COMPLEXITY AND EXPRESSIVENESS OF MODELS OF XML TRANSLATIONS

The University of Tokyo

December 17, 2008

Kazuhiro Inaba

ABSTRACT

XML has become widely used in computer industry, and the importance of static analysis and verification of applications manipulating XML documents is increasing. For analyzing or proving any properties on XML manipulating programs, it is essential to have some *model* with theoretically well-defined semantics. After a long history of researches on models of tree-to-tree translations, a recent trend is to regard macro tree transducers (mtts) as a standard model for XML manipulation. Mtts are known to cover tree translations expressible by other models such as attributed tree transducers, MSO-definable tree translations, or pebble tree transducers, and the high expressiveness allows representing a vast range of practical XML translations. Yet, they ensure various good properties such as exact typechecking, streaming, decidable emptiness, and so on. Nevertheless, mtts still lack some properties desired for modeling XML translations. In particular, (1) mtts have poor closure properties on composition and (2) computational complexities for many problems on mtts are still unknown. A consequence of the first point is that, for example, an mtt composed with even a very simple pre- or post- tree translation cannot be represented by a single mtt. The lack of the compositionality implies difficulty of modular modeling; even if we could construct a model for each smaller subpart of a program separately, it is in general impossible to compose them up to obtain the model for the whole program. For the second point, the decidability is proved for many problems on mtts or their compositions, while the complexities and concrete algorithms solving the problems have been left open. This makes it difficult to estimate the computational hardness of each verification problem. One example of such a case is the membership problem of output languages, i.e., the problem determining whether a tree is a valid output with respect to the translation and a given input regular type. Although the problem is essential for, e.g., verifying that a program never generates 'wrong' outputs, its complexity had not been analyzed.

The goal of the thesis is to improve these shortcomings of mtts. First, to address the compositionality issue, a new model of tree-to-tree translation called *multi-return macro tree transducer (mr-mtt)* is introduced. As its name shows, an mr-mtt is an mtt extended with the capability of returning multiple tree fragments simultaneously, in contrast to an mtt that can return only one. We show that mr-mtts are closed under pre- and post- compositions with arbitrary deterministic total top-down tree transducers, and therefore enable modular modeling of tree-translations. We also show that mr-mtts are strictly more expressive than mtts, which at the same time formally proves as a corollary the folklore conjecture that mtts are not closed under post-compositions with top-down tree transducers.

Second, the thesis investigates complexity on compositions of mtts. We show that the data-complexity of the membership problem of output languages is in the class DSPACE(n) and is NP-complete. The crucial lemma of the proof is that any composition of finite number of mtts can be transformed into so-called *garbage-free forms*, meaning that any subtrees of intermediate results are actually used for generating the final output. Besides the complexity of output languages, we give another application of the garbage-free form: the complexity is shown for the 'translation membership' problem which determines whether a given pair of trees is an input-output pair of the translation.

Acknowledgments

First of all, I thank Professor Sebastian Maneth, who has been my supervisor at National ICT Australia. He made me aware of various interesting topics around tree transducers. The thesis could not be completed without his helpful and insightful advice. I would also like to thank him for giving me a truly warm welcome during my one-year visit to Sydney.

I am grateful to my supervisor at the University of Tokyo, Professor Haruo Hosoya, who introduced me to the research on XML. He also encouraged me to study abroad. The invaluable experience greatly changed my life and my perspective.

Finally, I would like to thank my parents Takashi and Yoko for bringing me up till today. My early interest on computer science owes much to them who gave me a programmable computer in my childhood.

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

Kazuhiro Inaba,
December 17, 2008

Contents

List of Figures

Chapter 1

Introduction

1.1 XML and Verifications

XML (Extensible Markup Language) [BPSMM00] is a language for representing trees. It has now established its position as the standard format for describing and exchanging structured data on computers. More and more applications are adopting XML; they store their own data as XML, or they use a standard format based on XML for communicating with other applications.

As XML processing programs are getting more and more popular, the importance of verification of such programs is increasing. Suppose we have a program that converts a summary document of a website written in RSS format (which is a particular XML-based format) into an XHTML document (again an XML-based format), which is a suitable format to be displayed in a human-readable way. How can we make sure that the program always generates a correct XHTML document, or that the program never gets into errors by any valid input RSS document? More generally, how should we verify the correctness of XML processing programs in terms of the input and output XML structures?

1.2 Models of Tree Translations

Such a question has invoked an active body of research on the formal study of tree transformation models. Note that XML processing is essentially a process of *tree translation*. We query on input trees represented by XML documents and obtain necessary subparts (i.e., subtrees) of the input tree depending on the purpose, and convert them into another tree that are finally serialized into output XML documents. Many models of tree translations—top-down tree transducers [Rou70, Tha70], bottom-

up tree transducers [Tha73], attributed tree transducers [Fül81, Knu68], macro tree transducers [Eng80, CFZ82], MSO-definable tree translations [Cou94], high-level tree transducers [EV88], pebble tree transducers [MSV03], etc—have been introduced in the last few decades. Each of these models can be regarded as a kind of programming language, whose expressive power is restricted in some way so that typical verification problems on the model become decidable. Such verification problems include, e.g., decision problem for totality or emptiness of the translation, membership test for the range or the domain, finiteness of the domain, or exact typechecking that determines the translation always converts any tree in a given input set into a tree in a given output set. Note that the verification problems are basically undecidable for Turing-complete programming languages. Thus, a typical use of those formal models is to approximate some 'real' program, and then the decision procedures for verification problems are applied.

Various properties are desired for models of translations. In particular, the following three perspectives are important: (1) *decidability*—verification problems are desired to remain decidable in the model, (2) *expressiveness*—expressive models allow to obtain more accurate (or sometimes, exact) approximation of target programs in a real programming language, which results in a more accurate verification, and (3) *constructivity*—it is better if there exists a simple method for constructing model representations from target programs. From these points of view, a recent trend is to regard macro tree transducers (mtts) as a standard model for XML manipulation among the various models. They are known to subsume all the models mentioned above in expressiveness [EV85, EM99, EM03a], and moreover, they are shown to be able to capture the expressive fragment of XML programming languages popular in practice, such as XSLT or XML-QL [MSV03, EM03a]. Also, they naturally support a nondeterministic translation model, which often yields better approximation of real programs than a deterministic model (viz. a complicated if-then-else expression; it is translated into an mtt that nondeterministically chooses one of the conditional branches). Yet, verification problems remain decidable for mtts, despite of their high expressive power [EV85, DE98, FH07].

Nevertheless, mtts still lack some properties desired for modeling XML translations. In particular, (1) nondeterministic mtts have poor closure properties on composition and (2) computational complexities for many problems on mtts are still unknown. The goal of the thesis is to improve these shortcomings of mtts, which will be discussed in more detail in the subsequent sections.

$$q_0(\mathtt{s}(x)) \to \mathtt{a}(q_1(x, \mathtt{A}(\mathtt{E}))) \qquad q_1(\mathtt{s}(x), y) \to \mathtt{a}(q_1(x, \mathtt{A}(y)))$$
$$q_0(\mathtt{s}(x)) \to \mathtt{b}(q_1(x, \mathtt{B}(\mathtt{E}))) \qquad q_1(\mathtt{s}(x), y) \to \mathtt{b}(q_1(x, \mathtt{B}(y)))$$
$$q_0(\mathtt{z}) \to \mathtt{e} \qquad q_1(\mathtt{z}, y) \to y$$

Figure 1.1: An mtt M_{pal} generating palindromic monadic trees

1.3 A Composable Model—Multi-Return Macro Tree Transducers

In general, an mtt composed with even very simple pre- or post- tree translation cannot be represented by a single mtt. The lack of the compositionality implies difficulty of modular modeling; to model some real program by an mtt, we always have to construct the mtt as a whole, instead of modeling each smaller subpart of the program separately and then composing them up. This makes it quite difficult to automate the verification process of real programs.

Before explaining the issue, let us first briefly introduce the 'macro tree transducer' model by an example. Fig. 1.1 is an mtt M_{pal} representing a nondeterministic translation 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 $\delta_1(\delta_2(\cdots\delta_n(\Delta_n(\cdots\Delta_1(\mathtt{E})\cdots))\cdots)$ where n is the number of $\mathtt{s}$ nodes in the input, each δ_i is either $\mathtt{a}$ or $\mathtt{b}$ and each Δ_i is the capitalized letter of δ_i. In other words, given an input tree of height n, the mtt nondeterministically generates any one of the palindromes over symbols $\{\mathtt{a}, \mathtt{b}\}$ of length $2n$, with the latter half being capitalized. The rules of mtts can be naturally read as an ordinary functional programming language. An mtt processes the input tree in top-down direction, starting in its initial state (q_0 in this case) at the root node. Depending on its state and the label of the current input node, it selects a rule; if more than one rule matches (e.g., as in the case of the state q_0 with the label $\mathtt{s}$ that has two matching rules), either one of the rules is nondeterministically chosen. Then, according to the selected rule, the mtt produces an output subtree which possibly contains recursive state calls to children of the current node. Each state is allowed to receive additional parameters in addition to the current input node. The number of such "accumulating parameters" is fixed for each state of the transducer. For example, in our example, the state q_1 has one parameter y. The initial state has zero parameters, because we are interested in tree-to-tree, not (tuple of trees)-to-tree translations. It is well-known that accumulating parameters

$$
\begin{array}{lll}
p_0(\mathtt{a}(x)) \to \mathtt{root}(\mathtt{a}(p_1(x)), p_2(x)) & p_1(\mathtt{a}(x)) \to \mathtt{a}(p_1(x)) & p_2(\mathtt{a}(x)) \to p_2(x) \\
p_0(\mathtt{b}(x)) \to \mathtt{root}(\mathtt{b}(p_1(x)), p_2(x)) & p_1(\mathtt{b}(x)) \to \mathtt{b}(p_1(x)) & p_2(\mathtt{b}(x)) \to p_2(x) \\
p_0(\mathtt{A}(x)) \to \mathtt{root}(\mathtt{e}, \mathtt{A}(p_2(x))) & p_1(\mathtt{A}(x)) \to \mathtt{e} & p_2(\mathtt{A}(x)) \to \mathtt{A}(p_2(x)) \\
p_0(\mathtt{B}(x)) \to \mathtt{root}(\mathtt{e}, \mathtt{B}(p_2(x))) & p_1(\mathtt{B}(x)) \to \mathtt{e} & p_2(\mathtt{B}(x)) \to \mathtt{B}(p_2(x)) \\
p_0(\mathtt{E}) \to \mathtt{root}(\mathtt{e}, \mathtt{E}) & p_1(\mathtt{E}) \to \mathtt{e} & p_2(\mathtt{E}) \to \mathtt{E}
\end{array}
$$

Figure 1.2: An mtt M_{spl} splitting the output trees from M_{pal}

add an expressive power. Mtts realize strictly more translations than top-down tree transducers (mtts with no parameters); for instance, top-down tree transducers have at most exponential size increase while mtts can have double-exponential increase.

Let us introduce another mtt M_{spl} in Fig. 1.2, which is actually a top-down tree transducer, having no parameters. The transducer M_{spl} takes a tree of form $\delta_1(\delta_2(\cdots\delta_n(\Delta_1(\cdots\Delta_m(\mathtt{E})\cdots))\cdots))$, and splits them to two branches: the left branch becomes $\delta_1(\delta_2(\cdots\delta_n(\mathtt{e})\cdots))$ and the right branch becomes $\Delta_1(\Delta_2(\cdots\Delta_m(\mathtt{E})\cdots))$. For example, the input tree $\mathtt{a}(\mathtt{b}(\mathtt{B}(\mathtt{A}(\mathtt{E}))))$ is deterministically translated to the output tree $\mathtt{root}(\mathtt{a}(\mathtt{b}(\mathtt{e})), \mathtt{B}(\mathtt{A}(\mathtt{E})))$.

Now, suppose we want to model by an mtt some program consisting of three functions: one function *pal* modeled by M_{pal} (recall that, nondeterminism is typically used for approximating, e.g., a complicated if-then-else expression), another function *spl* modeled by M_{spl}, and a function *main* defined to first apply *pal* to the input and then apply *spl* to the output from *pal*. The problem is that there is no known method for composing nondeterministic mtts, even for the composition with simpler translations like a deterministic top-down tree transducer in this example! In fact, the translation M_{pal} followed by M_{spl} can be proved *not* to be representable by a single mtt. However, even if it were the case that a composition were representable by a single mtt, no systematic way to construct the composed mtt is known; some ingenious idea depending on the particular pair of transducers would be required.

To address the compositionality issue, we propose a new model of tree-to-tree translation called *multi-return macro tree transducers (mr-mtts)*. As its name shows, an mr-mtt is an mtt extended with the capability of each state returning multiple tree fragments simultaneously, in contrast to an mtt that can return only one tree. Compared to mtts, which can propagate multiple trees in a top-down direction by accumulating parameters but not in a bottom-up direction, mr-mtts may be regarded

$$
\begin{aligned}
q_0(\mathtt{s}(x)) &\rightarrow \mathsf{let}\ (z_1, z_2) \leftarrow q_1(x, \mathtt{A}(\mathtt{E}))\ \mathsf{in}\ \mathtt{root}(\mathtt{a}(z_1), z_2)\\
q_0(\mathtt{s}(x)) &\rightarrow \mathsf{let}\ (z_1, z_2) \leftarrow q_1(x, \mathtt{B}(\mathtt{E}))\ \mathsf{in}\ \mathtt{root}(\mathtt{b}(z_1), z_2)\\
q_0(\mathtt{z}) &\rightarrow \mathtt{root}(\mathtt{e}, \mathtt{E})\\
q_1(\mathtt{s}(x), y_1) &\rightarrow \mathsf{let}\ (z_1, z_2) \leftarrow q_1(x, \mathtt{A}(y_1))\ \mathsf{in}\ (\mathtt{a}(z_1), z_2)\\
q_1(\mathtt{s}(x), y_1) &\rightarrow \mathsf{let}\ (z_1, z_2) \leftarrow q_1(x, \mathtt{B}(y_1))\ \mathsf{in}\ (\mathtt{b}(z_1), z_2)\\
q_1(\mathtt{z}, y_1) &\rightarrow (\mathtt{e}, y_1)
\end{aligned}
$$

Figure 1.3: An mr-mtt representing the composition of M_{pal} and M_{spl}

as a model attaining symmetry between top-down and bottom-up propagation of information. Fig. 1.3 is an example of an mr-mtt. The 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.

The reader should be able to verify that this single mr-mtt realizes exactly the composition of M_{pal} with M_{spl}. This immediately implies one fact; mr-mtts are more expressive than normal mtts. Besides an increase in expressive power, mr-mtts are shown to have better closure properties than mtts: not only for the particular combination of M_{pal} and M_{spl} but in general they are closed under left and right composition with total deterministic top-down tree transducers ($\mathrm{D_tTs}$). This is rather surprising, because ordinary call-by-value mtts are not closed under composition with $\mathrm{D_tTs}$. The latter was already shown in [EV85] for the case of left-composition. For the case of right-composition, it is proved in this paper (using the composition of M_{pal} and M_{spl}, which we call *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 $\mathrm{D_tT}$ (this remained open in [EV85]). Thus, the two main classes of mtts, call-by-value and call-by-name are both not closed under right-composition with $\mathrm{D_tT}$, while call-by-value multi-return mtts are closed.

1.4 Complexities on Macro Tree Transducers

The second issue we address is that the computational complexity of verification problems are left unknown; although the decidability is proved for many problems on mtts and their compositions, the complexities and concrete algorithms solving the problems have been left open.

In the thesis, we mainly study the complexity of the membership problem for the output (string or tree) languages of the class of translations realized by compositions of mtts (the mtt-hierarchy). Note that we do not explicitly distinguish between string or tree output languages here, because the translation "yield" which turns a tree into its frontier string (seen as a monadic tree) is a particular simple macro tree translation itself and hence the corresponding classes have the same complexity. Small subclasses of our class of languages considered here are the IO-macro languages (or, equivalently, the yields of context-free-tree languages under IO-derivation) and the string languages generated by attribute grammars. Both of these classes are LOG(CFL)-complete by [Asv81] and [Eng86], respectively. Another subclass of our class is OI-macro languages, which are equivalent to the indexed languages [Aho68], by [Fis68]. This class is known to be NP-complete [Rou73]. Hence, our class is NP-hard too (even already at compositions of two mtts). Our first main result is that output languages of the mtt-hierarchy are NP-complete; thus, the complexity remains in NP when going from indexed languages to the full mtt-hierarchy. In terms of space complexity, languages generated by compositions of top-down tree transducers (mtts without accumulating parameters) are known to be in DSPACE(n) [Bak78]. This result was generalized in [Man02] to compositions of *total deterministic* mtts. Our second main result is that output languages of the mtt-hierarchy (generated by compositions of *nondeterministic* mtts) with regular tree languages as inputs still remains in DSPACE(n) and thus are context-sensitive. The approach of our proof can be seen as a generalization of the proofs in [Bak78] and [Man02]; moreover, we make essential use of the idea of compressed representation of backtracking information, used by Aho in [Aho68] for showing that the indexed languages are in NSPACE(n).

We show the NP and the DSPACE(n) upperbound by giving a concrete algorithm testing the membership of output languages. Our approach might look quite simple at the first sight: given a composition $\tau = \tau_1 ; \tau_2 ; \cdots ; \tau_n$ of mtts and a candidate output tree t, we iteratively guess the intermediate result of the previous stage. That is, we first seek s_n such that $(s_n, t) \in \tau_n$, and then we seek s_{n-1} such that $(s_{n-1}, s_n) \in \tau_{n-1}$, and so on. If we eventually reach the very first input tree s_1 (which should satisfy

$(s_1, s_2) \in \tau_1$, of course), then it tells us that t is indeed in the range of the composition τ. If we cannot find any such sequence of intermediate results, then it implies that t is not the member of the range.

To carry out this approach, however, we have to solve two big challenges: (1) how to find the previous intermediate result of a single mtt τ_i effectively, and (2) how to limit the size of the intermediate results s_i. The first issue is actually reduced to the "translation membership" problem, which asks whether a given input/output pair of trees is whether or not in τ_i. We identify the complexities of translation membership for several classes of mtts. For call-by-value (IO) mtts, we show the problem is in PTIME by using a technique based on inverse type inference of mtts [EV85]. For linear call-by-name (OI) mtts, we show NP-completeness and DSPACE(n) space complexity of the translation membership problem. The challenge here is the space complexity; we use a compressed representation of M_{pal}'s output trees for input s, inspired by [MB04], and then check if t is contained in the output set by using a recursive procedure in which nodes needed for backtracking are compressed using a trie, similar to Aho's compression of index strings in [Aho68].

For the second point, note that each τ_i may be a translation that deletes much of its input trees and generate a very small output. Hence, it can be the case $|s_n| \gg |t|$, i.e, there is no bound for the size $|s_n|$ with respect to $|t|$. If such a blow-up happens, no matter how efficiently we guess the tree $|s_n|$, it inherently takes $O(|s_n|)$ time, which means that we cannot pose any upperbound of the complexity with respect to $|t|$. Our key lemma is to avoid this issue. We show that for any composition sequence $\tau_1 ; \tau_2 ; \cdots ; \tau_n$ of mtts, there exists an equivalent *garbage-free* composition $\tau_0 ; \tau'_1 ; \tau'_2 ; \cdots ; \tau'_{2n}$ of mtts, meaning that each τ'_i do not delete much of its input, i.e., every output tree t has a corresponding input tree of size only linearly larger than $|t|$. In fact, also the initial input tree s_1 can be changed into a smaller tree s'_1 of size linear in $|t|$, for which $\tau(s'_1) = \tau(s_1)$. Once we have established the "garbage-free" form, together with the complexity results for the translation membership problem, we obtain the NP and DSPACE(n) upperbound by the "simple" approach explained above.

Besides the membership problem of the output languages, we believe that our garbage-free form is a useful tool that allows us to derive in a simple approach the complexity results for other verification problems. As an example, we apply our method and yield the complexity of the translation membership problem for arbitrary mtts in call-by-name semantics and their compositions.

1.5 Our Contributions

The outline and contributions of this thesis are as follows:

- In Section 2, we introduce *multi-return macro tree transducers (mr-mtts)*, which is a new model of tree translation that enables better compositionality than normal macro tree transducers (mtts). We prove closure properties of mr-mtts under left- and right- composition with total deterministic top-down tree transducers. Also, we characterize the class of translations realized by mr-mtts in terms of normal mtts.

- In Section 3, we prove that the "multi-return" facility does add power in terms of expressiveness. This is shown by exhibiting a counterexample that can be expressed by an mr-mtt but not by 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 [HU79].

  Besides the strict inclusion between the class of mr-mtts and mtts, the inexpressibility is applied for showing several other results that have been conjectured but left open in the literature. By using the counterexample, we formally show that (1) the class of mtt-realizable translations are *not* closed under right-composition with total deterministic top-down tree transducers, and (2) the class of mtt-realizable linear-size increase translations is not closed under composition within themselves.

- In Section 4, the complexity of the "translation membership" problem for a single mtt is considered. We prove that in call-by-value semantics the problem is in PTIME, and in call-by-name semantics, it is NP-complete and in DSPACE(n) for linear mtts. We generalize the PTIME result for several extensions of call-by-value mtts (including mr-mtts) and restrictions of call-by-name mtts. Those results are particularly useful as a basis for showing the complexity of other verification problems, as exemplified in the next section.

- In Section 5, we show that the membership problem of output languages for compositions of mtts is in the complexity class DSPACE(n) and is NP-complete. The crucial lemma of the proof is that any sequence of mtts can be transformed into so-called a *garbage-free form*, meaning that any subtrees of intermediate results are actually used for generating the final output. The garbage-free form

is also applied to derive the complexity of the translation membership problem for general non-linear mtts and their compositions.

Then, Section 6 concludes the thesis and suggests possible direction for future research.

Several parts of this thesis have been published in the proceedings of conferences and workshops. Multi-return macro tree transducers and the results on their expressiveness were presented in [IH08]. The compositionality of mr-mtts was presented in [IHM08]. The complexity of translation membership and output language membership for mtts were first presented in [IM08], and more extensive investigation on the translation membership problem appeared in [IM09].

Chapter 2

Multi-Return Macro Tree Transducers

Macro tree transducers have been one of the most popular model for XML translations. Their high expressive power covers tree translations expressible by other models and several XML translation languages existing in practice, yet they have most of the important properties to be decidable, which implies that static validations can be carried out on mtts. However, mtts have very poor compositionality, in the sense that a composition of an mtt and another very simple translation may not be representable by an mtt. The lack of compositionality makes it difficult to approximate a 'real' program by an mtt. To remedy the shortcoming, in this chapter we introduce a mild extension of mtts, namely *multi-return mtts*, and show that they posses better closure properties under composition.

2.1 Trees and Translations

We denote by ϵ the *empty list*, i.e., a list of length 0, and by $l_1.l_2$ the concatenation of two lists l_1 and l_2. A list l is said to be a *prefix* of a list l' if there is a list l'' such that $l.l'' = l'$, and to be a *proper prefix* if $l'' \neq \epsilon$.

A set Σ with a mapping $\mathit{rank} : \Sigma \to \mathbb{N}$ is called a *ranked set*. We often write $\sigma^{(k)}$ to indicate that $\mathit{rank}(\sigma) = k$ and write $\Sigma^{(k)}$ to denote the subset of Σ of rank-k symbols. The *product* of a ranked set Σ and a set B is the ranked set $\Sigma \times B = \{\langle \sigma, b \rangle^{(k)} \mid \sigma \in \Sigma^{(k)}, b \in B\}$. Throughout the thesis, we fix the sets of input variables $X = \{x_1, x_2, \ldots\}$, parameters $Y = \{y_1, y_2, \ldots\}$, let-variables $Z = \{z_1, z_2, \ldots\}$, and holes $H = \{\Box_1, \Box_2, \ldots\}$, which are all of rank 0. We assume any other alphabet to be disjoint with X, Y, Z, and H. The set X_i is defined as $\{x_1, \ldots, x_i\}$, and Y_i, Z_i, and H_i are defined similarly. We sometimes use $\Box$ to denote $\Box_1$.

The set T_Σ of *trees* t over a ranked set Σ is defined by the BNF

$$t ::= \sigma(\overbrace{t, \ldots, t}^{k}) \qquad \text{for } \sigma \in \Sigma^{(k)}$$

We often omit parentheses for rank-0 and rank-1 symbols and write them as strings. For example, we write abcd instead of $\mathsf{a}(\mathsf{b}(\mathsf{c}(\mathsf{d}())))$. We recursively define the function *label* from $T_\Sigma \times \mathbb{N}^*$ to Σ as follows. For $\sigma \in \Sigma^{(k)}$, $k \geq 0$:

$$\begin{aligned} \mathit{label}(\sigma(t_1, \ldots, t_k), \epsilon) &= \sigma \\ \mathit{label}(\sigma(t_1, \ldots, t_k), i.\nu) &= \mathit{label}(t_i, \nu). \end{aligned}$$

Thus, the empty list ϵ denotes the root node and $\nu.i$ denotes the i-th child of ν. We define the set $\mathit{pos}(t) = \{\nu \in \mathbb{N}^* \mid \mathit{label}(t, \nu) \text{ is defined}\}$ and call each element of $\mathit{pos}(t)$ a *node* of t. We denote by $|t|$ the number of nodes in the tree t. For a node v of t, we let $t|_v$ denote the subtree of t rooted at the node v. For trees $t, t_1, \ldots, t_n \in T_\Sigma$ and $\sigma_1, \ldots, \sigma_n \in \Sigma^{(0)}$, we denote by $t\,[\sigma_1/t_1, \ldots, \sigma_n/t_n]$ (or sometimes $t[\vec{\sigma}/\vec{t}]$ for brevity) the simultaneous substitution of the σ_i by the t_i. For a ranked set Σ, 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 $\mathcal{C}[\Box/t]$.

Let Σ and Δ be ranked alphabets. A relation $\tau \subseteq T_\Sigma \times T_\Delta$ is called a *tree translation* (over Σ and Δ) or simply a translation. We write $\tau(s)$ to denote the set $\{t \mid (s,t) \in \tau\}$, and if τ is a function (i.e., $|\tau(s)| = 1$ for any s), we abuse the notation and use $\tau(s)$ to denote the unique output tree rather than the output singleton set. We define $\mathit{dom}(\tau) = \{a \mid \exists b : (a,b) \in \tau\}$ and $\mathit{range}(\tau) = \{b \mid \exists a : (a,b) \in \tau\}$. For two translations τ_1 and τ_2, their sequential composition $\tau_1\,;\tau_2$ ("τ_1 followed by τ_2") is the translation $\{(a,c) \mid \exists b : ((a,b) \in \tau_1, (b,c) \in \tau_2)\}$. For two classes T_1 and T_2 of translations, we define $T_1\,;T_2 = \{\tau_1\,;\tau_2 \mid \tau_1 \in T_1, \tau_2 \in T_2\}$. The k-fold composition of the class T of translations is denoted by T^k, and by T^* we mean $\bigcup_{k \geq 0} T^k$.

Relationship to XML Representation Our definition of trees is *ranked*, i.e., the number of child nodes is fixed for each node label. On the other hand, XML often deals with *unranked* nodes, for which the number of child nodes varies from nodes to nodes even if their labels are the same. Such unrankedness is usually handled through an encoding into binary node representation. In binary representation, the first child of each node is mapped to the first child of the corresponding node in the original unranked tree, and the second child of each node is mapped to the *next sibling* in the unranked representation. For example, the following XML document

```
<list>
  <item>A</item>  <item>B</item> <item>C</item>
</list>
```

is encoded to a binary tree $\texttt{list}(\texttt{item}(\texttt{A}, \texttt{item}(\texttt{B}, \texttt{item}(\texttt{C}, \bot))), \bot)$ where $\bot$ is a special rank-0 symbol meaning that there are no more siblings. Note that the first `item` node has two children: the first child `A` which is the first child also in the original XML, and the second child $\texttt{item}(\texttt{B}, \cdots)$ which corresponds to the next sibling of the first `item` node in the original XML.

In this thesis, with the encoding in mind, we focus on ranked trees as defined above, rather than directly dealing with the concrete unranked XML documents.

2.2 Macro Tree Transducers

Macro tree transducers (mtts) are introduced in [Eng80, CFZ82] as a combination of top-down tree transducers [Rou70, Tha70] and macro grammars [Fis68]. An mtt is a finite-state machine that processes the input tree in top-down direction (like a top-down tree transducer), keeping context information by using accumulating parameters (like a macro grammar). Let us review the formal definition of macro tree transducers.

Definition 2.1. A *macro tree transducer (mtt)* M is a tuple $(Q, \Sigma, \Delta, q_0, R)$, where Q is the ranked alphabet of *states*, Σ and Δ are the *input* and *output* alphabets, $q_0 \in Q^{(0)}$ is the *initial state*, and R is the finite set of *rules* of the form

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

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

Note that, for technical simplicity and emphasizing the special role of the first parameter, we adopt the syntax that differs from the one we have used in the Introduction.

Big-Step Semantics A state q of a macro tree transducer can be regarded as a (nondeterministic) *function* in functional programming languages. Depending on the order of evaluation, two different semantics can be considered: call-by-value (or inside-out, IO) and call-by-name (or, outside-in, OI). For a tree $u \in T_{\Delta \cup (Q \times T_\Sigma) \cup Y}$, its IO-semantics $[\![u]\!]^M_{\mathrm{IO},\Gamma} \subseteq T_\Delta$ with respect to M under the *environment* $\Gamma : Y \to T_\Delta$ is

inductively defined as follows.

$$\begin{aligned}
[\![y_i]\!]^M_{\mathrm{IO},\Gamma} &= \{\Gamma(y_i)\} \\
[\![\delta(u_1,\ldots,u_n)]\!]^M_{\mathrm{IO},\Gamma} &= \{\delta(t_1,\ldots,t_n) \mid t_i \in [\![u_i]\!]^M_{\mathrm{IO},\Gamma} \text{ for all } i\} \\
[\![\langle q,\sigma(s_1,\ldots,s_k)\rangle(u_1,\ldots,u_m)]\!]^M_{\mathrm{IO},\Gamma} &= \\
&\bigcup \left\{ [\![r[x_1/s_1,\ldots,x_k/s_k]]\!]^M_{\mathrm{IO},\Gamma'} \,\middle|\, r \in R_{q,\sigma}, \Gamma'(y_i) \in [\![u_i]\!]^M_{\mathrm{IO},\Gamma} \text{ for all } i \right\}
\end{aligned}$$

The OI-semantics $[\![u]\!]^M_{\mathrm{OI},\Gamma} \subseteq T_\Delta$ is defined similarly, but under the environment Γ of type $Y \to 2^{T_\Delta}$, as follows.

$$\begin{aligned}
[\![y_i]\!]^M_{\mathrm{OI},\Gamma} &= \Gamma(y_i) \\
[\![\delta(u_1,\ldots,u_n)]\!]^M_{\mathrm{OI},\Gamma} &= \{\delta(t_1,\ldots,t_n) \mid t_i \in [\![u_i]\!]^M_{\mathrm{OI},\Gamma} \text{ for all } i\} \\
[\![\langle q,\sigma(s_1,\ldots,s_k)\rangle(u_1,\ldots,u_m)]\!]^M_{\mathrm{OI},\Gamma} &= \\
&\bigcup \left\{ [\![r[x_1/s_1,\ldots,x_k/s_k]]\!]^M_{\mathrm{OI},\Gamma'} \,\middle|\, r \in R_{q,\sigma}, \Gamma'(y_i) = [\![u_i]\!]^M_{\mathrm{OI},\Gamma} \text{ for all } i \right\}
\end{aligned}$$

We sometimes omit ^{M} if it is clear from the context, and $_\Gamma$ if it is an empty environment, i.e., $\mathit{range}(\Gamma) = \emptyset$.

In IO-semantics, each argument u_i of a function call $\langle q,\sigma(s_1,\ldots,s_k)\rangle(u_1,\ldots,u_m)$ is evaluated before entering the function body r, and the corresponding parameter variable y_i is bound to a tree in $[\![u_i]\!]^M_{\mathrm{IO},\Gamma}$. Although the choice of the tree from $[\![u_i]\!]^M_{\mathrm{IO},\Gamma}$ is nondeterministic, once fixed, each occurrence of y_i is always evaluated to the same tree $\Gamma(y_i)$. On the other hand, in OI-semantics, arguments are not evaluated at a function-call site; each parameter y_i is assigned a *set* of trees $[\![u_i]\!]^M_{\mathrm{OI},\Gamma}$ that remembers all possible values derivable from u_i. At each occurrence of y_i in function bodies, a tree in $\Gamma(y_i)$ is nondeterministically chosen. Let us illustrate the difference by the following mtt M:

$$\begin{aligned}
\langle q,\mathsf{a}\rangle(y_1) &\to \mathsf{b}(y_1,y_1) \\
\langle q,\mathsf{b}\rangle &\to \mathsf{c} \\
\langle q,\mathsf{b}\rangle &\to \mathsf{d}.
\end{aligned}$$

In IO-semantics, the set $[\![\langle q,\mathsf{a}\rangle(\langle q,\mathsf{b}\rangle)]\!]^M_{\mathrm{IO}}$ consists of two trees, namely,

$$[\![\langle q,\mathsf{a}\rangle(\langle q,\mathsf{b}\rangle)]\!]^M_{\mathrm{IO}} = [\![\mathsf{b}(y_1,y_1)]\!]^M_{\mathrm{IO},(y_1\mapsto\mathsf{c})} \cup [\![\mathsf{b}(y_1,y_1)]\!]^M_{\mathrm{IO},(y_1\mapsto\mathsf{d})} = \{\mathsf{b}(\mathsf{c},\mathsf{c}),\mathsf{b}(\mathsf{d},\mathsf{d})\}.$$

While in OI, we have four trees:

$$[\![\langle q,\mathsf{a}\rangle(\langle q,\mathsf{b}\rangle)]\!]^M_{\mathrm{OI}} = [\![\mathsf{b}(y_1,y_1)]\!]^M_{\mathrm{OI},(y_1\mapsto\{\mathsf{c},\mathsf{d}\})} = \{\mathsf{b}(\mathsf{c},\mathsf{c}),\mathsf{b}(\mathsf{c},\mathsf{d}),\mathsf{b}(\mathsf{d},\mathsf{c}),\mathsf{b}(\mathsf{d},\mathsf{d})\}.$$

In general, we always have $[\![u]\!]^M_{\mathrm{IO}} \subseteq [\![u]\!]^M_{\mathrm{OI}}$ for any expression u. In other words, we can say that OI have more nondeterminism than IO. Note that, however, this does not necessarily mean that OI is more expressive than IO; actually, the class of translations realized by IO- and OI- semantics are known to be incomparable.

For $\mu \in \{\mathrm{IO}, \mathrm{OI}\}$, the *translation realized by* M in μ-mode is the relation $\tau_{\mu,M} = \{(s,t) \in T_\Sigma \times T_\Delta \mid t \in [\![\langle q_0, s\rangle]\!]^M_\mu\}$. The class of all translations realized by all mtts in μ-mode is denoted by MT_μ. An mtt is called *deterministic* (respectively, *total*) if for every q, σ, the number of rules $|R_{q,\sigma}|$ is at most (at least) 1; the corresponding classes of translations are denoted by prefix D ($_{\mathrm{t}}$). An mtt is called *linear (in the input variables)* if in every right-hand side of the rules, each input variable x_i appears at most once; the corresponding class of translation is denoted by prefix L. For example, the class of translations realized by linear, deterministic, and total mtts in OI mode is denoted by $\mathrm{LD_tMT_{OI}}$. An mtt with all its states of rank 0 (i.e., without accumulating parameters) is called a *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 top-down tree transducer with one state only is also called a *linear tree homomorphism* and the corresponding class of translations is denoted by LHOM. Since for deterministic and total mtts the IO- and OI- semantics are known to coincide (Theorem 4.1 of [EV85]), and so for tts because the only difference Γ of the two semantics is never used in its evaluation, we always omit the subscripts $_{\mathrm{IO}}$ and $_{\mathrm{OI}}$ for the classes $\mathrm{D_tMT}$ and T.

Small Step Semantics Sometimes, it is convenient to regard the rules $\langle q, \sigma(\vec{x})\rangle(\vec{y}) \to r$ of mtts as rewrite rules over *sentential forms*. Intuitively, a sentential form is an output tree that contains state calls. In each rewriting step, we find a context that holds such a state call and expand the call to the right hand side of a matching rule with appropriate substitution. Since state calls can nest, different orders of evaluation yield different trees. We consider two strategy of rewriting, namely, *inside-out* (IO) that always rewrites the innermost state calls, and *outside-in* (OI) that always rewrites the outermost state calls.

For an mtt $M = (Q, q_0, \Sigma, \Delta, R)$, 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 $_M$ if clear from the context). Binary relations $\Rightarrow_{\mathrm{IO},M}$ and $\Rightarrow_{\mathrm{OI},M}$ over T_{Λ_M} is defined as follows. We have $u \Rightarrow_{\mathrm{IO},M} u'$ if and only if

$$u = \mathcal{C}[\langle q, \sigma(s_1, \ldots, s_k)\rangle(t_1, \ldots, t_m)], \quad \text{and}$$
$$u' = \mathcal{C}[r[x_1/s_1, \ldots, x_k/s_k, y_1/t_1, \ldots, y_m/t_m]]$$

for some one-hole Λ-context $\mathcal{C}$, a rule $\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r$, input trees $s_1, \ldots, s_k \in T_\Sigma$, and output trees $t_1, \ldots, t_m \in T_\Delta$. Note that each argument t_i of the state call is a concrete output tree not containing state calls. That is, in IO-derivation relation, only the innermost state calls are rewritten. Similarly, we define $u \Rightarrow_{\mathrm{OI},M} u'$ to hold if and only if

$$u = \mathcal{C}[\langle q, \sigma(s_1, \ldots, s_k)\rangle(u_1, \ldots, u_m)], \quad \text{and}$$
$$u' = \mathcal{C}[r[x_1/s_1, \ldots, x_k/s_k, y_1/u_1, \ldots, y_m/u_m]]$$

for some rule $\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r$, input trees $s_1, \ldots, s_k \in T_\Sigma$, sentential forms $u_1, \ldots, u_m \in T_\Lambda$, and a one-hole Λ-context $\mathcal{C}$ such that no position being prefix of the position of $\Box$ is labeled an symbol of $Q \times T_\Sigma$. In OI-derivation relation, rewrite is limited the outermost state calls. Note that each argument is not from T_Δ, but from T_Λ. It may contain inner state calls.

For $\mu \in \{\mathrm{IO}, \mathrm{OI}\}$, we say that v is derivable (in *μ-mode*) from u when $u \Rightarrow^*_{\mu,M} v$ and define $u{\downarrow_{\mu,M}} = \{t \in T_\Delta \mid u \Rightarrow^*_{\mu,M} t\}$. Two sentential forms u and v are equivalent, written $u \equiv_{\mu,M} v$, when $u{\downarrow_{\mu,M}} = v{\downarrow_{\mu,M}}$. The subscript $_M$ is omitted when it is clear.

The "big-step" and "small-step" semantics are well known to coincide as in the following proposition. In the rest of the thesis, we use both type of semantics, depending on the situation. Basically, when the structure of the rules or sentential forms is significant in the proof, we adopt small-step rewriting relations. When the value of generated output trees are important, we tend to exploit big-step semantics.

Proposition 2.2 (Theorem 3.15 and Theorem 3.21 of [EV85])**.** $[\![u]\!]^M_{\mathrm{IO}} = u{\downarrow_{M,\mathrm{IO}}}$ *and* $[\![u]\!]^M_{\mathrm{OI}} = u{\downarrow_{M,\mathrm{OI}}}$ *for any* $u \in T_\Lambda$*. (Note that in [EV85], the notation* $L_\mu(M, u)$ *is used in place of* $u{\downarrow_{\mu,M}}$ *and* $\uparrow_M$ *(*$\downarrow_M$*, respectively) is used for* $\downarrow_{\mathrm{IO},M}$ *(*$\downarrow_{\mathrm{OI},M}$*).)*

Mtts as Context Translations 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. For a ranked set Σ, a tree $\mathcal{C} \in T_{\Sigma \cup H_n}$ is called *Σ-n-context* (recall that $H_n = \{\Box_1, \ldots, \Box_n\}$). 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 $\Box_1$. 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 $\mathcal{C} \Rightarrow_{\mu,M} \mathcal{C}'$ when the relation holds in the extended mtt $M' = (Q, q_0, \Sigma, \Delta \cup \{\Box_1, \ldots, \Box_n\}, R)$. The relations $\mathcal{C} \Rightarrow^*_{\mu,M} \mathcal{C}'$ and $\mathcal{C}{\downarrow_{\mu,M}}$ 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 (mainly in Chapter 3).

Proposition 2.3 (Lemma 3.19 of [EV85]). *Let $\mathcal{C}$ be a Λ-n-context and $t_1, \ldots, t_n \in T_\Delta$. Then we have $\mathcal{C}[t_1, \ldots, t_n]{\downarrow_{\mathrm{IO}}} = \{\mathcal{D}[t_1, \ldots, t_n] \mid \mathcal{D} \in \mathcal{C}{\downarrow_{\mathrm{IO}}}\}$*

Proposition 2.4 (Lemma 5.2 of [EV85]). *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 $\square_1, \ldots, \square_n$, or (2) $\mathcal{C}$ contains at least one occurrence for each of $\square_1, \ldots, \square_n$ and the mtt is deterministic, then we have $\mathcal{C}[r_1, \ldots, r_n]{\downarrow_{\mathrm{IO}}} = \{\mathcal{D}[t_1, \ldots, t_n] \mid \mathcal{D} \in \mathcal{C}{\downarrow_{\mathrm{IO}}}, t_i \in r_i{\downarrow_{\mathrm{IO}}}\}$*

2.3 Multi-Return Macro Tree Transducers

Multi-return macro tree transducers extend mtts 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.

Definition 2.5. 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 $\langle q, \sigma(\vec{x})\rangle(\vec{y}) \to r$ where $r \in rhs^{D(q)}$ and, for $e \geq 1$ the set rhs^e is defined as:

$$
\begin{aligned}
r &::= l_1 \ldots l_n\ (u_1, \ldots, u_e) && (n \geq 0)\\
l &::= \mathsf{let}\ (z_{i_1}, \ldots, z_{i_{D(q')}}) \leftarrow \langle q'^{(k)}, x_j\rangle(u_1, \ldots, u_k)\ \mathsf{in}
\end{aligned}
$$

with $u_1, u_2, \ldots \in T_{\Delta \cup Y \cup Z}$, $q' \in Q$, $x_j \in X$, and $z_i \in Z$. We require that any rule is well-formed, that is, the leftmost occurrence of any variable z_i must appear at the "binding" position (between the symbol let and the symbol $\leftarrow$). We also require that each let-variable z_i occurs at most once at the binding position in a single right-hand side.

The IO semantics of multi-return mtts are defined under an environment $\Gamma : Y \cup$

$Z \to T_\Sigma$ as follows.

$$\begin{aligned}
[\![y_i]\!]^M_{\mathrm{IO},\Gamma} &= \{\Gamma(y_i)\} \\
[\![z_i]\!]^M_{\mathrm{IO},\Gamma} &= \{\Gamma(z_i)\} \\
[\![\delta(u_1,\ldots,u_n)]\!]^M_{\mathrm{IO},\Gamma} &= \{\delta(t_1,\ldots,t_n) \mid t_i \in [\![u_i]\!]^M_{\mathrm{IO},\Gamma} \text{ for all } i\} \\
[\![\langle q,\sigma(s_1,\ldots,s_k)\rangle(u_1,\ldots,u_m)]\!]^M_{\mathrm{IO},\Gamma} &= \\
&\bigcup \left\{ [\![r[x_1/s_1,\ldots,x_k/s_k]]\!]^M_{\mathrm{IO},\Gamma'} \,\middle|\, r \in R_{q,\sigma}, \Gamma'(y_i) \in [\![u_i]\!]^M_{\mathrm{IO},\Gamma} \text{ for all } i \right\} \\
[\![\mathsf{let}\ (z_1,\ldots,z_e) \leftarrow f\ \mathsf{in}\ \kappa]\!]^M_{\mathrm{IO},\Gamma} &= \\
&\bigcup \left\{ [\![\kappa]\!]^M_{\mathrm{IO},\Gamma +\!\!+ (z_1 \mapsto t_1,\ldots,z_d \mapsto t_d)} \,\middle|\, (t_1,\ldots,t_d) \in [\![f]\!]^M_{\mathrm{IO},\Gamma} \right\} \\
[\![(u_1,\ldots,u_d)]\!]^M_{\mathrm{IO},\Gamma} &= \{(t_1,\ldots,t_d) \mid t_i \in [\![u_i]\!]^M_{\mathrm{IO},\Gamma} \text{ for all } i\}
\end{aligned}$$

Here, $\Gamma +\!\!+ (z_1 \mapsto t_1,\ldots,z_d \mapsto t_d)$ denotes an environment Γ' such that $\Gamma'(z_i) = t_i$ and $\Gamma'(w) = \Gamma(w)$ for any other $w \in Y \cup Z$.

The *translation realized by* M in IO-mode is the relation $\tau_{\mathrm{IO},M} = \{(s,t) \in T_\Sigma \times T_\Delta \mid t \in [\![\langle q_0, s\rangle]\!]^M_{\mathrm{IO}}\}$. 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. *Total*, *deterministic*, and *linear* mr-mtts are defined as for mtts, and the prefixes D, $_t$, and L are used in the same way.

For mr-mtts we only consider IO semantics, because we could not find any sensible semantics for OI case. The problem of defining OI semantics for mr-mtts is in the semantics of let-bindings; how should we assign a *set* of trees for each let-variable z_i? The only meaningful way to define it seems to be: $\Gamma(z_i) = \{t_i \mid (\ldots,t_i,\ldots) \in [\![f]\!]\}$, but if we adopt this definition, then the two variables $\Gamma(z_1)$ and $\Gamma(z_2)$ that refer different component of the same return value of a state call behaves independently, which makes it useless to introduce tuple return values. In Chapter 2 and Chapter 3, we usually omit the $_{\mathrm{IO}}$ subscript, since the semantics considered is always IO.

2.4 Simulation of Multi-Return MTTs by MTTs

This section gives a characterization of mr-mtts by normal mtts. That is, it is shown that any mr-mtt can be decomposed into a three-fold composition of simpler transducers, namely, a pre-processing linear tree homomorphism for dealing with let-bindings, a single-return mtt doing the essential translation, and a post-processing linear total deterministic tt for dealing with tuples.

2.4.1 Dealing with tuple return values

To simulate tuple return values by single-return mtts, we use special symbols to represent tuples and selection. For $n \geq 1$, define $L^n_{tup} = \{\rho_1^{(1)}, \ldots, \rho_n^{(n)}, \pi_1^{(1)}, \ldots, \pi_n^{(1)}\}$. Intuitively, ρ_i means "make a tuple of i elements" and π_i means "select i-th element of". We construct the simulating single-return mr-mtt by translating each right-hand side of the original mr-mtt, so that (1) every function returns a single tree $\rho_e(t_1, \ldots, t_e)$ instead of a tuple $(t_1, \ldots, t_e)$ and (2) every use of a let-variable z is enclosed as $\pi_i(z)$ by a selection symbol π_i with an appropriate index i. Then, we apply a post-processing transducer that interprets these tupling and selection nodes. We define the transducer $tups^n_\Sigma$ whose purpose is to recursively convert subtrees of the form $\pi_i(\rho_k(t_1, \ldots, t_k))$ into t_i. The *tupling-and-selection transducer* $tups^n_\Sigma$ is the linear deterministic total top-down tree transducer with input alphabet $\Sigma \cup L^n_{tup}$, output alphabet Σ, set of states $\{q_1, \ldots, q_n\}$, initial state q_1, and the following rules for each q_i:

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

We add the third rule (the "otherwise" case) for two reasons: to make the transducer total, and to skip a ρ_1 node at the root position (note that in the simulating transducer, the initial state q_0 returns trees of the form $\rho_1(t_1)$).

Lemma 2.6. *For any $d \geq 1$, d-MM $\subseteq$ 1-MM$\,;\mathrm{LD_tT}$. Totality and determinism are preserved from the mr-mtt of dimension d to the mr-mtt of dimension 1. Also, the number 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 defined as follows. For the right-hand side r of the form

$$
\begin{aligned}
&\mathsf{let}\ (z_{i_1+1}, \ldots, z_{i_1+D(q_{j_1})}) \leftarrow \langle q_{j_1}, x_{k_1}\rangle(u_{1,1}, \ldots, u_{1,rank(q_{j_1})})\ \mathsf{in} \\
&\qquad\vdots \\
&\mathsf{let}\ (z_{i_n+1}, \ldots, z_{i_n+D(q_{j_n})}) \leftarrow \langle q_{j_n}, x_{k_n}\rangle(u_{n,1}, \ldots, u_{n,rank(q_{j_n})})\ \mathsf{in}\ (u_{0,1}, \ldots, u_{0,e}),
\end{aligned}
$$

the new right-hand side $et(r)$ is defined as follows

$$\begin{aligned}&\mathsf{let}\ z_1 \leftarrow \langle q_{j_1}, x_{k_1}\rangle(u_{1,1}\theta, \ldots, u_{1,rank(q_{j_1})}\theta)\ \mathsf{in}\\ &\qquad\qquad\vdots\\ &\mathsf{let}\ z_n \leftarrow \langle q_{j_n}, x_{k_n}\rangle(u_{n,1}\theta, \ldots, u_{n,rank(q_{j_n})}\theta)\ \mathsf{in}\ \rho_e(u_{0,1}\theta, \ldots, u_{0,e}\theta)\end{aligned}$$

where θ is the substitution that maps z_{i_x+y} to $\pi_y(z_x)$ for each $1 \leq x \leq n$ and $1 \leq y \leq D(q_{j_x})$. We now show the translation $\tau_{M'}$ followed by $\tau_{tups^d_\Delta}$ realizes τ_M. The proof is by induction on the structure of $s \in T_\Sigma$, showing the following equation

$$[\![\langle q, s\rangle(u_1, \ldots, u_m)]\!]^M_\Gamma = rmtup\Big([\![\langle q, s\rangle(u'_1, \ldots, u'_m)]\!]^{M'}_{\Gamma'}\Big)$$

for all states $q \in Q$, environments Γ and Γ' such that $[\![u_i]\!]^M_\Gamma = \tau_{tups^d_\Delta}([\![u'_i]\!]^{M'}_{\Gamma'})$ for all i, with $rmtup(X) = \{(\tau_{tups^d_\Delta}(t_1), \ldots, \tau_{tups^d_\Delta}(t_e)) \mid \rho_e(t_1, \ldots, t_e) \in X\}$. Note that it obviously follows from the definition of et that the root of any tree $t \in [\![\langle q, s\rangle(u'_1, \ldots, u'_m)]\!]^{M'}_\Gamma$ is labeled $\rho_{D(q)}$. Also note that when $X \subseteq \{\rho_1(t) \mid t \in T_{\Delta \cup L^d_{tup}}\}$, $rmtup$ coincides $tups^d_\Delta$. Therefore, if we apply the equation to q_0, we have $\tau_M = \tau_{M'}\,;\tau_{tups^d_\Delta}$, which proves the lemma. Let s be $\sigma(s_1, \ldots, s_k)$ (the base step for the induction is the case with $k = 0$). We immediately have

$$[\![\langle q, s\rangle(u_1, \ldots, u_m)]\!]^M_\Gamma = \bigcup\{[\![r[\vec{x}/\vec{s}]]\!]^M_\Xi \mid r \in R_{q,\sigma}, \Xi(y_i) \in [\![u_i]\!]_\Gamma\}$$

from the left-hand side and

$$\begin{aligned}&rmtup\Big([\![\langle q', s\rangle(u'_1, \ldots, u'_m)]\!]^{M'}_\Gamma\Big)\\ &\qquad= \bigcup\{[\![r'[\vec{x}/\vec{s}]]\!]^{M'}_{\Xi'} \mid r' \in R'_{q,\sigma}, \Xi'(y_i) \in [\![u'_i]\!]_{\Gamma'}\}.\\ &\qquad= \bigcup\{rmtup([\![et(r[\vec{x}/\vec{s}])]\!]^{M'}_{\Xi'}) \mid r \in R_{q,\sigma}, \Xi'(y_i) \in [\![u'_i]\!]_{\Gamma'}\}\end{aligned}$$

from the right-hand side. To prove these two sets are equal, it is sufficient to show $[\![r[\vec{x}/\vec{s}]]\!]^M_\Xi = rmtup([\![et(r[\vec{x}/\vec{s}])]\!]^{M'}_{\Xi'})$. Here, by nested induction on the structure of r, we prove a slightly stronger proposition. That is, $[\![r[\vec{x}/\vec{s}]]\!]^M_\Xi = rmtup([\![et(r[\vec{x}/\vec{s}])]\!]^{M'}_{\Xi'})$ for any Ξ and Ξ' such that the following condition holds: $\Xi(y) = \tau_{tups^d_\Delta}(\Xi'(y))$ for all $y \in Y$ and $(\Xi(z_{i_x+1}), \ldots, \Xi(z_{i_x+d})) = rmtup(\Xi'(z_x))$ for all corresponding variables $(z_{i_x+1}, \ldots, z_{i_x+d})$ and z_x. Note that for such Ξ and Ξ', we have $[\![u]\!]^M_\Xi = \tau_{tups^d_\Delta}([\![u\theta]\!]^{M'}_{\Xi'})$ for $u \in T_{\Delta \cup Y \cup Z}$ and θ as in the definition of et, because $[\![z_{i_x+y}]\!]^M_\Xi = \Xi(z_{i_x+y}) = \tau_{tups^d_\Delta}(\Xi'(z_x)) = \tau_{tups^d_\Delta}([\![z_x]\!]^{M'}_{\Xi'})$. Thus, if r is of the form $(u_{0,1}, \ldots, u_{0,e})$, we have $[\![r[\vec{x}/\vec{s}]]\!]^M_\Xi = rmtup([\![et(r[\vec{x}/\vec{s}])]\!]^{M'}_{\Xi'})$. If r is of the form

$$\mathsf{let}\ (z_{i_1+1}, \ldots, z_{i_1+D(q_{j_1})}) \leftarrow \langle q_{j_1}, x_{k_1}\rangle(u_{1,1}, \ldots, u_{1,rank(q_{j_1})})\ \mathsf{in}\ r',$$

then $[\![r[\vec{x}/\vec{s}]]\!]^M_\Xi$ is the union of all $[\![r'[\vec{x}/\vec{s}]]\!]^M_\zeta$'s over environments ζ such that $\zeta(z_{i_1+y}) = t_y$ for some $(t_1, \ldots, t_{D(q_{j_1})}) \in [\![f_1]\!] = [\![\langle q_{j_1}, s_{k_1}\rangle(u_{1,1}, \ldots, u_{1,rank(q_{j_1})})]\!]^M_\Xi$. Similarly, then $[\![et(r[\vec{x}/\vec{s}])]\!]^M_\Xi$ is the union over all ζ''s such that $\zeta'(z_1) = t_1$ for some $t_1 \in [\![f'_1]\!] = [\![\langle q_{j_1}, s_{k_1}\rangle(u_{1,1}\theta, \ldots, u_{1,rank(q_{j_1})})\theta]\!]^{M'}_{\Xi'}$. Since by outer induction hypothesis, $[\![f_1]\!] = rmtup([\![f'_1]\!])$, the environments ζ and ζ' satisfies the condition of inner induction. Therefore, we have $[\![r'[\vec{x}/\vec{s}]]\!]^M_\zeta = rmtup([\![et(r'[\vec{x}/\vec{s}])]\!]^{M'}_{\zeta'})$ by inner induction hypothesis, which proves the lemma. □

2.4.2 Dealing with 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) mr-mtt rule $\mathsf{let}\ z \leftarrow \langle q, x\rangle\ \mathsf{in}\ \delta(z, z)$ is not necessarily equivalent to the mtt rule $\delta(\langle q, x\rangle, \langle q, x\rangle)$. 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 a helper state p and a set of helper 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 helper states to simulate copying must be applied to some child of the current node. Consider the following rule:

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

To simulate the first let-binding, we need a helper 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 helper 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 the *sibling* of x_i.

Our solution is to insert helper nodes of rank-1 above each node of the input tree, similar as done for the removal of stay moves in [EM03a]. We can then run the helper 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

$$\langle p, \bar{\sigma}_1(x_1)\rangle \to \langle p, x_1\rangle(\langle\langle q, 1\rangle, x_1\rangle)$$
$$\langle p, \bar{\sigma}_2(x_1)\rangle(z_1) \to \langle p, x_1\rangle(z_1, \langle\langle q_2, 2\rangle, x_1\rangle(z_1)).$$

The new helper state $\langle q, i\rangle$ "skips" the following barred nodes and calls q at the i-th child of the next σ-node.

For $n \in \mathbb{N}$, we define the transducer mon^n_Σ ("*mon*adic insertion") with the single state q that inserts n helper nodes for each input node as follows:

$$\langle q, \sigma(x_1, \ldots, x_k)\rangle \to \bar{\sigma}_1(\bar{\sigma}_2(\cdots \bar{\sigma}_n(\sigma(\langle q, x_1\rangle, \ldots, \langle q, x_k\rangle))\cdots) \quad \text{for each } \sigma \in \Sigma^{(k)}.$$

Since mon^n_Σ is total deterministic top-down tree transducer with the number of states is one, it falls into the class of linear tree homomorphism.

Lemma 2.7. 1-MM $\subseteq$ LHOM ; $\mathrm{MT}_{\mathrm{IO}}$. *Totality and determinism are preserved from the mr-mtt to the mtt. If the mr-mtt has n states of rank $\leq k$, r rules, $\leq l$ let-bindings in any rule, and input symbols of rank $\leq b$, then the mtt has at most $n + nb + r$ states, $k + l$ parameters, and $rl + qbl$ rules.*

Proof. Let the mr-mtt be $M = (Q, \Sigma, \Delta, q_0, R, D)$. The state set of the simulating mtt M' is $Q \cup \{q_r^{(k+m)} \mid r \in R, m$ is the number of let-bindings in r and k is the 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:

$$\langle q, \sigma(\vec{x})\rangle(\vec{y}) \to \mathsf{let}\ z_1 \leftarrow \langle p_1, x_{i_1}\rangle(\cdots)\ \mathsf{in}\ \cdots \mathsf{let}\ z_m \leftarrow \langle p_m, x_{i_m}\rangle(\cdots)\ \mathsf{in}\ u.$$

The simulating mtt has the following rules each corresponding to one let-binding:

$$\begin{aligned}
\langle q, \bar{\sigma}_1(x_1)\rangle(\vec{y}) &\to \langle q_r, x\rangle(\vec{y}, \langle\langle p_1, i_1\rangle, x_1\rangle(\cdots), \mathtt{dmy}, \ldots, \mathtt{dmy})\\
\langle q_r, \bar{\sigma}_2(x_1)\rangle(\vec{y}, z_1, \ldots, z_m) &\to \langle q_r, x\rangle(\vec{y}, z_1, \langle\langle p_2, i_2\rangle, x_1\rangle(\cdots), \mathtt{dmy}, \ldots, \mathtt{dmy})\\
&\vdots\\
\langle q_r, \bar{\sigma}_m(x_1)\rangle(\vec{y}, z_1, \ldots, z_m) &\to \langle q_r, x\rangle(\vec{y}, z_1, \ldots, z_{m-1}, \langle\langle p_m, i_m\rangle, x_1\rangle(\cdots))\\
\langle q_r, \bar{\sigma}_{m+1}(x_1)\rangle(\vec{y}, z_1, \ldots, z_m) &\to \langle q_r, x\rangle(\vec{y}, z_1, \ldots, z_m)\\
&\vdots\\
\langle q_r, \bar{\sigma}_l(x_1)\rangle(\vec{y}, z_1, \ldots, z_m) &\to \langle q_r, x\rangle(\vec{y}, z_1, \ldots, z_m)\\
\langle q_r, \sigma(\vec{x})\rangle(\vec{y}, z_1, \ldots, z_m) &\to u
\end{aligned}$$

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

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

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

Obviously, $[\![\langle\langle p, j\rangle, \bar{\sigma}_i(\cdots\bar{\sigma}_i(\sigma(\vec{s}))\cdots)\rangle(\vec{t})]\!]^{M'}_{\Gamma} = [\![\langle p, s_j\rangle(\vec{t})]\!]^{M'}_{\Gamma}$ when the rank of σ is more than or equal to j.

In order to show $\tau_M = mon^l_\Sigma\,;\tau_{M'}$ and hence 1-MM $\subseteq$ LHOM$\,;$MT, it is sufficient to prove $[\![\langle q, s\rangle(\vec{u})]\!]^M_\Gamma = [\![\langle q, mon^l_\Sigma(s)\rangle(\vec{u})]\!]^{M'}_{\Gamma'}$ for all q, s, and $\vec{u}$ such that $[\![u_i]\!]^M_\Gamma = [\![u'_i]\!]^{M'}_{\Gamma'}$. The proof is by induction on the structure of s. Let $s = \sigma(s_1, \ldots, s_k)$ and r be a right-hand side in $R_{q,\sigma}$ of the form $\mathsf{let}\ z_1 \leftarrow \langle p_1, x_{i_1}\rangle(\cdots)\ \mathsf{in}\ \cdots \mathsf{let}\ z_m \leftarrow \langle p_m, x_{i_m}\rangle(\cdots)\ \mathsf{in}\ u$. By a straightforward induction on the number of let-bindings, for all $1 \le j \le m+1$ we can show the following equation

$$\begin{aligned}&[\![\mathsf{let}\ z_j \leftarrow \langle p_j, s_{i_j}\rangle(\cdots)\ \mathsf{in}\ \cdots \mathsf{let}\ z_m \leftarrow \langle p_m, s_{i_m}\rangle(\cdots)\ \mathsf{in}\ u]\!]^M_\Gamma \\ &\quad = [\![\langle q_r, \bar{\sigma}_i(\cdots\sigma(s_1, \ldots, s_k)\cdots)\rangle(\vec{y}, z_1, \cdots, z_{j-1}, \langle\langle p_j, i_j\rangle, x_1\rangle(\cdots), \mathtt{dmy}, \ldots, \mathtt{dmy})]\!]^{M'}_\Gamma\end{aligned}$$

which proves the outer induction.

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 q_r are deterministic, and they are total if the original state q was. Newly added states $\langle p, j\rangle$ are all deterministic. For the remaining undefined part (q-rules for $\bar{\sigma}_2, \ldots, \bar{\sigma}_l$ and σ and $\langle p, j\rangle$-rules for σ with rank $< j$), we can add dummy rules to regain totality if the original mr-mtt was total. □

By combining Lemmas 2.6 and 2.7, we obtain the main theorem of this section.

Theorem 2.8. MM $\subseteq$ LHOM$\,;$MT$_{\mathrm{IO}}\,;$LD$_{\mathrm{t}}$T. *Totality and determinism are preserved.*

2.5 Simulation of MTT compositions by Multi-Return MTTs

In this section, we show that a composition of an mtt with any total deterministic top-down tree transducer can be simulated by a single mr-mtt, i.e., we have the inverse inclusion of Lemma 2.8. In fact, the simulation can be generalized to a *composition of an mr-mtt* with total deterministic top-down tree transducers, which gives a better compositionality of mr-mtts.

2.5.1 Right Composition with a DtT

We show that MM is closed under right-composition with $\mathrm{D_tT}$. 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 $(\llbracket\langle p_1, t_1\rangle\rrbracket, \ldots, \llbracket\langle p_1, t_d\rangle\rrbracket, \ldots, \llbracket\langle p_n, t_1\rangle\rrbracket, \ldots, \llbracket\langle p_n, t_d\rangle\rrbracket)$.

Lemma 2.9. $\mathrm{MM}\,;\mathrm{D_tT} \subseteq \mathrm{MM}$. *Totality and determinism are preserved. The number of parameters and the dimension of the resulting mr-mtt is n times larger the original one, where n is the number of states of the tt. The number of states and the number of rules each increase by 1.*

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

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

where runN and runN_0 are defined inductively as follows.

$$\begin{aligned} & \mathit{runN}_0((u_1)) = (\llbracket\langle p_1, u_1\rangle\rrbracket^{N'}) \\ & \mathit{runN}_0(\mathsf{let}\ (z_1, \ldots, z_d) \leftarrow \langle q, x\rangle(u_1, \ldots, u_k)\ \mathsf{in}\ \kappa) = \\ & \qquad\qquad \mathsf{let}\ (z_1, \ldots, z_{dn}) \leftarrow \langle q, x\rangle(\mathbf{pu}_k)\ \mathsf{in}\ \mathit{runN}_0(\kappa) \\ & \mathit{runN}((u_1, \ldots, u_d)) = (\mathbf{pu}_d) \\ & \mathit{runN}(\mathsf{let}\ (z_1, \ldots, z_d) \leftarrow \langle q, x\rangle(u_1, \ldots, u_k)\ \mathsf{in}\ \kappa) = \\ & \qquad\qquad \mathsf{let}\ (z_1, \ldots, z_{dn}) \leftarrow \langle q, x\rangle(\mathbf{pu}_k)\ \mathsf{in}\ \mathit{runN}(\kappa) \end{aligned}$$

where $\mathbf{pu}_m = \llbracket\langle p_1, u_1\rangle\rrbracket^N, \ldots, \llbracket\langle p_1, u_m\rangle\rrbracket^N, \ldots, \llbracket\langle p_n, u_1\rangle\rrbracket^N, \ldots, \llbracket\langle p_n, u_m\rangle\rrbracket^N$ with 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 $1 \le i \le k$, $1 \le j \le n$.

By induction on the structure of s, we show the following equation

$$\begin{aligned} & \llbracket\langle q, s\rangle(u'_1, \ldots, u'_k, \ldots, u'_{n(k-1)+1}, \ldots, u'_{kn})\rrbracket^{M'}_{\Gamma'} = \\ & \qquad \left\{ \left(\llbracket\langle p_1, t_1\rangle\rrbracket^N, \ldots, \llbracket\langle p_n, t_d\rangle\rrbracket^N\right) \;\middle|\; (t_1, \ldots, t_d) \in \llbracket\langle q, s\rangle(u_1, \ldots, u_k)\rrbracket^M_\Gamma \right\} \end{aligned}$$

for all $q \in Q$, $u'_i \in T_{\Delta \cup Y}$, and environments Γ, Γ' that are satisfying $\llbracket u'_{(i-1)n+j}\rrbracket^{M'}_{\Gamma'} = \llbracket\langle p_j, \llbracket u_i\rrbracket^M_\Gamma\rangle\rrbracket^N$. (Note that here we abuse our notation so that $\llbracket u_i\rrbracket$ denotes its unique

element rather than the set itself. Since u_i's do not contain state calls, its semantic values is uniquely determined from each environment.) Let $s' = \sigma(s_1, \ldots, s_k)$ (the base case of the induction is the case $k = 0$). By the definition of the IO-semantics, we have

$$[\![\langle q, s\rangle(u'_1, \ldots, u'_k, \ldots, u'_{n(k-1)+1}, \ldots, u'_{kn})]\!]^{M'}_{\Gamma'} = \bigcup_{r \in R_{q,\sigma}} \{[\![runN(r)]\!]^M_{\Xi'} \mid \Xi(y_i) \in [\![u'_i]\!]^{M'}_{\Gamma'}\}$$

and

$$\left\{ \left([\![\langle p_1, t_1\rangle]\!]^N, \ldots, [\![\langle p_n, t_d\rangle]\!]^N\right) \,\middle|\, (t_1, \ldots, t_d) \in [\![\langle q, s\rangle(u_1, \ldots, u_k)]\!]^M_\Gamma \right\} = \bigcup_{r \in R_{q,\sigma}} \left\{ \left([\![\langle p_1, t_1\rangle]\!]^N, \ldots, [\![\langle p_n, t_d\rangle]\!]^N\right) \,\middle|\, (t_1, \ldots, t_d) \in [\![r]\!]^M_\Xi, \Xi(y_i) \in [\![u_i]\!]^M_\Gamma \right\}$$

To show these two sets are equal, it is sufficient to prove the following statement:

$$[\![runN(r)]\!]^{M'}_{\Xi'} = \left\{ \left([\![\langle p_1, t_1\rangle]\!]^N, \ldots, [\![\langle p_n, t_d\rangle]\!]^N\right) \,\middle|\, (t_1, \ldots, t_d) \in [\![r]\!]^M_\Xi \right\}$$

for any $\Xi'(y_{(i-1)n+j}) = [\![\langle p_j, \Xi(y_i)\rangle]\!]^N$ and $\Xi'(z_{(i-1)n+j}) = [\![\langle p_j, \Xi(z_i)\rangle]\!]^N$. We prove this by inner induction on the structure of r. If $r = (u_1, \ldots, u_d)$, then $runN(r)$ becomes $\mathbf{pu}_d = ([\![\langle p_1, u_1\rangle]\!]^N, \ldots, [\![\langle p_1, u_d\rangle]\!]^N, \ldots, [\![\langle p_n, u_1\rangle]\!]^N, \ldots, [\![\langle p_n, u_d\rangle]\!]^N)$. Hence the $((i-1)n + j)$-th element of unique tree $[\![runN(r)]\!]^{M'}_{\Xi'}$ is $[\![[\![\langle p_j, u_i\rangle]\!]^N]\!]^{M'}_{\Xi'}$, and the $((i-1)n+j)$-th element of $\left\{ \left([\![\langle p_1, t_1\rangle]\!]^N, \ldots, [\![\langle p_n, t_d\rangle]\!]^N\right) \,\middle|\, (t_1, \ldots, t_d) \in [\![r]\!]^M_\Xi \right\}$ is $[\![\langle p_j, [\![t_i]\!]^M_\Xi\rangle]\!]^N$. From the condition on Ξ and Ξ', these two trees are equal.

If $r = \mathsf{let}\ (z_1, \ldots, z_d) \leftarrow \langle q', x_i\rangle(u_1, \ldots, u_m)\ \mathsf{in}\ r^\dagger$ $(\in rhs^e)$, then we have $runN(r) = \mathsf{let}\ (z_1, \ldots, z_{dn}) \leftarrow \langle q', x_i\rangle(\mathbf{pu}_m)\ \mathsf{in}\ runN(r^\dagger)$. Therefore, by definition of IO-semantics, $[\![runN(r)]\!]^{M'}_{\Xi'}$ is equal to $\bigcup\{[\![runN(r^\dagger)]\!]^{M'}_{\Theta'} \mid \Theta' = \Xi' +\!\!+ (z_1 \mapsto t_1, \ldots, z_{dn} \mapsto t_{dn}), (t_1, \ldots, t_{dn}) \in [\![\langle q', x_i\rangle(\mathbf{pu}_m)]\!]^{M'}_{\Xi'}\}$, and by outer induction hypothesis, we can replace the state call $[\![\langle q', x_i\rangle(\mathbf{pu}_m)]\!]^{M'}_{\Xi'}$ with its counterpart in M and obtain the following equal set: $\bigcup\{[\![runN(r^\dagger)]\!]^{M'}_{\Theta'} \mid \Theta' = \Xi' +\!\!+ (z_1 \mapsto [\![\langle p_1, t_1\rangle]\!]^N, \ldots, z_{dn} \mapsto [\![\langle p_n, t_d\rangle]\!]^N), (t_1, \ldots, t_d) \in [\![\langle q', x_i\rangle(u_1, \ldots, u_m)]\!]^M_\Xi\}$. Then, by inner induction hypothesis, this set is equal to $\bigcup\{([\![\langle p_1, t^\dagger_1\rangle]\!]^N, \ldots, [\![\langle p_n, t^\dagger_e\rangle]\!]^N) \mid (t^\dagger_1, \ldots, t^\dagger_e) \in [\![r^\dagger]\!]^M_\Theta, \Theta = \Xi +\!\!+ (z_1 \mapsto t_1, \ldots, z_d \mapsto t_d), (t_1, \ldots, t_d) \in [\![\langle q', x_i\rangle(u_1, \ldots, u_m)]\!]^M_\Xi\}$. Finally, by definition of IO-semantics this is equal to $\left\{ \left([\![\langle p_1, t^\dagger_1\rangle]\!]^N, \ldots, [\![\langle p_n, t^\dagger_e\rangle]\!]^N\right) \,\middle|\, (t^\dagger_1, \ldots, t^\dagger_d) \in [\![r]\!]^M_\Xi \right\}$.

By a very similar induction on right-hand side structure (using $runN_0$ instead of $runN$), we can also show $[\![\langle \hat{q}, s\rangle]\!]^{M'} = \left\{ [\![\langle p_1, t\rangle]\!]^N \,\middle|\, t \in [\![\langle q_0, s\rangle]\!]^M_\Gamma \right\}$, which proves $\tau_M \,;\tau_N = \tau_{M'}$ and hence $\mathrm{MM}\,;\mathrm{D_tT} \subseteq \mathrm{MM}$. □

Note that the proof of Lemma 2.9 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.

2.5.2 Left Composition with a DtT

Next, we investigate the case of left-composition. The idea is, again, to simulate the composition $\mathrm{D_tT}\,;\mathrm{MT}$ 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:

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

Using a let-binding, we construct a rule of the simulating transducer as follows:

$$\langle\langle p, q\rangle, \mathtt{e}(x_1)\rangle(y_1) \to \mathsf{let}\ z \leftarrow \langle\langle p, q\rangle, x_1\rangle\ \mathsf{in}\ \mathtt{d}(z, z)$$

which correctly preserves the original semantics that the left and right child of the $\mathtt{d}$ node are equal. But without let-bindings, we cannot avoid duplicating a state call, and at best we will have the following rule:

$$\langle\langle p, q\rangle, \mathtt{e}(x_1)\rangle \to \mathtt{d}(\langle\langle p, q\rangle, x_1\rangle, \langle\langle p, q\rangle, x_1\rangle).$$

This is clearly incorrect, since the duplicated two state calls may behave nondeterministically, and yield different output trees, which is not intended by the original translation.

Lemma 2.10. $\mathrm{D_tT}\,;\mathrm{MT_{IO}} \subseteq$ 1-MM. *Totality and determinism are preserved. The number of states is n times larger, where n is the number of states of the* $\mathrm{D_tT}$. *The number of parameters remains same. The number of rules may be double exponential with respect to the depth of right-hand sides of the* $\mathrm{D_tT}$.

Proof. Let $M_1 = (Q, \Sigma, \Gamma, q_0, R_1)$ be a $\mathrm{D_tT}$ 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:

$$R = \{\langle\langle p, q\rangle, \sigma(\vec{x})\rangle(\vec{y}) \to \kappa \mid \kappa \in f_z(\langle p, r\rangle(\vec{y}), z),$$
$$r \text{ is the right-hand side of the unique } \langle q, \sigma\rangle\text{-rule of } R_1)\}.$$

Intuitively, a state $\langle p, q\rangle$ denotes the translation by q followed by p. The relation f_z is very similar to the IO-semantics of M_2 (thus, $f_z(\langle p, r\rangle(\vec{y}), u)$ should be intuitively read as $[\![u]\!]^{M_2}_{\Gamma}$ with $\Gamma(z) \in [\![\langle p, r\rangle(\vec{y})]\!]^{M_2}$. 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:

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

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 s, we show

$$[\![\langle\langle p, q\rangle, s\rangle(u'_1, \ldots, u'_m)]\!]^{M_\times}_{\Theta_\times} = [\![\langle p, [\![\langle q, s\rangle]\!]^{M_1}\rangle(u_1, \ldots, u_m)]\!]^{M_2}_{\Theta}$$

for any $p \in P$, $q \in Q$, and environments $\Theta_\times$ and Θ such that $[\![u'_i]\!]^{M_\times}_{\Theta_\times} = [\![u_i]\!]^{M_2}_{\Theta}$ for all i. From this proposition, it immediately follows that $\tau_{M_1}\,;\tau_{M_2} = \tau_{M_\times}$ and hence $\mathrm{D_t T}\,;\mathrm{MT} \subseteq$ 1-MM, by taking $p = p_0$ and $q = q_0$. Let $s = \sigma(s_1, \ldots, s_k)$ (the base case is the case $k = 0$) and r the unique right-hand side in $R_{p,\sigma}$. By definition of the IO-semantics, we have

$$[\![\langle\langle p, q\rangle, s\rangle(u'_1, \ldots, u'_m)]\!]^{M_\times}_{\Theta_\times} = \bigcup_{\kappa \in f_z(\langle p, r\rangle(\vec{y}), z)} \left\{ [\![\kappa[\vec{x}/\vec{s}]]\!]^{M_\times}_{\Xi_\times} \mid \Xi(y_i) \in [\![u'_i]\!]^{M_\times}_{\Theta_\times} \right\}$$

and

$$[\![\langle p, [\![\langle q, s\rangle]\!]^{M_1}\rangle(u_1, \ldots, u_m)]\!]^{M_2}_{\Theta} = [\![\langle p, [\![r[\vec{x}/\vec{s}]]\!]^{M_1}\rangle(u_1, \ldots, u_m)]\!]^{M_2}_{\Theta}$$

To show these two sets are equal, it is sufficient to show $\bigcup_{\kappa \in f_z(r', u)} [\![\kappa[\vec{x}/\vec{s}]]\!]^{M_\times}_{\Xi} = \{[\![u]\!]^{M_\times}_{\Xi +\!\!+ (z \mapsto t)} \mid t \in [\![ev1(r')]\!]^{M_2}_{\Xi}\}$ where $ev1(r')$ is $r'[\vec{x}/\vec{s}]$ whose all subexpressions of form $\langle p, \gamma(\ldots)\rangle(t_1, \ldots, t_k)$ is replaced with $\langle p, [\![\gamma(\ldots)]\!]^{M_1}\rangle(t_1, \ldots, t_k)$. The proof is by inner induction on r', where r' is ordered firstly by the structure of Λ_{M_1} trees (the sentential forms of M_1) contained in r' and secondly by the structure of r' itself.

If $r' = y$, we have $\bigcup_{\kappa \in f_z(r', u)} [\![\kappa[\vec{x}/\vec{s}]]\!]^{M_\times}_{\Xi} = [\![u[z/y]]\!]^{M_\times}_{\Xi}$ and $\{[\![u]\!]^{M_\times}_{\Xi +\!\!+ (z \mapsto t)} \mid t \in [\![ev1(r')]\!]^{M_2}_{\Xi}\} = [\![u]\!]^{M_\times}_{\Xi +\!\!+ (z \mapsto \Xi(y))}$, hence, these two sets are equal.

If $r' = \delta(r_1, \ldots, r_k)$, using the inner induction hypothesis k times, we have the following equality: $\bigcup_{\kappa \in f_z(r',u)} [\![\kappa[\vec{x}/\vec{s}]]\!]_{\Xi}^{M_\times} = \{[\![u[z/\delta(z_1, \ldots, z_k)]]\!]_{\Xi +\!\!+ (z_1 \mapsto t_1, \ldots, z_k \mapsto t_k)}^{M_\times} \mid t_i \in [\![ev1(r_i)]\!]_{\Xi}^{M_2}\}$, which is equal to $\{[\![u]\!]_{\Xi +\!\!+ (z \mapsto \delta(t_1, \ldots, t_k))}^{M_\times} \mid t_i \in [\![ev1(r_i)]\!]_{\Xi}^{M_2}\}$ and therefore by definition of the IO-semantics equal to $\{[\![u]\!]_{\Xi +\!\!+ (z \mapsto t)}^{M_\times} \mid t \in [\![ev1(r')]\!]_{\Xi}^{M_2}\}$ as desired.

Similarly if $r' = \langle p, \langle q, x_i \rangle\rangle(r_1, \ldots, r_k)$, using the inner induction hypothesis k times, we have $\bigcup_{\kappa \in f_z(r',u)} [\![\kappa[\vec{x}/\vec{s}]]\!]_{\Xi}^{M_\times} = \{[\![\mathsf{let}\ z \leftarrow \langle\langle p, q\rangle, s_i\rangle(z_1, \ldots, z_k)\ \mathsf{in}\ u]\!]_{\Xi +\!\!+ (z_1 \mapsto t_1, \ldots, z_k \mapsto t_k)}^{M_\times} \mid t_i \in [\![ev1(r_i)]\!]_{\Xi}^{M_2}\}$. By the definition of the IO-semantics and by the fact u does not contain any occurrences of $z_1, \ldots, z_k$ from definition, this is equal to $\{[\![u]\!]_{\Xi +\!\!+ (z \mapsto t)}^{M_\times} \mid t \in [\![\langle\langle p, q\rangle, s_i\rangle(z_1, \ldots, z_k)]\!]_{\Xi +\!\!+ (z_1 \mapsto t_1, \ldots, z_k \mapsto t_k)}^{M_\times}\}$, and by outer induction hypothesis equal to $\{[\![u]\!]_{\Xi +\!\!+ (z \mapsto t)}^{M_\times} \mid t \in [\![ev1(r')]\!]_{\Xi}^{M_2}\}$.

Finally, if $r' = \langle p, \gamma(\vec{g})\rangle(t_1, \ldots, t_k)$, again by using the inner induction hypothesis $k+1$ times, we have $\bigcup_{\kappa \in f_z(r',u)} [\![\kappa[\vec{x}/\vec{s}]]\!]_{\Xi}^{M_\times} = \bigcup_{r'' \in R_{p,\gamma}} \{[\![u]\!]_{\Xi +\!\!+ (z_1 \mapsto t_1, \ldots, z_k \mapsto t_k, z \mapsto t)}^{M_\times} \mid t_i \in [\![ev1(r_i)]\!]_{\Xi}^{M_2}, t \in [\![ev1(r''[\vec{x}/\vec{g}, \vec{y}/\vec{z}])]\!]_{\Xi}^{M_2}\}$, which is equal to $\bigcup_{r'' \in R_{p,\gamma}} \{[\![u]\!]_{\Xi +\!\!+ (z \mapsto t)}^{M_\times} \mid t \in [\![ev1(r''[\vec{x}/\vec{g}])]\!]_{\Xi +\!\!+ (y_1 \mapsto t_1, \ldots, y_k \mapsto t_k)}^{M_2}, t_i \in [\![ev1(r_i)]\!]_{\Xi}^{M_2}\}$ and hence to $\{[\![u]\!]_{\Xi +\!\!+ (z \mapsto t)}^{M_\times} \mid t \in \bigcup_{r'' \in R_{p,\gamma}} \{[\![ev1(r''[\vec{x}/\vec{g}])]\!]_{\Xi +\!\!+ (y_1 \mapsto t_1, \ldots, y_k \mapsto t_k)}^{M_2} \mid t_i \in [\![ev1(r_i)]\!]_{\Xi}^{M_2}\}\}$. By definition of IO-semantics, this is equal to $\{[\![u]\!]_{\Xi +\!\!+ (z \mapsto t)}^{M_\times} \mid t \in [\![ev1(r')]\!]_{\Xi}^{M_2}\}$. □

We can now generalize the lemma in two directions. First, we generalize the latter transducer from MT to MM. Next, we relax the restriction that the former top-down transducer must be total.

Lemma 2.11. $\mathrm{D_tT}\,;\mathrm{MM} \subseteq \mathrm{MM}$. *Totality and determinism are preserved.*

Proof. By Lemma 6.9 of [Tha70], the class $\mathrm{D_tT}$ is closed under the composition. Hence, we have the following sequence of inequations that proves the lemma.

$$\begin{aligned}
\mathrm{D_tT}\,;\mathrm{MM} &\subseteq \mathrm{D_tT}\,;\mathrm{LHOM}\,;\mathrm{MT_{IO}}\,;\mathrm{LD_tT} && \text{by Theorem 2.8}\\
&\subseteq \mathrm{D_tT}\,;\mathrm{MT_{IO}}\,;\mathrm{LD_tT} && \text{by Lemma 6.9 of [Tha70]}\\
&\subseteq \mathrm{MM}\,;\mathrm{LD_tT} && \text{by Lemma 2.10}\\
&\subseteq \mathrm{MM} && \text{by Lemma 2.9.}
\end{aligned}$$

□

Lemma 2.12. $\mathrm{DT}\,;\mathrm{MM} \subseteq \mathrm{MM}$.

Proof. We have $\mathrm{DT} \subseteq \mathrm{DTFTA}\,;\mathrm{D_tT}$ (Lemma 5.22 of [EV85]) where DTFTA is the class of partial identity translations recognized by deterministic top-down tree automaton, Lemma 2.11, and $\mathrm{DTFTA}\,;\mathrm{MM} \subseteq \mathrm{MM}$ (can be proved by the same construction for Lemma 5.21 of [EV85]. For every rule of the initial state, we add one

let-binding that carries out the run of the automata.) These three lemmas prove $\mathrm{DT}\,;\mathrm{MM} \subseteq \mathrm{MM}$. □

2.6 Results

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 2.13. $\mathrm{MM} = \mathrm{LHOM}\,;\mathrm{MT}_{\mathrm{IO}}\,;\mathrm{LD}_{\mathrm{t}}\mathrm{T}$. *Determinism and totality are preserved between the mr-mtt and the mtt.*

Theorem 2.14. MM *is closed under left-composition with* DT *and right-composition with* $\mathrm{D}_{\mathrm{t}}\mathrm{T}$.

Chapter 3

Expressive Power of Multi-Return MTTs

A natural question that arises here is: what is the exact relationship between mtts and mr-mtts in terms of expressiveness? It is known in the literature that mtts are not closed under left-composition with LHOM's, while mr-mtts are closed as we have shown in the previous chapter. This immediately implies that mr-mtts are strictly more powerful than mtts. Recall, however, that the closure under left-composition is already achieved for a restricted class of mr-mtts, i.e., 1-MM. Therefore, what we can tell so far is only that let bindings do add power for mtts in terms of expressiveness. How about the *multi-return* facility? In this chapter, we give a formal proof of the fact that 1-MM (and hence $\mathrm{MT}_{\mathrm{IO}}$) is not closed under right-composition with $\mathrm{D_tT}$'s. Together with the results from the previous chapter, the result tells us that multi-return facility also adds expressive power for mtts, in the form of right-compositionality.

3.1 Deterministic and Dimension-1 mr-mtts

First of all, let us summarize several results concerning expressiveness that are immediately derived from the characterization given in the previous chapter. Recall that we have $\mathrm{DMM} = \mathrm{LHOM}\,;\mathrm{DMT}\,;\mathrm{LD_tT}$ and $\mathrm{D_tMM} = \mathrm{LHOM}\,;\mathrm{D_tMT}\,;\mathrm{LD_tT}$ by Theorem 2.13. Since, by Theorem 7.6 of [EV85], DMT and $\mathrm{D_tMT}$ are both closed under left- and- right- composition with $\mathrm{D_tT}$, we obtain the following corollary. (Note that the right part follows also from the result of [EV94], that total deterministic tree generating top-down tree-to-graph transducers (trgen-tg) are equivalent to $\mathrm{D_tMT}$, because mr-mtts are a special case of trgen-tgs.)

Corollary 3.1. $\mathrm{DMM} = \mathrm{DMT_{IO}}$ *and* $\mathrm{D_tMM} = \mathrm{D_tMT}$.

Hence, for deterministic case, mtts and multi-return mtts are equally powerful.

The intuition is that each state in an mr-mtt returning a k-tuple of trees can be split to k states each returning a single tree, and let bindings can be eliminated by simply replacing each let-variable z_i with the state call expression bound to it.

For mr-mtts that are nondeterministic but with their dimensions restricted to 1, we obtain the following characterization from Lemma 2.7 and Lemma 2.10.

Corollary 3.2. 1-MM = LHOM ; $\mathrm{MT}_{\mathrm{IO}}$.

On page 123 of [EV85], a counterexample to show $\mathrm{MT}_{\mathrm{IO}} \subsetneq \mathrm{LHOM}\,;\mathrm{MT}_{\mathrm{IO}}$ is given without proof. (The difficult part of their counterexample to be realized in $\mathrm{MT}_{\mathrm{IO}}$ is the generation of two *identical* pairs of a nondeterministic relabeling of the input, which is similar to our example given later that generates *mutually reverse* pair of nondeterministic relabelings.) Thus, by this example we have the following theorem, which shows that binding intermediate trees by let-expression itself adds expressiveness.

Theorem 3.3. $\mathrm{MT}_{\mathrm{IO}} \subsetneq$ 1-MM.

3.2 The Power of Multi-Return

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:

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

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).

Our claim is that this translation essentially requires the power of tuple-return facility. To see that a naive attempt to realize it by a single-return fails, let us try splitting the state q_1 returning pairs of trees into two single-return states q_{1L} and q_{1R}

by syntactically rewriting every rule.

$$\langle q_{1L}, \mathtt{s}(x)\rangle(y_2) \to \mathsf{let}\ z_1 \leftarrow \langle q_{1L}, x\rangle(\mathtt{A}(y_2))\ \mathsf{in}\ \mathsf{let}\ z_2 \leftarrow \langle q_{1R}, x\rangle(\mathtt{A}(y_2))\ \mathsf{in}\ \mathtt{a}(z_1)$$
$$\langle q_{1L}, \mathtt{s}(x)\rangle(y_2) \to \mathsf{let}\ z_1 \leftarrow \langle q_{1L}, x\rangle(\mathtt{B}(y_2))\ \mathsf{in}\ \mathsf{let}\ z_2 \leftarrow \langle q_{1R}, x\rangle(\mathtt{B}(y_2))\ \mathsf{in}\ \mathtt{b}(z_1)$$
$$\langle q_{1L}, \mathtt{z}\rangle(y_2) \to \mathtt{E}$$
$$\langle q_{1R}, \mathtt{s}(x)\rangle(y_2) \to \mathsf{let}\ z_1 \leftarrow \langle q_{1L}, x\rangle(\mathtt{A}(y_2))\ \mathsf{in}\ \mathsf{let}\ z_2 \leftarrow \langle q_{1R}, x\rangle(\mathtt{A}(y_2))\ \mathsf{in}\ z_2$$
$$\langle q_{1R}, \mathtt{s}(x)\rangle(y_2) \to \mathsf{let}\ z_1 \leftarrow \langle q_{1L}, x\rangle(\mathtt{B}(y_2))\ \mathsf{in}\ \mathsf{let}\ z_2 \leftarrow \langle q_{1R}, x\rangle(\mathtt{B}(y_2))\ \mathsf{in}\ z_2$$
$$\langle q_{1R}, \mathtt{z}\rangle(y_2) \to y_2$$

Then, we have a different translation realized. That is, although the split version still generates trees holding two strings (one with symbols $\mathtt{a}$ and $\mathtt{b}$, and the other with $\mathtt{A}$ and $\mathtt{B}$), 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* 1-dimensional mr-mtt cannot realize this translation. We refer to the translation by *twist* throughout this paper. More precisely, abbreviating $\bar{k}$ for $\overbrace{\mathtt{s}\ldots\mathtt{s}}^{k}\mathtt{z}$, we define

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

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 lemma, which takes *twist* as a witness that the multi-return facility does add expressive power for mr-mtts under nondeterminism.

Lemma 3.4. *twist* $\notin$ 1-MM.

The outline of the proof is as follows. We first reduce the problem to the non-membership of $\mathrm{MT}_{\mathrm{IO}}$, from that of 1-MM, i.e., we show that *twist* $\in$ 1-MM only if *twist* $\in \mathrm{MT}_{\mathrm{IO}}$. Then, we 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 $\Sigma = \{\mathtt{s}^{(1)}, \mathtt{z}^{(0)}\}$, 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., $O = \bigcup_{n\geq 0} O_{\bar{n}}$.

3.2.1 From 1-MM to $\mathrm{MT}_{\mathrm{IO}}$

The only difference of 1-MM and $\mathrm{MT}_{\mathrm{IO}}$ is that the former can bind and copy intermediate trees by let-expressions inside each rule. In other words, if an mr-mtt of dimension 1 has no rule that uses some let-variable z_i twice or more, then there exists an equivalent mtt. For realizing the particular translation *twist*, we indeed do not need parameter copying. This is because, in any output tree $t \in O$ of *twist*, no two different subtrees are equal. If there exists a rule of form $\mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \cdots z \cdots z \cdots$, then during the evaluation either the expression itself or of one occurrence of z must be discarded; otherwise, the final output must contain an identical pair of subtrees generated from the two occurrences of z, which cannot be a member of O.

Lemma 3.5. *Let M be an mr-mtt $(Q, q_0, \Sigma, \Delta, R)$ of dimension 1 realizing twist, and let $r \in R_{q,\sigma}$ be a right-hand side that has a let-variable z that occurs more than once (excluding the binding occurrence). That is, $r = l_1 \cdots l_k \mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa$ with z occurring in κ twice or more. Then, let M' be a new mr-mtt obtained from M by replacing r with the new rule $r' = l_1 \cdots l_k \mathsf{let}\ z_1 \leftarrow f\ \mathsf{in}\ \mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa'$ where κ' is κ whose first occurrence of z is replaced with z_1. Then, if $\tau_M = twist$, we have $\tau_{M'} = twist$.*

Proof. Let us assume a fixed input tree $\bar{n}$. Let $U_{\bar{n}}$ be the set of all subtrees of $O_{\bar{n}}$. That is, $U_{\bar{n}} = O_{\bar{n}} \cup \{(\mathtt{a}|\mathtt{b})^i\mathtt{e} \mid 0 \leq i \leq n\} \cup \{(\mathtt{A}|\mathtt{B})^i\mathtt{E} \mid 0 \leq i \leq n\}$. To prove the lemma, it is sufficient to show the following statement: for any Γ, if $[\![\mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa]\!]^M_\Gamma \subseteq U_{\bar{n}}$

then $[\![\mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa]\!]^M_\Gamma = [\![\mathsf{let}\ z_1 \leftarrow f\ \mathsf{in}\ \mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa']\!]^{M'}_\Gamma$. By definition,

$$[\![\mathsf{let}\ z_1 \leftarrow f\ \mathsf{in}\ \mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa']\!]^M_\Gamma = \bigcup\{[\![\kappa']\!]^M_{\Gamma +\!\!+ (z_1 \mapsto t_1, z \mapsto t)} \mid t_1 \in [\![f]\!]^M_\Gamma, t \in [\![f]\!]^M_\Gamma\}$$

and

$$\begin{aligned}[\![\mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa]\!]^M_\Gamma &= \bigcup\{[\![\kappa]\!]^M_{\Gamma +\!\!+ (z \mapsto t)} \mid t \in [\![f]\!]^M_\Gamma\} \\ &= \bigcup\{[\![\kappa']\!]^M_{\Gamma +\!\!+ (z_1 \mapsto t, z \mapsto t)} \mid t \in [\![f]\!]^M_\Gamma\}\end{aligned}$$

Thus, we obviously have $[\![\mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa]\!]^M_\Gamma \subseteq [\![\mathsf{let}\ z_1 \leftarrow f\ \mathsf{in}\ \mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa']\!]^{M'}_\Gamma$. For the inverse inclusion, we show that it must be the case

$$[\![\kappa']\!]^M_{\Gamma +\!\!+ (z_1 \mapsto t_1, z \mapsto t)} \subseteq [\![\kappa']\!]^M_{\Gamma +\!\!+ (z_1 \mapsto t_1, z \mapsto t_1)} \cup [\![\kappa']\!]^M_{\Gamma +\!\!+ (z_1 \mapsto t, z \mapsto t)}$$

under the assumption $[\![\mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa]\!]^M_\Gamma \subseteq U_{\bar{n}}$. Suppose not. Then, both z_1 and z must contribute to the final output, i.e., $[\![\kappa']\!]^M_{\Gamma +\!\!+ (z_1 \mapsto \square_1, z \mapsto \square)}$ must contain an output tree that has both $\square_1$ and $\square$ in it. However, since this implies that $[\![\kappa']\!]^M_{\Gamma +\!\!+ (z_1 \mapsto t, z \mapsto t)}$ contains a tree that has two subtree equal to t, which contradict the assumption $\subseteq U_{\bar{n}}$. Hence we have $[\![\mathsf{let}\ z_1 \leftarrow f\ \mathsf{in}\ \mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa']\!]^{M'}_\Gamma \subseteq [\![\mathsf{let}\ z \leftarrow f\ \mathsf{in}\ \kappa]\!]^M_\Gamma$. □

Repeating the operation defined in Lemma 3.5 for the innermost let-variables, we can eventually obtain an mr-mtt that has no syntactic variable copying, and therefore have the following lemma.

Lemma 3.6. *twist* $\in$ 1-MM *only if twist* $\in \mathrm{MT}_{\mathrm{IO}}$.

3.2.2 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 state call that takes as arguments sentential forms producing some and only desired outputs.

Definition 3.7. 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* $W^{\bar{n}}_\Lambda \subseteq T_\Lambda$ (recall $\Lambda = \Delta \cup (Q \times T_\Sigma)$) as the minimum set satisfying the following conditions:

- $O_{\bar{n}} \cup T_{\Delta\setminus\{\mathtt{root}\}} \subseteq W^{\bar{n}}_\Lambda$
- $v \in W^{\bar{n}}_\Lambda$ if $v = \langle q^{(k)}, \bar{m}\rangle(v_1, \ldots, v_k)$ and $v_1, \ldots, v_k \in W^{\bar{n}}_\Lambda$ 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 t with no `root`, we do not have to consider other derivations starting from the subtree r, that is, replacing the subtree r by its specific result t will not change the set of outputs from the whole sentential form.

Lemma 3.8. *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 $r \Rightarrow^* t$ where $s \in T_{\Delta\setminus\{\texttt{root}\}}$. Then $\mathcal{C}[r] \equiv \mathcal{C}[t]$.*

Proof. Choose any $\mathcal{D} \in \mathcal{C}\downarrow$. By Prop 2.4, it must be the case that $\mathcal{D}[t] \in \mathcal{C}[r]\downarrow$. Since $\mathcal{C}[r]\downarrow\subseteq O$ and $t \in T_{\Delta\setminus\{\texttt{root}\}}$, the context $\mathcal{D}$ must be either in the form

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

In the first case, it is trivial that, for all $t' \in r\downarrow$, we have $\mathcal{D}[t'] = \mathcal{D} = \mathcal{D}[t]$. In the second case, by Prop 2.3, for all $t' \in r\downarrow$ we have $\mathcal{D}[t'] \in O$. So here, $\mathsf{revUp}(t')$ must be equal to $\mathsf{revUp}(t)$. Since revUp is one-to-one mapping, this implies $t' = t$, and thus we conclude $\mathcal{D}[t'] = \mathcal{D}[t]$. The third case is similar to the second case, and again we have $\mathcal{D}[t'] = \mathcal{D}[t]$ for all $t' \in r\downarrow$. So in any cases, we have $\mathcal{D}[t'] = \mathcal{D}[t]$ for all $\mathcal{D} \in \mathcal{C}\downarrow$ and $s' \in r\downarrow$. So $\mathcal{C}[r]\downarrow= \{\mathcal{D}[t'] \mid \mathcal{D} \in \mathcal{C}\downarrow, t' \in r\downarrow\} = \{\mathcal{D}[t] \mid \mathcal{D} \in \mathcal{C}\downarrow\} = \mathcal{C}[t]\downarrow$ □

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 `root` containing a `root` or an `A` containing a `root`) with a legitimate but nonce tree (`E` in the proof), we will obtain a weak normal form.

Lemma 3.9. *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 $w_{\bar{n}}(r) \in W^{\bar{n}}_\Lambda$ 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 $r_c\downarrow \not\subseteq O_{\bar{n}} \cup T_{\Delta\setminus\{\texttt{root}\}}$. If found, replace r_c in r by `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 `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 $r_c \Rightarrow^* t \in T_{\Delta\setminus\{\mathtt{root}\}}$. 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 3.8, 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 2.4, $r_c{\downarrow} \neq \emptyset$ for all subtrees r_c of r, since $r{\downarrow} \neq \emptyset$. Thus the translated tree $w(r)$ is now contained in $W^{\bar{n}}_\Lambda$. □

3.2.3 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 3.10. 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

$$N^{\bar{n}}_\Lambda = \{v \mid v = \langle q^{(k)}, \bar{m}\rangle(t_1, \ldots, t_k),\ q^{(k)} \in Q, \bar{m} \in T_\Sigma,\\ t_i \in \{\mathsf{dummy}_n\} \cup T_{\Delta\setminus\{\mathtt{root}\}},\ \emptyset \subsetneq v{\downarrow} \subseteq O_{\bar{n}}\}$$

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 3.11. *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 $\mathcal{C}[t_1, \ldots, t_c]{\downarrow} \subseteq O_{\bar{n}}$ for some $t_1, \ldots, t_c \in O_{\bar{n}}$. 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 $\mathcal{C}[t_1, \ldots, t_c]{\downarrow} \subseteq O_{\bar{n}}$, it must be the case $\mathcal{D}[t_1, \ldots, t_c] \in O_{\bar{n}}$ (by Prop 2.3). 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, $\mathcal{D}[t_1, \ldots, t_c]$ contains t_i as a subtree. Hence, the tree $\mathcal{D}[t_1, \ldots, t_c]$ contains at least two `root` nodes, one at the root position and the other at the position of t_i. This implies $\mathcal{D}[t_1, \ldots, t_c] \notin O_{\bar{n}}$, 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 $T_{\Delta \setminus \{\texttt{root}\}}$ 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 v has other procedure calls as arguments, e.g.,

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

(where the arguments are two calls and a tree from $T_{\Delta \setminus \{\texttt{root}\}}$) 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,

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

By repeating this "expansion" on the first and second clauses, the result set of v can be bounded by the union of clauses like the third clause above, that is, the results of any procedure calls appearing in v 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 3.12. *Suppose $M = (Q, q_0, \Sigma, \Delta, R)$ to be an mtt realizing twist. Let $\bar{n} \in T_\Sigma$. There are mappings $N_Q : W_\Lambda^{\bar{n}} \to \mathcal{P}(N_\Lambda^{\bar{n}})$ and $N_V : W_\Lambda^{\bar{n}} \to \mathcal{P}(T_\Delta)$ such that, for any $v \in W_\Lambda^{\bar{n}}$,*

- $v{\downarrow} \subseteq \bigcup\{u{\downarrow} \mid u \in N_Q(v)\} \cup N_V(v)$
- $|N_Q(v)| \leq$ *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:

$$\begin{aligned} N_Q(v) = \{&\mathcal{R}_{v_s}[\mathsf{dummy}_n, \ldots, \mathsf{dummy}_n] \mid \\ &v_s = \langle q^{(k)}, \bar{m}\rangle(v_1, \ldots, v_k) \text{ is a subtree of } v\} \\ N_V(v) = \{&v_\delta \mid v_\delta \text{ is a maximal subtree of } v \text{ labeled by } \Delta\} \end{aligned}$$

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

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

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 $\emptyset \subsetneq v_s{\downarrow} \subseteq O_{\bar{n}}$. So by Lemma 3.11 and Prop 2.4, we have $\mathcal{R}_{v_s}{\downarrow} \subseteq \{\Box_1, \ldots, \Box_k\} \cup O_{\bar{n}}$. Thus, for all $u = \mathcal{R}_{v_s}[\mathsf{dummy}_n, \ldots, \mathsf{dummy}_n] \in N_Q(v)$, 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 $\langle q^{(k)}, \bar{m}\rangle(v_1, \ldots, v_k) \in W_\Lambda^{\bar{n}}$ where $v_1, \ldots, v_k \in W_\Lambda^{\bar{n}}$ and $\emptyset \subsetneq v{\downarrow} \subseteq O_{\bar{n}}$.

$$
\begin{aligned}
&\langle q^{(k)},\bar{m}\rangle(v_1, \ldots, v_k){\downarrow} \\
&\subseteq \{\mathcal{D}[v'_1, \ldots, v'_k] \mid \mathcal{D} \in \{\Box_1, \ldots, \Box_k\} \cup (\mathcal{R}_v{\downarrow} \cap O_{\bar{n}}), v'_i \in v_i{\downarrow}\} \\
&\qquad \text{by Prop 2.4 and Lemma 3.11} \\
&= v_1{\downarrow} \cup \cdots \cup v_k{\downarrow} \cup (\mathcal{R}_v{\downarrow} \cap O_{\bar{n}}) \\
&\subseteq v_1{\downarrow} \cup \cdots \cup v_k{\downarrow} \cup \mathcal{R}_v[\mathsf{dummy}_n, \ldots, \mathsf{dummy}_n]{\downarrow} \\
&\qquad \text{Since } \Box_i \text{ does not contribute any outputs in } O_{\bar{n}}, \text{ substitution with} \\
&\qquad \mathsf{dummy}_n \text{ is safe.} \\
&\subseteq \bigcup_{i=1}^{k} \big(\bigcup \{u{\downarrow} \mid u \in N_Q(v_i)\} \cup N_V(v_i)\big) \cup \mathcal{R}_v[\mathsf{dummy}_n, \ldots, \mathsf{dummy}_n]{\downarrow} \\
&\qquad \text{by I.H.} \\
&\subseteq \bigcup \{u{\downarrow} \mid u \in N_Q(v)\} \cup N_V(v) \text{ by definition of } N_Q \text{ and } N_V
\end{aligned}
$$

□

3.2.4 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 3.14 will show that taking only the representatives leaves only a polynomial number of remainders.

Lemma 3.13. *Suppose $M = (Q, q_0, \Sigma, \Delta, R)$ to be an mtt realizing twist. Let $\bar{n} \in T_\Sigma$. Let $v = \langle q, \bar{m}\rangle(t_1, \ldots, t_k)$ and $v' = \langle q, \bar{m}\rangle(t'_1, \ldots, t'_k)$ both in $N^{\bar{n}}_\Lambda$. Then we have*

$$|v{\downarrow} \setminus v'{\downarrow}| \leq k(k-1)$$

Proof. Let $\mathcal{C}$ be a context $\langle q, \bar{m}\rangle(\Box_1, \ldots, \Box_k)$. Then by Prop 2.3, we have

$$\begin{aligned} v{\downarrow} \setminus v'{\downarrow} &= \{\mathcal{D}[t_1, \ldots, t_k] \mid \mathcal{D} \in \mathcal{C}{\downarrow}\} \setminus \{\mathcal{D}[t'_1, \ldots, t'_k] \mid \mathcal{D} \in \mathcal{C}{\downarrow}\} \\ &\subseteq \{\mathcal{D}[t_1, \ldots, t_k] \mid \mathcal{D} \in \mathcal{C}{\downarrow}, \mathcal{D}[t_1, \ldots, t_k] \neq \mathcal{D}[t'_1, \ldots, t'_k]\} \end{aligned}$$

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 $\mathcal{D}[t_1, \ldots, t_k] \in t{\downarrow} \subseteq O_{\bar{n}}$ and $\mathcal{D}[t'_1, \ldots, t'_k] \in t'{\downarrow} \subseteq O_{\bar{n}}$. So $\mathcal{D}$ must have either one of the following forms:

1. $\mathcal{D} = \Box_i$ for some i such that $t_i = t'_i = \mathsf{dummy}_n$.
2. $\mathcal{D} \in O_{\bar{n}}$.
3. $\mathcal{D} = \mathtt{root}(\mathcal{D}_1[\Box_i], t_\mathcal{D})$ where $\mathcal{D}_1$ is a one-hole context, $t_\mathcal{D} \in T_{\Delta\setminus\{\mathtt{root}\}}$ and for some i such that $t_i, t'_i \in T_{\Delta\setminus\{\mathtt{root}\}}$
4. $\mathcal{D} = \mathtt{root}(t_\mathcal{D}, \mathcal{D}_1[\Box_i])$ where $\mathcal{D}_1$ is a one-hole context, $t_\mathcal{D} \in T_{\Delta\setminus\{\mathtt{root}\}}$ and for some i such that $t_i, t'_i \in T_{\Delta\setminus\{\mathtt{root}\}}$.
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 $t_i, t_j, t'_i, t'_j \in T_{\Delta\setminus\{\mathtt{root}\}}$.

In the case 1 and case 2, trivially $\mathcal{D}[t_1, \ldots, t_k] = \mathcal{D}[t'_1, \ldots, t'_k]$ holds. In the case 3, by the definition of $O_{\bar{n}}$, it must be the case that $\mathsf{revUp}(\mathcal{D}_1[t_i]) = t_\mathcal{D} = \mathsf{revUp}(\mathcal{D}_1[t'_i])$. Since revUp is a one-to-one mapping, we can conclude $t_i = t'_i$. Hence, $\mathcal{D}[t_1, \ldots, t_k] = \mathcal{D}[t'_1, \ldots, t'_k]$. The case 4 is similar.

In the case 5, by definition of $O_{\bar{n}}$, the lengths of $\mathcal{D}_1[t_i]$, $\mathcal{D}_2[t_j]$, $\mathcal{D}_1[t'_i]$, and $\mathcal{D}_2[s'_j]$ are all equal to n. So it must be that $\mathit{length}(t_i) = \mathit{length}(t'_i)$ and $\mathit{length}(t_j) = \mathit{length}(t'_j)$, where $\mathit{length}(t)$ denotes the number of rank-1 nodes in s. Also by the definition of $O_{\bar{n}}$, we have $\mathsf{revUp}(\mathcal{D}_1[t_i]) = \mathcal{D}_2[t_j]$ and $\mathsf{revUp}(\mathcal{D}_1[t'_i]) = \mathcal{D}_2[t'_j]$. So the reverse of t_i is being the prefix of $\mathcal{D}_2[t_j]$.

Here, we consider two cases, namely: the case $\mathit{length}(t_i) + \mathit{length}(t_j) \leq n$ and the case $\mathit{length}(t_i) + \mathit{length}(t_j) > n$. For the former case, $\mathit{length}(t_i)$ is less than or equal to $\mathit{length}(\mathcal{D}_2)$. So t_i must be the reverse of a prefix of $\mathcal{D}_2$. Since exactly the same thing holds for t'_i, we can conclude $t_i = t'_i$, and similarly $t_j = t'_j$. So in this case, again we have $\mathcal{D}[t_1, \ldots, t_k] = \mathcal{D}[t'_1, \ldots, t'_k]$. For the latter case, since $\mathit{length}(t_i) > \mathit{length}(\mathcal{D}_2)$,

the context $\mathcal{D}_2$ is being a prefix of the reverse oft_i. Thus, $\mathcal{D}_2$ is uniquely determined from t_i and the length of t_j. By a similar argument, we also have that $\mathcal{D}_1$ is uniquely determined from t_j and the length of t_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 $\mathcal{D}[t_1, \ldots, t_k] \neq \mathcal{D}[t'_1, \ldots, t'_k]$ is at most $k(k-1)$. This proves the lemma. □

Lemma 3.14. *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 3.13, 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 3.15. *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 3.9. Then we have $S_W \subseteq W_\Lambda^{\bar{n}}$ 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 3.12. Then by the lemma we have $S_Q \subseteq N_\Lambda^{\bar{n}}$, $S_V \subseteq T_\Delta$, 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 3.14 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 lemmas (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 lemmas 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*.

Corollary 3.16. *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 of this section.

Lemma 3.17. *twist* $\notin \mathrm{MT}_{\mathrm{IO}}$

Proof. In order for an mr-mtt M of dimension 1 to realize the *twist* translation, by Lemma 3.5, there must be an mtt M' realizing *twist*. Then, from Corollary 3.16 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. □

Proof of Lemma 3.4. Immediately follows from Lemmas 3.6 and 3.17. □

3.3 Results

As shown in the beginning of the previous section, we have *twist* $\in$ 2-MM. Note that the composition $\mathrm{MT}_{\mathrm{IO}}\,;\mathrm{D_tT}$ can also realize *twist*. Let M be an mtt with the rules

$$\begin{aligned}
\langle q_0, \mathtt{s}(x)\rangle &\to \mathtt{a}(\langle q_1, x\rangle(\mathtt{A}(\mathtt{E}))) & \langle q_1, \mathtt{s}(x)\rangle(y) &\to \mathtt{a}(\langle q_1, x\rangle(\mathtt{A}(y))) \\
\langle q_0, \mathtt{s}(x)\rangle &\to \mathtt{b}(\langle q_1, x\rangle(\mathtt{B}(\mathtt{E}))) & \langle q_1, \mathtt{s}(x)\rangle(y) &\to \mathtt{b}(\langle q_1, x\rangle(\mathtt{B}(y))) \\
\langle q_0, \mathtt{z}\rangle &\to \mathtt{e} & \langle q_1, \mathtt{z}\rangle(y) &\to y
\end{aligned}$$

and N be a total deterministic tt with the rules

$$\begin{array}{lll}
\langle p_0,\mathtt{a}(x)\rangle \to \mathtt{root}(\mathtt{a}(\langle p_1,x\rangle),\langle p_2,x\rangle) & \langle p_1,\mathtt{a}(x)\rangle \to \mathtt{a}(\langle p_1,x\rangle) & \langle p_2,\mathtt{a}(x)\rangle \to \langle p_2,x\rangle \\
\langle p_0,\mathtt{b}(x)\rangle \to \mathtt{root}(\mathtt{b}(\langle p_1,x\rangle),\langle p_2,x\rangle) & \langle p_1,\mathtt{b}(x)\rangle \to \mathtt{b}(\langle p_1,x\rangle) & \langle p_2,\mathtt{b}(x)\rangle \to \langle p_2,x\rangle \\
\langle p_0,\mathtt{A}(x)\rangle \to \mathtt{root}(\mathtt{e},\mathtt{E}) & \langle p_1,\mathtt{A}(x)\rangle \to \mathtt{e} & \langle p_2,\mathtt{A}(x)\rangle \to \mathtt{A}(\langle p_2,x\rangle) \\
\langle p_0,\mathtt{B}(x)\rangle \to \mathtt{root}(\mathtt{e},\mathtt{E}) & \langle p_1,\mathtt{B}(x)\rangle \to \mathtt{e} & \langle p_2,\mathtt{B}(x)\rangle \to \mathtt{B}(\langle p_2,x\rangle) \\
\langle p_0,\mathtt{E}\rangle \to \mathtt{root}(\mathtt{e},\mathtt{E}) & \langle p_1,\mathtt{E}\rangle \to \mathtt{e} & \langle p_2,\mathtt{E}\rangle \to \mathtt{E}.
\end{array}$$

Then we have $\tau_{\mathrm{IO},M}\,;\tau_N = \mathit{twist}$. Hence, by Lemma 3.4, we have the following two theorems; the strict inclusion between 1-MM and 2-MM, and non-closure under right-composition of normal IO-mtts.

Theorem 3.18. $\mathrm{MT}_{\mathrm{IO}} \subsetneq \text{1-MM} \subsetneq \text{2-MM}$.

Theorem 3.19. $\mathrm{MT}_{\mathrm{IO}}\,;\mathrm{D_tT} \not\subseteq \mathrm{MT}_{\mathrm{IO}}$.

Note that *twist* can also be realized by an *OI*-mtt followed by a total deterministic tree transducer, namely, $\mathit{twist} = \tau_{\mathrm{OI},M}\,;\tau_N$ by the above mtt M; it realizes the same translation in both IO- and OI- semantics, because it has no nested state calls. In fact, our proof that *twist* cannot be realized by an IO-mtt can even be applied to OI-mtts with a little modification. Note that Prop 2.3 also holds in OI (a special case of Lemma 3.20 of [EV85]), while Prop 2.4 is not. Instead, we have by Lemma 3.20 of [EV85] the following relation: $\mathcal{C}[r_1,\ldots,r_n]{\downarrow_{\mathrm{OI}}} = \{\mathcal{D} \underset{\mathrm{OI}}{\leftarrow} [r_i{\downarrow_{\mathrm{OI}}},\ldots,r_n{\downarrow_{\mathrm{OI}}}] \mid \mathcal{D} \in \mathcal{C}{\downarrow_{\mathrm{OI}}}\}$ where $\underset{\mathrm{OI}}{\leftarrow} [L_1,\ldots,L_n]$ denotes so-called OI-substitution that substitutes each occurrence of $\Box_i$ with independent element from L_i. In particular, when $\mathcal{D}$ has no occurrence of $\Box_i$, the result of substitution is not necessary empty even if L_i is.

Regarding this difference, we have to modify the definition of weak normal form so that it allows subexpression v to be $v{\downarrow_{\mathrm{OI}}} = \emptyset$. That is, weak normal form is defined for the OI-case as follows:

- $O_{\bar{n}} \cup T_{\Delta\setminus\{\mathtt{root}\}} \subseteq W^{\bar{n}}_{\Lambda}$
- $v \in W^{\bar{n}}_{\Lambda}$ if $v = \langle q^{(k)},\bar{m}\rangle(v_1,\ldots,v_k)$ and $v_1,\ldots,v_k \in W^{\bar{n}}_{\Lambda}$ with $v{\downarrow_{\mathrm{OI}}} \subseteq O_{\bar{n}}$.

Note the difference from IO-case that we do not have the condition "$\emptyset \subsetneq$" now. Then, by the same proof as Lemma 3.9, we can obtain the weak normal form for each OI sentential forms; this is because the only point we have essentially used 2.4 is to prove non-emptiness (the first line of the proof of Lemma 3.8 can also be derived from the OI version of Prop 2.4).

The definition of normal form is modified as follows to allow subexpressions generating the empty set:

$$N^{\bar{n}}_{\Lambda} = \{v \mid v = \langle q^{(k)}, \bar{m}\rangle(t_1, \ldots, t_k),\ q^{(k)} \in Q, \bar{m} \in T_{\Sigma},\\ t_i \in \{\theta\} \cup \{\mathsf{dummy}_n\} \cup T_{\Delta\setminus\{\mathtt{root}\}},\ \emptyset \subsetneq v{\downarrow} \subseteq O_{\bar{n}}\}$$

where θ is an arbitrary sentential form such that $\theta{\downarrow_{\mathrm{OI}}} = \emptyset$ (in the case there is no such sentential form, we first add a new state p with no rules to the mtt, and take $\langle p, \bar{0}\rangle$ as θ). Then by a similar proof as Lemma 3.12, we can construct the normal form. In the construction of the normal form, we change the definition of the context $\mathcal{R}_{v_s}$ as follows:

$$\mathcal{R}_{v_s} = \langle q^{(k)}, \bar{m}\rangle(\alpha_1, \ldots, \alpha_k) \text{ where } \alpha_i = \begin{cases} \theta & \text{if } v_i{\downarrow_{\mathrm{OI}}} = \emptyset \\ \Box_i & \text{if } \emptyset \subsetneq v_i{\downarrow_{\mathrm{OI}}} \subseteq O_{\bar{n}} \\ v_i & \text{otherwise} \end{cases}$$

Then by the same argument as Lemma 3.11, we are able to show that $\mathcal{R}_{v_s}{\downarrow_{\mathrm{OI}}}$ is contained in $\{\Box_1, \ldots, \Box_k\} \cup O_{\bar{n}}$, if intersected with the set of contexts that do not have any occurrence of $\Box_i$ such that $\alpha_i = \theta$. Hence, by the same inductive proof we obtain the normal form. This normal form also yields the same polynomial upperbound on the number of the outputs; the size condition corresponding to Lemma 3.13 of the IO-case is proved by the same (five) case analysis but taking $\mathcal{D}$ as each element from $\mathcal{C}{\downarrow_{\mathrm{OI}}}$ such that there is no occurrence of $\Box_i$ for i with $t_i = \theta$. The subsequent estimation of the number of the outputs goes exactly as same as for the IO-case. Altogether, we have the following result.

Theorem 3.20. $\mathrm{MT}_{\mathrm{OI}}\,;\mathrm{D_tT} \not\subseteq \mathrm{MT}_{\mathrm{OI}}$.

As a concluding remark, we would like to point out that the *twist* translation is an example showing the following theorem, which is conjectured on page 95 of [Voi05].

Theorem 3.21. *Let* LSI *be the class of translations of linear-size increase, i.e.,* $\mathrm{LSI} = \{\tau \mid \exists c \in \mathbb{N} : \forall (s,t) \in \tau : |t| \leq c|s|\}$. $(\mathrm{LSI} \cap \mathrm{MT}_{\mu})\,;(\mathrm{LSI} \cap \mathrm{MT}_{\mu}) \subsetneq (\mathrm{LSI} \cap \mathrm{MT}_{\mu})$. *for* $\mu \in \{\mathrm{IO}, \mathrm{OI}\}$.

Proof. While obviously $\tau_{\mu,M}, \tau_N \in \mathrm{LSI} \cap \mathrm{MT}_{\mu}$, we have $\tau_{\mu,M}\,;\tau_N = \mathit{twist} \notin \mathrm{MT}_{\mu}$. □

Note that for total deterministic mtts of linear-size increase, it is known that the composition hierarchy collapses, i.e., $(\mathrm{LSI} \cap \mathrm{D_tMT})\,;(\mathrm{LSI} \cap \mathrm{D_tMT}) \subseteq (\mathrm{LSI} \cap \mathrm{D_tMT})$ [Man03].

Chapter 4

Complexities on Single MTTs

Macro tree transducers have many decidable properties such as exact typechecking, emptiness and finiteness and membership of their domains as well as ranges. These make mtts a useful device for static verification of XML translation programs. In the algorithms that decide such properties, we sometimes encounter as a sub-problem the "translation membership problem". For a fixed translation, the translation membership problem asks whether a given input/output pair is an element of the translation. Although the problem itself seems simple, effectively solving it is far beyond trivial, in particular if we consider nondeterministic mtts. In this chapter, we investigate the complexity of translation membership problem for both IO- and OI- mtts. For OI-mtts this problem is shown to be NP-complete and in DSPACE(n). On the other hand, translation membership for IO-mtts is shown to be solvable in polynomial time. For several extensions, such as addition of regular look-ahead or the generalization to multi-return mtts, it is shown that translation membership still remains in PTIME.

4.1 Definitions

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

4.2 Translation Membership for OI-MTTs

4.2.1 Lowerbound

The first result is that translation membership for OI-mtts is NP-hard, even for linear mtts. The proof is based on the reduction to 3-SAT, which resembles [Rou73] which shows NP-completeness of the membership problem for indexed languages.

Lemma 4.1. *Translation membership for* $\mathrm{LMT}_{\mathrm{OI}}$ *(and hence* $\mathrm{MT}_{\mathrm{OI}}$*) is NP-hard.*

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

$$
\begin{aligned}
\langle q_0, \mathtt{a}(x_1)\rangle &\to \langle q, x_1\rangle(\mathtt{v}(\mathtt{e}), \mathtt{e}, \neg(\mathtt{e}))\\
\langle q_0, \mathtt{a}(x_1)\rangle &\to \langle q, x_1\rangle(\mathtt{v}(\mathtt{e}), \neg(\mathtt{e}), \mathtt{e})\\
\langle q, \mathtt{b}(x_1, x_2, x_3)\rangle(y_v, y_t, y_f) &\to \langle q, x_1\rangle(\mathtt{v}(y_v), \langle q_c, x_2\rangle(y_t, y_v), \langle q_c, x_3\rangle(y_f, \neg(y_v)))\\
\langle q, \mathtt{b}(x_1, x_2, x_3)\rangle(y_v, y_t, y_f) &\to \langle q, x_1\rangle(\mathtt{v}(y_v), \langle q_c, x_2\rangle(y_t, \neg(y_v)), \langle q_c, x_3\rangle(y_f, y_v))\\
\langle q_c, \mathtt{d}\rangle(y_1, y_2) &\to y_1\\
\langle q_c, \mathtt{d}\rangle(y_1, y_2) &\to y_2\\
\langle q, \mathtt{c}(x_1)\rangle(y_v, y_t, y_f) &\to \wedge(\vee(y_t, y_t, y_t), \langle q, x_1\rangle(y_v, y_t, y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v, y_t, y_f) &\to \wedge(\vee(y_t, y_t, y_f), \langle q, x_1\rangle(y_v, y_t, y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v, y_t, y_f) &\to \wedge(\vee(y_t, y_f, y_t), \langle q, x_1\rangle(y_v, y_t, y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v, y_t, y_f) &\to \wedge(\vee(y_f, y_t, y_t), \langle q, x_1\rangle(y_v, y_t, y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v, y_t, y_f) &\to \wedge(\vee(y_t, y_f, y_f), \langle q, x_1\rangle(y_v, y_t, y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v, y_t, y_f) &\to \wedge(\vee(y_f, y_f, y_t), \langle q, x_1\rangle(y_v, y_t, y_f))\\
\langle q, \mathtt{c}(x_1)\rangle(y_v, y_t, y_f) &\to \wedge(\vee(y_f, y_t, y_f), \langle q, x_1\rangle(y_v, y_t, y_f))\\
\langle q, \mathtt{d}\rangle(y_v, y_t, y_f) &\to \vee(y_t, y_t, y_f)\\
&\vdots \quad \text{(same as the } \vee(\cdots) \text{ part of } \langle q, c\rangle\text{-rules)}\\
\langle q, \mathtt{d}\rangle(y_v, y_t, y_f) &\to \vee(y_f, y_t, y_f).
\end{aligned}
$$

From an input tree $\mathtt{a}(\overbrace{\mathtt{b}(\mathtt{b}(\cdots\mathtt{b}(}^{n}\mathtt{c}^m\mathtt{d}, \mathtt{d}, \mathtt{d})\cdots), \mathtt{d}, \mathtt{d}))$ of size $3n + m + 2$, it generates all satisfiable boolean formulas in 3-conjunctive normal form with n variables and

m conjuncts. The output language encodes boolean formulas as follows: a boolean variable p_i for $0 \leq i < n$ is represented as $\mathtt{v}^i\mathtt{e}$, and three boolean operations $\neg$, $\wedge$, and $\vee$ are represented as they are. For example, the formula $(p_0 \vee \neg p_1 \vee p_2) \wedge (\neg p_0 \vee p_1 \vee p_2)$ is encoded as $\wedge(\vee(\mathtt{e}, \neg\mathtt{ve}, \mathtt{vve}), \vee(\neg\mathtt{e}, \mathtt{ve}, \mathtt{vve}))$.

Intuitively, when the mtt reads the root node of the input, it nondeterministically assigns a truth-value to the first variable p_0. The first $\langle q_0, a\rangle$-rule is the case when it assigned 'true' and the other rule is for 'false'. Three parameters are passed to the state q. Intuitively, the first parameter y_v denotes the name of the next variable to be assigned a truth-value. The second (and the third, respectively) parameter y_t (y_f) denotes the set of 'true' ('false') literals (namely, variables or negated variables) that have been constructed up to now. While reading b nodes in the state q, the mtt nondeterministically assigns a truth-value to each variable p_1 to p_{n-1}, similarly to p_0. Here, OI-nondeterminism is crucially used to represent arbitrary choice of positive and negative literals; each time y_t and y_f are copied to the output, they contain unevaluated "combs" of q_c-calls (on $\mathtt{d}$-nodes). Each such comb represents the nondeterministic choice of any of the positive (y_t) or negative (y_f) literals that have been generated so far. The state q_c means a union of two sets, by taking two parameters and nondeterministically returns either one of them. The parameter y_t is assigned an unevaluated expression, e.g., like $\langle q_c, d\rangle(\langle q_c, d\rangle(\neg p_0, p_1), p_2)$, and each time the value of y_t is needed, it is nondeterministically evaluated to either $\neg p_0$, p_1, or p_2. Then, while reading $\mathtt{c}$ nodes in the input, the transducer generates m conjunctions of 'true' clauses. Since we generate 3-CNF formulas, each clause consists of a disjunction of exactly three literals. There are seven possibilities (all combinations of y_t and y_f, except $\vee(y_f, y_f, y_f)$), which are generated by the $\langle q, \mathtt{c}\rangle$-rules of the transducer.

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

It is also obvious that, given any 3-CNF formula, we can in polynomial time encode the formula to the above explained encoding to obtain t, and count the number of variables and clauses to obtain s. Then, $(s, t) \in \tau_{\mathrm{OI},M}$ if and only if the original formula is satisfiable. It is well known that the satisfiability of 3-CNF is NP-complete (see, e.g., [GJ79]). $\square$

4.2.2 Upperbounds

We prove two more results on the translation membership problem of linear mtts in OI-mode. The first result is that the NP lower bound is tight, i.e., the problem is in fact NP-complete. The second result concerns space complexity. We prove that the problem can be solved in deterministic linear space (DSPACE(n)). Actually, the same upperbound holds also for mtts that are not necessary linear, and moreover, even for finite compositions of mtts. Since the proof for general mtts requires as a basis the complexity result for linear mtts, in this section, we focus only on linear mtts, which have their own interest. The complexity bound of the translation membership for general mtts and their compositions are shown later in Chapter 5.

Overview Compared to the proof of the NP-hardness lowerbound, giving the upperboud complexity result requires much more involved techniques. Our approach to show NP and DSPACE(n) complexity for the translation membership is summarized as follows. First, given a linear mtt M, we convert it to a total deterministic mtt N, whose unique output tree $\tau_N(s)$ symbolically denotes the set of all output trees $\{t \mid t \in \tau_{\mathrm{OI},M}(s)\}$ of M. We call the output trees of τ_N *choice trees*. The translation membership $(s,t) \in \tau_M$ now reduces to the problem that asks whether the choice tree $t' = \tau_N(s)$ contains t. Here, if it were the case $|t'| = O(|s| + |t|)$, we could simply compute t' (using the linear time/space algorithm for total deterministic mtts [Man02]) and then recursively compare with t to obtain NP and DSPACE(n) algorithm. However, this is not necessary the case: $|t'|$ may be much larger than $|s| + |t|$. To overcome this difficulty, we exploit the *compressed representation* of mtt outputs introduced in [MB04, BLM08]. In the compressed representation, each node of the output tree can be represented by a $O(|s|)$ size data structure, no matter how large the actual output tree is. On the compressed representation of [MB04, BLM08], we can obtain its label in constant time and its child nodes in $O(|s|)$ time. In the following, we extend the representation specifically for the case of choice trees generated from linear mtts, and enable obtaining *parent node* also in $O(|s|)$ time, which allows O($|s| + |t|$) space comparison between the choice tree t' and the given output tree t.

Two previous works on the same membership problem for restricted classes of macro tree transducers – for total deterministic mtts [Man02] and for nondeterministic mtts without parameters (top-down tree transducers) [Bak78] – both give DSPACE(n) algorithms. First let us briefly explain where the difficulty arises in our case, i.e., with nondeterminism and parameters. For total deterministic mtts, the DSPACE(n)

complexity is proved via a reduction to the case of linear total deterministic mtts, and then to attribute grammars (which are deterministic by default), whose output languages are LOG(CFL)-complete and therefore have DSPACE($\log(n)^2$) membership test [Eng86]. For nondeterministic tts, the complexity is achieved by a straightforward backtracking-based algorithm; given the input tree s and the output tree t, it generates each possible output of s by simulating the recursive execution of state calls, while comparing with t. The following two facts imply the DSPACE(n) complexity: (1) the depth of the recursion is at most the height of s, and (2) to backtrack we only need to remember for each state call the rule that was applied (which requires constant space). Note that neither (1) nor (2) holds for mtts; the recursion depth can be exponential and the actual parameters passed to each state call must also be remembered for backtracking. Our choice that we first focus only on linear mtts rather than general mtts is, to avoid those difficulties. As we will show shortly, the depth of the recursion linear mtts can carry out is linearly bounded by the size $|s|$ of the input tree, which is much smaller than exponential of $|s|$. Also, the linearity is shown to allow a concise representation of backtracking information. Then in Chapter 5, the complexity for general mtts is given by using further proof techniques.

MTT with Choice and Failure For a technical reason, we prove the result on a slight extension of macro tree transducers, called *macro tree transducers with choice and failure (mttcf)*. We fix the alphabet $C = \{\theta, +\}$ distinct from all other alphabets.

Definition 4.2. An *mtt with choice and failure (mttcf) M* is a tuple $(Q, \Sigma, \Delta, q_0, R)$ defined as for normal mtts, except that the right-hand sides of rules are trees in $T_{\Delta \cup (Q \times X_k) \cup C}(Y_m)$. For $\mu \in \{\mathrm{IO}, \mathrm{OI}\}$, the big-step semantics $[\![\cdots]\!]^M_\mu$, the small-step semantics $\to_{\mu,M}$ and $\downarrow_{\mu,M}$, and the realized translation $(\tau_{\mu,M})$ are defined similarly as for mtts, with additional rules: $[\![+(t_1,t_2)]\!]^M_{\mu,\Gamma} = [\![t_1]\!]^M_{\mu,\Gamma} \cup [\![t_2]\!]^M_{\mu,\Gamma}$, $[\![\theta]\!]^M_{\mu,\Gamma} = \emptyset$, $+(t_1,t_2) \to_{\mu,M} t_1$, and $+(t_1,t_2) \to_{\mu,M} t_2$.

For a right-hand side r of an mttcf, we say a position $i \in V(r)$ is *top-level* if for all proper prefixes j of i, $\mathit{label}(r,j) \in \Delta \cup C$. We say an mttcf is *canonical* if for every right-hand side r and for every top-level position $i \in V(r)$, $\mathit{label}(r,i) \notin C$.

Intuitively, $+$ denotes nondeterministic choice and θ denotes failure (because there is no output tree in $[\![\theta]\!]$). The idea of the choice and failure nodes come from [EV85]; there they show that $\mathrm{MT}_{\mathrm{OI}} = \mathrm{D_t MT}\,;\mathrm{SET}$, where SET is the class of translations

$set_\Delta \in T_{\Delta\cup C} \times T_\Delta$ with

$$\begin{aligned} set_\Delta(\theta) &= \emptyset \\ set_\Delta(+(c_1, c_2)) &= set(c_1) \cup set(c_2) \\ set_\Delta(\delta(c_1, \ldots, c_k)) &= \{\delta(t_1, \ldots, t_k) \mid t_i \in set_\Delta(c_i)\} \qquad \text{for } \delta \in \Delta. \end{aligned}$$

Let us briefly summarize the proof. For any mttcf (or mtt) M, we can always construct a total deterministic mttcf M' that realizes the same translation, by taking the $\langle q,\sigma\rangle$-rule of M' as $\langle q,\sigma(\cdots)\rangle(\cdots) \to +(r_1, +(r_2, \ldots, +(r_n, \theta)\cdots))$ where $\{r_1, \ldots, r_n\} = R_{q,\sigma}$. Also note that the mttcf $M' = (Q, \Sigma, \Delta, q_0, R')$ can be regarded as the mtt $N = (Q, \Sigma, \Delta\cup C, q_0, R')$, by merely interpreting the θ and $+$ nodes as output symbols. Each output of N is a "choice tree" denoting the set of possible output trees. Obviously, the translation set_Δ carries out this interpretation of choice trees, and thus we have $\tau_{\mathrm{OI},M} = \tau_N ; set_\Delta$.

The reason why we introduce mttcfs is twofold. One reason is the decomposition result explained above. The approach of our proof is that we first convert a mttcf into a total deterministic mtt, and calculate its output tree in a compressed form. and then give a membership checking algorithm for the representation. The other reason is its more flexible use of nondeterminism and partiality. Suppose an mttcf rule $\langle q, \sigma(x_1)\rangle(y_1) \to \langle p, x_1\rangle(+(\delta_1, \delta_2))$. This is not necessarily equivalent to the pair of mtt rules $\langle q, \sigma(x_1)\rangle(y_1) \to \langle p, x_1\rangle(\delta_1)$ and $\langle q, \sigma(x_1)\rangle(y_1) \to \langle p, x_1\rangle(\delta_2)$, because the state p may copy its parameter. In other words, normal mtts can behave nondeterministically only at the point of state calls, while mttcfs can introduce nondeterminism at arbitrary positions in the rewriting rules. Note that in general, this does not add any expressive power in terms of the corresponding class of translations; we can still emulate $+$ and θ by introducing auxiliary state calls. However, if we restrict the forms of rules in some particular form, then the existence of choice nodes does make a difference. For example, any mttcf has an equivalent mttcf in a simpler normal form, namely, non-erasure (Section 5.2), while for mtts this is not the case (this is due to the fact that in mttcfs we can freely apply the state-calls 'inline' inside the rules, while not in mtts). Such normal forms allow us much easier construction of 'garbage-free' mttcfs, as we will see later in Chapter 5.

Path-linearity The conversion from a linear mttcf to a total deterministic mttcf do not preserve linearity in general. For example, if we have two rules $\langle q, a(x_1)\rangle \to a(\langle q, x_1\rangle)$ and $\langle q, a(x_1)\rangle \to b(\langle q, x_1\rangle)$ in the original linear mttcf, they are translated into a single rule $\langle q, a(x_1)\rangle \to +(a(\langle q, x_1\rangle), b(\langle q, x_1\rangle))$, in which the variable x_1 occurs

twice. But since the conversion only inserts + and θ nodes, we can easily verify the conversion preserves what we call *path-linearity*. An mttcf is called *path-linear* if a subtree of the form $\langle q, x_i\rangle(\cdots\langle p, x_j\rangle(\cdots)\cdots)$ in its rules implies $i \neq j$. For instance, an mttcf is not linear but is path-linear if there is a right-hand side $\langle q, x_1\rangle(\langle p, x_2\rangle, \langle p, x_2\rangle)$.

Due to the preservation of path-linearity (and non-preservation of linearity) of our determinization, we consider path-linear mtts rather than linear mtts from now on. Also note that, linear mtts and top-down tree transducers are trivially path-linear; in the former all input variables in the right-hand side are distinct and in the latter there are no nested state calls by definition. Thus, by the known decomposition $\mathrm{MT}_{\mathrm{OI}} \subseteq \mathrm{D_tT}\,;\mathrm{LMT}_{\mathrm{OI}}$ (page 138 of [EV85]), we know that 2-fold composition of path-linear OI-mtts are capable of expressing any translation realizable by an OI-mtt. This implies that the class of translations realizable by finite compositions of OI-mtts coincides with that of path-linear OI-mtts. Therefore, when we extend our result to *compositions* of translations in the next Chapter, the class of path-linear mtts is in some sense powerful enough for our purpose.

Before going into the detailed construction of our NP/DSPACE(n) algorithm, here we prove an important lemma about the height property of path-linear mtts.

Lemma 4.3. *Let M be a path-linear mtt. For any pair of trees $(s,t) \in \tau_{\mathrm{OI},M}$, the height of t is at most $H_M \cdot |s|$ where H_M is the maximum height of right-hand sides of M.*

Proof. Let s' be any subtree of s. By induction on the size of s', we prove that the height of the trees in $[\![\langle q, s'\rangle(u_1, \ldots, u_m)]\!]^M_{\mathrm{OI},\Gamma}$ is at most $H_M \cdot |s'| + h$ where h is the height of the highest tree in $\bigcup_i [\![u_i]\!]^M_{\mathrm{OI},\Gamma}$. Let $s' = \sigma(s_1, \ldots, s_k)$ and r be an arbitrary right-hand side in $R_{q,\sigma}$. It is sufficient to show that the height of trees in $[\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},(y_1 \mapsto [\![u_1]\!]^M_\Gamma, \ldots, y_k \mapsto [\![u_k]\!]^M_\Gamma)}$ is at most $H_M \cdot |s'| + h$. Here, on every path from root to leaves of r, there are at most one node labeled $y \in Y$, at most H_M node labeled $\delta \in \Delta$, and by path-linearity, at most one node labeled $\langle q, x_i\rangle$ for each i. Hence, by inductive hypothesis, the height of the resulting trees are at most $h + H_M + \Sigma_i H_M \cdot |s_i|$, which is equal to $h + H_M \cdot |s'|$. □

Compressed Node Representation Let us review the compressed representation of the output trees of a total deterministic mtt introduced in [BLM08]. Let N be a total, deterministic, and path-linear mtt with output alphabet $\Delta \cup C$ and let s be an input tree. Let $E = \{(r, \nu) \mid q \in Q, \sigma \in \Sigma, r \in R_{q,\sigma}, \nu \in pos(r)\}$. For a list $e = (r_0, \nu_0)\ldots(r_n, \nu_n)$ of elements of E, we define $orig(e)$ (the *orig*in of e) as $\epsilon.i_0 \ldots i_{k-1}$

where k is the smallest index satisfying $\mathit{label}(r_k, \nu_k) \notin Q \times X$ (or, let $k = n+1$ when all labels are in $Q \times X$) and i_j is the number such that $\langle q, x_{i_j} \rangle = \mathit{label}(r_j, \nu_j)$ for some q. We call e *well-formed* if $r_i[\nu_i] \in Q \times X$ for $i < n$, $\mathit{label}(r_n, \nu_n) \in \Delta \cup C$, and $\mathit{orig}(e) \in \mathit{pos}(s)$. Intuitively, e is a partial derivation or a "call stack" of the mtt N. Each node of $\tau_N(s)$ can be represented by a well-formed list, which can be stored in $O(|s|)$ space because its length is at most $1 +$ (height of s) and the size of each element depends only on the size of the fixed mtt, not on $|s|$.

Note that e can represent many nodes in $\tau_N(s)$ if the mtt is non-linear in the parameters. For example, consider the mtt M_{dexp} with the following three rules r_0, r_1, and r_2

$$\langle q_0, \mathtt{a}(x) \rangle \to \langle q, x \rangle(\langle q, x \rangle(\mathtt{e})) \qquad (r_0) \qquad\qquad \langle q, \mathtt{e} \rangle(y) \to +(\mathtt{b}(y, y), \mathtt{c}(y, y)) \qquad (r_2)$$
$$\langle q, \mathtt{a}(x) \rangle(y) \to \langle q, x \rangle(\langle q, x \rangle(y)) \qquad (r_1)$$

and the input tree $s_3 = \mathtt{a}(\mathtt{a}(\mathtt{a}(\mathtt{e})))$, the list $(r_0, \epsilon.1)(r_1, \epsilon.1)(r_1, \epsilon.1)(r_2, \epsilon.1)$ represents all $\mathtt{b}$-nodes at depth 16 of the tree $\tau_{M_{\mathrm{dexp}}}(s_3)$, of which there are 2^8 many.

The label *c-label*(e) of the node represented by e is $\mathit{label}(r_n, \nu_n)$. The operation *c-child*(e, i) which calculates the representation of the i-th child of the node represented by e is defined in terms of the following three operations. For a well-formed list $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ with $\mathit{rank}(\textit{c-label}(e)) = m$, we define $\mathit{down}_i(e)$ for $1 \leq i \leq m$ as $(r_0, \nu_0) \dots (r_n, \nu_n.i)$. For a list $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ where $\mathit{label}(r_j, \nu_j (= \langle q_{i_j}, x_{k_j} \rangle \in Q \times X$ for every $0 \leq j \leq n$, we define $\mathit{expand}(e) = (r_0, \nu_0) \dots (r_n, \nu_n)(r_{n+1}, \epsilon)$ where r_{n+1} is the right-hand side of the unique $\langle q_{i_n}, \mathit{label}(s, \epsilon.k_0 \dots k_n) \rangle$-rule. For $e = (r_0, \nu_0) \dots (r_n, \nu_n)$ such that $\mathit{label}(r_n, \nu_n) = y_i \in Y$, we define $\mathit{pop}(e) = (r_0, \nu_0) \dots (r_{n-1}, \nu_{n-1}.i)$. Then, the operation *c-child*(e, i) is realized by the following algorithm. First apply down_i to e, then repeatedly apply *pop* and *expand* as long as possible. The repetition is assured to terminate in $O(|s|)$ steps for a path-linear mtt, by the following proposition.

Proposition 4.4. *Let $[e_0, e_1, \dots, e_n]$ be a list of lists satisfying the condition $e_{i+1} = \mathit{pop}(e_i)$ or $e_{i+1} = \mathit{expand}(e_i)$ for every $i < n$. Let $0 \leq i < j \leq n$, $m \in \mathbb{N}$, $e_i = (r_0, \nu_0) \dots (r_m, \nu_m)$, $e_j = (r'_0, \nu'_0) \dots (r'_m, \nu'_m)$, and $\mathit{orig}(e_i) = \mathit{orig}(e_j)$. Then $r_k = r'_k$ and $\nu_k = \nu'_k$ for all $k < m$, $r_m = r'_m$, and ν_m is a proper prefix of ν'_m.*

Proof. Let k be the largest index $i < k \leq j$ such that e_k is the shortest list among e_{i+1} to e_j, and $l + 1$ be the length of e_k. Then by definition of *pop* and *expand*, e_k must be of the form $(r_0, \nu_0) \dots (r_{l-1}, \nu_{l-1})(r_l, \nu_l.p)$ for some p.

MATCH (e, v)

1: **while** *label*$(e) = +$ **do**
2: $\quad e \leftarrow$ *c-child*(e, k) **where** k = 1 or 2, nondeterministically chosen
3: **if** *c-label*$(e) \neq$ *label*(v) **then**
4: $\quad$ **return** false
5: **else if** *rank*(*label*$(v)) = 0$ **then**
6: $\quad$ **return** true
7: **else**
8: $\quad$ **for** $i = 1$ **to** *rank*(*label*(v)) **do**
9: $\quad\quad$ **if not** MATCH(*c-child*(e, i), *child*(v, i)) **then**
10: $\quad\quad\quad$ **return** false
11: $\quad$ **return** true

Figure 4.1: Algorithm MATCH

There are two cases: $k < j$ or $k = j$. First, we show that it cannot be the case $k < j$ in fact. Suppose $k < j$. Since we took k so that e_k is the shortest and the rightmost list among $e_{i+1}, \ldots, e_j$, the sequence of *pop*'s and *expand*'s applied to obtain e_j from e_k preserves the prefix e_k unchanged. Thus, e_k must be a proper prefix of e_j. Note that, by path-linearity, the input variable at the position ν_l and the position $\nu_l.p$ in the rule r_l is different. Therefore in this case, *orig*(e_i) cannot be equal to *orig*(e_j).

Hence, it must be the case $k = j$. In this case, obviously the statement of the lemma holds. □

The proposition means that the length of such a list $[e_0, e_1, \ldots, e_n]$ can be at most $H_N \cdot |s|$ where H_N is the maximum depth of right-hand sides of the rules, because each e_i and e_j must have either a different *orig*in, or the same *orig*in Hence, *c-child* runs in linear time with respect to $|s|$. Similarly, the representation of the root of $\tau_N(s)$ is obtained in polynomial time by repeatedly applying *pop* and *expand* as long as possible to $e_0 = (r_0, \epsilon)$ where r_0 denotes the right-hand side of the unique $\langle q_0, \mathit{label}(s, \epsilon)\rangle$-rule.

Matching Algorithm with NP Time Complexity Let $t \in T_\Delta$. Figure 4.2.2 shows the nondeterministic algorithm MATCH that decides, given a well-formed list e and a node v of t, whether the set of trees represented by the choice tree at e contains the subtree of t rooted at v. The operations *c-label* and *c-child* are defined as above. The operations *label*, *rank*, and *child* are basic tree operations, assumed to run in

polynomial time with respect to $|t|$. If we apply MATCH to the representations of the root nodes of $\tau_N(s)$ and $v = \epsilon$, we can decide whether $(s,t) \in \tau_M$. Since this is the standard top-down recursive comparison of two trees, the correctness of the algorithm should be clear.

In each nondeterministic computation, MATCH is called once for each node of t. In each call, the while-loop iterates at most $c|s|$ time for a constant c. This is because, due to the linear-height property of path-linear mtts; by Lemma 4.3 the height of the output tree is bounded by $c|s|$ for some constant c. Altogether, the total running time is polynomial in $|s| + |t|$.

Linear Space Complexity If naively implemented, the MATCH algorithm takes $O((|s| + \log|t|)|t|)$ space, because in the worst case the depth of recursion is $O(|t|)$ and we have to remember e (which costs $O(|s|)$ space) and v ($O(\log(|t|))$ space at least, depending on the tree node representation) in each step of the recursion. However, note that in the algorithm we need to traverse both trees only in depth-first and left-to-right order. If we can calculate the parent and siblings of each node, the MATCH algorithm can be easily rewritten in tail-recursive form, even without nondeterminism (See Figure 4.2.2).

Unfortunately, our compressed representation e of a node cannot support *c-parent* or *c-nextSibling* directly; recall that one list may represent many nodes at the same time. It is inherently ambiguous. One way to eliminate this ambiguity is to represent each node not by a single list but by a list of lists, i.e., instead of remembering single node in a compressed form, remember all the nodes in the path from the root to the node. You may think that it then takes superlinear space with respect to $|s|$ – each compressed node takes linear space, and the length of the path cannot be bound by a constant. However, note that the lists of nodes share common prefixes! Suppose the root node is represented by $(r_0,\nu_0)(r_1,\nu_1)(r_2,\nu_2)(r_3,\nu_3)$ and its child node is obtained by applying $down_1$, *pop*, and *expand*. Then the child node is of the form $(r_0,\nu_0)(r_1,\nu_1)(r_2,\nu_2')(r_3',\nu_3')$, which shares the first two elements with the root node representation. We show that if we store lists of nodes with common prefixes maximally shared, then, in the case of path-linear mtts, their space consumption becomes $O(|s|+|t|)$. The idea of sharing lists resembles the proof of context-sensitivity of indexed languages [Aho68].

We encode a list of well-formed lists as a tree, written in parenthesized notation on the tape. For example, the list of three lists $[\rho_1\rho_2\rho_3, \rho_1\rho_2\rho_4, \rho_1\rho_5\rho_6]$ is encoded as $\rho_1(\rho_2(\rho_3,\rho_4),\rho_5(\rho_6))$. Since the number of parentheses is $\leq 2n$ and that of commas is

MATCH-TAILREC (e, v)

1: **return** CONTAINS?(e, v)

CONTAINS? (e, v)

1: **if** *c-label*$(e) =$ *label*(v) **then**
2: **if** *rank*(*label*$(v)) = 0$ **then**
3: **return** MATCHED(e, v)
4: **else**
5: **return** CONTAINS?(*c-child*$(e, 1)$, *child*$(v, 1)$)
6: **else if** *c-label*$(e) = +$ **then**
7: **return** CONTAINS?(*c-child*$(e, 1), v$)
8: **else**
9: **return** FAILED(e, v)

MATCHED (e, v)

1: **if** *isRoot*(e) **then**
2: **return** true
3: **else if** *c-label*(*c-parent*$(e)) = +$ **then**
4: **return** MATCHED(*c-parent*$(e), v$)
5: **else if** *isLastchild*(e) **then**
6: **return** MATCHED(*c-parent*(e), *parent*(v))
7: **else**
8: **return** CONTAINS?(*c-nextSibling*(e), *nextSibling*(v))

FAILED (e, v)

1: **if** *isRoot*(e) **then**
2: **return** false
3: **else if** *c-label*(*c-parent*$(e)) = +$ **and not** *isLastchild*(e) **then**
4: **return** CONTAINS?(*c-nextSibling*$(e), v$)
5: **else if** *c-label*(*c-parent*$(e)) = +$ **and** *isLastchild*(e) **then**
6: **return** FAILED(*c-parent*$(e), v$)
7: **else**
8: **return** FAILED(*c-parent*(e), *parent*(v))

Figure 4.2: Algorithm MATCH-TAILREC

$\leq n$ where n denotes the number of nodes, the size of this representation is $O(n)$. When we add a new node e to the end of the list, the addition is represented as an addition to the rightmost path. As an example, let $e = \rho_1\rho_5\rho_7\rho_8$. The common prefix $\rho_1\rho_5$ with the current rightmost path $\rho_1\rho_5\rho_6$ is shared, and the suffix $\rho_7\rho_8$ is added as the rightmost child of the ρ_5-node. Then, we have a new tree $\rho_1(\rho_2(\rho_3, \rho_4), \rho_5(\rho_6, \rho_7(\rho_8)))$. Removal of the last list, which means we are going up to the parent node, is the reverse operation of addition; the rightmost leaf and its ancestors that have only one descendant leaf are removed. Note that, since by definition a well-formed list cannot be a prefix of any other well-formed lists, each well-formed list always corresponds to a leaf node of the tree. It should be straightforward to implement these two operations in linear space.

Let us consider what happens if we apply this encoding to the output of a *path-linear* mtt. Note that we always apply the list-of-list representation to a path in the tree, i.e., the lists $[e_0, e_1, \ldots, e_n]$ we have to store always satisfy the relation $e_j \in \mathit{c\text{-}child}^+(e_i)$ for every $i < j$. Let $e = (r_0, \nu_0) \ldots (r_m, \nu_m)$ and $e' = (r'_0, \nu'_0) \ldots (r'_m, \nu'_m)$ be proper prefixes of different elements in the same list satisfying the condition (here we assume that e is taken from the element preceding the one where e' is taken). Then, $\mathit{orig}(e) = \mathit{orig}(e')$ only if $e = e'$. This can be proved by contradiction. Suppose $\mathit{orig}(e) = \mathit{orig}(e')$ and $e \neq e'$, and the j-th elements are the first difference between e and e'. Then, by the condition that e' is obtained by repeatedly applying *c-child* to e, it must be the case that $r_j = r'_j$ and ν_j is a proper prefix of ν'_j. However, due to path-linearity, the input variable at ν_j and ν'_j must be different, which contradicts $\mathit{orig}(e) = \mathit{orig}(e')$. Therefore, we can associate a unique node in $\mathit{pos}(s)$ with each proper prefix of the lists, which means that the number of distinct proper prefixes is at most $|s|$. Similarly, it can be shown that adding only to the rightmost path is sufficient for maximally sharing all common prefixes. Suppose not, then there must be in the list three nodes of the forms $e_1 = e.(r, \nu).e'_1$, $e_2 = e.(r, \nu').e'_2$, and $e_3 = e.(r, \nu).e'_3$ with $\nu \neq \nu'$ in this order. Note that if this happened, then the prefix $e.(r, \nu)$ would not be shared by the rightmost addition. However, $e_2 \in \mathit{c\text{-}child}^+(e_1)$ implies that ν is a proper prefix of ν', and by $e_3 \in \mathit{c\text{-}child}^+(e_2)$, ν' is a proper prefix of ν, which is a contradiction. Hence, the number of nodes except leaves in the tree encoding equals the number of distinct proper prefixes, which is at most $|s|$. We can bound the number of leaves by $|t|$, the maximum depth of the recursion. So, the size of the tree encoding of a list of nodes is $O(|s| + |t|)$.

Theorem 4.5. *Let M be a path-linear mttcf. There effectively exists a nondetermin-*

istic Turing machine which, given any s and t as input, determines whether or not $(s,t) \in \tau_{\mathrm{OI},M}$ in polynomial time with respect to $|s|+|t|$. Also, there effectively exists a deterministic Turing machine that determines $(s,t) \in \tau_{\mathrm{OI},M}$ in $O(|s|+|t|)$ space.

4.3 Tractable Classes

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

The idea of the proof is based on inverse type inference for mtts M (Theorem 7.4 of [EV85]); given a finite tree automaton $\mathcal{B}$ (accepting output trees), we can effectively construct a finite tree automaton that recognizes the corresponding input trees $\tau^{-1}_{\mathrm{IO},M}(L(\mathcal{B}))$. The technique is widely used for exact typechecking of XML translations (see, e.g., [Toz01, MSV03, MBPS05, FH07]). Now, given an output tree t, by constructing its minimal dag representation (i.e., the pointer representation of t such that all isomorphic subtrees are shared), we can simply consider it as the trivial deterministic automaton $\mathcal{B}_t$ with at most $|t|$-many states which recognizes $\{t\}$. Once we have constructed the automaton $\mathcal{A}$ for $\tau^{-1}_{\mathrm{IO},M}(L(\mathcal{B}_t))$, we merely need to check whether $s \in L(\mathcal{A})$, in order to solve translation membership for (s,t). However, the automaton $\mathcal{A}$ can be very large: its worst case number of states is exponential in $|\mathcal{B}_t|$. Thus, we must avoid to fully construct $\mathcal{A}$ in order to obtain PTIME complexity. Our idea is to construct $\mathcal{A}$ on demand, while running it on the tree s. Note that inverse type inference of an IO-mtt constructs an input type automaton which has states that are functions p from Q to $(V^m \to 2^V)$ where V is the set of states of $\mathcal{B}_t$, Q is the set of states of M, and m is the maximum rank of states in Q. Such a state p tells us for each $q \in Q$, which state of $\mathcal{B}_t$ is obtained if we apply the state q to an input tree. That is, if $\mathcal{A}$ reaches the state p after reading a tree s, it means that running $\mathcal{B}_t$ on output trees in $\langle q, s\rangle(t|_{v_1}, \ldots, t|_{v_m})$ obtains the states $(p(q))(v_1, \ldots, v_m)$.

Theorem 4.6. *Let M be an mtt. Translation membership for $\tau_{\mathrm{IO},M}$ can be determined in time $O(|s| \cdot |t|^{2m+2} \cdot |M|)$ where m is the maximum rank of M's states.*

Proof. Let t_{dag} be the minimal dag representing t. It is folklore that t_{dag} can be computed in amortized linear time in $|t|$, using hashing, and even in linear time using pseudo radix sorting, see [ST80]. Let V_t be the set of nodes of t_{dag}. We define $\mathit{label}(v)$ to denote the label in Σ of the node $v \in V_t$, and $\mathit{child}(v,i)$ to denote the i-th child node of v. Assuming a standard pointer structure representing dags, we regard each execution of *label* and *child* takes $O(1)$ time.

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

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

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

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

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

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

$$\begin{aligned}
f_{\vec{v},\vec{a}}(y_i, v') &= true && \text{if } v' = v_i \\
f_{\vec{v},\vec{a}}(y_i, v') &= false && \text{if } v' \neq v_i \\
f_{\vec{v},\vec{a}}(\delta(r_1, \ldots, r_n), v') &= label(v') = \delta \wedge \bigwedge_{1 \leq i \leq n} f_{\vec{v},\vec{a}}(r_i, child(v', i)) && \text{if } v' \in V_t \\
f_{\vec{v},\vec{a}}(\delta(r_1, \ldots, r_n), \perp) &= \exists \vec{u} \in V^n : \Big(\bigwedge_{1 \leq i \leq n} f_{\vec{v},\vec{a}}(r_i, u_i) \\
&\qquad \wedge \forall u' \in V_t : \big(label(u') \neq \delta \vee \bigvee_{1 \leq i \leq n} child(u', i) \neq u_i \big) \Big) \\
f_{\vec{v},\vec{a}}(\langle q', x_j \rangle(r_1, \ldots, r_n), v') &= \exists \vec{u} \in V^n : \Big((q', \vec{u}, v') \in a_j \wedge \bigwedge_{1 \leq i \leq n} f_{\vec{v},\vec{a}}(r_i, u_i) \Big).
\end{aligned}$$

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

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

Claim . For every input tree s', we have the following equation for all $q \in Q$, $r_i \in T_{\Delta \cup (Q \times T_\Sigma) \cup Y}$, and an environment Γ:

$$\begin{aligned}
&\rho\Big([\![\langle q, s' \rangle(r_1, \ldots, r_n)]\!]^M_{\mathrm{IO},\Gamma} \Big) \\
&\qquad = \Big\{ v' \;\Big|\; (q, (v_1, \ldots, v_n), v') \in run(s'), v_i \in \rho([\![r_i]\!]^M_{\mathrm{IO},\Gamma}) \text{ for all } i \Big\}
\end{aligned}$$

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

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

$$\rho\Big([\![\langle q, s'\rangle(r_1, \ldots, r_n)]\!]^M_{\mathrm{IO},\Gamma}\Big) = \bigcup_{r \in R_{q,\sigma}} \Big\{\rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi}) \mid \Xi(y_i) \in [\![r_i]\!]^M_{\mathrm{IO},\Gamma}\Big\}$$

and by definition of run, we have

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

where $\vec{a} = (run(s_1), \ldots, run(s_k))$. To show these two sets are equal, it is sufficient to prove the the following statement: if $\rho(\Xi(y_i)) = v_i$ then $\rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi}) = \{v' \mid f_{\vec{v},\vec{a}}(r, v')\}$. The proof is by inner induction on the structure of r.

If $r = y_i$, we have $\rho([\![y_i]\!]^M_{\mathrm{IO},\Xi}) = \{v_i\}$ and $\{v' \mid f_{\vec{v},\vec{a}}(r, v')\} = \{v' \mid v' = v_i\} = \{v_i\}$, thus they are equal.

If $r = \langle q', x_i\rangle(r_1, \ldots, r_n)$, we have $\{v' \mid f_{\vec{v},\vec{a}}(\langle q', x_i\rangle(r_1, \ldots, r_n), v')\} = \{v' \mid (q', \vec{u}, v') \in a_i, f_{\vec{v},\vec{a}}(r_i, u_i)$ for all $i\}$, which is by inner induction hypothesis equal to $\{v' \mid (q', \vec{u}, v') \in a_i, u_i \in \rho([\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi}$ for all $i\}$, and then by outer induction hypothesis equal to $\rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi})$.

If $r = \delta(r_1, \ldots, r_n)$, we consider two cases. First, we show for any $v' \in V_t$ that $v' \in \rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi})$ if and only if $f_{\vec{v},\vec{a}}(r, v')$. If $label(v') \neq \delta$, it obviously holds. If $label(v') = \delta$, we have $v' \in \rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi})$ if and only if $v' = \rho(\delta(t_1, \ldots, t_m))$ for some $t_i \in [\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi}$, This is equivalent to $child(v', i) \in \rho([\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi})$ for all i, which is by inner induction hypothesis equivalent to $f_{\vec{v},\vec{a}}(r, child(v', i))$, as desired. Second, we show $\bot \in \rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi})$ if and only if $f_{\vec{v},\vec{a}}(r, \bot)$. Note that we have $\bot \in \rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi})$ if and only if there exists a tree $t' \in [\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi}$ that is not a subtree of t. By the definition of IO-semantics, this is equivalent to the following condition: there exists $t_i \in [\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{IO},\Xi}$ for each i such that $\delta(t_1, \ldots, t_n)$ is not a subtree of t, i.e., $\rho(\delta(t_1, \ldots, t_n)) = \bot$. It holds if and only if for any δ-labeled node $u' \in V_t$ there exists i such that $child(u', i) \neq \rho(t_i)$. The reader should be able to verify that if we take $\vec{u}$ as $(\rho(t_1), \ldots \rho(t_n))$, this is exactly the definition of $f_{\vec{v},\vec{a}}(r, \bot)$.

The time complexity for testing $(q_0, (), v_\epsilon) \in run(s)$ is computed as follows. The value $run(s)$ for the whole input tree s can be computed by executing the tr function on each node of s. The computation is done in bottom-up fashion as bottom-up

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

The reader may wonder why the same approach does not work for OI-mtts, whose inverses also preserve the regular tree languages. The problem is, for OI, the states of the inferred automata are in $A = 2^{\bigcup_i Q^{(i)} \times (2^V)^i \times V}$ instead of $A = 2^{\bigcup_i Q^{(i)} \times V^i \times V}$. The difference is intuitively explained as follows: in IO-mtts, every copy of a same parameter is an identical output tree and thus corresponds to a single node in V, while in OI-mtts, each copy is evaluated independently and thus may correspond to different output nodes. To capture this phenomenon in the inverse type inference, each parameter must be represented by a *set of* nodes rather than a *single* output node. The additional exponential implies that a single state in A (a subset of $\bigcup_i Q^{(i)} \times (2^V)^i \times V$) can already be exponentially large. Therefore, on-the-fly construction does not help to obtain a PTIME algorithm. Of course, Lemma 4.1 implies that there is no PTIME algorithm for translation membership for OI-mtts (unless NP=P).

Nevertheless, some subclasses of OI-mtts still admit PTIME translation membership. Note that the essential difficulty of OI-translation membership comes from the copying of parameters. Consider, for example, an OI-mtt that is linear in the parameters (i.e., in every right-hand side each parameter y_i occurs at most once); then each parameter is either used once or is never used. In this case, it can be represented in the inverse-type automaton by a set of size ≤ 1. More generally, if an OI-mtt is *finite copying* in the parameter, its translation membership can be tested in polynomial time. An mtt is finite copying in the parameter if there exists a constant c such that

for any q, s, and $u \in [\![\langle q, s\rangle(y_1, \ldots, y_k)]\!]$, the number of occurrences of y_i in u is no more than c; the number c is called a (parameter) *copying bound* by M. Note that "linear-in-parameter" mtts are a special case of finite copying mtts; they are not only finite copying with copying bound 1, but also the finiteness can be known by simply counting the number of syntactic occurrences of each variable in the rules, while finite copying in general is a semantic property of mtts. Also note that finite copying is a decidable property, and the copying bound of an mtt can be effectively obtained. (See Lemma 4.10 of [EM03b]. Although it is proved only for total deterministic mtts, the same technique also works for IO- and OI- nondeterministic mtts.)

Theorem 4.7. *Let M be an mtt that is finite copying in the parameter with copying bound c. Then, translation membership for $\tau_{\mathrm{OI},M}$ can be determined in time $O(|s| \cdot |t|^{c(2m+2)} \cdot c \cdot |M|)$ where m is the maximum rank of M's states.*

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

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

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

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

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

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

$$\begin{aligned}
f_{\vec{\beta},\vec{a}}(y_i, v') &= \mathit{true} && \text{if } v' \in \beta_i \\
f_{\vec{\beta},\vec{a}}(y_i, v') &= \mathit{false} && \text{if } v' \notin \beta_i \\
f_{\vec{\beta},\vec{a}}(\delta(r_1, \ldots, r_n), v') &= \bigwedge_{1 \leq i \leq n} f_{\vec{\beta},\vec{a}}(r_i, \mathit{child}(v', i)) && \text{if } \mathit{label}(v') = \delta \\
f_{\vec{\beta},\vec{a}}(\delta(r_1, \ldots, r_n), v') &= \mathit{false} && \text{if } \mathit{label}(v') \neq \delta \\
f_{\vec{\beta},\vec{a}}(\langle q', x_j\rangle(r_1, \ldots, r_n), v') &= \\
&\exists \vec{\gamma} : \Big((q', \vec{\gamma}, v') \in a_j \wedge \bigwedge_{1 \leq i \leq n} \forall u \in \gamma_i : f_{\vec{\beta},\vec{a}}(r_i, u)\Big).
\end{aligned}$$

Note that we do not have the $\perp$ element in V this time. Instead, the empty set $\emptyset$ plays the same role. The complexity of this algorithm is computed similarly to the case of IO-mtts: we need to test by $f_{\vec{\beta},\vec{a}}$ all combinations of $a \in \bigcup_i Q^{(i)} \times \mathcal{P}_c(V)^i \times V$ (which

is of size $O(|Q| \cdot |V|^{cm+1})$ this time) and $r \in R_{q,\sigma}$, then $f_{\vec{\beta},\vec{a}}$ receives $|r| \cdot |V|$ different pairs of arguments, and finally the computation of $f_{\vec{v},\vec{a}}(\langle q', x_j\rangle(\cdots))$ takes $O(|V|^{cm} \cdot c)$ time where $|V|^{cm}$ comes from the part "$\exists\vec{\gamma}$" and c comes from the part "$u \in \gamma_i$". The correctness is shown by proving the following claim.

Claim . For every input tree s', we have $\rho\Big([\![\langle q, s'\rangle(r_1, \ldots, r_n)]\!]^M_{\mathrm{OI},\Gamma}\Big) = \Big\{v' \;\Big|\; (q, (\beta_1, \ldots, \beta_n), v') \in \mathit{run}(s'), \beta_i \subseteq \rho([\![r_i]\!]^M_{\mathrm{OI},\Gamma}) \text{ for all } i\Big\}$ for all $q \in Q$, $r_i \in T_{\Delta\cup(Q\times T_\Sigma)\cup Y}$, and an environment Γ.

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

$$\rho\Big([\![\langle q, s'\rangle(r_1, \ldots, r_n)]\!]^M_{\mathrm{OI},\Gamma}\Big) = \bigcup_{r\in R_{q,\sigma}} \Big\{\rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi}) \mid \Xi(y_i) = [\![r_i]\!]^M_{\mathrm{OI},\Gamma}\Big\}$$

and moreover, by finite-copying property, we know that during the evaluation of $[\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi}$, the elements of $\Xi(y_i)$ is used at most c times for each y_i. Hence, it must be the case $\rho\big([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi}\big) = \bigcup\big\{\rho\big([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Theta}\big) \mid \exists\Theta : \Theta(y_i) \subseteq \Xi(y_i) \cap \mathcal{P}_c(T_\Delta)\big\}$. By definition of *run*, we have

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

where $\vec{a} = (\mathit{run}(s_1), \ldots, \mathit{run}(s_k))$. To show these two sets are equal, it is sufficient to prove the the following statement: if $\rho(\Theta(y_i)) = \beta_i$ then $\rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Theta}) = \{v' \mid f_{\vec{\beta},\vec{a}}(r, v')\}$. The proof is by inner induction on the structure of r.

If $r = y_i$, we have $\rho([\![y_i]\!]^M_{\mathrm{OI},\Theta}) = \rho(\Theta(y_i)) = \beta_i$ and $\{v' \mid f_{\vec{\beta},\vec{a}}(r, v')\} = \{v' \mid v' \in \beta_i\} = \beta_i$, thus they are equal.

If $r = \langle q', x_j\rangle(r_1, \ldots, r_n)$, we have $\{v' \mid f_{\vec{\beta},\vec{a}}(\langle q', x_i\rangle(r_1, \ldots, r_n), v')\} = \big\{v' \mid \exists\vec{\gamma} : \big((q', \vec{\gamma}, v') \in a_j \wedge \bigwedge_{1\le i\le n} \forall u \in \gamma_i : f_{\vec{\beta},\vec{a}}(r_i, u)\big)\big\}$, which is by inner induction hypothesis equal to $\big\{v' \mid \exists\vec{\gamma} : \big((q', \vec{\gamma}, v') \in a_j, \gamma_i \subseteq \rho([\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Theta})\big\}$, and then by outer induction hypothesis equal to $\rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Theta})$.

If $r = \delta(r_1, \ldots, r_n)$ and $\mathit{label}(v') \neq \delta$, then both $\rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Theta})$ and $\{v' \mid f_{\vec{\beta},\vec{a}}(r, v')\}$ are empty and thus equal. If $r = \delta(r_1, \ldots, r_n)$ and $\mathit{label}(v') = \delta$, we have $v' \in \rho([\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Theta})$ if and only if $v' = \rho(\delta(t_1, \ldots, t_m))$ for some $t_i \in [\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Theta}$. This is equivalent to $\mathit{child}(v', i) \in \rho([\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Theta})$ for all i, which is by inner induction hypothesis equivalent to $f_{\vec{v},\vec{a}}(r, \mathit{child}(v', i))$, as desired. □

On the other hand, the PTIME result for IO-mtts can be generalized to a more powerful extension of IO-mtts. One popular way to extend mtts is by *regular look-ahead*. Mtts with regular look-ahead are equipped with one deterministic bottom-up

tree automaton and are allowed to select a rule with respect to the state of the tree automaton, in addition to the current state and the label of the current node. Since any $\mathrm{MT_{IO}}$'s with regular look-ahead can be simulated by a normal $\mathrm{MT_{IO}}$ (Theorem 5.19 of [EV85]), the translation membership for $\mathrm{MT_{IO}}$ with regular look-ahead is also in PTIME. In fact, we can further extend the model to use a more expressive model of look-ahead, namely, tree automata with equality and disequality constraints [BT92], while still preserving the PTIME translation membership.

Definition 4.8. A *bottom-up tree automaton with equality and disequality constraints* (TAC) is a tuple $B = (P, \Sigma, \delta)$, where P is the set of states, Σ the input alphabet, and δ is a set of transitions of the form $(\sigma^{(m)}, p_1, \ldots, p_m, E, D, p)$ where $E, D \subseteq \{1, \ldots, m\}^2$ are the sets of equality and disequality constraints, respectively. A list of trees $t_1, \ldots, t_m$ is said to satisfy the constraints if $\forall (i,j) \in E : t_i = t_j$ and $\forall (i,j) \in D : t_i \neq t_j$. We define $\tilde{\delta}$ inductively as follows:

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

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

Note that, as well as a normal bottom-up tree automaton, we can run a TAC on a tree in (amortized) linear time, by first computing the minimal dag representation of the input tree; due to its minimality, the equality (or disequality) test of two subtrees can be carried out in constant time, by a single pointer comparison. Also note that total deterministic TACs are equally expressive as its nondeterministic version (as shown in Proposition 4.2 of [BT92] by a variant of usual powerset construction). Hence, we adopt total deterministic TACs as our look-ahead model for mtts, without sacrificing the expressiveness.

Definition 4.9. An *mtt with TAC look-ahead* is a tuple $M = (Q, q_0, \Sigma, \Delta, R, B)$ where $B = (P, \Sigma, \delta)$ is a total and deterministic TAC, and all other components are defined as for mtts, except that the form of rules are as follows:

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

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

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

$$[\![\langle q, \sigma(s_1,\ldots,s_k)\rangle(u_1,\ldots,u_m)]\!]^M_{\mu,\Gamma} = \bigcup_{r\in R'} \left\{ [\![r[x_1/s_1,\ldots,x_k/s_k]]\!]^M_{\mu,(y_1\mapsto t_1,\ldots,y_m\mapsto t_m)} \mid t_1 \in [\![u_1]\!]^M_{\mu,\Gamma},\ldots,t_m \in [\![u_m]\!]^M_{\mu,\Gamma} \right\}$$

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

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

Mtts with TAC look-ahead are strictly more expressive than normal mtts. For example, the translation $\{(\pi(s,s),e) \mid s \in T_\Sigma\}$ where π is a symbol of rank 2 and e is of rank 0, can be done by a transducer with TAC look-ahead. But no mtt-composition can realize this translation because the domain is not regular (by Corollary 5.6 of [EV85], the domain of any mtt must be a regular tree language). Nevertheless, the PTIME translation membership for $\mathrm{MT}_{\mathrm{IO}}$ can be extended to mtts with TAC look-ahead.

Theorem 4.10. *Let M be an mtt with TAC look-ahead. Translation membership for $\tau_{\mathrm{IO},M}$ can be determined in time $O(|s| \cdot |t|^{2m+2} \cdot |M|)$ where m is the maximum rank of M's states.*

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

Let s_{dag} be the minimal dag representation of s, which can be computed in $O(|s|)$ time. As explained before, the equality (or disequality) test of two subtrees of s_{dag} can be carried out in constant time. Let V_s be the set of nodes of s_{dag}. Let V_t be the set of nodes of t_{dag} and $V = V_t \cup \{\bot\}$. The functions $\mathit{label}(v)$, $\mathit{child}(v,i)$, and $\rho(t)$ are defined as in the proof of Theorem 4.6.

Let $A = 2^{\bigcup_i Q^{(i)} \times V^i \times V}$ and $\mathit{run} : T_\Sigma \to V_s \times P \times A$ (note the difference of the return

value of *run*, compared to that in Theorem 4.6) be the function defined as follows

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

where the function *tr* is:

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

The definition of $f_{\vec{v},\vec{a}}$ remains exactly the same as in Theorem 4.6.

The look-ahead state $\tilde{\delta}(s')$ can be computed from σ, $p_1, \ldots, p_k$, and $s_1, \ldots, s_k$ in constant time. By the same argument as the case of normal mtts, we obtain the $O(|s| \cdot |t|^{2m+2} \cdot |M|)$ time complexity. The correctness of the construction is proved also in the same way as for normal mtts. That is, we can prove the following claim by nested induction on the structure of s', and then on the structure of right-hand sides of the rules. The only difference from Theorem 4.6 is the side condition of the choice of right-hand sides r, which is defined to be coincide between the IO-semantics of mtts with TAC and the definition of *tr* function.

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

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

Another extension of mtts that admits a polynomial time translation membership is multi-return macro tree transducers.

Theorem 4.11. *Let M be an mr-mtt. Translation membership for $\tau_{\mathrm{IO},M}$ can be determined in time $O(|s| \cdot |t|^{2m+2d+2l} \cdot |M|)$ where m is the maximum rank of the states, d is the maximum dimension, and l is the maximum number of* let *-variables in each right-hand side.*

Proof. Let $Q^{(i,j)}$ is the set of states q of $rank(q) = i$ and $D(q) = j$. For mr-mtts, we take the set A of inverse-type automaton as $A = 2^{\bigcup_{i,j} Q^{(i,j)} \times V^i \times V^j}$. The intuition of the set of states A is similar to the case of normal mtts. That is, "$(q, \vec{v}, \vec{w}) \in run(s')$"

means that "if q is applied to the input subtree s' with output subtrees rooted at $\vec{v}$ as parameters, then it may return a tuple of output subtrees $\vec{w}$". The construction is quite similar to that of the proof for the case of normal IO-mtts. We define *run* and *tr* as follows:

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

$$tr(\sigma, \vec{a}) = \{(q, \vec{v}, \vec{w}) \in \bigcup_i Q^{(i,j)} \times V^i \times V^j \mid \exists r \in R_{q,\sigma} : f_{\vec{v},\vec{a}}(r, \vec{w})\}$$

where $f_{\vec{v},\vec{a}}(r, \vec{w})$ is defined for

$$r = \mathsf{let}\ (z_1, \ldots z_{e_1}) \leftarrow \langle q_{j_1}, x_{i_1}\rangle(u_1, \ldots, u_{m_1})\ \mathsf{in}\ \cdots$$
$$\mathsf{let}\ (z_{e_{n-1}+1}, \ldots z_{e_n}) \leftarrow \langle q_{j_n}, x_{i_n}\rangle(u_{m_{n-1}+1}, \ldots, u_{m_n})\ \mathsf{in}\ (u_{m_n+1}, \ldots, u_{m_{n+1}})$$

as

$$\begin{aligned}
&\exists v'_1, \ldots, v'_{e_n} \in V : \Big(\\
&\quad \exists \vec{v}^\dagger : \big((q_{j_1}, \vec{v}^\dagger, (v'_1, \ldots, v'_{e_1})) \in a_{i_1} \wedge \bigwedge_{1 \le i \le m_1} g_{\vec{v},\vec{v}',\vec{a}}(u_i, v_i^\dagger)\big) \wedge \cdots \wedge \\
&\quad \exists \vec{v}^\dagger : \big((q_{j_n}, \vec{v}^\dagger, (v'_{e_{n-1}+1}, \ldots, v'_{e_n})) \in a_{i_n} \wedge \bigwedge_{1 \le i \le m_n - m_{n-1}} g_{\vec{v},\vec{v}',\vec{a}}(u_{m_{n-1}+i}, v_i^\dagger)\big) \wedge \\
&\qquad \bigwedge_{1 \le i \le m_{n+1} - m_n} g_{\vec{v},\vec{v}',\vec{a}}(u_{m_n+i}, w_i) \\
&\Big)
\end{aligned}$$

with $g_{\vec{v},\vec{v}',\vec{a}}$ defined as same as the function $f_{\vec{v},\vec{a}}$ in the proof of Theorem 4.6, except that we add two rules of let-variables:

$$\begin{aligned}
g_{\vec{v},\vec{v}',\vec{a}}(z_i, w) &= \mathit{true} && \text{if } w = v'_i \\
g_{\vec{v},\vec{v}',\vec{a}}(y_i, w) &= \mathit{false} && \text{if } w \neq v'_i.
\end{aligned}$$

The translation membership then becomes equivalent to $(q_0, (), v_\epsilon) \in run(s)$ with v_ϵ the root node of t_{dag}. The correctness is proved again by nested-induction first on structure of s' and then on structure of r. The complexity $O(|s| \cdot |t|^{2m+2d+2l} \cdot |M|)$ is calculated in the same way as the previous theorems: $f_{\vec{v},\vec{a}}$ is called $|s| \cdot |t|^{m+d}$ times each $f_{\vec{v},\vec{a}}$ requires $|t|^{m+d+2l} \cdot |r|$ steps of computation. Multiplying them gives the desired complexity. □

As a final remark we would like to mention the complexity of translation membership for deterministic mtts; it can be determined in linear time. Since domains of compositions of mtts are regular, we can factor out the partiality and have the following decomposition: for $\mu \in \{\mathrm{IO}, \mathrm{OI}\}$, $\mathrm{DMT}^n_\mu \subseteq \mathrm{FTA}\,;\mathrm{D_tMT}^n$ where FTA is the class

of partial identities whose domain is regular (analogous to Theorem 6.18 of [EV85]). Therefore, to compute the translation membership for a composition of deterministic mtts, we first check in $O(|s|)$ time whether the given input s is contained in the domain of the translation, and then check the translation membership for composition of deterministic *and total* mtts. Here, by Theorem 15 of [Man02], for a translation $\tau \in \mathrm{D_tMT}^n$ we can compute the unique output tree $t' \in \tau(s)$ from the input s in time $O(|s| + |t'|)$, and during the computation, the size of every intermediate tree is less than or equal to $2^n \cdot |t'|$. Hence, for testing $(s,t) \in \tau$, we simply compute $\tau(s)$; if the size of any intermediate tree exceeds $2^n \cdot |t|$ then (s,t) cannot be an element of τ, and otherwise, we compare the computed tree $\tau(s)$ with t. The time complexity of the above procedure is $O(|s| + 2^n \cdot |t|)$.

Theorem 4.12. *Let* $\mu \in \{\mathrm{IO}, \mathrm{OI}\}$ *and* $n \geq 1$. *Translation membership for* DMT^n_μ *is in* $O(|s| + 2^n|t|)$.

Chapter 5

Complexities on Compositions of MTTs

Sequential composition of mtts gives rise to a powerful hierarchy (the "mtt-hierarchy") of tree translations which contains most known classes of tree translations. In this chapter we mainly study the data complexity of the membership test for the output languages of the mtt-hierarchy—they are proved to be NP-complete and in DSPACE(n). The key idea for obtaining the complexity is to transform the sequence of mtts into what we call the "garbage-free" form, meaning that each mtt does not delete much of its input, in the sense that every output tree t has a corresponding input tree of size only linearly larger than $|t|$.

5.1 Overview

As explained in the Introduction, the key idea for obtaining linear-size complexity for compositions of mtts is to bound the size of all intermediate input trees, and this is achieved by putting the mtts into "garbage-free" or "non-deleting" forms. In the same way as for total deterministic mtts [Man02], we classify the "deletion" in mtts into three categories – *erasing*, *input-deletion*, and *skipping* (a similar classification without erasing, which is a specific use of parameters, is also used in the case of nondeterministic tts [Bak78]). The resolution of each kind of deletion, however, requires several new techniques and considerations compared to previous work, due to the interaction of nondeterminism and parameters. In the rest of this paper, we first explain how we eliminate each kind of deletion, and then show the main results.

Note that, in this chapter, we basically deal only with OI-mtts and IO-mtts are dealt through a simulation by compositions of OI-mtts. This is because our results concerning garbage-free forms (e.g., Lemma 5.6) heavily rely on the right compositions of *linear nondeterministic* tts, which is known to require OI-nondeterminism.

5.2 Erasing

We first consider "erasing" rules – rules of the form $\langle q, \sigma(\cdots)\rangle(y_1, \ldots, y_m) \to y_i$. An application of such a rule consumes one input σ-node without producing any new output symbols; hence it is deleting a part of the input. Note that if the rank of σ is non-zero, then a rule as above is at the same time also input-deleting, which is handled in Section 5.3.

In the case of total deterministic mtts, "non-erasing" is a normal form, i.e., for every total deterministic mtt there is an equivalent one without erasing rules. Unfortunately, we could not find such a normal form for nondeterministic mtts with OI semantics. Note that for OI context-free tree grammars (essentially mtts without input: think of $\langle q, x_i\rangle$ as a nonterminal N_q, or equivalently, think of macro grammars [Fis68] or indexed grammars [Aho68], with trees instead of strings in right-hand sides), it has been shown [Leg81] that there is an erasing grammar that has *no* non-erasing normal form: erasing grammars are strictly more powerful than non-erasing ones. To see where the difficulty arises, let us consider the following example of a deterministic mtt and the input tree $\mathtt{a}(\mathtt{b},\mathtt{b})$:

$$\begin{aligned}\langle q_1, \mathtt{a}(x_1, x_2)\rangle &\to \langle q_2, x_1\rangle(\langle q_3, x_2\rangle(\mathtt{B},\mathtt{C}))\\ \langle q_2, \mathtt{b}\rangle(y_1) &\to \mathtt{A}(y_1, y_1)\\ \langle q_3, \mathtt{b}\rangle(y_1, y_2) &\to y_1.\end{aligned}$$

The $\langle q_3, \mathtt{b}\rangle$-rule is erasing. The basic idea of obtaining the non-erasing normal form for total deterministic mtts is to apply all erasing rules inline where they are called in a right-hand side. That is, we remove the erasing rule and modify the q_1 rule to $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\mathtt{B})$ (plus a look-ahead check "$x_2 = \mathtt{b}$"). This approach does not work properly under OI-nondeterminism. Let us suppose the case when the $\langle q_3, \mathtt{b}\rangle$-rules are nondeterministic as follows:

$$\langle q_3, \mathtt{b}\rangle(y_1, y_2) \to y_1 \qquad \langle q_3, \mathtt{b}\rangle(y_1, y_2) \to y_2 \qquad \langle q_3, \mathtt{b}\rangle(y_1, y_2) \to \mathtt{A}(y_1, y_2).$$

Note that the q_2 rule duplicates its argument $\langle q_3, x_2\rangle(\mathtt{B},\mathtt{C})$ *before* calling q_3, and evaluates the two copies independently. Thus, $\langle q_1, \mathtt{a}(\mathtt{b},\mathtt{b})\rangle{\downarrow}$ contains all the nine trees $\mathtt{A}(t_1, t_2)$ with $t_1, t_2 \in \{\mathtt{B}, \mathtt{C}, \mathtt{A}(\mathtt{B},\mathtt{C})\}$. However, the inline application of erasing rules now gives: $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\mathtt{B})$, $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\mathtt{C})$, and $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(\mathtt{A}(\mathtt{B},\mathtt{C}))$, which implies copying *after* evaluation of the q_3 call. So, in order to perform the expansion correctly, we need some way to preserve the nondeterministic choice after the expansion.

It is for this purpose that we move from normal mtts to *mtts with choice and failure*. The example above can be represented by an mttcf rule $\langle q_1, \mathtt{a}(x_1, x_2)\rangle \to \langle q_2, x_1\rangle(+(\mathtt{B}, +(\mathtt{C}, \mathtt{A}(\mathtt{B}, \mathtt{C}))))$, for instance. We will show that under OI-semantics, every mtt can be simulated by a canonical non-erasing mttcf. Recall that canonicity means there is no top-level $+$ or θ symbol in right-hand sides. Hence, by posing the canonicity, we can also avoid rules like $\langle q, x_1\rangle(y_1) \to +(y_1, \delta)$.

Lemma 5.1. *Let M be a mtt. There exists effectively a linear tt E and a canonical mttcf M' such that M' is non-erasing and $\tau_E ; \tau_{\mathrm{OI},M'} = \tau_{\mathrm{OI},M}$. Path-linearity is preserved from M to M'.*

Proof. The idea is, we first predict all erasing beforehand and annotate each input node by the information of erasing, by using a preprocessing linear tt. Then we replace all erasing state calls (e.g., $\langle q, x_1\rangle(u_1)$ with the rule $\langle q, \ldots\rangle(y_1) \to y_1$) in the right-hand sides of rules with the result of the erasing call (e.g., u_1). Note that we have to deal with nondeterminism. Suppose we have two rules $\langle q, \sigma\rangle(y_1, y_2) \to y_1$ and $\langle q, \sigma\rangle(y_1, y_2) \to y_2$ and a state call $\langle q, x_1\rangle(u_1, u_2)$ in a right-hand side. In order to preserve the nondeterminism, we replace the state call by $+(u_1, u_2)$.

Let $M = (Q, \Sigma, \Delta, q_0, R)$. We define E to be a nondeterministic linear tt with the set of states $P = [Q \to 2^{\{1,\ldots,n\}}] \cup \{p_0\}$ (functions from Q to $2^{\{1,\ldots,n\}}$ where n is the maximum rank of the states of Q, and one distinct state p_0, which is the initial state), the input alphabet Σ, the output alphabet $\Sigma_p = \{(\sigma, p_1, \ldots, p_k)^{(k)} \mid \sigma^{(k)} \in \Sigma, p_i \in P\}$, and the following rules for every $\sigma^{(k)} \in \Sigma$ and $p_1, \ldots, p_k \in [Q \to 2^{\{1,\ldots,n\}}]$: $\langle p, \sigma(x_1, \ldots, x_k)\rangle \to (\sigma, p_1, \ldots, p_k)(\langle p_1, x_1\rangle, \ldots, \langle p_k, x_k\rangle)$ where $p \in \{p_0, (q \mapsto \bigcup\{f(r) \mid \langle q, \ldots\rangle(\ldots) \to r \in R\})\}$ with f recursively defined as follows: $f(y_i) = \{i\}$, $f(\delta(\ldots)) = \emptyset$, and $f(\langle q', x_j\rangle(r_1, \ldots, r_m)) = \bigcup\{f(r_i) \mid i \in p_j(q')\}$. The transducer E modifies the label $\sigma^{(k)}$ of each input node into the form $(\sigma^{(k)}, p_1, \ldots, p_k)$. The annotated information p_i intuitively means "if a state q of M is applied to the i-th child of the node, it will erase and return directly the e-th parameter for $e \in p_i(q)$". If $p_i(q) = \emptyset$ then no erasing will happen. The rule of E is naturally understood if it is read from right to left, as a bottom-up translation. Formally speaking, the following claim holds. It is easily proved by induction on the structure of s.

Claim . (1) For each $s \in T_\Sigma$ and $q \in Q^{(m)}$, there is a unique $p \in P \setminus \{p_0\}$ such that $[\![\langle p, s\rangle]\!]^E \neq \emptyset$, and $e \in p(q)$ if and only if $\square_e \in [\![\langle q, s\rangle(\square_1, \ldots, \square_m)]\!]^M_{\mathrm{OI}}$. (2) Let us denote by $[s]$ such p determined by s. The output $s' \in \tau_E(s)$ is unique. For $b \in pos(s) = pos(s')$, $label(s', b) = (label(s, b), [s|_{b.1}], \ldots, [s|_{b.k}])$.

We next define a non-erasing mttcf, using the annotation added by E. Let $M' =$

$(Q, \Sigma_p, \Delta, q_0, R')$ with $R' = \{\langle q, (\sigma, p_1, \ldots, p_k)(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r' \mid r \in R_{q,\sigma}, r' \in ne(r), r' \notin Y\}$ where the set $ne(r)$ is defined inductively by

$$\begin{aligned} ne(y_j) &= \{y_j\} \\ ne(\delta(r_1, \ldots, r_l)) &= \{\delta(r'_1, \ldots, r'_l) \mid r'_i \in ne(r_i)\} \\ ne(\langle q', x_j\rangle(r_1, \ldots, r_l)) &= \bigcup\{ne(r_i) \mid i \in p_j(q')\} \cup \{\langle q', x_j\rangle(nep(r_1), \ldots, nep(r_l))\}, \end{aligned}$$

and nep defined as follows: $nep(y_j) = y_j$, $nep(\delta(r_1, \ldots, r_l)) = \delta(nep(r_1), \ldots, nep(r_l))$, and $nep(\langle q', x_j\rangle(r_1, \ldots, r_l)) = +(u_1, +(u_2, \ldots, +(u_z, \theta)\cdots))$ where $\{u_1, \ldots, u_z\} = ne(\langle q', x_j\rangle(r_1, \ldots, r_l))$. Intuitively, ne adds rules by replacing each top-level state calls with its argument if the state call is erasing according to the annotation p_j's. The other function nep does essentially the same thing for non top-level positions, but by replacing erasing state calls with $+$ choices instead of adding rules to preserve the OI-nondeterminism. It should be clear from the definition that M' is canonical and non-erasing. Since ne only returns a set of subtrees of a right-hand side, ne and nep never add any new nesting among state calls, and thus M' is path-linear if M is.

The correctness of this construction is proved by induction on the structure of the input tree s, by showing that if $[\![u_i]\!]^M_{\mathrm{OI},\Gamma} = [\![u'_i]\!]^{M'}_{\mathrm{OI},\Gamma}$ then $[\![\langle q, s\rangle(u_1, \ldots, u_m)]\!]^M_{\mathrm{OI},\Gamma} = [\![\langle q, \tau_E(s)\rangle(u'_1, \ldots, u'_m)]\!]^{M'}_{\mathrm{OI},\Gamma} \cup \bigcup\{[\![u_i]\!]^M_{\mathrm{OI},\Gamma} \mid \square_i \in [\![\langle q, s\rangle(\square_1, \ldots, \square_m)]\!]^M_{\mathrm{OI},\Gamma}\}$. Applying this to the initial state q_0 proves the equation $\tau_{\mathrm{OI},M} = \tau_E\,;\tau_{\mathrm{OI},M'}$.

Let $s = \sigma(s_1, \ldots, s_k)$ (the base case is the case $k = 0$) and $\tau_E(s) = (\sigma, p_1, \ldots, p_k)(s'_1, \ldots, s'_k)$. Then we have

$$[\![\langle q, s\rangle(u_1, \ldots, u_m)]\!]^M_{\mathrm{OI},\Gamma} = \bigcup_{r \in R_{q,\sigma}} \{[\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi} \mid \Xi(y_i) = [\![u_i]\!]^M_{\mathrm{OI},\Gamma} \text{ for all } i\}$$

and

$$\begin{aligned} &[\![\langle q, \tau_E(s)\rangle(u'_1, \ldots, u'_m)]\!]^{M'}_{\mathrm{OI},\Gamma} \cup \bigcup\{[\![u_i]\!]^M_{\mathrm{OI},\Gamma} \mid \square_i \in [\![\langle q, s\rangle(\square_1, \ldots, \square_m)]\!]^M_{\mathrm{OI},\Gamma}\} \\ &= \bigcup_{r \in R_{q,\sigma}} \{[\![r'[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi'} \mid \Xi'(y_i) = [\![u'_i]\!]^{M'}_{\mathrm{OI},\Gamma} \text{ for all } i, r' \in ne(r) \setminus Y\} \\ &\quad \cup \bigcup\{[\![u_i]\!]^M_{\mathrm{OI},\Gamma} \mid \square_i \in [\![\langle q, s\rangle(\square_1, \ldots, \square_m)]\!]^M_{\mathrm{OI},\Gamma}\} \\ &= \bigcup_{r \in R_{q,\sigma}} \{[\![r'[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi'} \mid \Xi'(y_i) = [\![u'_i]\!]^{M'}_{\mathrm{OI},\Gamma} \text{ for all } i, r' \in ne(r)\}. \end{aligned}$$

The last equality is derived by showing $\square_i \in [\![\langle q, s\rangle(\square_1, \ldots, \square_m)]\!]^M_{\mathrm{OI},\Gamma}$ if and only if $\exists r \in R_{q,\sigma} : y_i \in ne(r)$, because it implies the the equivalence between $\bigcup\{[\![u_i]\!]^M_{\mathrm{OI},\Gamma} \mid \square_i \in [\![\langle q, s\rangle(\square_1, \ldots, \square_m)]\!]^M_{\mathrm{OI},\Gamma}\}$ and $\bigcup_{r \in R_{q,\sigma}}\{[\![r'[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi'} \mid \Xi'(y_i) = [\![u'_i]\!]^{M'}_{\mathrm{OI},\Gamma} \text{ for all } i, r' \in ne(r) \cap Y\}$. Here, let us assume $[\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi} = \bigcup_{r' \in ne(r)} [\![r'[\vec{x}/\vec{s}']]\!]^M_{\mathrm{OI},\Xi}$ for all $r \in R_{q,\sigma}$.

Then we have the desired equivalence as follows:

$$
\begin{aligned}
&\Box_i \in [\![\langle q, s\rangle(\Box_1, \ldots, \Box_m)]\!]^M_{\mathrm{OI},\Gamma} \\
&\quad\Longleftrightarrow\ \exists r \in R_{q,\sigma} \colon \Box_i \in [\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},(y_1 \mapsto \{\Box_1\},\ldots,y_m \mapsto \{\Box_m\})} \\
&\quad\Longleftrightarrow\ \exists r \in R_{q,\sigma} \colon \exists r' \in ne(r) \colon \Box_i \in [\![r'[\vec{x}/\vec{s}']]\!]^{M'}_{\mathrm{OI},(y_1 \mapsto \{\Box_1\},\ldots,y_m \mapsto \{\Box_m\})} \text{ (assumption)} \\
&\quad\Longleftrightarrow\ \exists r \in R_{q,\sigma} \colon \exists r' \in ne(r) \colon r' = y_i \qquad \text{(since } M' \text{ is non-erasing)} \\
&\quad\Longleftrightarrow\ \exists r \in R_{q,\sigma} \colon y_i \in ne(r).
\end{aligned}
$$

Also note that $[\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi} = \bigcup_{r' \in ne(r)} [\![r'[\vec{x}/\vec{s}']]\!]^M_{\mathrm{OI},\Xi}$ is sufficient for showing the induction statement. Hence, all we have to prove is the equality $[\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi} = \bigcup_{r' \in ne(r)} [\![r'[\vec{x}/\vec{s}']]\!]^M_{\mathrm{OI},\Xi}$, which is shown by induction on the structure of r.

If $r = y_i$, obviously the both side become $\Xi(y)$ and therefore equal. If $r = \delta(r_1, \ldots, r_n)$, we have $[\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi} = \{\delta(t_1, \ldots, t_n) \mid t_i \in [\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi}\}$, and by inner induction hypothesis, this is equal to $\{\delta(t_1, \ldots, t_n) \mid t_i \in \bigcup_{r' \in ne(r)} [\![r'_i[\vec{x}/\vec{s}']]\!]^{M'}_{\mathrm{OI},\Xi}\}$, which is by definition of ne and OI-semantics, equal to $\bigcup_{r' \in ne(r)} [\![r'[\vec{x}/\vec{s}']]\!]^M_{\mathrm{OI},\Xi}$. Lastly we consider the case $r = \langle q', x_j\rangle(r_1, \ldots, r_n)$. By inner induction hypothesis, we have $[\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi} = \bigcup_{r'_i \in ne(r)} [\![r'_i[\vec{x}/\vec{s}']]\!]^M_{\mathrm{OI},\Xi}$, which is equal to $[\![nep(r_i)[\vec{x}/\vec{s}']]\!]^M_{\mathrm{OI},\Xi}$. Therefore, by the outer induction hypothesis, $[\![r[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Xi}$ is equal to $[\![\langle q', x_j\rangle)(nep(r'_1), \ldots, nep(r'_n))[\vec{x}/\vec{s}']]\!]^{M'}_{\mathrm{OI},\Xi} \cup \bigcup\{[\![r_i[\vec{x}/\vec{s}]]\!]^M_{\mathrm{OI},\Gamma} \mid \Box_i \in [\![\langle q, s\rangle(\Box_1, \ldots, \Box_m)]\!]^M_{\mathrm{OI},\Gamma}\}$. By the previous claim and the inner induction hypothesis, this is exactly the set $\bigcup_{r' \in ne(r)} [\![r'[\vec{x}/\vec{s}']]\!]^M_{\mathrm{OI},\Xi}$. □

5.3 Input-Deletion

The second kind of deletion we investigate is "input-deletion". For instance, if there is the rule $\langle q_0, \mathtt{a}(x_1, x_2)\rangle \to \mathtt{A}(\langle q_0, x_2\rangle)$ for the initial state q_0 and the input is of the form $\mathtt{a}(t_1, t_2)$, then the subtree t_1 is never used for the output calculation. Although total deterministic mtts can be made *nondeleting* (i.e., to always traverse all subtrees of *every* input tree) by preprocessing with a deleting linear tt [Man02], it becomes more difficult for nondeterministic mtts. This is because of the nondeterminism, which means that there can be more than one possible computation for a single input tree, and we cannot avoid the situation that one of the computations traverses all subtrees while others do not. Consider the input $\mathtt{a}(\mathtt{b}, \mathtt{c}(t_1, t_2))$ and the following set of rules:

$$
\begin{aligned}
\langle q_1, \mathtt{a}(x_1, x_2)\rangle &\to \langle q_2, x_1\rangle(\langle q_3, x_2\rangle) & \langle q_3, \mathtt{c}(x_1, x_2)\rangle &\to \langle q_4, x_1\rangle \\
\langle q_2, \mathtt{b}\rangle(y_1) &\to \mathtt{A}(y_1, y_1) & \langle q_3, \mathtt{c}(x_1, x_2)\rangle &\to \langle q_4, x_2\rangle
\end{aligned}
$$

Note that the state call for q_3 is duplicated by OI semantics; even though the mtt is linear. There are three possibilities with respect to input-deletion: either t_1 is deleted (the case all duplicated q_3 calls choose the second q_3-rule), t_2 is deleted (the case all choose the first rule), or no deletion occurs. We can still construct a linear tt that does preliminarily deletion, in such a way that it nondeterministically returns $\mathtt{a}(\mathtt{b}, \mathtt{c}_2(t_2))$, $\mathtt{a}(\mathtt{b}, \mathtt{c}_1(t_1))$, or $\mathtt{a}(\mathtt{b}, \mathtt{c}_{12}(t_1, t_2))$ (the subscript on $\mathtt{c}$ identifies the non-deleted children). We can also modify the mtt as follows

$$\langle q_3, \mathtt{c}_1(x_1)\rangle \to \langle q_4, x_1\rangle \qquad \langle q_3, \mathtt{c}_{12}(x_1, x_2)\rangle \to \langle q_4, x_1\rangle$$
$$\langle q_3, \mathtt{c}_2(x_1)\rangle \to \langle q_4, x_1\rangle \qquad \langle q_3, \mathtt{c}_{12}(x_1, x_2)\rangle \to \langle q_4, x_2\rangle$$

in which rules using the "deleted" input subtrees are removed. Then, for the former two "deleted" instances of the input trees, the mtt is successfully non-input-deleting. But sadly, this mtt still may delete for the last instance of the input tree, when all duplicated $\langle q_3, \mathtt{c}_{12}\rangle$ calls choose the same rule. The point is, under nondeterminism, we cannot argue the input-deleting property of each *transducer*. Rather, we can only argue whether each *computation* is input-deleting or not. This is a weaker version of the nondeletion condition used for total deterministic mtts, but it is sufficient for our purpose.

In order to speak more formally, here we define the notion of *computation tree* (following the method of [Bak78], but extending it to deal with accumulating parameters). For any finite set P, we define the ranked alphabet $\underline{P} = \{\underline{p}^{(1)} \mid p \in P\}$. Let $M = (Q, \Sigma, \Delta, q_0, R)$ be an mttcf and $s \in T_\Sigma$. The set $COMP(M, s)$ is the set of trees $comp\langle q_0, \underline{\epsilon}\rangle\downarrow \subseteq T_{\Delta \cup \underline{pos(s)}}$ called *computation trees* (or sometimes, simply *computations*). The derivation $comp\langle q_0, \underline{\epsilon}\rangle\downarrow$ is carried out under the following set of rewriting rules with outside-in derivation: $+(u_1, u_2) \to u_1$, $+(u_1, u_2) \to u_2$, and $comp\langle q, \underline{\nu}\rangle(\vec{y}) \to f_\nu(r)$ for $q \in Q$, $\nu \in pos(s)$, $r \in R_{q, label(s,p)}$ where f_ν is inductively defined as follows:

$$f_\nu(y_i) = y_i$$
$$f_\nu(\delta(r_1, \ldots, r_k)) = \underline{\nu}(\delta(f_\nu(r_1), \cdots, f_\nu(r_k))$$
$$f_\nu(\langle q', x_j\rangle(r_1, \ldots, r_k)) = comp\langle q', \underline{\nu.j}\rangle(f_\nu(r_1), \cdots, f_\nu(r_k))).$$

Intuitively, $COMP(M, s)$ is the set of trees $\langle q_0, s\rangle\downarrow$ where the parent of each Δ-node is a monadic node labeled by the position in the input tree s that generated the Δ-node. For example, the output tree $\underline{\epsilon}(\alpha(\underline{\epsilon.1}(\beta), \underline{\epsilon.2}(\gamma(\underline{\epsilon}(\delta)))))$ means that the α and δ nodes are generated at the root node of the input tree, and the β and γ nodes are

generated at the first and the second child of the root node, respectively. Let *delpos* be the translation that removes all $\underline{\nu} \in \underline{pos(s)}$ nodes. It is easily proved by induction on the number of derivation steps that $delpos(COMP(M,s)) = \langle q_0, s\rangle\downarrow_{\mathrm{OI},M}$, i.e., if we remove all $\underline{pos(s)}$ nodes from a computation tree, we obtain an output tree of the original mtt.

We say that a computation tree u is *non-input-deleting* if for every leaf position $\nu \in pos(s)$, there is at least one node in u labeled by $\underline{\nu}$. Note that the rewriting rules of *comp* corresponding to erasing rules do not generate any $\underline{pos(s)}$ node. Thus, non-input-deletion implies that not only some state is applied to every leaf, but also a *non-erasing* rule of some state must be applied.

Lemma 5.2. *Let M be a canonical non-erasing mttcf. There effectively exists a linear tt I and a canonical non-erasing mttcf M' such that $\tau_{\mathrm{OI},M} = \tau_I\,;\tau_{\mathrm{OI},M'}$, and for every input-output pair $(s,t) \in \tau_{\mathrm{OI},M}$, there exists a tree s' and a computation tree $u \in COMP(M',s')$ such that $(s,s') \in \tau_I$, $t = delpos(u)$, and u is non-input-deleting. Also, M' is path-linear if M is.*

Proof. Let $M = (Q,\Sigma,\Delta,q_0,R)$. We define I as $(\{d\},\Sigma,\Sigma',d,U)$ where $\Sigma' = \{(\sigma,i_1,\ldots,i_m)^{(k)} \mid \sigma^{(k)} \in \Sigma, 1 \le i_1 < \cdots < i_m \le k\}$ and

$$U = \{\langle d,\sigma(x_1,\ldots,x_k)\rangle \to (\sigma,i_1,\ldots,i_m)(\langle d,x_{i_1}\rangle,\ldots,\langle d,x_{i_m}\rangle) \mid (\sigma,i_1,\ldots,i_m) \in \Sigma'\}.$$

The transducer I reads the input tree and nondeterministically deletes subtrees while encoding the numbers of the undeleted subtrees in the current label. We define the mttcf M' as $(Q,\Sigma',\Delta,q_0,R')$ where

$$\begin{aligned} R' = \{&\langle q,(\sigma,i_1,\ldots,i_m)(x_1,\ldots,x_m)\rangle(\vec{y}) \to r' \\ &\mid r \in R_{q,\sigma} \text{ such that for all top-level calls } \langle q',x_p\rangle \text{ in } r, p \in \{i_1,\ldots,i_m\}, \\ &\quad \text{and } r' \text{ is obtained by replacing } \langle q',x_{i_j}\rangle \text{ in } r \text{ with } \langle q',x_j\rangle \\ &\quad \text{and } \langle q',x_p\rangle \text{ with } \theta \text{ for } p \notin \{i_1,\ldots,i_m\}\}. \end{aligned}$$

The transducer M' has basically the same rules as M, except that state calls on 'deleted' children are replaced by θ (or, if it is at the top-level then the rule is removed, to preserve canonicity). It should be easy to see that M' is canonical and non-erasing, and preserves the path-linearity of M.

The correctness of this construction is proved as follows. Note that there is the natural one-to-one correspondence between the set of $\langle q,\sigma\rangle$-rules of M and the set of $\langle q,(\sigma,i_1,\ldots,i_m)\rangle$-rules of M'. First, we have (1) $\tau_{\mathrm{OI},M'}(\tau_I(s)) \subseteq \tau_{\mathrm{OI},M}(s)$ because for

each derivation step in M' we can always apply the corresponding rewriting rule in M and obtain the corresponding (i.e., differs only at state calls on 'deleted' nodes that are replaced with θ) sentential form, which proves that we can obtain the same final output as $\tau_I ; \tau_{\mathrm{OI},M'}$ by $\tau_{\mathrm{OI},M}$. Next, we show that (2) for any $u \in COMP(M, s)$, there exists an equivalent non-input-deleting computation. Let s' be the minimal substructure of s that contains all nodes and their ancestors contained in u. Then $s' \in \tau_I(s)$ assuming that s' is appropriately relabeled from Σ to Σ' as the transducer I does, and we can obtain a computation $u' \in COMP(M', s')$ corresponding to u similarly as in (1). (Here, two thing assures that we can always choose the 'corresponding' rule. First, due to the OI-semantics, every rewriting is done at outermost positions and hence must be a part of the final output u. Second, non-erasure assures that if a state call $comp\langle q, \underline{p}\rangle$ is rewritten in the derivation of M, then a node labeled $\underline{p'}$ must be generated for some descendant p' of p, and thus the node corresponding to p' and its ancestor p are kept not removed in s'). Then, by the construction we have $delpos(u) = delpos(u')$, and since s' is the minimal substructure of s that contains all nodes occurred in u, all leaf nodes of s' occur in u', which means that u' is non-input-deleting. Note that (2) implies $\tau_{\mathrm{OI},M}(s) \subseteq \tau_{\mathrm{OI},M'}(\tau_I(s))$. Therefore, together with (1), we have $\tau_I ; \tau_{\mathrm{OI},M'} = \tau_{\mathrm{OI},M}$ as desired. □

5.4 Skipping

The third and last kind of deletion is "skipping". A computation tree u is *skipping* if there is a node $\nu \in pos(s)$ labeled by a rank-1 symbol such that no node in u is labeled $\underline{\nu}$. For a canonical, non-erasing, and path-linear mttcf, skipping is caused by either one of the following two forms of rules. One type is of the form $\langle q, \sigma(x_1)\rangle(y_1, \ldots, y_m) \to \langle q', x_1\rangle(u_1, \ldots, u_v)$ where $u_i \in T_{Y \cup C}$, and such rules are called *skipping*. The others are rules which are not skipping but are of the form $\langle q, \sigma(x_1)\rangle(y_1, \ldots, y_m) \to \langle q', x_1\rangle(u_1, \ldots, u_v)$ where $u_i \in T_{\Delta \cup Y \cup C}$, and such rules are called *quasi-skipping*. Note that, since the mttcf is path-linear, there are no nested state calls in right-hand sides of rules for input symbols of rank 1. Also note that if the root node of the right-hand side of a rule is not a state call, then it must be a Δ-node since the mttcf is canonical and non-erasing. So an application of such a rule generates a Δ-node and thus a $\underline{\nu} \in \underline{pos(s)}$ node for the current input node. Therefore, it is sufficient to consider only skipping and quasi-skipping rules.

Quasi-skipping rules may cause skipping computations due to parameter deletion: for example, consider the quasi-skipping rule $\langle q, \sigma(x_1)\rangle(y_1) \to \langle q', x_1\rangle(\delta(y_1))$; if there

is a q'-rule with a right-hand side not using y_1, then the σ-node may be skipped. For total deterministic mtts [Man02], there is a "parameter non-deleting" normal form, i.e., every total deterministic mtt is equivalent to one that uses all parameters in the right-hand sides of its rules, and thus only skipping rules (without choice nodes) were considered there. Unfortunately, as for non-erasure, we could not find such a normal form for nondeterministic mtts. Instead, we add some auxiliary skipping rules to mttcfs, so that we only need to consider skipping rules. Note that quasi-skipping rules cause skipping computations only when parameters are deleted. The idea is, if a parameter in some rule is never used for a computation, then replacing the parameter by a failure symbol θ does not change the translation, and moreover, such replacement changes a quasi-skipping rule into a skipping rule. Thus we may assume that all skipping computations are caused by skipping rules, and hence we can straightforwardly extend the proofs for total deterministic mtts [Man02] and nondeterministic tts [Bak78].

Lemma 5.3. *Let M be an canonical, non-erasing, and path-linear mttcf. There exists effectively a linear tt S and a canonical, non-erasing, and path-linear mttcf M' such that (1) $\tau_S ; \tau_{\mathrm{OI},M'} = \tau_{\mathrm{OI},M}$ and (2) for every input tree s and non-input-deleting computation tree $u \in COMP(M,s)$, there exists a tree s' and a computation tree u' such that $s' \in \tau_S(s)$, $u' \in COMP(M',s')$, $delpos(u') = delpos(u)$, and u' is both non-input-deleting and non-skipping.*

Proof. Let $M = (Q, \Sigma, \Delta, q_0, R)$. We define $N = (Q, \Sigma, \Delta, q_0, R \cup \overline{R})$ with:

$$\overline{R} = \{\langle q, \sigma(x_1)\rangle(\vec{y}) \to r' \mid q \in Q, \sigma \in \Sigma^{(1)}, r \in R_{q,\sigma}, r \text{ is quasi-skipping,}$$
$$\text{and } r' \text{ is obtained by replacing all subtrees of } r \text{ of the form } \delta(\cdots), \delta \in \Delta \text{ by } \theta\}.$$

Obviously $\tau_{\mathrm{OI},M} \subseteq \tau_{\mathrm{OI},N}$, and it should be also clear that $\tau_{\mathrm{OI},N} \subseteq \tau_{\mathrm{OI},M}$, because in each derivation of N, we can replace every application of $\overline{R}$ rules by the corresponding rules in R.

Furthermore, we can similarly prove that for any non-input-deleting computation u of $COMP(N,s)$ and U the set of rank-1 nodes of s that are skipped in u, there is a derivation that derives u and does not apply quasi-skipping rules to any $p \in U$. Suppose a quasi-skipping rule ρ is applied to a node $p \in U$ in a derivation of u. Then, since all Δ-nodes in ρ are skipped in u (that means, they never come to top-level position during the derivation), we can always replace the application with the corresponding skipping rule in $\overline{R}$ (recall that it is obtained by replacing all Δ-nodes

by θ) without changing the final output computation u. Thus, for N, without loss of generality we may assume that all skipping computation are caused by skipping rules.

We define S as $(H, \Sigma, \Sigma \times H, h_0, U)$ where

$$H = \bigcup_m [Q^{(m)} \to \bigcup_n \mathcal{P}(Q^{(n)} \times \mathcal{P}(\{1, \ldots, m\})^n)]$$

with $\mathcal{P}$ denoting power set, $h_0 = q \mapsto \{(q, \{1\}, \{2\}, \ldots, \{rank(q)\})\}$, and

$$\begin{aligned} U = \{&\langle h, \sigma(x_1, \ldots, x_k)\rangle \to (\sigma, h)(\langle h_0, x_1\rangle, \ldots, \langle h_0, x_k\rangle) \mid \sigma \in \Sigma, h \in H\} \\ \cup \{&\langle h, \sigma(x_1)\rangle \to \langle h'_\sigma, x_1\rangle \mid \sigma \in \Sigma^{(1)}, h \in H\}\} \end{aligned}$$

where h'_σ is the function

$$\begin{aligned} h'_\sigma = q \mapsto \{&(q'', f(r_1, \vec{\iota}), \ldots, f(r_l, \vec{\iota})\}) \mid \\ &(q', \iota_1, \ldots, \iota_n) \in h(q), \\ &\langle q', \sigma(x_1)\rangle(y_1, \ldots, y_n) \to \langle q'', x_1\rangle(r_1, \ldots, r_l) \in R \cup \overline{R}, r_i \in T_{Y \cup C}\} \end{aligned}$$

with f inductively defined as: $f(y_j, \iota_1, \ldots, \iota_m) = \iota_j$, $f(\theta, \iota_1, \ldots, \iota_m) = \emptyset$, and $f(+(r_1, r_2), \iota_1, \ldots, \iota_m) = f(r_1, \iota_1, \ldots, \iota_m) \cup f(r_2, \iota_1, \ldots, \iota_m)$.

The transducer S reads the input tree and nondeterministically deletes sequences of rank-1 nodes, while encoding the possible way to pass parameters by skipping rules. Such information on parameter passing is encoded as a function $h \in H$. For example, one possible output of S from the input $\mathtt{a}(\mathtt{b}(\mathtt{c}(\cdots))))$ is $(\mathtt{c}, h)(\cdots)$, in which $\mathtt{a}$ and $\mathtt{b}$ nodes are deleted and the information is encoded in h. Intuitive meaning of each h is, "if $(q', \iota_1, \ldots, \iota_n) \in h(q)$, then when a state q were applied to the root of the deleted sequence of rank-1 nodes with parameters $t_1, \ldots, t_m$, then N would have skipped the sequence, and reached a state q' with parameters $t_{i_1}, \ldots t_{i_n}, i_j \in \iota_j$ for $1 \leq j \leq n$". The initial state h_0 means that "no node was skipped so far".

We then define M' as $(Q, \Sigma \times H, \Delta, q_0, R')$ where

$$\begin{aligned} R' = \{&\langle q, (\sigma, h)(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to r[y_1/iset(\iota_1), \cdots, y_n/iset(\iota_n)] \\ &\mid (q', \iota_1, \ldots, \iota_n) \in h(q), r \in R_{q', \sigma}\} \end{aligned}$$

with $iset(\{i_1, \ldots, i_p\}) = +(y_{i_1}, +(y_{i_2}, \cdots, +(y_{i_p}, \theta)\cdots))$. Since M is canonical, nonerasing, and path-linear, clearly so is M'.

It should be easy for the reader to verify that the definition of M' and S follow the above intuition. The composition $\tau_S ; \tau_{\mathrm{OI}, M'} = \tau_{\mathrm{OI}, M}$ and the existence of a non-skipping computation can be proved similarly as for Lemma 5.2. The inclusion

$\tau_S ; \tau_{\mathrm{OI},M'} \supset \tau_{\mathrm{OI},M}$ is shown by taking the intermediate output from τ_S as the tree with same shape as the input s, i.e., the tree obtained by replacing the label σ of each node by (σ, h_0). Then by definition of R', for each rewrite step of M using a $\langle q, \sigma\rangle$-rule r, we can always choose the corresponding $\langle q, (\sigma, h_0)\rangle$-rule $r[y_1/iset(\iota_1), \ldots] \equiv r$. To show the inverse inclusion $\tau_S ; \tau_{\mathrm{OI},M'} \subset \tau_{\mathrm{OI},M}$, we construct a non-skipping computation u' from a computation $u \in \mathit{COMP}(M, s)$ by taking the intermediate tree s' as the tree obtained from s by deleting all rank-1 nodes not contained in u. Then for any sequence of skipping derivation $\mathit{comp}\langle q, \sigma_1(\cdots\sigma_n(\sigma(\vec{s}))\cdots)\rangle(\vec{v}) \Rightarrow^* \mathit{comp}\langle q', \sigma(\vec{s})\rangle(\vec{v}\,')$ in M where $\vec{v}'$ is some permutation of $\vec{v}$, we can always take in M' the corresponding sentential form $\mathit{comp}\langle q, (\sigma, h)(\vec{s}\,')\rangle(\vec{v})$. such that h contains the permutation that maps $\vec{v}$ to $\vec{v}\,'$. In this way, we can construct an equivalent non-skipping computation $u' \in \mathit{COMP}(M', s')$ from u. Hence, we have $\tau_S ; \tau_{\mathrm{OI},M'} \subset \tau_{\mathrm{OI},M}$. □

5.5 Counting the Number

The following lemma implies that after we have removed the three types of deletion, namely, erasure, input-deletion, and skipping, any translation become to have linearly bounded inputs.

Lemma 5.4. *Let $M = (Q, \Sigma, \Delta, q_0, R)$ be an mttcf, s an input tree, and u a non-input-deleting, non-skipping computation tree in $\mathit{COMP}(M, s)$ with $\mathit{delpos}(u) = t$. Then $|s| \leq 2|t|$.*

Proof. Since u is non-input-deleting and non-skipping, for all nodes $\nu \in \mathit{pos}(s)$ of rank zero or one, there exists a node labeled $\underline{\nu}$ in u, and by definition of computation trees, its child node is labeled by a symbol in Δ. Thus, $\mathit{leaves}(s) + \mathit{rank1nodes}(s) \leq |t|$ where $\mathit{leaves}(s)$ is the number of leaf nodes of s and $\mathit{rank1nodes}(s)$ is the number of nodes of s labeled by rank-1 symbols. Since $|s| \leq 2 \times \mathit{leaves}(s) + \mathit{rank1nodes}(s)$ (this holds for any tree s), we have $|s| \leq 2|t|$ as desired. □

5.6 Complexity of The Output Language

Lemma 5.5. *Let $\mathcal{K} \in \{\mathrm{DSPACE}(n), \mathrm{NP}\}$ and F a class of $\mathcal{K}$ languages effectively closed under* LT. *Then $\mathrm{LMT}_{\mathrm{OI}}(F)$ and $\mathrm{T}(F)$ are also in $\mathcal{K}$.*

Proof. Let M be a linear mtt or a tt. Note that in both cases, M is path-linear. First, we make it non-erasing; by Lemma 5.1, there exist a linear tt E and a canonical, non-erasing, and path-linear mttcf M_1 such that $\tau_E ; \tau_{\mathrm{OI},M_1} = \tau_{\mathrm{OI},M}$. Next, we make

each computation non-input-deleting; by Lemma 5.2, there exist a linear tt I and a canonical, non-erasing, and path-linear mttcf M_2 such that $\tau_I ; \tau_{\mathrm{OI},M_2} = \tau_{\mathrm{OI},M_1}$. For every $(s_1, t) \in \tau_{\mathrm{OI},M_1}$, there is an intermediate tree s_2 and a non-input-deleting computation $u \in COMP(M_2, s_2)$ such that $(s_1, s_2) \in \tau_I$ and $delpos(u) = t$. Then, we make each computation non-skipping; by Lemma 5.3, there exist a linear tt S and a canonical, non-erasing, and path-linear mttcf M_3 such that $\tau_S ; \tau_{\mathrm{OI},M_3} = \tau_{\mathrm{OI},M_2}$. For every non-input-deleting computation $u \in COMP(M_2, s_2)$, there is an intermediate tree s_3 and a non-input-deleting, non-skipping computation $u' \in COMP(M_3, s_3)$ such that $(s_2, s_3) \in \tau_S$ and $delpos(u') = delpos(u)$. Altogether, we have $\tau_E ; \tau_I ; \tau_S ; \tau_{\mathrm{OI},M_3} = \tau_{\mathrm{OI},M}$, and for every $(s, t) \in \tau_{\mathrm{OI},M}$ there exists a tree s_3 such that $(s, s_3) \in \tau_E ; \tau_I ; \tau_S$ and a non-input-deleting, non-skipping computation $u' \in COMP(M_3, s_3)$ such that $delpos(u') = t$. By Lemma 5.4, $|s_3| \leq 2|t|$.

Let L be a language in F. To check whether $t \in \tau_{\mathrm{OI},M}(L)$, we nondeterministically generate (for the case $\mathcal{K} = \mathrm{NP}$) or deterministically generate one-by-one (for the case $\mathcal{K} = \mathrm{DSPACE}(n)$) every tree s' of size $|s'| \leq 2|t|$ and for each of them, test whether $(s', t) \in \tau_{\mathrm{OI},M_3}$ and $s' \in (\tau_E ; \tau_I ; \tau_S)(L)$. By Theorem 4.5 (the NP and $\mathrm{DSPACE}(n)$ translation membership for a path-linear mtt), the former test can be done in complexity $\mathcal{K}$ with respect to $|t|$. By the assumption that F is closed under LT, the language $(\tau_E ; \tau_I ; \tau_S)(L)$ is also in $\mathcal{K}$. Thus the latter test is in complexity $\mathcal{K}$ with respect to $|s'| = O(|t|)$. □

Note that, for T, the same decomposition as shown in the first paragraph of the proof of Lemma 5.5 is already known (Lemma 1 of [Bak78]).

The proof of our next Lemma relies on several composition results developed in the history of transducer theory. Let $\mathrm{D_tQREL}$ be the class of *total deterministic bottom-up relabeling*s. A total deterministic relabeling is a tuple $B = (Q, \Sigma, \Delta, R)$ where Q, Σ, Δ is as for tts and R is a finite set of rules of the form: $\sigma(\langle q_1, x_1\rangle, \ldots, \langle q_k, x_k\rangle) \to \langle q, \delta(x_1, \ldots, x_k)\rangle$. For any $\sigma, q_1, \ldots, q_k$, there exists exactly one rule of the form (hence the name *total deterministic*). By regarding the rules as bottom-up rewrite rules, the quintuple B naturally represents a translation from $\mathrm{T}_\Sigma \to \mathrm{T}_\Delta$ that does not change the shape of the trees and just modifies the labels of each node. A top-down tree transducer preceded by a total deterministic bottom-up relabeling is called a top-down tree transducer *with regular look-ahead* and known to have good compositionality [Eng77]. Here we list the results that will be used later. Note that $\mathrm{D_tQREL}$ contains the

identity mapping. Therefore, $\mathrm{T} \subseteq \mathrm{D_t QREL}\,;\mathrm{T}$ and $\mathrm{LT} \subseteq \mathrm{D_t QREL}\,;\mathrm{LT}$.

$$(\mathrm{D_t QREL}\,;\mathrm{T})\,;(\mathrm{D_t QREL}\,;\mathrm{LT}) = (\mathrm{D_t QREL}\,;\mathrm{T}) \quad \text{Lemma 2.11 of [Eng77]} \quad \text{(C1)}$$

$$(\mathrm{D_t QREL}\,;\mathrm{LT})\,;(\mathrm{D_t QREL}\,;\mathrm{LT}) = (\mathrm{D_t QREL}\,;\mathrm{LT}) \quad \text{Lemma 2.11 of [Eng77]} \quad \text{(C2)}$$

$$\mathrm{D_t MT}\,;\mathrm{D_t QREL} = \mathrm{D_t MT} \quad \text{Lemma 11 of [EM02]} \quad \text{(C3)}$$

$$\mathrm{D_t QREL} \subseteq \mathrm{LT} \quad \text{Lemma 2.9 of [Eng75]} \quad \text{(C4)}$$

Another series of results required is about decompositions of OI-mtts. Two types of decomposition is known:

$$\mathrm{MT_{OI}} = \mathrm{D_t MT}\,;\mathrm{T} \quad \text{Corollary 6.12 of [EV85]} \quad \text{(C5)}$$

$$\mathrm{MT_{OI}} \subseteq \mathrm{D_t T}\,;\mathrm{LMT_{OI}} \quad \text{page 138 of [EV85]} \quad \text{(C6)}$$

Lemma 5.6. *Let* $\mathcal{K} \in \{\mathrm{DSPACE}(n), \mathrm{NP}\}$ *and* F *a class of* $\mathcal{K}$ *languages effectively closed under* LT. *Then* $\mathrm{MT_{OI}}(F)$ *is also in* $\mathcal{K}$ *and effectively closed under* LT.

Proof. The closure under LT immediately follows from C5, C1, C3, and again C5: $\mathrm{MT_{OI}}\,;\mathrm{LT} = \mathrm{D_t MT}\,;\mathrm{T}\,;\mathrm{LT} \subseteq \mathrm{D_t MT}\,;\mathrm{D_t QREL}\,;\mathrm{T} = \mathrm{D_t MT}\,;\mathrm{T} = \mathrm{MT_{OI}}$. We have $\mathrm{MT_{OI}}(F) \subseteq \mathrm{LMT_{OI}}(\mathrm{T}(F))$ by the decomposition C6, and by applying Lemma 5.5 twice, $\mathrm{LMT_{OI}}(\mathrm{T}(F))$ is in $\mathcal{K}$. □

Theorem 5.7. $\mathrm{MT^*_{OI}}(\mathrm{REGT}) \subseteq \mathrm{DSPACE}(n) \cap \mathrm{NP\text{-}complete}$.

Proof. The class REGT is closed under LT (Propositions 16.5 and 20.2 of [GS97]) and is in $\mathrm{DSPACE}(n) \cap \mathrm{NP}$ (see, e.g., [GS97]). By induction on $k \geq 1$ it follows from Lemma 5.6 that $\mathrm{MT^k_{OI}}(\mathrm{REGT})$ is in $\mathrm{DSPACE}(n)$ and NP. As noted in the Introduction, NP-hardness follows from [Rou73] and the fact that the indexed languages, which are equivalent to the yields of context-free-tree languages under OI-derivation, are in $\mathrm{MT^2_{OI}}(\mathrm{REGT})$. □

Although we only have considered outside-in evaluation order up to here, the previous result holds for compositions of mtts in *inside-out* evaluation order. This is because $\mathrm{MT^*_{IO}} = \mathrm{MT^*_{OI}}$ (more specifically, $\mathrm{MT_{OI}} \subseteq \mathrm{MT^2_{IO}}$ and $\mathrm{MT_{IO}} \subseteq \mathrm{MT^2_{OI}}$) by Theorem 7.3 of [EV85], where $\mathrm{MT_{IO}}$ denotes the class of translations realized by mtts in inside-out evaluation order.

Corollary 5.8. $\mathrm{MT^*_{IO}}(\mathrm{REGT}) \subseteq \mathrm{DSPACE}(n) \cap \mathrm{NP\text{-}complete}$.

The *yield* translation, which translates a tree into its string of leaf labels from left to right (seen as a monadic tree), is in $\mathrm{D_t MT}$. Therefore the output string languages

yield($\mathrm{MT}^*(\mathrm{REGT})$) of mtts are also in the same complexity class as Theorem 5.7. Especially, this class contains the IO- and OI- hierarchies [Dam82]. Note that the IO-hierarchy is in $\mathrm{D_tMT}^*(\mathrm{REGT})$ and hence in DSPACE(n) by Corollary 17 of [Man02]. Since the first level of the OI-hierarchy are the indexed languages [Fis68] which are NP-complete [Rou73], we obtain the following.

Corollary 5.9. *The OI-hierarchy is in* $\mathrm{DSPACE}(n) \cap \text{NP-complete}$.

5.7 Garbage-Free Form

The inductive proof of the output language complexity can be seen as iteratively making the last mtt in the composition sequence non-deleting and obtaining the non-deleting sequence of mtts eventually. By re-stating the proof strategy formally, we obtain the following main theorem of this chapter.

Theorem 5.10 (Garbage-Free Form)**.** *Let* $\tau \in \mathrm{MT}^k_{\mathrm{OI}}$ *(*$k \geq 1$*). There effectively exist translations* $\rho_1, \rho_2, \ldots, \rho_{2k}$ *realized by path-linear OI mttcfs and* $\rho_D \in \mathrm{D_tQREL\,;LT}$ *satisfying the following two conditions:*

1. $\tau = \rho_D\,;\rho_1\,;\cdots\,;\rho_{2k}$
2. *For any* $(s,t) \in \tau$ *there exists* $s_1, \ldots, s_{2k}$ *such that* $(s,s_1) \in \rho_D$, $(s_i, s_{i+1}) \in \rho_i$ *for* $1 \leq i \leq 2k-1$, $(s_{2k}, t) \in \rho_{2k}$, $|s_i| \leq 2|s_{i+1}|$ *for* $1 \leq i \leq 2k-1$, *and* $|s_{2k}| \leq 2|t|$.

Proof. The proof is by induction on k. For the case $k = 1$, by the decomposition C6, there exist translations $\tau_1 \in \mathrm{D_tT}$ and $\tau_2 \in \mathrm{LMT_{OI}}$ such that $\tau = \tau_1\,;\tau_2$. First we make the latter translation τ_2 non-deleting. By Lemmas 5.1, 5.2, 5.3, and 5.4, we can decompose τ_2 to $\tau_E\,;\tau_I\,;\tau_S\,;\rho_2$, where $\tau_E, \tau_I, \tau_S \in \mathrm{LT}$, ρ_2 a path-linear mttcf, and for every $(s,t) \in \tau_2$ there exists a tree s' such that $(s,s') \in \tau_E\,;\tau_I\,;\tau_S$ and $(s',t) \in \rho_2$ with $|s'| \leq 2|t|$. Then by C1, $\tau_1\,;\tau_E\,;\tau_I\,;\tau_S \in \mathrm{T\,;LT\,;LT\,;LT} \subseteq \mathrm{D_tQREL\,;T}$. Hence, we let $\tau_Q\,;\tau'_1 = \tau_1\,;\tau_E\,;\tau_I\,;\tau_S$ with $\tau_Q \subseteq \mathrm{D_tQREL}$ and $\tau'_1 \in \mathrm{T}$. Then again by using Lemmas 5.1, 5.2, 5.3, and 5.4, we decompose τ'_1 into $\tau'_E\,;\tau'_I\,;\tau'_S\,;\rho_1$ similarly. Here, let $\rho_D = \tau_Q\,;\tau'_E\,;\tau'_I\,;\tau'_S$. By C2, $\rho_D \in \mathrm{D_tQREL\,;LT\,;LT\,;LT} \subseteq \mathrm{D_tQREL\,;LT}$. Altogether,

the process of the transformation is summarized as follows

$$\begin{aligned}
\tau &= \tau_1 ; \tau_2 && (\mathrm{MT_{OI}} = \mathrm{D_t T} ; \mathrm{LMT_{OI}}) \\
&= \tau_1 ; (\tau_E ; \tau_I ; \tau_S ; \rho_2) && \text{(Eliminate Three Types of Deletion)} \\
&= (\tau_1 ; \tau_E ; \tau_I ; \tau_S) ; \rho_2 && \text{(Associativity)} \\
&= (\tau_Q ; \tau_1') ; \rho_2 && (\mathrm{T} ; \mathrm{LT}^* \subseteq \mathrm{D_t QREL} ; \mathrm{T}) \\
&= (\tau_Q ; (\tau_E' ; \tau_I' ; \tau_S' ; \rho_1)) ; \rho_2 && \text{(Eliminate Three Types of Deletion)} \\
&= (\tau_Q ; \tau_E' ; \tau_I' ; \tau_S') ; \rho_1 ; \rho_2 && \text{(Associativity)} \\
&= \rho_D ; \rho_1 ; \rho_2 && (\mathrm{D_t QREL} ; \mathrm{LT}^* \subseteq \mathrm{D_t QREL} ; \mathrm{LT})
\end{aligned}$$

with both the condition 1 and 2 satisfied.

Now we consider the case $k > 1$. Let $\tau = \tau_1 ; \cdots ; \tau_{k-1} ; \tau_k$ with $\tau_i \in \mathrm{MT_{OI}}$ for all i. Then by exactly the same transformation as the case $k = 1$, we can decompose τ_k to $\rho_D' ; \rho_{2k-1} ; \rho_{2k}$ such that $\rho_D' \in \mathrm{D_t QREL} ; \mathrm{LT}$, ρ_{2k-1}, ρ_{2k} are realized by path-linear mttcfs, and for any $(s', t) \in \tau_k$ there exist s_{2k-1} and s_{2k} with $(s', s_{2k-1}) \in \rho_D'$, $(s_{2k-1}, s_{2k}) \in \rho_{2k-1}$, $(s_{2k}, t) \in \rho_{2k}$, and $|s_{2k-1}| \leq 2|s_{2k}|$ and $|s_{2k}| \leq 2|t|$. Note that, by C1, C3, and C5, we have $\mathrm{MT_{OI}} ; \mathrm{D_t QREL} ; \mathrm{LT} \subseteq \mathrm{MT_{OI}}$. It implies that $(\tau_1 ; \cdots ; \tau_{k-1}) ; \rho_D' \in \mathrm{MT_{OI}^{k-1}}$. Therefore, by inductive hypothesis, $(\tau_1 ; \cdots ; \tau_{k-1}) ; \rho_D'$ is equal to the composition sequence $\rho_D ; \rho_1 ; \cdots ; \rho_{2k-2}$ satisfying the conditions 1 and 2. Hence, we have $\tau = \rho_D ; \rho_1 ; \cdots ; \rho_{2k-2} ; \rho_{2k-1} ; \rho_{2k}$ as desired. □

We show another applications of the garbage-free form, namely, the NP and the DSPACE(n) upperbound for OI-mtts, which have been shown only for linear mtts in the previous chapter. The result actually extends to finitely many compositions of mtts.

Theorem 5.11. *Translation membership of* $\mathrm{MT_{OI}^k}$ *for* $k \geq 1$ *and* $\mathrm{MT_{IO}^k}$ *for* $k \geq 2$ *is in* DSPACE(n) *and NP-complete.*

Proof. NP-hardness immediately follows from Theorem 4.1 and the fact that two compositions of IO-mtts can simulate one OI-mtts (Theorem 7.8 of [EV85]: $\mathrm{MT_{OI}} \subseteq \mathrm{MT_{IO}^2}$). Also, it is known that two compositions of OI-mtts can simulate IO-mtts: $\mathrm{MT_{IO}} \subseteq \mathrm{MT_{OI}^2}$ (Theorem 6.10 of [EV85]). Hence, it is sufficient to show the NP and the DSPACE(n) complexity only for the OI case.

Let $\tau \in \mathrm{MT_{OI}^k}$. Then we can take the garbage-free form $\rho_D ; \rho_1 ; \cdots ; \rho_{2k} = \tau$ so that it satisfy the condition of τ by Theorem 5.10. To determine for a given pair of trees (s, t) whether or not it is in τ, it is sufficient to test whether there exists

trees $s_1, \cdots, s_{2k}$ with $|s_1| \leq 2|s_2| \leq 4|s_3| \leq \cdots \leq 2^{2k}|s_{2k}| \leq 2^{2k+1}|t|$, $(s, s_1) \in \rho_D$, $(s_i, s_{i+1}) \in \rho_i$ for $1 \leq 2k-1$, and $(s_{2k}, t) \in \rho_{2k}$. For carrying out the check in NP time, we first nondeterministically generate all intermediate results s_1 to s_{2k} in linear time with respect to $|t|$, then run the NP translation membership test (Theorem 4.5) for each ρ_i. Note that the translation membership of $\rho_D \in \mathrm{D_t QREL}\,;\mathrm{LT}$ is also in NP (and in DSPACE(n) too), because we can check it by first running the $\mathrm{D_t QREL}$ part on s in deterministic linear space and time, and then check the translation membership of the LT part. For DSPACE(n) complexity, we generate one-by-one all trees of size $2^{2k-i+1}|t|$ as each s_i, and test the translation membership for each ρ_i and ρ_D. □

Chapter 6

Conclusion and Future Work

Although in general macro tree transducers seem to be a promising formal model for XML translations, they still have some shortcomings and many open problems. We have improved the issues from two directions: introduction of a mild extension of mtts that has better compositionality, and investigation on the complexity of several verification problems on the mtt-hierarchy. The contributions presented in the thesis are as follows.

- We have introduced an extension of mtts, namely, multi-return macro tree transducers. We have shown that multi-return macro tree transducers have better compositionality than normal mtts, and have also shown the strict increase in terms of expressiveness.
- We have shown the "garbage-free" form of compositions of mtts, which has allowed us to derive the complexity for membership of output languages and translation membership of the mtt-hierarchy.

6.1 Future Work

Open Classes of Translation Membership We wish to continue investigating the complexity of verification problems for tree translation models. From a theoretical point of view, there remain two variants of mtts whose complexities of translation membership are left open. One is macro forest transducers [PS04] and their generalization, the mtts with holes [MN08] in IO mode. Note that, similar to Theorem 4.6 of [MN08], hole-mtts in IO mode are equal to $\mathrm{MT}_{\mathrm{IO}}\,;\mathrm{YIELD}$, which is included in $\mathrm{MT}_{\mathrm{IO}}\,;\mathrm{LD}_{\mathrm{t}}\mathrm{MT}$. An algorithm based on inverse type inference does not work, because the parameter part of the states of the inverse-type automaton is a set of functions

$[V \to V]$, which is exponential in size with respect to the output tree $|t|$. On the other hand, it is not clear either whether it is NP-hard. Note that the class of translation is equal to the composition of $\mathrm{MT}_{\mathrm{IO}}$ with total deterministic transducers. Hence, the amount of nondeterminism is equal to IO-mtts and thus less than OI-mtts, which leaves some hope for PTIME translation membership. Another interesting class is that of *1-parameter* mtts in OI mode. Our encoding of 3-SAT used three parameters. In fact, the number of parameters can be reduced to two by embedding the encodings of boolean variables in the input tree s. But so far, we could not find any reduction from NP-hard problems to 1-parameter mtts. On the other hand, since a 1-parameter OI-mtt can generate $2^{2^{2^n}}$ output trees from a single input tree of size n in contrast to IO-mtts that have at most $O(2^{2^n})$ outputs, its amount of nondeterminism is similar to that of full OI-mtts. We expect that identifying the complexity of these border cases will shed some light on the relationship between the amount of nondeterminism and the computational hardness of the translation membership problem.

Applications of the Garbage-Free Form Another direction of further research is to seek more applications of the garbage-free form for showing the complexity of other problems concerning mtts. Note that, for the total deterministic version of the garbage-free form, several applications are known, such as a decision algorithm for the property called "linear-size increase" [Man03], or a decision algorithm for the finiteness of output languages. As an application of the nondeterministic version of garbage-free forms, we are particularly interested in the enumeration of the output set $\tau(s)$ from a single input tree, as well as the retrieval of arbitrary one tree from the set $\tau(s)$. Algorithms for solving these two problems have an application to, e.g., query optimization. Queries on XML documents written in regular expression patterns can be represented as a nondeterministic mtt translation that, given a database document, returns any one of the node satisfying the query condition. It is common to write a query in a compositional form. That is, programmers often construct a partial *view* of the original XML document, and run each query on the view document. Such a form of querying can be thought as a composition sequence of tree translations, and, our transformation to "garbage-free" form of mtts should imply an automatic elimination of unused part of view documents. We hope that the elimination can not only give us a theoretical complexity upperbound, but also bring a practical efficiency into the implementation of XML query engines.

Efficient Implementation of the Output Language Membership Our NP and DSPACE(n) algorithm for testing the membership of mtt output languages contained a part that "generates all intermediate trees (of size linearly bounded by $|t|$) one-by-one". Although it was sufficient for giving an upperbound complexity of the problem, naively implementing the strategy is far less efficient for use in the practice. We would like to give a more efficient method for truly 'guessing' and generating only the intermediate trees that indeed generates the given final output tree. Although the problem is NP-complete as we have shown in the thesis, there still remain some hope to (1) obtain a "practically efficient" algorithm, and (2) establish a subclass of translations that has tractable output language membership yet keeps subsuming many practical XML translations. The idea is that many parts of typical XML translations are *bidirectional*, i.e., most of the fragments of output XML documents are generated from a uniquely determined input fragment. Typically, it is not rare that a particular label (*tag* in XML) occurs in exactly one rule of a translation. In such a case, since we know that the rule must be used in the translation, we can uniquely identify some input fragment from the output.

References

[Aho68] Alfred V. Aho. Indexed grammars–an extension of context-free grammars. *Journal of the ACM*, 15:647–671, 1968.

[Asv81] Peter R.J. Asveld. Time and space complexity of inside-out macro languages. *International Journal of Computer Mathematics*, 10:3–14, 1981.

[Bak78] Brenda S. Baker. Generalized syntax directed translation, tree transducers, and linear space. *SIAM Journal on Computing*, 7:376–391, 1978.

[BLM08] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of XML document trees. *Information Systems*, 33:456–474, 2008.

[BPSMM00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible markup language (XML™), 2000. `http://www.w3.org/XML/`.

[BT92] Bruno Bogaert and Sophie Tison. Equality and disequality constraints on direct subterms in tree automata. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 161–171, 1992.

[CFZ82] Bruno Courcelle and Paul Franchi-Zannettacci. Attribute grammars and recursive program schemes I. *Theoretical Computer Science*, 17:163–191, 1982.

[Cou94] Bruno Courcelle. Monadic second-order definable graph transductions: A survey. *Theoretical Computer Science*, 126:53–75, 1994.

[Dam82] Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20:95–207, 1982.

[DE98] Frank Drewes and Joost Engelfriet. Decidability of the finiteness of ranges of tree transductions. *Information and Computation*, 145:1–50, 1998.

[EM99] Joost Engelfriet and Sebastian Maneth. Macro tree transducers, attribute grammars, and MSO definable tree translations. *Information and Computation*, 154:34–91, 1999.

[EM02] Joost Engelfriet and Sebastian Maneth. Output string languages of compositions of deterministic macro tree transducers. *Journal of Computer and System Sciences*, 64:350–395, 2002.

[EM03a] Joost Engelfriet and Sebastian Maneth. A comparison of pebble tree transducers with macro tree transducers. *Acta Informatica*, 39:613–698, 2003.

[EM03b] Joost Engelfriet and Sebastian Maneth. Macro tree translations of linear size increase are mso definable. *SIAM Journal on Computing*, 32:950–1006, 2003.

[Eng75] Joost Engelfriet. Bottom-up and top-down tree transformations – a comparison. *Mathematical Systems Theory*, 9:198–231, 1975.

[Eng77] Joost Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10:289–303, 1977.

[Eng80] Joost Engelfriet. Some open questions and recent results on tree transducers and tree languages. In *Formal Language Theorey; Perspectives and Open Problems*, pages 241–286. Academic Press, 1980.

[Eng86] Joost Engelfriet. The complexity of languages generated by attribute grammars. *SIAM Journal on Computing*, 15:70–86, 1986.

[EV85] Joost Engelfriet and Heiko Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31:71–146, 1985.

[EV88] Joost Engelfriet and Heiko Vogler. High level tree transducers and iterated pushdown tree transducers. *Acta Informatica*, 26:131–192, 1988.

[EV94] Joost Engelfriet and Heiko Vogler. The translation power of top-down tree-to-graph transducers. *Journal of Computer and System Sciences*, 49:258–305, 1994.

[FH07] Alain Frisch and Haruo Hosoya. Towards practical typechecking for macro tree transducers. In *Database Programming Languages (DBPL)*, pages 246–260, 2007.

[Fis68] Michael J. Fischer. *Grammars with Macro-Like Productions*. PhD thesis, Harvard University, Cambridge, 1968.

[Fül81] Zoltán Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[GS97] Ferenc Gécseg and Magnus Steinby. Tree languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol 3: Beyond Words*, pages 1–68. Springer-Verlag, 1997.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[IH08] Kazuhiro Inaba and Haruo Hosoya. Multi-return macro tree transducers. In *Programming Language Technologies for XML (PLAN-X)*, 2008.

[IHM08] Kazuhiro Inaba, Haruo Hosoya, and Sebastian Maneth. Multi-return macro tree transducers. In *Conference on Implementation and Application of Automata (CIAA)*, pages 102–111, 2008.

[IM08] Kazuhiro Inaba and Sebastian Maneth. The complexity of tree transducer output languages. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 244–255, 2008.

[IM09] Kazuhiro Inaba and Sebastian Maneth. The complexity of translation membership for macro tree transducers. In *Programming Language Technologies for XML (PLAN-X)*, 2009.

[Knu68] Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2:127–145, 1968.

[Leg81] Bernard Leguy. Grammars without erasing rules. the OI case. In *Trees in Algebra and Programming*, pages 268–279, 1981.

[Man02] Sebastian Maneth. The complexity of compositions of deterministic tree transducers. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 265–276, 2002.

[Man03] Sebastian Maneth. The macro tree transducer hierarchy collapses for functions of linear size increase. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 326–337, 2003.

[MB04] Sebastian Maneth and Giorgio Busatto. Tree transducers and tree compressions. In *Foundations of Software Science and Computation Structures (FoSSaCS)*, pages 363–377, 2004.

[MBPS05] Sebastian Maneth, Alexandru Berlea, Thomas Perst, and Helmut Seidl. XML type checking with macro tree transducers. In *Principles of Database Systems (PODS)*, pages 283–294, 2005.

[MN08] Sebastian Maneth and Keisuke Nakano. XML type checking for macro tree transducers with holes. In *Programming Language Technologies for XML (PLAN-X)*, 2008.

[MSV03] Tova Milo, Dan Suciu, and Victor Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66:66–97, 2003.

[PS04] Thomas Perst and Helmut Seidl. Macro forest transducers. *Information Processing Letters*, 89:141–149, 2004.

[Rou70] William C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4:257–287, 1970.

[Rou73] William C. Rounds. Complexity of recognition in intermediate-level languages. In *Foundations of Computer Science (FOCS)*, pages 145–158, 1973.

[ST80] Peter J. Downeyand Ravi Sethi and Robert Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27:758–771, 1980.

[Tha70] James W. Thatcher. Generalized^2 sequential machine maps. *Journal of Computer and System Sciences*, 4:339–367, 1970.

[Tha73] James W. Thatcher. Tree automata: an informal survey. In *Currents in the Theory of Computing*, pages 143–172. Prentice-Hall, 1973.

[Toz01] Akihiko Tozawa. Towards static type checking for XSLT. In *ACM Symposium on Document Engineering*, pages 18–27, 2001.

[Voi05] Janis Voigtländer. *Tree Transducer Composition as Program Transformation.* PhD thesis, Technische Universität Dresden, 2005.