sides, of the form r and $\mathsf{build}(r,\emptyset)$. In the non-existing case, both sides become empty and thus equal. For the case that the corresponding rules exist, what we have to prove is: $r[\vec{x}/\vec{t},\vec{y}/\vec{s}]{\downarrow_M} = \mathsf{build}(r,\emptyset)[\vec{x}/\vec{t},\vec{y}/\vec{s}]{\downarrow_{\mathsf{build}(M)}}$. We prove this by showing that for any sentential form κ of M such that each input tree contained in κ is a subtree of t, for any $Z = \{z_1,\ldots,z_n\}$, and $u_1,\ldots,u_n \in T_\Delta$, the following holds:

$$\kappa[\vec{z}/\vec{u}]{\downarrow_M} = \mathsf{build}(\kappa,Z)[\vec{z}/\vec{u}]{\downarrow_{\mathsf{build}(M)}}$$

The proof is done by induction on the number of lets in sentential forms. When κ does not have any let bindings, it is that $\kappa \in T_\Delta$. So both sides derives only a tree κ, thus are equal. If rule κ has the form:

$$\text{let } z' = \langle q,t_i\rangle(s_1,\ldots,s_k) \text{ in } \kappa'$$

We have

$$\begin{aligned} &\kappa[\vec{z}/\vec{u}]{\downarrow_M} \\ =&(\text{let } z' = \langle q,t_i\rangle(s_1,\ldots,s_k) \text{ in } \kappa')[\vec{z}/\vec{u}]{\downarrow_M} \\ =&\bigcup\{\kappa'[\vec{z}/\vec{u}][z'/u']{\downarrow_M} \mid u' \in \langle q,t_i\rangle(s_1[\vec{z}/\vec{u}],\ldots,s_k[\vec{z}/\vec{u}]){\downarrow_M}\} \\ &\quad \text{by Lemma 1} \\ =&\bigcup\{\mathsf{build}(\kappa',Z\cup\{z'\})[\vec{z}/\vec{u}][z'/u']{\downarrow_{\mathsf{build}(M)}} \\ &\quad \mid u' \in \langle q,t_i\rangle(s_1[\vec{z}/\vec{u}],\ldots,s_k[\vec{z}/\vec{u}]){\downarrow_{\mathsf{build}(M)}}\} \\ &\quad \text{by the hypotheses of both the inner and the outer induction} \\ =&\mathsf{build}(\kappa'[z'/\square_1],Z\cup\{\square_1\})[\vec{z}/\vec{u}] \\ &\quad [\langle q,t_i\rangle(s_1[\vec{z}/\vec{u}],\ldots,s_k[\vec{z}/\vec{u}])]{\downarrow_{\mathsf{build}(M)}} \\ &\quad \text{by Prop 2 (by definition, } \mathsf{build}(\cdots,Z\cup\{\square_1\}) \text{ has at least} \\ &\quad \text{one occurrence of } \square_1 \text{ and } \mathsf{build}(M) \text{ is deterministic)} \\ =&\mathsf{build}(\kappa,Z)[\vec{z}/\vec{u}]{\downarrow_{\mathsf{build}(M)}} \text{ by commuting substitutions} \end{aligned}$$

as desired. □

Again, as a remark, we have $\tau(M) \subseteq \tau(\mathsf{build}(M))$ for a nondeterministic mr-MTT M of dimension 1. The proof is omitted.

Now, by Lemma 2 and 3, we have the following proposition as desired.

Prop 3. *For any deterministic mr-MTT, there effectively exists a deterministic MTT that realizes the same translation.*

4.2 The nondeterministic case

When nondeterministic choices are used, the split operation on an mr-MTT does not retain the realized translation. Consider the following mr-MTT with the input alphabet $\{\mathtt{s}^1,\mathtt{z}^0\}$, the output alphabet $\{\mathtt{root}^2,\mathtt{a}^1,\mathtt{b}^1,\mathtt{e}^0,\mathtt{A}^1,\mathtt{B}^1,\mathtt{E}^0\}$, and the set of rules:

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

This mr-MTT nondeterministically translates a string $\mathtt{ss}\cdots\mathtt{ssz}$ of length n to a `root` node holding two strings of the same length n. The first string in the output consists of symbols `a` and `b` and terminates by `e`. The second consists of `A` and `B` and terminates by `E`. Moreover, the second string is always the reverse of the first, ignoring the case and the leaf symbol (`e` or `E`). Note that if we split the procedure q_1 returning pairs of trees into two single-return procedures, we have a different translation realized. That is, although the split version still generates trees holding two strings (one with symbols `a` and `b`, and the other with `A` and `B`), the two strings, this time, are not necessarily related (not always the reverse of each other).

In fact, not only the split version of the above mr-MTT but *any* MTT cannot realize the translation realized by the above mr-MTT.

We refer to this translation by twist throughout this paper. More precisely, abbreviating $\bar{k}$ for $\overbrace{\mathtt{s}\ldots\mathtt{s}}^{k}\mathtt{z}$, we define

$$\mathsf{twist} = \{(\bar{n},\ \mathtt{root}(t, \mathsf{revUp}(t))) \mid t \in (\mathtt{a}|\mathtt{b})^n\mathtt{e}\}$$

where:

$$\begin{aligned}\mathsf{revUp}(x) &= \mathsf{revUp}'(x, \mathtt{E}())\\ \mathsf{revUp}'(\mathtt{a}(x), y) &= \mathsf{revUp}'(x, \mathtt{A}(y))\\ \mathsf{revUp}'(\mathtt{b}(x), y) &= \mathsf{revUp}'(x, \mathtt{B}(y))\\ \mathsf{revUp}'(\mathtt{e}(), y) &= y\end{aligned}$$

Now, the goal of the rest of this section is to prove the following proposition, which takes twist as a witness that nondeterministic mr-MTTs are strictly more expressive than MTTs.

Prop 4. *No macro tree transducer realizes* twist.

The outline of the proof is as follows. We first suppose that an MTT realizes twist and derive a contradiction. How we derive it is to show that the MTT that supposedly realizes twist yields a set of outputs whose size has a polynomial upper bound with respect to the length of the input string, while the size of twist's output set is actually exponential.

However, giving this upper bound directly is difficult since too many possibilities need to be considered for the supposed MTT. For this reason, we introduce two restricted forms, called *weak normal form* and *(strong) normal form* and use these as follows. Given a sentential form that produces the desired outputs (in particular, the one to start with, i.e., the initial procedure applied to an input $\bar{n}$), we will convert it to a weak normal form, and then to a set of normal forms. We can show that, in the first conversion, the set of outputs is preserved and, in the second conversion, the union set of outputs is either preserved or enlarged. We then give an upper bound for the size of the final union set, which is also an upper bound for the original one.

Below, without loss of generality, we consider only MTTs with input alphabet $\Sigma = \{\mathtt{s}^1, \mathtt{z}^0\}$ and output alphabet $\Delta = \{\mathtt{root}^2, \mathtt{a}^1, \mathtt{b}^1, \mathtt{e}^0, \mathtt{A}^1, \mathtt{B}^1, \mathtt{E}^0\}$. Also, we write $O_{\bar{n}}$ to denote the output language of twist from a particular input $\bar{n}$, i.e., $O_{\bar{n}} = \{\mathtt{root}(t, \mathsf{revUp}(t)) \mid t \in (\mathtt{a}|\mathtt{b})^n\mathtt{e}\}$, and O to denote the output language from any input, i.e., $O = \bigcup_{n\geq 0} O_{\bar{n}}$.

Conversion to weak normal form As in the following definition, a weak normal form is a sentential form whose each subtree is either a desired output ($O_{\bar{n}}$), a tree with no $\mathtt{root}$ ($T_{\Delta\setminus\{\mathtt{root}\}}$), or a procedure call where its arguments are sentential forms and that produces some and only desired outputs.

Definition 14. Let $M = (Q, q_0, \Sigma, \Delta, R)$ be an MTT realizing twist, and $\bar{n} \in T_\Sigma$. We define the set of *weak normal forms* $W_\Lambda^{\bar{n}} \subseteq T_\Lambda$ (recall $\Lambda = \Delta \cup (Q \times T_\Sigma)$) as the minimum set satisfying the following conditions:

- $O_{\bar{n}} \cup T_{\Delta\setminus\{\mathtt{root}\}} \subseteq W_\Lambda^{\bar{n}}$
- $v \in W_\Lambda^{\bar{n}}$ if $v = \langle q^k, \bar{m}\rangle(v_1, \ldots, v_k)$ and $v_1, \ldots, v_k \in W_\Lambda^{\bar{n}}$ with $\emptyset \subsetneq v{\downarrow} \subseteq O_{\bar{n}}$.

To show that a sentential form yielding only desired outputs can be converted to an equivalent weak normal form, the following lemma is crucial. Intuitively, this states that, when we know that such a sentential form contains a subtree r that derives a tree s with no $\mathtt{root}$, we do not have to consider other derivations starting from the subtree r, that is, replacing the subtree r by its specific result s will not change the set of outputs from the whole sentential form.

Lemma 4. *Suppose* $M = (Q, q_0, \Sigma, \Delta, R)$ *to be an MTT realizing* twist. *Let* $\mathcal{C}$ *be a one-hole* Λ*-context and* r *be a tree in* T_Λ*, such that* $\mathcal{C}[r]{\downarrow} \subseteq O$ *and* $r \Rightarrow^* s$ *where* $s \in T_{\Delta\setminus\{\mathtt{root}\}}$. *Then* $\mathcal{C}[r] \equiv \mathcal{C}[s]$.

Proof. Choose any $\mathcal{D} \in \mathcal{C}{\downarrow}$. By Prop 2, it must be the case that $\mathcal{D}[s] \in \mathcal{C}[r]{\downarrow}$. Since $\mathcal{C}[r]{\downarrow} \subseteq O$ and $s \in T_{\Delta\setminus\{\mathtt{root}\}}$, the context $\mathcal{D}$ must be either in the form

- that does not contain any $\square_1$,
- $\mathtt{root}(\cdots\square_1\cdots, \cdots)$, or
- $\mathtt{root}(\cdots, \cdots\square_1\cdots)$.

In the first case, it is trivial that, for all $s' \in r{\downarrow}$, we have $\mathcal{D}[s'] = \mathcal{D} = \mathcal{D}[s]$. In the second case, by Prop 1, for all $s' \in r{\downarrow}$ we have $\mathcal{D}[s'] \in O$. So here, $\mathsf{revUp}(s')$ must be equal to $\mathsf{revUp}(s)$. Since revUp is one-to-one mapping, this implies $s' = s$, and thus we conclude $\mathcal{D}[s'] = \mathcal{D}[s]$. The third case is similar to the second case, and again we have $\mathcal{D}[s'] = \mathcal{D}[s]$ for all $s' \in r{\downarrow}$. So in any cases, we have $\mathcal{D}[s'] = \mathcal{D}[s]$ for all $\mathcal{D} \in \mathcal{C}{\downarrow}$ and $s' \in r{\downarrow}$. So $\mathcal{C}[r]{\downarrow} = \{\mathcal{D}[s'] \mid \mathcal{D} \in \mathcal{C}{\downarrow}, s' \in r{\downarrow}\} = \{\mathcal{D}[s] \mid \mathcal{D} \in \mathcal{C}{\downarrow}\} = \mathcal{C}[s]{\downarrow}$ □

By using the above lemma, we show that, after applying this replacement for all such subtrees r as well as replacing all illegitimate trees (like a $\mathtt{root}$ containing a $\mathtt{root}$ or an $\mathtt{a}$ containing a $\mathtt{root}$) with a legitimate but nonce tree ($\mathtt{e}()$ in the proof), we will obtain a weak normal form.

Lemma 5. *Suppose* $M = (Q, q_0, \Sigma, \Delta, R)$ *to be an MTT realizing* twist. *For any* $r \in T_\Lambda$ *where* $\emptyset \subsetneq r{\downarrow} \subseteq O_{\bar{n}}$ *for some* $\bar{n} \in T_\Sigma$*, there exists a tree* $w_{\bar{n}}(r) \in W_\Lambda^{\bar{n}}$ *such that* $w_{\bar{n}}(r) \equiv r$.

Proof. To obtain $w_{\bar{n}}(r)$, we first apply the following translation to r repeatedly, until subtrees satisfying the condition are not found.

- Find a subtree r_c such that $r_c{\downarrow} \not\subseteq O_{\bar{n}} \cup T_{\Delta\setminus\{\mathtt{root}\}}$. If found, replace r_c in r by $\mathtt{e}()$.

Since such r_cs will never be contained in the final output, i.e., there is no context $\mathcal{D} \in \mathcal{C}{\downarrow}$ containing an occurrence of $\square_1$, where $r = \mathcal{C}[r_c]$ and $\mathcal{C}$ is a one-hole context. So replacing it by a nonce subtree $\mathtt{e}()$ still yields an equivalent sentential form.

Next, we repeatedly apply the following translation.

- Find a subtree $r_c = \langle q, \bar{m}\rangle(r_1, \ldots, r_k)$ of r such that $r_c \Rightarrow^* s \in T_{\Delta\setminus\{\mathtt{root}\}}$. If found, replace r_c in r by s.

Since one iteration of this translation decreases the number of $Q \times T_\Sigma$ nodes in r, this process terminates. By Lemma 4, the result of this translation is equivalent to r.

Finally, we repeatedly apply the following translation.

- Find a subtree $r_c = \langle q, \bar{m}\rangle(r_1, \ldots, r_k)$ of r such that r_c is an descendant of some node of the form $\delta(\cdots)$ where $\delta \in \Delta$. If found, replace r_c in r by $\mathtt{e}()$.

At this stage after the preceding two translation, it must be that $r_c{\downarrow} \subseteq O_{\bar{n}}$. Again, such r_cs will never be contained in the final output. So replacing it by $\mathtt{e}()$ still yields an equivalent sentential form.

After these three translations, each subtree of form $r_c = \langle q, \bar{m}\rangle(\cdots)$ does not appear below $\delta(\cdots)$ nodes, and $r_c{\downarrow} \subseteq O_{\bar{n}}$. Also for each subtree of form $r_c = \delta(\cdots)$, we have $r_c{\downarrow} \subseteq O_{\bar{n}} \cup T_{\Delta\setminus\{\mathtt{root}\}}$. Also by Prop 2, $r_c{\downarrow} \neq \emptyset$ for all subtrees r_c of r, since $r{\downarrow} \neq \emptyset$. Thus the translated tree $w(r)$ is now contained in $W_\Lambda^{\bar{n}}$. □

Conversion to normal form As in the following definition, a normal form is a sentential form that has a unique procedure call at the root whose each argument is either a tree with no `root` or a dummy tree and that yields only desired outputs.

Definition 15. Let M be an MTT realizing twist, and $\bar{n} \in T_\Sigma$. We define the set of *normal forms* $N^{\bar{n}}_\Lambda \subseteq W^{\bar{n}}_\Lambda$ as

$$N^{\bar{n}}_\Lambda = \{s \mid s = \langle q^k, \bar{m}\rangle(s_1, \ldots, s_k),\ q^k \in Q, \bar{m} \in T_\Sigma,\\ s_i \in \{\mathsf{dummy}_n\} \cup T_{\Delta\setminus\{\mathtt{root}\}},\ \emptyset \subsetneq s{\downarrow} \subseteq O_{\bar{n}}\}$$

where:

$$\mathsf{dummy}_n = \mathtt{root}(\overbrace{\mathtt{a}\cdots\mathtt{a}}^{n}\mathtt{e}, \overbrace{\mathtt{A}\cdots\mathtt{A}}^{n}\mathtt{E})$$

(We take here a dummy tree as the above, but it can be any tree if it is in $O_{\bar{n}}$.)

Before showing that a weak normal form can be converted to a set of normal forms, we give the following lemma. Intuitively, this states that if a sentential form that derives only desired outputs can be split to a context with several holes and desired output trees, then the context itself derives a hole or a desired output tree without using any hole. (The lemma holds for not only weak normal forms but any sentential forms.)

Lemma 6. *Suppose $M = (Q, q_0, \Sigma, \Delta, R)$ to be an MTT realizing* twist*, and $\bar{n} \in T_\Sigma$. Let $\mathcal{C}$ be a Λ-c-context such that $\mathcal{C}[s_1, \ldots, s_c]{\downarrow} \subseteq O_{\bar{n}}$ for some $s_1, \ldots, s_c \in O_{\bar{n}}$. Then $\mathcal{C}{\downarrow} \subseteq \{\Box_1, \ldots, \Box_c\} \cup O_{\bar{n}}$.*

Proof. Suppose a Δ-context $\mathcal{D} \in \mathcal{C}{\downarrow}$ such that $\mathcal{D} \notin \{\Box_1, \ldots, \Box_c\} \cup O_{\bar{n}}$. By the condition $\mathcal{C}[s_1, \ldots, s_c]{\downarrow} \subseteq O_{\bar{n}}$, it must be the case $\mathcal{D}[s_1, \ldots, s_c] \in O_{\bar{n}}$ (by Prop 1). Since the root node of every tree in $O_{\bar{n}}$ is `root` and $\mathcal{D} \notin \{\Box_1, \ldots, \Box_c\}$ by the assumption, the root node of $\mathcal{D}$ is `root`. Also by the assumption $\mathcal{D} \notin O_{\bar{n}}$, for some $1 \leq i \leq n$, the hole $\Box_i$ is a part of $\mathcal{D}$. So, $\mathcal{D}[s_1, \ldots, s_c]$ contains s_i as a subtree. Hence, the tree $\mathcal{D}[s_1, \ldots, s_c]$ contains at least two `root` nodes, one at the root position and the other at the position of s_i. This implies $\mathcal{D}[s_1, \ldots, s_c] \notin O_{\bar{n}}$, which is a contradiction. □

Conversion from a weak normal form to a set of normal forms works as follows. Recall that a weak normal form is a tree of procedure calls where each leaf is either from $O_{\bar{n}}$ or from $T_{\Delta\setminus\{\mathtt{root}\}}$ and each procedure call derives an output tree from $O_{\bar{n}}$. We repeat the following. If the root procedure call of a weak normal form t has other procedure calls as arguments, e.g.,

$$t = \langle q, \bar{m}\rangle(\langle q', \bar{m'}\rangle(\ldots), \langle q'', \bar{m''}\rangle(\ldots), s)$$

(where the arguments are two calls and a tree from $T_{\Delta\setminus\{\mathtt{root}\}}$) then the previous lemma ensures that the root call results in either a result of one of the inner calls or a result of the root call without using the inner calls. Thus,

$$t{\downarrow} \subseteq \langle q', \bar{m'}\rangle(\ldots){\downarrow} \cup \langle q'', \bar{m''}\rangle(\ldots){\downarrow} \cup (\langle q, \bar{m}\rangle(\Box_1, \Box_2, s){\downarrow} \cap O_{\bar{n}})$$

By repeating this "expansion" on the first and second clauses, the result set of t can be bounded by the union of clauses like the third clause above, that is, the results of any procedure calls appearing in t without using any of its inner procedure call arguments (more precisely, in fact, any argument deriving an output in $O_{\bar{n}}$). Note that the results of each clause like the third clause above are further bounded by the call form where each hole is replaced by the dummy tree defined above, like

$$\langle q, \bar{m}\rangle(\mathsf{dummy}_n, \mathsf{dummy}_n, s).$$

We take the set of all such forms constructed from the original weak normal form as the set of the converted normal forms.

Lemma 7. *Suppose $M = (Q, q_0, \Sigma, \Delta, R)$ to be an MTT realizing* twist*. Let $\bar{n} \in T_\Sigma$. There are mappings $N_Q : W^{\bar{n}}_\Lambda \to \mathcal{P}(N^{\bar{n}}_\Lambda)$ and $N_V : W^{\bar{n}}_\Lambda \to \mathcal{P}(T_\Delta)$ such that, for any $v \in W^{\bar{n}}_\Lambda$,*

- $v{\downarrow} \subseteq \bigcup\{u{\downarrow} \mid u \in N_Q(v)\} \cup N_V(v)$
- $|N_Q(v)| \leq$ *the number of $Q \times T_\Sigma$ nodes in v*
- $|N_V(v)| \leq$ *the number of maximal Δ nodes in v*

Here, a maximal Δ node means a node that is labeled by an element of Δ and no symbol of Δ occurs further up in v.

Proof. We take N_Q and N_V as follows:

$$N_Q(v) = \{\mathcal{R}_{v_s}[\mathsf{dummy}_n, \ldots, \mathsf{dummy}_n] \mid\\ v_s = \langle q^k, \bar{m}\rangle(v_1, \ldots, v_k) \text{ is a subtree of } v\}$$

$$N_V(v) = \{v_\delta \mid v_\delta \text{ is a maximal subtree of } v \text{ labeled by } \Delta\}$$

where $\mathcal{R}_{v_s}$ is a Λ-k-context defined as follows:

$$\mathcal{R}_{v_s} = \langle q^k, \bar{m}\rangle(\alpha_1, \ldots, \alpha_k)\\ \text{where } \alpha_i = \Box_i \text{ if } \emptyset \subsetneq v_i{\downarrow} \subseteq O_{\bar{n}}, \text{ or } \alpha_i = v_i \text{ otherwise}$$

Intuitively, $\mathcal{R}_{v_s}$ is a context obtained from v_s by opening holes at the position of the children that generate $O_{\bar{n}}$ trees. The size conditions and $N_V(v) \subseteq T_\Delta$ immediately follow from the definition of N_Q and N_V.

Since $v \in W^{\bar{n}}_\Lambda$, every subtree v_s of v that has the form $\langle q^k, \bar{m}\rangle(v_1, \ldots, v_k)$, it is that $v_s{\downarrow} \subseteq O_{\bar{n}}$. So by Lemma 6 and Prop 2, we have $\mathcal{R}_{v_s}{\downarrow} \subseteq \{\Box_1, \ldots, \Box_k\} \cup O_{\bar{n}}$. Thus, for all $u = \mathcal{C}_{v_s}[\mathsf{dummy}_n, \ldots, \mathsf{dummy}_n] \in N_Q(v)$, it must be the case $u{\downarrow} \subseteq O_{\bar{n}}$, which implies that $N_Q(v) \subseteq N^{\bar{n}}_\Lambda$.

The inclusion $v{\downarrow} \subseteq \bigcup\{u{\downarrow} \mid u \in N_Q(v)\} \cup N_V(v)$ is proved by induction on the structure of v. Base case is the case when $v \in O_{\bar{n}} \cup T_{\Delta\setminus\{\mathtt{root}\}}$. Since $v{\downarrow} = \{v\}$ and $v \in N_V(v)$ in this case, the inclusion trivially holds. The essential case is when v is in the form $\langle q^k, \bar{m}\rangle(v_1, \ldots, v_k) \in W^{\bar{n}}_\Lambda$ where $v_1, \ldots, v_k \in W^{\bar{n}}_\Lambda$ and $\emptyset \subsetneq v{\downarrow} \subseteq O_{\bar{n}}$.

$$\begin{aligned}
&\langle q^k, \bar{m}\rangle(v_1, \ldots, v_k){\downarrow}\\
&= \{\mathcal{D}[v'_1, \ldots, v'_k] \mid \mathcal{D} \in \mathcal{R}_v{\downarrow}, v'_i \in v_i{\downarrow}\} \text{ by Prop 2}\\
&\subseteq \{\mathcal{D}[v'_1, \ldots, v'_k] \mid\\
&\qquad \mathcal{D} \in \{\Box_1, \ldots, \Box_k\} \cup (\mathcal{R}_v{\downarrow} \cap O_{\bar{n}}), v'_i \in v_i{\downarrow}\} \text{ by Lemma 6}\\
&= v_1{\downarrow} \cup \cdots \cup v_k{\downarrow} \cup (\mathcal{R}_v{\downarrow} \cap O_{\bar{n}})\\
&\subseteq v_1{\downarrow} \cup \cdots \cup v_k{\downarrow} \cup \mathcal{R}_v[\mathsf{dummy}_n, \ldots, \mathsf{dummy}_n]{\downarrow}\\
&\qquad \text{by Prop 2. Since } \Box_i \text{ does not contribute any outputs in } O_{\bar{n}},\\
&\qquad \text{substitution with } \mathsf{dummy}_n \text{ is safe.}\\
&\subseteq \bigcup_{i=1}^{k}\Big(\bigcup\{u{\downarrow} \mid u \in N_Q(v_i)\} \cup N_V(v_i)\Big) \cup\\
&\qquad \mathcal{R}_v[\mathsf{dummy}_n, \ldots, \mathsf{dummy}_n]{\downarrow} \text{ by IH}\\
&\subseteq \bigcup\{u{\downarrow} \mid u \in N_Q(v)\} \cup N_V(v) \text{ by definition of } N_Q \text{ and } N_V
\end{aligned}$$

□

Polynomial upper bound The proof for the polynomial upper bound roughly goes as follows. First, we can consider a series of sets of normal forms given as follows. We start with taking the set of normal forms of the weak normal forms of the initial sentential form. Each normal form here has the form $\langle q, \bar{n}\rangle(\ldots)$. We take a one-step derivation. The resulting sentential form is not a normal form, and therefore we again take the set of normal forms of the weak normal form of this. Each normal form must now have the form $\langle q, \overline{n-1}\rangle(\ldots)$. By repeating this, we obtain $n+1$ sets of normal forms.

The above way of taking a series of sets of normal forms does not, however, ensure that the size of each set is polynomial and might actually grow exponentially. For this reason, each time we

take a set of normal forms, we only choose the set of representatives, namely, only one procedure call for each pair of state and symbol.

The next lemma is crucial to justify this. It intuitively states that if two normal forms are procedure calls with the same pair of state and input node, then no matter what arguments they have, the sets of trees produced by them are *almost* the same where the size of the difference is bounded by a constant not dependent on n. By using this, Lemma 9 will show that taking only the representatives leaves only a polynomial number of remainders.

Lemma 8. *Suppose $M = (Q, q_0, \Sigma, \Delta, R)$ to be an MTT realizing* twist. *Let $\bar{n} \in T_\Sigma$. Let $t = \langle q^k, \bar{m}\rangle(s_1, \ldots, s_k)$ and $t' = \langle q^k, \bar{m}\rangle(s'_1, \ldots, s'_k)$ both in $N^{\bar{n}}_\Lambda$. Then we have*

$$|t{\downarrow} \setminus t'{\downarrow}\ | \leq k(k-1)$$

Proof. Let $\mathcal{C}$ be a context $\langle q^k, \bar{m}\rangle(\Box_1, \ldots, \Box_k)$. Then by Prop 1, we have

$$\begin{aligned} t{\downarrow} \setminus t'{\downarrow} &= \{\mathcal{D}[s_1, \ldots, s_k] \mid \mathcal{D} \in \mathcal{C}{\downarrow}\} \setminus \{\mathcal{D}[s'_1, \ldots, s'_k] \mid \mathcal{D} \in \mathcal{C}{\downarrow}\} \\ &\subseteq \{\mathcal{D}[s_1, \ldots, s_k] \mid \mathcal{D} \in \mathcal{C}{\downarrow}, \mathcal{D}[s_1, \ldots, s_k] \neq \mathcal{D}[s'_1, \ldots, s'_k]\} \end{aligned}$$

We show the size of the last set is less than or equal to $k(k-1)$.

For all $\mathcal{D} \in \mathcal{C}{\downarrow}$, we have $\mathcal{D}[s_1, \ldots, s_k] \in t{\downarrow} \subseteq O_{\bar{n}}$ and $\mathcal{D}[s'_1, \ldots, s'_k] \in t'{\downarrow} \subseteq O_{\bar{n}}$. So $\mathcal{D}$ must have either one of the following forms:

1. $\mathcal{D} = \Box_i$ for some i such that $s_i = s'_i = \mathsf{dummy}_n$.
2. $\mathcal{D} \in O_{\bar{n}}$.
3. $\mathcal{D} = \mathtt{root}(\mathcal{D}_1[\Box_i], s_\mathcal{D})$ where $\mathcal{D}_1$ is a one-hole context, $s_\mathcal{D} \in T_{\Delta\setminus\{\mathtt{root}\}}$ and for some i such that $s_i, s'_i \in T_{\Delta\setminus\{\mathtt{root}\}}$
4. $\mathcal{D} = \mathtt{root}(s_\mathcal{D}, \mathcal{D}_1[\Box_i])$ where $\mathcal{D}_1$ is a one-hole context, $s_\mathcal{D} \in T_{\Delta\setminus\{\mathtt{root}\}}$ and for some i such that $s_i, s'_i \in T_{\Delta\setminus\{\mathtt{root}\}}$.
5. $\mathcal{D} = \mathtt{root}(\mathcal{D}_1[\Box_i], \mathcal{D}_2[\Box_j])$ where $\mathcal{D}_1$ and $\mathcal{D}_2$ are one-hole contexts, for some i, j such that $i \neq j$ and $s_i, s_j, s'_i, s'_j \in T_{\Delta\setminus\{\mathtt{root}\}}$.

In the case 1 and case 2, trivially $\mathcal{D}[s_1, \ldots, s_k] = \mathcal{D}[s'_1, \ldots, s'_k]$ holds. In the case 3, by the definition of $O_{\bar{n}}$, it must be the case that $\mathsf{revUp}(\mathcal{D}_1[s_i]) = s_\mathcal{D} = \mathsf{revUp}(\mathcal{D}_1[s'_i])$. Since revUp is a one-to-one mapping, we can conclude $s_i = s'_i$. Hence, $\mathcal{D}[s_1, \ldots, s_k] = \mathcal{D}[s'_1, \ldots, s'_k]$. The case 4 is similar.

In the case 5, by definition of $O_{\bar{n}}$, the lengths of $\mathcal{D}_1[s_i]$, $\mathcal{D}_2[s_j]$, $\mathcal{D}_1[s'_i]$, and $\mathcal{D}_2[s'_j]$ are all equal to n. So it must be that $length(s_i) = length(s'_i)$ and $length(s_j) = length(s'_j)$, where $length(s)$ denotes the number of rank-1 nodes in s. Also by the definition of $O_{\bar{n}}$, we have $\mathsf{revUp}(\mathcal{D}_1[s_i]) = \mathcal{D}_2[s_j]$ and $\mathsf{revUp}(\mathcal{D}_1[s'_i]) = \mathcal{D}_2[s'_j]$. So the reverse of s_i is being the prefix of $\mathcal{D}_2[s_j]$.

Here, we consider two cases, namely: the case $length(s_i) + length(s_j) \leq n$ and the case $length(s_i) + length(s_j) > n$. For the former case, $length(s_i)$ is less than or equal to $length(\mathcal{D}_2)$. So s_i must be the reverse of a prefix of $\mathcal{D}_2$. Since exactly the same thing holds for s'_i, we can conclude $s_i = s'_i$, and similarly $s_j = s'_j$. So in this case, again we have $\mathcal{D}[s_1, \ldots, s_k] = \mathcal{D}[s'_1, \ldots, s'_k]$. For the latter case, since $length(s_i) > length(\mathcal{D}_2)$, the context $\mathcal{D}_2$ is being a prefix of the reverse ofs_i. Thus, $\mathcal{D}_2$ is uniquely determined from s_i and the length of s_j. By a similar argument, we also have that $\mathcal{D}_1$ is uniquely determined from s_j and the length of s_i. So for a specific i and j, the corresponding context $\mathcal{D}$ of the 5th form is determined uniquely. Since the number of ways to choose two numbers i and j from $1 \cdots k$ is $k(k-1)$, the number of the contexts $\mathcal{D}$ in this form is at most $k(k-1)$.

So by the argument above, we can conclude that the number of context that satisfies $\mathcal{D}[s_1, \ldots, s_k] \neq \mathcal{D}[s'_1, \ldots, s'_k]$ is at most $k(k-1)$. This proves the lemma. □

Lemma 9. *Suppose $M = (Q, q_0, \Sigma, \Delta, R)$ to be an MTT realizing* twist. *Let $\bar{n} \in T_\Sigma$. For all $N \subseteq N^{\bar{n}}_\Lambda$, there exist sets N' and D satisfying the following conditions:*

- *$N' \subseteq N^{\bar{n}}_\Lambda$ such that $|N'| \leq (m+1)|Q|$, where m is the maximum number such that $\bar{m}$ is contained in N.*
- *$D \subseteq O_{\bar{n}}$ such that $|D| \leq K(K-1)|N|$, where K is the maximum rank of states in Q.*
- *$\bigcup\{m{\downarrow}\mid m \in N\} = \bigcup\{n'{\downarrow}\mid n' \in N'\} \cup D$.*

Proof. Let $rep : Q \times T_\Sigma \to N$ be any partial function that $rep(q, \bar{m})$ returns an element of N whose root is $\langle q, \bar{m}\rangle$. We extend rep to $N^{\bar{n}}_\Lambda$ and take N' as $\{rep(n) \mid n \in N\}$. Since the size of the domain of rep is less than or equal to $(n+1)|Q|$, we have $|N'| \leq (n+1)|Q|$. Here, we take D as

$$\begin{aligned} D &= \bigcup\{n{\downarrow}\mid n \in N\} \setminus \bigcup\{n'{\downarrow}\mid n' \in N'\} \\ &= \bigcup\{n{\downarrow}\mid n \in N\} \setminus \bigcup\{rep(n){\downarrow}\mid n \in N\} \\ &\subseteq \bigcup\{(n{\downarrow} \setminus rep(n){\downarrow}) \mid n \in N\} \end{aligned}$$

By Lemma 8, we have $|n{\downarrow} \setminus rep(n){\downarrow}\ | \leq K(K-1)$. Hence, the size of D is less than or equal to $K(K-1)|N|$. The last equation in the statement is satisfied by the definition of D. □

The following lemma then proves that, at each step, the set of normal forms and the set of remainders produced there both have polynomial sizes.

Lemma 10. *Suppose $M = (Q, q_0, \Sigma, \Delta, R)$ to be an MTT realizing* twist. *Let $\bar{n}, \bar{m} \in T_\Sigma$. S be a set of trees in $N^{\bar{n}}_\Lambda$ and each member of S has the form $\langle q, \bar{m}\rangle(\cdots)$ for some $q \in Q$. Then there is sets $S_{D'} \subseteq O_{\bar{n}}$ and $S_{N'} \subseteq N^{\bar{n}}_\Lambda$ whose each member has the form $\langle q, \overline{m-1}\rangle(\cdots)$ for some $q \in Q$, such that*

- *$\bigcup\{n{\downarrow}\mid n \in S\} \subseteq \bigcup\{n'{\downarrow}\mid n' \in S_{N'}\} \cup S_D$*
- *$|S_{N'}| \leq m|Q|$*
- *$|S_{D'}| \leq (1 + K(K-1))H|S||R|$*

where H is the maximum number of subtrees contained in the right hand side of rules in R.

Proof. Let S_x be $\{t \mid s \Rightarrow t, s \in S\}$. Then we have $|S_x| \leq |R||S|$, and by definition of $\downarrow$, we have $\bigcup\{s{\downarrow}\mid s \in S\} = \bigcup\{s{\downarrow}\mid s \in S_x\}$.

Let $S_W = \{w(s) \mid s \in S_x, s{\downarrow} \neq \emptyset\}$ where w is the function whose existence is proved in Lemma 5. Then we have $S_W \subseteq W^{\bar{n}}_\Lambda$ and $\bigcup\{s{\downarrow}\mid s \in S_x\} = \bigcup\{s{\downarrow}\mid s \in S_W\}$ by the lemma. Since the size $|S_W|$ is less than or equal to $|S_x|$, we have $|S_W| \leq |S||R|$.

Let $S_Q = \bigcup\{N_Q(s) \mid s \in S_W\}$ and $S_V = \bigcup\{N_V(s) \mid s \in S_W\}$ where N_Q and N_V are the functions in Lemma 7. Then by the lemma we have $S_Q \subseteq N^{\bar{n}}_\Lambda$, $S_V \subseteq T_\Delta$, and $\bigcup\{s{\downarrow}\mid s \in S_W\} \subseteq \bigcup\{v{\downarrow}\mid v \in S_Q\} \cup S_V$. The sizes of the sets is, by the lemma, $|S_Q| \leq H|S||R|$ and $|S_V| \leq H|S||R|$.

Finally, let $S_{N'}$ and S_D be the sets that the existence of them is assured by Lemma 9 by taking S_Q as N. Then by the lemma we have $|S_{N'}| \leq m|Q|$, $|S_D| \leq K(K-1)H|S||R|$, and $\bigcup\{v{\downarrow}\mid v \in S_Q\} = \bigcup\{s{\downarrow}\mid s \in S_{N'}\} \cup S_D$.

Note that all procedure calls in trees in S_x must have the form $\langle q, \overline{m-1}\rangle(\cdots)$, by the definition of $\Rightarrow$, which also holds for trees in $S_{N'}$, by the construction of all preceding lemmata (for the case $m = 0$, the set S_x does not contain any procedure calls, and thus S_Q, S'_N, and S_D becomes empty). Let $S_{D'} = (S_V \cap O_{\bar{n}}) \cup S_D$. Then by $S_{N'}$ and $S_{D'}$, all conditions in the statement are satisfied. □

Putting the preceding lemmata in this section altogether, we obtain the following corollary, which poses a polynomial upper

bound on the size of the output language of an MTT that realizes twist.

Cor 1. *Suppose* $M = (Q, q_0, \Sigma, \Delta, R)$ *to be an MTT realizing* twist. *Let* $\bar{n} \in T_\Sigma$. *Then*

$$|\langle q_0, \bar{n}\rangle()\downarrow| \leq (n+1)^2(1 + K(K-1))H|Q||R|$$

Proof. Let $S_n = \{\langle q_0, \bar{n}\rangle()\} \subseteq N_\Lambda^{\bar{n}}$. Let S_i and D_i be the sets whose existence is assured by the preceding lemma by taking S_{i+1} as S. Then by the lemma,

$$\begin{aligned}
|\langle q_0, \bar{n}\rangle()\downarrow| \leq & |\bigcup_{i=0}^{n} D_{i-1}| \\
\leq & \sum_{i=0}^{n}(1 + K(K-1))H|S_i||R| \\
\leq & \sum_{i=0}^{n}(1 + K(K-1))H(i+1)|Q||R| \\
\leq & \sum_{i=0}^{n}(1 + K(K-1))H(n+1)|Q||R| \\
= & (n+1)^2(1 + K(K-1))H|Q||R|
\end{aligned}$$

□

The corollary, however, leads to a contradiction, recalling that the size of twist's output set is actually exponential. Hence there exists no MTT realizing twist, proving the goal proposition of this section.

Proof of Prop 4. In order for an MTT M to realize the twist translation, it must be that $|\langle q_0, \bar{n}\rangle()\downarrow| = |O_{\bar{n}}|$. However, $|O_{\bar{n}}| = 2^n$. The corollary implies that an inequation $2^n \leq (n+1)^2(1+K(K-1))H|Q||R|$, which does not hold when we take sufficiently large n. □

5. Future Work

This work is the very first step of our investigation on mr-MTTs and many interesting questions concerning the expressiveness are still left unanswered. For example, what is the relationship between mr-MTTs and *composition* of conventional MTTs? We have not yet encountered any translation that can be realized by a mr-MTT but not by a composition of MTTs (the `twist` translation, which we used as a counterexample not realizable by an MTT, is realized by the composition of two MTTs: one MTT that nondeterministically generates a sequence of `a`s and `b`s, and the other MTT that generates its reversal deterministically). But it is still open whether inclusion of mr-MTT by MTT^n holds or not. Another example of open questions is whether mr-MTTs create a proper hierarchy by dimension, i.e., whether there is any translation that can be expressed by an mr-MTT with dimension d but not by mr-MTTs with a smaller dimension.

References

[1] James Clark. XSL Transformations (XSLT), 1999. `http://www.w3.org/TR/xslt`.

[2] Joost Engelfriet. Bottom-up and top-down tree transformations - a comparison. *Mathematical Systems Theory*, 9(3):198–231, 1975.

[3] Joost Engelfriet and Sebastian Maneth. A comparison of pebble tree transducers with macro tree transducers. *Acta Informatica*, 39(9):613–698, 2003.

[4] Joost Engelfriet and Heiko Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71–146, 1985.

[5] Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In *Programming Language Design and Implementation (PLDI)*, pages 237–247, 1993.

[6] Alain Frisch and Haruo Hosoya. Towards practial typechecking for macro tree transducers. In *Database Programming Languages (DBPL)*, 2007.

[7] Zoltan Fülöp and H. Vogler. *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. Springer-Verlag, 1998.

[8] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[9] Haruo Hosoya and Benjamin C. Pierce. XDuce: A typed XML processing language. *ACM Transactions on Internet Technology*, 3(2):117–148, 2003.

[10] Sebastian Maneth, Alexandru Berlea, Thomas Perst, and Helmut Seidl. XML type checking with macro tree transducers. In *Symposium on Principles of Database Systems (PODS)*, pages 283–294, 2005.

[11] Sebastian Maneth, Thomas Perst, and Helmut Seidl. Exact XML type checking in polynomial time. In *International Conference on Database Theory (ICDT)*, pages 254–268, 2007.

[12] Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. In *Symposium on Principles of Database Systems (PODS)*, pages 11–22, 2000.

[13] Keisuke Nakano and Shin-Cheng Mu. A pushdown machine for recursive XML processing. In *Asian Symposium on Programming Language and Systems (APLAS)*, pages 340–356, 2006.

[14] Thomas Perst and Helmut Seidl. Macro forest transducers. *Information Processing Letters*, 89(3):141–149, 2004.

[15] Akihiko Tozawa. Towards static type checking for XSLT. In *ACM Symposium on Document Engineering (DocEng)*, 2001.

[16] Akihiko Tozawa. XML type checking using high-level tree transducer. In *Functional and Logic Programming (FLOPS)*, pages 81–96, 2006.

Acknowledgments

We are grateful to Sebastian Maneth for valuable comments and suggestions. We would also like to thank Janis Voigtläender for his detailed and thorough comments. This work was partly supported by Japan Society for the Promotion of Science.