Technische Universität Dresden
Fakultät Informatik

Diplomarbeit

Masters Thesis

von / by

Andreas Maletti

zum Thema / on the topic

"Direct construction and efficiency analysis for the accumulation technique for 2-modular tree transducers"

Verantwortlicher Hochschullehrer:	Prof. Dr.-Ing. habil. Heiko Vogler
Betreuer:	Dr.-Ing. Armin Kühnemann, Dipl. Inf. Janis Voigtländer
Institut:	Theoretische Informatik
Professur:	Grundlagen der Programmierung
Begonnen am:	23.05.2002
Eingereicht am:	21.11.2002

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbstständig und nur unter Zuhilfenahme der angegebenen Literatur verfasst habe.

Bannewitz, den 21. November 2002

....
Andreas Maletti

Contents

1 Introduction

A central topic in computer science is the development and implementation of algorithms in order to solve a given problem. The classical approach includes the creation of a procedure, which if executed step-wise on a designated architecture solves the problem. Imperative programming languages provide the basis for the notation of such procedures. However, this approach has several drawbacks, which eventually led to different programming paradigms such as declarative programming, where the programmer provides a description of the solution, and its instance functional programming, which will be studied in this thesis.

Functional programming languages describe the solution of a given problem by means of mathematical functions. It was shown that this approach often yields very elegant and; utmost important for the industry; short programs [1]. Additionally, since the notation is based on mathematics and thereby enforces rigor, provability of the correctness of programs is greatly enhanced. In spite of the fact that those programs are run on imperative architectures and; thus, tend to run slower than their equivalent imperative counterparts; programs generated by modern compilers for functional programming languages achieve promising execution speed results [2].

Nevertheless, elegant and modular programs tend to engender inefficiency in terms of execution speed, because modular solutions combine several solutions to subproblems. Consequently, intermediate results may occur, which might possibly be avoided by a less elegant program. Especially, if the intermediate results are (large) structured objects, i.e. elements of an algebraic data type, their unnecessary creation could amount to a large share of the total execution time. To counteract that the programmer is forced to fiddle with his specification, several optimization techniques, which translate given (elegant) functional programs into equivalent (possibly cryptic) ones; hopefully; with a positive impact on efficiency, are studied in the literature.

Well-known examples of techniques which eliminate intermediate results include the Fold-Unfold strategy [BD77], Deforestation [Wad90] and a strategy usually called Accumulation (survey in [Boi91]), which was adapted from imperative programming [Str71] to functional programming [Bir84]. In [Küh98] the use of tree transducers; language-theoretic devices; in the field of functional program optimization was exposed, and in [KGK01] an instance of the accumulation strategy is discussed in the setting of modular tree transducers. Particularly, this construction provides the basis of this thesis.

The accumulation strategy was applied successfully, i.e. with an efficiency gain, to many examples, of which the reverse function, which reverses the order of the elements of a list, is certainly a rather prominent one. A novice in functional programming might write the following (evident) specification for the reverse function in Haskell [Tho99].

```
naive_reverse :: [a] -> [a]
```

[1] "Description of the results of an experiment in which several conventional programming languages, together with the functional language Haskell, were used to prototype a Naval Surface Warfare Center requirement for Geometric Region Servers. The resulting programs and development metrics were reviewed by a committee chosen by the US Navy. The results indicate that the Haskell prototype took significantly less time to develop and was considerably more concise and easier to understand than the corresponding prototypes written in several different imperative languages, including Ada and C++." in *Haskell vs. Ada vs. C++ vs. Awk vs. ..., An Experiment in Software Prototyping Productivity* by Paul Hudak and Mark P. Jones

[2] at least when compared to modern object-oriented (imperative) programming languages

```
naive_reverse   []   = []
naive_reverse (x:xs) = naive_reverse xs ++ [x]
```

However evident this solution is, the naive reverse function constitutes an algorithm, which requires computation time quadratic in the length of the input list. Another script realizing the reverse function is the following:

```
reverse :: [a] -> [a]
reverse xs = r xs []
          where r :: [a] -> [a] -> [a]
                r   []   ys = ys
                r (x:xs) ys = r xs (x:ys)
```

In the literature this version is called efficient reverse, since its time complexity is only linear in the length of the input list. Further, the efficient version can be derived from the inefficient version by means of the accumulation strategy.

The construction of [KGK01] provides an implementation of the accumulation strategy in the setting of modular tree transducers [EV91], which exactly characterize the class of primitive recursive tree translations. Additionally, modular tree transducers naturally constitute functional programs, so the results gained for modular tree transducers are readily applicable to restricted functional programs as well.

As already shown in [KGK01], the indirect construction of [KGK01] is able to derive accumulating programs from particular non-accumulating functional programs, like inefficient reverse. Thereby, the script of inefficient reverse can automatically be translated into the script of efficient reverse (at least in principle, since polymorphism cannot be handled). This thesis ought to supply a direct construction for three apparent purposes.

- The direct construction is expected to be faster and to require less memory, since it avoids the creation of intermediate program scripts. Thus, it is better suited for integration into an optimizing compiler.

- In order to relate the input and output program with respect to some property, a direct construction might prove to be beneficial, because the relation may be established directly without relating the intermediate programs.

- As already noted in [Küh01], the construction can be extended by lifting some of its restrictions. There is a chance that an indirect construction enforces a certain property on the input, which is only required to construct some intermediate program. Thus, a direct construction might be able to avoid this restriction and, thereby, broaden the applicability of the construction.

Moreover, two essential properties are usually required for any construction to be considered for integration into an optimizing compiler. The first of which is the correctness, i.e. the construction shall preserve the semantics of the input program, and the second is the non-deterioration (or even gain) of efficiency. While the first property is usually supplied with the construction, the second one is usually discarded by formal language theorists. Nonetheless, its importance cannot be denied, since an optimizing compiler should not replace a program by one with degraded efficiency.

It is common practice to place restrictions on the input programs, such that an efficiency non-deterioration can formally be proven. Another technique is the adaptation of the construction in order to avoid the deteriorating effect. The latter approach is used in this thesis to gain non-deterioration with respect to efficiency, where the number of call-by-need derivation steps required to reach normal form is used as the measure of efficiency. This measure is realistic, since many modern functional programming languages employ call-by-need as derivation strategy.

Thus, roughly speaking the integration of the presented direct construction into an optimizing compiler is prepared by providing a formal efficiency analysis together with a sample implementation of the construction in Haskell. The implementation uses a special markup language, namely Haskell^{+} as input language, but the implementation should readily be adaptable to pure Haskell.

Naturally the question arises, whether the presented construction can be extended to less restricted settings. [Küh01] already pointed out that the construction can be extended to cope with a restricted form of context information usage. In this thesis the such extended construction is formally presented and investigated with respect to efficiency.

For example, the construction can afterwards be applied to the following example program, which sorts the colors in a list according to the stripes of the American flag with `Break` denoting the end of a pair of stripes.

```
data Stripe = Red | White | Break

coll :: [Stripe] -> [Stripe]
coll x = coll' x []
     where coll' :: [Stripe] -> [Stripe] -> [Stripe]
           coll'       []     ys = ys
           coll' (Break : xs) ys = ys ++ coll' xs []
           coll'   (Red : xs) ys = Red : coll' xs ys
           coll' (White : xs) ys = coll' xs (White : ys)
```

However, the gained output program is rather inefficient. A particular cause of inefficiency can be eliminated, but the general problem remains.

Finally, the thesis relates the gained constructions to similar implementations of the accumulation strategy. There the main emphasis is laid on the indirect construction of [KGK01] and the transformation presented in [Wad87].

This thesis is divided into 10 sections. The second section is subdivided into several subsections; each establishing the basic notions and notations of a particular field of study. Those notions and notations are used throughout the thesis without providing references to their definition. Section 3 introduces modular tree transducers as the setting of the thesis by first defining the syntax and afterwards assigning a formal semantics. Additionally, a particular derivation relation, namely the call-by-need derivation relation, is defined for modular tree transducers.

The fourth section contains the main construction of this thesis along with examples of its application. A subsection is devoted to proving the correctness of the given construction, independently of the indirect construction of [KGK01], which is the basis of our presented construction. The next section concerns the efficiency effects that appear when using the construction of the previous section. Therefore, a formal efficiency measure is established and the construction is refined to avoid a certain deteriorating effect. Afterwards correctness of the refined construction is proven as well as efficiency non-deterioration.

Section 6 broadens the applicability of the presented construction by allowing a restricted use of context information. Another direct construction is derived by extending the indirect approach of [KGK01] and composing the several steps to gain the direct construction. We use the correctness results of the several stages in the indirect construction to gain a correctness proof of the direct construction. In section 7 this extended construction is studied with respect to efficiency. For some; severely restricted; modular tree transducers we claim efficiency non-deterioration results.

The comparison of the presented constructions to the accumulation techniques found in the literature is the main topic of section 8. Particularly, we will consider the original indirect construction of [KGK01] and the accumulation technique of [Wad87]. The implementations of the constructions presented in this thesis are described and demonstrated in section 9 and finally section 10 contains acknowledgements and pointers to future work.

2 Preliminaries

2.1 Sets

Without delving deeply into set theory as introduced by CANTOR [3], we will establish some basic notions and notations. A *set* is a collection of distinguishable objects, which are called *elements* of that set. Whenever s is an element of a set S, we will write $s \in S$, otherwise $s \notin S$. Two sets S_1 and S_2 are equal, denoted $S_1 = S_2$, if they have exactly the same elements, hence sets are uniquely identified by enumerating their elements. In the following we will refer to the set of natural numbers $\mathbb{N}$ including 0 and to the set of positive integers $\mathbb{N}_+$.

Given that a set S consists only of the distinguishable elements $s_1, s_2, \ldots, s_n$ with $n \in \mathbb{N}$, we will write $S = \{s_1, s_2, \ldots, s_n\}$, call the set S *finite* and associate to S the natural number n called the *cardinality* of S, denoted by $\operatorname{card}(S) = n$. The set having no elements is denoted $\emptyset$, i.e. $\operatorname{card}(\emptyset) = 0$. We will also use the so-called *set building principle* due to FREGE [4], which uniquely defines a set S by restricting a set U to all elements u bearing a property $p(u)$, abbreviated $S = \{\, u \in U \mid p(u) \,\}$ (we will omit U, if and when it is understood from the context) [5].

S' is a *subset* of a set S, denoted $S' \subseteq S$, if and only if $s \in S'$ implies $s \in S$. If $S' \subseteq S$ and there exists an element $s \in S$ with $s \notin S'$, then S' is a *proper* subset of S, designated $S' \subset S$. Since we will often deal with special subsets of the natural numbers $\mathbb{N}$, we are going to introduce several shorthands, namely $[l, u] = \{\, n \in \mathbb{N} \mid l \leq n \leq u \,\}$ and $[u] = [1, u]$ for every $l, u \in \mathbb{N}$, thus $[0] = \emptyset$. The *power set* $\mathfrak{P}(S)$ of a given set S is the set of all subsets of S defined to be

$$\mathfrak{P}(S) = \{\, S' \mid S' \subseteq S \,\}.$$

For sets S_1 and S_2 *set union* ($\cup$) and *intersection* ($\cap$) shall be defined as usual. In addition we call two sets S_1 and S_2 *disjoint*, whenever $S_1 \cap S_2 = \emptyset$, and alongside we define the *set difference* to be $S_1 \setminus S_2 = \{\, e \in S_1 \mid e \notin S_2 \,\}$.

The *cartesian product* of two sets S_1 and S_2, designated $S_1 \times S_2$, is defined to be the set of all ordered pairs with the first component of S_1 and the second of S_2, i.e.

$$S_1 \times S_2 = \{\, (s_1, s_2) \mid s_1 \in S_1,\ s_2 \in S_2 \,\}.$$

We extend this notion to finitely many sets $S_1, S_2, \ldots, S_n$ with $n \in \mathbb{N}$ by stating

$$S_1 \times S_2 \times \cdots \times S_n = \{\, (s_1, s_2, \ldots, s_n) \mid s_i \in S_i \text{ for every } i \in [n] \,\}.$$

In case the sets $S_1 = S_2 = \cdots = S_n$ coincide, we will also use powers, i.e. S_1^n to denote the n-fold cartesian product of S_1, where we set $S_1^0 = \{()\}$.

Whenever we deal with "incredibly" [6] large collections of objects, we will rather call them *classes* to avoid contradictions, and in particular, we use classes of sets, also called

[3] Georg Cantor. Beiträge zur Begründung der transfiniten Mengelehre [Contributions to the founding of the theory of transfinite numbers]. Mathematische Annalen 46, 481-512, B. G. Teubner, Leipzig, 1895.

[4] Gottlob Frege. Grundgesetze der Arithmetik I - begriffsschriftlich abgeleitet [Basic Laws of Arithmetic I]. H. Pohle, Jena, 1893.

[5] We are well aware of the fact that unrestricted (e.g. $\{\, u \mid p(u) \,\}$) use of this facility may cause contradictions, e.g. RUSSELL's paradox, but we turn a blind eye to that to avoid cumbersome notions.

[6] Schoenfield: "sets are intended to be those safe, comfortable classes which are used by mathematicians in their daily life and work, whereas proper classes are thought of as monstrously large collections which, if permitted to be sets, would engender contradictions".

families. Nevertheless we use the established notation also for classes with one notable exception, namely that we will use parentheses for the building principle.

2.2 Relations and mappings

Every subset of $D \times C$ for sets D and C is called *relation* from D to C. If D and C coincide, then every subset of D^2 is also said to be a *relation* on D. A particular relation is the *identity relation* on a set D denoted id_D, which is defined by $id_D = \{ (d,d) \mid d \in D \}$. We will often use pointer-like symbols to denote relations on a set D and write them infix, i.e. $d_1 \to d_2$ instead of $(d_1, d_2) \in \to$. If a relation $\to$ on D admits no infinite chains $d_1 \to d_2 \to \dots$ with $d_i \in D$ for every $i \in \mathbb{N}$, then it is called *terminating*.

Whenever $(d, c_1), (d, c_2) \in R$ with R being a relation from D to C implies $c_1 = c_2$, we will call R a *partial mapping* from D to C. In the following let f be a partial mapping from D to C. The *domain* of f, denoted $\mathrm{dom}(f) \subseteq D$, is the set of all elements $d \in D$ such that there exists an element $c \in C$ with $(d, c) \in f$.

For partial mappings we use the established functional notation $y = f(x)$ to denote $(x, y) \in f$. Furthermore we define $f(X)$ for every subset $X \subseteq D$ to be $f(X) = \{ f(x) \mid x \in X \}$ and consequently the *range* of f is $f(D)$. In particular, if $D = \mathrm{dom}(f)$, then f is said to be a (total) *mapping*, designated $f : D \longrightarrow C$.

The mapping $f : D \longrightarrow C$ is called:

- *surjective*, *surjection* or *onto* C, if and only if the range of f is C,
- *injective* or an *injection*, if and only if $f(x) = z$ and $f(y) = z$ imply $x = y$, and
- *bijective* or a *bijection*, if and only if f is both, a surjection and an injection.

Inversion ($f^{-1}(y) = x$ if and only if $f(x) = y$) is well-defined, only if f is injective. Therefore, we define $f^{-1} : C \longrightarrow \mathfrak{P}(D)$ with $f^{-1}(c) = \{ d \in D \mid f(d) = c \}$. Note that if f is injective, then for every $c \in C : f^{-1}(c)$ is either empty or a singleton set (a set with exactly one element).

Using the definition of bijections, we can generalize the notion of cardinality as follows: The cardinality of two sets S_1 and S_2 is equal (alternatively S_1 and S_2 are called *equipotent*), if and only if there exists a bijection from S_1 to S_2. A set is *countably infinite*, if and only if it is equipotent to the natural numbers. In the following we will often identify equipotent sets.

For convenience we assume three arbitrary countably infinite, but pairwise disjoint sets of variables $X = \{ x_i \mid i \in \mathbb{N} \}$, $Y = \{ y_i \mid i \in \mathbb{N} \}$ and $Z = \{ z_i \mid i \in \mathbb{N}_+ \}$ and additionally the finite subsets $X_n = \{ x_i \mid i \in [n] \} \subset X$, $Y_n = \{ y_i \mid i \in [n] \} \subset Y$ and $Z_n = \{ z_i \mid i \in [n] \} \subset Z$ for every $n \in \mathbb{N}$.

2.3 Closure operators and closure systems

A *closure operator* cl over a set $\mathcal{U}$ is a mapping $cl : \mathfrak{P}(\mathcal{U}) \longrightarrow \mathfrak{P}(\mathcal{U})$ with the following properties:

- Extensionality, i.e. for every $U \subseteq \mathcal{U}$: $U \subseteq cl(U)$,
- Monotonicity, i.e. $V \subseteq U \subseteq \mathcal{U}$ implies $cl(V) \subseteq cl(U)$ and

- Idempotency, i.e. for every $U \subseteq \mathcal{U}$: $cl(U) = cl(cl(U))$.

Every set S of subsets of a given set $\mathcal{U}$ ($S \subseteq \mathfrak{P}(\mathcal{U})$) is called a *system* over $\mathcal{U}$. For a system S over $\mathcal{U}$ we define intersection and union as follows:

$$\begin{aligned} \bigcap S &= \{\, u \in \mathcal{U} \mid u \in U \text{ for every } U \in S \,\} \\ \bigcup S &= \{\, u \in \mathcal{U} \mid u \in U \text{ for some } U \in S \,\} \end{aligned}$$

Consequently a *closure system* $\mathcal{C}$ over $\mathcal{U}$ is a system $\mathcal{C}$ over the *universe* $\mathcal{U}$, where the intersection of each subsystem of $\mathcal{C}$ is also in $\mathcal{C}$. It is known, that a closure operator cl over $\mathcal{U}$ uniquely defines a closure system $\mathcal{C}$ over $\mathcal{U}$ (and vice versa) by stating $\mathcal{C} = \{\, U \subseteq \mathcal{U} \mid cl(U) = U \,\}$ [7]. Elements of a closure system $\mathcal{C}$ are called *closed sets*, thus for each set U we have $U \in \mathcal{C}$, if and only if $cl(U) = U$. Given a closure system $\mathcal{C}$ over a set $\mathcal{U}$, its corresponding closure operator cl over $\mathcal{U}$ and $C \in \mathcal{C}$, we call a subset $G \subseteq \mathcal{U}$ a *generator* for C, if and only if $cl(G) = C$. In such a case, C is said to be *generated* by G.

In order to define some common closures, we first have to introduce two properties of relations. A relation R on D is called *reflexive*, if and only if $id_D \subseteq R$, and R is *transitive*, if and only if $(d_1, d_2), (d_2, d_3) \in R$ implies $(d_1, d_3) \in R$. Furthermore, we define the composition of a relation R_1 from D to C with a relation R_2 from C to B as $R_1 \circ R_2 = \{\, (d, b) \in (D \times B) \mid (d, c) \in R_1,\ (c, b) \in R_2 \text{ for some } c \in C \,\}$. For every relation $\rightarrow$ on D we recursively define $\rightarrow^0 = id_D$ and $\rightarrow^{n+1} = \rightarrow^n \circ \rightarrow$ for every $n \in \mathbb{N}$. Let $\rightarrow$ be a relation on a set D, then the *reflexive and transitive closure* of $\rightarrow$, denoted $\rightarrow^*$, is defined $\rightarrow^* = \bigcup_{n \in \mathbb{N}} \rightarrow^n$, whereas the *transitive closure* of $\rightarrow$, denoted $\rightarrow^+$, is defined by $\rightarrow^+ = \bigcup_{n \in \mathbb{N}_+} \rightarrow^n$.

To illustrate closure operators, closure systems and the one-to-one correspondence between them, we study the reflexive closure in some detail. Obviously for every universe $\mathcal{U}^2$ the operator $\mathrm{ref} : \mathfrak{P}(\mathcal{U}^2) \longrightarrow \mathfrak{P}(\mathcal{U}^2)$ defined by $\mathrm{ref}(U) = U \cup id_{\mathcal{U}}$ is extensive, monotone and idempotent. Thus, ref is a closure operator over $\mathcal{U}^2$. The closure system corresponding to ref is defined by

$$\mathcal{C} = \{\, U \subseteq \mathcal{U}^2 \mid \mathrm{ref}(U) = U \,\} = \{\, U \subseteq \mathcal{U}^2 \mid id_{\mathcal{U}} \subseteq U \,\}.$$

Apparently the system $\mathcal{C}$ is a closure system, since the intersection of each subsystem is again an element of $\mathcal{C}$ [8]. Note $\mathcal{C}$ contains every reflexive relation on $\mathcal{U}$. Given this closure system, we derive the corresponding closure operator $\mathrm{cl}_{\mathcal{C}} : \mathfrak{P}(\mathcal{U}^2) \longrightarrow \mathfrak{P}(\mathcal{U}^2)$ defined for every $U \subseteq \mathcal{U}^2$ by

$$\mathrm{cl}_{\mathcal{C}}(U) = \bigcap \{\, C \in \mathcal{C} \mid U \subseteq C \,\} = id_{\mathcal{U}} \cup U = \mathrm{ref}(U).$$

Evidently the mapping $\mathrm{cl}_{\mathcal{C}}$, which maps U to the least closed set containing U, is extensive, monotone and idempotent, and thus a closure operator. Further, note that whenever $\mathcal{C}$ is the closure system corresponding to a closure operator cl, then cl is also the closure operator corresponding to $\mathcal{C}$.

Again let $\rightarrow$ be a relation on D, $\rightarrow$ is called *confluent*, if and only if for every $d, d_1, d_2 \in D$ with $d \rightarrow^* d_1$ and $d \rightarrow^* d_2$ there exists $d' \in D$ with $d_1 \rightarrow^* d'$ and $d_2 \rightarrow^* d'$. Terminating

[7] cf. Definition 3 of [Wec92]

[8] The intersection of each subsystem, also $\bigcap \emptyset = \mathcal{U}^2$, contains at least $id_{\mathcal{U}}$ and is thereby an element of $\mathcal{C}$.

and confluent relations are called *canonical* and admit a *normal form* $\mathrm{nf}_{\to}(d)$ for every element $d \in D$, i.e. there exists a unique element [9] $\mathrm{nf}_{\to}(d) \in D$ such that $d \to^{*} \mathrm{nf}_{\to}(d)$ and there is no $d' \in D$ with $\mathrm{nf}_{\to}(d) \to d'$ (if some $d'' \in D$ bears the latter property it is also said to be *irreducible* with respect to $\to$).

2.4 Alphabets, signatures and algebras

Each non-empty finite set Σ is also called *alphabet* and its elements are called *symbols*. A *ranked alphabet* or *signature* is a pair (Σ, ar) consisting of an alphabet Σ and a total mapping $\mathrm{ar} : \Sigma \longrightarrow \mathbb{N}$ associating to every symbol its finite *arity*. If $\mathrm{ar}(\sigma) = n$ with $\sigma \in \Sigma$ and $n \in \mathbb{N}$, then the symbol σ has *arity* n or short: σ is n-ary. We will often treat signatures like sets and annotate the parenthesized arity of each symbol as a superscript, i.e. instead of $(\{\,\sigma_1, \sigma_2, \ldots, \sigma_n\,\}, \mathrm{ar})$ with $\mathrm{ar}(\sigma_i) = a_i$ for every $i \in [n]$, where $a_1, a_2, \ldots, a_n \in \mathbb{N}$ with $n \in \mathbb{N}_+$, we will simply write $\{\,{\sigma_1}^{(a_1)}, {\sigma_2}^{(a_2)}, \ldots, {\sigma_n}^{(a_n)}\,\}$ and assume the ar-function as being implicitly given. Furthermore, we define the restriction of a signature (Σ, ar) to n-ary symbols as $\Sigma^{(n)} = \{\,\sigma \in \Sigma \mid \mathrm{ar}(\sigma) = n\,\}$ for every $n \in \mathbb{N}$.

To establish algebras, we define an *operation* on a set D to be a mapping from D^n to D, where $n \in \mathbb{N}$. In particular, if $n = 0$, we will call such operations *constants* and identify them with elements of D. Additionally, we will refer to *unary*, if $n = 1$, *binary*, if $n = 2$, and, generally speaking, to n-ary operations. An *algebra* is a pair (S, F), where S is an arbitrary set and F is a family of finitary operations on S. If the operations are understood from the context, we will simply use S instead of (S, F). Specifically we make use of *monoids*, which are algebras $(S, (\cdot, 1)$ with an *associative* binary operation $\cdot$ on S, i.e. for every $s_1, s_2, s_3 \in S$ the equality $(s_1 \cdot s_2) \cdot s_3 = s_1 \cdot (s_2 \cdot s_3)$ holds, and a constant $1 \in S$ acting as *unit element*, i.e. for every $s \in S$ the equation $1 \cdot s = s \cdot 1 = s$ holds [BS81, Wec92].

Given an alphabet Σ the *word monoid* $W(\Sigma)$ over Σ is the set of all n-tuples $\bigcup_{i \in \mathbb{N}} \Sigma^i$ with $n \in \mathbb{N}$ of elements of Σ together with the operation of *concatenation* $\circ$ usually written infix, i.e. $(u_1, \ldots, u_n) \circ (v_1, \ldots, v_m) = (u_1, \ldots, u_n, v_1, \ldots, v_m)$ for all words $(u_1, \ldots, u_n) \in W(\Sigma)$ and $(v_1, \ldots, v_m) \in W(\Sigma)$ with $m, n \in \mathbb{N}$. The elements of the word monoid are called *words* and the empty word $()$, which acts as unit element, is often denoted ε. In addition, we will drop the tuple notation, when considering words and simply write $u_1 \ldots u_n$ instead of $(u_1, \ldots, u_n)$ with $n \in \mathbb{N}$. Obviously, the *length* of a word w is $n \in \mathbb{N}$, denoted $|w| = n$, if and only if $w \in \Sigma^n$. A word $(u_1, \ldots, u_n) \in W(\Sigma)$ with $n \in \mathbb{N}$ is said to be a prefix of a word $(v_1, \ldots, v_m) \in W(\Sigma)$ with $m \in \mathbb{N}$, if and only if $n \leq m$ and $u_i = v_i$ for every $i \in [n]$.

To exactly characterize the arities of the operation symbols, we define Σ-*algebras* for a signature Σ to be a pair (J, Σ^J) where J is an arbitrary set called the *carrier* and $\Sigma^J = (\,\sigma^J \mid \sigma \in \Sigma\,)$ is a family of realizations $\sigma^J : J^n \longrightarrow J$ of operation symbols $\sigma \in \Sigma^{(n)}$ with $n \in \mathbb{N}$. We assume the standard proof principles, especially structural induction on Σ-algebras, where the abbreviation I.H. shall refer to the local induction hypothesis.

A mapping $\psi : S \longrightarrow S'$ from an Σ-algebra $\mathcal{S} = (S, (\,\sigma^S \mid \sigma \in \Sigma\,))$ to another Σ-algebra $\mathcal{S}' = (S', (\,\sigma^{S'} \mid \sigma \in \Sigma\,))$ is called *homomorphism*, if and only if it respects the operations, i.e. for every $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$ and $s_1, s_2, \ldots, s_n \in S$:

$$\psi(\sigma^S(s_1, s_2, \ldots, s_n)) = \sigma^{S'}(\psi(s_1), \psi(s_2), \ldots, \psi(s_n)).$$

[9] compare Theorem 2.1.9 in [BN98] and the remarks

A homomorphism $\varphi : \mathcal{S} \longrightarrow \mathcal{S}$ is also called *endomorphism*. Bijective homomorphisms are called *isomorphisms* and the corresponding algebras $\mathcal{S}$ and $\mathcal{S}'$ are called *isomorphic* to each other, designated $\mathcal{S} \cong \mathcal{S}'$. We will often identify isomorphic algebras. Additionally we define the notion of a *subalgebra* of an algebra (S, F), which is an algebra (S', F') with $S' \subseteq S$ and F' contains exactly the operations of F but restricted to S' [10].

The set $\mathcal{T}_\Sigma(V)$ of Σ-*terms over* V with Σ being a signature and V being a set of variables is the smallest set $\mathcal{T} \subseteq W(\Sigma \cup V \cup \{\, (\, ,\,)\, \})$ such that

- $V \subseteq \mathcal{T}$ and
- if $\sigma \in \Sigma^{(n)}$ with $n \in \mathbb{N}$ and $\xi_1, \xi_2, \ldots, \xi_n \in \mathcal{T}$, then also $(\sigma\, \xi_1\, \xi_2\, \ldots\, \xi_n) \in \mathcal{T}$.

We abbreviate the set of *ground* Σ-*terms* (or just Σ-terms) $\mathcal{T}_\Sigma(\emptyset)$ by $\mathcal{T}_\Sigma$. For the sake of simplicity we drop outermost parentheses and drop parentheses completely for nullary symbols, i.e. we will write c instead of (c), whenever $c \in \Sigma^{(0)}$.

Let Σ be a signature and, furthermore, let V be a set of variables. The Σ-*term algebra over* V or short just *term algebra* with carrier $J = \mathcal{T}_\Sigma(V)$ is $(J, (\, \sigma^J \mid \sigma \in \Sigma\,))$, where $\sigma^J(t_1, t_2, \ldots, t_n) = (\sigma\, t_1\, t_2\, \ldots\, t_n)$ for every $\sigma \in \Sigma^{(n)}$. Having introduced terms, we define the following mappings on terms. Note that we will delimit numbers in a sequence by dots, e.g. 1.2.4 to mean 124 as a word or $(1, 2, 4)$ in the original tuple notation. Let Σ be a signature and V a set of variables then for every $t \in \mathcal{T}_\Sigma(V)$ we define the following mappings.

$$\mathrm{occ} : \mathcal{T}_\Sigma(V) \longrightarrow \mathfrak{P}(W(\mathbb{N}_+)) \qquad \mathrm{label}_t : \mathrm{occ}(t) \longrightarrow \Sigma \cup V \qquad \mathrm{var} : \mathcal{T}_\Sigma(V) \longrightarrow \mathfrak{P}(V)$$

For every $v \in V$ let:

$$\mathrm{occ}(v) = \{\varepsilon\} \qquad \mathrm{label}_v(\varepsilon) = v \qquad \mathrm{var}(v) = \{v\}.$$

For every $t = (\sigma\, t_1\, t_2\, \ldots\, t_n)$ with $\sigma \in \Sigma^{(n)}$ with $n \in \mathbb{N}$ and $t_1, t_2, \ldots, t_n \in \mathcal{T}_\Sigma(V)$ let:

$$\begin{aligned}
\mathrm{occ}(t) &= \{\varepsilon\} \cup \{\, i.\pi \mid i \in [n],\ \pi \in \mathrm{occ}(t_i)\,\} \\
\mathrm{label}_t(\pi) &= \begin{cases} \mathrm{label}_{t_i}(\pi') & \text{, if } \pi = i.\pi' \text{ with } i \in [n],\ \pi' \in \mathrm{occ}(t_i) \\ \sigma & \text{, otherwise} \end{cases} \\
\mathrm{var}(t) &= \bigcup_{i=1}^{n} \mathrm{var}(t_i).
\end{aligned}$$

Additionally we define $t|_. : \mathrm{occ}(t) \longrightarrow \mathcal{T}_\Sigma(V)$ and $t[.]_. : \mathcal{T}_\Sigma(V) \times \mathrm{occ}(t) \longrightarrow \mathcal{T}_\Sigma(V)$ for every $t' \in \mathcal{T}_\Sigma V$ to be:

$$t|_\varepsilon = t \qquad t[t']_\varepsilon = t'$$

and for every $n \in \mathbb{N}$, $\sigma \in \Sigma^{(n)}$, $t_1, t_2, \ldots, t_n \in \mathcal{T}_\Sigma(V)$ and $\pi \in \mathrm{occ}(\sigma\, t_1\, t_2\, \ldots\, t_n)$ with $\pi = i.\pi'$ and $i \in [n]$, $\pi' \in \mathrm{occ}(t_i)$

$$(\sigma\, t_1\, t_2\, \ldots\, t_n)|_{i.\pi'} = t_i|_{\pi'} \qquad (\sigma\, t_1\, t_2\, \ldots\, t_n)[t']_{i.\pi'} = (\sigma\, t_1\, \ldots\, t_{i-1}\, t_i[t']_{\pi'}\, t_{i+1}\, \ldots\, t_n).$$

Finally, for every $a \in \Sigma \cup V$ we define $\#_a(t) = \mathrm{card}(\{\, p \in \mathrm{occ}(t) \mid \mathrm{label}_t(p) = a\,\})$, and we generalize the previous notation also to $\#_A(t) = \mathrm{card}(\{\, p \in \mathrm{occ}(t) \mid \mathrm{label}_t(p) \in A\,\})$ for each $A \subseteq \Sigma \cup V$.

[10] i.e. for every n-ary operation f of F, we compute the restriction as $f' = f \cap (S')^{n+1}$

A (first order term) *substitution* is a mapping $\theta : V \longrightarrow \mathcal{T}_\Sigma(V)$, where V is a set of variables and Σ is a signature. As a shorthand we denote a substitution θ by

$$\theta = \{v_1 \mapsto t_1,\, v_2 \mapsto t_2, \ldots, v_n \rightarrow t_n\},$$

if and only if $\theta(v_i) = t_i$ with $t_i \in \mathcal{T}_\Sigma(V)$ for every $i \in [n]$ with $n \in \mathbb{N}$ with $\{v_1, v_2, \ldots, v_n\} \subseteq V$ and for every $v \in V \setminus \{v_1, v_2, \ldots, v_n\} : \theta(v) = v$. Occasionally we need to refer to the set $\{v_1, v_2, \ldots, v_n\}$ according to the above specification as $\mathcal{V}_\theta$. Note that a substitution can uniquely be extended to an endomorphism over the term algebra induced by $\mathcal{T}_\Sigma(V)$, so we can apply substitutions to arbitrary terms of $\mathcal{T}_\Sigma(V)$, usually denoted by writing the substitution as a postfix operator.

2.5 Term graphs

Let Σ be a signature and V be a set of variables. A *non-rooted term graph* in $\mathcal{T}_\Sigma(V)$ [11] is a triple (N, E, l) with a finite node set N, a partial labelling mapping l from N to Σ and a partial mapping E from $N \times \mathbb{N}_+$ to N ordering the successor nodes of a node, where $E(n, i)$ is defined, if $l(n) \in \Sigma^{(k)}$ with $k \in \mathbb{N}_+$ and $i \in [k]$, and undefined otherwise. A *term graph* in $\mathcal{T}_\Sigma(V)$ is a quadruple (N, E, l, r), where (N, E, l) is a non-rooted term graph and $r \in N$ is the root node. All notions and notations for non-rooted term graphs extend naturally to term graphs.

A *path* $p \in W(\mathbb{N}_+)$ connects two nodes $n_1, n_2 \in N$ of a non-rooted term graph (N, E, l), if and only if either $p = \varepsilon$ and $n_1 = n_2$, or $p = i.p'$ with $i \in \mathbb{N}_+$, $p' \in W(\mathbb{N}_+)$ and $E(n_1, i) = n'$ as well as p' connects n' and n_2. We will denote that p connects n_1 and n_2 by $\overline{n_1 n_2}^p$. Furthermore, we demand that every non-rooted term graph is *acyclic*, thus for every $n \in N : \overline{nn}^p$ implies $p = \varepsilon$. To simplify the use of (non-rooted) term graphs, we refer to the individual components in the specification of a (non-rooted) term graph $G = (N, E, l, r)$ (non-rooted (N, E, l)) by N_G, E_G, l_G and r_G [12]. The class of all non-rooted term graphs in T is designated $\mathcal{TG}_{\mathrm{nr}}(T)$ and the class of term graphs in T is denoted $\mathcal{TG}(T)$.

Let $G \in \mathcal{TG}(\mathcal{T}_\Sigma(V))$. G is said to be *connected*, if and only if for every node $n \in N_G$ there exists $p \in W(\mathbb{N}_+)$ such that $\overline{r_G n}^p$ [13]. This property allows us to conveniently omit the root node (in graphical representations) from the specification of a term graph [14]. The *set of occurrences* $\mathrm{occ}(G)$ is defined as $\mathrm{occ}(G) = \{\, p \in W(\mathbb{N}_+) \mid \overline{r_G n}^p \text{ for some } n \in N_G \,\}$. Note that $p_1, p_2 \in \mathrm{occ}(G)$ with $p_1 \neq p_2$ might actually refer to the same node.

Given a non-rooted term graph $G \in \mathcal{TG}_{\mathrm{nr}}(\mathcal{T}_\Sigma(V))$, a *term subgraph* of G is a term graph $G' = (N', E', l', r')$, which fulfills the connectedness property, i.e. $N' = \{\, n \in N_G \mid \overline{r' n}^p \text{ for some } p \in W(\mathbb{N}_+) \,\}$, as well as E' and l' being restrictions of E_G and l_G, respectively. The term subgraph of G with root node n is denoted $G|_n$ and, alongside, if G is a term graph, then we also use $G|_p$ for every $p \in \mathrm{occ}(G)$ to denote $(N_G, E_G, l_G)|_n$ with $\overline{r_G n}^p$ [15].

[11] The term universe $\mathcal{T}_\Sigma(V)$ is casually omitted.
[12] E.g. $l_{G'}$ refers to the partial labelling mapping of a term graph G'.
[13] Obviously, we do not define this property for non-rooted term graphs.
[14] The root can be identified, because there is only one $r_G \in N_G$ with $r_G \notin E_G(N_G \times \mathbb{N}_+)$.
[15] This overloading of $G|_\cdot$ should not engender confusion, since it will always be clear the argument is a node or a path.

Let $G \in \mathcal{TG}(\mathcal{T}_\Sigma(V))$, we define the *redirection* $G[\cdot \rightsquigarrow \cdot] : N_G^2 \longrightarrow \mathcal{TG}(\mathcal{T}_\Sigma(V))$, such that for every node $n_1, n_2 \in N_G : G[n_1 \rightsquigarrow n_2] = (N_G, E', l_G, r')$ [16], where

$$r' = \begin{cases} n_2 & \text{, if } r_G = n_1 \\ r_G & \text{, otherwise} \end{cases} \qquad \text{and} \qquad E'(n,i) = \begin{cases} n_2 & \text{, if } (n,i) \in E_G^{-1}(n_1) \\ E_G(n,i) & \text{, otherwise} \end{cases}$$

Finally for every $a \in \Sigma \cup V$ and $A \subseteq \Sigma \cup V$, $\#_a(G) = \mathrm{card}(\{\, p \in \mathrm{occ}(G) \mid \overline{rn}^p, l(n) = a \,\})$ and $\#_A(G) = \mathrm{card}(\{\, p \in \mathrm{occ}(G) \mid \overline{rn}^p, l(n) \in A \,\})$ are defined as they are for terms.

The term which corresponds to a given non-rooted term graph $G \in \mathcal{TG}_{\mathrm{nr}}(\mathcal{T}_\Sigma(V))$ below $n \in N_G$ is $\mathrm{term}_G(n)$, where $\mathrm{term}_G : N_G \longrightarrow \mathcal{T}_\Sigma(N_G)$ [17] is defined recursively by

$$\mathrm{term}_G(n) = \begin{cases} (l_G(n)\ \mathrm{term}_G(E_G(n,1))\ \ldots\ \mathrm{term}_G(E_G(n,k))) & \text{, if } l_G(n) \in \Sigma^{(k)}, k \in \mathbb{N} \\ n & \text{, otherwise} \end{cases}.$$

For term graphs G we abbreviate $\mathrm{term}_G(r_G)$ by $\mathrm{term}(G)$. Obviously, for a given term $t \in \mathcal{T}_\Sigma(V)$ there may exist different term graphs G, such that $\mathrm{term}(G) = t$. We highlight two representations, namely the *tree representation* of t, where for every $n \in N_G$, $p_1, p_2 \in W(\mathbb{N}_+)$ with $\overline{r_G n}^{p_1}$ and $\overline{r_G n}^{p_2}$ implies $p_1 = p_2$, and the *variable-shared* term graph, denoted G_t, where the previous condition holds only for every $n \in N_G$ with $l_G(n)$ defined.

If for two term graphs G_1 and G_2: $\mathrm{term}(G_1) = \mathrm{term}(G_2)$, then G_1 and G_2 are *bisimular*. Since there is a one-to-one correspondence between terms and tree representations, we also call G_1 and $\mathrm{term}(G_1)$ bisimular.

In graphical representations we will adopt the following conventions:

1. Nodes are depicted by displaying their label, if we display all ingoing as well as outgoing edges of that node. Alternatively, we use a small circle to depict a node and annotate the node and occasionally its label in parentheses next to the circle.

2. Edges are directed from top to bottom unless otherwise specified by arrows.

3. Successors of a node are ordered left to right with ascending order unless numbers at the edges suggest otherwise.

Given two term graphs $G, G' \in \mathcal{TG}(\mathcal{T}_\Sigma(V))$ a *term graph homomorphism* from G to G' is a mapping $\psi : N_G \longrightarrow N_{G'}$ with

- for every $n \in N_G$: if $l_G(n)$ is defined, then $l_{G'}(\psi(n)) = l_G(n)$,

- for every $n \in N_G$ and $i \in \mathbb{N}_+$: if $E_G(n,i)$ is defined, then $E_{G'}(\psi(n), i) = \psi(E_G(n,i))$ and

- $r_{G'} = \psi(r_G)$.

---

[16] Usually we enforce the connectedness property after executing the redirection. Therefore, we remove every node, for which there is no path from the root to this node. This process is called *garbage collection* and abbreviated G.C. Two term graphs G_1 and G_2 are called equivalent under garbage collection, denoted $G_1 \overset{\text{G.C.}}{=} G_2$, if and only if $G_1|_{r_{G_1}} = G_2|_{r_{G_2}}$.

[17] To get a term of $\mathcal{T}_\Sigma(V)$, we require another mapping $\varphi : N_G \longrightarrow V$, which translates nodes to variables. If in the second case n is replaced by $\varphi(n)$, then $\mathrm{term}_{G,\varphi} : N_G \longrightarrow \mathcal{T}_\Sigma(V)$. However, this mapping is required only for non-variable shared term graphs.

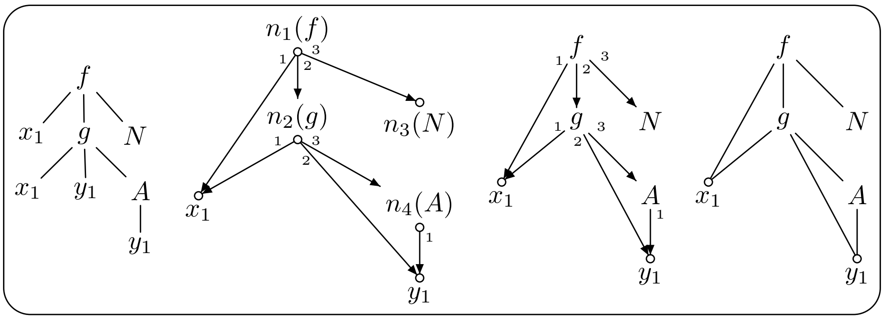

Figure 1: Tree representation and variable-shared term graphs with various detail levels for $(f\ x_1\ (g\ x_1\ y_1\ (A\ y_1))\ N) \in \mathcal{T}_{\{f^{(3)},g^{(3)},A^{(1)},N^{(0)}\}}(\{x_1, y_1\})$

Thus, a term graph homomorphism preserves the root, labels, successors and their order. Particularly, the term graph homomorphism ψ is *non-collapsing*, if for every $n_1, n_2 \in N_G$ with $l_G(n_1)$ and $l_G(n_2)$ defined: if $n_1 \neq n_2$, then $\psi(n_1) \neq \psi(n_2)$.

For two non-rooted term graphs $G_1 = (N_1, E_1, l_1)$ and $G_2 = (N_2, E_2, l_2)$ with $E_1(n, i) = E_2(n, i)$ and $l_1(n) = l_2(n)$ for every $n \in N_1 \cap N_2$ and $i \in \mathbb{N}_+$, we define the *non-rooted term graph union* $G_1 \cup G_2 = (N_1 \cup N_2, E_1 \cup E_2, l_1 \cup l_2)$.

2.6 Orderings

A relation $\preceq$ on D is *antisymmetric*, if and only if $d_1 \preceq d_2 \preceq d_1$ implies $d_1 = d_2$ for every $d_1, d_2 \in D$. A pair $(S, \preceq)$ with S being an arbitrary non-empty set, also called *carrier*, and $\preceq \subseteq S^2$ being a reflexive, antisymmetric and transitive relation on S is called a *partially ordered set* and $\preceq$ is said to be a *partial order*. If the partial order is understood from the context, we will identify the partially ordered set with the carrier set S. Furthermore, in case all elements $s_1, s_2 \in S$ are *comparable*, i.e. either $s_1 \preceq s_2$ or $s_2 \preceq s_1$, then $(S, \preceq)$ is a *totally ordered set* (also called *chain*). *Incomparable* elements $s_1, s_2 \in S$ are denoted $s_1 \| s_2$ and the *strict ordering* corresponding to $\preceq$ is $\prec = \preceq \setminus id_S$.

An element $m \in S$ of a partially ordered set $(S, \preceq)$ is defined to be *minimal*, if and only if $s \in S$ with $s \preceq m$ implies $s = m$, and, supplementary, we define the least element to be m, if and only if $m \preceq s$ for every $s \in S$. Maximal elements and the greatest element shall be defined dually. Further, we note that on totally ordered sets the notions of minimal and least elements obviously coincide.

Assuming the well-known total order $\leq$ on the natural numbers, we define $\min N$ for every finite set $N \subset \mathbb{N}$ to be the *minimal* element of N with respect to $\leq$, i.e. for every $n \in N$, $\min N \leq n$ with $\min N \in N$. The maximum $\max N$ shall be defined dually. Given partially ordered sets $(S_1, \preceq_1)$ and $(S_2, \preceq_2)$, we construct the following partial order called *lexicographic ordering* on the cartesian product of S_1 and S_2, i.e. $(S_1 \times S_2, \preceq)$ with $(s_1, s_2) \preceq (s_1', s_2')$, if and only if $s_1 \prec_1 s_1'$ or both $s_1 = s_1'$ and $s_2 \preceq_2 s_2'$. This principle extends naturally to cartesian products of finitely many sets. Further, we also extend the lexicographic ordering to words of $W(S)$ for every partial order $(S, \preceq)$ by requiring that $w \preceq w'$ with $w, w' \in W(S)$, if and only if either $w = \varepsilon$, or there exist $s, s' \in S$, $\bar{w}, \bar{w}' \in W(S)$ such that $w = s \circ \bar{w}$, $w' = s' \circ \bar{w}'$ and either $s \prec s'$ or $s = s'$ and $\bar{w} \preceq \bar{w}'$. We note that the construction of the lexicographic ordering preserves totality, i.e. if the partially ordered sets used as input are totally ordered sets, then the lexicographic ordering is also totally ordered.

2.7 Bibliographic remarks

Most of the notions and notations of this section are well accepted and usually mathematical folklore. They can be found similarly in many graduate textbooks of mathematics. The notations concerning universal algebra were mostly taken out of [Wec92, BS81]. The subsection on relations introduces notations of [Ave95, BN98, Bün98, Jou93], while the subsection on term graphs was greatly inspired by notions and notations of [Blo01, Cla96, BKdV+02]. Many abbreviations are adapted out of [Küh97]. An axiomatic approach to set theory can be found in [Pot90].

3 Modular tree transducers

Aiming towards the optimization of functional programs, we first need to establish a formal computational model to capture the computation process. Most commonly term rewriting systems [Ave95, BN98, Bün98], especially the lambda calculus [Bar84] as the most prominent instance, were proposed as formal models in the literature. However, in this thesis we will consider restricted term rewrite systems, called *modular tree transducers* [EV91]. Originally introduced as a language theoretic device to formally capture the class of primitive recursive tree functions, KÜHNEMANN in [Küh98] proposed tree transducers to optimize restricted functional programs by exploiting famous composition and decomposition results (e.g. [EV91, Küh97]). The particular construction, which is the basis of our direct construction, on modular tree transducers is studied in [KGK01].

In this section we establish this setting by specifying the syntax of the rewrite rules, also called equations. Proceeding along, we introduce modular tree transducers and, lastly, we associate a formal semantics to those devices, thereby defining the computed translation. Additionally we refine the non-deterministic derivation relation towards a deterministic one using the call-by-name strategy with variable sharing (together commonly named call-by-need) to more appropriately model the computation process of lazy modern functional programming languages (e.g. HASKELL [Tho99])

3.1 Syntax

Modular tree transducers are special term rewriting systems and as such they possess rewrite rules (equations) to control rewriting. The left hand sides of those equations are linear [18] and non-overlapping [19]; those properties will trivially be met in Definition 3.3. However, the restrictions on the right hand sides will be captured separately in the next definition. In contrast to macro tree transducers [EV85] [20], a modular tree transducer partitions its states (function symbols) into modules, where a module contains all rewrite rules for the function symbols of that particular module.

Definition 3.1 (Modular tree transducer right hand sides). The set of *right hand sides* $\mathrm{RHS}(k, F, m, \Delta, V_x, V_y)$ of the module $k \in \mathbb{N}_+$ is the following set of terms RHS made of *function symbols* in the signature F, mapped to modules via $m : F \longrightarrow \mathbb{N}_+$, and *constructors* of the signature Δ over *input subtree variables* $V_x \subseteq X$ and *output subtree variables* $V_y \subseteq Y \cup Z$, such that

- $V_y \subseteq \mathrm{RHS}$,
- for every $n \in \mathbb{N}$, $\delta \in \Delta^{(n)}$ and $t_1, \ldots, t_n \in \mathrm{RHS}$ also $(\delta\, t_1\, \ldots\, t_n) \in \mathrm{RHS}$,
- for every $n \in \mathbb{N}$, $f \in F^{(n+1)}$ with $m(f) > k$ and $t, t_1, \ldots, t_n \in \mathrm{RHS}$ we have $(f\, t\, t_1\, \ldots\, t_n) \in \mathrm{RHS}$,
- for every $n \in \mathbb{N}$, $f \in F^{(n+1)}$ with $m(f) = k$ and $x \in V_x$, $t_1, \ldots, t_n \in \mathrm{RHS}$ also $(f\, x\, t_1\, \ldots\, t_n) \in \mathrm{RHS}$ and

[18] A certain variable occurs at most once.

[19] Overlapping would give rise to critical pairs, thus excluding overlaps and enforcing linearity of the left hand sides immediately enforces confluence [BN98].

[20] All function symbols of a macro tree transducer occur in the same partition.

- all elements of RHS can be obtained using a finite number of applications of the above rules.

Thus $\mathrm{RHS}(k, F, m, \Delta, V_x, V_y) \subseteq \mathcal{T}_{F\cup\Delta}(V_x \cup V_y)$. Furthermore we define for disjoint function symbol signatures F_1 and F_2

$$\mathrm{RHS}_{F_1,F_2,\Delta}(V_x, V_y) = \mathrm{RHS}(1, F_1 \cup F_2, m', \Delta, V_x, V_y),$$

where m' is defined by $m'(F_1) = \{1\}$ and $m'(F_2) = \{2\}$. □

Compared to [EV91], we defined the admissible right hand sides inductively and, furthermore, defined permissible right hand sides for particular modular tree transducers as a notational convenience; to be used when dealing with 2-modular tree transducers. The first parameter of a function symbol is usually called *recursion argument*, while all other parameters are called *context parameters*. Roughly speaking, a right hand side must be a valid term over the function symbols, constructors and variables with three notable limitations. Namely input subtree variables can only occur as recursion parameters of function symbols defined in the same module, secondly a module must not call function symbols (i.e. such function symbols are prohibited to occur) of a module with a lower module number, and, finally, if a module calls a function symbol defined in the same module, then the recursion argument is restricted to some input subtree variable.

Example 3.2 (Right hand sides). Figure 2 presents the right hand sides, which are both right hand sides out of Example 3.4, as trees.

- Let $\Delta = \{S^{(1)}, Z^{(0)}\}$, $F = \{\mathrm{fact}^{(1)}, \mathrm{mult}^{(2)}, \mathrm{id}^{(1)}, \mathrm{add}^{(2)}\}$ and the module mapping m shall be defined as follows: $m(\mathrm{add}) = 3$, $m(\mathrm{mult}) = 2$ and $m(\mathrm{fact}) = m(\mathrm{id}) = 1$. Then
$$\big(\mathrm{mult}\ (\mathrm{fact}\ x_1)\ (S\ (\mathrm{id}\ x_1))\big) \ \in\ \mathrm{RHS}(1, F, m, \Delta, \{x_1\}, \emptyset).$$
- Let $\Delta = \{S^{(1)}, Z^{(0)}\}$, $F_1 = \{\mathrm{mult}^{(2)}\}$ and $F_2 = \{\mathrm{add}^{(2)}\}$. Then
$$\big(\mathrm{add}\ y_1\ (\mathrm{mult}\ x_1\ y_1)\big) \ \in\ \mathrm{RHS}_{F_1,F_2,\Delta}(\{x_1\}, \{y_1\}).$$

□

Having defined the shape of the right hand sides, we are ready to introduce modular tree transducers and some syntactic restrictions thereof. Specifically, we will establish the property of a modular tree transducer being k-modular (i.e. having $k \in \mathbb{N}_+$ modules), and in case $k = 1$ we basically [21] specify a macro tree transducer [EV85]. If, furthermore, all function symbols of the macro tree transducer have rank 1, we gain the classical top-down tree transducer [Rou70, Eng75, Tha70].

Since our main interest is functional program optimization, we will immediately restrict ourselves to total and deterministic modular tree transducers. Additionally, we define 1-ary [22] modular tree transducers according to [EV91]. Lastly, there is another slight difference to the original definition of [EV91], namely we require that input subtree variables occur only in recursion arguments. This deviation can also be found in [KGK01], where the implications are mentioned as well [23].

[21] Macro tree transducers usually endorse a stricter typing by separating input and output signatures.

[22] We can model k-ary modular tree transducers using the construction in Lemma 5.4 of [EV91].

[23] Basically we replace a free occurrence of x according to the original definition of [EV91] by (id x), where id is a function symbol of rank 1 defined in the current module and computes the identity function.

Definition 3.3 (Modular tree transducer). A *modular tree transducer* M is a quintuple (F, m, Δ, e, R), where:

- F is the signature of *function symbols* with $F^{(0)} = \emptyset$,
- $m : F \longrightarrow \mathbb{N}_+$ is the *module mapping* associating every function symbol its module number,
- Δ is the *working symbol* signature disjoint to F,
- $e = (f\ x\ t_1\ \ldots\ t_n)$ is the *initial expression* with the free variable x and $t_i \in \mathcal{T}_\Delta$ for each $i \in [n]$ with $n \in \mathbb{N}$ and some $f \in {m^{-1}(1)}^{(n+1)}$ and
- R contains for every $k, r \in \mathbb{N}$ with $f \in F^{(r+1)}$ and $\delta \in \Delta^{(k)}$ exactly one equation:

$$f(\delta\ x_1\ x_2\ \ldots\ x_k)\ y_1\ y_2\ \ldots\ y_r = \mathrm{rhs}_M(f, \delta),$$

  where $\mathrm{rhs}_M(f, \delta) \in \mathrm{RHS}(m(f), F, m, \Delta, X_k, Y_r)$.

A modular tree transducer is *k-modular*, if and only if $k = \mathrm{card}(m(F))$. 1-modular tree transducers are also called *macro tree transducers* [EV85] and we will omit the irrelevant module mapping in this case, gaining (F, Δ, e, R). Additionally we define the shorthand $\mathrm{RHS}_{\mathrm{MAC}}(F, \Delta, V_x, V_y) = \mathrm{RHS}(1, F, \{\ (f, 1) \mid f \in F\ \}, \Delta, V_x, V_y)$, which is especially useful with macro tree transducers. $\square$

Example 3.4 (Modular tree transducer). We define the 3-modular tree transducer $M_{\mathrm{fact}} = (F, m, \Delta, (\mathrm{fact}\ x), R)$ with

- $\Delta = \{S^{(1)}, Z^{(0)}\}$,
- $F = \{\mathrm{fact}^{(1)}, \mathrm{mult}^{(2)}, \mathrm{id}^{(1)}, \mathrm{add}^{(2)}\}$,
- $m(\mathrm{fact}) = m(\mathrm{id}) = 1$, $m(\mathrm{mult}) = 2$ and $m(\mathrm{add}) = 3$ and
- R contains the equations:

$$\begin{array}{llll}
\mathrm{fact}\ Z & = (S\ Z) & \mathrm{id}\ Z & = Z \\
\mathrm{fact}\ (S\ x_1) & = \mathrm{mult}\ (\mathrm{fact}\ x_1)\ (S\ (\mathrm{id}\ x_1)) & \mathrm{id}\ (S\ x_1) & = S\ (\mathrm{id}\ x_1) \\
\mathrm{mult}\ Z\ y_1 & = Z & \mathrm{add}\ Z\ y_1 & = y_1 \\
\mathrm{mult}\ (S\ x_1)\ y_1 & = \mathrm{add}\ y_1\ (\mathrm{mult}\ x_1\ y_1) & \mathrm{add}\ (S\ x_1)\ y_1 & = S\ (\mathrm{add}\ x_1\ y_1).
\end{array}$$

Intuitively, this modular tree transducer shall compute the factorial of a natural number given as a term of a successor algebra [Pot90], however, we will not be able to formally justify this claim till the next subsection. $\square$

The contribution of this thesis is concerned with 2-modular tree transducers with restrained modules, hence we introduce the notion of a macro tree transducer module. Roughly speaking, a macro tree transducer module is just the set of function symbols of that particular module together with all the equations for those function symbols. Consequently, a modular tree transducer can be specified by listing all its modules (in proper sequence) together with an initial expression.

Additionally, we introduce the notion of a top-down tree transducer module, which is a module, of which the function symbols have rank 1 solely. Consequently, a top-down tree transducer module can be seen as a top-down tree transducer [Rou70, Eng75, Tha70] with external function calls.

Definition 3.5 (Macro tree transducer module). Let $M = (F, m, \Delta, e, R)$ be a modular tree transducer, then the n-th *macro tree transducer module* (or just n-th *module*) of M is (F', E', Δ, R'), where for some $n \in m(F)$:

- $F' = m^{-1}(n)$,
- $E' = \{\, f \in F \mid m(f) > n \,\}$ and
- $R' = \{\, \rho \in R \mid \rho = (f \ldots = \ldots),\ f \in m^{-1}(n) \,\}$.

E' is called the set of *external function symbols*. Specifically, we call a macro tree transducer module a *top-down tree transducer module*, if and only if $F' = F'^{(1)}$. □

Example 3.6 (Macro tree transducer module). The second module of the modular tree transducer M_{fact} of Example 3.4 is $M = (\{\text{mult}^{(2)}\}, \{\text{add}^{(2)}\}, \Delta, R)$ with

- $\Delta = \{S^{(1)}, Z^{(0)}\}$ and
- R contains the equations:

$$\begin{array}{llcll} \text{mult} & Z & y_1 & = & Z \\ \text{mult} & (S\, x_1) & y_1 & = & \text{add}\, y_1\, (\text{mult}\, x_1\, y_1). \end{array}$$

Note that the first module of M_{fact} is indeed a top-down tree transducer module with external function symbols mult and add, though only the external function symbol mult is referenced in the right hand sides of its equations. □

3.2 Semantics

Having settled the syntax of a modular tree transducer, we assign a formal semantics to the newly defined construct. Intuitively, a modular tree transducer computes by applying rewrite rules stepwise starting with the instantiated initial expression, where the free occurrence of the variable x is replaced by a concrete input tree. The meaning of this particular input tree is then defined to be the output tree which is the normal form of that derivation process. A translation, in the sense of a mapping, is only achieved, if for each input tree, the output tree and thus the normal form exists. In [EV91] exactly this property was shown for the non-deterministic derivation relation.

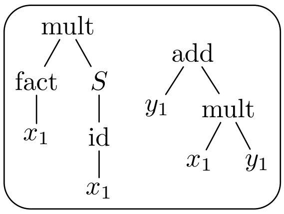

Figure 2: Tree representation for the right hand sides of Example 3.2

We start by defining the set of sentential forms, of which the elements are the terms which may be encountered during a derivation. Compared to the definition found in [EV91], we generalize the notion of a sentential form to include input as well as output subtree variables at certain positions. In the derivation relation we will instantiate a left hand side of an equation and replace this instantiated left hand side by the corresponding instantiated right hand side. To model this process in a uniform framework (where right hand sides and sentential forms can be treated), we introduce this generalization.

Definition 3.7 (Sentential forms). The set of *sentential forms* $\mathrm{SF}_{V_x,V_y}(M)$ of a modular tree transducer $M = (F, m, \Delta, e, R)$ with input subtree variables of V_x and output subtree variables of V_y is the set SF, where

- for every $k \in \mathbb{N}$, $s_1, \ldots, s_k \in \mathcal{T}_\Delta \cup V_x$ and $\xi \in \mathrm{RHS}(1, F, m, \Delta, X_k, V_Y)$

$$\xi\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k \,\} \in \mathrm{SF}$$

- and every element of SF can be derived using the above mentioned rule.

If $V_x = V_y = \emptyset$, we will simply write $\mathrm{SF}(M)$ instead of $\mathrm{SF}_{\emptyset,\emptyset}(M)$. Since for every $k \in \mathbb{N}$, $n \in \mathbb{N}_+$ with $n > 1$, grounding substitution $\theta = \{\, x_1 \mapsto t_1, \ldots, x_k \mapsto t_k \,\}$ with $t_1, \ldots, t_k \in \mathcal{T}_\Delta$ and $\mathrm{rhs} \in \mathrm{RHS}(n, F, m, \Delta, X_k, V_Y) : \mathrm{rhs}\theta \in \mathrm{RHS}(1, F, m, \Delta, X_k, V_Y)$, the instantiation of $\mathrm{RHS}(1, F, m, \Delta, X_k, V_Y)$ is actually sufficient [24]. □

This definition will be illustrated in a more elaborate example following the next definition, whereas each right hand side of Example 3.2 qualifies as an example for the generalized notion using variables [25]. The next step is to fix, how rewriting is applied. This will be done by defining a derivation relation which is induced by a given modular tree transducer [EV91]. Finally, we define the semantics of a modular tree transducer as a mapping (called translation) on terms by relating an input tree and the output tree gained by rewriting the initial expression with the input tree substituted in to normal form (using the derivation relation).

Definition 3.8 (Derivation relation and translation). Let $M = (F, m, \Delta, e, R)$ be a modular tree transducer. The *derivation relation* induced by M is the relation $\Rightarrow_M \subseteq \mathrm{SF}(M)^2$ with $\xi_1, \xi_2 \in \mathrm{SF}(M)$ and $\xi_1 \Rightarrow_M \xi_2$, if and only if

1. **Locate redex:** there exists $p \in \mathrm{occ}(\xi_1)$ with $\xi_1|_p = f\ (\delta\ s_1\ \ldots\ s_k)\ t_1\ \ldots\ t_r$, where $k, r \in \mathbb{N}$, $f \in F^{(r+1)}$, $\delta \in \Delta^{(k)}$, $s_1, \ldots, s_k, t_1, \ldots, t_r \in \mathrm{SF}(M)$,
2. **Build:** $t = \mathrm{rhs}_M(f, \delta)\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k,\ y_1 \mapsto t_1, \ldots, y_r \mapsto t_r \,\}$, if R contains a rule $f\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r = \mathrm{rhs}_M(f, \delta)$, and
3. **Replace:** $\xi_2 = \xi_1[t]_p$.

The induced *translation* is the mapping $\tau_M : \mathcal{T}_\Delta \longrightarrow \mathcal{T}_\Delta$, defined by

$$\tau_M(t) = \mathrm{nf}_{\Rightarrow_M}(e\{\, x \mapsto t \,\})$$

for every $t \in \mathcal{T}_\Delta$. □

The next theorem justifies that the translation is a valid mapping.

Theorem 3.9 (Existence and uniqueness of the normal form). *For every modular tree transducer $M = (F, m, \Delta, e, R)$ the induced derivation relation $\Rightarrow_M$ is terminating and confluent (hence canonical), and admits a unique normal form $\mathrm{nf}_{\Rightarrow_M}(\xi) \in \mathcal{T}_\Delta$ for every $\xi \in \mathrm{SF}(M)$.*

---

[24] Obviously, when proving statements over a sentential form, then we will often refer to the cases of Definition 3.1 of right hand sides. Moreover, we note that input subtree variables can only occur as recursion arguments of function symbols f with $m(f) = 1$.

[25] Both examples are right hand sides of a function symbol f with $m(f) = 1$, we instantiate using the identity substitution and, thus gain e.g. (mult (fact x_1) (S (id x_1))) $\in \mathrm{SF}_{\{x_1\},\emptyset}(M_{\mathrm{fact}})$.

Proof. Immediate from [EV91] and e.g. [BN98]. □

Example 3.10 (Derivation using M_{fact}). We assume the modular tree transducer M_{fact} of Example 3.4. In Figure 3 we illustrate the connection between the rewrite rules and their application in a derivation step and, additionally, we provide a complete derivation. Each term which occurs in the derivation is a sentential form (cf. Definition 3.7), i.e. is an element of $\text{SF}(M_{\text{fact}})$. In each sentential form (except the normal form) we underlined the redex, which we found in step 1 of Definition 3.8.

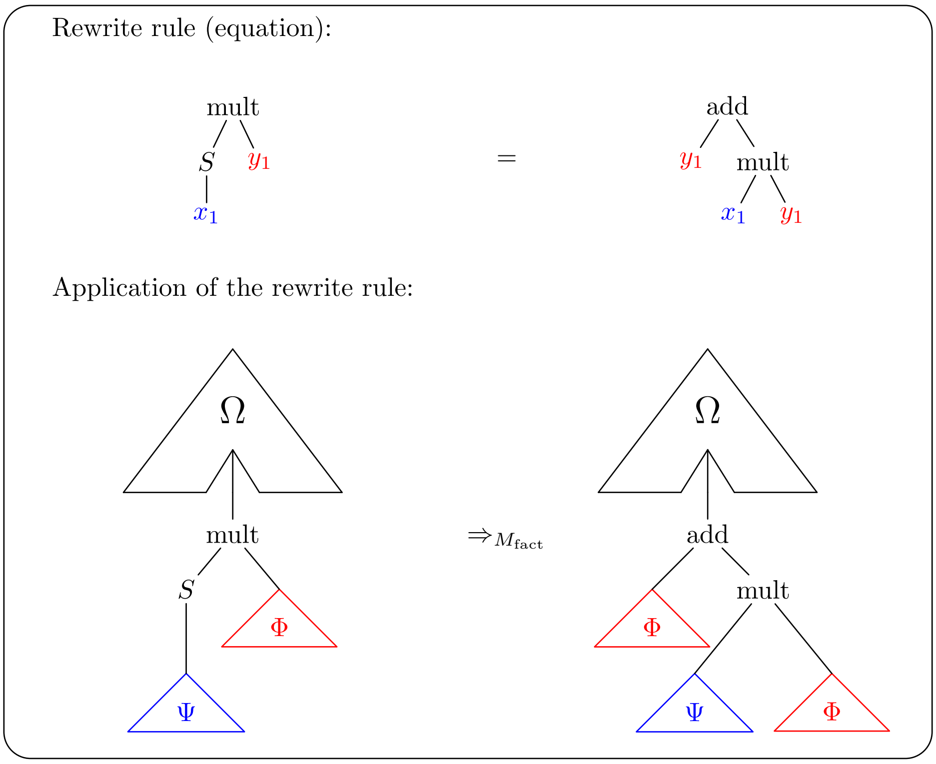

Figure 3: Rewrite rule and derivation step compared

$$
\begin{aligned}
& \underline{\text{fact}\ (S\ (S\ Z))} \Rightarrow_{M_{\text{fact}}} \text{mult}\ (\text{fact}\ (S\ Z))\ (S\ (\underline{\text{id}\ (S\ Z)}))\\
\Rightarrow_{M_{\text{fact}}}\ & \text{mult}\ (\text{fact}\ (S\ Z))\ (S\ (S\ (\underline{\text{id}\ Z}))) \Rightarrow_{M_{\text{fact}}} \text{mult}\ (\underline{\text{fact}\ (S\ Z)})\ (S\ (S\ Z))\\
\Rightarrow_{M_{\text{fact}}}\ & \text{mult}\ (\text{mult}\ (\underline{\text{fact}\ Z})\ (S\ (\text{id}\ Z)))\ (S\ (S\ Z))\\
\Rightarrow_{M_{\text{fact}}}\ & \text{mult}\ (\text{mult}\ (S\ Z)\ (S\ (\underline{\text{id}\ Z})))\ (S\ (S\ Z))\\
\Rightarrow_{M_{\text{fact}}}\ & \text{mult}\ (\underline{\text{mult}\ (S\ Z)\ (S\ Z)})\ (S\ (S\ Z))\\
\Rightarrow_{M_{\text{fact}}}\ & \text{mult}\ (\underline{\text{add}\ (S\ Z)\ (\text{mult}\ Z\ (S\ Z))})\ (S\ (S\ Z))\\
\Rightarrow_{M_{\text{fact}}}\ & \text{mult}\ (S\ (\text{add}\ Z\ (\underline{\text{mult}\ Z\ (S\ Z)})))\ (S\ (S\ Z))\\
\Rightarrow_{M_{\text{fact}}}\ & \text{mult}\ (S\ (\underline{\text{add}\ Z\ Z}))\ (S\ (S\ Z)) \Rightarrow_{M_{\text{fact}}} \underline{\text{mult}\ (S\ Z)\ (S\ (S\ Z))}\\
\Rightarrow_{M_{\text{fact}}}\ & \text{add}\ (S\ (S\ Z))\ (\underline{\text{mult}\ Z\ (S\ (S\ Z))}) \Rightarrow_{M_{\text{fact}}} \underline{\text{add}\ (S\ (S\ Z))\ Z}\\
\Rightarrow_{M_{\text{fact}}}\ & S\ (\underline{\text{add}\ (S\ Z)\ Z}) \Rightarrow_{M_{\text{fact}}} S\ (S\ (\underline{\text{add}\ Z\ Z}))\\
\Rightarrow_{M_{\text{fact}}}\ & S\ (S\ Z)
\end{aligned}
$$

Indeed the induced translation $\tau_{M_{\text{fact}}}$ is the following mapping

$$\tau_{M_{\text{fact}}}(S^n\ Z) = (S^{n!}\ Z) \quad \text{where for every } n \in \mathbb{N} :\ (S^n\ Z) = (\underbrace{S\ (S\ (\dots (S}_{n\text{-times S}}\ Z)\dots))).$$

Although the induced non-deterministic derivation relation incorporates a so-called "don't-care" non-determinism [26] with respect to the gained normal form, the choice of the redexes (the only non-deterministic part of the process) does affect the number of derivation steps necessary to reach the normal form. To formally capture the options, we first introduce the set of redexes of a given sentential form and afterwards highlight the influence of the choice of redexes on the number of derivation steps on an example.

Definition 3.11 (Set of redexes). Let $M = (F, m, \Delta, e, R)$ be a modular tree transducer. We define the set of redexes $R_M(\xi) \subseteq \text{occ}(\xi)$ for every $\xi \in \text{SF}(M)$ to be

$$R_M(\xi) = \{\, p \in \text{occ}(\xi) \mid \text{label}_\xi(p) \in F,\ \text{label}_\xi(p.1) \in \Delta \,\}.$$

$\square$

Example 3.12 (The choice of the redexes matters). Let M_{fact} be the modular tree transducer of Example 3.4. Let

$$\xi_1 = \text{mult}(\text{mult}(SZ)(S(\text{id}Z)))(S(SZ)) \quad \text{and} \quad \xi_2 = \text{mult}(\text{add}(SZ)(\text{mult}Z(SZ)))(S(SZ)),$$

in Example 3.10 we showed that $\xi_1 \Rightarrow_M^2 \xi_2$, but $R_M(\xi_1) = \{1, 1.2.1\}$ contains two redexes and selecting the other redex at position 1 (the redex at position 1.2.1 was chosen in Example 3.10) yields the partial derivation (chosen redexes underlined):

$$\begin{aligned}
& \text{mult}\ (\underline{\text{mult}\ (S\ Z)\ (S\ (\text{id}\ Z))})\ (S\ (S\ Z)) \\
\Rightarrow_M\ & \text{mult}\ (\text{add}\ (S\ (\underline{\text{id}\ Z}))\ (\text{mult}\ Z\ (S\ (\text{id}\ Z))))\ (S\ (S\ Z)) \\
\Rightarrow_M\ & \text{mult}\ (\text{add}\ (S\ Z)\ (\text{mult}\ Z\ (S\ (\underline{\text{id}\ Z}))))\ (S\ (S\ Z)) \\
\Rightarrow_M\ & \text{mult}\ (\text{add}\ (S\ Z)\ (\text{mult}\ Z\ (S\ Z)))\ (S\ (S\ Z)).
\end{aligned}$$

Obviously this alternative requires 3 derivation steps to gain the same result, hence also $\xi_1 \Rightarrow_M^3 \xi_2$ and thereby fact $(S\ (S\ Z)) \Rightarrow_M^{15} S\ (S\ Z)$ and fact $(S\ (S\ Z)) \Rightarrow_M^{16} S\ (S\ Z)$, if we draw all other choices as we did in Example 3.10. $\square$

3.3 Call-by-need derivation relation

In this thesis we want to analyze the efficiency impact of a certain construction, thus we need an efficiency measure to compare the efficiency of the original modular tree transducer to the one of the result. An appealing and simple measure of the computation time [27] is the number of derivation steps needed to compute the normal form. As we have seen in Example 3.12 the non-deterministic derivation relation seems unfit for this purpose, which

[26] A "don't-care" non-determinism offers some choice which does not influence the final result; here: the normal form. Therefore, the choice does not matter and we "don't-care"; opposite is a "don't-know" non-determinism where the choice might influence the result, so we "don't-know" which option to choose to gain the desired result (in the worst case we have to try all options).

[27] We are not trying to establish a true measure of the computation time consumed on a particular architecture, rather we search a measure such that the computation time scales according to it.

is supported by the fact that most implementations use a deterministic derivation relation by fixing a strategy to locate the redex required in step 1 of Definition 3.8 (e.g. normal order reduction strategy or applicative order reduction strategy).

Thus, we fix a deterministic derivation relation by selecting the normal order reduction strategy supplemented by an implementation detail; namely the use of pointers to reference subterms. The use of pointers avoids an arising cause of inefficiency when applying normal order reduction strategy: Values which were already computed are recomputed due to a copied sentential form containing redexes. In Example 3.12 this copying happens in the first step, where the redex (id Z) is duplicated and evaluated twice independently in the following steps. Obviously, the duplication was triggered by the non-linearity of the right hand side of mult at S (cf. Example 3.4), where the context parameter y_1 occurs twice.

Implementors opted to reference the subterms matched against the variables in the derivation step via pointers, instead of copying the matched subterms. This is called *variable sharing* and together with the normal order reduction strategy, the such defined derivation relation is called *call-by-need* derivation relation [AFM+95, BKdV+02, BW88]. It is implemented in the functional programming language Haskell [Tho99], for example.

The rest of this section is devoted to establishing the call-by-need derivation relation by introducing sentential form graphs, the set of redexes of a sentential form graph and finally stating the call-by-need derivation relation.

Definition 3.13 (Sentential form graphs). We define the set $\mathrm{SFG}_{V_X,V_Y}(M)$ for sets $V_X \subseteq X$, $V_Y \subseteq Y \cup Z$ and for every modular tree transducer $M = (F, m, \Delta, e, R)$ by stating

$$\mathrm{SFG}_{V_X,V_Y}(M) = \{\, \Xi \in \mathcal{TG}(\mathcal{T}_{F\cup\Delta}(V_X \cup V_Y)) \mid \mathrm{term}(\Xi) \in \mathrm{SF}_{V_X,V_Y}(M) \,\}.$$

As a shorthand we also define $\mathrm{SFG}(M) = \mathrm{SFG}_{\emptyset,\emptyset}(M)$. □

We immediately notice that any sentential form of a modular tree transducer M corresponds to a sentential form graph of M by using the tree representation of that sentential form and exploiting the one-to-one correspondence between terms and tree representations. We proceed with the definition of the redexes in a given term graph, where also term graph rewrite rules are introduced. Thereafter, we define the call-by-need derivation relation.

Definition 3.14 (Set of redexes in a term graph). Let $M = (F, m, \Delta, e, R)$ be a modular tree transducer. For each $\rho \in R$ with $\rho = (\mathrm{lhs} = \mathrm{rhs})$ we denote the *variable-shared term graph rewrite rule* corresponding to ρ by the variable-shared term graph G_ρ in $\mathcal{T}_{F\cup\Delta\cup\{=^{(2)}\}}(\mathrm{var}(\mathrm{lhs}))$ with $\mathrm{term}(G_\rho) = (=\ \mathrm{lhs}\,\mathrm{rhs})$. Additionally, as a notational convenience we define $G_{\rho,\mathrm{lhs}} = G_\rho|_1$ and $G_{\rho,\mathrm{rhs}} = G_\rho|_2$. The set $R_M(\Xi) \subseteq \mathrm{occ}(\Xi)$ of redexes in a sentential form graph $\Xi \in \mathrm{SFG}(M)$ is defined as:

$$R_M(\Xi) = \{\, p \in \mathrm{occ}(\Xi) \mid \rho \in R, \text{ there is a term graph homomorphism } \psi \text{ from } G_\rho|_1 \text{ to } \Xi|_p \,\}$$

□

Example 3.15 (Set of redexes in a term graph). Consider the sentential form graph $\Xi \in \mathrm{SFG}(M)$ of Figure 4 with a modular tree transducer

$$M = (\{f^{(3)},\ g^{(3)}\}, m, \{A^{(1)},\ N^{(0)}\}, e, R)$$

with the equations $\rho_1 = (f\ N\ y_1\ y_2\ =\ N)$ and $\rho_2 = (g\ N\ y_1\ y_2\ =\ N)$ in R and $m(f) = 1$, $m(g) = 2$ [28]. We assume the nodes of Figure 4. Then

$$R_M(\Xi) = \{\varepsilon, 2\}$$

and the necessary term graph homomorphisms are ψ_1 from $G_{\rho_1,\mathrm{lhs}}$ to $\Xi|_\varepsilon$ and ψ_2 from $G_{\rho_2,\mathrm{lhs}}$ to $\Xi|_2$, defined by

$$\psi_1(m_1) = n_1 \quad \psi_1(m_2) = n_2 \quad \psi_1(y_1) = n_3 \quad \psi_1(y_2) = n_6$$

and

$$\psi_2(k_1) = n_3 \quad \psi_2(k_2) = n_2 \quad \psi_2(y_1) = n_5 \quad \psi_2(y_2) = n_4.$$

One easily verifies that the specified mappings are term graph homomorphisms. □

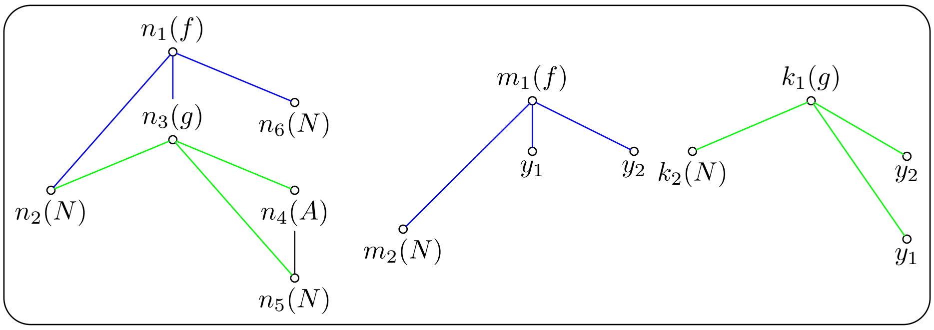

Figure 4: A sample sentential form graph Ξ and two term graphs $G_{\rho_1,\mathrm{lhs}}, G_{\rho_2,\mathrm{lhs}}$ representing left hand sides

Definition 3.16 (Call-by-need derivation relation). Let $M = (F, m, \Delta, e, R)$ be a modular tree transducer. The *call-by-need* derivation relation $\Mapsto_M$ on sentential form graphs induced by M relates $\Xi_1 \Mapsto_M \Xi_2$ for every $\Xi_1, \Xi_2 \in \mathrm{SFG}(M)$, if and only if

1. **Locate redex:** there exists a least element $p \in R_M(\Xi_1)$ with respect to the lexicographic ordering on paths, which requires a term graph homomorphism ψ from $G_{\rho,\mathrm{lhs}}$ to $\Xi_1|_p$ for some $\rho \in R$,

2. **Build:** let $G_{\rho,\mathrm{lhs}} = (N_{\mathrm{lhs}}, E_{\mathrm{lhs}}, l_{\mathrm{lhs}}, r_{\mathrm{lhs}})$ and $G_{\rho,\mathrm{rhs}} = (N_{\mathrm{rhs}}, E_{\mathrm{rhs}}, l_{\mathrm{rhs}}, r_{\mathrm{rhs}})$ with $N_{\mathrm{rhs}} \cap N_{\Xi_1} = \emptyset$, then $G' = (N', E', l', r_{\Xi_1})$ is defined as

$$N' = N_{\Xi_1} \cup (N_{\mathrm{rhs}} \setminus N_{\mathrm{lhs}})$$
$$E'(n,i) = \begin{cases} E_{\Xi_1}(n,i) & \text{, if } n \in N_{\Xi_1} \\ E_{\mathrm{rhs}}(n,i) & \text{, if } n, E_{\mathrm{rhs}}(n,i) \in N_{\mathrm{rhs}} \setminus N_{\mathrm{lhs}} \\ \psi(E_{\mathrm{rhs}}(n,i)) & \text{, otherwise} \end{cases}$$
$$l'(n) = \begin{cases} l_{\Xi_1}(n) & \text{, if } n \in N_{\Xi_1} \\ l_{\mathrm{rhs}}(n) & \text{, otherwise} \end{cases}$$

[28] The rest of R and e are irrelevant and, therefore, omitted.

for every $n \in N'$ and $i \in \mathbb{N}_+$ [29] and

3. **Redirect:** $\Xi_2 = G'[\psi(r_{\text{lhs}}) \rightsquigarrow r_{\text{rhs}}]$, where we ensure the connectedness property, i.e. trigger garbage collection, if necessary.

Alternatively, $\Xi_1 \Rrightarrow_M \Xi_2$, if and only if there exists a least element $p \in R_M(\Xi_1)$ with respect to the lexicographic ordering on paths, there exists for some $\rho \in R$ a non-collapsing term graph homomorphism ψ_{lhs} from $G_{\rho,\text{lhs}}$ to $\Xi_1|_p$ and a non-collapsing term graph homomorphism ψ_{rhs} from $G_{\rho,\text{rhs}}$ to $\Xi_2|_p$ with $\psi_{\text{lhs}}(n) = \psi_{\text{rhs}}(n)$ for every $n \in N_{G_{\rho,\text{lhs}}} \cap N_{G_{\rho,\text{rhs}}}$, and, additionally, there exists a term graph isomorphism ϖ from $\Xi_1[\psi_{\text{lhs}}(r_{G_{\rho,\text{lhs}}}) \rightsquigarrow z]$ to $\Xi_2[\psi_{\text{rhs}}(r_{G_{\rho,\text{rhs}}}) \rightsquigarrow z]$ [30] for some $z \notin N_{\Xi_1} \cup N_{\Xi_2} \cup N_{G_\rho}$ [31]. $\square$

Example 3.17 (Call-by-need derivation relation). Throughout this example we will use the modular tree transducer M_{fact} of Example 3.4.

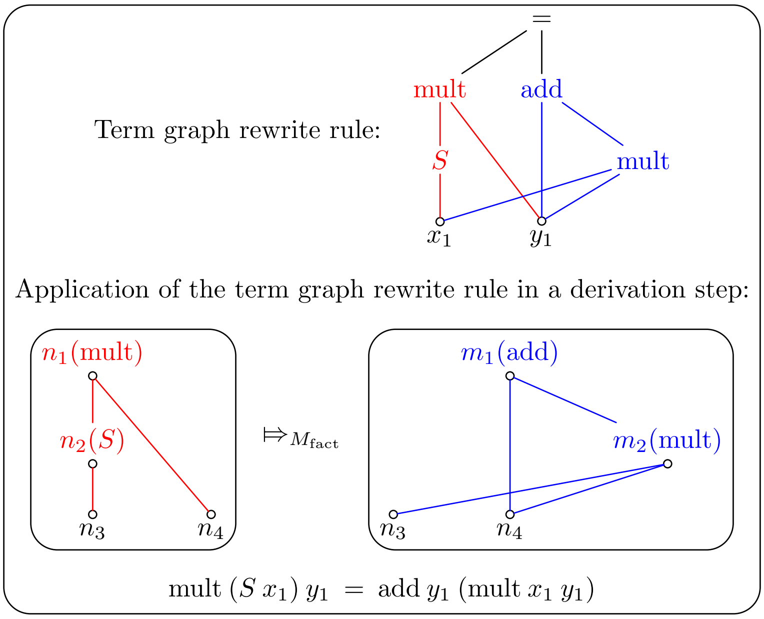

Figure 5: Term graph rewrite rule and call-by-need derivation step

In Figure 5 we try to illuminate the connection between a term graph rewrite rule and its application in a call-by-need derivation step. The fact that the shown redex is leftmost-outermost is not displayed and furthermore all edges pointing to n_1 are redirected to point to m_1. Figure 6 illustrates the actual four-phase process (locate-redex, build, redirect and garbage collect) on an example, where the root of the term graphs is framed. Finally Figure 7 shows part of the call-by-need derivation of fact $(S\ (S\ Z))$. $\square$

[29] We assume that the functions are undefined for the given parameters, if none of the above displayed cases yields a definite result.

[30] We implicitly perform garbage collection.

[31] This approach is less constructive and uses the notion of a context, since both $\Xi_1[\psi_{\text{lhs}}(r_{G_{\rho,\text{lhs}}}) \rightsquigarrow z]$ and $\Xi_2[\psi_{\text{rhs}}(r_{G_{\rho,\text{rhs}}}) \rightsquigarrow z]$ can be interpreted as contexts which do not change (due to the isomorphism). The matching of the left hand side is performed by the homomorphism ψ_{lhs}, while the right hand side matches with the subgraph $\Xi_2|_p$ via homomorphism ψ_{rhs}. The additional restriction on ψ_{lhs} and ψ_{rhs} ensures that variable nodes are instantiated equally.

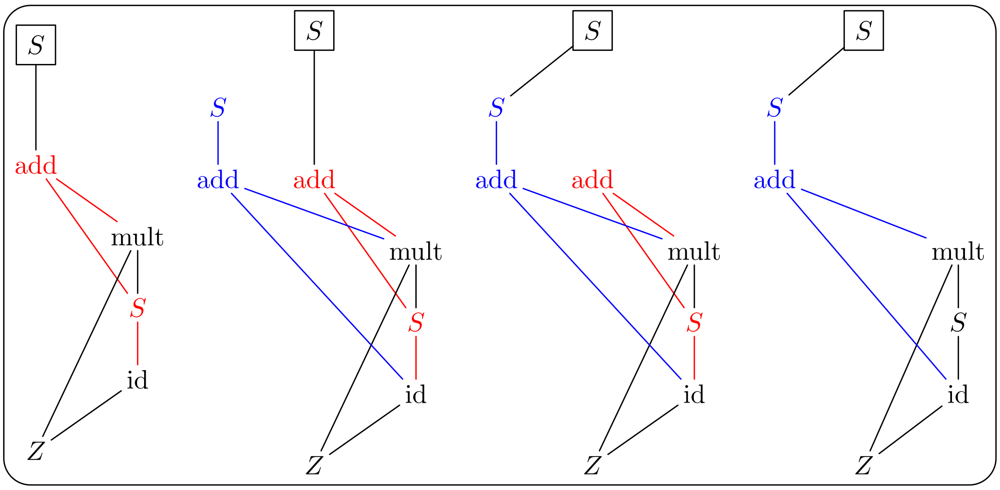

Figure 6: Illustrating the stages (locate redex, build, redirect, garbage collect) of the rewriting process.

We could have introduced another translation, which would be defined by taking the tree representation of $e\{x \mapsto t\}$ (with e being the initial expression of some modular tree transducer $M = (F, m, \Delta, e, R)$ and $t \in \mathcal{T}_\Delta$) and computing the normal form of that tree with respect to the call-by-need derivation relation. The translation would then map t to the term representation of the normal form. However, we conclude this section by stating some important properties of the call-by-need derivation relation mainly taken out of the literature, which specifically ensure that the call-by-need translation coincides with the non-deterministic translation. Additionally, we compare the call-by-need derivation relation to the non-deterministic derivation relation in terms of the number of derivation steps necessary to compute the normal form.

Theorem 3.18 (Soundness and completeness of the call-by-need derivation relation). *Given a modular tree transducer $M = (F, m, \Delta, e, R)$ we relate $\Rightarrow_M$ and $\Mapsto_M$ as follows.*

1. *$\Mapsto_M$ is deterministic, while $\Rightarrow_M$ is nondeterministic.*
2. *$\Mapsto_M$ is canonical as well as $\Rightarrow_M$.*
3. ***Soundness:** If $\Xi_1 \Mapsto_M \Xi_2$ for $\Xi_1, \Xi_2 \in \mathrm{SFG}(M)$, then $\mathrm{term}(\Xi_1) \Rightarrow_M^+ \mathrm{term}(\Xi_2)$.*
4. ***Completeness:** For every $\xi \in \mathrm{SF}(M)$ and $\Xi_1 \in \mathrm{SFG}(M)$ with $\mathrm{term}(\Xi_1) = \xi$, there exists $\Xi_2 \in \mathrm{SFG}(M)$ such that $\Xi_1 \Mapsto_M^* \Xi_2$ and $\mathrm{term}(\Xi_2) = \mathrm{nf}_{\Rightarrow_M}(\xi)$ [32].*
5. *For every $n \in \mathbb{N}$, $\xi \in \mathrm{SF}(M)$ with $\xi \Rightarrow_M^n \mathrm{nf}_{\Rightarrow_M}(\xi)$ and $\Xi_1 \in \mathrm{SFG}(M)$ with $\mathrm{term}(\Xi_1) = \xi$, there exists a unique $\Xi_2 \in \mathrm{SFG}(M)$ such that $\Xi_1 \Mapsto_M^m \Xi_2$ with $m \in \mathbb{N}$ with $\mathrm{term}(\Xi_2) = \mathrm{nf}_{\Rightarrow_M}(\xi)$ and additionally $m \leq n$.*

Proof. The enumeration refers to the enumeration of the statements above.

[32]Note, however, that the following property does not hold: for every $\xi_1, \xi_2 \in \mathrm{SF}(M)$ with $\xi_1 \Rightarrow_M \xi_2$ and $\Xi_1 \in \mathrm{SFG}(M)$ with $\mathrm{term}(\Xi_1) = \xi_1$, there exists $\Xi_2 \in \mathrm{SFG}(M)$ such that $\Xi_1 \Mapsto_M^* \Xi_2$ and $\mathrm{term}(\Xi_2) = \xi_2$.

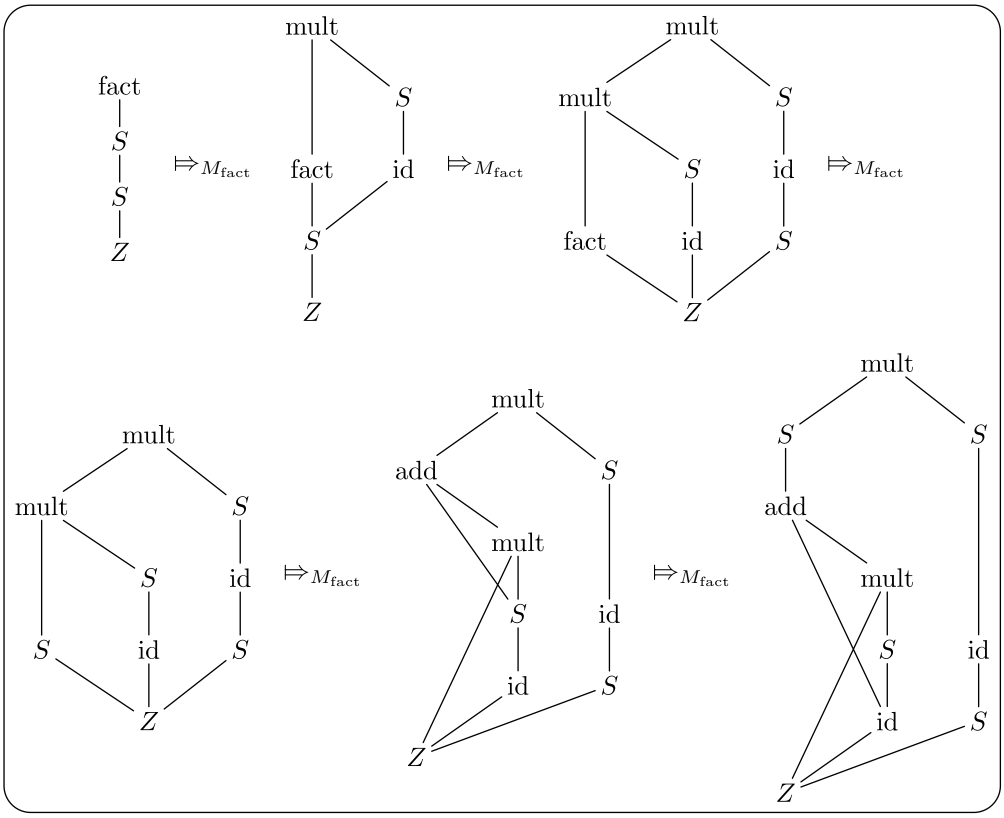

Figure 7: Part of the call-by-need derivation of fact $(S\ (S\ Z))$

(1.) The existence of the least element of $R_M(\Xi)$ for every $\Xi \in \mathrm{SFG}(M)$ is guaranteed, because the lexicographic ordering on paths is total. Thus, only if $R_M(\Xi) = \emptyset$ we encounter a problem, but then $\mathrm{term}(\Xi) \in \mathcal{T}_\Delta$, i.e. Ξ is the normal form. The remaining steps (build, redirect, garbage collect) are obviously deterministic and uniquely determine a resulting sentential form graph.

(2.–4.) Immediate from [BvEG$^+$87] and Theorem 3.9, since $\Rrightarrow_M$ is the rewrite relation of a regular term graph rewriting system (using the vocabulary of [BvEG$^+$87]).

(5.) Immediate from (1.), (3.) and (4.).

□

4 The construction

Given a program written in a functional programming language, we want to alter the program such that the computed function remains the same, but the efficiency of the program is affected positively (or not affected at all). This section is concerned with the process of creating another program, which computes the same function as the original program. In our established setting, programs are modelled by modular tree transducers, which already constitutes a restriction on the input programs.

In the literature [Boi91, Wad87] a strategy called accumulation is studied. Roughly speaking, calls to a function symbol of a substitution module can possibly be delayed or even eliminated by computing in a context parameter. This is possible due to the associativity of the function associated with the function symbol, which is applied to replace expressions like $f\ (f\ x\ y)\ z$ by $f\ x\ (f\ y\ z)$, thereby moving the nested function call from the recursion argument to a context parameter, thus the name accumulation technique.

This section is concerned with an implementation of this optimization strategy in our setting using restricted modular tree transducers. A construction is presented and the correctness of the construction is proven by showing the equivalence of the induced translations using the non-deterministic derivation relations. By Theorem 3.18 this is sufficient to ensure equivalence of the computed normal forms using the call-by-need derivation relations. The efficiency impact of the construction is not studied until the next section.

4.1 Accumulation technique

In [KGK01] an indirect construction, which transforms restricted 2-modular tree transducers into macro tree transducers [33], was studied. The aim of this section is to establish a direct construction equivalent to the one presented in [KGK01]. In order to state the construction formally, we first need to clarify the restrictions; the first module is supposed to be a top-down tree transducer module and the second module should be a substitution module. Thus we should formally define substitution modules.

Definition 4.1 (Substitution module). A module $(F, \emptyset, \Delta, R)$ of a modular tree transducer is called *substitution module*, if and only if there are $\mathrm{mx} \in \mathbb{N}$ and *substitution variables* $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\} \subseteq \Delta^{(0)}$ such that $\mathrm{card}(F^{(\mathrm{mx}+1)}) = 1$, $\mathrm{card}(F^{(i)}) \leq 1$ for every $i \in [\mathrm{mx}]$ and, furthermore, the equations in R are of the following format:

- for every $r \in \mathbb{N}$, $\mathrm{sub}_r \in F^{(r+1)}$ and $j \in [r]$:

$$\mathrm{sub}_r\ \Pi_j\ y_1\ \ldots\ y_r\ =\ y_j$$

- and for every $k, r \in \mathbb{N}$ with $\delta \in \Delta^{(k)} \setminus \{\Pi_1, \ldots, \Pi_r\}$ and $\mathrm{sub}_r \in F^{(r+1)}$:

$$\mathrm{sub}_r\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r\ =\ \delta\ (\mathrm{sub}_r\ x_1\ y_1\ \ldots\ y_r)\ \ldots\ (\mathrm{sub}_r\ x_k\ y_1\ \ldots\ y_r).$$

□

[33] Preserving semantics, and thus the induced translation, of course.

Example 4.2 (Reversing a list). The 2-modular tree transducer M_{rev} is defined as

$$M_{\text{rev}} = (\{\text{rev}^{(1)}, \text{app}^{(2)}\}, m, \{A^{(1)}, B^{(1)}, N^{(0)}\}, (\text{rev}\ x), R)$$

with $m(\text{rev}) = 1$, $m(\text{app}) = 2$ and rule-set R :

$$\begin{array}{lllll|lllll} \text{rev} & N & = & N & & \text{app} & N & y_1 & = & y_1 \\ \text{rev} & (A\,x_1) & = & \text{app}\,(\text{rev}\,x_1)\,(A\,N) & & \text{app} & (A\,x_1) & y_1 & = & A\,(\text{app}\,x_1\,y_1) \\ \text{rev} & (B\,x_1) & = & \text{app}\,(\text{rev}\,x_1)\,(B\,N) & & \text{app} & (B\,x_1) & y_1 & = & B\,(\text{app}\,x_1\,y_1). \end{array}$$

One easily justifies that the induced translation is $\tau_{M_{\text{rev}}}(t) = t^R$ for every $t \in \mathcal{T}_\Delta$, where t^R is the reversed list [34]. This version of the list reversal function is also known as inefficient reverse, the reason why will be made apparent in the next section. □

Example 4.3 (Substitution module). The third module of Example 3.4 is in fact a substitution module with substitution variables $\Pi = \{Z\}$ as well as the second module of Example 4.2 with substitution variables $\Pi = \{N\}$. □

In the next lemma we justify the name substitution module by providing an alternative semantics of a call to a sub_r-function symbol of a substitution module.

Lemma 4.4 (Substitution modules provide substitutions). *Let M be a modular tree transducer of which the n-th module $(F', \emptyset, \Delta, R')$ is a substitution module with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\text{mx}}\}$ with $\text{mx} \in \mathbb{N}$. Moreover, assume $\text{sub}_r \in F^{(r+1)}$ with $r \in \mathbb{N}$, then for every $(\text{sub}_r\ \xi\ \xi_1\ \ldots\ \xi_r) \in \text{SF}(M)$*

$$\text{nf}_{\Rightarrow_M}(\text{sub}_r\ \xi\ \xi_1\ \ldots\ \xi_r) = \text{nf}_{\Rightarrow_M}(\xi)\{\,\Pi_1 \mapsto \text{nf}_{\Rightarrow_M}(\xi_1), \ldots, \Pi_r \mapsto \text{nf}_{\Rightarrow_M}(\xi_r)\,\}.$$

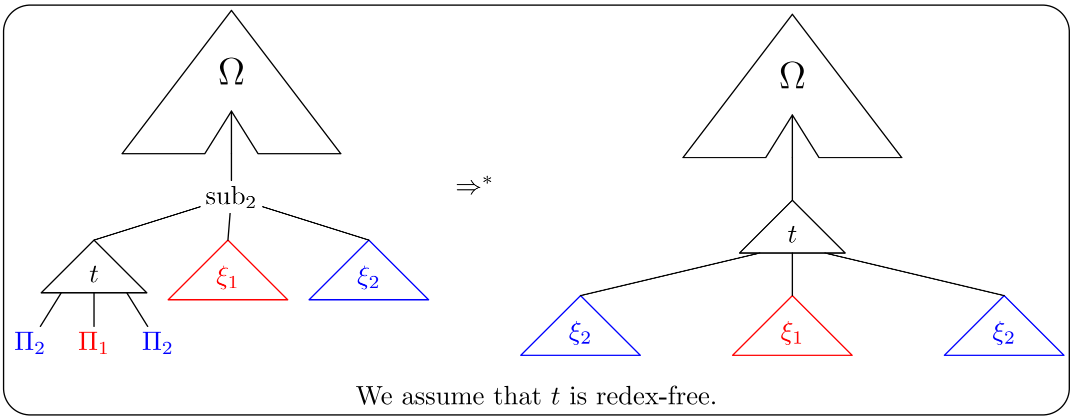

Figure 8: Illustrating the effect of a call to a function symbol of a substitution module, where $t = \text{nf}_{\Rightarrow_M}(\xi)$

Proof. We set $\theta = \{\,\Pi_1 \mapsto \text{nf}_{\Rightarrow_M}(\xi_1), \ldots, \Pi_r \mapsto \text{nf}_{\Rightarrow_M}(\xi_r)\,\}$. Since $\Rightarrow_M$ is the non-deterministic derivation relation, we use the equality

$$\text{nf}_{\Rightarrow_M}(\text{sub}_r\ \xi\ \xi_1\ \ldots\ \xi_r) = \text{nf}_{\Rightarrow_M}(\text{sub}_r\ (\text{nf}_{\Rightarrow_M}(\xi))\ \xi_1\ \ldots\ \xi_r)$$

[34] e.g. $(A\,(B\,(B\,N)))^R = B\,(B\,(A\,N))$

and prove the statement

$$\mathrm{nf}_{\Rightarrow_M}(\mathrm{sub}_r\ (\mathrm{nf}_{\Rightarrow_M}(\xi))\ \xi_1\ \ldots\ \xi_r) = \mathrm{nf}_{\Rightarrow_M}(\xi)\theta$$

via structural induction over $\mathrm{nf}_{\Rightarrow_M}(\xi) \in \mathcal{T}_\Delta$.

- **Induction base:** Let $\mathrm{nf}_{\Rightarrow_M}(\xi) \in \Delta^{(0)}$ in

$$\begin{aligned}
& \mathrm{nf}_{\Rightarrow_M}(\mathrm{sub}_r\ (\mathrm{nf}_{\Rightarrow_M}(\xi))\ \xi_1\ \ldots\ \xi_k) \\
\overset{\text{Def. 4.1}}{=} & \begin{cases} \mathrm{nf}_{\Rightarrow_M}(\xi_j) & \text{, if } \mathrm{nf}_{\Rightarrow_M}(\xi) = \Pi_j \text{ for some } j \in [r] \\ \mathrm{nf}_{\Rightarrow_M}(\mathrm{nf}_{\Rightarrow_M}(\xi)) & \text{, otherwise} \end{cases} \\
= & \begin{cases} \Pi_j\ \theta & \text{, if } \mathrm{nf}_{\Rightarrow_M}(\xi) = \Pi_j \text{ for some } j \in [r] \\ \mathrm{nf}_{\Rightarrow_M}(\xi) & \text{, otherwise } (\mathrm{nf}_{\Rightarrow_M}(\xi) \notin \{\Pi_1, \ldots, \Pi_r\}) \end{cases} \\
= & \mathrm{nf}_{\Rightarrow_M}(\xi)\ \theta.
\end{aligned}$$

- **Induction step:** Let $k \in \mathbb{N}_+$, $\delta \in \Delta^{(k)}$ and $\mathrm{nf}_{\Rightarrow_M}(\xi) = (\delta\ t_1\ \ldots\ t_k)$ for some $t_1, \ldots, t_k \in \mathcal{T}_\Delta$.

  By induction hypothesis $\mathrm{nf}_{\Rightarrow_M}(\mathrm{sub}_r\ t_i\ \xi_1\ \ldots\ \xi_r) = t_i\theta$ for every $i \in [k]$. We deduce

$$\begin{aligned}
& \mathrm{nf}_{\Rightarrow_M}(\mathrm{sub}_r\ (\delta\ t_1\ \ldots\ t_k)\ \xi_1\ \ldots\ \xi_r) \\
\overset{\text{Def. 4.1}}{=} & \mathrm{nf}_{\Rightarrow_M}(\delta\ (\mathrm{sub}_r\ t_1\ \xi_1\ \ldots\ \xi_r)\ \ldots\ (\mathrm{sub}_r\ t_k\ \xi_1\ \ldots\ \xi_r)) \\
= & \delta\ (\mathrm{nf}_{\Rightarrow_M}(\mathrm{sub}_r\ t_1\ \xi_1\ \ldots\ \xi_r))\ \ldots\ (\mathrm{nf}_{\Rightarrow_M}(\mathrm{sub}_r\ t_k\ \xi_1\ \ldots\ \xi_r)) \\
\overset{\text{I.H.}}{=} & \delta\ (t_1\theta)\ \ldots\ (t_k\theta) \\
= & (\delta\ t_1\ \ldots\ t_k)\ \theta \\
= & \mathrm{nf}_{\Rightarrow_M}(\xi)\ \theta.
\end{aligned}$$

Thus, the proof is completed. Figure 8 shows an example of an application of the lemma. □

Using the previous lemma we can argue that one substitution function symbol sub with maximal rank is actually sufficient, since calls to substitution function symbols with lower rank can be emulated by calls to the substitution function symbol sub with the corresponding substitution variables as additional context parameters. This motivates us to introduce $\mathrm{mx} = (\max \mathrm{ar}(m^{-1}(2))) - 1$ context parameters for every function symbol of the first module of a given 2-modular tree transducer $M = (F, m, \Delta, e, R)$, of which the first module is a top-down tree transducer and the second module is a substitution module with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$. Intuitively, we keep track of the replacements in those context parameters, i.e. for every $i \in [\mathrm{mx}]$ we store, whatever is to be substituted for a Π_i in the input, in the i-th context parameter. This idea will prove to be effective, since the associativity of substitutions allows us to simply adjust the replacements without yet performing the substitution itself; thereby accumulating the replacements.

Construction 4.5 (Accumulation technique). Let $M = (F, m, \Delta, e, R)$ be a modular tree transducer with 2 modules, of which the first module is a top-down tree transducer module and the second module is a substitution module with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ for some $\mathrm{mx} \in \mathbb{N}$. We construct a macro tree transducer $M' = (F', \Delta, e', R')$ as follows:

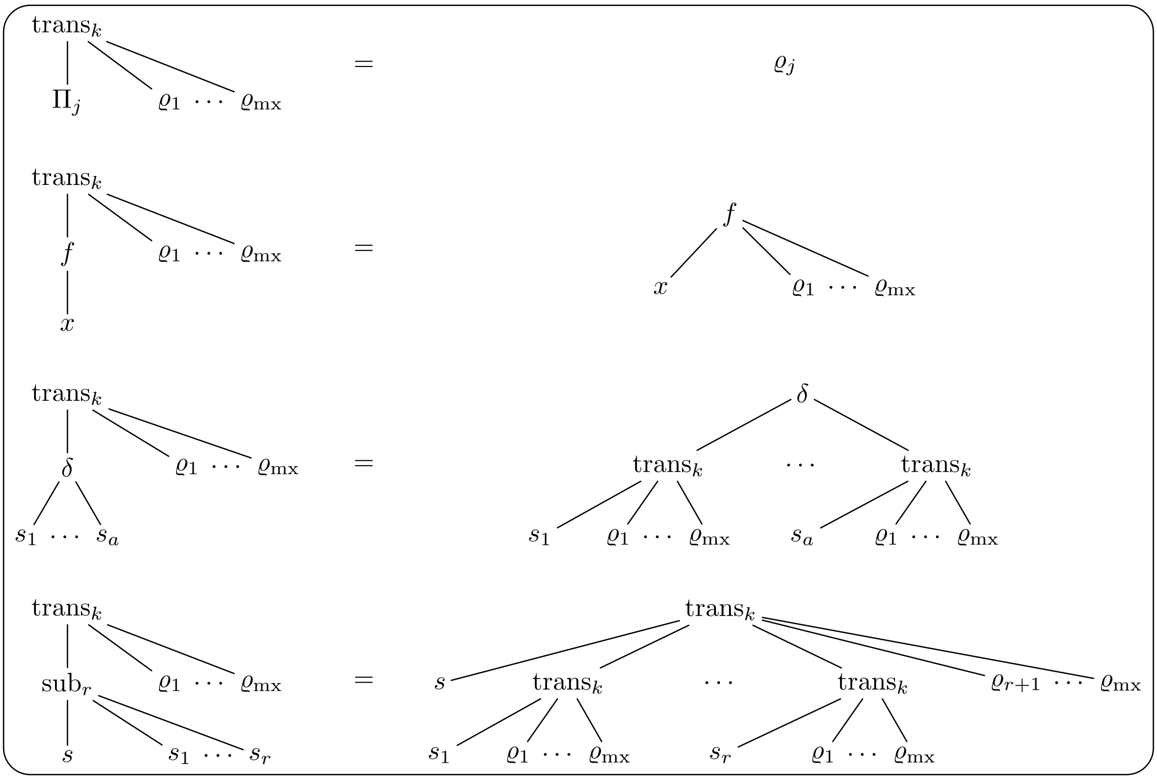

Figure 9: The defining equations of the trans_k-mapping

- $F' = \{\, f^{(\text{mx}+1)} \mid f \in m^{-1}(1)\,\}$,
- $e' = (f\; x\; \Pi_1\; \dots\; \Pi_{\text{mx}})$, if $e = (f\; x)$ with $f \in m^{-1}(1)$, and
- R' contains for every $k \in \mathbb{N}$, $f \in F'$ and $\delta \in \Delta^{(k)}$ with $(f\,(\delta x_1 \dots x_k) = \text{rhs}_M(f,\delta)) \in R$ the equation

$$f\;(\delta\; x_1\; \dots\; x_k)\; y_1\; \dots\; y_{\text{mx}} = \text{trans}_k(\text{rhs}_M(f,\delta), y_1, \dots, y_{\text{mx}}),$$

where for every $k \in \mathbb{N}$, we let $\text{MRHS} = \text{RHS}_{\text{MAC}}(F', \Delta, X_k, Y_{\text{mx}})$, $F_1 = m^{-1}(1)$ and $F_2 = m^{-1}(2)$ in

$$\text{trans}_k : \text{RHS}_{F_1,F_2,\Delta}(X_k, \emptyset) \times \text{MRHS}^{\text{mx}} \longrightarrow \text{MRHS},$$

which is defined for every $\varrho_1, \dots, \varrho_{\text{mx}} \in \text{RHS}(F', \Delta, X_k, Y_{\text{mx}})$ by the following equations, which are also illustrated in Figure 9.

$$\text{trans}_k(\Pi_j, \varrho_1, \dots, \varrho_{\text{mx}}) = \varrho_j \tag{4.1}$$

for every $j \in [\text{mx}]$,

$$\text{trans}_k((f\; x), \varrho_1, \dots, \varrho_{\text{mx}}) = (f\; x\; \varrho_1\; \dots\; \varrho_{\text{mx}}) \tag{4.2}$$

for every $f \in F_1$, $x \in X_k$,

$$\begin{aligned}\mathrm{trans}_k((\delta\, s_1 \ldots s_a), \varrho_1, \ldots, \varrho_{\mathrm{mx}}) = (\delta\, &\mathrm{trans}_k(s_1, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\ &\ldots \\ &\mathrm{trans}_k(s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))\end{aligned} \tag{4.3}$$

for every $a \in \mathbb{N}$, $\delta \in (\Delta^{(a)} \setminus \Pi)$ and $s_1, \ldots, s_a \in \mathrm{RHS}_{F_1,F_2,\Delta}(X_k, \emptyset)$, and

$$\begin{aligned}\mathrm{trans}_k((\mathrm{sub}_r\, s\, s_1 \ldots s_r), \varrho_1, \ldots, \varrho_{\mathrm{mx}}) = \mathrm{trans}_k(s, &\mathrm{trans}_k(s_1, \varrho_1, \ldots, \varrho_{\mathrm{mx}}), \\ &\ldots, \\ &\mathrm{trans}_k(s_r, \varrho_1, \ldots, \varrho_{\mathrm{mx}}), \\ &\varrho_{r+1}, \ldots, \varrho_{\mathrm{mx}})\end{aligned} \tag{4.4}$$

for every $r \in [0, \mathrm{mx}]$, $\mathrm{sub}_r \in {F_2}^{(r+1)}$ and $s, s_1, \ldots, s_r \in \mathrm{RHS}_{F_1,F_2,\Delta}(X_k, \emptyset)$.

In the following, we will denote the result of this construction as $M' = \mathfrak{C}_{4.5}(M)$. □

Since we will often deal with modular tree transducers suitable for Construction 4.5, we will establish the class of 2-modular tree transducers, of which the first module is a top-down tree transducer module and the second module is a substitution module, denoted $\mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$. Note that in contrast to common denotations used in the theory of formal languages this denotes a syntactic class. Following up, we state a well-known example, which already occurs in [KGK01] and is known to improve efficiency rather drastically.

Example 4.6 (Reversing a list efficiently). The modular tree transducer M_{rev} of Example 4.2 obeys all the restrictions necessary for Construction 4.5, thus the construction is applicable. The result is known as efficient reverse

$$M'_{\mathrm{rev}} = (\{\mathrm{rev}^{(2)}\}, \{A^{(1)}, B^{(1)}, N^{(0)}\}, (\mathrm{rev}\; x\; N), R') = \mathfrak{C}_{4.5}(M)$$

with rule-set R' containing:

$$\begin{array}{llcll} \mathrm{rev} & N & y_1 & = & y_1 \\ \mathrm{rev} & (A\, x_1) & y_1 & = & \mathrm{rev}\; x_1\; (A\, y_1) \\ \mathrm{rev} & (B\, x_1) & y_1 & = & \mathrm{rev}\; x_1\; (B\, y_1). \end{array}$$

The right hand side of the rev function symbol at A is computed as follows [35]:

$$\begin{aligned} & \mathrm{trans}_1(\mathrm{app}\; (\mathrm{rev}\; x_1)\; (A\, \Pi_1), y_1) \\ \overset{(4.4)}{=}\; & \mathrm{trans}_1((\mathrm{rev}\; x_1), \mathrm{trans}_1((A\, \Pi_1), y_1)) \\ \overset{(4.2)}{=}\; & \mathrm{rev}\; x_1\; (\mathrm{trans}_1((A\, \Pi_1), y_1)) \\ \overset{(4.3)}{=}\; & \mathrm{rev}\; x_1\; (A\; (\mathrm{trans}_1(\Pi_1, y_1))) \\ \overset{(4.1)}{=}\; & \mathrm{rev}\; x_1\; (A\, y_1). \end{aligned}$$

□

[35] Note that $\Pi_1 = N$.

4.2 Correctness proof

Construction 4.5 just provides a means of creating a macro tree transducer out of a given 2-modular tree transducer with certain restrictions. It does not state any properties of the resulting macro tree transducer, the foremost important of which would be the equivalence of the induced translations. Given this property both devices could interchangeably be used to compute the translation, so we prepare the proof of this property by extending the definition of the trans_M-mapping to sentential forms.

Definition 4.7 (trans_k on sentential forms). Let

$$M = (F, m, \Delta, e, R) \in \text{ModTT}(\text{TOP}, \text{SUB})$$

with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\text{mx}}\}$ for some $\text{mx} \in \mathbb{N}$ and $M' = \mathfrak{C}_{4.5}(M)$. Additionally, let $F_1 = m^{-1}(1)$ and $F_2 = m^{-1}(2)$. Since $\text{RHS}_{\text{MAC}}(F', \Delta, X_k, Y_{\text{mx}}) \subseteq \text{SF}_{X_k, Y_{\text{mx}}}(M')$ and $\text{RHS}_{F_1, F_2, \Delta}(X_k, \emptyset) \subseteq \text{SF}_{X_k, \emptyset}(M)$ for every $k \in \mathbb{N}$, we extend trans_k to sentential forms

$$\text{trans}_k : \text{SF}_{X_k, \emptyset}(M) \times \text{SF}_{X_k, Y_{\text{mx}}}(M')^{\text{mx}} \longrightarrow \text{SF}_{X_k, Y_{\text{mx}}}(M')$$

by stating essentially the same equations for every $\xi_1, \ldots, \xi_{\text{mx}} \in \text{SF}_{X_k, Y_{\text{mx}}}(M')$ with relaxed restrictions.

$$\text{trans}_k(\Pi_j, \xi_1, \ldots, \xi_{\text{mx}}) = \xi_j \tag{4.5}$$

for every $j \in [\text{mx}]$,

$$\text{trans}_k((f\ s), \xi_1, \ldots, \xi_{\text{mx}}) = (f\ s\ \xi_1\ \ldots\ \xi_{\text{mx}}) \tag{4.6}$$

for every $f \in F_1$, $s \in X_k \cup \mathcal{T}_\Delta$,

$$\begin{aligned} \text{trans}_M((\delta\ s_1 \ldots s_a), \xi_1, \ldots, \xi_{\text{mx}}) = (\delta\ & \text{trans}_k(s_1, \xi_1, \ldots, \xi_{\text{mx}}) \\ & \ldots \\ & \text{trans}_k(s_a, \xi_1, \ldots, \xi_{\text{mx}})) \end{aligned} \tag{4.7}$$

for every $a \in \mathbb{N}$, $\delta \in (\Delta^{(a)} \setminus \Pi)$ and $s_1, \ldots, s_a \in \text{SF}_{X_k, \emptyset}(M)$ and

$$\begin{aligned} \text{trans}_k((\text{sub}_r\ s\ s_1 \ldots s_r), \xi_1, \ldots, \xi_{\text{mx}}) = \text{trans}_k(s, & \text{trans}_k(s_1, \xi_1, \ldots, \xi_{\text{mx}}), \\ & \ldots, \\ & \text{trans}_k(s_r, \xi_1, \ldots, \xi_{\text{mx}}), \\ & \xi_{r+1}, \ldots, \xi_{\text{mx}}) \end{aligned} \tag{4.8}$$

for every $r \in [0, \text{mx}]$, $\text{sub}_r \in F_2{}^{(r+1)}$ and $s, s_1, \ldots, s_r \in \text{SF}_{X_k, \emptyset}(M)$. □

Example 4.8 (trans_k on sentential forms). We assume the modular tree transducer M_{rev} of Example 4.2 and the sentential form $\xi = \text{app (rev } (B\ (B\ N)))\ (A\ N)$ [36]. The

[36]This sentential form appears in the derivation of rev $(A\ (B\ (B\ N)))$ to its normal form $B\ (B\ (A\ N))$.

application of trans_0 to ξ yields [37]

$$\begin{aligned}
& \mathrm{trans}_0(\mathrm{app}\ (\mathrm{rev}\ (B\ (B\ \Pi_1)))\ (A\ \Pi_1), \Pi_1) \\
\overset{(4.8)}{=}\ & \mathrm{trans}_0(\mathrm{rev}\ (B\ (B\ \Pi_1)), \mathrm{trans}_0((A\ \Pi_1), \Pi_1)) \\
\overset{(4.6)}{=}\ & \mathrm{rev}\ (B\ (B\ \Pi_1))\ (\mathrm{trans}_0((A\ \Pi_1), \Pi_1)) \\
\overset{(4.7)}{=}\ & \mathrm{rev}\ (B\ (B\ \Pi_1))\ (A\ (\mathrm{trans}_0(\Pi_1, \Pi_1))) \\
\overset{(4.5)}{=}\ & \mathrm{rev}\ (B\ (B\ \Pi_1))\ (A\ \Pi_1) \\
=\ & \mathrm{rev}\ (B\ (B\ N))\ (A\ N).
\end{aligned}$$

Note that this is indeed a sentential form of M'_{rev} of Example 4.6, but the specific relation will be made precise later. □

Before proceeding to the actual proof, we establish some key properties of the trans_k-mapping for every $k \in \mathbb{N}$. They will be used later in the main proof of this section.

Lemma 4.9 (Properties of trans_k). *Given $(F, m, \Delta, e, R) = M \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$ with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ for some $\mathrm{mx} \in \mathbb{N}$ and $M' = \mathfrak{C}_{4.5}(M)$, the following properties hold:*

(a) Let $\xi \in \mathrm{SF}_{X_k,\emptyset}(M)$ for some $k \in \mathbb{N}$, then for arbitrary substitutions $\theta_1 : X_k \longrightarrow \mathcal{T}_\Delta$, $\theta_2 : Y_{\mathrm{mx}} \longrightarrow \mathrm{SF}(M')$ and $\xi_1, \ldots, \xi_{\mathrm{mx}} \in \mathrm{SF}_{X_k,Y_{\mathrm{mx}}}(M')$

$$\mathrm{trans}_0(\xi\theta_1, \xi_1\theta_1\theta_2, \ldots, \xi_{\mathrm{mx}}\theta_1\theta_2) = \mathrm{trans}_k(\xi, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta_1\theta_2.$$

(b) For every $t \in \mathcal{T}_\Delta$: $\mathrm{trans}_0(t, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = t$.

(c) Let $\xi, \xi', \bar{\xi} \in \mathrm{SF}(M')$ and $p \in \mathrm{occ}(\xi')$ with $\xi = \xi'[\bar{\xi}]_p$. If

$$\mathrm{trans}_0(\bar{\xi}, \xi_1, \ldots, \xi_{\mathrm{mx}}) = \mathrm{trans}_0(\xi'|_p, \xi_1, \ldots, \xi_{\mathrm{mx}})$$

for every $\xi_1, \ldots, \xi_{mx} \in \mathrm{SF}(M')$, then also

$$\mathrm{trans}_0(\xi, \Pi_1, \ldots, \Pi_{mx}) = \mathrm{trans}_0(\xi', \Pi_1, \ldots, \Pi_{mx}).$$

Proof. The enumeration in the proof refers to the enumeration of the statements above.

(a) We prove the statement via structural induction over $\xi \in \mathrm{SF}_{X_k,\emptyset}(M)$. Therefore, we let $\theta = \theta_1\theta_2$.

- **Induction base:**
  - Let $\xi \in \Delta^{(0)}$, then

$$\begin{aligned}
& \mathrm{trans}_0(\xi\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) = \mathrm{trans}_0(\xi, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) \\
\overset{(4.5)\ \&\ (4.7)}{=}\ & \begin{cases} \xi_j\theta & \text{, if } \xi = \Pi_j \text{ for some } j \in [\mathrm{mx}] \\ \xi\theta & \text{, otherwise} \end{cases} \\
\overset{(4.5)\ \&\ (4.7)}{=}\ & \mathrm{trans}_k(\xi, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta.
\end{aligned}$$

[37] Note that again $\Pi_1 = N$.

– Let $\xi = (f\ s)$ with $f \in m^{-1}(1)$ and $s \in X_k \cup \mathcal{T}_\Delta$ [38], then

$$\begin{aligned} & \mathrm{trans}_0((f\ s)\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) \ = \ \mathrm{trans}_0((f\ s\theta_1), \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) \\ \overset{(4.6)}{=} \ & (f\ (s\theta_1)\ (\xi_1\theta)\ \ldots\ (\xi_{\mathrm{mx}}\theta)) \ = \ (f\ s\ \xi_1\ \ldots\ \xi_{\mathrm{mx}})\theta \\ \overset{(4.6)}{=} \ & \mathrm{trans}_k((f\ s), \xi_1, \ldots, \xi_{\mathrm{mx}})\theta. \end{aligned}$$

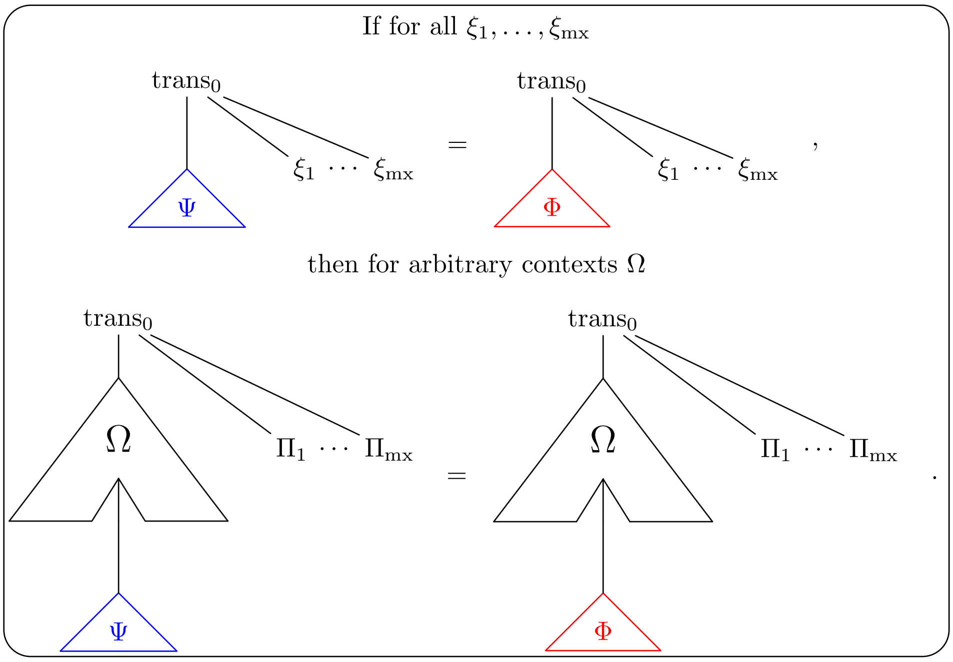

Figure 10: Local equivalence extends naturally to global equivalence.

- **Induction step:**
  – Let $\xi = (\sigma\ s_1\ \ldots\ s_a)$ with $a \in \mathbb{N}$, $\sigma \in \Delta^{(a)}$, $s_1, \ldots, s_a \in \mathrm{SF}_{X_k,\emptyset}(M)$, then for every $i \in [a]$ by induction hypothesis

$$\mathrm{trans}_0(s_i\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) = \mathrm{trans}_k(s_i, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta$$

and we conclude

$$\begin{aligned} & \mathrm{trans}_0((\sigma\ s_1\ \ldots\ s_a)\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) \\ = \ & \mathrm{trans}_0((\sigma\ (s_1\theta_1)\ \ldots\ (s_a\theta_1)), \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) \\ \overset{(4.7)}{=} \ & (\sigma\ \mathrm{trans}_0(s_1\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta)\ \ldots\ \mathrm{trans}_0(s_a\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta)) \\ \overset{\mathrm{I.H.}}{=} \ & (\sigma\ (\mathrm{trans}_k(s_1, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta)\ \ldots\ (\mathrm{trans}_k(s_a, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta)) \\ = \ & (\sigma\ \mathrm{trans}_k(s_1, \xi_1, \ldots, \xi_{\mathrm{mx}})\ \ldots\ \mathrm{trans}_k(s_a, \xi_1, \ldots, \xi_{\mathrm{mx}}))\theta \\ \overset{(4.7)}{=} \ & \mathrm{trans}_k((\sigma\ s_1\ \ldots\ s_a), \xi_1, \ldots, \xi_{\mathrm{mx}})\theta. \end{aligned}$$

[38] Recapture that sentential forms are right hand sides with input trees possibly substituted in for recursion arguments. Hence we also have to consider the case $s \in \mathcal{T}_\Delta$.

– Let $\xi = (\mathrm{sub}_r s s_1 \ldots s_r)$ with $r \in [0, \mathrm{mx}]$, $\mathrm{sub}_r \in m^{-1}(2)^{(r+1)}$, $s, s_1, \ldots, s_r \in \mathrm{SF}_{X_k,\emptyset}(M)$, then for every $i \in [r]$ by induction hypothesis

$$\mathrm{trans}_0(s_i\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) = \mathrm{trans}_k(s_i, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta$$

and since $\mathrm{trans}_k(s_i, \xi_1, \ldots, \xi_{\mathrm{mx}}) \in \mathrm{SF}_{X_k,Y_{\mathrm{mx}}}(M')$ for every $i \in [r]$ also

$$\begin{aligned}
&\mathrm{trans}_0(s\theta_1, \mathrm{trans}_k(s_1, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta, \ldots, \\
&\qquad \mathrm{trans}_k(s_r, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta, \xi_{r+1}\theta, \ldots, \xi_{\mathrm{mx}}\theta) \\
&= \mathrm{trans}_k(s, \mathrm{trans}_k(s_1, \xi_1, \ldots, \xi_{\mathrm{mx}}), \ldots, \\
&\qquad \mathrm{trans}_k(s_r, \xi_1, \ldots, \xi_{\mathrm{mx}}), \xi_{r+1}, \ldots, \xi_{\mathrm{mx}})\theta.
\end{aligned}$$

We conclude

$$\begin{aligned}
& \mathrm{trans}_0((\mathrm{sub}_r\ s\ s_1\ \ldots\ s_r)\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) \\
= \;& \mathrm{trans}_0((\mathrm{sub}_r\ (s\theta_1)\ (s_1\theta_1)\ \ldots\ (s_r\theta_1)), \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta) \\
\overset{(4.8)}{=} \;& \mathrm{trans}_0(s\theta_1, \mathrm{trans}_0(s_1\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta), \ldots, \\
& \qquad \mathrm{trans}_0(s_r\theta_1, \xi_1\theta, \ldots, \xi_{\mathrm{mx}}\theta), \xi_{r+1}\theta, \ldots, \xi_{\mathrm{mx}}\theta) \\
\overset{\text{I.H.}}{=} \;& \mathrm{trans}_0(s\theta_1, \mathrm{trans}_k(s_1, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta, \ldots, \\
& \qquad \mathrm{trans}_k(s_r, \xi_1, \ldots, \xi_{\mathrm{mx}})\theta, \xi_{r+1}\theta, \ldots, \xi_{\mathrm{mx}}\theta) \\
\overset{\text{I.H.}}{=} \;& \mathrm{trans}_k(s, \mathrm{trans}_k(s_1, \xi_1, \ldots, \xi_{\mathrm{mx}}), \ldots, \\
& \qquad \mathrm{trans}_k(s_r, \xi_1, \ldots, \xi_{\mathrm{mx}}), \xi_{r+1}, \ldots, \xi_{\mathrm{mx}})\theta \\
\overset{(4.8)}{=} \;& \mathrm{trans}_k((\mathrm{sub}_r\ s\ s_1\ \ldots\ s_r), \xi_1, \ldots, \xi_{\mathrm{mx}})\theta.
\end{aligned}$$

Hence the proof is finished [39].

(b) We prove the statement via structural induction over $t \in \mathcal{T}_\Delta$.

- **Induction base:** Let $t \in \Delta^{(0)}$, thus immediately by (4.5) and (4.7)

$$\mathrm{trans}_0(t, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = \begin{cases} \Pi_j & \text{, if } t = \Pi_j \text{ for some } j \in [\mathrm{mx}] \\ t & \text{, otherwise} \end{cases} = t.$$

- **Induction step:** Let $t = (\delta t_1 \ldots t_a)$ with $a \in \mathbb{N}_+$, $\delta \in \Delta^{(a)}$ and $t_1, \ldots, t_a \in \mathcal{T}_\Delta$. By induction hypothesis $\mathrm{trans}_0(t_i, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = t_i$ for every $i \in [a]$ and, consequently,

$$\begin{aligned}
& \mathrm{trans}_0((\delta\, t_1\ \ldots\ t_a), \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\
\overset{(4.7)}{=} \;& (\delta\ \mathrm{trans}_0(t_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}})\ \ldots\ \mathrm{trans}_0(t_a, \Pi_1, \ldots, \Pi_{\mathrm{mx}})) \\
\overset{\text{I.H.}}{=} \;& (\delta\, t_1\ \ldots\ t_a)\ =\ t.
\end{aligned}$$

This completes the proof.

[39] A similar proposition in a somewhat different setting can be found in [Voi01].

(c) We will prove the following more general statement: Let $\xi, \xi', \bar{\xi} \in \mathrm{SF}(M')$ and $p \in \mathrm{occ}(\xi')$ with $\xi = \xi'[\bar{\xi}]_p$. If

$$\mathrm{trans}_0(\bar{\xi}, \xi_1, \ldots, \xi_{\mathrm{mx}}) = \mathrm{trans}_0(\xi'|_p, \xi_1, \ldots, \xi_{\mathrm{mx}})$$

for every $\xi_1, \ldots, \xi_{\mathrm{mx}} \in \mathrm{SF}(M')$, then also

$$\mathrm{trans}_0(\xi, \bar{\xi}_1, \ldots, \bar{\xi}_{\mathrm{mx}}) = \mathrm{trans}_0(\xi', \bar{\xi}_1, \ldots, \bar{\xi}_{\mathrm{mx}})$$

for arbitrary $\bar{\xi}_1, \ldots, \bar{\xi}_{\mathrm{mx}} \in \mathrm{SF}(M')$. The proof is an induction on the length of p.

- **Induction base:** Let $|p| = 0$ and, thereby $p = \varepsilon$. We set $\xi_i = \bar{\xi}_i$ for every $i \in [\mathrm{mx}]$ and immediately have the desired result by assumption.
- **Induction step:** Let $|p| = n + 1$ for some $n \in \mathbb{N}$, thus $p = \bar{p}.j$ with $j \in \mathbb{N}_+$, $\bar{p} \in \mathrm{occ}(\xi)$, $|\bar{p}| = n$ and, since $p \in \mathrm{occ}(\xi')$, obviously $\xi'|_{\bar{p}} = (\sigma\ s_1\ \ldots\ s_a)$ with $a \in \mathbb{N}_+$, $\sigma \in (\Delta \cup F)^{(a)}$ and $s_1, \ldots, s_a \in \mathrm{SF}(M)$. Thereby, $j \in [a]$, $\xi|_p = \bar{\xi}$ and $\xi'|_p = s_j$, hence by definition

$$\xi = \xi'[(\sigma\ s_1\ \ldots\ s_{j-1}\ \bar{\xi}\ s_{j+1}\ \ldots\ s_a)]_{\bar{p}}.$$

The induction hypothesis is: If

$$\mathrm{trans}_0(\xi|_{\bar{p}}, \chi_1, \ldots, \chi_{\mathrm{mx}}) = \mathrm{trans}_0(\xi'|_{\bar{p}}, \chi_1, \ldots, \chi_{\mathrm{mx}})$$

for every $\chi_1, \ldots, \chi_{\mathrm{mx}} \in \mathrm{SF}(M')$, then also

$$\mathrm{trans}_0(\xi, \bar{\chi}_1, \ldots, \bar{\chi}_{\mathrm{mx}}) = \mathrm{trans}_0(\xi', \bar{\chi}_1, \ldots, \bar{\chi}_{\mathrm{mx}})$$

for arbitrary $\bar{\chi}_1, \ldots, \bar{\chi}_{\mathrm{mx}} \in \mathrm{SF}(M')$.

  – Let $\sigma = \delta \in \Delta$:

$$\begin{aligned}
& \mathrm{trans}_0(\xi|_{\bar{p}}, \chi_1, \ldots, \chi_{\mathrm{mx}}) \\
= \;& \mathrm{trans}_0((\delta\ s_1\ \ldots\ s_{j-1}\ \bar{\xi}\ s_{j+1}\ \ldots\ s_a), \chi_1, \ldots, \chi_{\mathrm{mx}}) \\
\overset{(4.7)}{=} \;& (\delta\ \mathrm{trans}_0(s_1, \chi_1, \ldots, \chi_{\mathrm{mx}})\ \ldots\ \mathrm{trans}_0(\bar{\xi}, \chi_1, \ldots, \chi_{\mathrm{mx}})\ \ldots \\
& \quad \mathrm{trans}_0(s_a, \chi_1, \ldots, \chi_{\mathrm{mx}})) \\
\overset{\text{Ass.}}{=} \;& (\delta\ \mathrm{trans}_0(s_1, \chi_1, \ldots, \chi_{\mathrm{mx}})\ \ldots\ \mathrm{trans}_0(s_a, \chi_1, \ldots, \chi_{\mathrm{mx}})) \\
\overset{(4.7)}{=} \;& \mathrm{trans}_0((\delta\ s_1\ \ldots\ s_a), \chi_1, \ldots, \chi_{\mathrm{mx}}) \\
= \;& \mathrm{trans}_0(\xi'|_{\bar{p}}, \chi_1, \ldots, \chi_{\mathrm{mx}}).
\end{aligned}$$

  – Let $\sigma = f \in m^{-1}(1)$, thus $a = j = 1$ and $s_1 \in \mathcal{T}_\Delta$. The assumption requires

$$\mathrm{trans}_0(\bar{\xi}, \xi_1, \ldots, \xi_{\mathrm{mx}}) = \mathrm{trans}_0(s_1, \xi_1, \ldots, \xi_{\mathrm{mx}})$$

  for every $\xi_1, \ldots, \xi_{\mathrm{mx}} \in \mathrm{SF}(M')$, hence we instantiate $\xi_i = \Pi_i$ for every $i \in [\mathrm{mx}]$ and by Lemma 4.9(b) gain that $\bar{\xi} = s_1$. This immediately proves

$$\mathrm{trans}_0(\xi|_{\bar{p}}, \chi_1, \ldots, \chi_{\mathrm{mx}}) = \mathrm{trans}_0(\xi'|_{\bar{p}}, \chi_1, \ldots, \chi_{\mathrm{mx}}).$$

– Let $\sigma = \mathrm{sub}_{a-1} \in m^{-1}(2)$ with $a \in \mathbb{N}_+$. Since $\mathrm{trans}_0(s_i, \chi_1, \ldots, \chi_{\mathrm{mx}}) \in \mathrm{SF}(M')$ for every $i \in [a]$ we conclude

$$
\begin{aligned}
& \mathrm{trans}_0(\xi|_{\bar{p}}, \chi_1, \ldots, \chi_{\mathrm{mx}}) \\
= \; & \mathrm{trans}_0((\mathrm{sub}_{a-1}\; s_1 \; \ldots \; s_{j-1}\; \bar{\xi}\; s_{j+1} \; \ldots \; s_a), \chi_1, \ldots, \chi_{\mathrm{mx}}) \\
\overset{(4.8)}{=} \; & \begin{cases} \mathrm{trans}_0(\bar{\xi},\; \mathrm{trans}_k(s_2, \chi_1, \ldots, \chi_{\mathrm{mx}}), \ldots, \\ \qquad\qquad \mathrm{trans}_0(s_a, \chi_1, \ldots, \chi_{\mathrm{mx}}), \chi_{a+1}, \ldots, \chi_{\mathrm{mx}}) & \text{, if } j = 1 \\ \mathrm{trans}_0(s_1,\; \mathrm{trans}_0(s_2, \chi_1, \ldots, \chi_{\mathrm{mx}}), \ldots, \\ \qquad\qquad \mathrm{trans}_0(\bar{\xi}, \chi_1, \ldots, \chi_{\mathrm{mx}}), \ldots, & \text{, otherwise} \\ \qquad\qquad \mathrm{trans}_0(s_a, \chi_1, \ldots, \chi_{\mathrm{mx}}), \chi_{a+1}, \ldots, \chi_{\mathrm{mx}}) \end{cases} \\
\overset{\text{Ass.}}{=} \; & \mathrm{trans}_0(s_1,\; \mathrm{trans}_0(s_2, \chi_1, \ldots, \chi_{\mathrm{mx}}), \ldots, \\
& \qquad\qquad \mathrm{trans}_0(s_a, \chi_1, \ldots, \chi_{\mathrm{mx}}), \chi_{a+1}, \ldots, \chi_{\mathrm{mx}}) \\
\overset{(4.8)}{=} \; & \mathrm{trans}_0((\mathrm{sub}_{a-1}\; s_1 \; \ldots \; s_a), \chi_1, \ldots, \chi_{\mathrm{mx}}) \\
= \; & \mathrm{trans}_0(\xi'|_{\bar{p}}, \chi_1, \ldots, \chi_{\mathrm{mx}}).
\end{aligned}
$$

Thus, in all cases we could show the prerequisite of the induction hypothesis, such that the application of the induction hypothesis proves the statement [40].

Altogether, this successfully proves the statement.

□

The first property states that we can delay substitutions and apply them to the result of the trans_k-mapping (in a restricted way), instead of applying them to its parameters (and vice versa). This fact bridges the gap between instantiated right hand sides [41] and sentential forms, namely instead of applying the matching substitution to the translated right hand side, we can apply the substitution to the original right hand side of M and then translate (via trans_k) this instantiated right hand side.

Example 4.10 (Property of Lemma 4.9(a)). Assume M_{rev} of Example 4.2 and the right hand side $\mathrm{app}\;(\mathrm{rev}\; x_1)\;(A\,\Pi_1)$ with $\Pi_1 = N$. Furthermore, let $\theta_1 = \{\, x_1 \mapsto (A\,\Pi_1)\,\}$

[40] The conclusions of both implications (the proof obligation and the induction hypothesis) coincide.

[41] In the derivation relation we require a substitution θ which matches the left hand side of an equation to a redex in a sentential form. The instantiated right hand side is the corresponding right hand side with the substitution θ applied.

and $\theta_2 = \{\, y_1 \mapsto (\mathrm{rev}\,\Pi_1\,(B\,\Pi_1))\,\}$. Then

$$
\begin{aligned}
& \mathrm{trans}_1(\mathrm{app}\,(\mathrm{rev}\,x_1)\,(A\,\Pi_1), y_1)\,\theta_1\theta_2 \\
\overset{(4.8)}{=}\; & \mathrm{trans}_1((\mathrm{rev}\,x_1), \mathrm{trans}_1((A\,\Pi_1), y_1))\,\theta_1\theta_2 \\
\overset{(4.6)}{=}\; & (\mathrm{rev}\,x_1\,\mathrm{trans}_1((A\,\Pi_1), y_1))\,\theta_1\theta_2 \\
\overset{(4.7)}{=}\; & (\mathrm{rev}\,x_1\,(A\,\mathrm{trans}_1(\Pi_1, y_1)))\,\theta_1\theta_2 \\
\overset{(4.5)}{=}\; & (\mathrm{rev}\,x_1\,(A\,y_1))\,\theta_1\theta_2 \\
=\; & (\mathrm{rev}\,(A\,\Pi_1)\,(A\,y_1))\,\theta_2 \\
=\; & \mathrm{rev}\,(A\,\Pi_1)\,(A\,(\mathrm{rev}\,\Pi_1\,(B\,\Pi_1))) \\
\overset{(4.5)}{=}\; & \mathrm{rev}\,(A\,\Pi_1)\,(A\,\mathrm{trans}_0(\Pi_1, (\mathrm{rev}\,\Pi_1\,(B\,\Pi_1)))) \\
\overset{(4.7)}{=}\; & \mathrm{rev}\,(A\,\Pi_1)\,\mathrm{trans}_0((A\,\Pi_1), (\mathrm{rev}\,\Pi_1\,(B\,\Pi_1))) \\
\overset{(4.6)}{=}\; & \mathrm{trans}_0((\mathrm{rev}\,(A\,\Pi_1)), \mathrm{trans}_0((A\,\Pi_1), (\mathrm{rev}\,\Pi_1\,(B\,\Pi_1)))) \\
\overset{(4.8)}{=}\; & \mathrm{trans}_0((\mathrm{app}\,(\mathrm{rev}\,(A\,\Pi_1))\,(A\,\Pi_1)), (\mathrm{rev}\,\Pi_1\,(B\,\Pi_1))) \\
=\; & \mathrm{trans}_0((\mathrm{app}\,(\mathrm{rev}\,x_1)\,(A\,\Pi_1))\,\theta_1, y_1\theta_1\theta_2).
\end{aligned}
$$

□

Secondly, the lemma presents that for all terms solely made of constructors the trans_0-translation with $\Pi_1, \ldots, \Pi_{\mathrm{mx}}$ as the context computes the identity. This will be required to show the correctness of the construction, where we will use closed sets, so we establish the corresponding closure operator.

Lastly, in the third part of the lemma we provided a rather obvious property: Whenever two sentential forms ξ and ξ' differ only at a certain subtree located at path p and, additionally, for every context $\xi_1, \ldots, \xi_{\mathrm{mx}}$ the translation of the subtrees at p is equivalent, then so is the translation of the trees ξ and ξ'.

Definition 4.11 (Derivation closure). Assume a modular tree transducer M and a set of sentential forms $S \subseteq \mathrm{SF}(M)$. Then the (non-deterministic) *derivation closure* of S (derivation driven by $\Rightarrow_M$) is denoted $\mathrm{cl}_{\Rightarrow_M}(S)$ and defined by

$$\mathrm{cl}_{\Rightarrow_M}(S) = \{\, \xi \in \mathrm{SF}(M) \mid s \in S,\, s \Rightarrow_M^* \xi \,\}.$$

Equivalently, we define $\mathrm{cl}_{\mapsto_M}(\mathfrak{G}) = \{\, \Xi \in \mathrm{SFG}(M) \mid G \in \mathfrak{G},\, G \mapsto_M^* \Xi \,\}$ for every $\mathfrak{G} \subseteq \mathrm{SFG}(M)$. □

Example 4.12 (Derivation closure). In Figure 11 we will display all elements of

$$\mathrm{cl}_{\Rightarrow_{M_{\mathrm{rev}}}}(\{\mathrm{rev}\,(A\,(B\,(B\,N)))\}),$$

where M_{rev} is the modular tree transducer of Example 4.2. □

Lemma 4.13 (Derivation closure is a closure operator). *For a fixed modular tree transducer M the derivation closure $\mathrm{cl}_{\Rightarrow_M} : \mathfrak{P}(\mathrm{SF}(M)) \to \mathfrak{P}(\mathrm{SF}(M))$ indeed defines a closure operator over $\mathfrak{P}(\mathrm{SF}(M))$.*

$$\begin{array}{c}
\mathrm{rev}\,(A\,(B\,(B\,N))) \\
\Downarrow \\
\mathrm{app}\,(\mathrm{rev}\,(B\,(B\,N)))\,(A\,N) \\
\Downarrow \\
\mathrm{app}\,(\mathrm{app}\,(\mathrm{rev}\,(B\,N))\,(B\,N))\,(A\,N) \\
\Downarrow \\
\mathrm{app}\,(\mathrm{app}\,(\mathrm{app}\,(\mathrm{rev}\,N)\,(B\,N))\,(B\,N))\,(A\,N) \\
\Downarrow \\
\mathrm{app}\,(\mathrm{app}\,(\mathrm{app}\,N\,(B\,N))\,(B\,N))\,(A\,N) \\
\Downarrow \\
\mathrm{app}\,(\mathrm{app}\,(B\,N)\,(B\,N))\,(A\,N) \\
\Downarrow \\
\mathrm{app}\,(B\,(\mathrm{app}\,N\,(B\,N)))\,(A\,N) \\
\swarrow \qquad\qquad \searrow \\
\mathrm{app}\,(B\,(B\,N))\,(A\,N) \quad B\,(\mathrm{app}\,(\mathrm{app}\,N\,(B\,N))\,(A\,N)) \\
\searrow \qquad\qquad \swarrow \\
B\,(\mathrm{app}\,(B\,N)\,(A\,N)) \\
\Downarrow \\
B\,(B\,(\mathrm{app}\,N\,(A\,N))) \\
\Downarrow \\
B\,(B\,(A\,N))
\end{array}$$

Figure 11: All elements of the derivation closure of rev $(A\,(B\,(B\,N)))$, where $\Rightarrow = \Rightarrow_{M_{\mathrm{rev}}}$.

Proof. Obviously, $\mathrm{cl}_{\Rightarrow_M}$ is an operator on $\mathfrak{P}(\mathrm{SF}(M))$, so in order to show that $\mathrm{cl}_{\Rightarrow_M}$ is a closure operator, we need to show extensionality, monotonicity and idempotency.

- **Extensionality:** We need to show that for every $S \subseteq \mathrm{SF}(M)$ the following holds:

  $$S \subseteq \mathrm{cl}_{\Rightarrow_M}(S) \overset{\text{Def. 4.11}}{=} \{\, \xi \in \mathrm{SF}(M) \mid s \in S,\, s \Rightarrow_M^* \xi \,\}.$$

  This is trivially true, since $\Rightarrow_M^0 = id_{\mathrm{SF}(M)} \subseteq \Rightarrow_M^*$.

- **Monotonicity:** We assume $S_1 \subseteq S_2 \subseteq \mathrm{SF}(M)$. Monotonicity requires to show, that $\mathrm{cl}_{\Rightarrow_M}(S_1) \subseteq \mathrm{cl}_{\Rightarrow_M}(S_2)$. Consequently, we select $\xi \in \mathrm{cl}_{\Rightarrow_M}(S_1)$ and by Definition 4.11 there exists $s \in S_1$ with $s \Rightarrow_M^* \xi$. Since $S_1 \subseteq S_2$ we conclude $s \in S_2$ and, thus, $\xi \in \mathrm{cl}_{\Rightarrow_M}(T)$ by Definition 4.11.

- **Idempotency:** For every $S \subseteq \mathrm{SF}(M)$ we prove the following statement:

  $$\mathrm{cl}_{\Rightarrow_M}(S) = \mathrm{cl}_{\Rightarrow_M}(\mathrm{cl}_{\Rightarrow_M}(S)).$$

  Actually it is sufficient to show $\mathrm{cl}_{\Rightarrow_M}(\mathrm{cl}_{\Rightarrow_M}(S)) \subseteq \mathrm{cl}_{\Rightarrow_M}(S)$, since $\mathrm{cl}_{\Rightarrow_M}$ is extensive and monotone [42].

  We arbitrarily select $\xi' \in \mathrm{cl}_{\Rightarrow_M}(\mathrm{cl}_{\Rightarrow_M}(S))$, hence by Definition 4.11 there exists $\xi \in \mathrm{cl}_{\Rightarrow_M}(S)$ with $\xi \Rightarrow_M^* \xi'$. Furthermore, again by Definition 4.11 there exists $s \in S$ such that $s \Rightarrow_M^* \xi$. Consequently, $s \Rightarrow_M^* \xi'$, since $\Rightarrow_M^*$ is transitive, and, thus, $\xi' \in \mathrm{cl}_{\Rightarrow_M}(S)$.

[42] By extensionality $S \subseteq \mathrm{cl}_{\Rightarrow_M}(S)$ and, thus, by monotonicity $\mathrm{cl}_{\Rightarrow_M}(S) \subseteq \mathrm{cl}_{\Rightarrow_M}(\mathrm{cl}_{\Rightarrow_M}(S))$.

Further, we add that the proof also applies to other derivation relations, in particular the call-by-need derivation relation, since the necessary properties of reflexivity and transitivity are fulfilled trivially by the reflexive and transitive closure of each derivation relation. Thus, $\mathrm{cl}_{\Rrightarrow_M}$ is a closure operator over sentential form graphs. $\square$

Lemma 4.14 (Correspondence on sentential forms). *Let* $M \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$ *with substitution variables* $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ *for some* $\mathrm{mx} \in \mathbb{N}$, $M = (F, m, \Delta, e, R)$ *and let* $M' = (F', \Delta, e', R') = \mathfrak{C}_{4.5}(M)$. *Then for every input tree* $t \in \mathcal{T}_\Delta$ *and substitution* $\theta = \{\, x \mapsto t \,\}$

$$\mathrm{trans}_0(\cdot, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) : \mathrm{cl}_{\Rightarrow_M}(\{e\theta\}) \longrightarrow \mathrm{cl}_{\Rightarrow_{M'}}(\{e'\theta\}).$$

Proof. Note that $\mathcal{C} = \{\, S \subseteq \mathrm{SF}(M) \mid S = \mathrm{cl}_{\Rightarrow_M}(S) \,\}$ is the closure system, which corresponds to $\mathrm{cl}_{\Rightarrow_M}$. Additionally, $\mathrm{cl}_{\Rightarrow_M}(\{e\theta\}) \in \mathcal{C}$, so in order to prove

$$\mathrm{trans}_0(\xi, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \in \mathrm{cl}_{\Rightarrow_{M'}}(\{e'\theta\}) \tag{*}$$

for every $\xi \in \mathrm{cl}_{\Rightarrow_M}(\{e\theta\})$ it suffices to show

1. Statement (*) holds for each $\xi \in \{e\theta\}$, i.e. Statement (*) holds for $e\theta$ and
2. the set S of all elements of $\mathrm{cl}_{\Rightarrow_M}(\{e\theta\})$, for which Statement (*) holds, is closed,

since $\mathrm{cl}_{\Rightarrow_M}(\{e\theta\})$ is obviously generated by $\{e\theta\}$ [43].

(1.) $\xi = e\theta = (f\ x)\theta = f\ t$ for some $f \in m^{-1}(1)$ and $t \in \mathcal{T}_\Delta$ by Definition 3.3 and, thus,

$$\mathrm{trans}_0((f\ t, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \overset{(4.6)}{=} (f\ t\ \Pi_1\ \ldots\ \Pi_{\mathrm{mx}}) \overset{\text{Cons. 4.5}}{=} e'\theta,$$

which is trivially an element of $\mathrm{cl}_{\Rightarrow_{M'}}(\{e'\theta\})$ (since $\mathrm{cl}_{\Rightarrow_{M'}}$ is extensive).

(2.) Note that $\mathrm{cl}_{\Rightarrow_M}(\{e\theta\})$ is finite, because $\Rightarrow_M$ is terminating. To show that S is closed, we need to prove $S \in \mathcal{C}$ or equivalently $\mathrm{cl}_{\Rightarrow_M}(S) = S$. Obviously $S \subseteq \mathrm{cl}_{\Rightarrow_M}(S)$, so we arbitrarily select $\xi' \in \mathrm{cl}_{\Rightarrow_M}(S)$ and hence by Definition 4.11 there exists $\xi \in S$ with $\xi \Rightarrow_M^* \xi'$. We prove that given ξ, which fulfills Statement (*), $\xi' \in S$ by natural induction over the length $n \in \mathbb{N}$ of the derivation $\xi \Rightarrow_M^n \xi'$.

- **Induction basis:** Let $n = 0$, then $\xi \Rightarrow_M^0 \xi'$ yields $\xi = \xi' \in S$.
- **Induction step:** Let $n = n' + 1$ and $\xi \Rightarrow_M^{n'} \xi'' \Rightarrow_M \xi'$ with $n' \in \mathbb{N}$, where by induction hypothesis $\xi'' \in S$. By Definition 3.8 there exists $p \in \mathrm{occ}(\xi'')$ with $\xi''|_p = f\ (\delta\ s_1\ \ldots\ s_k)\ t_1\ \ldots\ t_r$, where $k, r \in \mathbb{N}$, $f \in F^{(r+1)}$, $\delta \in \Delta^{(k)}$ and $s_1, \ldots, s_k, t_1, \ldots, t_r \in \mathrm{SF}(M)$, and also $\xi' = \xi''[\mathrm{rhs}_M(f, \delta)\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k,\ y_1 \mapsto t_1, \ldots, y_r \mapsto t_r \,\}]_p$ with $(f\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r = \mathrm{rhs}_M(f, \delta)) \in R$. We perform the following case distinction:

  (a) Let $f = \mathrm{sub}_r \in m^{-1}(2)$, thus $r \in [0, \mathrm{mx}]$. By Lemma 4.9(c), if we could show
  $$\mathrm{trans}_0(\xi''|_p, \xi_1, \ldots, \xi_{\mathrm{mx}}) = \mathrm{trans}_0(\xi'|_p, \xi_1, \ldots, \xi_{\mathrm{mx}})$$

[43] By definition $\mathrm{cl}_{\Rightarrow_M}(\{e\theta\})$ is the least closed set containing $\{e\theta\}$, thus accordingly, $\mathrm{cl}_{\Rightarrow_M}(\{e\theta\}) \subseteq S$ and trivially $S \subseteq \mathrm{cl}_{\Rightarrow_M}(\{e\theta\})$.

for every $\xi_1, \ldots, \xi_{\mathrm{mx}} \in \mathrm{SF}(M)$, then

$$\mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = \mathrm{trans}_0(\xi', \Pi_1, \ldots, \Pi_{\mathrm{mx}}),$$

hence by induction hypothesis $\mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \in \mathrm{cl}_{\Rightarrow_{M'}}(\{e'\theta\}) : \xi' \in S$. Accordingly, we proceed by

$$
\begin{aligned}
& \mathrm{trans}_0(\xi''|_p, \xi_1, \ldots, \xi_{\mathrm{mx}}) \\
= \; & \mathrm{trans}_0((\mathrm{sub}_r\ (\delta\ s_1\ \ldots\ s_k)\ t_1\ \ldots\ t_r), \xi_1, \ldots, \xi_{\mathrm{mx}}) \\
\overset{(4.8)}{=} \; & \mathrm{trans}_0((\delta\ s_1\ \ldots\ s_k),\ \mathrm{trans}_0(t_1, \xi_1, \ldots, \xi_{\mathrm{mx}}), \\
& \qquad \ldots, \\
& \qquad \mathrm{trans}_0(t_r, \xi_1, \ldots, \xi_{\mathrm{mx}}), \xi_{r+1}, \ldots, \xi_{\mathrm{mx}}) \\
\overset{(4.5)\ \&\ (4.7)}{=} \; & \begin{cases}
\mathrm{trans}_0(t_j, \xi_1, \ldots, \xi_{\mathrm{mx}}) & \text{, if } \delta = \Pi_j \text{ for some } j \in [r] \\
\xi_j & \text{, if } \delta = \Pi_j \text{ for some } j \notin [r] \\
(\delta\ \mathrm{trans}_0(s_1, & \\
\quad \mathrm{trans}_0(t_1, \xi_1, \ldots, \xi_{\mathrm{mx}}), & \\
\quad \ldots, & \\
\quad \mathrm{trans}_0(t_r, \xi_1, \ldots, \xi_{\mathrm{mx}}), & \\
\quad \xi_{r+1}, \ldots, \xi_{\mathrm{mx}}) & \\
\quad \ldots & \text{, otherwise} \\
\quad \mathrm{trans}_0(s_k, & \\
\quad \mathrm{trans}_0(t_1, \xi_1, \ldots, \xi_{\mathrm{mx}}), & \\
\quad \ldots, & \\
\quad \mathrm{trans}_0(t_r, \xi_1, \ldots, \xi_{\mathrm{mx}}), & \\
\quad \xi_{r+1}, \ldots, \xi_{\mathrm{mx}})) &
\end{cases} \\
\overset{(4.5)\ \&\ (4.8)}{=} \; & \begin{cases}
\mathrm{trans}_0(t_j, \xi_1, \ldots, \xi_{\mathrm{mx}}) & \text{, if } \delta = \Pi_j \text{ for some } j \in [r] \\
\mathrm{trans}_0(\delta, \xi_1, \ldots \xi_{\mathrm{mx}}) & \text{, if } \delta = \Pi_j \text{ for some } j \notin [r] \\
(\delta\ \mathrm{trans}_0((\mathrm{sub}_r\ s_1\ t_1\ \ldots\ t_r), & \\
\quad \xi_1, \ldots, \xi_{\mathrm{mx}}) & \\
\quad \ldots & \text{, otherwise} \\
\quad \mathrm{trans}_0((\mathrm{sub}_r\ s_k\ t_1\ \ldots\ t_r), & \\
\quad \xi_1, \ldots, \xi_{\mathrm{mx}})) &
\end{cases} \\
\overset{(4.7)}{=} \; & \begin{cases}
\mathrm{trans}_0(t_j, \xi_1, \ldots, \xi_{\mathrm{mx}}) & \text{, if } \delta = \Pi_j \text{ for some } j \in [r] \\
\mathrm{trans}_0((\delta\ (\mathrm{sub}_r\ s_1\ t_1\ \ldots\ t_r) & \\
\quad \ldots & \text{, otherwise} \\
\quad (\mathrm{sub}_r\ s_k\ t_1\ \ldots\ t_r)), & \\
\quad \xi_1, \ldots, \xi_{\mathrm{mx}}) &
\end{cases} \\
\overset{\text{Def. 4.1}}{=} \; & \mathrm{trans}_0(\xi'|_p, \xi_1, \ldots, \xi_{\mathrm{mx}}).
\end{aligned}
$$

We conclude $\xi' \in S$.

(b) Let $f \in m^{-1}(1)$.

$$\begin{tikzcd} \xi'' \arrow[r, "\mathrm{trans}"] \arrow[d, Rightarrow, "M"'] & \mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \arrow[d, Rightarrow, "M"', "*"] \\ \xi' \arrow[r, "\mathrm{trans}"] & \mathrm{trans}_0(\xi', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \end{tikzcd}$$

We would like to show that the diagram above commutes. Therefore we firstly need to determine the impact of the redex $f\ (\delta\ s_1\ \ldots\ s_k)$ in ξ'' on $\mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}})$. Therefore, let g be a function symbol [44] with $f \neq g \in m^{-1}(1)$. We define the sets P' and P'' by

$$P' = \{\, \pi \in R_{M'}(t) \mid t = \mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}),\ \mathrm{label}_t(\pi) = f \,\}$$

$$P'' = \{\, \pi \in R_{M'}(t) \mid t = \mathrm{trans}_0(\xi''[g\ (\delta\ s_1\ \ldots\ s_k)]_p, \Pi_1, \ldots, \Pi_{\mathrm{mx}}), \\ \mathrm{label}_t(\pi) = f \,\}.$$

Note

$$\mathrm{occ}(\mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}})) = \mathrm{occ}(\mathrm{trans}_0(\xi''[g\,(\delta\, s_1\ \ldots\ s_k)]_p, \Pi_1, \ldots, \Pi_{\mathrm{mx}}))$$

and, thus, the finitely many redexes in $\mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}})$ created by the redex $\xi''|_p$ are at occurrences in $P = P' \setminus P''$. If $P = \emptyset$, then we deduce

$$\mathrm{trans}_M(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = \mathrm{trans}_M(\xi', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \in \mathrm{cl}_{\Rightarrow_{M'}}(\{e'\theta\})$$

and, consequently, $\xi' \in S$. Otherwise, according to Definition 3.7 we conclude $s_1, \ldots, s_k \in \mathcal{T}_\Delta$ and determine the context parameters $\xi_1, \ldots, \xi_{\mathrm{mx}} \in \mathrm{SF}(M')$ by stating that $\mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}})|_\pi = f\,(\delta\, s_1\ \ldots\ s_k)\,\xi_1\ \ldots\ \xi_{\mathrm{mx}}$ for some $\pi \in P$. Note that for every choice of π the resulting context parameters will be the same [45]. It is also easy to see that no element $\pi' \in P$ is a prefix of another element $\pi' \neq \pi'' \in P$ [46], hence having redexes at every $\pi \in P$ in $\mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}})$, we can apply rewriting [47] to all of the

[44] If no other function symbol is available, we simply add one together with dummy rules.

[45] Assume a term $t \in \mathrm{SF}(M)$ and an occurrence $p \in \mathrm{occ}(t)$ in t. By a simple induction on the length of p, we show that if trans_0 is applied to $t|_p$, then the context is unique. If the length of p is zero, this is immediate. Otherwise $p = p'.j$ with $p' \in \mathrm{occ}(t)$, $\mathrm{label}_t(p') \in (F \cup \Delta)^{(i)}$ and $j \in [i]$ for some $i \in \mathbb{N}_+$. Since by induction hypothesis the context is unique (else trans_0 is never applied to $t|_{p'}$ and, thus, not to $t|_p$), we get $\mathrm{trans}_0((\sigma\, t|_{p'.1}\ \ldots\ t|_{p'.i}), \xi_1, \ldots, \xi_{\mathrm{mx}})$. Exactly one defining equation of trans_0 is applicable and there is at most one call to $\mathrm{trans}_0(t|_{p'.j}, \ldots)$ with a unique context in its right hand side. Since translated (via trans_0) terms are just copied (or deleted), the context parameters of redexes of M' created by $t|_p$ are unique.

[46] Assume π' is a proper prefix of π''. By definition of sentential forms $\pi'.1$ is not a prefix of π''. However, for some $j \in \mathbb{N}_+$ with $j > 1 : \pi'.j$ must be a prefix of π'', but this violates the unique context parameter claim.

[47] We use the notation $t[t']_P$ with $P = \{p_1, \ldots, p_v\}$ with $v \in \mathbb{N}$ to denote $(\cdots((t[t']_{p_1})[t']_{p_2})\cdots)[t']_{p_v}$, if no element of P is a prefix of another (cf. [BN98]).

redexes (since no occurrence is a prefix of another) in parallel, yielding

$$
\begin{array}{rl}
& \mathrm{trans}_0(\xi', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\
= & \mathrm{trans}_0(\xi''[\mathrm{rhs}_M(f,\delta)\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k \,\}]_p, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\
= & \mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\
& \quad [\mathrm{trans}_0(\mathrm{rhs}_M(f,\delta)\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k \,\}, \xi_1, \ldots, \xi_{\mathrm{mx}})]_P \\
= & \mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\
& \quad [\mathrm{trans}_0(\mathrm{rhs}_M(f,\delta)\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k \,\}, \\
& \qquad y_1\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k, y_1 \mapsto \xi_1, \ldots, y_{\mathrm{mx}} \mapsto \xi_{\mathrm{mx}} \,\}, \\
& \qquad \ldots, \\
& \qquad y_{\mathrm{mx}}\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k, y_1 \mapsto \xi_1, \ldots, y_{\mathrm{mx}} \mapsto \xi_{\mathrm{mx}} \,\})]_P \\
\stackrel{\text{Lem. 4.9(a)}}{=} & \mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}})[\mathrm{trans}_0(\mathrm{rhs}_M(f,\delta), y_1, \ldots, y_{\mathrm{mx}}) \\
& \quad \{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k, y_1 \mapsto \xi_1, \ldots, y_{\mathrm{mx}} \mapsto \xi_{\mathrm{mx}} \,\}]_P \\
\stackrel{\text{Cons. 4.5}}{=} & \mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\
& [\mathrm{rhs}_{M'}(f,\delta)\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k, y_1 \mapsto \xi_1, \ldots, y_{\mathrm{mx}} \mapsto \xi_{\mathrm{mx}} \,\}]_P
\end{array}
$$

where $(f\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_{\mathrm{mx}}\ =\ \mathrm{rhs}_{M'}(f,\delta)) \in R'$. By Definition 3.8

$$
\begin{array}{rl}
& \mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\
\Rightarrow_{M'}^{\mathrm{card}(P)} & \mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\
& [\mathrm{rhs}_{M'}(f,\delta)\{\, x_1 \mapsto s_1, \ldots, x_k \mapsto s_k, y_1 \mapsto \xi_1, \ldots, y_{\mathrm{mx}} \mapsto \xi_{\mathrm{mx}} \,\}]_P
\end{array}
$$

and, thereby, since $\mathrm{trans}_0(\xi'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \in \mathrm{cl}_{\Rightarrow_{M'}}(\{e'\theta\})$ by induction hypothesis, we deduce $\mathrm{trans}_0(\xi', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \in \mathrm{cl}_{\Rightarrow_{M'}}(\{e'\theta\})$ and, consequently, $\xi' \in S$.

□

Example 4.15 (Illustrating the correspondence). Given the derivation closures

$$\mathrm{cl}_{\Rightarrow_{M_{\mathrm{rev}}}}(\{\mathrm{rev}\ (A\ (B\ (B\ N)))\}) \qquad \text{and} \qquad \mathrm{cl}_{\Rightarrow_{M'_{\mathrm{rev}}}}(\{\mathrm{rev}\ (A\ (B\ (B\ N)))\ N\}),$$

Figure 12 illuminates the correspondence between the sentential forms, where M_{rev} and M'_{rev} are the modular tree transducers of Example 4.2 and Example 4.6, respectively. Note the mere coincidence that the displayed mapping is surjective. □

Theorem 4.16 (Correctness of Construction 4.5). *Let* $M \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$ *and let* $M' = \mathfrak{C}_{4.5}(M)$.

$$\tau_M = \tau_{M'}.$$

Proof. Let $M = (F, m, \Delta, e, R)$ with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ for some $\mathrm{mx} \in \mathbb{N}$ and, further, let $M' = (F', \Delta, e', R')$. Arbitrarily select $t \in \mathcal{T}_\Delta$.

$$\mathrm{cl}_{\Rightarrow_M}(\{e\{\, x \mapsto t \,\}\}) \cap \mathcal{T}_\Delta = \{\tau_M(t)\} \qquad \text{and} \qquad \mathrm{cl}_{\Rightarrow_{M'}}(\{e'\{\, x \mapsto t \,\}\}) \cap \mathcal{T}_\Delta = \{\tau_{M'}(t)\},$$

since $\Rightarrow_M$ and $\Rightarrow_{M'}$ are canonical [EV91].

$$\tau_M(t) \stackrel{\text{Lem. 4.9(b)}}{=} \mathrm{trans}_0(\tau_M(t), \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \stackrel{\text{Lem. 4.14}}{\in} \mathrm{cl}_{\Rightarrow_{M'}}(\{e'\{\, x \mapsto t \,\}\})$$

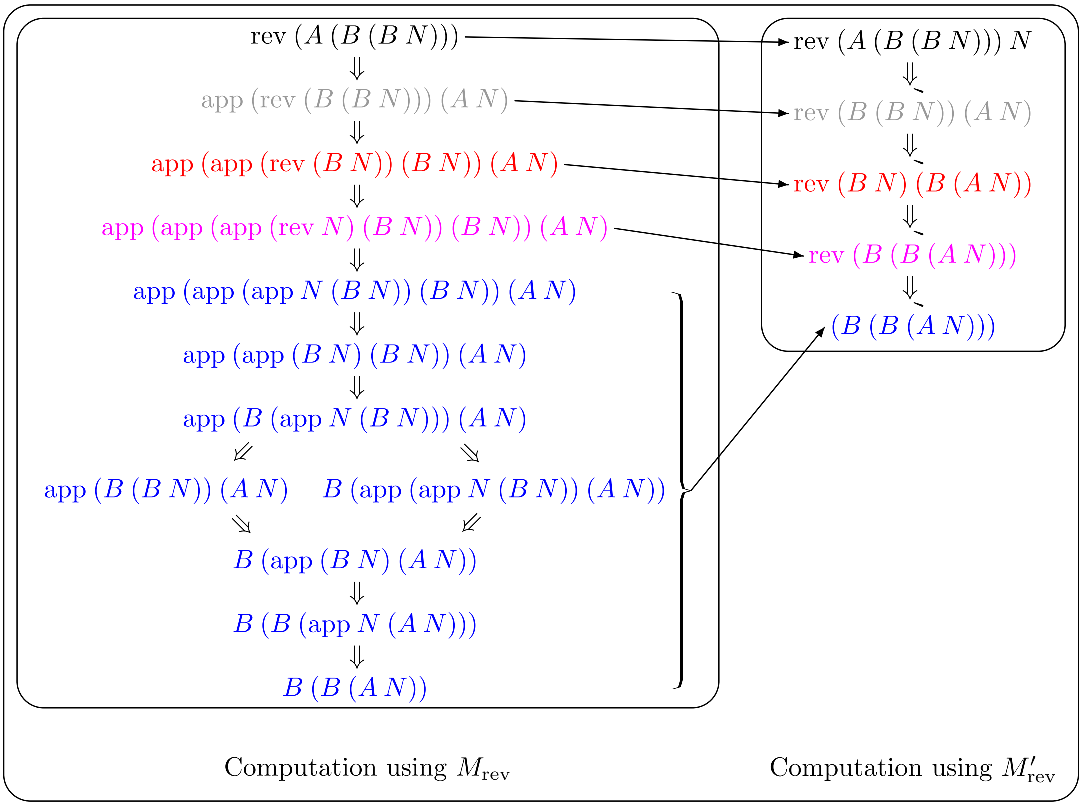

Figure 12: Correspondence between the sentential forms, where $\Rightarrow = \Rightarrow_{M_{\text{rev}}}$ and $\Rightarrow' = \Rightarrow_{M'_{\text{rev}}}$. The arrows denote the translation via $\text{trans}_0(\cdot, N)$ pointing towards the result.

Additionally, $\tau_M(t) \in \mathcal{T}_\Delta$, thus also $\tau_M(t) \in \text{cl}_{\Rightarrow_{M'}}(\{e'\{x \mapsto t\}\}) \cap \mathcal{T}_\Delta = \{\tau_{M'}(t)\}$. Consequently, $\tau_M(t) = \tau_{M'}(t)$. $\square$

5 A first efficiency analysis

In the last section we introduced a construction, which given a suitable modular tree transducer, constructs a macro tree transducer that induces the same translation as the original modular tree transducer does. This provides us with two equivalent means of computing this translation. Thus, the more efficient device should be used in implementations.

Before proceeding with efficiency considerations, we need to establish a formal efficiency measure. Within this thesis we identify the efficiency with the number of derivation steps needed to compute the normal form using a deterministic derivation relation (preferably call-by-need). Usually the amount of memory used is considered to be another important aspect of efficiency, but memory usage is out of the scope of this thesis.

Definition 5.1 (Efficiency measure). Let $M = (F, m, \Delta, e, R)$ be a modular tree transducer. The *number of derivation steps* needed to compute the normal form (with respect to the call-by-need derivation relation) of $e\{\, x \mapsto t \,\}$ for every $t \in \mathcal{T}_\Delta$, denoted $\mathrm{steps}_{\Mapsto_M}(t)$, is

$$\mathrm{steps}_{\Mapsto_M}(t) = n, \qquad \text{if and only if} \qquad e\{\, x \mapsto t \,\} \Mapsto_M^n \tau_M(t).$$

We will say that a modular tree transducer M' with $\tau_M = \tau_{M'}$ is *more efficient* than M, if and only if $\mathrm{steps}_{\Mapsto_{M'}}(t) < \mathrm{steps}_{\Mapsto_M}(t)$ for every $t \in \mathcal{T}_\Delta$. Consequently, we will call M' *at least as efficient* as M, if and only if $\mathrm{steps}_{\Mapsto_{M'}}(t) \leq \mathrm{steps}_{\Mapsto_M}(t)$ for every $t \in \mathcal{T}_\Delta$. □

To support that our efficiency measure truly scales according to the computation time, we feed the example modular tree transducers of Example 5.3 into the HASKELL interpreter HUGS [48] and compared the number of derivation steps according to our definition of efficiency with the number of reduction steps [49] outputted by the interpreter. The correlation coefficients [50] are 0.9998626981 and 0.9999999997 for the modular tree transducers M_{fib} and M'_{fib}, respectively. Thus a strong (almost linear) correlation cannot be denied [51].

Example 5.2 (Efficiency of M_{rev} and M'_{rev}). Reconsidering Example 4.2 together with the derivations of Figure 12 (there for the non-deterministic derivation relation), we can state

$$\mathrm{steps}_{\Mapsto_{M_{\mathrm{rev}}}}(A\,(B\,(B\,N))) = 10 \qquad \text{and} \qquad \mathrm{steps}_{\Mapsto_{M'_{\mathrm{rev}}}}(A\,(B\,(B\,N))) = 4.$$

Thus M'_{rev} is more efficient than M_{rev} on the input $A\,(B\,(B\,N))$, but not yet in general as required by the definition. Given an input list $t \in \mathcal{T}_\Delta$ of length $k \in \mathbb{N}$ (i.e. $\#_{\{A,B\}}(t) = k$) we derive the equations

$$\mathrm{steps}_{\Mapsto_{M_{\mathrm{rev}}}}(t) = \sum_{i=1}^{k+1} i = \frac{k^2+3k+2}{2} = \frac{k^2+k}{2} + k + 1 \qquad \text{and} \qquad \mathrm{steps}_{\Mapsto_{M'_{\mathrm{rev}}}}(t) = k+1.$$

[48] available under: <http://www.haskell.org/hugs>

[49] The interpreter uses different atomic units of computation; generally speaking this measure is known to provide a poor measure, if different control structures, e.g. a recursive and an iterative version of a program, are compared. As this is obviously not the case in our setting, we can rely on this measure, which has the advantage of being machine-independent.

[50] Statistical measure, denoted $\rho_{u,v}$ for sequences of numbers u and v both of length n, in the range $-1 \leq \rho_{u,v} \leq 1$, which is computed as $\rho_{u,v} = \frac{\frac{1}{n}\sum_{i=1}^{n}(u_i-\mu_u)(v_i-\mu_v)}{\sigma_u * \sigma_v}$ where σ_u and σ_v are the standard deviations of u and v, respectively, and μ_u and μ_v are the averages over u and v, respectively. Note that $|\rho_{u,v}| = 1$ signals a linear dependency.

[51] Profiler runs with the GLASGOW HASKELL COMPILATION SYSTEM <http://www.haskell.org/ghc> also support this claim.

Given those equations we can conclude that M'_{rev} is at least as efficient as M_{rev}. M'_{rev} is not more efficient than M_{rev}, since $\text{steps}_{\Rightarrow_{M_{\text{rev}}}}(N) = \text{steps}_{\Rightarrow_{M'_{\text{rev}}}}(N) = 1$. Nevertheless we experience an efficiency gain from quadratic complexity (with respect to the input tree size) to linear complexity.

□

Example 5.3 (Efficiency of M_{fib} and M'_{fib}). Let us establish a modular tree transducer $M_{\text{fib}} = (\{\text{fib}^{(1)}, \text{fib}'^{(1)}, \text{add}^{(2)}\}, m, \{S^{(1)}, Z^{(0)}\}, (\text{fib}\ x), R)$ with

- $m(\text{fib}) = m(\text{fib}') = 1$, $m(\text{add}) = 2$ and
- R contains the equations:

$$\begin{array}{lllcl|lllcl}
\text{fib} & Z & & = & S\,Z & \text{fib}' & Z & & = & Z \\
\text{fib} & (S\,x_1) & & = & \text{add}\,(\text{fib}'\,x_1)\,(\text{fib}\,x_1) & \text{fib}' & (S\,x_1) & & = & \text{fib}\,x_1 \\
\text{add} & Z & y_1 & = & y_1 & & & & & \\
\text{add} & (S\,x_1) & y_1 & = & S\,(\text{add}\,x_1\,y_1). & & & & &
\end{array}$$

According to Construction 4.5, the constructed macro tree transducer is

$$M'_{\text{fib}} = (\{\text{fib}^{(2)}, \text{fib}'^{(2)}\}, \{S^{(1)}, Z^{(0)}\}, (\text{fib}\ x\ Z), R')$$

with equations

$$\begin{array}{lllcl|lllcl}
\text{fib} & Z & y_1 & = & S\,y_1 & \text{fib}' & Z & y_1 & = & y_1 \\
\text{fib} & (S\,x_1) & y_1 & = & \text{fib}'\,x_1\,(\text{fib}\,x_1\,y_1) & \text{fib}' & (S\,x_1) & y_1 & = & \text{fib}\,x_1\,y_1.
\end{array}$$

By Theorem 4.16, for each $t = (S^n\ Z) \in \mathcal{T}_{\{S^{(1)}, Z^{(0)}\}}$ with $n \in \mathbb{N}$

$$\tau_{M_{\text{fib}}}(t) = \tau_{M'_{\text{fib}}}(t) = (S^{\text{Fibonacci}(n)}\ Z),$$

where Fibonacci(n) is the n-th fibonacci number [52]. Analyzing the efficiency of M_{fib} and M'_{fib} we gain the following statements for every input $t = (S^n\ Z) \in \mathcal{T}_{\{S^{(1)}, Z^{(0)}\}}$ with $n \in \mathbb{N}$

$$\begin{array}{rcl}
\text{steps}_{\Rightarrow_{M_{\text{fib}}}}(t) & = & \text{Fibonacci}(n+1) + \text{Fibonacci}(n+3) + \\
& & +(\sum_{i=0}^{n-2} \text{Fibonacci}(i) * \text{Fibonacci}(n-2-i)) - 3 \\
\text{steps}_{\Rightarrow_{M'_{\text{fib}}}}(t) & = & \text{Fibonacci}(n+3) - 2.
\end{array}$$

n	0	1	2	3	4	5	6	7
$\text{steps}_{\Rightarrow_{M_{\text{fib}}}}(S^n\ Z)$	1	4	9	17	31	54	93	158
$\text{steps}_{\Rightarrow_{M'_{\text{fib}}}}(S^n\ Z)$	1	3	6	11	19	32	53	87

Consequentially, M'_{fib} is at least as efficient as M_{fib}.

Proof. First, we will prove the more complicated statement (concerning M_{fib}) via natural induction on n.

[52]The first fibonacci numbers are $1, 1, 2, 3, 5, 8, 13, 21$.

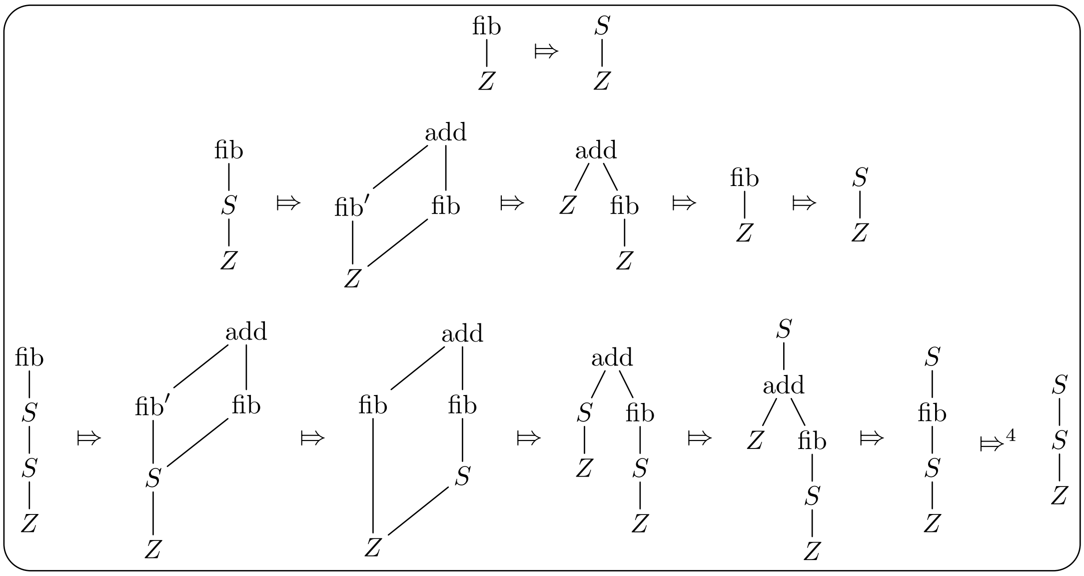

Figure 13: Verification for the induction base, where $\mapsto\!\!\!\!\Rightarrow \,=\, \mapsto\!\!\!\!\Rightarrow_{M_{\mathrm{fib}}}$

- **Induction base:** We consider the cases $n = 0$, $n = 1$ and $n = 2$, then

$$\begin{aligned}
\mathrm{steps}_{\mapsto\!\!\!\!\Rightarrow_{M_{\mathrm{fib}}}}(Z) &= \mathrm{Fibonacci}(1) + \mathrm{Fibonacci}(3) - 3 = 1\\
\mathrm{steps}_{\mapsto\!\!\!\!\Rightarrow_{M_{\mathrm{fib}}}}(S\ Z) &= \mathrm{Fibonacci}(2) + \mathrm{Fibonacci}(4) - 3 = 4\\
\mathrm{steps}_{\mapsto\!\!\!\!\Rightarrow_{M_{\mathrm{fib}}}}(S\ (S\ Z)) &= \mathrm{Fibonacci}(3) + \mathrm{Fibonacci}(5) + 1 - 3 = 9.
\end{aligned}$$

  The verification for those statements can be found in Figure 13.

- **Induction step:** Let $n \geq 3$. By induction hypothesis

$$\begin{aligned}
\mathrm{steps}_{\mapsto\!\!\!\!\Rightarrow_{M_{\mathrm{fib}}}}(S^{(n-1)}Z) = \mathrm{Fibonacci}(n) + \mathrm{Fibonacci}(n+2)+&\\
+ \left(\sum_{i=0}^{n-3} \mathrm{Fibonacci}(i) * \mathrm{Fibonacci}(n-3-i)\right) - 3&
\end{aligned}$$

  and also

$$\begin{aligned}
\mathrm{steps}_{\mapsto\!\!\!\!\Rightarrow_{M_{\mathrm{fib}}}}(S^{(n-2)}Z) = \mathrm{Fibonacci}(n-1) + \mathrm{Fibonacci}(n+1)+&\\
+ \left(\sum_{i=0}^{n-4} \mathrm{Fibonacci}(i) * \mathrm{Fibonacci}(n-4-i)\right) - 3.&
\end{aligned}$$

  A straightforward analysis of the recursive structure [53] of the equations yields [54]

  $\mathrm{fib}(S^n Z) \mapsto\!\!\!\!\Rightarrow_{M_{\mathrm{fib}}} \mathrm{add}(\mathrm{fib}'(S^{n-1}Z))(\mathrm{fib}(S^{n-1}Z)) \mapsto\!\!\!\!\Rightarrow_{M_{\mathrm{fib}}} \mathrm{add}(\mathrm{fib}(S^{n-2}Z))(\mathrm{fib}(S^{n-1}Z))$

[53] along with the knowledge that add $(S^{n_1}\ Z)\ (S^{n_2}\ Z) \mapsto\!\!\!\!\Rightarrow^{n_1+1}_{M_{\mathrm{fib}}} (S^{n_1+n_2}\ Z)$ for every $n_1, n_2 \in \mathbb{N}$

[54] Strictly speaking, the call-by-need derivation relation is only defined on sentential form graphs, however, whenever the sharing is not emphasized, we will also use the terms corresponding to the sentential form graphs.

$$
\begin{aligned}
& \text{steps}_{\Rightarrow_{M_{\text{fib}}}}(S^n\ Z) \\
= \;& 2 + \text{steps}_{\Rightarrow_{M_{\text{fib}}}}(S^{(n-1)}\ Z) + \text{steps}_{\Rightarrow_{M_{\text{fib}}}}(S^{(n-2)}\ Z) + \\
& \quad + \text{Fibonacci}(n-2) + 1 \\
\overset{\text{I.H.}}{=} \;& \text{steps}_{\Rightarrow_{M_{\text{fib}}}}(S^{(n-1)}\ Z) + \text{Fibonacci}(n-1) + \text{Fibonacci}(n+1) + \\
& \quad + (\sum_{i=0}^{n-4} \text{Fibonacci}(i) * \text{Fibonacci}(n-4-i)) + \text{Fibonacci}(n-2) \\
\overset{\text{I.H.}}{=} \;& \text{Fibonacci}(n-2) + \text{Fibonacci}(n-1) + \text{Fibonacci}(n) + \\
& \quad + \text{Fibonacci}(n+1) + \text{Fibonacci}(n+2) + \\
& \quad + (\sum_{i=0}^{n-3} \text{Fibonacci}(i) * \text{Fibonacci}(n-3-i)) + \\
& \quad + (\sum_{i=0}^{n-4} \text{Fibonacci}(i) * \text{Fibonacci}(n-4-i)) - 3 \\
= \;& \text{Fibonacci}(n-2) + \text{Fibonacci}(n+1) + \text{Fibonacci}(n+3) + \\
& \quad + (\sum_{i=0}^{n-4} \text{Fibonacci}(i) * \text{Fibonacci}(n-2-i)) + \text{Fibonacci}(n-3) - 3 \\
= \;& \text{Fibonacci}(n+1) + \text{Fibonacci}(n+3) + \\
& \quad + (\sum_{i=0}^{n-2} \text{Fibonacci}(i) * \text{Fibonacci}(n-2-i)) - 3
\end{aligned}
$$

This completes the proof of the first statement; the second statement will be proven in a similar fashion.

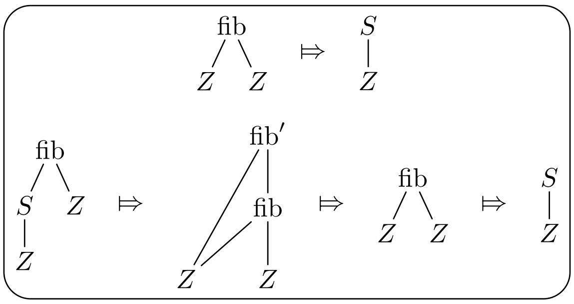

Figure 14: Verification for the induction base, where $\Rightarrow\; = \;\Rightarrow_{M'_{\text{fib}}}$

- **Induction base:** Let $n = 0$ and $n = 1$, then

$$
\begin{aligned}
\text{steps}_{\Rightarrow_{M'_{\text{fib}}}}(Z) &= \text{Fibonacci}(3) - 2 = 1 \\
\text{steps}_{\Rightarrow_{M'_{\text{fib}}}}(S\ Z) &= \text{Fibonacci}(4) - 2 = 3.
\end{aligned}
$$

Both statements are verified by Figure 14.

- **Induction step:** Let $n \geq 2$. By induction hypothesis

$$\begin{aligned} \mathrm{steps}_{\Rightarrow_{M'_{\mathrm{fib}}}}(S^{(n-1)}Z) &= \mathrm{Fibonacci}(n+2)-2 \\ \mathrm{steps}_{\Rightarrow_{M'_{\mathrm{fib}}}}(S^{(n-2)}Z) &= \mathrm{Fibonacci}(n+1)-2. \end{aligned}$$

Another straightforward analysis of the recursive structure of the equations of M'_{fib} yields

$$\mathrm{fib}\,(S^n\,Z) \Rightarrow_{M'_{\mathrm{fib}}} \mathrm{fib}'\,(S^{n-1}\,Z)\,(\mathrm{fib}\,(S^{n-1}\,Z)\,Z) \Rightarrow_{M'_{\mathrm{fib}}} \mathrm{fib}\,(S^{n-2}\,Z)\,(\mathrm{fib}\,(S^{n-1}\,Z)\,Z)$$

$$\begin{aligned} \mathrm{steps}_{\Rightarrow_{M'_{\mathrm{fib}}}}(S^n\,Z) &= 2+\mathrm{steps}_{\Rightarrow_{M'_{\mathrm{fib}}}}(S^{(n-1)}\,Z)+\mathrm{steps}_{\Rightarrow_{M'_{\mathrm{fib}}}}(S^{(n-2)}\,Z) \\ &\overset{\text{I.H.}}{=} 2+\mathrm{Fibonacci}(n+2)-2+\mathrm{Fibonacci}(n+1)-2 \\ &= \mathrm{Fibonacci}(n+3)-2. \end{aligned}$$

Hence, both statements are finally justified. □

The previous examples showed that the construction might be well beneficial concerning the efficiency, but in the example to follow we will show that an efficiency deterioration is possible as well.

Example 5.4 (An efficiency deteriorating example). We introduce the modular tree transducer $M_{\mathrm{doub}} = (\{\mathrm{doub}^{(1)}, \mathrm{sub}^{(2)}\}, m, \{\sigma^{(2)}, \alpha^{(0)}\}, (\mathrm{doub}\,x), R)$ with

- $m(\mathrm{doub}) = 1$, $m(\mathrm{sub}) = 2$ and
- R contains the equations:

$$\begin{array}{lllcl} \mathrm{doub} & \alpha & & = & \alpha \\ \mathrm{doub} & (\sigma\,x_1\,x_2) & & = & \mathrm{sub}\,(\sigma\,\alpha\,\alpha)\,(\sigma\,(\mathrm{doub}\,x_1)\,(\mathrm{doub}\,x_2)) \\ \mathrm{sub} & \alpha & y_1 & = & y_1 \\ \mathrm{sub} & (\sigma\,x_1\,x_2) & y_1 & = & \sigma\,(\mathrm{sub}\,x_1\,y_1)\,(\mathrm{sub}\,x_2\,y_1). \end{array}$$

Construction 4.5 yields the macro tree transducer

$$M'_{\mathrm{doub}} = (\{\mathrm{doub}^{(2)}\}, \{\sigma^{(2)}, \alpha^{(0)}\}, (\mathrm{doub}\,x\,\alpha), R')$$

with equations

$$\begin{array}{lllcl} \mathrm{doub} & \alpha & y_1 & = & y_1 \\ \mathrm{doub} & (\sigma\,x_1\,x_2) & y_1 & = & \sigma\,(\sigma\,(\mathrm{doub}\,x_1\,y_1)\,(\mathrm{doub}\,x_2\,y_1))\,(\sigma\,(\mathrm{doub}\,x_1\,y_1)\,(\mathrm{doub}\,x_2\,y_1)). \end{array}$$

Assuming an input tree $t \in \mathcal{T}_{\{\sigma^{(2)},\alpha^{(0)}\}}$ we derive the statements [55]

$$\mathrm{steps}_{\Rightarrow_{M_{\mathrm{doub}}}}(t) = \#_\alpha(t) + 4 * \#_\sigma(t) \qquad \text{and} \qquad \mathrm{steps}_{\Rightarrow_{M'_{\mathrm{doub}}}}(t) = \sum_{p \in \mathrm{occ}(t)} 2^{|p|}.$$

[55] The proofs are again straightforward, though redexes can potentially be shared (cf. Example 5.5 for an illustration of the effect). It is obvious that M_{doub} might share redexes, while M'_{doub} cannot. Thus, given sufficiently long computations the shared redexes will eventually pay off, especially since the penalty incurred by the call to the substitution function symbol sub in a right hand side of M_{doub} is constant.

t	α	$(\sigma\,\alpha\,\alpha)$	$(\sigma\,\alpha(\sigma\,\alpha\,\alpha))$	$(\sigma\,(\sigma\,\alpha\,(\sigma\,\alpha\,\alpha))\,(\sigma\,(\sigma\,(\sigma\,\alpha\,\alpha)\,\alpha)\,\alpha))$
$\text{steps}_{\Rightarrow_{M_{\text{doub}}}}(t)$	1	6	11	31
$\text{steps}_{\Rightarrow_{M'_{\text{doub}}}}(t)$	1	5	13	85

Thus, M_{doub} and M'_{doub} are incomparable with respect to the number of derivation steps (efficiency), since the choice of the more efficient device depends on the actual input. However, in this example we have to confront an efficiency loss from linear complexity (in the size of the input tree) to exponential complexity. □

5.1 Refinement of the construction

As we have already seen in Example 5.4, Construction 4.5 might seriously deteriorate the efficiency of a program. In order to integrate the construction into an optimizing compiler, we need to establish syntactic checks which ensure efficiency non-deterioration. We already remarked that the deteriorating effect is only due to the duplication of redexes ($\text{card}(P) > 1$ in the proof of Lemma 4.14). Such a duplication of a redex also occurs in the derivation using the original program with the notable difference that all the duplicates will be shared, i.e. they all share a common root node in the sentential form graph. We will illustrate this particular effect in the following example.

Example 5.5 (The cause of the deterioration). We reuse the modular tree transducer M_{doub} and the constructed macro tree transducer M'_{doub} of Example 5.4. We will show part of a derivation using both devices.

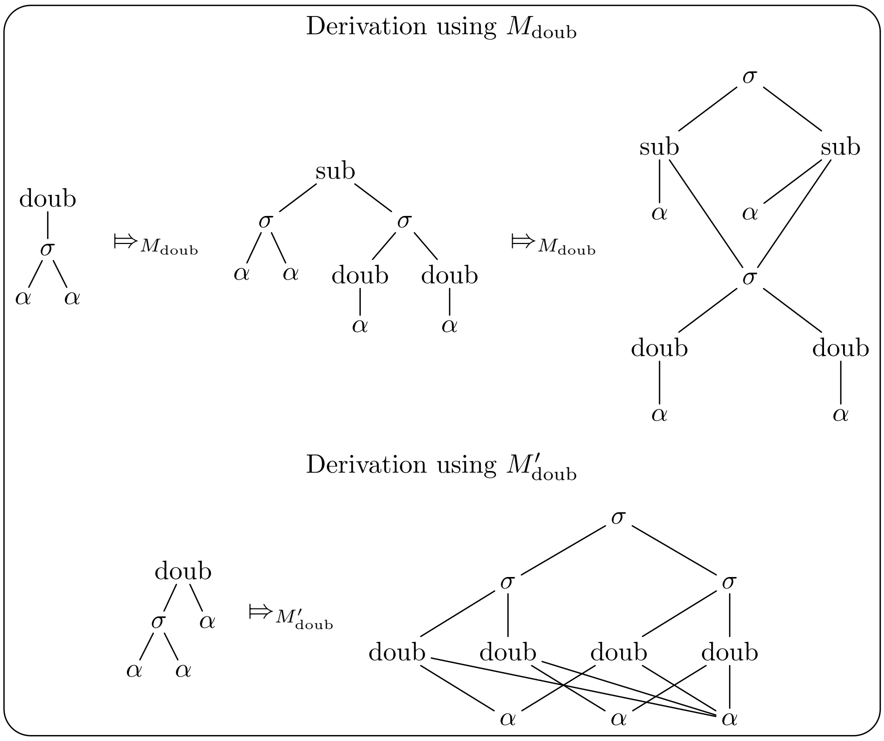

Figure 15: Illustrating the cause of the inefficiency of M'_{doub} compared to M_{doub}.

In this example the macro tree transducer is still more efficient than its corresponding modular tree transducer, but as we have seen in Example 5.4 for larger input trees the additional redexes far outweigh the penalty of the derivation steps of M_{doub} invested to reduce redexes with root sub.

□

Instead of restricting the input modular tree transducer, such that this effect is avoided, we manipulate the construction to cope with this situation. The result will no longer be a macro tree transducer, rather it will be a macro tree transducer with term graph rewrite rules. Modular tree transducers with term graph rewrite rules will be defined below.

Definition 5.6 (Modular tree transducer with term graph rewrite rules). A *modular tree transducer with term graph rewrite rules* is a quintuple $M = (F, m, \Delta, e, R)$ with R being a set of term graph rewrite rules, such that $M' = (F, m, \Delta, e, \{\, \text{term}(G) \,|\, G \in R\,\})$ is a regular modular tree transducer. All properties translate in an obvious fashion to modular tree transducers with term graph rewrite rules. □

Definition 5.7 (Semantics of the newly defined construct). Let $M = (F, m, \Delta, e, R)$ be a modular tree transducer with term graph rewrite rules. The *call-by-need* derivation relation $\Mapsto_M$ on sentential form graphs induced by M relates $\Xi_1 \Mapsto_M \Xi_2$ for every $\Xi_1, \Xi_2 \in \text{SFG}(M)$, if and only if

1. **Locate redex:** there exists a least element $p \in R_M(\Xi_1)$ with respect to the lexicographic ordering, which requires a term graph homomorphism ψ from $G|_1$ to $\Xi_1|_p$ for some $G \in R$,

2. **Build:** let $G|_1 = (N_{\text{lhs}}, E_{\text{lhs}}, l_{\text{lhs}}, r_{\text{lhs}})$ and $G|_2 = (N_{\text{rhs}}, E_{\text{rhs}}, l_{\text{rhs}}, r_{\text{rhs}})$ with $N_{\text{rhs}} \cap N_{\Xi_1} = \emptyset$, then $G' = (N', E', l', r_{\Xi_1})$ is defined as

$$
\begin{aligned}
N' &= N_{\Xi_1} \cup (N_{\text{rhs}} \setminus N_{\text{lhs}}) \\
E'(n,i) &= \begin{cases} E_{\Xi_1}(n,i) & \text{, if } n \in N_{\Xi_1} \\ E_{\text{rhs}}(n,i) & \text{, if } n, E_{\text{rhs}}(n,i) \in N_{\text{rhs}} \setminus N_{\text{lhs}} \\ \psi(E_{\text{rhs}}(n,i)) & \text{, otherwise} \end{cases} \\
l'(n) &= \begin{cases} l_{\Xi_1}(n) & \text{, if } n \in N_{\Xi_1} \\ l_{\text{rhs}}(n) & \text{, otherwise} \end{cases}
\end{aligned}
$$

for every $n \in N'$ and $i \in \mathbb{N}_+$ [56] and

3. **Redirect:** $\Xi_2 = G'[\psi(r_{\text{lhs}}) \rightsquigarrow r_{\text{rhs}}]$, where we ensure the connectedness property, i.e. trigger garbage collection if necessary.

Alternatively, $\Xi_1 \Mapsto_M \Xi_2$, if and only if there exists a least element $p \in R_M(\Xi_1)$ with respect to the lexicographic ordering, there exists for some $G \in R$ a non-collapsing term graph homomorphism ψ_{lhs} from $G|_1$ to $\Xi_1|_p$ and a non-collapsing term graph homomorphism ψ_{rhs} from $G|_2$ to $\Xi_2|_p$ with $\psi_{\text{lhs}}(n) = \psi_{\text{rhs}}(n)$ for every $n \in N_{G|_1} \cap N_{G|_2}$,

[56]We assume that the functions are undefined for the given parameters, if none of the above displayed cases yields a definite result.

and, additionally, there exists a term graph isomorphism ϖ from $\Xi_1[\psi_{\text{lhs}}(r_{G|_1}) \rightsquigarrow z]$ to $\Xi_2[\psi_{\text{rhs}}(r_{G|_2}) \rightsquigarrow z]$ [57] for some $z \notin N_{\Xi_1} \cup N_{\Xi_2} \cup N_G$ [58].

The *translation* induced by M, denoted τ_M, is a mapping $\tau_M : \mathcal{T}_\Delta \longrightarrow \mathcal{T}_\Delta$ defined for every $t \in \mathcal{T}_\Delta$ by

$$\tau_M(t) = \text{term}(\text{nf}_{\mapsto\!\!\!\Rightarrow_M}(G_{e\{\,x\mapsto t\,\}})).$$

□

Lemma 5.8 (Induced translations coincide). *Let $M = (F, m, \Delta, e, R)$ be a modular tree transducer with term graph rewrite rules and $M' = (F, m, \Delta, e, \{\,\text{term}(G) \mid G \in R\,\})$.*

$$\tau_M = \tau_{M'}$$

Proof. The results of Theorem 3.18 extend naturally to $\mapsto\!\!\!\Rightarrow_M$ with M being a modular tree transducer with term graph rewrite rules and $\Rightarrow_{M'}$ (cf. [BvEG$^+$87]) [59].

- **Soundness:** Thus, by Theorem 3.18(3), for every $t \in \mathcal{T}_\Delta$:

  if $G_{e\{\,x\mapsto t\,\}} \mapsto\!\!\!\Rightarrow^*_M \text{nf}_{\mapsto\!\!\!\Rightarrow_M}(G_{e\{\,x\mapsto t\,\}})$, then $e\{\,x \mapsto t\,\} \Rightarrow^*_{M'} \text{term}(\text{nf}_{\mapsto\!\!\!\Rightarrow_M}(G_{e\{\,x\mapsto t\,\}}))$.

  The precondition is fulfilled for every $t \in \mathcal{T}_\Delta$, due to Theorem 3.18(2), and, additionally, by Definition 5.7: $\text{term}(\text{nf}_{\mapsto\!\!\!\Rightarrow_M}(G_{e\{\,x\mapsto t\,\}})) = \tau_M(t)$. Since $\tau_M(t) \in \mathcal{T}_\Delta$, $\tau_M(t)$ is also irreducible with respect to $\Rightarrow_{M'}$ and, thus, by the uniqueness (cf. Theorem 3.9) of the normal form, also $\tau_{M'}(t) = \tau_M(t)$.

- **Completeness:** For every $t \in \mathcal{T}_\Delta$, there exists $\Xi \in \text{SFG}(M)$ such that $G_{e\{\,x\mapsto t\,\}} \mapsto\!\!\!\Rightarrow^*_M \Xi$ and $\text{term}(\Xi) = \text{nf}_{\Rightarrow_{M'}}(e\{\,x \mapsto t\,\})$ by Theorem 3.18(4). Since $\text{nf}_{\Rightarrow_{M'}}(e\{\,x \mapsto t\,\}) \in \mathcal{T}_\Delta$ by Theorem 3.9, we conclude that Ξ is irreducible with respect to $\mapsto\!\!\!\Rightarrow_M$, thus $\tau_M(t) = \text{term}(\Xi)$ by Definition 5.7. Consequently, $\tau_{M'}(t) = \tau_M(t)$.

□

Although this thesis is not focussed on the language-theoretic aspects of tree transducer theory, we will informally discuss the implications of the modification. In particular, the transformational power, i.e. the class of translations that can be computed, of the newly defined construct is equal to the transformational power of regular modular tree transducers. The above definition already shows that the translations induced by modular tree transducer with term graph rewrite rules can also be induced by regular modular tree transducers. On the other hand a translation induced by a modular tree transducer can also be induced by a modular tree transducer with term graph rewrite rules by using the tree representation or even the variable-shared term graphs corresponding to the original (term) rewrite rules.

[57] We implicitly perform garbage collection.

[58] This approach is less constructive and uses the notion of a context, since both $\Xi_1[\psi_{\text{lhs}}(r_{G|_1}) \rightsquigarrow z]$ and $\Xi_2[\psi_{\text{rhs}}(r_{G|_2}) \rightsquigarrow z]$ can be interpreted as contexts which do not change (due to the isomorphism). The matching of the left hand side is performed by the homomorphism ψ_{lhs}, while the right hand side matches with the subgraph $\Xi_2|_p$ via homomorphism ψ_{rhs}. The additional restriction on ψ_{lhs} and ψ_{rhs} ensures that variable nodes are instantiated equally.

[59] The one notable exception is item (5), which does not hold true, if some term graph rewrite rule of R is not variable-shared. Any term graph rewrite rule constructed in Construction 5.10, however, is trivially variable-shared.

Example 5.9 (M'_{doub} with term graph rewrite rules). Let $M'_{\mathrm{doub}} = (F, \Delta, e, R)$ be the macro tree transducer of Example 5.4. Then $M''_{\mathrm{doub}} = (F, \Delta, e, R')$ is a macro tree transducer with term graph rewrite rules R' depicted in Figure 16, such that $R = \{\, \mathrm{term}(G) \mid G \in R' \,\}$. Reconsidering the number of derivation steps for this modular tree

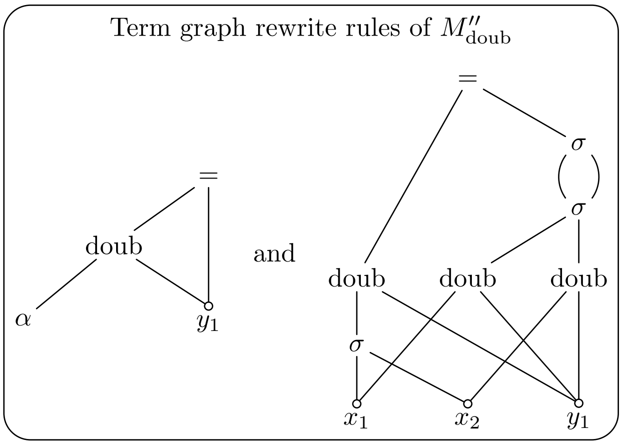

Figure 16: Term graph rewrite rules of M''_{doub}.

transducer with term graph rewrite rules, we gain the following equation

$$\mathrm{steps}_{\Rightarrow_{M''_{\mathrm{doub}}}}(t) = \#_{\{\alpha,\sigma\}}(t)$$

for every input tree $t \in \mathcal{T}_\Delta$. Thus M''_{doub} is at least as efficient as M'_{doub} as well as M_{doub}. □

Construction 5.10 (Refinement of Construction 4.5). Let $M \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$ with $M = (F, m, \Delta, e, R)$ and substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ for some $\mathrm{mx} \in \mathbb{N}$. We construct a macro tree transducer with term graph rewrite rules $M' = (F', \Delta, e', R')$, denoted $\mathfrak{C}_{5.10}(M)$, as follows:

- $F' = \{\, f^{(\mathrm{mx}+1)} \mid f \in m^{-1}(1) \,\}$,
- $e' = f\, x\, \Pi_1\, \ldots\, \Pi_{\mathrm{mx}}$, if $e = (f\ x)$ with $f \in m^{-1}(1)$, and
- R' contains for every $k \in \mathbb{N}$, $f \in F'$ and $\delta \in \Delta^{(k)}$ with $\rho = (f\ (\delta\ x_1\ \ldots\ x_k) = \mathrm{rhs}_M(f, \delta)) \in R$ the term graph rewrite rule G'_ρ constructed out of the variable-shared term graph rewrite rule $G_\rho = (N_\rho, E_\rho, l_\rho, r_\rho)$ via the term graph G', which is defined as
$$G' = (N_\rho \cup \{\, y_i \mid i \in [\mathrm{mx}] \,\}, E', l_\rho, r_\rho)$$
with
$$E' = E_{G_\rho|_1} \cup \{\, ((E_\rho(r_\rho, 1), i+1), y_i) \mid i \in [\mathrm{mx}] \,\} \cup \{((r_\rho, 2), r_H)\} \cup E_H$$
where $H = \mathrm{trans}_{G_Y,k}(G_\rho|_2, y_1, \ldots, y_{\mathrm{mx}})$ with $G_Y = (Y_{\mathrm{mx}}, \emptyset, \emptyset)$. However, the result may be a non-connected term graph, so we invoke garbage collection, if necessary, to get the final term graph rewrite rule G'_ρ. The application of the $\mathrm{trans}_{G_Y,k}$-mapping is displayed in Figure 17.

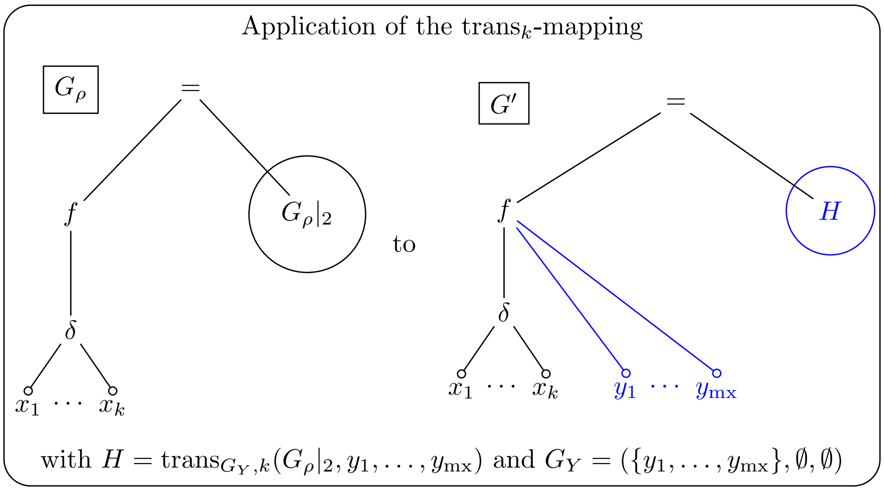

Figure 17: Applying the $\text{trans}_{G_Y,k}$-mapping.

We will call the first parameter of the $\text{trans}_{G,k}$-mapping *input graph*, the result *output graph* and G *intermediate result graph*. For every $G \in \mathcal{TG}_{\text{nr}}(\text{RHS}_{\text{MAC}}(F', \Delta, X_k, Y_{\text{mx}}))$ the mapping

$$\text{trans}_{G,k} : \{ G_{\text{rhs}} \mid \text{rhs} \in \text{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, \emptyset), N_{G_{\text{rhs}}} \cap N_G \subseteq X_k \} \times N_G^{\text{mx}} \longrightarrow \mathcal{TG}(\text{RHS}_{\text{MAC}}(F', \Delta, X_k, Y_{\text{mx}}))$$

is defined for every $\varrho_1, \dots, \varrho_{\text{mx}} \in N_G$ via case analysis over its first argument $H = (N_H, E_H, l_H, r_H)$ with

$$H \in \{ G_{\text{rhs}} \mid \text{rhs} \in \text{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, \emptyset), N_{G_{\text{rhs}}} \cap N_G \subseteq X_k \}.$$

- Let $l_H(r_H) = \Pi_j$ for some $j \in [\text{mx}]$, then
$$\text{trans}_{G,k}(H, \varrho_1, \dots, \varrho_{\text{mx}}) = (N_G, E_G, l_G, \varrho_j). \tag{5.1}$$
- Let $l_H(r_H) \in m^{-1}(1)$, then
$$\text{trans}_{G,k}(H, \varrho_1, \dots, \varrho_{\text{mx}}) = (N_{H'}, E', l_{H'}, r_H) \tag{5.2}$$
where $H' = H \cup G$ and $E' = E_{H'} \cup \{ ((r_H, i+1), \varrho_i) \mid i \in [\text{mx}] \}$.
- Let $l_H(r_H) \in \Delta^{(a)} \setminus \Pi$ with $a \in \mathbb{N}$, then
$$\text{trans}_{G,k}(H, \varrho_1, \dots, \varrho_{\text{mx}}) = (N_{H'} \cup \{r_H\}, E_{H'} \cup E', l_{H'} \cup \{(r_H, l_H(r_H))\}, r_H) \tag{5.3}$$
where $H' = \bigcup_{i \in [a]} H_i$ with $H_i = \text{trans}_{G,k}(H|_i, \varrho_1, \dots, \varrho_{\text{mx}})$ for every $i \in [a]$ and $E' = \{ ((r_H, i), r_{H_i}) \mid i \in [a] \}$.
- Let $l_H(r_H) \in m^{-1}(2)^{(a+1)}$ with $a \in \mathbb{N}$, then
$$\text{trans}_{G,k}(H, \varrho_1, \dots, \varrho_{\text{mx}}) = \text{trans}_{H',k}(H|_1, r_{H_1}, \dots, r_{H_a}, \varrho_{a+1}, \dots, \varrho_{\text{mx}}) \tag{5.4}$$
where $H' = G \cup \bigcup_{i \in [a]} H_i$ with $H_i = \text{trans}_{G,k}(H|_{i+1}, \varrho_1, \dots, \varrho_{\text{mx}})$ for every $i \in [a]$.

□

In the figures to follow, we often identify a node and the term graph rooted at that particular node. This allows us to express the $\mathrm{trans}_{G,k}$-mapping graphically without inconvenient descriptions as in Figure 17. However, we should keep in mind that in general the difference is noteworthy. Additionally, we drop the first subscript of the $\mathrm{trans}_{G,k}$-mapping in graphical depictions, assuming it is the (non-rooted) term graph containing exactly all the term graphs, of which the context parameters are the root. Keeping this in mind, we can also think of the mapping as a term graph rewrite system [BvEG+87], of

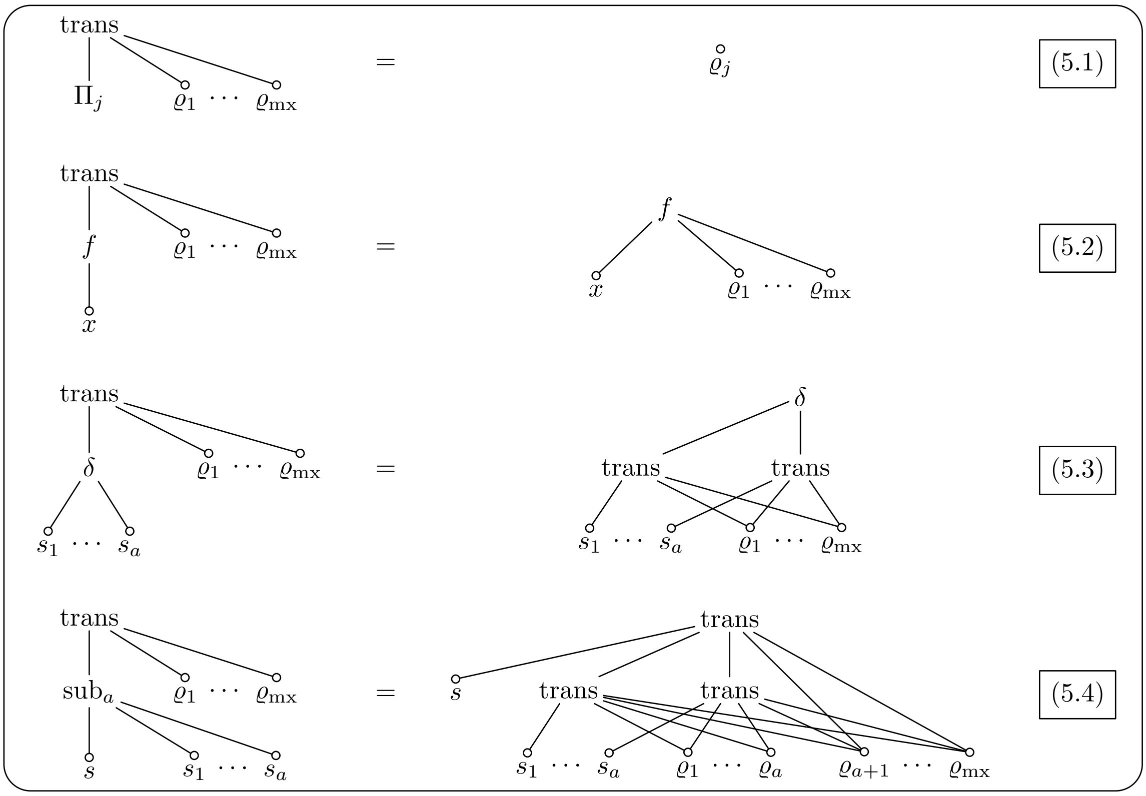

Figure 18: The transformation $\mathrm{trans} = \mathrm{trans}_{G,k}$ on term graphs.

which the defining equations are depicted in Figure 18. Note that we implicitly perform garbage collection.

We should verify that all calls to the $\mathrm{trans}_{G,k}$-mapping occurring in the definition are valid (respect the typing) and that the term graph unions are well-defined. Therefore we prove the following lemma.

Lemma 5.11 **(**$\mathrm{trans}_{G,k}$**-mapping is well-defined).** *Every call to* $\mathrm{trans}_{G,k}$ *in Construction 5.10 is well-defined, i.e. respects the typing and the term graph union conditions.*

Proof. Let $M = (F, m, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$ be the considered modular tree transducer and let $F' = \{\, f^{(\mathrm{mx}+1)} \mid f \in m^{-1}(1) \,\}$. Assume a call

$$G' = \mathrm{trans}_{G,k}(H, \varrho_1, \ldots, \varrho_{\mathrm{mx}})$$

for $k \in \mathbb{N}$, $G \in \mathcal{TG}_{\text{nr}}(\text{RHS}_{\text{MAC}}(F', \Delta, X_k, Y_{\text{mx}}))$, $\varrho_1, \ldots, \varrho_{\text{mx}} \in N_G$ and

$$H \in \{\, G_{\text{rhs}} \mid \text{rhs} \in \text{RHS}_{m^{-1}(1), m^{-1}(2), \Delta}(X_k, \emptyset) \,\}.$$

We prove the statements

1. $N_{G'} \subseteq N_G \cup N_H$ and
2. $N_G \cap N_H \subseteq X_k$.

The latter property assures that the call respects the typing and both statements together verify that each term graph union appearing in Construction 5.10 is well-defined.

(1) We prove the statement inductively on the term graph structure of H.

- **Induction base:**
  - Let $l_H(r_H) \in \Pi$, then $G' = (N_G, E_G, l_G, \varrho_j)$ for some $j \in [\text{mx}]$ and trivially $N_{G'} \subseteq N_G$.
  - Let $l_H(r_H) \in m^{-1}(1)$, then $G' = (N_{H'}, E', l_{H'}, r_H)$ with $H' = H \cup G$ and $E' = E_{H'} \cup \{\, ((r_H, i+1), \varrho_i) \mid i \in [\text{mx}] \,\}$, so $N_{G'} \subseteq N_G \cup N_H$.
- **Induction step:**
  - Let $l_H(r_H) \in \Delta^{(a)}$ for some $a \in \mathbb{N}$, hence by induction hypothesis for every $i \in [a]$ with $H_i = \text{trans}_{G,k}(H|_i, \varrho_1, \ldots, \varrho_{\text{mx}})$, we have $N_{H_i} \subseteq N_G \cup N_{H|_i}$.

    $$G' = (N_{H'} \cup \{r_H\}, E_{H'} \cup E', l_{H'} \cup \{(r_H, l_H(r_H))\}, r_H),$$

    where $H' = \bigcup_{i \in [a]} H_i$ with $H_i = \text{trans}_{G,k}(H|_i, \varrho_1, \ldots, \varrho_{\text{mx}})$ for every $i \in [a]$ and $E' = \{\, ((r_H, i), r_{H_i}) \mid i \in [a] \,\}$. Thus, $N_{G'} \subseteq \bigcup_{i \in [a]} (N_G \cup N_{H|_i}) \cup \{r_H\} \subseteq N_G \cup N_H$.
  - Let $l_H(r_H) \in m^{-1}(2)^{(a+1)}$ for some $a \in \mathbb{N}$, hence by induction hypothesis for every $i \in [a]$ with $H_i = \text{trans}_{G,k}(H|_{i+1}, \varrho_1, \ldots, \varrho_{\text{mx}})$, we have $N_{H_i} \subseteq N_G \cup N_{H|_{i+1}}$.

    $$G' = \text{trans}_{H',k}(H|_1, r_{H_1}, \ldots, r_{H_a}, \varrho_{a+1}, \ldots, \varrho_{\text{mx}}),$$

    where $H' = G \cup \bigcup_{i \in [a]} H_i$ with $H_i = \text{trans}_{G,k}(H|_{i+1}, \varrho_1, \ldots, \varrho_{\text{mx}})$ for every $i \in [a]$. Since $H' \in \mathcal{TG}_{\text{nr}}(\text{RHS}_{\text{MAC}}(F', \Delta, X_k, Y_{\text{mx}}))$, $r_{H_1}, \ldots, r_{H_a} \in N_{H'}$ and $\varrho_{a+1}, \ldots, \varrho_{\text{mx}} \in N_{H'}$, additionally, by induction hypothesis $N_{G'} \subseteq N_{H'} \cup N_{H|_1}$. Consequently,

    $$N_{G'} \subseteq N_{H|_1} \cup N_G \cup \bigcup_{i \in [a]} N_{H_i} \subseteq N_{H|_1} \cup N_G \cup \bigcup_{i \in [a]} (N_G \cup N_{H|_{i+1}}) \subseteq N_G \cup N_H.$$

(2) Since the call emerges from a call $\text{trans}_{G_Y,k}(G_\rho|_2, y_1, \ldots, y_{\text{mx}})$ with $G_Y = (Y_{\text{mx}}, \emptyset, \emptyset)$ and $\rho \in R$, H obviously is a term subgraph of $G_\rho|_2$, thus for some unique [60] $p \in \text{occ}(G_\rho|_2) : G_\rho|_{2.p} = H$. We prove the statement via induction over the path structure of p.

[60] Only variable nodes are shared in G_ρ, so only for variable nodes there exist different paths p, but $r_H \notin X_k$ (subgraphs rooted at variable nodes do not constitute valid right hand sides).

- **Induction base:** Let $p = \varepsilon$, then $H = G_\rho|_2$ and $G = G_Y$. Obviously $N_{G_Y} \cap N_{G_\rho|_2} = Y_{\mathrm{mx}} \cap N_{G_\rho|_2} = \emptyset$.
- **Induction step:** Let $p = \bar{p}.j$ with $j \in \mathbb{N}_+$ and by $p \in \mathrm{occ}(G_\rho|_2)$ also $l_{G_\rho}(r_{G_\rho|_{2.\bar{p}}}) \in (\Sigma \cup F)^{(a)}$ with $a \in \mathbb{N}_+$. Thus, $j \in [a]$.
  - Let $l_{G_\rho}(r_{G_\rho|_{2.\bar{p}}}) \in m^{-1}(1)$. Obviously $j = 1$ and this case is not applicable, since there is no call $\mathrm{trans}_{G,k}(G_\rho|_p, \varrho_1, \ldots, \varrho_{\mathrm{mx}})$ (the recursion argument of a function symbol of the first module is an input subtree variable $r_{G_\rho|_p} \in X_k$, which is not translated).
  - Let $l_{G_\rho}(r_{G_\rho|_{2.\bar{p}}}) \in \Delta$. Induction hypothesis $N_G \cap N_H \subseteq X_k$ assures $N_G \cap N_{H|_i} \subseteq X_k$ for every $i \in [a]$ and, thereby, the property trivially holds for every $j \in [a]$ and $\mathrm{trans}_{G,k}(H|_j, \varrho_1, \ldots, \varrho_{\mathrm{mx}})$.
  - Let $l_{G_\rho}(r_{G_\rho|_{2.\bar{p}}}) \in m^{-1}(2)$. Again the induction hypothesis $N_G \cap N_H \subseteq X_k$ assures $N_G \cap N_{H|_i} \subseteq X_k$ for every $i \in [a]$ and, thereby, the property trivially holds for every $j \in [2, \mathrm{a}]$ and $\mathrm{trans}_{G,k}(H|_j, \varrho_1, \ldots, \varrho_{\mathrm{mx}})$. Finally, for $j = 1$ the call $\mathrm{trans}_{H',k}(H|_1, r_{H_1}, \ldots, r_{H_{a-1}}, \varrho_a, \ldots, \varrho_{\mathrm{mx}})$ also fulfills $N_{H'} \cap N_{H|_1} \subseteq X_k$ with $H' = G \cup \bigcup_{i \in [a-1]} H_i$ and $H_i = \mathrm{trans}_{G,k}(H|_{i+1}, \varrho_1, \ldots, \varrho_{\mathrm{mx}})$ for every $i \in [a-1]$, since

$$
\begin{aligned}
N_{H'} \cap N_{H|_1} &\overset{\text{Lem. 5.11(1)}}{\subseteq} (N_G \cup \bigcup_{i \in [a-1]} N_{H|_{i+1}}) \cap N_{H|_1} \\
&= (N_G \cap N_{H|_1}) \cup (\bigcup_{i \in [a-1]} N_{H|_{i+1}} \cap N_{H|_1}) \\
&\overset{\text{I.H.}}{\subseteq} X_k \cup (\bigcup_{i \in [a-1]} N_{H|_{i+1}} \cap N_{H|_1}) \\
&\subseteq X_k
\end{aligned}
$$

    by the property that H is (only) variable-shared.

□

Example 5.12 (The refined construction on M_{doub}). Let M_{doub} and $M'_{\mathrm{doub}} = (F', \Delta, e', R')$ be the modular tree transducer and the macro tree transducer, respectively, of Example 5.4. Construction 5.10 constructs the macro tree transducer $M''_{\mathrm{doub}} = \mathfrak{C}_{5.10}(M'_{\mathrm{doub}}) = (F', \Delta, e', R'')$, which coincides with M''_{doub} of Example 5.9, with term graph rewrite rules and their construction depicted in Figure 19. □

Having modified the construction, we again need to show the validity of the construction in the sense that the computed translation remains the same.

Lemma 5.13 (Correctness of the refined construction). *Let*

$$M = (F, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$$

with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ *for some* $\mathrm{mx} \in \mathbb{N}$ *and, additionally, let* $M'' = \mathfrak{C}_{5.10}(M)$ *be the result of Construction 5.10 being applied to* M.

$$\tau_M = \tau_{M''}$$

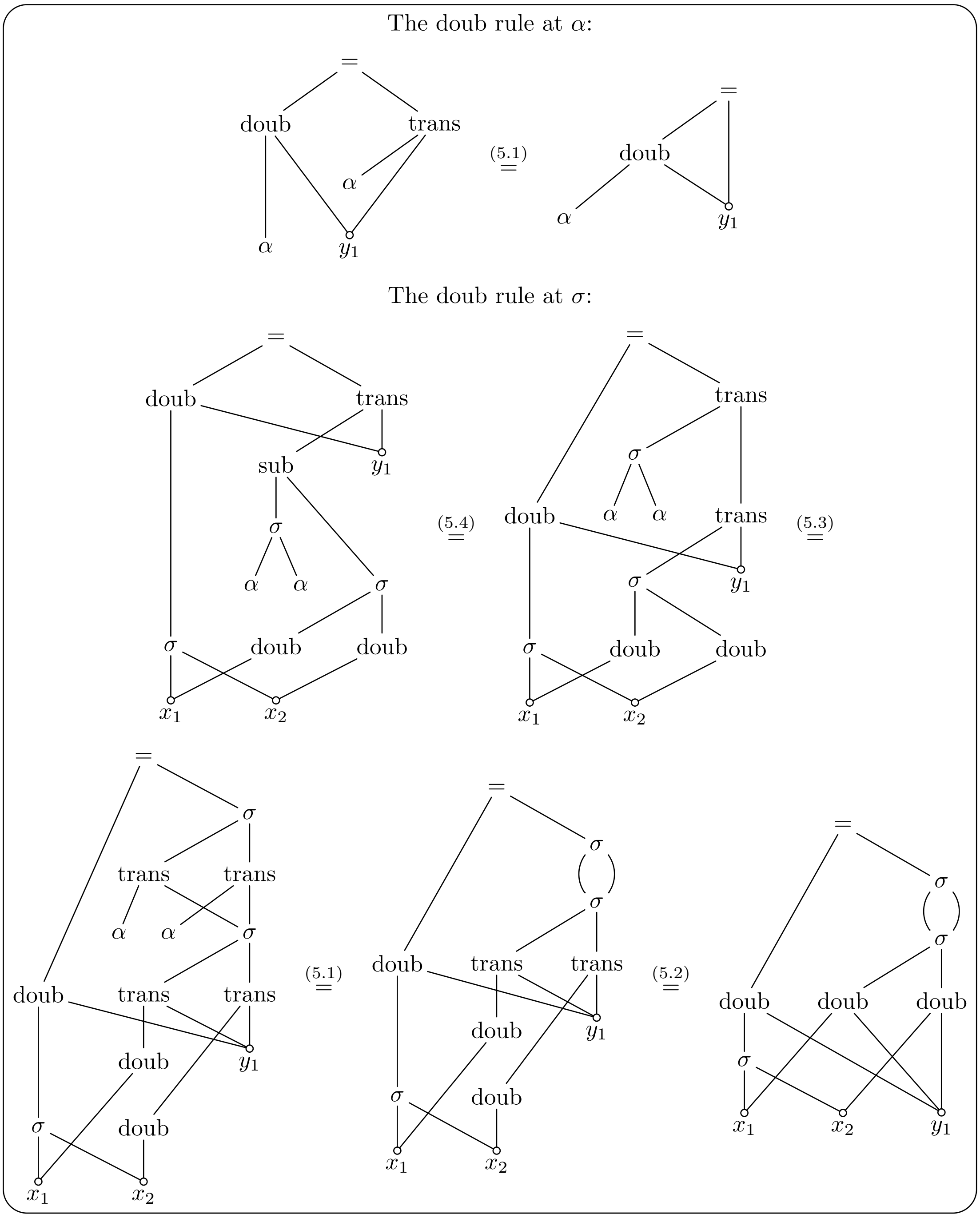

Figure 19: The term graph rewrite rules of M''_{doub} and their construction. Note $\alpha = \Pi_1$.

Proof. Let $M' = (F'', \Delta, e'', R') = \mathfrak{C}_{4.5}(M)$ and $M'' = (F'', \Delta, e'', R'')$. By Lemma 5.8, it is sufficient to show $R' = \{\, \text{term}(G) \mid G \in R'' \,\}$ in order to prove $\tau_{M''} = \tau_{M'}$. Additionally, $\tau_{M'} = \tau_M$ by Theorem 4.16, thus $\tau_M = \tau_{M''}$. Since $\text{card}(R') = \text{card}(R'')$, we just show $\{\, \text{term}(G) \mid G \in R'' \,\} \subseteq R'$ and conclude by the cardinality argument

$R' = \{\, \mathrm{term}(G) \mid G \in R'' \,\}$. Apparently, by Construction 4.5 and Construction 5.10 the left hand sides immediately coincide, thus we need to prove

$$\mathrm{term}(\mathrm{trans}_{G_Y,k}(G_{\mathrm{rhs}_M(f,\delta)}, y_1, \ldots, y_{\mathrm{mx}})) = \mathrm{trans}_k(\mathrm{rhs}_M(f,\delta), y_1, \ldots, y_{\mathrm{mx}})$$

for every $k \in \mathbb{N}$, $\delta \in \Delta^{(k)}$, $f \in m^{-1}(1)$ and $(f\ (\delta\ x_1\ \ldots\ x_k) = \mathrm{rhs}_M(f,\delta)) \in R$ with $G_Y = (Y_{\mathrm{mx}}, \emptyset, \emptyset)$. This will be done by using structural induction on $\mathrm{term}(H)$ to prove the following, more general statement for any non-rooted term graph $G \in \mathcal{TG}_{\mathrm{nr}}(\mathrm{RHS}_{\mathrm{MAC}}(F'', \Delta, X_k, Y_{\mathrm{mx}}))$ and $\varrho_1, \ldots, \varrho_{\mathrm{mx}} \in N_G$:

$$\mathrm{term}(\mathrm{trans}_{G,k}(H, \varrho_1, \ldots, \varrho_{\mathrm{mx}})) = \mathrm{trans}_k(\mathrm{term}(H), \mathrm{term}_G(\varrho_1), \ldots, \mathrm{term}_G(\varrho_{\mathrm{mx}})).$$

- **Induction base:**
  - Let $\alpha = l_H(r_H) \in \Delta^{(0)}$, then

$$\begin{aligned}
&\mathrm{term}(\mathrm{trans}_{G,k}(H, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))\\
\overset{(5.1)\ \&\ (5.3)}{=}\ &\begin{cases}\mathrm{term}((N_G, E_G, l_G, \varrho_j)) & \text{, if } \alpha = \Pi_j \text{ for some } j \in [\mathrm{mx}]\\ \alpha & \text{, otherwise}\end{cases}\\
=\ &\begin{cases}\mathrm{term}_G(\varrho_j) & \text{, if } \alpha = \Pi_j \text{ for some } j \in [\mathrm{mx}]\\ \mathrm{term}(H) & \text{, otherwise}\end{cases}\\
\overset{(4.1)\ \&\ (4.3)}{=}\ &\mathrm{trans}_k(\mathrm{term}(H), \mathrm{term}_G(\varrho_1), \ldots, \mathrm{term}_G(\varrho_{\mathrm{mx}})).
\end{aligned}$$

  - Let $f = l_H(r_H) \in m^{-1}(1)$, then

$$\begin{aligned}
&\mathrm{term}(\mathrm{trans}_{G,k}(H, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))\\
\overset{(5.2)}{=}\ &\mathrm{term}((N_{(G\cup H)}, E_{H'} \cup \{\, ((r_H, i+1), \varrho_i) \mid i \in [\mathrm{mx}] \,\}, l_{(G\cup H)}, r_H))\\
=\ &f\ \mathrm{term}(H|_1)\ \mathrm{term}_G(\varrho_1)\ \ldots\ \mathrm{term}_G(\varrho_{\mathrm{mx}})\\
\overset{(4.2)}{=}\ &\mathrm{trans}_k(f\ \mathrm{term}(H|_1), \mathrm{term}_G(\varrho_1), \ldots, \mathrm{term}_G(\varrho_{\mathrm{mx}}))\\
=\ &\mathrm{trans}_k(\mathrm{term}(H), \mathrm{term}_G(\varrho_1), \ldots, \mathrm{term}_G(\varrho_{\mathrm{mx}})).
\end{aligned}$$

- **Induction step:**
  - let $\sigma = l_H(r_H) \in \Delta^{(a)}$ with $a \in \mathbb{N}_+$, then the induction hypothesis is

$$\begin{aligned}
&\mathrm{term}(\mathrm{trans}_{G,k}(H|_i, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))\\
=\ &\mathrm{trans}_k(\mathrm{term}(H|_i), \mathrm{term}_G(\varrho_1), \ldots, \mathrm{term}_G(\varrho_{\mathrm{mx}}))
\end{aligned}$$

  for every $i \in [a]$ and we deduce

$$\begin{aligned}
&\mathrm{term}(\mathrm{trans}_{G,k}(H, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))\\
\overset{(5.3)}{=}\ &\mathrm{term}((N_{H'} \cup \{r_H\}, E_{H'} \cup \{\, ((r_H, i), r_{H_i}) \mid i \in [a] \,\}, l_{H'} \cup \{(r_H, \sigma)\}, r_H))
\end{aligned}$$

  where $H' = \bigcup_{i\in[a]} H_i$ with $H_i = \mathrm{trans}_{G,k}(H|_i, \varrho_1, \ldots, \varrho_{\mathrm{mx}})$ for every $i \in [a]$

$$\begin{aligned}
=\ &(\sigma\ \mathrm{term}(\mathrm{trans}_{G,k}(H|_1, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))\ \ldots\ \mathrm{term}(\mathrm{trans}_{G,k}(H|_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))\\
\overset{\mathrm{I.H.}}{=}\ &\sigma\ \mathrm{trans}_k(\mathrm{term}(H|_1), \mathrm{term}_G(\varrho_1), \ldots, \mathrm{term}_G(\varrho_{\mathrm{mx}}))\ \ldots\\
&\quad \mathrm{trans}_k(\mathrm{term}(H|_a), \mathrm{term}_G(\varrho_1), \ldots, \mathrm{term}_G(\varrho_{\mathrm{mx}}))\\
\overset{(4.3)}{=}\ &\mathrm{trans}_k(\mathrm{term}(H), \mathrm{term}_G(\varrho_1), \ldots, \mathrm{term}_G(\varrho_{\mathrm{mx}})).
\end{aligned}$$

– Finally let $\mathrm{sub}_a = l_H(r_H) \in m^{-1}(2)^{(a+1)}$ with $a \in \mathbb{N}$, then the induction hypothesis is again

$$\begin{aligned} & \mathrm{term}(\mathrm{trans}_{G,k}(H|_i, \varrho_1, \dots, \varrho_{\mathrm{mx}})) \\ = \ & \mathrm{trans}_k(\mathrm{term}(H|_i), \mathrm{term}_G(\varrho_1), \dots, \mathrm{term}_G(\varrho_{\mathrm{mx}})) \end{aligned}$$

and, especially, since $G' = (G \cup \bigcup_{i\in[a]} H_i) \in \mathcal{TG}_{\mathrm{nr}}(\mathrm{RHS}_{\mathrm{MAC}}(F', \Delta, X_k, Y_{\mathrm{mx}}))$ with $H_i = \mathrm{trans}_{G,k}(H|_{i+1}, \varrho_1, \dots, \varrho_{\mathrm{mx}})$ for every $i \in [a]$ and $r_{H_1}, \dots, r_{H_a} \in N_{G'}$, $\varrho_{a+1}, \dots, \varrho_{\mathrm{mx}} \in N_{G'}$, also

$$\begin{aligned} & \mathrm{term}(\mathrm{trans}_{G',k}(H|_1, r_{H_1}, \dots, r_{H_a}, \varrho_{a+1}, \dots, \varrho_{\mathrm{mx}})) \\ = \ & \mathrm{trans}_k(\mathrm{term}(H|_1), \mathrm{term}_{G'}(r_{H_1}), \dots, \mathrm{term}_{G'}(r_{H_a}), \\ & \qquad \mathrm{term}_{G'}(\varrho_{a+1}), \dots, \mathrm{term}_{G'}(\varrho_{\mathrm{mx}})). \end{aligned}$$

We gain:

$$\begin{aligned} & \mathrm{term}(\mathrm{trans}_{G,k}(H, \varrho_1, \dots, \varrho_{\mathrm{mx}})) \\ \overset{(5.4)}{=} \ & \mathrm{term}(\mathrm{trans}_{G',k}(H|_1, r_{H_1}, \dots, r_{H_a}, \varrho_{a+1}, \dots, \varrho_{\mathrm{mx}})) \\ \overset{\text{I.H.}}{=} \ & \mathrm{trans}_k(\mathrm{term}(H|_1), \mathrm{term}_{G'}(r_{H_1}), \dots, \mathrm{term}_{G'}(r_{H_a}), \\ & \qquad \mathrm{term}_{G'}(\varrho_{a+1}), \dots, \mathrm{term}_{G'}(\varrho_{\mathrm{mx}})) \\ = \ & \mathrm{trans}_k(\mathrm{term}(H|_1), \mathrm{term}(H_1), \dots, \mathrm{term}(H_a), \\ & \qquad \mathrm{term}_G(\varrho_{a+1}), \dots, \mathrm{term}_G(\varrho_{\mathrm{mx}})) \\ \overset{\text{I.H.}}{=} \ & \mathrm{trans}_k(\mathrm{term}(H|_1), \mathrm{trans}_k(\mathrm{term}(H|_2), \mathrm{term}_G(\varrho_1), \dots, \mathrm{term}_G(\varrho_{\mathrm{mx}})), \dots \\ & \qquad \mathrm{trans}_k(\mathrm{term}(H|_{a+1}), \mathrm{term}_G(\varrho_1), \dots, \mathrm{term}_G(\varrho_{\mathrm{mx}})), \\ & \qquad \mathrm{term}_G(\varrho_{a+1}), \dots, \mathrm{term}_G(\varrho_{\mathrm{mx}})) \\ \overset{(4.4)}{=} \ & \mathrm{trans}_k(\mathrm{term}(H), \mathrm{term}_G(\varrho_1), \dots, \mathrm{term}_G(\varrho_{\mathrm{mx}})). \end{aligned}$$

Thus the proof is complete. □

In this subsection we introduced the refinement of Construction 4.5, namely Construction 5.10, and showed that this construction is preserving the semantics. So far, we have not yet explored the applicability of term graphs in currently used functional programming languages. However, most functional programming languages support explicit sharing via especially designed constructs, such as `let`-constructs or `where`-clauses. Thus, we can effectively implement our modular tree transducers with term graph rewrite rules on stock hardware. An example will be provided that shows an implementation in Haskell (cf. [Tho99]).

Example 5.14 (M''_{doub} in Haskell). The macro tree transducer M''_{doub} of Example 5.9 can be implemented in the functional programming language Haskell (cf. [Tho99]) as follows:

```
data Tree = S Tree Tree | A

doub :: Tree -> Tree -> Tree
doub A          y1 = y1
doub (S x1 x2) y1 = let z = S (doub x1 y1) (doub x2 y1) in S z z
```

Here the term graph rewrite rules were implemented using a `let`-construct, however, we could also use `where`-clauses. □

In the next subsection we study the efficiency impact of the refinement and show our first major theorem concerning efficiency. We first extend the $\mathrm{trans}_{G,k}$-mapping to sentential form graphs in the obvious fashion (cf. Definition 4.7) and, additionally, restate the propositions of Lemma 4.9 in the setting of modular tree transducers with term graph rewrite rules.

Lemma 5.15 (Lemma 4.9 for modular tree transducers with term graph rewrite rules). *Given* $M = (F, m, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$ *with substitution variables* $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ *for some* $\mathrm{mx} \in \mathbb{N}$*, let* $M' = (F', \Delta, e', R') = \mathfrak{C}_{5.10}(M)$ *and* $G_\Pi = (\Pi, \emptyset, id_\Pi)$.

(a) Let $k \in \mathbb{N}$, $G \in \mathcal{TG}_{\mathrm{nr}}(\mathrm{RHS}_{\mathrm{MAC}}(F', \Delta, X_k, Y_{\mathrm{mx}}))$, $G' = \mathcal{TG}_{\mathrm{nr}}(\mathrm{SF}(M'))$ *and* $H' \in \mathrm{SFG}(M')$ *and*

$$H \in \{\, G_{\mathrm{rhs}} \mid \mathrm{rhs} \in \mathrm{RHS}_{m^{-1}(1), m^{-1}(2), \Delta}(X_k, \emptyset), N_{G_{\mathrm{rhs}}} \cap N_G \subseteq X_k \,\}.$$

Additionally, let $\psi_1 : N_H \longrightarrow N_{H'}$ *be a non-collapsing term graph homomorphism from* H *to* H' *and* $\psi_2 : N_G \longrightarrow N_{G'}$ *be a non-collapsing term graph homomorphism from* G *to* G'*, such that for every* $x \in X_k \cap N_H$: $\mathrm{term}_{H'}(\psi_1(x)) \in \mathcal{T}_\Delta$, $H'|_{\psi_1(x)} = G'|_{\psi_2(x)}$ *and* $\psi_1(x) = \psi_2(x)$ [61]*. Then for every* $\varrho_1, \ldots, \varrho_{\mathrm{mx}} \in N_G$ *there exists a non-collapsing term graph homomorphism* ψ *from* $\Gamma = \mathrm{trans}_{G,k}(H, \varrho_1, \ldots, \varrho_{\mathrm{mx}})$ *to* $\Gamma' = \mathrm{trans}_{G',0}(H', \psi_2(\varrho_1), \ldots, \psi_2(\varrho_{\mathrm{mx}}))$ *with* $\Gamma'|_{\psi(v)} = G'|_{\psi_2(v)}$ *and* $\psi(v) = \psi_2(v)$ *for every* $v \in (X_k \cup Y_{\mathrm{mx}}) \cap N_\Gamma$.

(b) For every $G \in \mathrm{SFG}(M)$ *with* $\mathrm{term}(G) \in \mathcal{T}_\Delta$: $\mathrm{trans}_{M,G_\Pi}(G, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = G$.

(c) Let $H, G, G' \in \mathrm{SFG}(M)$, $G'' = G \cup G'$ *and* $H = (N_{G''}, E_{G''}, l_{G''}, r_G)[n \rightsquigarrow r_{G'}]$ *for some* $n \in N_G$*. If for every* $\Gamma \in \mathcal{TG}_{\mathrm{nr}}(\mathrm{SF}(M'))$ *and* $\xi_1, \ldots, \xi_{\mathrm{mx}} \in N_\Gamma$

$$\mathrm{trans}_{\Gamma,0}(G', \xi_1, \ldots, \xi_{\mathrm{mx}}) = \mathrm{trans}_{\Gamma,0}(G|_n, \xi_1, \ldots, \xi_{\mathrm{mx}}),$$

then also

$$\mathrm{trans}_{G_\Pi,0}(H, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = \mathrm{trans}_{G_\Pi,0}(G, \Pi_1, \ldots, \Pi_{\mathrm{mx}}).$$

Proof. The enumeration refers to the one above.

(a) We prove the statement via structural induction on the term graph structure of H.]

- **Induction base:**

[61] Intuitively, ψ_1 models the grounding substitution θ_1 of Lemma 4.9(a) and ψ_2 models the substitution $\theta_1\theta_2$.

- Let $\alpha = l_H(r_H) \in \Delta^{(0)}$:

$$\begin{aligned} & \operatorname{trans}_{G',0}(H', \psi_2(\varrho_1), \ldots, \psi_2(\varrho_{\mathrm{mx}})) \\ \overset{(5.1)\ \&\ (5.3)}{=} & \begin{cases} (N_{G'}, E_{G'}, l_{G'}, \psi_2(\varrho_j)) & \text{, if } \alpha = \Pi_j \text{ for some } j \in [\mathrm{mx}] \\ (\{r_{H'}\}, \emptyset, \{(r_{H'}, \alpha)\}, r_{H'}) & \text{, otherwise} \end{cases} \end{aligned}$$

and

$$\begin{aligned} & \begin{cases} (N_G, E_G, l_G, \varrho_j) & \text{, if } \alpha = \Pi_j \text{ for some } j \in [\mathrm{mx}] \\ (\{r_H\}, \emptyset, \{(r_H, \alpha)\}, r_H) & \text{, otherwise} \end{cases} \\ \overset{(5.1)\ \&\ (5.3)}{=} & \operatorname{trans}_{G,k}(H, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \end{aligned}$$

Thus, obviously, in the first case $\psi = \psi_2$ (Γ' is a term subgraph of G'), while in the second case $\psi = \{(r_H, r_{H'})\}$. Both are non-collapsing (trivially or by assumption) and fulfill the required property concerning the variable nodes.

- Let $f = l_H(r_H) \in m^{-1}(1)$:

$$\begin{aligned} & \operatorname{trans}_{G',0}(H', \psi_2(\varrho_1), \ldots, \psi_2(\varrho_{\mathrm{mx}})) \\ \overset{(5.2)}{=} & (N_{(G' \cup H')}, E_{(G' \cup H')} \cup \{\, ((r_{H'}, i+1), \psi_2(\varrho_i)) \mid i \in [\mathrm{mx}] \,\}, l_{(G' \cup H')}, r_{H'}) \end{aligned}$$

and

$$\begin{aligned} & (N_{(G \cup H)}, E_{(G \cup H)} \cup \{\, ((r_H, i+1), \varrho_i) \mid i \in [\mathrm{mx}] \,\}, l_{(G \cup H)}, r_H) \\ \overset{(5.2)}{=} & \operatorname{trans}_{G,k}(H, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \end{aligned}$$

Thus $\psi = \psi_1 \cup \psi_2$ is the required non-collapsing term graph homomorphism. Since $N_G \cap N_H \subseteq X_k$ by Lemma 5.11, ψ fulfills the property on variables nodes.

- **Induction step:**
  - let $\sigma = l_H(r_H) \in \Delta^{(a)}$ with $a \in \mathbb{N}_+$, then the induction hypothesis assures the existence of non-collapsing term graph homomorphisms $\bar{\psi}_i$ from

$$\operatorname{trans}_{G,k}(H|_i, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \ \text{ to } \ \operatorname{trans}_{G',0}(H'|_i, \psi_2(\varrho_1), \ldots, \psi_2(\varrho_{\mathrm{mx}}))$$

for every $i \in [a]$. We compute

$$\begin{aligned} & \operatorname{trans}_{G',0}(H', \psi_2(\varrho_1), \ldots, \psi_2(\varrho_{\mathrm{mx}})) \\ \overset{(5.3)}{=} & (N_{\bar{H}'} \cup \{r_{H'}\}, E_{\bar{H}'} \cup \{\, ((r_{H'}, i), r_{\bar{H}'_i}) \mid i \in [a] \,\}, l_{\bar{H}'} \cup \{(r_{H'}, \sigma)\}, r_{H'}) \end{aligned}$$

where $\bar{H}' = \bigcup_{i \in [a]} \bar{H}'_i$ with $\bar{H}'_i = \operatorname{trans}_{G',0}(H'|_i, \psi_2(\varrho_1), \ldots, \psi_2(\varrho_{\mathrm{mx}}))$
for every $i \in [a]$
and

$$(N_{\bar{H}} \cup \{r_H\}, E_{\bar{H}} \cup \{\, ((r_H, i), r_{\bar{H}_i}) \mid i \in [a] \,\}, l_{\bar{H}} \cup \{(r_H, \sigma)\}, r_H)$$

where $\bar{H} = \bigcup_{i \in [a]} \bar{H}_i$ with $\bar{H}_i = \operatorname{trans}_{G,k}(H|_i, \varrho_1, \ldots, \varrho_{\mathrm{mx}})$ for every $i \in [a]$

$$\overset{(5.3)}{=} \operatorname{trans}_{G,k}(H, \varrho_1, \ldots, \varrho_{\mathrm{mx}}).$$

Thus, $\psi = \{(r_H, r_{H'})\} \cup \bigcup_{i\in[a]} \bar{\psi}_i$ is the required non-collapsing term graph homomorphism, which apparently possesses the demanded properties.

– Finally let $\mathrm{sub}_a = l_H(r_H) \in m^{-1}(2)^{(a+1)}$ with $a \in \mathbb{N}$, then the induction hypothesis assures the existence of non-collapsing term graph homomorphisms $\bar{\psi}_i$ from

$$\mathrm{trans}_{G,k}(H|_{i+1}, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \text{ to } \mathrm{trans}_{G',0}(H'|_{i+1}, \psi_2(\varrho_1), \ldots, \psi_2(\varrho_{\mathrm{mx}}))$$

for every $i \in [a]$. Especially, since $\bar{G} = (G \cup \bigcup_{i\in[a]} \bar{H}_i)$ is an element of $\mathcal{TG}_{\mathrm{nr}}(\mathrm{RHS}_{\mathrm{MAC}}(F', \Delta, X_k, Y_{\mathrm{mx}}))$ with $\bar{H}_i = \mathrm{trans}_{G,k}(H|_{i+1}, \varrho_1, \ldots, \varrho_{\mathrm{mx}})$ for every $i \in [a]$, $r_{\bar{H}_1}, \ldots, r_{\bar{H}_a}, \varrho_{a+1}, \ldots, \varrho_{\mathrm{mx}} \in N_{\bar{G}}$ and $\bar{G}' = (G' \cup \bigcup_{i\in[a]} \bar{H}'_i) \in \mathcal{TG}_{\mathrm{nr}}(\mathrm{SF}(M')$ with

$$\bar{H}'_i = \mathrm{trans}_{G',0}(H'|_{i+1}, \psi_2(\varrho_1), \ldots, \psi_2(\varrho_{\mathrm{mx}}))$$

for every $i \in [a]$, $r_{\bar{H}'_1}, \ldots, r_{\bar{H}'_a}, \psi_2(\varrho_{a+1}), \ldots, \psi_2(\varrho_{\mathrm{mx}}) \in N_{\bar{G}'}$, there is also a non-collapsing term graph homomorphism $\bar{\psi}$ from

$$\mathrm{trans}_{\bar{G},k}(H|_1, r_{\bar{H}_1}, \ldots, r_{\bar{H}_a}, \varrho_{a+1}, \ldots, \varrho_{\mathrm{mx}})$$

to

$$\mathrm{trans}_{\bar{G}',0}(H'|_1, r_{\bar{H}'_1}, \ldots, r_{\bar{H}'_a}, \psi_2(\varrho_{a+1}), \ldots, \psi_2(\varrho_{\mathrm{mx}})).$$

This is due to induction hypothesis, which assures the existence of a non-collapsing term graph homomorphism $\bar{\psi}$ from $\bar{G}$ to $\bar{G}'$, namely $\bar{\psi} = \psi_2 \cup \bigcup_{i\in[a]} \bar{\psi}_i$. Additionally, ψ_1 induces a non-collapsing term graph homomorphism from $H|_1$ to $H'|_1$, and, thus, the application of the induction hypothesis yields the required non-collapsing term graph homomorphism.

The other proofs are analogous to the ones for Lemma 4.9. □

5.2 Proof of efficiency non-deterioration

The general layout of the proof is very similar to the correctness proof found in the previous section. However, in the call-by-need case which is considered in the lemma to follow, we cannot arbitrarily rewrite any redex, rather the leftmost outermost redex is to reduced next. Thus, additional effort is required to assure that the leftmost outermost redex of any sentential form graph of M; being the original modular tree transducer; is "reproduced" (with context parameters) leftmost outermost in the translated (via trans) sentential form graph, if it is reproduced at all.

Lemma 5.16 (Correspondence on the redex level). *Let* $M = (F, m, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$ *with substitution variables* $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ *for some* $\mathrm{mx} \in \mathbb{N}$*, and let* $M'' = (F', \Delta, e', R'') = \mathfrak{C}_{5.10}(M)$*. Then for every* $t \in \mathcal{T}_\Delta$*, let* $\theta = \{ x \mapsto t \}$ *and, additionally,* $G_1 \in \mathrm{cl}_{\Rightarrow_M}(\{G_{e\theta}\})$*. Let* $p \in R_M(G_1)$ *be the leftmost outermost redex (p is the least element with respect to the lexicographic ordering) with* $f = l_{G_1}(r_{G_1|_p}) \in m^{-1}(1)$*, and, furthermore,* $G_\Pi = (\Pi, \emptyset, id_\Pi)$ *and* $P(G_1) = P'(G_1) \setminus P'(G'_1)$ *contains the occurrences of translations of* $G_1|_p$ *in* $\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}})$ *with the following definition of* $P'(G)$ *for every* $G \in \mathcal{TG}(\mathrm{SF}(M))$ *and* $G'_1 = (N_{G_1}, E_{G_1}, l', r_{G_1})$ *is essentially* G_1*, but the root* *of* $G_1|_p$ *is relabelled to another function symbol* $g \in m^{-1}(1)$ *with* $f \neq g$ [62], *i.e.* $l' = (l_{G_1} \setminus \{(r_{G_1|_p}, f)\}) \cup \{(r_{G_1|_p}, g)\}$ [63].

$$P'(G) = \{\, \pi \in R_{M''}(H) \mid H = \mathrm{trans}_{G_\Pi,0}(G, \Pi_1, \ldots, \Pi_{\mathrm{mx}}),\, l_H(r_{H|_\pi}) = f \,\}$$

1. *for every* $p_1, p_2 \in P(G_1)$

$$(\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}}))|_{p_1} = (\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}}))|_{p_2}$$

   and

2. *the least element of* $R_{M''}(\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}}))$ *is an element of* $P(G_1)$.

Proof. The first statement is immediate by the property $N_{G'} \subseteq N_{G_\Pi} \cup G_1$ with $G' = \mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}})$ proven in Lemma 5.11 [64].

We prove the second statement using structural induction over the sentential form graph G_1.

- **Induction base:**
  - $l_{G_1}(r_{G_1}) \in \Delta^{(0)}$ is not applicable.
  - Let $l_{G_1}(r_{G_1}) = f$, then $\mathrm{term}(G_1|_1) \in \mathcal{T}_\Delta$ and thus

    $$R_{M''}(\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}})) = \{\varepsilon\} = P(G_1),$$

    so the properties are trivially fulfilled. The case $l_{G_1}(r_{G_1}) \neq f$, but $l_{G_1}(r_{G_1}) \in m^{-1}(1)$ is not possible, because it contradicts the leftmost-outermost property of the redex at p.

- **Induction step:**
  - Let $l_{G_1}(r_{G_1}) \in m^{-1}(2)^{(k+1)}$ with $k \in \mathbb{N}$. Obviously in order for the leftmost-outermost redex to be at p, $p = 1.\bar{p}$ with $\bar{p} \in \mathrm{occ}(G_1|_1)$. Further note that G_1 is a term graph in S, where S is the following set:
    * $(f\ t) \in S$ with $t = \mathrm{term}(G_1|_{p.1})$,
    * if $u \in S$, then $(g\ u\ u_1\ \ldots\ u_i) \in S$ for every $i \in \mathbb{N}$, $g \in m^{-1}(2)^{(i+1)}$ and $u_1, \ldots, u_i \in \mathrm{SF}(M)$, and
    * all elements of S can be obtained by a finite number of applications of the above rules.

    Thus, we perform structural induction on the term graph structure induced by S to prove that $\varepsilon \in R_{M''}(\mathrm{trans}_{G,0}(H, \xi_1, \ldots, \xi_{\mathrm{mx}}))$ and $\varepsilon \in P(H)$ for every term graph $G \in \mathcal{TG}_{\mathrm{nr}}(\mathrm{SF}(M'))$ with $\xi_1, \ldots, \xi_{\mathrm{mx}} \in N_G$ and $H \in \mathcal{TG}(S)$ being a subgraph of G_1.

[62] If no other function symbol with those properties exists, then add a new one with dummy rules.

[63] Roughly speaking, we first mark every call to the function symbol f in $\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}})$ and then subtract those calls that do not correspond to the redex.

[64] Since the relabelled node $r_{G_1|_p}$ exists at most once in G' by the stated property, all $\bar{p} \in P(G_1)$ must actually point to this node. There may, however, exist different paths to that particular node.

- ∗ **Induction base:** Let $l_H(r_H) = f$, then $\mathrm{term}(H|_1) = \mathrm{term}(G_1|_{p.1})$ and, thus,
$$\varepsilon \in R_{M''}(\mathrm{trans}_{G,0}(H, \xi_1, \dots, \xi_{\mathrm{mx}}))$$
and also $\varepsilon \in R_{M''}(\mathrm{trans}_{G,0}(H, \Pi_1, \dots, \Pi_{\mathrm{mx}}))$, hence $\varepsilon \in P(H)$.
- ∗ **Induction step:** Let $l_H(r_H) \in m^{-1}(2)^{(k+1)}$ with $k \in \mathbb{N}$. Obviously, in order for the leftmost-outermost redex to be at p, $p = 1.\bar{p}$ with $\bar{p} \in \mathrm{occ}(H|_1)$. This yields
$$\begin{aligned} & \mathrm{trans}_{G,0}(H, \xi_1, \dots, \xi_{\mathrm{mx}}) \\ \overset{(5.4)}{=} \; & \mathrm{trans}_{G',0}(H|_1, r_{G'_1}, \dots, r_{G'_k}, \xi_{k+1}, \dots, \xi_{\mathrm{mx}}) \end{aligned}$$
where $G'_i = \mathrm{trans}_{M,G}(H|_{i+1}, \xi_1, \dots, \eta_{\mathrm{mx}})$ and $G' = G \cup \bigcup_{i \in [k]} G'_i$. Since $r_{G'_1}, \dots, r_{G'_k}, \xi_{k+1}, \dots, \xi_{\mathrm{mx}} \in N_{G'}$ and $G' \in \mathcal{TG}_{\mathrm{nr}}(\mathrm{SF}(M'))$, by induction hypothesis $\varepsilon \in R_{M''}(\mathrm{trans}_{G,0}(H|_1, r_{G'_1}, \dots, r_{G'_k}, \xi_{k+1}, \dots, \xi_{\mathrm{mx}}))$, hence also $\varepsilon \in R_{M''}(\mathrm{trans}_{G,0}(H, \xi_1, \dots, \xi_{\mathrm{mx}}))$ and $\varepsilon \in P(H)$.

Consequently, we instantiate the proven statement with $\xi_i = \Pi_i$ for every $i \in [\mathrm{mx}]$ and $H = G_1$, $G = G_\Pi$ to gain $\varepsilon \in P(G_1)$ and $\varepsilon \in R_{M''}(\mathrm{trans}_{G,0}(G_1, \Pi_1, \dots, \Pi_{\mathrm{mx}}))$.

- Let $l_{G_1}(r_{G_1}) \in \Delta^{(k)}$ with $k \in \mathbb{N}_+$, then there exists $j \in [k]$ with $p = j.\bar{p}$ where $\bar{p} \in \mathrm{occ}(G_1|_j)$. Note that $G_1|_1, \dots, G_1|_{j-1}$ are necessarily redex-free [65]. Thus,
$$\begin{aligned} & \mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \dots, \Pi_{\mathrm{mx}}) \\ \overset{(5.3)}{=} \; & (N_H \cup \{r_{G_1}\}, E_H \cup \{\, ((r_{G_1}, i), r_{H_i}) \mid i \in [k] \,\}, l_H \cup \{(r_{G_1}, l_{G_1}(r_{G_1}))\}, r_{G_1}) \end{aligned}$$
where $H = \bigcup_{i \in [k]} H_i$ with $H_i = \mathrm{trans}_{G_\Pi,0}(G_1|_i, \Pi_1, \dots, \Pi_{\mathrm{mx}})$ for every $i \in [k]$. Obviously, $H_1, \dots, H_{j-1}$ are redex-free by Lemma 5.15(b), since $G_1|_1, \dots, G_1|_{j-1}$ are redex-free. By induction hypothesis the leftmost-outermost redex of
$$R_{M''}(\mathrm{trans}_{G_\Pi,0}(G_1|_j, \Pi_1, \dots, \Pi_{\mathrm{mx}}))$$
is at $\bar{\pi} \in P(G_1|_j)$ and, consequently, $\pi = j.\bar{\pi}$ is the leftmost-outermost redex of
$$R_{M''}(\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \dots, \Pi_{\mathrm{mx}}))$$
and $\pi \in P(G_1)$.

This completes the proof of the proposition. □

Lemma 5.17 (Strong correspondence on sentential forms). *Let*
$$M = (F, m, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$$
with substitution variables $\Pi = \{\Pi_1, \dots, \Pi_{\mathrm{mx}}\}$ *for some* $\mathrm{mx} \in \mathbb{N}$ *and let* $M'' = \mathfrak{C}_{5.10}(M) = (F', \Delta, e', R'')$. *Then for every* $t \in \mathcal{T}_\Delta$, *let* $\theta = \{\, x \mapsto t \,\}$ *and we claim*
$$\mathrm{trans}_{G_\Pi,0}(\cdot, \Pi_1, \dots, \Pi_{\mathrm{mx}}) : \mathrm{cl}_{\Rightarrow_M}(\{G_{e\theta}\}) \longrightarrow \mathrm{cl}_{\Rightarrow_{M''}}(\{G_{e'\theta}\})$$
is a surjective mapping with $G_\Pi = (\Pi, \emptyset, id_\Pi)$, *where we require* $\Pi \cap N_{\bar{G}} = \emptyset$ *for every* $\bar{G} \in \mathrm{cl}_{\Rightarrow_M}(\{G_{e\theta}\})$.

[65] else the leftmost-outermost redex is not at p

Proof. First we prove the validity of the typing specified above. The proof of the surjectivity is presented at end. We apply the principle of natural induction over the length of the derivation using $\Rrightarrow_M$ in order to prove the claim

$$\mathrm{trans}_{G_\Pi,0}(H, \Pi_1, \dots, \Pi_{\mathrm{mx}}) \in \mathrm{cl}_{\Rrightarrow_{M''}}(\{G_{e'\theta}\}) \tag{**}$$

for every $H \in \mathrm{cl}_{\Rrightarrow_M}(\{G_{e\theta}\})$, thus for some unique $n \in \mathbb{N} : G_{e\theta} \Rrightarrow_M^n H$. Alongside we prove that the $\mathrm{trans}_{G_\Pi,0}$-mapping is well-defined.

- **Induction base:** Let $n = 0$, hence $H = G_{e\theta}$. We show Equation (**) for $G_{e\theta}$, where the modified $\mathrm{trans}_{G_\Pi,0}$-mapping on sentential forms is well-defined by Lemma 5.11.

$$\begin{aligned} & \mathrm{trans}_{G_\Pi,0}(G_{e\theta}, \Pi_1, \dots, \Pi_{\mathrm{mx}}) \\ \overset{(5.2)}{=} \; & (N_{G_{e\theta}} \cup \Pi, E_{G_{e\theta}} \cup \{\, ((r_{G_{e\theta}}, i+1), \Pi_i) \mid i \in [\mathrm{mx}] \,\}, l_{G_{e\theta}} \cup id_\Pi, r_{G_{e\theta}}) \\ \overset{\text{Cons. 5.10}}{=} \; & G_{e'\theta}, \end{aligned}$$

  which is trivially an element of $\mathrm{cl}_{\Rrightarrow_{M''}}(\{G_{e'\theta}\})$. Note that all unions are in fact disjoint unions by the assumption that Π and $N_{G_{e\theta}}$ are disjoint.

- **Induction step:** By induction hypothesis assume $G_{e\theta} \Rrightarrow_M^n G_1 \Rrightarrow_M G_2$ with $G_1, G_2 \in \mathrm{cl}_{\Rrightarrow_M}(\{G_{e\theta}\})$ for some $n \in \mathbb{N}$ and

$$\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \dots, \Pi_{\mathrm{mx}}) \in \mathrm{cl}_{\Rrightarrow_{M''}}(\{G_{e'\theta}\})$$

  is well-defined. Consequently there exists a least (with respect to the lexicographic ordering) element $p \in R_M(G_1)$ with $\overline{r_{G_1} n_1}^p$, $k, a \in \mathbb{N}$, $l_{G_1}(n_1) = f \in F^{(a+1)}$ and $l_{G_1}(E_{G_1}(n_1, 1)) = \delta \in \Delta^{(k)}$. G_2 is constructed according to Definition 3.16 with $n_2 \in N_{G_2}$ such that $\overline{r_{G_2} n_2}^p$. We perform the following case distinction on the leftmost-outermost redex at p.

  1. Let $f \in m^{-1}(2)$. According to Lemma 5.15(c), it is sufficient to show the local equality

$$\mathrm{trans}_{G',0}(G_1|_p, \xi_1, \dots, \xi_{\mathrm{mx}}) = \mathrm{trans}_{G',0}(G_2|_p, \xi_1, \dots, \xi_{\mathrm{mx}})$$

     for every term graph $G' \in \mathcal{TG}_{\mathrm{nr}}(\mathrm{SF}(M''))$ and $\xi_1, \dots, \xi_{\mathrm{mx}} \in N_{G'}$ to prove

$$\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \dots, \Pi_{\mathrm{mx}}) = \mathrm{trans}_{G_\Pi,0}(G_2, \Pi_1, \dots, \Pi_{\mathrm{mx}}).$$

     We show this statement graphically in Figures 20 and 21, which correspond to our subcase distinction.

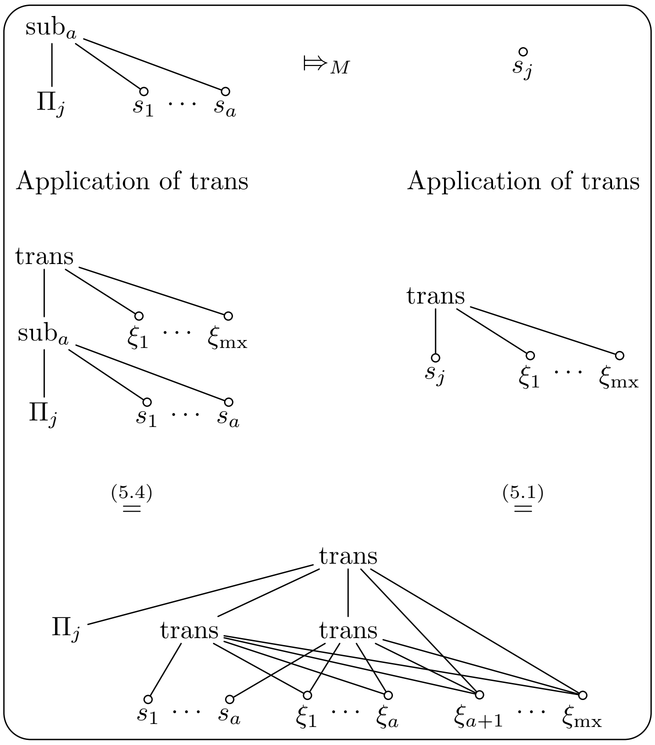

Figure 20: The local equality for the case $\delta = \Pi_j \in \Pi$ for some $j \in [a]$.

– Let $\delta = \Pi_j$ for some $j \in [a]$.

$$\begin{aligned}
&\mathrm{trans}_{G',0}(G_1|_p, \xi_1, \ldots, \xi_{\mathrm{mx}})\\
\stackrel{(5.4)}{=}\;&\mathrm{trans}_{G'',0}(G_1|_{p.1}, r_{H_1}, \ldots, r_{H_a}, \xi_{a+1}, \ldots, \xi_{\mathrm{mx}})\\
&\text{where } H_i = \mathrm{trans}_{G',0}(G_1|_{p.i+1}, \xi_1, \ldots, \xi_{\mathrm{mx}}) \text{ for each } i \in [a]\\
&G'' = G' \cup \bigcup_{i \in [a]} H_i\\
\stackrel{(5.1)}{=}\;&(N_{G''}, E_{G''}, l_{G''}, r_{H_j})\\
&\text{where } H_i = \mathrm{trans}_{G',0}(G_1|_{p.i+1}, \xi_1, \ldots, \xi_{\mathrm{mx}}) \text{ for each } i \in [a]\\
&G'' = G' \cup \bigcup_{i \in [a]} H_i\\
\stackrel{\text{G.C.}}{=}\;&\mathrm{trans}_{G',0}(G_1|_{p.j+1}, \xi_1, \ldots, \xi_{\mathrm{mx}})\\
\stackrel{\text{Def. 3.16}}{=}\;&\mathrm{trans}_{G',0}(G_2|_p, \xi_1, \ldots, \xi_{\mathrm{mx}}),
\end{aligned}$$

since $n_2 = E_{G_1}(n_1, j+1)$ and $G_2 = G_1[n_1 \rightsquigarrow n_2]$.

– Let $\delta \in \Delta \setminus \{\Pi_1, \ldots, \Pi_a\}$.

$$\begin{aligned}
& \mathrm{trans}_{G',0}(G_1|_p, \xi_1, \ldots, \xi_{\mathrm{mx}}) \\
\overset{(5.4)}{=}\ & \mathrm{trans}_{G'',0}(G_1|_{p.1}, r_{H_1}, \ldots, r_{H_a}, \xi_{a+1}, \ldots, \xi_{\mathrm{mx}})
\end{aligned}$$

where $H_i = \mathrm{trans}_{G',0}(G_1|_{p.i+1}, \xi_1, \ldots, \xi_{\mathrm{mx}})$ for each $i \in [a]$,

$$G'' = G' \cup \bigcup_{i\in[a]} H_i$$

$$\begin{aligned}
\overset{(5.3)}{=}\ & (N_{H'} \cup \{E_{G_1}(n_1, 1)\}, E_{H'} \cup \{\, ((E_{G_1}(n_1, 1), i), r_{H'_i}) \mid i \in [k] \,\}, \\
& \qquad l_{H'} \cup \{(E_{G_1}(n_1, 1), \delta)\}, E_{G_1}(n_1, 1))
\end{aligned}$$

where $H_i = \mathrm{trans}_{G',0}(G_1|_{p.i+1}, \xi_1, \ldots, \xi_{\mathrm{mx}})$ for every $i \in [a]$,

$H'_j = \mathrm{trans}_{G'',0}(G_1|_{p.1.j}, r_{H_1}, \ldots, r_{H_a}, \xi_{a+1}, \ldots, \xi_{\mathrm{mx}})$ for all $j \in [k]$,

$G'' = G' \cup \bigcup_{i\in[a]} H_i$ and $H' = \bigcup_{j\in[k]} H'_j$

By the definition of the call-need derivation relation (Definition 3.16) and the restriction on right hand sides of a substitution module (Definition 4.1), we conclude the following equivalences in the sentential form graphs.

$$G_1|_{p.1.j} = G_2|_{p.j.1} \quad \text{and} \quad G_1|_{p.i+1} = G_2|_{p.j.i+1} \quad \text{for every } j \in [k], i \in [a]$$

Using the second equivalence, we deduce for every $j \in [k]$ and $i \in [a]$ the facts $H_i = J'_{j,i}$ and thereby also $G'' = J'_j$. Moreover using the first concluded equivalence, we derive the statements $J_j = H'_j$ for every $j \in [k]$ and consequently $J = H'$. Summing up, all the previously mentioned term graphs are well-defined by induction hypothesis and the following equalities. We continue

$$\begin{aligned}
\overset{\text{Def. 3.16}}{=}\ & (N_J \cup \{r_{G_2|_p}\}, E_J \cup \{\, ((r_{G_2|_p}, i), r_{J_i}) \mid i \in [k] \,\}, \\
& \qquad l_J \cup \{(r_{G_2|_p}, \delta)\}, r_{G_2|_p})
\end{aligned}$$

where $J'_{j,i} = \mathrm{trans}_{G',0}(G_2|_{p.j.i+1}, \xi_1, \ldots, \xi_{\mathrm{mx}})$ for each $j \in [k], i \in [a]$

$J_j = \mathrm{trans}_{J'_j,0}(G_2|_{p.j.1}, r_{J'_{j,1}}, \ldots, r_{J'_{j,a}}, \xi_{a+1}, \ldots, \xi_{\mathrm{mx}})$ for all $j \in [k]$,

$J'_j = G' \cup \bigcup_{i\in[a]} J'_{j,i}$ for every $j \in [k]$ and $J = \bigcup_{j\in[k]} J_j$

$$\begin{aligned}
\overset{(5.4)}{=}\ & (N_J \cup \{r_{G_2|_p}\}, E_J \cup \{\, ((r_{G_2|_p}, i), r_{J_i}) \mid i \in [k] \,\}, \\
& \qquad l_J \cup \{(r_{G_2|_p}, \delta)\}, r_{G_2|_p})
\end{aligned}$$

where $J_j = \mathrm{trans}_{G',0}(G_2|_{p.j}, \xi_1, \ldots, \xi_{\mathrm{mx}})$ for each $j \in [k]$,

$$J = \bigcup_{j\in[k]} J_j$$

$$\overset{(5.3)}{=}\ \mathrm{trans}_{G',0}(G_2|_p, \xi_1, \ldots, \xi_{\mathrm{mx}}).$$

In both cases we could show local equivalence, which allows us to claim

$$\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = \mathrm{trans}_{G_\Pi,0}(G_2, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \in \mathrm{cl}_{\Rightarrow_{M''}}(\{G_{e'\theta}\}),$$

i.e. fulfills Equation (**), by Lemma 5.15(c) and, furthermore, we have shown that $\mathrm{trans}_{G_\Pi,0}(G_2, \Pi_1, \ldots, \Pi_{\mathrm{mx}})$ is well-defined given that the sentential form graph $\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}})$ is, which completes this part of the proof.

2. Let $f \in m^{-1}(1)$. Roughly speaking we try to prove the commutation of the following diagram:

$$\begin{tikzcd} G_1 \arrow[r, "\mathrm{trans}"] \arrow[d, Rightarrow, "M"] & \mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \arrow[d, Rightarrow, "M''"] \\ G_2 \arrow[r, "\mathrm{trans}"] & \mathrm{trans}_{G_\Pi,0}(G_2, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \end{tikzcd}$$

By Lemma 5.16 the least element $p' \in R_{M''}(\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}}))$ (i.e. a leftmost outermost redex occurrence) is the occurrence of the redex corresponding to $G_1|_p$. Note, however, that this redex might be shared. First we fix some additional shorthands, namely

$$H_1 = \mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \quad \text{and} \quad H_2 = \mathrm{trans}_{G_\Pi,0}(G_2, \Pi_1, \ldots, \Pi_{\mathrm{mx}}).$$

So we need to prove $H_1 \mapsto_{M''} H_2$. The derivation step from H_1 to H_2 apparently should use the redex at p'. The situation is displayed in Figure 22, which also illustrates the relationships between the many introduced term graphs. Consequently, there exists a non-collapsing term graph homomorphism $\psi_1 : N_{G_{\rho,\mathrm{lhs}}} \longrightarrow N_{G_1}$ from $G_{\rho,\mathrm{lhs}}$ to $G_1|_p$ [66] with $\rho = (f\ (\delta \ldots) = \ldots) \in R$. Moreover, by Definition 3.8 there also exists a non-collapsing term graph homomorphism $\psi_2 : N_{G_{\rho,\mathrm{rhs}}} \longrightarrow N_{G_2}$ from $G_{\rho,\mathrm{rhs}}$ to $G_2|_p$. By $p' \in R_{M''}(H_1)$ and again Definition 3.8, there exists a non-collapsing term graph homomorphism $\psi_3 : N_{G_{\rho',\mathrm{lhs}}} \longrightarrow N_{H_1}$ from $G_{\rho',\mathrm{lhs}}$ to $H_1|_{p'}$ with $G_{\rho'} \in R''$ and $\mathrm{term}(G_{\rho'}) = (f\ (\delta \ldots) \ldots = \ldots)$. Furthermore, there exists a term graph isomorphism ϖ_G from $G_1[\psi_1(r_{G_{\rho,\mathrm{lhs}}}) \rightsquigarrow z]$ to $G_2[\psi_2(r_{G_{\rho,\mathrm{rhs}}}) \rightsquigarrow z]$, where $z \notin N_{G_1} \cup N_{G_2} \cup N_{H_1} \cup N_{H_2}$.

Due to the isomorphism ϖ_G and the application of the (deterministic) trans-mapping, we gain an isomorphism ϖ_H from $H_1[\psi_3(r_{G_{\rho',\mathrm{lhs}}}) \rightsquigarrow z]$ to $H_2[n \rightsquigarrow z]$ with $n \in N_{H_2}$ and $\overline{r_{H_2} n}^{p'}$ [67].

Let $\xi_i = \psi_3(y_i)$ (alternatively $\xi_i = r_{H_1|_{p'.i+1}}$) for every $i \in [\mathrm{mx}]$ such that $\xi_i \in N_G$ for some $G \in \mathcal{TG}_{\mathrm{nr}}(\mathrm{SF}(M''))$. Intuitively, the translation via the trans-mapping approaches at n_1 with the context $\xi_1, \ldots, \xi_{\mathrm{mx}}$ and intermediate term graph G, i.e. a call

$$\mathrm{trans}_{G,0}(G_1|_{n_1}, \xi_1, \ldots, \xi_{\mathrm{mx}})$$

occurs during the computation of $\mathrm{trans}_{G_\Pi,0}(G_1, \Pi_1, \ldots, \Pi_{\mathrm{mx}})$.

According to Lemma 5.15(a) there is a non-collapsing term graph homomorphism ψ_4 from

$$\mathrm{trans}_{G_{\rho',\mathrm{lhs}},k}(G_{\rho,\mathrm{rhs}}, y_1, \ldots, y_{\mathrm{mx}}) \overset{\text{G.C. \& Cons. 5.10}}{=} G_{\rho',\mathrm{rhs}}$$

[66] exists due to $p \in R_M(G_1)$

[67] since for each prefix $p_<$ of p we have $\overline{r_{G_1} w}^{(p_<)}$ implies $\overline{r_{G_2} w}^{(p_<)}$ (by Definition 3.16 and ϖ_G), thus, for each prefix $p'_<$ of p' we also have $\overline{r_{H_1} w}^{(p'_<)}$ implies $\overline{r_{H_2} w}^{(p'_<)}$

to

$$\mathrm{trans}_{G,0}(G_2|_p, \xi_1, \ldots, \xi_{\mathrm{mx}}) = H_2|_{p'},$$

since ψ_2 and ψ_3 constitute the required non-collapsing term graph homomorphisms [68].

Altogether, we have a term graph isomorphism ϖ_H and two non-collapsing term graph homomorphisms ψ_3 and ψ_4 with $\psi_3(n) = \psi_4(n)$ for every $n \in N_{G_{p',\mathrm{lhs}}} \cap N_{G_{p',\mathrm{rhs}}}$ by Lemma 5.15(a). Using Definition 3.16 we can conclude $H_1 \Mapsto_{M''} H_2$.

Thus, the property is proven and consequently also $H_2 \in \mathrm{cl}_{\Mapsto_{M''}}(\{G_{e'\theta}\})$. Additionally it is also well-defined, since trans on right hand sides is well-defined by Lemma 5.11.

This completes the proof that the mapping is indeed well-typed and well-defined, however, we still have to prove that the such defined mapping is surjective. We will prove this property via induction on the length of the derivation $G_{e'\theta} \Mapsto^n_{M''} G$ with $n \in \mathbb{N}$ and $G \in \mathrm{cl}_{\Mapsto_{M''}}(\{G_{e'\theta}\})$.

- **Induction base:** Let $n = 0$, then $G = G_{e'\theta}$. However, we have already evidenced $\mathrm{trans}_{G_\Pi,0}(G_{e\theta}, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = G$ with $G_\Pi = (\Pi, \emptyset, id_\Pi)$, hence G is the image of $G_{e\theta}$.

- **Induction step:** Assume that every $G' \in \mathrm{cl}_{\Mapsto_{M''}}(\{G_{e'\theta}\})$ with $G_{e'\theta} \Mapsto^n_{M''} G' \Mapsto_{M''} G$ is the image of some element of $\mathrm{cl}_{\Mapsto_M}(\{G_{e\theta}\})$ by induction hypothesis. Let

$$S = \{\, H \in \mathrm{cl}_{\Mapsto_M}(\{G_{e\theta}\}) \mid \mathrm{trans}_{G_\Pi,0}(H, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = G' \,\}$$

and $H' \in S$ be the maximal element of S with respect to $\Mapsto^*_M$. Since $\Mapsto^*_M$ is a total order on $\mathrm{cl}_{\Mapsto_M}(\{G_{e\theta}\})$ and S is non-empty by induction hypothesis, H' is guaranteed to exist. $H' \notin \mathcal{T}_\Delta$, else $H' \overset{\text{Lem. 5.15(b)}}{=} \mathrm{trans}_{G_\Pi,0}(H', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = G' \in \mathcal{T}_\Delta$ which contradicts $G' \Mapsto_{M''} G$. Consequently, there exists $H'' \in \mathrm{cl}_{\Mapsto_M}(\{G_{e\theta}\})$ with $H' \Mapsto_M H''$. Let $p \in R_M(H')$ be the position of the redex that was used according to Definition 3.16 to construct H''. We perform a case distinction on $f = l_{H'}(r_{H'|_p})$.

  - $f \in m^{-1}(2)$, then

$$\mathrm{trans}_{G_\Pi,0}(H'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = \mathrm{trans}_{G_\Pi,0}(H', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = G',$$

    which was proven in the previous part of the proof. This, however, contradicts the maximality of H', since $H' \Mapsto_M H''$, hence this case is not possible.

  - $f \in m^{-1}(1)$, then

$$G' = \mathrm{trans}_{G_\Pi,0}(H', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \Mapsto_{M''} \mathrm{trans}_{G_\Pi,0}(H'', \Pi_1, \ldots, \Pi_{\mathrm{mx}}) = G,$$

    by the previous part of the proof and the fact that $\Mapsto_{M''}$ is deterministic. Consequently, G is the image of H''.

Thus, the mapping is surjective, which completes the proof. □

Example 5.18 (Illustrating this correspondence). In Figure 23, we computed the correspondence for the modular tree transducer M_{doub} of Example 5.4 and the macro tree transducer with term graph rewrite rules M''_{doub} of Example 5.9. □

Having established and illustrated the close correspondence on sentential form graphs, we are ready to state the main theorem of this section. Informally this theorem assures that the refined construction always yields a macro tree transducer which is at least as efficient as the 2-modular tree transducer which was the input of the construction. Thus according to our measure of efficiency the application of the refined construction is safe, i.e. we can always apply it.

Theorem 5.19 (Efficiency non-deterioration). *Let*

$$M = (F, m, \Delta, e, R) \in \text{ModTT}(\text{TOP}, \text{SUB})$$

and let $M'' = (F', \Delta, e', R'') = \mathfrak{C}_{5.10}(M)$ *be the result of Construction 5.10 applied to* M*. Then for every* $t \in \mathcal{T}_\Delta : \text{steps}_{\Rightarrow_{M''}}(t) \leq \text{steps}_{\Rightarrow_M}(t)$*, thus,* M'' *is at least as efficient as* M*.*

Proof. Note that

$$\text{steps}_{\Rightarrow_M}(t) = \text{card}(\text{cl}_{\Rightarrow_M}(\{G_{e\{x \mapsto t\}}\})) - 1$$

and

$$\text{steps}_{\Rightarrow_{M''}}(t) = \text{card}(\text{cl}_{\Rightarrow_{M''}}(\{G_{e'\{x \mapsto t\}}\})) - 1.$$

In Lemma 5.17 we showed that there exists a surjective mapping between those two closed sets, hence

$$\text{card}(\text{cl}_{\Rightarrow_M}(\{G_{e\{x \mapsto t\}}\})) \geq \text{card}(\text{cl}_{\Rightarrow_{M''}}(\{G_{e'\{x \mapsto t\}}\})),$$

which immediately establishes the proposition. □

[68]The property $(G_2|_p)|_{\psi_2(x)} = G|_{\psi_3(x)}$ and $\text{term}_{G_2|_p}(\psi_2(x))$ for every $x \in X_k \cap N_{G_{\rho,\text{rhs}}}$ is easily verified.

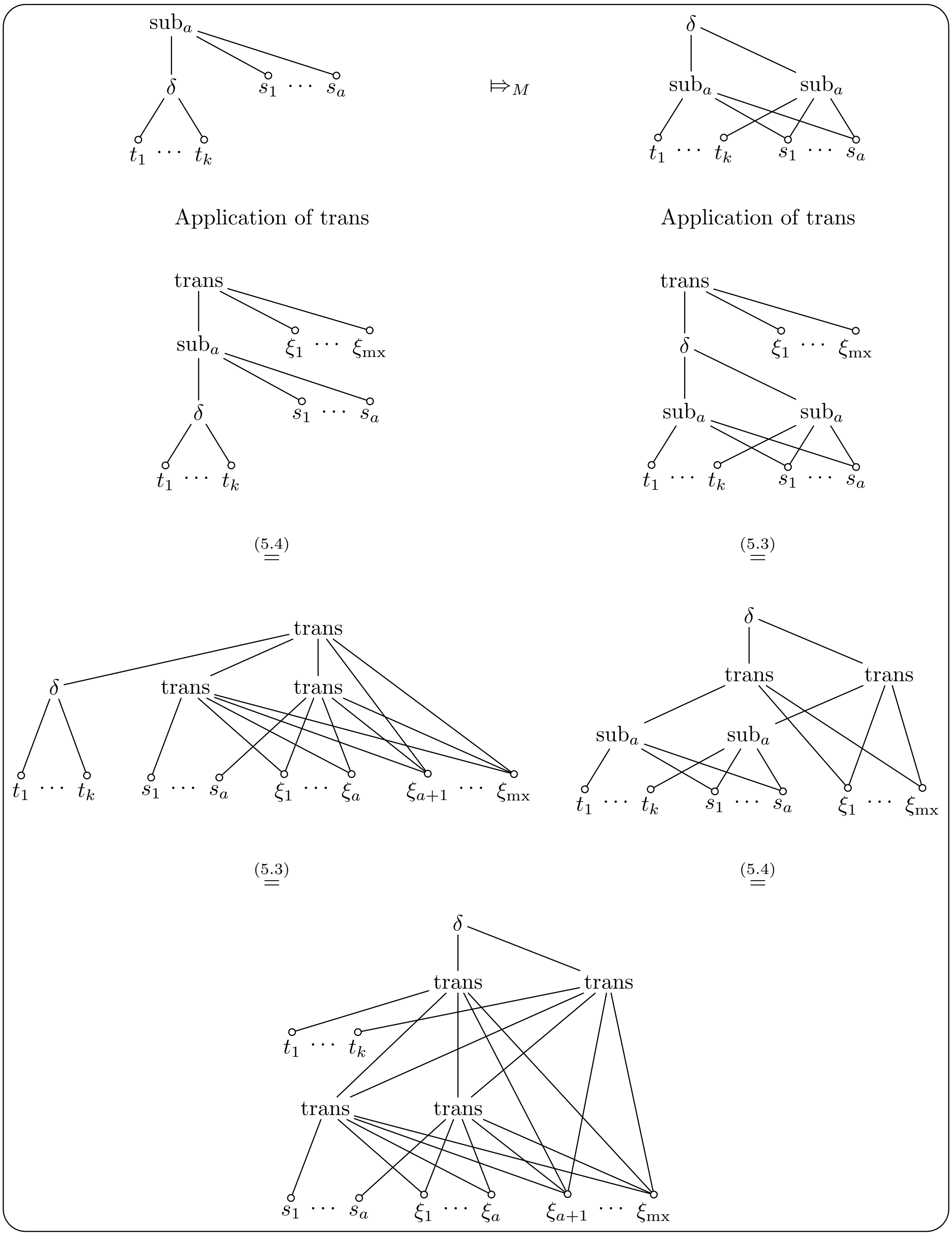

Figure 21: The local equality for the case $\delta \in \Delta \setminus \{\Pi_1, \ldots, \Pi_a\}$.

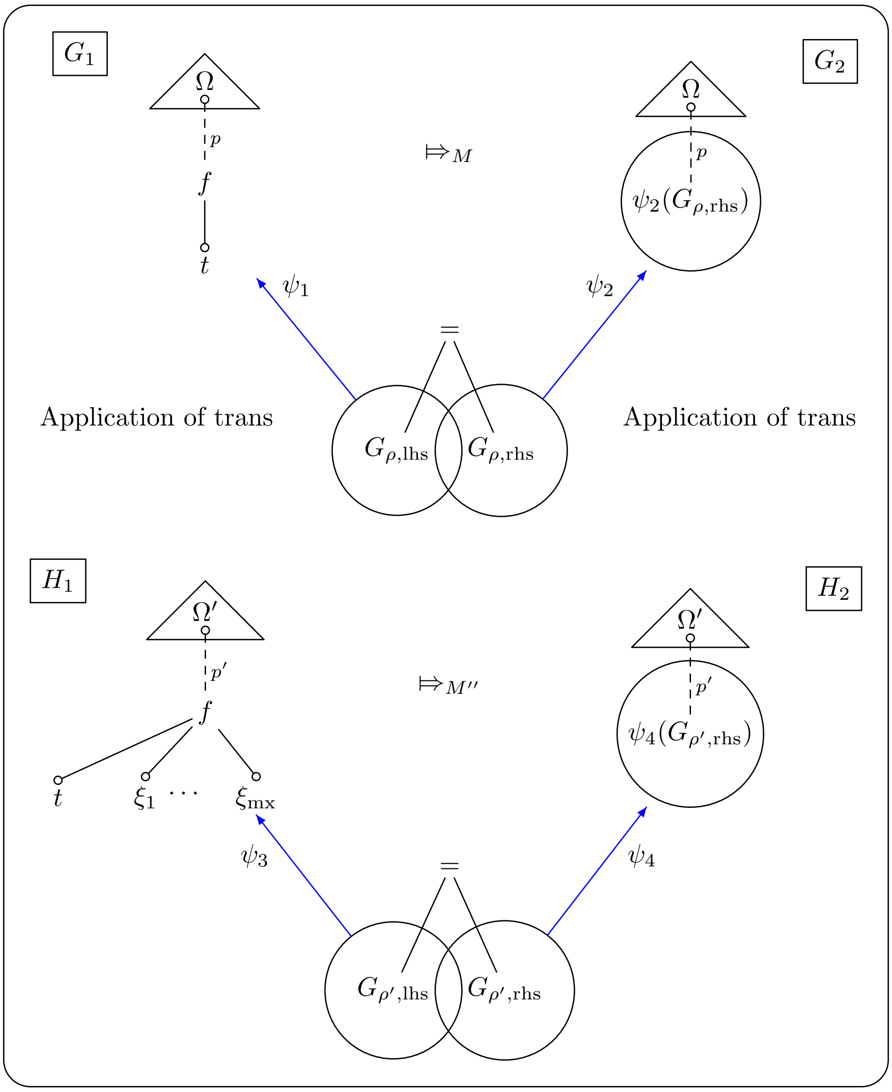

Figure 22: Term graphs and their relationships; we use $\psi(G)$ to denote the image of G under ψ.

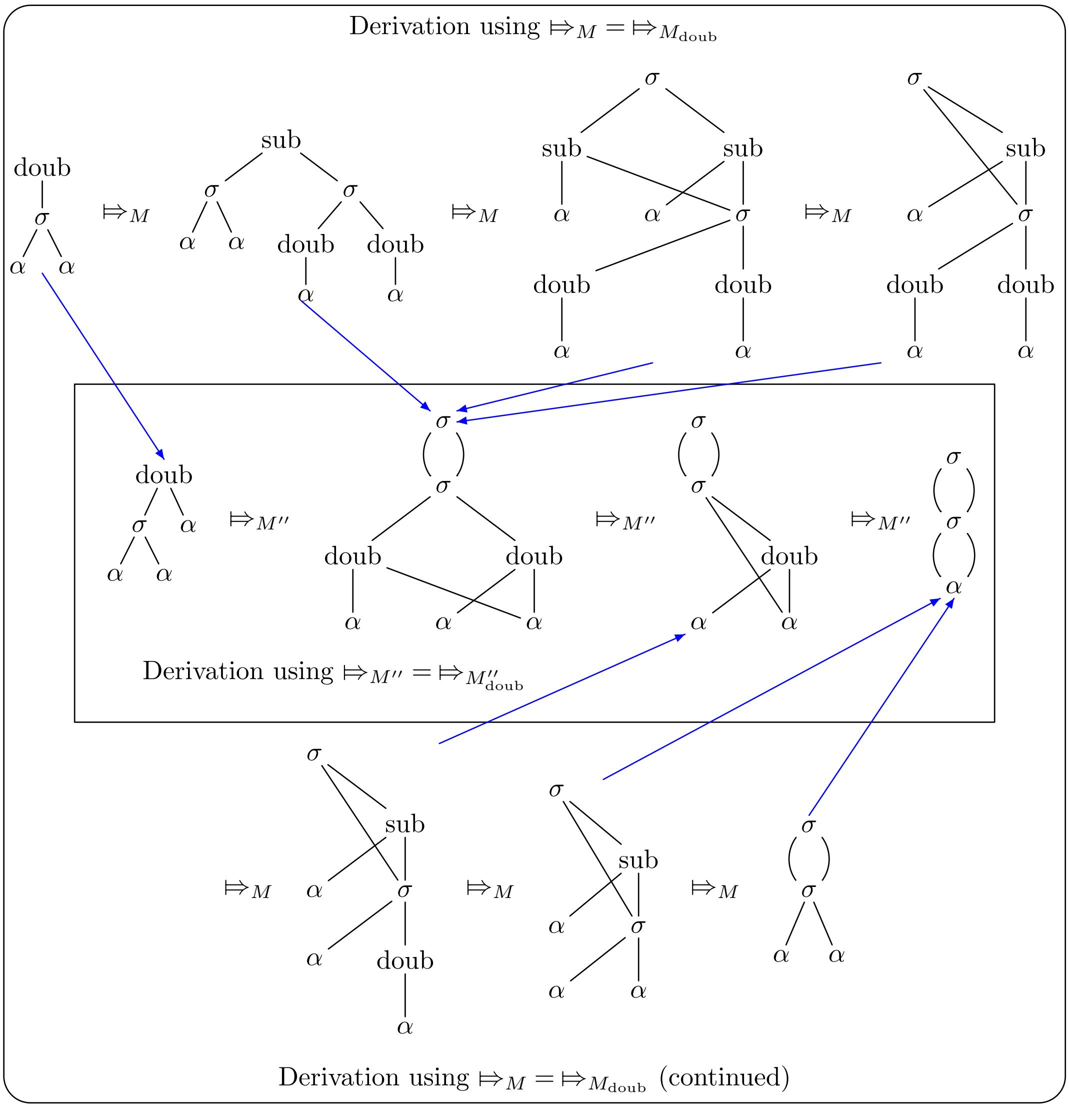

Figure 23: Illustrating the correspondence; the arrows connect corresponding sentential form graphs.

6 The extended construction

The previous sections established a construction, which given a 2-modular tree transducer consisting of a top-down tree transducer module and a substitution module (in that order), constructs a semantically equivalent macro tree transducer. In [Küh01] the possibility to extend the indirect construction of [KGK01] was already discussed. Namely, the condition that the first module is a top-down tree transducer module is lifted, so, roughly speaking, the composition phase of the indirect construction has to be altered to cope with macro tree transducers.

However, the composition of the translation induced by an unrestricted macro tree transducer with a yield translation (the macro tree transducer at the composition phase of [KGK01] realizes a yield function [FV98]) is, in general, not computable by a macro tree transducer. We will prove this claim formally, but beforehand we introduce yield macro tree transducers.

Definition 6.1 (Yield macro tree transducer). A macro tree transducer

$$M = (\{\mathrm{yield}^{(\mathrm{mx}+1)}\}, \Delta', (\mathrm{yield}\, x\, \Pi_1\, \ldots\, \Pi_{\mathrm{mx}}), R)$$

for some $\mathrm{mx} \in \mathbb{N}$ is a *yield macro tree transducer*, if and only if there exists a partition $\Delta' = \Delta \cup \{\mathrm{SUB}^{(\mathrm{mx}+1)}\}$ of Δ' into the SUB-*constructor* and the *output constructors* with $\Delta \cap \{\mathrm{SUB}^{(\mathrm{mx}+1)}\} = \emptyset$, such that

$$(\{\mathrm{yield}^{(\mathrm{mx}+1)}\},\, \emptyset,\, \Delta,\, R \setminus \{\mathrm{yield}\,(\mathrm{SUB}\, x_1\, \ldots\, x_{\mathrm{mx}+1})\, y_1\, \ldots\, y_{\mathrm{mx}} = \mathrm{rhs}_M(\mathrm{yield}, \mathrm{SUB})\})$$

is a substitution module with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ and R contains

$$\mathrm{yield}\,(\mathrm{SUB}\, x_1 \ldots x_{\mathrm{mx}+1})\, y_1 \ldots y_{\mathrm{mx}} = \mathrm{yield}\, x_1\, (\mathrm{yield}\, x_2\, y_1 \ldots y_{\mathrm{mx}})\, \ldots\, (\mathrm{yield}\, x_{\mathrm{mx}+1}\, y_1 \ldots y_{\mathrm{mx}}).$$

□

Example 6.2 (Yield macro tree transducer M_{yield}). We construct the yield macro tree transducer

$$M_{\mathrm{yield}} = (\{\mathrm{yield}^{(2)}\},\, \{\mathrm{SUB}^{(2)}, S^{(1)}, Z^{(0)}\},\, (\mathrm{yield}\, x\, Z),\, R),$$

where R contains the equations

$$\begin{array}{llcl} \mathrm{yield} & Z & y_1 = & y_1 \\ \mathrm{yield} & (S\, x_1) & y_1 = & S\, (\mathrm{yield}\, x_1\, y_1) \\ \mathrm{yield} & (\mathrm{SUB}\, x_1\, x_2) & y_1 = & \mathrm{yield}\, x_1\, (\mathrm{yield}\, x_2\, y_1). \end{array}$$

The set of substitution variables is $\Pi = \{Z\}$ and the partition is $\{S, Z\} \cup \{\mathrm{SUB}\}$. □

The proof uses results of formal language theory and there one usually defines classes of translations computable by certain devices. We will introduce the classes of translations for macro tree transducers, top-down tree transducers and yield macro tree transducers.

Definition 6.3 (Classes of induced translations). We denote the *class of translations* computable by

- macro tree transducers as $\mathrm{MAC} = (\, \tau_M \mid M \text{ is a macro tree transducer}\,)$,

- top-down tree transducers as $\mathrm{TOP} = (\,\tau_M \mid M$ is a top-down tree transducer $)$ and
- yield macro tree transducers as $\mathrm{YIELD} = (\,\tau_M \mid M$ is a yield macro tree transducer $)$.

Additionally, we define the composition of classes of relations C_1 and C_2 to be

$$C_1 \circ C_2 = (\, c_1 \circ c_2 \mid c_1 \in C_1,\, c_2 \in C_2\,).$$

$\square$

The next lemma states an important property of macro tree transducers, namely the property that the class of translations induced by macro tree transducers is not closed under composition. We will use this statement to derive a contradiction in the proof of Lemma 6.5.

Lemma 6.4 (The class of macro tree transductions is not closed under composition).

$$\mathrm{MAC} \subset \mathrm{MAC} \circ \mathrm{MAC}$$

Proof. The proof can be found in [EV85]. $\square$

Lemma 6.5 (Composition of MAC **and** YIELD**).**

$$\mathrm{MAC} \circ \mathrm{YIELD} \not\subseteq \mathrm{MAC}$$

Proof. Assume $\mathrm{MAC} \circ \mathrm{YIELD} \subseteq \mathrm{MAC}$. By classical results found in [EV85, Eng80]

$$\mathrm{MAC} = \mathrm{MAC} \circ \mathrm{TOP} \tag{6.1}$$

and

$$\mathrm{TOP} \circ \mathrm{YIELD} = \mathrm{MAC}. \tag{6.2}$$

Note, that the class YIELD is defined differently in [Eng80] [69]. Thus, we get the characterization

$$\mathrm{MAC} \circ \mathrm{YIELD} \overset{(6.1)}{=} \mathrm{MAC} \circ \mathrm{TOP} \circ \mathrm{YIELD} \overset{(6.2)}{=} \mathrm{MAC} \circ \mathrm{MAC} \subseteq \mathrm{MAC},$$

which is a contradiction to Lemma 6.4. Hence, $\mathrm{MAC} \circ \mathrm{YIELD} \not\subseteq \mathrm{MAC}$. $\square$

Since the result of the construction shall be a macro tree transducer, we have to restrict the first module appropriately. In [Voi01] a construction which composes a non-copying macro tree transducer with a weakly single-use macro tree transducer was introduced. This construction can be applied, because a second order term substitution (yield mapping) can be induced by a weakly single-use macro tree transducer. We establish the mentioned properties in the next definitions.

[69] There several SUB-constructors are used, but we have already seen that those might be emulated by one SUB-constructor with maximal rank by simply supplying the corresponding substitution variables as additional subtrees, i.e. we replace a term $\mathrm{SUB}_k\ t\, t_1\ \ldots\ t_k$ by $\mathrm{SUB}_{\mathrm{mx}}\ t\, t_1\ \ldots\ t_k\ \Pi_{k+1}\ \ldots\ \Pi_{\mathrm{mx}}$ for every $k \in [\mathrm{mx}]$ with $\{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ being the substitution variables for some $\mathrm{mx} \in \mathbb{N}$. Additionally, all symbols of the output signature were set to rank 0, however, the yield-mapping mapped the new nullary output symbols to their original counterparts.

Definition 6.6 (Non-copying macro tree transducer module). A macro tree transducer module (F, E, Δ, R) is *non-copying* (in context parameters) or *context-linear*, if and only if for every $(\text{lhs} = \text{rhs}) \in R$ and $i \in \mathbb{N}_+$ the condition

$$\#_{y_i}(\text{rhs}) \leq 1$$

holds [70]. Obviously top-down tree transducer modules are always non-copying. □

Definition 6.7 (Weakly single-use macro tree transducer). A macro tree transducer $M = (F, \Delta, e, R)$ is called *weakly single-use*, if and only if for every $k, r \in \mathbb{N}, \delta \in \Delta^{(r)}$, $f \in F^{(r+1)}$ and $x_i \in X_k$ a call $(f\ x_i\ \ldots)$ occurs in at most one right hand side of $\{\, \text{rhs}_M(f', \delta) \mid (f'\ (\delta\ \ldots)\ \ldots = \text{rhs}_M(f', \delta)) \in R, f' \in F\,\}$, and there at most once. □

Example 6.8 (Non-copying and weakly single-use). We will illustrate both properties on examples.

- The macro tree transducer M'_{rev} of Example 4.2 (considered as a macro tree transducer module) is non-copying, while the second module of the modular tree transducer M_{fact} in Example 3.4 is not non-copying (or alternatively copying) as well as M'_{doub} of Example 5.4.
- The macro tree transducer M'_{rev} of Example 4.2 is weakly single-use, whereas the macro tree transducer M'_{doub} in Example 5.4 is not weakly single-use.
- Any yield macro tree transducer is weakly single-use.

□

The construction of [Voi01] requires a non-copying macro tree transducer M_1 and a weakly single-use macro tree transducer M_2 and outputs a macro tree transducer, which induces the composition of the induced translations of M_1 and M_2. Since we will apply the method of [KGK01] extended to cope with non-copying macro tree transducer modules as the first module, we will introduce another distinguished form of a top-down tree transducer module. This type of top-down tree transducer module is created as an intermediate result during the construction process, however, in the direct construction stated in the next subsection interpretation modules will not occur.

Definition 6.9 (Interpretation module). An *interpretation module* is a top-down tree transducer module $(\{\text{int}^{(1)}\}, E, \Delta, R)$, for which there exists a partition of $\Delta = \Sigma \cup C_F$ into *regular constructors* Σ and *frozen function symbols* C_F with $\Sigma \cap C_F = \emptyset$ and an injective *interpretation mapping* $\imath : C_F \longrightarrow E$, such that

- for every $n \in \mathbb{N}$ and $\sigma \in \Sigma^{(n)}$
$$(\text{int}(\sigma\ x_1\ \ldots\ x_n) = \sigma\ (\text{int}\ x_1)\ \ldots\ (\text{int}\ x_n)) \in R$$
- and for every $n \in \mathbb{N}$ and $f \in C_F^{(n)}$
$$(\text{int}(f\ x_1\ \ldots\ x_n) = \imath(f)\ (\text{int}\ x_1)\ \ldots\ (\text{int}\ x_n)) \in R.$$

□

[70] We will also call macro tree transducers non-copying, if they fulfill the presented linearity condition.

6.1 Extended accumulation technique

In this subsection we will instantiate the construction of [Voi01] to gain an extended construction, which will coincide with Construction 4.5 when applied to suitable 2-modular tree transducers, of which the first module is a top-down tree transducer module. The example to follow will be used throughout this subsection and all the presented constructions are illustrated on this particular example.

The syntactic class of all 2-modular tree transducers, of which the first module is non-copying and the second is a substitution module, is denoted $\mathrm{ModTT}(\mathrm{MAC_{nc}}, \mathrm{SUB})$, while the syntactic class of all 2-modular tree transducers, of which the first module is an interpretation module and the second module is a substitution module, is denoted $\mathrm{ModTT}(\mathrm{INT}, \mathrm{SUB})$.

Example 6.10 (Sorting the stripes). We define the 2-modular tree transducer

$$M_{\mathrm{usa}} = (\{\mathrm{coll}^{(2)}, \mathrm{app}^{(2)}\}, \{(\mathrm{coll}, 1), (\mathrm{app}, 2)\}, \{W^{(1)}, R^{(1)}, |^{(1)}, N^{(0)}\}, (\mathrm{coll}\ x\ N), P),$$

where P contains the equations:

$$\begin{array}{lccclllcccl}
\mathrm{coll} & N & y_1 & = & y_1 & \vline & \mathrm{app} & N & y_1 & = & y_1 \\
\mathrm{coll} & (\,|\,x_1) & y_1 & = & \mathrm{app}\,y_1\,(\mathrm{coll}\,x_1\,N) & \vline & \mathrm{app} & (\,|\,x_1) & y_1 & = & |\,(\mathrm{app}\,x_1\,y_1) \\
\mathrm{coll} & (R\,x_1) & y_1 & = & R\,(\mathrm{coll}\,x_1\,y_1) & \vline & \mathrm{app} & (R\,x_1) & y_1 & = & R\,(\mathrm{app}\,x_1\,y_1) \\
\mathrm{coll} & (W\,x_1) & y_1 & = & \mathrm{coll}\,x_1\,(W\,y_1) & \vline & \mathrm{app} & (W\,x_1) & y_1 & = & W\,(\mathrm{app}\,x_1\,y_1).
\end{array}$$

Obviously, $M_{\mathrm{usa}} \in \mathrm{ModTT}(\mathrm{MAC_{nc}}, \mathrm{SUB})$. □

Lemma 6.11 (Extended Lemma 11 **of [KGK01]).** *Let* $M \in \mathrm{ModTT}(\mathrm{MAC_{nc}}, \mathrm{SUB})$ *with substitution variables* $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ *for some* $\mathrm{mx} \in \mathbb{N}$*. Then there exist a non-copying macro tree transducer* M_1 *and a modular tree transducer* $M_2 \in \mathrm{ModTT}(\mathrm{INT}, \mathrm{SUB})$ *with substitution variables* Π*, such that*

$$\tau_M = \tau_{M_1} \circ \tau_{M_2}.$$

Proof. Let $M = (F, m, \Delta, e, R)$ consist of the modules $m_1 = (m^{-1}(1), m^{-1}(2), \Delta, R_1)$ and $m_2 = (m^{-1}(2), \emptyset, \Delta, R_2)$ (in that order), where m_1 is non-copying and m_2 is a substitution module with substitution variables Π. We construct the non-copying macro tree transducer M_1 by freezing external function symbols in m_1, thus let $\Delta' = \Delta \cup \{\mathrm{SUB}^{(\mathrm{mx}+1)}\}$ in

$$M_1 = (m^{-1}(1), \Delta', e, R'),$$

where $\mathrm{SUB} \notin \Delta$ is a new constructor and R' contains for every $k, r \in \mathbb{N}$, $f \in m^{-1}(1)^{(r+1)}$, $\delta \in \Delta^{(k)}$ and $(f\ (\delta\, x_1\ \ldots\ x_k)\, y_1\ \ldots\ y_r = \mathrm{rhs}_M(f, \delta)) \in R_1$, the equation

$$f\ (\delta\, x_1\ \ldots\ x_k)\, y_1\ \ldots\ y_r = \mathrm{freeze}_{k,r}(\mathrm{rhs}_M(f, \delta)).$$

For every $k, r \in \mathbb{N}$ the mapping

$$\mathrm{freeze}_{k,r} :\ \mathrm{RHS}(1, F, m, \Delta, X_k, Y_r) \longrightarrow \mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r)$$

is defined by structural recursion over its only argument.

- for $y \in Y_r$:
$$\mathrm{freeze}_{k,r}(y) = y \tag{6.3}$$

- for $a \in \mathbb{N}$, $\delta \in \Delta^{(a)}$ and $s_1, \ldots, s_a \in \mathrm{RHS}(1, F, m, \Delta, X_k, Y_r)$:

$$\mathrm{freeze}_{k,r}(\delta\, s_1\, \ldots\, s_a) = (\delta\, \mathrm{freeze}_{k,r}(s_1)\, \ldots\, \mathrm{freeze}_{k,r}(s_a)) \tag{6.4}$$

- for $a \in \mathbb{N}$, $f \in m^{-1}(1)^{(a+1)}$, $x \in X_k$ and $s_1, \ldots, s_a \in \mathrm{RHS}(1, F, m, \Delta, X_k, Y_r)$:

$$\mathrm{freeze}_{k,r}(f\, x\, s_1\, \ldots\, s_a) = (f\, x\, \mathrm{freeze}_{k,r}(s_1)\, \ldots\, \mathrm{freeze}_{k,r}(s_a)) \tag{6.5}$$

- for $a \in \mathbb{N}$, $\mathrm{sub}_a \in m^{-1}(2)^{(a+1)}$ and $s, s_1, \ldots, s_a \in \mathrm{RHS}(1, F, m, \Delta, X_k, Y_r)$:

$$\mathrm{freeze}_{k,r}(\mathrm{sub}_a\, s\, s_1 \ldots s_a) = \mathrm{SUB}\, \mathrm{freeze}_{k,r}(s)\, \mathrm{freeze}_{k,r}(s_1) \ldots \mathrm{freeze}_{k,r}(s_a)\, \Pi_{a+1} \ldots \Pi_{\mathrm{mx}} \tag{6.6}$$

Thus, for every $k, r \in \mathbb{N}$, $\delta \in \Delta^{(k)}$, $f \in m^{-1}(1)^{(r+1)}$ and $(f\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r = \mathrm{rhs}_M(f, \delta)) \in R$ no function symbol $\mathrm{sub} \in m^{-1}(2)$ occurs in $\mathrm{freeze}_{k,r}(\mathrm{rhs}_M(f, \delta))$ and also $\#_{y_i}(\mathrm{freeze}_{k,r}(\mathrm{rhs}_M(f, \delta))) \leq 1$ for every $i \in [r]$, since no context variable is duplicated. Consequently, M_1 is a well-defined non-copying macro tree transducer.

The 2-modular tree transducer M_2 is defined by

$$M_2 = (\{\mathrm{int}^{(1)}, \mathrm{sub}^{(\mathrm{mx}+1)}\},\ \{(\mathrm{int}, 1), (\mathrm{sub}, 2)\},\ \Delta',\ (\mathrm{int}\ x),\ R''),$$

where $\{\mathrm{sub}\} = m^{-1}(2)^{(\mathrm{mx}+1)}$, $\mathrm{int} \notin F$ and R'' contains

- $\mathrm{sub}\,(\mathrm{SUB}\, x_1 \ldots x_{\mathrm{mx}+1})\, y_1 \ldots y_{\mathrm{mx}} = \mathrm{SUB}\,(\mathrm{sub}\, x_1\, y_1 \ldots y_{\mathrm{mx}}) \ldots (\mathrm{sub}\, x_{\mathrm{mx}+1}\, y_1 \ldots y_{\mathrm{mx}})$ [71],
- the equation $(\mathrm{sub}\,(\delta\, x_1\, \ldots\, x_k)\, y_1\, \ldots\, y_{\mathrm{mx}} = \mathrm{rhs}_M(\mathrm{sub}, \delta))$ of R_2 for every $k \in \mathbb{N}$ and $\delta \in \Delta^{(k)}$,
- for every $k \in \mathbb{N}$ and $\delta \in \Delta^{(k)}$, the equation $\mathrm{int}\,(\delta\, x_1\, \ldots\, x_k) = (\delta\,(\mathrm{int}\, x_1)\, \ldots\, (\mathrm{int}\, x_k))$ and
- the equation $\mathrm{int}\,(\mathrm{SUB}\, x_1\, \ldots\, x_{\mathrm{mx}+1}) = \mathrm{sub}\,(\mathrm{int}\, x_1)\, \ldots\, (\mathrm{int}\, x_{\mathrm{mx}+1})$.

Thus, M_2 is a 2-modular tree transducer consisting of an interpretation module (partition $\Delta \cup \{\mathrm{SUB}^{(\mathrm{mx}+1)}\}$ and mapping $\imath = \{(\mathrm{SUB}, \mathrm{sub})\}$) together with a substitution module with substitution variables Π. In the following we will use the shorthand $(M_1, M_2) = \mathfrak{C}_{6.11}(M)$ to denote the application of this construction to M. The proof that $\tau_M = \tau_{M_1} \circ \tau_{M_2}$ is then immediate from Lemma 5.1 of [EV91]. □

Example 6.12 (Sorting the stripes, continued). Let M_{usa} be the 2-modular tree transducer of Example 6.10. Then $\mathfrak{C}_{6.11}(M_{\mathrm{usa}}) = (M_{\mathrm{usa},1}, M_{\mathrm{usa},2})$ with

$$M_{\mathrm{usa},1} = (\{\mathrm{coll}^{(2)}\}, \{W^{(1)}, R^{(1)}, |^{(1)}, N^{(0)}, \mathrm{SUB}^{(2)}\}, (\mathrm{coll}\, x\, N), P'),$$

where P' contains the equations:

[71] This equation is only required to satisfy the totality restriction in accordance with the restrictions on right hand sides of substitution modules. In particular, outputting an arbitrary output symbol (dummy symbol) is not possible, since the second module of M_2 shall be a substitution module.

$$
\begin{aligned}
\mathrm{coll}\, N\, y_1 &= \mathrm{freeze}_{0,1}(y_1) \overset{(6.3)}{=} y_1 \\
\mathrm{coll}\, (\,|\, x_1)\, y_1 &= \mathrm{freeze}_{1,1}(\mathrm{app}\, y_1\, (\mathrm{coll}\, x_1\, N)) \overset{(6.6)}{=} \mathrm{SUB}\, \mathrm{freeze}_{1,1}(y_1)\, \mathrm{freeze}_{1,1}(\mathrm{coll}\, x_1\, N) \\
&\overset{(6.3)}{=} \mathrm{SUB}\, y_1\, \mathrm{freeze}_{1,1}(\mathrm{coll}\, x_1\, N) \overset{(6.5)}{=} \mathrm{SUB}\, y_1\, \mathrm{coll}\, x_1\, \mathrm{freeze}_{1,1}(N) \\
&\overset{(6.4)}{=} \mathrm{SUB}\, y_1\, \mathrm{coll}\, x_1\, N \\
\mathrm{coll}\, (R\, x_1)\, y_1 &= \mathrm{freeze}_{1,1}(R\, (\mathrm{coll}\, x_1\, y_1)) \overset{(6.4)\,\&\,(6.5)\,\&\,(6.3)}{=} R\, (\mathrm{coll}\, x_1\, y_1) \\
\mathrm{coll}\, (W\, x_1)\, y_1 &= \mathrm{freeze}_{1,1}(\mathrm{coll}\, x_1\, (W\, y_1)) \overset{(6.5)\,\&\,(6.4)\,\&\,(6.3)}{=} \mathrm{coll}\, x_1\, (W\, y_1).
\end{aligned}
$$

Obviously, $M_{\mathrm{usa},1}$ is a non-copying macro tree transducer. $M_{\mathrm{usa},2}$ is

$$(\{\mathrm{int}^{(1)}, \mathrm{app}^{(2)}\},\ \{(\mathrm{int}, 1), (\mathrm{sub}, 2)\},\ \{W^{(1)}, R^{(1)},\ |^{(1)}, N^{(0)}, \mathrm{SUB}^{(2)}\},\ (\mathrm{int}\, x),\ P''),$$

where P'' contains:

$$
\begin{array}{lccclc|lccl}
\mathrm{app} & N & y_1 & = & y_1 & & \mathrm{int} & N & = & N \\
\mathrm{app} & (\,|\, x_1) & y_1 & = & |\, (\mathrm{app}\, x_1\, y_1) & & \mathrm{int} & (\,|\, x_1) & = & |\, (\mathrm{int}\, x_1) \\
\mathrm{app} & (R\, x_1) & y_1 & = & R\, (\mathrm{app}\, x_1\, y_1) & & \mathrm{int} & (R\, x_1) & = & R\, (\mathrm{int}\, x_1) \\
\mathrm{app} & (W\, x_1) & y_1 & = & W\, (\mathrm{app}\, x_1\, y_1) & & \mathrm{int} & (W\, x_1) & = & W\, (\mathrm{int}\, x_1)
\end{array}
$$

along with the two equations

$$\mathrm{app}(\mathrm{SUB} x_1 x_2) y_1 = \mathrm{SUB}(\mathrm{app} x_1 y_1)(\mathrm{app} x_2 y_1) \quad \text{and} \quad \mathrm{int}(\mathrm{SUB} x_1 x_2) = \mathrm{app}(\mathrm{int} x_1)(\mathrm{int} x_2).$$

Apparently, $M_{\mathrm{usa},2} \in \mathrm{ModTT}(\mathrm{INT}, \mathrm{SUB})$. □

Lemma 6.13 (Extended Lemma 13 of [KGK01]). *Let*

$$M_2 = (\{\mathrm{int}^{(1)}, \mathrm{sub}^{(\mathrm{mx}+1)}\}, \{(\mathrm{int}, 1), (\mathrm{sub}, 2)\}, \Delta', e, R) \in \mathrm{ModTT}(\mathrm{INT}, \mathrm{SUB})$$

with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ *for some* $\mathrm{mx} \in \mathbb{N}$ *and an interpretation mapping* $\imath = \{(\mathrm{SUB}, \mathrm{sub})\}$. *Then there exists a weakly single-use (yield) macro tree transducer* M_2' *with*

$$\tau_{M_2} = \tau_{M_2'}.$$

Proof. The macro tree transducer M_2' is defined by

$$M_2' = (\{\mathrm{sub}^{(\mathrm{mx}+1)}\}, \Delta', e', R'),$$

where $e' = (\mathrm{sub}\, x\, \Pi_1\, \ldots\, \Pi_{\mathrm{mx}})$ and R' contains

- the equation $(\mathrm{sub}\, (\delta\, x_1\, \ldots\, x_k)\, y_1\, \ldots\, y_{\mathrm{mx}} = \mathrm{rhs}_{M_2}(\mathrm{sub}, \delta)) \in R$ for every $k \in \mathbb{N}$ and $\delta \in \Delta^{(k)} \setminus \{\mathrm{SUB}\}$ and
- the equation

  $$\mathrm{sub}(\mathrm{SUB} x_1 \ldots x_{\mathrm{mx}+1}) y_1 \ldots y_{\mathrm{mx}} = \mathrm{sub} x_1 (\mathrm{sub} x_2 y_1 \ldots y_{\mathrm{mx}}) \ldots (\mathrm{sub} x_{\mathrm{mx}+1} y_1 \ldots y_{\mathrm{mx}}).$$

Obviously, M_2' is a weakly single-use (yield) macro tree transducer by Definition 4.1. We will use $M_2' = \mathfrak{C}_{6.13}(M_2)$ to denote the application of this construction. The equivalence of the induced translations, i.e. $\tau_{M_2} = \tau_{M_2'}$, is immediate from [EV85, FV98], since we have already proven in Lemma 4.4 that substitution modules provide (first order term) substitutions [72], so together with the interpretation module M_2 realizes a yield-mapping [73], which was shown to be implemented by M' [74]. □

Example 6.14 (Sorting the stripes (3)). Let $M_{\text{usa},2}$ be the 2-modular tree transducer of Example 6.12. Then $\mathfrak{C}_{6.13}(M_{\text{usa},2}) = M'_{\text{usa},2}$ with

$$M'_{\text{usa},2} = (\{\text{app}^{(2)}\},\ \{W^{(1)}, R^{(1)}, |^{(1)}, N^{(0)}, \text{SUB}^{(2)}\},\ (\text{app}\ x\ N),\ P_2),$$

where P_2 contains:

$$\begin{array}{llcll}
\text{app} & N & y_1 & = & y_1 \\
\text{app} & (\,|\,x_1) & y_1 & = & |\,(\text{app}\ x_1\ y_1) \\
\text{app} & (R\ x_1) & y_1 & = & R\ (\text{app}\ x_1\ y_1) \\
\text{app} & (W\ x_1) & y_1 & = & W\ (\text{app}\ x_1\ y_1) \\
\text{app} & (\text{SUB}\ x_1\ x_2) & y_1 & = & \text{app}\ x_1\ (\text{app}\ x_2\ y_1).
\end{array}$$

$M'_{\text{usa},2}$ is weakly single-use. □

The heart of the extended construction is the composition of a non-copying macro tree transducer with a weakly single-use macro tree transducer. In [Voi01] a direct construction for the given scenario is presented. Another presentation of the construction can be found in [VK01], but we prefer to use the version of [Voi01], because this way the gained construction will subsume Construction 4.5.

Before stating Construction 3.2 of [Voi01], we give a brief overview and the underlying idea of this construction. This is basically an excerpt of [Voi01], where elucidation on every single detail of the construction can be found.

The construction is a generalization of a classical construction of [EV85], which was used to prove MAC∘TOP ⊆ MAC. Given sets of function symbols F_1 and F_2 of the macro tree transducers M_1 and M_2, respectively, the constructed macro tree transducer M uses paired function symbols $(f,g) \in F_1 \times F_2$ and expressions like $(f,g)\ x\ ?\cdots?$ to correspond to the composition expression [75] $g\ (f\ x\ ?\cdots?)\ ?\cdots?$. Obviously, we can keep any context parameter of g, such that

$$(f,g)\ x\ ?\cdots?\ z_1\ \ldots\ z_\kappa \qquad \text{corresponds to} \qquad g\ (f\ x\ ?\cdots?)\ z_1\ \ldots\ z_\kappa,$$

if $\kappa \in \mathbb{N}$, $g \in F_2^{(\kappa+1)}$. Following the idea of [EV85], we have to provide all translations of context parameters $y_1, \ldots, y_r$ with $r \in \mathbb{N}$ of f as context parameters to (f,g). This is necessary, because the call $(f\,x\,y_1\ \ldots\ y_r)$ might reduce to some y_j with $j \in [r]$ and $(g\,y_j\ \ldots)$

[72] The base function of the yield-mapping.

[73] In Lemma 6.5 we have already elucidated on the slight differences encountered in [EV85] as well as [FV98].

[74] Alternatively, in [KGK01] the proof is spelled out with the marginal difference that the substitution module might possibly contain several function symbols, which are preserved in the result.

[75] Strictly speaking, the correspondence is established on sentential forms, i.e. with concrete input and output tree substituted for the input as well as output subtree variables.

does not constitute a valid right hand side of a macro tree transducer, thus $(g\, y_j\ \dots)$ is supplied as context parameter. The translation of y_j via g is denoted $y_{1,g}$, thus

$$(f,g)\, x\, y_{1,g_1}\ \dots\ y_{r,g_1}\, y_{1,g_2}\ \dots\ y_{r,g_{\mathrm{card}(F_2)}}\, z_1\ \dots\ z_\kappa \quad \text{corresponds to} \quad g\,(f\, x\, y_1\ \dots\ y_r)\, z_1\ \dots\ z_\kappa,$$

where $F_2 = \{g_1, \dots, g_{\mathrm{card}(F_2)}\}$. Consequently, the rank of the new function symbol (f, g) is $\mathrm{card}(F_2) * r + \kappa + 1$.

The right hand side of the new paired function symbol (f, g) at δ is constructed by rewriting the right hand side $\mathrm{rhs}_{M_1}(f, \delta)$ of f at a constructor δ using the equations of M_2. This rewriting is straightforward, if $\mathrm{rhs}_{M_1}(f, \delta)$ consists of constructors solely. If, however, a context parameter is encountered, then we output the appropriate (pre-translated) supplied context parameter, i.e. if during the rewriting process a function symbol $g' \in F_2$ approaches at some y_j, then we output $y_{j,g'}$. Finally, during rewriting the right hand side a function symbol $g' \in F_2^{(\kappa'+1)}$ with $\kappa' \in \mathbb{N}$ might also hit calls to function symbols, like $(f'\ x_i\ \phi_1\ \dots\ \phi_a)$ for some $a \in \mathbb{N}$, $f' \in F_1^{(a+1)}$, with context parameters $\phi_1, \dots, \phi_a$. By the established correspondence we imagine that we should perform the following rewrite:

$$g'\,(f'\, x_i\, \phi_1\ \dots\ \phi_a)\, z_1\ \dots\ z_{\kappa'} = (f', g')\, x_i\, (g'\, \phi_1\, ?\cdots?)\ \dots\ (g'\, \phi_a\, ?\cdots?)\, z_1\ \dots\ z_{\kappa'}.$$

In case M_2 is a top-down tree transducer we are done, since g' does not have context parameters. Otherwise, we somehow need to determine the values at the question mark positions and we will use designated function symbols to accomplish this task, namely the function symbol (h_f, l_g) [76] shall compute the l-th context parameter of g when g (during the derivation process of M_2) is applied to the h-th context parameter of f. Accordingly, we should demand that we can actually (uniquely) determine the l-th context parameter of g in the situation mentioned above. Therefore, we restrict the macro tree transducer M_1 to be non-copying [77] and M_2 to be weakly single-use. The latter property allows us to determine the sought l-th context parameter (of g) in a bottom-up fashion starting at the unique occurrence of the value stored in the h-th context parameter of f in the output tree of M_1 in the following manner:

- Assume a call $(g'\ s\ \eta_1\ \dots\ \eta_{\kappa'})$ and $s = y_h$, i.e. y_h occurs at the root of s, then if $g' = g$, the sought context parameter is η_l, otherwise we output some dummy value.

- Again assume a call $(g'\, s\, \eta_1\ \dots\ \eta_{\kappa'})$, but y_h occurs nested under some output symbol δ of M_1, say as the i-th descendant of δ. Then there is at most one way, how a state $g'' \in F_2^{(r+1)}$ with $r \in \mathbb{N}$ can reach y_h, namely by the unique; if existing; (due to the weakly single-use property) call to $(g''\ x_i\ \chi_1\ \dots\ \chi_r)$ in a right hand side at δ. Thus, we can determine the required context parameters depending on the context parameters occurring in χ_l. These context parameters can be determined in a similar fashion, thereby walking upwards in the output tree of M_1; eventually reaching the root.

We note that this explanation is not complete, nor is it readily applicable. Additional effort is required to ensure that the above mentioned strategy can be implemented by a

[76] This shall not denote indexed integers, rather we use it as a shorthand for $((f, h), (g, l))$.

[77] Consequently, any subtree stored in a context variable is outputted at most once by M_1.

macro tree transducer, which as such is not able to compute the normal form of some term and then compute the sought context parameter in a bottom-up fashion starting at the image of some context parameter of M_1.

We conclude our overview by stating the general purpose of the mappings used in the construction. The trans-mappings perform the rewriting of the right hand sides of M_1 in combination with the thru-mappings, which perform the rewrites using equations of the right hand sides of M_2. The unfold-mappings compute the context parameter translations provided and, finally, the ctx-mappings generate the right hand sides of the auxiliary function symbols, like (h_f, l_g).

Construction 6.15 (Construction 3.2 of [Voi01]). Let $M_1 = (F_1, \Delta, e_1, R_1)$ and $M_2' = (F_2, \Delta, e_2, R_2)$ be macro tree transducers, such that M_1 is non-copying and M_2' is weakly single-use. We construct the macro tree transducer $M = (F, \Delta, e, R) = \mathfrak{C}_{6.15}(M_1, M_2')$ as follows:

Let $\mu = \mathrm{card}(F_2)$ and fix some ordering of states in F_2, such that $F_2 = \{g_1, \ldots, g_\mu\}$. For $1 \le i \le \mu$, let $\kappa_i \in \mathbb{N}$ be such that $g_i \in {F_2}^{(\kappa_i+1)}$. Additionally, let nil $\in \Delta^{(0)}$ be some dummy output symbol and $Z_{F_2} = \{z_{g_1,1}, \ldots, z_{g_1,\kappa_1}, \ldots, z_{g_\mu,1}, \ldots, z_{g_\mu,\kappa_\mu}\}$. Then the components of M are obtained as follows:

- $F = \{ \, (f,g)^{(r*\mu+\kappa+1)} \mid r, \kappa \in \mathbb{N},\ f \in {F_1}^{(r+1)},\ g \in {F_2}^{(\kappa+1)} \, \} \cup$
  $\cup \ \{ \, (h_f, l_g)^{(r*\mu+\mathrm{card}(Z_{F_2})+1)} \mid r, \kappa \in \mathbb{N},\ f \in {F_1}^{(r+1)},\ h \in [r],\ g \in {F_2}^{(\kappa+1)},\ l \in [\kappa] \, \}$
- $e = \mathrm{thru}_{1,0,1,0,\emptyset}(e_2, e_1)$
- R contains:
  - for $k, r, \kappa \in \mathbb{N}$, $\delta \in \Delta^{(k)}$, $f \in {F_1}^{(r+1)}$ and $g \in {F_2}^{(\kappa+1)}$, the rule:

    $$(f,g)\,(\delta\, x_1 \, \ldots \, x_k)\, y_{1,g_1} \, \ldots \, y_{r,g_\mu}\, z_1 \, \ldots \, z_\kappa = \mathrm{trans}_{g,k,r,Z_\kappa}(\mathrm{rhs}_{M_1}(f,\delta), z_1, \ldots, z_\kappa),$$

    if R_1 contains the rule $f\,(\delta\, x_1 \, \ldots \, x_k)\, y_1 \, \ldots \, y_r = \mathrm{rhs}_{M_1}(f,\delta)$
  - for $k, r, \kappa \in \mathbb{N}$, $\delta \in \Delta^{(k)}$, $f \in {F_1}^{(r+1)}$, $h \in [r]$, $g \in {F_2}^{(\kappa+1)}$ and $l \in [\kappa]$, the rule:

    $$(h_f, l_g)\,(\delta\, x_1 \, \ldots \, x_k)\, y_{1,g_1} \, \ldots \, y_{r,g_\mu}\, z_{g_1,1} \, \ldots \, z_{p_\mu,\kappa_\mu} = \psi,$$

    if R_1 contains the rule $f\,(\delta\, x_1 \, \ldots \, x_k)\, y_1 \, \ldots \, y_r = \mathrm{rhs}_{M_1}(f,\delta)$, where $\psi =$ nil, if $\mathrm{rhs}_{M_1}(f,\delta) \in \mathrm{RHS}(F_1, \Delta, X_k, Y_r)$ does not contain the context variable y_h; otherwise $\psi = \mathrm{ctx}_{g,\mathrm{rhs}_{M_1}(f,\delta),k,r}(p,l)$ with $p \in \mathrm{occ}(\mathrm{rhs}_{M_1}(f,\delta))$ as the unique occurrence of y_h in $\mathrm{rhs}_{M_1}(f,\delta)$ (M_1 is non-copying, thus $\#_{y_h}(\mathrm{rhs}_{M_1}(f,\delta)) \le 1$!).

For every $k, r \in \mathbb{N}$ and finite set of variables V, the following set of mappings is defined by simultaneous recursion.

$$\begin{aligned}\{ \, \mathrm{trans}_{g,k,r,V} \mid g \in F_2 \, \} \cup \{ \, \mathrm{unfold}_{f(x_i),k,r,V} \mid f \in F_1,\ x_i \in X_k \, \} \cup \\ \cup \{ \, \mathrm{thru}_{a,\kappa,k,r,V} \mid a \in \mathrm{ar}(\Delta),\ (\kappa+1) \in \mathrm{ar}(F_2) \text{ or } a = 1,\ \kappa = 0 \, \}\end{aligned}$$

With $\mathrm{RHS}_{k,r,V} = \mathrm{RHS}(F, \Delta, X_k, \{y_{1,g_1}, \ldots, y_{r,g_\mu}\} \cup V)$, these have the following types:

- for $\kappa \in \mathbb{N}$, $g \in {F_2}^{(\kappa+1)}$:

  $$\mathrm{trans}_{g,k,r,V} : \ \mathrm{RHS}(F_1, \Delta, X_k, Y_r) \times (\mathrm{RHS}_{k,r,V})^\kappa \longrightarrow \mathrm{RHS}_{k,r,V}$$

- for $a \in \mathrm{ar}(\Delta)$, $(\kappa+1) \in \mathrm{ar}(F_2)$ or $a = 1$, $\kappa = 0$:

  $\mathrm{thru}_{a,\kappa,k,r,V} : \mathrm{RHS}(F_2, \Delta, X_a, Y_\kappa) \times \mathrm{RHS}(F_1, \Delta, X_k, Y_r)^a \times (\mathrm{RHS}_{k,r,V})^\kappa \longrightarrow \mathrm{RHS}_{k,r,V}$

- for $a \in \mathbb{N}$, $f \in {F_1}^{(a+1)}$ and $x_i \in X_k$:

$$\mathrm{unfold}_{f(x_i),k,r,V} : [a] \times F_2 \times \mathfrak{P}([a] \times F_2) \times \mathrm{RHS}(F_1, \Delta, X_k, Y_r)^a \times (\mathrm{RHS}_{k,r,V})^{\mathrm{card}(Z_{F_2})} \\ \longrightarrow \mathrm{RHS}_{k,r,V}$$

For every $\kappa \in \mathbb{N}$ and $g \in {F_2}^{(\kappa+1)}$, the mapping $\mathrm{trans}_{g,k,r,V}$ is defined with $\varrho_1, \ldots, \varrho_\kappa \in \mathrm{RHS}_{k,r,V}$ as follows:

- for $y_h \in Y_r$
$$\mathrm{trans}_{g,k,r,V}(y_h, \varrho_1, \ldots, \varrho_\kappa) = y_{h,g} \tag{6.7}$$

- for $a \in \mathbb{N}$, $f \in {F_1}^{(a+1)}$, $x_i \in X_k$ and $s_1, \ldots, s_a \in \mathrm{RHS}(F_1, \Delta, X_k, Y_r)$, where for every $\kappa' \in \mathbb{N}$, $g' \in {F_2}^{(\kappa'+1)}$ and $l' \in [\kappa']$, we have $\varrho_{g',l'} = (\text{if } g' = g \text{ then } \varrho_{l'} \text{ else nil})$

$$\begin{aligned} &\mathrm{trans}_{g,k,r,V}(f\ x_i\ s_1\ \ldots\ s_a, \varrho_1, \ldots, \varrho_\kappa) \\ &\quad = (f, g)\ x_i\ \mathrm{unfold}_{f(x_i),k,r,V}(1, g_1, \emptyset, s_1, \ldots, s_a, \varrho_{g_1,1}, \ldots, \varrho_{g_\mu,\kappa_\mu}) \\ &\qquad\qquad \ldots \\ &\qquad\qquad \mathrm{unfold}_{f(x_i),k,r,V}(a, g_\mu, \emptyset, s_1, \ldots, s_a, \varrho_{g_1,1}, \ldots, \varrho_{g_\mu,\kappa_\mu}) \\ &\qquad\qquad \varrho_1\ \cdots\ \varrho_\kappa \end{aligned} \tag{6.8}$$

- for $a \in \mathbb{N}$, $\delta \in \Delta^{(a)}$, $s_1, \ldots, s_a \in \mathrm{RHS}(F_1, \Delta, X_k, Y_r)$ and rule $g(\delta x_1 \ldots x_a) y_1 \ldots y_\kappa = \mathrm{rhs}_{M_2'}(g, \delta)$ in R_2, with $\mathrm{rhs}_{M_2'}(p, \delta) \in \mathrm{RHS}(F_2, \Delta, X_a, Y_\kappa)$

$$\mathrm{trans}_{g,k,r,V}((\delta\ s_1\ \ldots\ s_a), \varrho_1, \ldots, \varrho_\kappa) = \mathrm{thru}_{a,\kappa,k,r,V}(\mathrm{rhs}_{M_2'}(g, \delta), s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_\kappa) \tag{6.9}$$

For every $a \in \mathrm{ar}(\Delta)$, $(\kappa+1) \in \mathrm{ar}(F_2)$ or $a = 1$, $\kappa = 0$, the mapping $\mathrm{thru}_{a,\kappa,k,r,V}$ is defined with $s_1, \ldots, s_a \in \mathrm{RHS}(F_1, \Delta, X_k, Y_r)$ and $\varrho_1, \ldots, \varrho_\kappa \in \mathrm{RHS}_{k,r,V}$ as follows:

- for $y_l \in Y_\kappa$
$$\mathrm{thru}_{a,\kappa,k,r,V}(y_l, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_\kappa) = \varrho_l \tag{6.10}$$

- for $n \in \mathbb{N}$, $\delta \in \Delta^{(n)}$ and $\phi_1, \ldots, \phi_n \in \mathrm{RHS}(F_2, \Delta, X_a, Y_\kappa)$

$$\begin{aligned} &\mathrm{thru}_{a,\kappa,k,r,V}((\delta\ \phi_1, \ldots, \phi_n), s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_\kappa) \\ &\quad = (\delta\ \mathrm{thru}_{a,\kappa,k,r,V}(\phi_1, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_\kappa) \\ &\qquad\quad \ldots \\ &\qquad\quad \mathrm{thru}_{a,\kappa,k,r,V}(\phi_n, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_\kappa)) \end{aligned} \tag{6.11}$$

- for $n \in \mathbb{N}$, $g \in {F_2}^{(n+1)}$, $x_i \in X_a$ and $\phi_1, \ldots, \phi_n \in \mathrm{RHS}(F_2, \Delta, X_a, Y_\kappa)$

$$\begin{aligned} &\mathrm{thru}_{a,\kappa,k,r,V}(g\ x_i\ \phi_1\ \ldots\ \phi_n, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_\kappa) \\ &\quad = \mathrm{trans}_{g,k,r,V}(s_i, \mathrm{thru}_{a,\kappa,k,r,V}(\phi_1, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_\kappa), \\ &\qquad\qquad \ldots, \\ &\qquad\qquad \mathrm{thru}_{a,\kappa,k,r,V}(\phi_n, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_\kappa)) \end{aligned} \tag{6.12}$$

For every $a \in \mathbb{N}$, $f \in {F_1}^{(a+1)}, x_i \in X_k$, the mapping $\text{unfold}_{f(x_i),k,r,V}$ is defined with $h \in [a]$, $g \in F_2$, $S \subseteq ([a]{\times}F_2)$, $s_1, \ldots, s_a \in \text{RHS}(F_1, \Delta, X_k, Y_r)$ and $\varrho_{g_1,1}, \ldots, \varrho_{g_\mu,\kappa_\mu} \in \text{RHS}_{k,r,V}$ as follows:

- if $(h, g) \in S$

$$\text{unfold}_{f(x_i),k,r,V}(h, g, S, s_1, \ldots, s_a, \varrho_{g_1,1}, \ldots, \varrho_{g_\mu,\kappa_\mu}) = \text{nil} \tag{6.13}$$

- if $(h, g) \notin S$, $\kappa \in \mathbb{N}$ and $g \in {F_2}^{(\kappa+1)}$

$$\begin{aligned}
&\text{unfold}_{f(x_i),k,r,V}(h, g, S, s_1, \ldots, s_a, \varrho_{g_1,1}, \ldots, \varrho_{g_\mu,\kappa_\mu})\\
&= \text{trans}_{g,k,r,V}(s_h,\\
&\qquad ((h_f, 1_g)\; x_i\; \text{unfold}_{f(x_i),k,r,V}(1, g_1, S \cup \{(h, g)\}, s_1, \ldots, s_a, \varrho_{g_1,1}, \ldots, \varrho_{g_\mu,\kappa_\mu})\\
&\qquad\qquad \ldots\\
&\qquad\qquad \text{unfold}_{f(x_i),k,r,V}(a, g_\mu, S \cup \{(h, g)\}, s_1, \ldots, s_a, \varrho_{g_1,1}, \ldots, \varrho_{g_\mu,\kappa_\mu})\\
&\qquad\qquad \varrho_{g_1,1}\; \cdots\; \varrho_{g_\mu,\kappa_\mu}),\\
&\qquad \ldots,\\
&\qquad ((h_f, \kappa_g)\; x_i\; \text{unfold}_{f(x_i),k,r,V}(1, g_1, S \cup \{(h, g)\}, s_1, \ldots, s_a, \varrho_{g_1,1}, \ldots, \varrho_{g_\mu,\kappa_\mu})\\
&\qquad\qquad \ldots\\
&\qquad\qquad \text{unfold}_{f(x_i),k,r,V}(a, g_\mu, S \cup \{(h, g)\}, s_1, \ldots, s_a, \varrho_{g_1,1}, \ldots, \varrho_{g_\mu,\kappa_\mu})\\
&\qquad\qquad \varrho_{g_1,1}\; \cdots\; \varrho_{g_\mu,\kappa_\mu}))
\end{aligned} \tag{6.14}$$

Using the mappings defined above, we additionally define for every $k, r \in \mathbb{N}$ and $s \in \text{RHS}(F_1, \Delta, X_k, Y_r)$, the following set of mappings by simultaneous recursion:

$$\{\, \text{ctx}_{g,s,k,r} : \{\, p \in \text{occ}(s) \mid \text{label}_s(p) \notin X_k \,\} \times [\kappa] \longrightarrow \text{RHS}_{k,r,Z_{F_2}} \mid \kappa \in \mathbb{N},\, g \in {F_2}^{(\kappa+1)} \,\}$$

For every $\kappa \in \mathbb{N}$, $g \in {F_2}^{(\kappa+1)}$, the mapping $\text{ctx}_{g,s,k,r}$ is defined with $l \in [\kappa]$ as follows:

- $$\text{ctx}_{g,s,k,r}(\varepsilon, l) = z_{g,l} \tag{6.15}$$

- in case $\text{label}_s(p) = \delta$ with $p \in \text{occ}(s)$, $a \in \mathbb{N}$, $\delta \in \Delta^{(a)}$ and $i \in [a]$, if, with $\kappa' \in \mathbb{N}$, $g' \in {F_2}^{(\kappa'+1)}$ and $\phi_1, \ldots, \phi_\kappa \in \text{RHS}(F_2, \Delta, X_a, Y_{\kappa'})$, the only occurrence of a $(g\; x_i\; \ldots)$ call in the δ-equations of the weakly single-use M_2' looks as follows:

$$g'\; (\delta\; x_1\; \ldots\; x_a)\; y_1\; \ldots\; y_{\kappa'} = \cdots (g\; x_i\; \phi_1\; \ldots \phi_\kappa) \cdots,$$

  then

$$\begin{aligned}
\text{ctx}_{g,s,k,r}(p.i, l) = \text{thru}_{a,\kappa',k,r,Z_{F_2}}(&\phi_l, s|_{p.1}, \ldots, s|_{p.a},\\
&\text{ctx}_{g',s,k,r}(p, 1), \ldots, \text{ctx}_{g',s,k,r}(p, \kappa'))
\end{aligned} \tag{6.16}$$

  If no such call exists in the δ-rules of M_2', then

$$\text{ctx}_{g,s,k,r}(p.i, l) = \text{nil} \tag{6.17}$$

- in case $\mathrm{label}_s(p) = f$ with $p \in \mathrm{occ}(s)$, $a \in \mathbb{N}$, $f \in {F_1}^{(a+1)}$, $\mathrm{label}_s(p.1) = x_b \in X_k$ and $2 \le i \le a+1$:

$$
\begin{aligned}
&\mathrm{ctx}_{g,s,k,r}(p.i, l)\\
&\quad = ((i-1)_f, l_g)\; x_b\\
&\qquad\qquad \mathrm{unfold}_{f(x_b),k,r,Z_{F_2}}(1, g_1, \{(i-1, g)\}, s|_{p.2}, \ldots, s|_{p.a+1}, \mathrm{ctx}_{g_1,s,k,r}(p,1),\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots,\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{ctx}_{g_\mu,s,k,r}(p,\kappa_\mu))\\
&\qquad\qquad \ldots\\
&\qquad\qquad \mathrm{unfold}_{f(x_b),k,r,Z_{F_2}}(a, g_\mu, \{(i-1, g)\}, s|_{p.2}, \ldots, s|_{p.a+1}, \mathrm{ctx}_{g_1,s,k,r}(p,1),\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \ldots,\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{ctx}_{g_\mu,s,k,r}(p,\kappa_\mu))\\
&\qquad\qquad \mathrm{ctx}_{g_1,s,k,r}(p,1)\ \ldots\ \mathrm{ctx}_{g_\mu,s,k,r}(p,\kappa_\mu)
\end{aligned}
\tag{6.18}
$$

□

Theorem 6.16 (Correctness of Construction 6.15). *Let M_1 and M_2' be macro tree transducers, such that M_1 is non-copying and M_2' is weakly single-use, and, furthermore, let $M = \mathfrak{C}_{6.15}(M_1, M_2')$ be the macro tree transducer constructed by Construction 6.15 applied to M_1 and M_2'. Then*

$$\tau_{M_1} \circ \tau_{M_2'} = \tau_M .$$

Proof. The proof of the above theorem can be found in the appendix of [Voi01]. □

Example 6.17 (Sorting the stripes (3)). Let $M_{\mathrm{usa},1}$ be the non-copying macro tree transducer of Example 6.12 and $M'_{\mathrm{usa},2}$ be the weakly single-use (yield) macro tree transducer of Example 6.14. Then $\mathfrak{C}_{6.15}(M_{\mathrm{usa},1}, M'_{\mathrm{usa},2}) = M'_{\mathrm{usa}}$, which is [78]

$$
\begin{aligned}
M'_{\mathrm{usa}} = (&\{(\mathrm{coll},\mathrm{app})^{(3)}, (1_{\mathrm{coll}}, 1_{\mathrm{app}})^{(3)}\},\ \{W^{(1)}, R^{(1)}, |^{(1)}, N^{(0)}\},\\
&((\mathrm{coll},\mathrm{app})\ x\ ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x\ N\ N)\ N), P)
\end{aligned}
$$

with equations:

$$
\begin{aligned}
&(\mathrm{coll},\mathrm{app})\ N && y_{1,\mathrm{app}}\ z_1 = y_{1,\mathrm{app}}\\
&(\mathrm{coll},\mathrm{app})\ (|\ x_1) && y_{1,\mathrm{app}}\ z_1 = y_{1,\mathrm{app}}\\
&(\mathrm{coll},\mathrm{app})\ (R\ x_1) && y_{1,\mathrm{app}}\ z_1 = R\ ((\mathrm{coll},\mathrm{app})\ x_1\ y_{1,\mathrm{app}}\ z_1)\\
&(\mathrm{coll},\mathrm{app})\ (W\ x_1) && y_{1,\mathrm{app}}\ z_1 = (\mathrm{coll},\mathrm{app})\ x_1\ (W\ y_{1,\mathrm{app}})\ z_1
\end{aligned}
$$

$$
\begin{aligned}
&(1_{\mathrm{coll}}, 1_{\mathrm{app}})\ N && y_{1,\mathrm{app}}\ z_{\mathrm{app},1} = && z_{\mathrm{app},1}\\
&(1_{\mathrm{coll}}, 1_{\mathrm{app}})\ (|\ x_1) && y_{1,\mathrm{app}}\ z_{\mathrm{app},1} = && (\mathrm{coll},\mathrm{app})\ x_1\ ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ N\ z_{\mathrm{app},1})\ z_{\mathrm{app},1}\\
&(1_{\mathrm{coll}}, 1_{\mathrm{app}})\ (R\ x_1) && y_{1,\mathrm{app}}\ z_{\mathrm{app},1} = && (1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ N\ z_{\mathrm{app},1}\\
&(1_{\mathrm{coll}}, 1_{\mathrm{app}})\ (W\ x_1) && y_{1,\mathrm{app}}\ z_{\mathrm{app},1} = && (1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ N\ z_{\mathrm{app},1}.
\end{aligned}
$$

[78]Strictly speaking, M'_{usa} should also have the constructor SUB and equations at SUB, but since SUB does not appear in the input and output, we dropped that constructor from the specification. This is also evident from [Voi01], where the input and output signatures can be different.

To illustrate the generation, we compute:

$$
\begin{array}{rl}
\multicolumn{2}{l}{(\mathrm{coll},\mathrm{app})\ (W\ x_1)\ y_{1,\mathrm{app}}\ z_1}\\
= & \mathrm{trans}_{\mathrm{app},1,1,\{z_1\}}((\mathrm{coll}\ x_1\ (W y_1)), z_1)\\
\stackrel{(6.8)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ \mathrm{unfold}_{\mathrm{coll}(x_1),1,1,\{z_1\}}(1, \mathrm{app}, \emptyset, (W\ y_1), z_1)\ z_1\\
\stackrel{(6.14)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ \mathrm{trans}_{\mathrm{app},1,1,\{z_1\}}((W\ y_1),\\
& \quad ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ \mathrm{unfold}_{\mathrm{coll}(x_1),1,1,\{z_1\}}(1, \mathrm{app}, \{(1,\mathrm{app})\}, (W\ y_1), z_1)\ z_1))\ z_1\\
\stackrel{(6.9)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ \mathrm{thru}_{1,1,1,1,\{z_1\}}((W\ (\mathrm{app}\ x_1\ y_1)), y_1\\
& \quad ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ \mathrm{unfold}_{\mathrm{coll}(x_1),1,1,\{z_1\}}(1, \mathrm{app}, \{(1,\mathrm{app})\}, (W\ y_1), z_1)\ z_1))\ z_1\\
\stackrel{(6.11)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ (W\ \mathrm{thru}_{1,1,1,1,\{z_1\}}((\mathrm{app}\ x_1\ y_1), y_1\\
& \quad ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ \mathrm{unfold}_{\mathrm{coll}(x_1),1,1,\{z_1\}}(1, \mathrm{app}, \{(1,\mathrm{app})\}, (W\ y_1), z_1)\ z_1)))\ z_1\\
\stackrel{(6.12)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ (W\ \mathrm{trans}_{\mathrm{app},1,1,\{z_1\}}(y_1, \mathrm{thru}_{1,1,1,1,\{z_1\}}(y_1, y_1\\
& \quad ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ \mathrm{unfold}_{\mathrm{coll}(x_1),1,1,\{z_1\}}(1, \mathrm{app}, \{(1,\mathrm{app})\}, (W\ y_1), z_1)\ z_1))))\ z_1\\
\stackrel{(6.7)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ (W\ y_{1,\mathrm{app}})\ z_1
\end{array}
$$

and

$$
\begin{array}{rl}
\multicolumn{2}{l}{(1_{\mathrm{coll}}, 1_{\mathrm{app}})\ (\mid x_1)\ y_{1,\mathrm{app}}\ z_{\mathrm{app},1}}\\
= & \mathrm{ctx}_{\mathrm{app},(\mathrm{SUB}\ y_1\ (\mathrm{coll}\ x_1\ N)),1,1}(1,1)\\
\stackrel{(6.16)}{=} & \mathrm{thru}_{2,1,1,1,\{z_{\mathrm{app},1}\}}((\mathrm{app}\ x_2\ y_1), y_1, (\mathrm{coll}\ x_1\ N), \mathrm{ctx}_{\mathrm{app},(\mathrm{SUB}\ y_1\ (\mathrm{coll}\ x_1\ N)),1,1}(\varepsilon, 1))\\
\stackrel{(6.15)}{=} & \mathrm{thru}_{2,1,1,1,\{z_{\mathrm{app},1}\}}((\mathrm{app}\ x_2\ y_1), y_1, (\mathrm{coll}\ x_1\ N), z_{\mathrm{app},1})\\
\stackrel{(6.12)}{=} & \mathrm{trans}_{\mathrm{app},1,1,\{z_{\mathrm{app},1}\}}((\mathrm{coll}\ x_1\ N), \mathrm{thru}_{2,1,1,1,\{z_{\mathrm{app},1}\}}(y_1, y_1, (\mathrm{coll}\ x_1\ N), z_{\mathrm{app},1}))\\
\stackrel{(6.10)}{=} & \mathrm{trans}_{\mathrm{app},1,1,\{z_{\mathrm{app},1}\}}((\mathrm{coll}\ x_1\ N), z_{\mathrm{app},1})\\
\stackrel{(6.8)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ \mathrm{unfold}_{\mathrm{coll}(x_1),1,1,\{z_{\mathrm{app},1}\}}(1, \mathrm{app}, \emptyset, N, z_{\mathrm{app},1})\ z_{\mathrm{app},1}\\
\stackrel{(6.14)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ \mathrm{trans}_{\mathrm{app},1,1,\{z_{\mathrm{app},1}\}}(N, ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\\
& \quad \mathrm{unfold}_{\mathrm{coll}(x_1),1,1,\{z_{\mathrm{app},1}\}}(1, \mathrm{app}, \{(1,\mathrm{app})\}, N, z_{\mathrm{app},1})\ z_{\mathrm{app},1}))\ z_{\mathrm{app},1}\\
\stackrel{(6.13)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ \mathrm{trans}_{\mathrm{app},1,1,\{z_{\mathrm{app},1}\}}(N, ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ N\ z_{\mathrm{app},1}))\ z_{\mathrm{app},1}\\
\stackrel{(6.9)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ \mathrm{thru}_{0,1,1,1,\{z_{\mathrm{app},1}\}}(y_1, ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ N\ z_{\mathrm{app},1}))\ z_{\mathrm{app},1}\\
\stackrel{(6.10)}{=} & (\mathrm{coll},\mathrm{app})\ x_1\ ((1_{\mathrm{coll}}, 1_{\mathrm{app}})\ x_1\ N\ z_{\mathrm{app},1})\ z_{\mathrm{app},1}.
\end{array}
$$

Theorem 6.18 (Correctness of the extended indirect construction). *Let $M \in \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})$ be a 2-modular tree transducer, of which the first module is non-copying and the second module is a substitution module. Then there exists a macro tree transducer M', such that*

$$\tau_M = \tau_{M'}.$$

Proof. Let $(M_1, M_2) = \mathfrak{C}_{6.11}(M)$ and $M_2' = \mathfrak{C}_{6.13}(M_2)$ and, finally, $M' = \mathfrak{C}_{6.15}(M_1, M_2')$. By Lemma 6.11, Lemma 6.13 and Theorem 6.16 we gain

$$\tau_M \stackrel{\text{Lem. 6.11}}{=} \tau_{M_1} \circ \tau_{M_2} \stackrel{\text{Lem. 6.13}}{=} \tau_{M_1} \circ \tau_{M_2'} \stackrel{\text{Thm. 6.16}}{=} \tau_{M'}.$$

□

Theorem 6.19 (Characterization theorem).

$$(\, \tau_M \mid M \in \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})\,) = \mathrm{MAC} = (\, \tau_M \mid M \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})\,)$$

Proof.

$$(\, \tau_M \mid M \in \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})\,) \stackrel{\text{Thm. 6.18}}{\subseteq} \mathrm{MAC}$$

holds by Theorem 6.18,

$$\mathrm{MAC} \subseteq (\, \tau_M \mid M \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})\,)$$

was proven in [KGK01] and, since $\mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB}) \subset \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})$, immediately also

$$(\, \tau_M \mid M \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})\,) \subseteq (\, \tau_M \mid M \in \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})\,).$$

□

Lemma 6.20 (Composing the constructions $\mathfrak{C}_{6.11}$ and $\mathfrak{C}_{6.13}$). *Let*

$$M = (F, m, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})$$

with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ *for some* $\mathrm{mx} \in \mathbb{N}$. *Then there exists a non-copying macro tree transducer* M_1 *and a weakly single-use (yield) macro tree transducer* M_2'*, such that*

$$\tau_M = \tau_{M_1} \circ \tau_{M_2'}.$$

Proof. Let $(M_1, M_2) = \mathfrak{C}_{6.11}(M)$ and $M_2' = \mathfrak{C}_{6.13}(M_2)$. We want to provide a construction, such that $(M_1, M_2') = \mathfrak{C}_{6.20}(M)$. Obviously the first component M_1 is constructed according to Lemma 6.11, while the second is constructed according to Lemma 6.11 and Lemma 6.13.

The weakly single-use (yield) macro tree transducer M_2' is defined as

$$M_2' = (\{\mathrm{sub}^{(\mathrm{mx}+1)}\},\, \Delta \cup \{\mathrm{SUB}^{(\mathrm{mx}+1)}\},\, (\mathrm{sub}\, x\, \Pi_1\, \ldots\, \Pi_{\mathrm{mx}}),\, R_2'),$$

where $\mathrm{sub} \in m^{-1}(2)^{(\mathrm{mx}+1)}$ and R_2' contains

- for every $r \in \mathbb{N}$ and $\delta \in \Delta^{(r)} \setminus \Pi$ by Construction $\mathfrak{C}_{6.13}$ and Definition 4.1 the equation

$$\mathrm{sub}\, (\delta\, x_1\, \ldots\, x_r)\, y_1\, \ldots\, y_{\mathrm{mx}} = \delta\, (\mathrm{sub}\, x_1\, y_1\, \ldots\, y_{\mathrm{mx}})\, \ldots\, (\mathrm{sub}\, x_r\, y_1\, \ldots\, y_{\mathrm{mx}}), \quad (6.19)$$

- for every $j \in [\mathrm{mx}]$ and $\Pi_j \in \Pi$ by Construction $\mathfrak{C}_{6.13}$ and Definition 4.1 the equation

$$\mathrm{sub}\, \Pi_j\, y_1\, \ldots\, y_{\mathrm{mx}} = y_j \quad (6.20)$$

- and by Construction $\mathfrak{C}_{6.13}$ the equation

$$\mathrm{sub}(\mathrm{SUB} x_1 \dots x_{\mathrm{mx}+1}) y_1 \dots y_{\mathrm{mx}} = \mathrm{sub} x_1 (\mathrm{sub} x_2 y_1 \dots y_{\mathrm{mx}}) \dots (\mathrm{sub} x_{\mathrm{mx}+1} y_1 \dots y_{\mathrm{mx}}). \tag{6.21}$$

The SUB-constructor is added to Δ in Construction $\mathfrak{C}_{6.11}$ and preserved in Construction $\mathfrak{C}_{6.13}$. Furthermore, the substitution variables remain invariant over both constructions and, together with the only function symbol of the substitution module created in Construction $\mathfrak{C}_{6.11}$, this uniquely determines the substitution module of M_2. Since the only reference to the interpretation module of M_2 in Construction $\mathfrak{C}_{6.13}$ is the interpretation mapping which maps SUB to sub; defined to be such in Construction $\mathfrak{C}_{6.11}$; we can identify the SUB-constructor without this reference.

Apparently, M_2' is a weakly single-use macro tree transducer and

$$\tau_{M_1} \circ \tau_{M_2'} \overset{\text{Lem. 6.13}}{=} \tau_{M_1} \circ \tau_{M_2} \overset{\text{Lem. 6.11}}{=} \tau_M.$$

□

Construction 6.21 (Composing $\mathfrak{C}_{6.20}$ and $\mathfrak{C}_{6.15}$). Let $M' = (F', m, \Delta, e', R') \in \mathrm{ModTT}(\mathrm{MAC_{nc}}, \mathrm{SUB})$ with substitution variables $\Pi = \{\Pi_1, \dots, \Pi_{\mathrm{mx}}\}$ for some $\mathrm{mx} \in \mathbb{N}$. Additionally, let $(M_1, M_2) = \mathfrak{C}_{6.20}(M')$ with $M_1 = (m^{-1}(1), \Delta', e', R_1)$ and

$$M_2 = (\{\mathrm{sub}^{(\mathrm{mx}+1)}\},\ \Delta',\ (\mathrm{sub}\, x\, \Pi_1\ \dots\ \Pi_{\mathrm{mx}}),\ R_2)$$

be the macro tree transducers created by Lemma 6.20 with $\Delta' = \Delta \cup \{\mathrm{SUB}^{(\mathrm{mx}+1)}\}$. We construct the macro tree transducer $M = (F, \Delta', e, R)$ as follows:

Let $\mu = \mathrm{card}(\{\mathrm{sub}\}) = 1$ and for $1 \le i \le \mu$, let $\kappa_i \in \mathbb{N}$ be such that $g_i \in \{\mathrm{sub}^{(\mathrm{mx}+1)}\}^{(\kappa_i+1)}$, thus $g_1 = \mathrm{sub}$ and $\kappa_1 = \mathrm{mx}$. Additionally, let $\mathrm{nil} \in \Delta^{(0)}$ be some dummy output symbol and $\{z_{g_1,1}, \dots, z_{g_1,\kappa_1}, \dots, z_{g_\mu,1}, \dots, z_{g_\mu,\kappa_\mu}\} = \{z_1, \dots, z_{\mathrm{mx}}\} = Z_{\mathrm{mx}}$. Then the components of M are obtained as follows:

- the new set of function symbols:

$$\begin{aligned}
F \;&=\; \{\, (f,g)^{(r*\mu+\kappa+1)} \mid r,\kappa \in \mathbb{N},\ f \in m^{-1}(1)^{(r+1)},\ g \in \{\mathrm{sub}^{(\mathrm{mx}+1)}\}^{(\kappa+1)} \,\} \cup \\
&\quad \cup \{\, (h_f, l_g)^{(r*\mu+\mathrm{card}(Z_{\mathrm{mx}})+1)} \mid r,\kappa \in \mathbb{N}, f \in m^{-1}(1)^{(r+1)},\ h \in [r], \\
&\qquad\qquad g \in \{\mathrm{sub}^{(\mathrm{mx}+1)}\}^{(\kappa+1)},\ l \in [\kappa] \,\} \\
&=\; \{\, (f,\mathrm{sub})^{(r+\mathrm{mx}+1)} \mid r \in \mathbb{N},\ f \in m^{-1}(1)^{(r+1)} \,\} \cup \\
&\quad \cup \{\, (h_f, l_{\mathrm{sub}})^{(r+\mathrm{mx}+1)} \mid r \in \mathbb{N},\ f \in m^{-1}(1)^{(r+1)},\ h \in [r],\ l \in [\mathrm{mx}] \,\} \\
&\cong\; \{\, f^{(r+\mathrm{mx}+1)} \mid r \in \mathbb{N},\ f \in m^{-1}(1)^{(r+1)} \,\} \cup && (6.22) \\
&\quad \cup \{\, (f,h,l)^{(r+\mathrm{mx}+1)} \mid r \in \mathbb{N},\ f \in m^{-1}(1)^{(r+1)},\ h \in [r],\ l \in [\mathrm{mx}] \,\} && (6.23)
\end{aligned}$$

- the new initial expression:

$$\begin{aligned}
e \;&=\; \mathrm{thru}_{1,0,1,0,\emptyset}(\mathrm{sub}\, x\, \Pi_1\ \dots\ \Pi_{\mathrm{mx}}, e') \\
&\overset{(6.12)}{=}\; \mathrm{trans}_{\mathrm{sub},1,0,\emptyset}(e', \mathrm{thru}_{1,0,1,0,\emptyset}(\Pi_1, e'), \dots, \mathrm{thru}_{1,0,1,0,\emptyset}(\Pi_{\mathrm{mx}}, e')) \\
&\overset{(6.11)}{=}\; \mathrm{trans}_{\mathrm{sub},1,0,\emptyset}(e', \Pi_1, \dots, \Pi_{\mathrm{mx}}) \\
&\overset{\text{Def.}}{=}\; \mathrm{trans}_{1,0,\emptyset}(e', \Pi_1, \dots, \Pi_{\mathrm{mx}})
\end{aligned}$$

- the new set of equations R contains:
  - for $r, k \in \mathbb{N}$, $f \in m^{-1}(1)^{(r+1)}$ and $\delta \in \Delta'^{(k)}$, the equation:

$$\begin{aligned} f\,(\delta\, x_1\,\ldots\,x_k)\,y_1\,\ldots\,y_r\,z_1\,\ldots\,z_{\mathrm{mx}} &= \mathrm{trans}_{\mathrm{sub},k,r,Z_{\mathrm{mx}}}(\mathrm{rhs}_{M_1}(f,\delta), z_1, \ldots, z_{\mathrm{mx}}) \\ &\stackrel{\text{Def.}}{=} \mathrm{trans}_{k,r,Z_{\mathrm{mx}}}(\mathrm{rhs}_{M_1}(f,\delta), z_1, \ldots, z_{\mathrm{mx}}) \end{aligned}$$

    if R_1 contains the equation $f\,(\delta\, x_1\,\ldots\,x_k)\,y_1\,\ldots\,y_r = \mathrm{rhs}_{M_1}(f,\delta)$.
  - for $r, k \in \mathbb{N}$, $f \in m^{-1}(1)^{(r+1)}$, $h \in [r]$, $l \in [\mathrm{mx}]$ and $\delta \in \Delta'^{(k)}$, the equation:

$$(f,h,l)\,(\delta\, x_1\,\ldots\,x_k)\,y_1\,\ldots\,y_r\,z_1\,\ldots\,z_{\mathrm{mx}} = \psi,$$

    if R_1 contains the equation $f\,(\delta\, x_1 \ldots x_k)\, y_1 \ldots y_r = \mathrm{rhs}_{M_1}(f,\delta)$, where $\psi = \mathrm{nil}$, if $\mathrm{rhs}_{M_1}(f,\delta) \in \mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r)$ does not contain the context variable y_h; otherwise

$$\psi = \mathrm{ctx}_{\mathrm{sub},\mathrm{rhs}_{M_1}(f,\delta),k,r}(p,l) \stackrel{\text{Def.}}{=} \mathrm{ctx}_{\mathrm{rhs}_{M_1}(f,\delta),k,r}(p,l)$$

    with $p \in \mathrm{occ}(\mathrm{rhs}_{M_1}(f,\delta))$ as the unique occurrence of y_h in $\mathrm{rhs}_{M_1}(f,\delta)$ (M_1 is non-copying, thus $\#_{y_h}(\mathrm{rhs}_{M_1}(f,\delta)) \leq 1$!)

For every $k, r \in \mathbb{N}$ and finite set of variables V, the following set of mappings is defined by simultaneous recursion:

$$\begin{aligned} \{\mathrm{trans}_{k,r,V}\} \,\cup\, &\{\,\mathrm{unfold}_{f(x_i),k,r,V} \mid f \in m^{-1}(1),\, x_i \in X_k\,\} \cup \\ \cup\, &\{\,\mathrm{thru}_{a,\mathrm{mx},k,r,V} \mid a \in \mathrm{ar}(\Delta')\,\} \cup \{\mathrm{thru}_{1,0,k,r,V}\} \end{aligned}$$

All mappings are modified to output function symbols of F as specified in lines 6.22 and 6.23, but otherwise they shall be equivalent to the original mappings of Construction 6.15. We use $\stackrel{\text{Def.}}{=}$ to denote this particular equivalence. With $\mathrm{RHS}_{k,r,V} = \mathrm{RHS}(F, \Delta', X_k, Y_r \cup V)$, these have the following types:

- $\mathrm{trans}_{k,r,V} : \mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r) \times (\mathrm{RHS}_{k,r,V})^{\mathrm{mx}} \longrightarrow \mathrm{RHS}_{k,r,V}$ [79]
- $\mathrm{thru}_{1,0,k,r,V} : \mathrm{RHS}(\{\mathrm{sub}^{(\mathrm{mx}+1)}\}, \Delta', \{x_1\}, \emptyset) \times \mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r) \longrightarrow \mathrm{RHS}_{k,r,V}$,
- for $a \in \mathrm{ar}(\Delta')$:

$$\begin{aligned} \mathrm{thru}_{a,\mathrm{mx},k,r,V} : \mathrm{RHS}(\{\mathrm{sub}^{(\mathrm{mx}+1)}\}, \Delta', X_a, Y_{\mathrm{mx}}) \times (\mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r))^a \times \\ \times (\mathrm{RHS}_{k,r,V})^{\mathrm{mx}} \longrightarrow \mathrm{RHS}_{k,r,V} \end{aligned}$$

- and for $a \in \mathbb{N}$, $f \in m^{-1}(1)^{(a+1)}$ and $x_i \in X_k$ [80]:

$$\begin{aligned} \mathrm{unfold}_{f(x_i),k,r,V} : [a] \times \mathfrak{P}([a]) \times (\mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r))^a \times (\mathrm{RHS}_{k,r,V})^{\mathrm{mx}} \\ \longrightarrow \mathrm{RHS}_{k,r,V} \end{aligned}$$

[79] Compared to Construction 6.15, we simply dropped the first subscript, since it is always sub.
[80] Since $[a] \times \{\mathrm{sub}\} \cong [a]$, we used this isomorphism twice to gain the new type.

The mapping $\mathrm{trans}_{k,r,V} \stackrel{\text{Def.}}{=} \mathrm{trans}_{\mathrm{sub},k,r,V}$ is defined with $\varrho_1, \ldots, \varrho_{\mathrm{mx}} \in \mathrm{RHS}_{k,r,V}$ as follows:

- for $y_h \in Y_r$

$$\mathrm{trans}_{k,r,V}(y_h, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \stackrel{\text{Def.}}{=} \mathrm{trans}_{\mathrm{sub},k,r,V}(y_h, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) = y_h \tag{6.24}$$

- for $a \in \mathbb{N}$, $\delta \in \Delta'^{(a)}$, $s_1, \ldots, s_a \in \mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r)$ and equation

$$\mathrm{sub}\ (\delta\ x_1\ \ldots\ x_a)\ y_1\ \ldots\ y_{\mathrm{mx}} = \mathrm{rhs}_{M_2}(\mathrm{sub}, \delta)$$

in R_2, with $\mathrm{rhs}_{M_2}(\mathrm{sub}, \delta) \in \mathrm{RHS}(\{\mathrm{sub}^{(\mathrm{mx}+1)}\}, \Delta', X_a, Y_{\mathrm{mx}})$

$$\begin{aligned}
& \mathrm{trans}_{k,r,V}((\delta\ s_1\ \ldots\ s_a), \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\
\stackrel{\text{Def.}}{=}\ & \mathrm{trans}_{\mathrm{sub},k,r,V}((\delta\ s_1\ \ldots\ s_a), \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\
\stackrel{(6.9)}{=}\ & \mathrm{thru}_{a,\mathrm{mx},k,r,V}(\mathrm{rhs}_{M_2}(\mathrm{sub}, \delta), s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}})
\end{aligned}$$

We perform the following case distinction.

- let $\delta \in \Delta \setminus \Pi$, then by Equation (6.19) we continue

$$\begin{aligned}
& \mathrm{thru}_{a,\mathrm{mx},k,r,V}(\mathrm{rhs}_{M_2}(\mathrm{sub}, \delta), s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\
\stackrel{(6.19)}{=}\ & \mathrm{thru}_{a,\mathrm{mx},k,r,V}((\delta\ (\mathrm{sub}\ x_1\ y_1\ \ldots\ y_{\mathrm{mx}})\ \ldots \\
& \qquad (\mathrm{sub}\ x_a\ y_1\ \ldots\ y_{\mathrm{mx}})), s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\
\stackrel{(6.11)}{=}\ & (\delta\ \mathrm{thru}_{a,\mathrm{mx},k,r,V}(\mathrm{sub}\ x_1\ y_1\ \ldots\ y_{\mathrm{mx}}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}})\ \ldots \\
& \quad \mathrm{thru}_{a,\mathrm{mx},k,r,V}(\mathrm{sub}\ x_a\ y_1\ \ldots\ y_{\mathrm{mx}}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}})) \\
\stackrel{(6.12)}{=}\ & (\delta\ \mathrm{trans}_{\mathrm{sub},k,r,V}(s_1, \mathrm{thru}_{a,\mathrm{mx},k,r,V}(y_1, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}), \ldots, \\
& \qquad \mathrm{thru}_{a,\mathrm{mx},k,r,V}(y_{\mathrm{mx}}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))\ \ldots \\
& \quad \mathrm{trans}_{\mathrm{sub},k,r,V}(s_a, \mathrm{thru}_{a,\mathrm{mx},k,r,V}(y_1, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}), \ldots, \\
& \qquad \mathrm{thru}_{a,\mathrm{mx},k,r,V}(y_{\mathrm{mx}}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))) \\
\stackrel{(6.10)}{=}\ & (\delta\ \mathrm{trans}_{\mathrm{sub},k,r,V}(s_1, \varrho_1, \ldots, \varrho_{\mathrm{mx}})\ \ldots\ \mathrm{trans}_{\mathrm{sub},k,r,V}(s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}})) \\
\stackrel{\text{Def.}}{=}\ & (\delta\ \mathrm{trans}_{k,r,V}(s_1, \varrho_1, \ldots, \varrho_{\mathrm{mx}})\ \ldots\ \mathrm{trans}_{k,r,V}(s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))
\end{aligned}$$

- let $\delta = \Pi_j$ for some $j \in [\mathrm{mx}]$, then by Equation (6.20) we continue

$$\begin{aligned}
& \mathrm{thru}_{a,\mathrm{mx},k,r,V}(\mathrm{rhs}_{M_2}(\mathrm{sub}, \Pi_j), s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\
\stackrel{(6.20)}{=}\ & \mathrm{thru}_{a,\mathrm{mx},k,r,V}(y_j, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\
\stackrel{(6.10)}{=}\ & \varrho_j
\end{aligned}$$

– let $\delta = \text{SUB}$, then by Equation (6.21) we continue

$$\begin{aligned}
&\text{thru}_{a,\text{mx},k,r,V}(\text{rhs}_{M_2}(\text{sub},\text{SUB}), s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}})\\
\overset{(6.21)}{=}\ &\text{thru}_{a,\text{mx},k,r,V}(\text{sub}\ x_1\ (\text{sub}\ x_2\ y_1\ \ldots\ y_{\text{mx}})\ \ldots\\
&\qquad\qquad\qquad (\text{sub}\ x_a\ y_1\ \ldots\ y_{\text{mx}}), s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}})\\
\overset{(6.12)}{=}\ &\text{trans}_{\text{sub},k,r,V}(s_1, \text{thru}_{a,\text{mx},k,r,V}(\text{sub}\ x_2\ y_1\ \ldots\ y_{\text{mx}}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}}),\\
&\qquad \ldots,\\
&\qquad \text{thru}_{a,\text{mx},k,r,V}(\text{sub}\ x_a\ y_1\ \ldots\ y_{\text{mx}}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}}))\\
\overset{(6.12)}{=}\ &\text{trans}_{\text{sub},k,r,V}(s_1,\\
&\quad \text{trans}_{\text{sub},k,r,V}(s_2, \text{thru}_{a,\text{mx},k,r,V}(y_1, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}}), \ldots,\\
&\qquad\qquad \text{thru}_{a,\text{mx},k,r,V}(y_{\text{mx}}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}})), \ldots,\\
&\quad \text{trans}_{\text{sub},k,r,V}(s_a, \text{thru}_{a,\text{mx},k,r,V}(y_1, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}}), \ldots,\\
&\qquad\qquad \text{thru}_{a,\text{mx},k,r,V}(y_{\text{mx}}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}})))\\
\overset{(6.10)}{=}\ &\text{trans}_{\text{sub},k,r,V}(s_1, \text{trans}_{\text{sub},k,r,V}(s_2, \varrho_1, \ldots, \varrho_{\text{mx}}), \ldots,\\
&\qquad \text{trans}_{\text{sub},k,r,V}(s_a, \varrho_1, \ldots, \varrho_{\text{mx}}))\\
\overset{\text{Def.}}{=}\ &\text{trans}_{k,r,V}(s_1, \text{trans}_{k,r,V}(s_2, \varrho_1, \ldots, \varrho_{\text{mx}}), \ldots, \text{trans}_{k,r,V}(s_a, \varrho_1, \ldots, \varrho_{\text{mx}}))
\end{aligned}$$

- for $a \in \mathbb{N}$, $f \in m^{-1}(1)^{(a+1)}$, $x_i \in X_k$ and $s_1, \ldots, s_a \in \text{RHS}(m^{-1}(1), \Delta', X_k, Y_r)$

$$\begin{aligned}
&\text{trans}_{k,r,V}((f\ x_i\ s_1\ \ldots\ s_a), \varrho_1, \ldots, \varrho_{\text{mx}})\\
\overset{\text{Def.}}{=}\ &\text{trans}_{\text{sub},k,r,V}(f\ x_i\ s_1\ \ldots\ s_a, \varrho_1, \ldots, \varrho_{\text{mx}})\\
\overset{(6.8)}{=}\ &f\ x_i\ \text{unfold}_{f(x_i),k,r,V}(1, \emptyset, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}})\\
&\quad \ldots\\
&\quad \text{unfold}_{f(x_i),k,r,V}(a, \emptyset, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}})\\
&\quad \varrho_1\ \ldots\ \varrho_{\text{mx}}
\end{aligned}$$

For every $a \in \text{ar}(\Delta')$ the mapping $\text{thru}_{a,\text{mx},k,r,V}$ is defined with $\varrho_1, \ldots, \varrho_{\text{mx}} \in \text{RHS}_{k,r,V}$ and $s_1, \ldots, s_a \in \text{RHS}(m^{-1}(1), \Delta', X_k, Y_r)$ as follows:

- for $y_l \in Y_{\text{mx}}$

$$\text{thru}_{a,\text{mx},k,r,V}(y_l, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}}) = \varrho_l$$

- for $n \in \mathbb{N}$, $\delta \in \Delta'^{(n)}$ and $\phi_1, \ldots, \phi_n \in \text{RHS}(\{\text{sub}^{(\text{mx}+1)}\}, \Delta', X_a, Y_{\text{mx}})$

$$\begin{aligned}
&\text{thru}_{a,k,r,V}((\delta\ \phi_1, \ldots, \phi_n), s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}}) =\\
&\qquad \delta\ \text{thru}_{a,\text{mx},k,r,V}(\phi_1, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}})\\
&\qquad\ \ldots\\
&\qquad\ \text{thru}_{a,\text{mx},k,r,V}(\phi_n, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\text{mx}})
\end{aligned}$$

- for $x_i \in X_a$ and $\phi_1, \dots, \phi_{\mathrm{mx}} \in \mathrm{RHS}(\{\mathrm{sub}^{(\mathrm{mx}+1)}\}, \Delta', X_a, Y_{\mathrm{mx}})$

$$\begin{aligned}
&\mathrm{thru}_{a,\mathrm{mx},k,r,V}((\mathrm{sub}\, x_i\, \phi_1\, \dots\, \phi_{\mathrm{mx}}), s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\
&\quad = \mathrm{trans}_{\mathrm{sub},k,r,V}(s_i, \mathrm{thru}_{a,\mathrm{mx},k,r,V}(\phi_1, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\
&\qquad\qquad \dots \\
&\qquad\qquad \mathrm{thru}_{a,\mathrm{mx},k,r,V}(\phi_{\mathrm{mx}}, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}})) \\
&\quad = \mathrm{trans}_{k,r,V}(s_i, \mathrm{thru}_{a,\mathrm{mx},k,r,V}(\phi_1, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\
&\qquad\qquad \dots \\
&\qquad\qquad \mathrm{thru}_{a,\mathrm{mx},k,r,V}(\phi_{\mathrm{mx}}, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}))
\end{aligned}$$

The mapping $\mathrm{thru}_{1,0,k,r,V}$ is defined with $s_1 \in \mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r)$ as follows:

- for $n \in \mathbb{N}$, $\delta \in \Delta'^{(n)}$ and $\phi_1, \dots, \phi_n \in \mathrm{RHS}(\{\mathrm{sub}^{(\mathrm{mx}+1)}\}, \Delta', \{x_1\}, \emptyset)$

$$\mathrm{thru}_{1,0,k,r,V}((\delta\, \phi_1, \dots, \phi_n), s_1) = (\delta\, \mathrm{thru}_{1,0,k,r,V}(\phi_1, s_1)\, \dots\, \mathrm{thru}_{1,0,k,r,V}(\phi_n, s_1))$$

- for $\phi_1, \dots, \phi_{\mathrm{mx}} \in \mathrm{RHS}(\{\mathrm{sub}^{(\mathrm{mx}+1)}\}, \Delta', \{x_1\}, \emptyset)$

$$\begin{aligned}
\mathrm{thru}_{1,0,k,r,V}((\mathrm{sub}\, x_1\, \phi_1\, \dots\, \phi_{\mathrm{mx}}), s_1) = \mathrm{trans}_{\mathrm{sub},k,r,V}(s_1, &\mathrm{thru}_{1,0,k,r,V}(\phi_1, s_1) \\
&\dots \\
&\mathrm{thru}_{1,0,k,r,V}(\phi_n, s_1))
\end{aligned}$$

For every $a \in \mathbb{N}$, $f \in m^{-1}(1)^{(a+1)}, x_i \in X_k$, the mapping $\mathrm{unfold}_{f(x_i),k,r,V}$ is defined with $h \in [a]$, $S \subseteq [a]$, $s_1, \dots, s_a \in \mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r)$ and $\varrho_1, \dots, \varrho_{\mathrm{mx}} \in \mathrm{RHS}_{k,r,V}$ as follows:

- if $h \in S$

$$\mathrm{unfold}_{f(x_i),k,r,V}(h, S, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) = \mathrm{nil} \tag{6.25}$$

- otherwise

$$\begin{aligned}
&\mathrm{unfold}_{f(x_i),k,r,V}(h, S, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\
&\quad = \mathrm{trans}_{k,r,V}(s_h, (f, h, 1)\, x_i \\
&\qquad\qquad \mathrm{unfold}_{f(x_i),k,r,V}(1, S \cup \{h\}, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\
&\qquad\qquad \dots \\
&\qquad\qquad \mathrm{unfold}_{f(x_i),k,r,V}(a, S \cup \{h\}, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\
&\qquad\qquad \varrho_1\, \dots\, \varrho_{\mathrm{mx}}, \dots, \\
&\qquad\quad (f, h, \mathrm{mx})\, x_i \\
&\qquad\qquad \mathrm{unfold}_{f(x_i),k,r,V}(1, S \cup \{h\}, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\
&\qquad\qquad \dots \\
&\qquad\qquad \mathrm{unfold}_{f(x_i),k,r,V}(a, S \cup \{h\}, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\
&\qquad\qquad \varrho_1\, \dots\, \varrho_{\mathrm{mx}})
\end{aligned}$$

Using the mappings defined above, we additionally define for every $k, r \in \mathbb{N}$ and $s \in \mathrm{RHS}(m^{-1}(1), \Delta', X_k, Y_r)$, the following mapping by simultaneous recursion:

$$\mathrm{ctx}_{s,k,r} : \{ p \in \mathrm{occ}(s) \mid \mathrm{label}_s(p) \notin X_k \} \times [\mathrm{mx}] \longrightarrow \mathrm{RHS}_{k,r,Z_{\mathrm{mx}}}$$

The mapping $\mathrm{ctx}_{s,k,r}$ is defined with $l \in [\mathrm{mx}]$ as follows [81]:

- $\mathrm{ctx}_{s,k,r}(\varepsilon, l) = z_l$
- in case $\mathrm{label}_s(p) = \delta$ with $p \in \mathrm{occ}(s)$, $a \in \mathbb{N}$, $\delta \in \Delta'^{(a)}$ and $i \in [a]$, if, $\phi_1, \ldots, \phi_{\mathrm{mx}} \in \mathrm{RHS}(\{\mathrm{sub}^{(\mathrm{mx}+1)}\}, \Delta', X_a, Y_{\mathrm{mx}})$, the only occurrence of a (sub x_i ...)-call in the δ-equations of the weakly single-use macro tree transducer M_2 looks as follows:

  $$\mathrm{sub}\,(\delta\, x_1 \,\ldots\, x_a)\, y_1 \,\ldots\, y_{\mathrm{mx}} = \cdots (\mathrm{sub}\, x_i\, \phi_1 \,\ldots \phi_{\mathrm{mx}}) \cdots ,$$

  then

  $$\mathrm{ctx}_{s,k,r}(p.i, l) = \mathrm{thru}_{a,\mathrm{mx},k,r,Z_{\mathrm{mx}}}(\phi_l, s|_{p.1}, \ldots, s|_{p.a}, \mathrm{ctx}_{s,k,r}(p, 1), \ldots, \mathrm{ctx}_{s,k,r}(p, \mathrm{mx})).$$

  We perform the following case distinction:

  - $\delta \in \Delta^{(0)}$ is not applicable,
  - let $\delta \in \Delta^{(a)}$ with $a \in \mathbb{N}_+$, then for every $j \in [\mathrm{mx}]$ by Equation (6.19) $\phi_j = y_j$ and we continue

    $$\begin{aligned} & \mathrm{thru}_{a,\mathrm{mx}k,r,Z_{\mathrm{mx}}}(y_l, s|_{o.1}, \ldots, s|_{o.a}, \mathrm{ctx}_{s,k,r}(p, 1), \ldots, \mathrm{ctx}_{s,k,r}(p, \mathrm{mx})) \\ \overset{(6.10)}{=} \; & \mathrm{ctx}_{s,k,r}(p, l) \end{aligned}$$

  - let $\delta = \mathrm{SUB}$, then for every $j \in [\mathrm{mx}]$
    * if $i = 1$, then by Equation (6.21) $\phi_j = \mathrm{sub}\, x_{j+1}\, y_1 \,\ldots\, y_{\mathrm{mx}}$ and

      $$\begin{aligned} & \mathrm{thru}_{a,\mathrm{mx},k,r,Z_{\mathrm{mx}}}((\mathrm{sub}\, x_{l+1}\, y_1 \,\ldots y_{\mathrm{mx}}), s|_{p.1}, \ldots, s|_{p.a}, \\ & \qquad \mathrm{ctx}_{s,k,r}(p, 1), \ldots, \mathrm{ctx}_{s,k,r}(p, \mathrm{mx})) \\ \overset{(6.12)}{=} \; & \mathrm{trans}_{k,r,Z_{\mathrm{mx}}}(s|_{p.l+1}, \\ & \mathrm{thru}_{a,\mathrm{mx},k,r,Z_{\mathrm{mx}}}(y_1, s|_{p.1}, \ldots, s|_{p.a}, \mathrm{ctx}_{s,k,r}(p, 1), \ldots, \mathrm{ctx}_{s,k,r}(p, \mathrm{mx})) \\ & \ldots \\ & \mathrm{thru}_{a,\mathrm{mx},k,r,Z_{\mathrm{mx}}}(y_{\mathrm{mx}}, s|_{p.1}, \ldots, s|_{p.a}, \mathrm{ctx}_{s,k,r}(p, 1), \ldots, \mathrm{ctx}_{s,k,r}(p, \mathrm{mx}))) \\ \overset{(6.10)}{=} \; & \mathrm{trans}_{k,r,Z_{\mathrm{mx}}}(s|_{p.l+1}, \mathrm{ctx}_{s,k,r}(p, 1) \,\ldots\, \mathrm{ctx}_{s,k,r}(p, \mathrm{mx})) \end{aligned}$$

    * if $i > 1$, then $\phi_j = y_j$ and, thus

      $$\begin{aligned} & \mathrm{thru}_{a,\mathrm{mx},k,r,Z_{\mathrm{mx}}}(y_l, s|_{p.1}, \ldots, s|_{p.a}, \mathrm{ctx}_{s,k,r}(p, 1), \ldots, \mathrm{ctx}_{s,k,r}(p, \mathrm{mx})) \\ \overset{(6.10)}{=} \; & \mathrm{ctx}_{s,k,r}(p, l) \end{aligned}$$

[81] Compared to Construction 6.15, we dropped the first subscript, since it is always sub.

If no such call exists in the δ-rules of M_2, then

$$\mathrm{ctx}_{s,k,r}(p.i, l) = \mathrm{nil}$$

- in case $\mathrm{label}_s(p) = f$ with $p \in \mathrm{occ}(s)$, $a \in \mathbb{N}$, $f \in m^{-1}(1)^{(a+1)}$, $\mathrm{label}_s(p.1) = x_b \in X_k$ and $2 \leq i \leq a+1$:

$$\begin{aligned}
&\mathrm{ctx}_{s,k,r}(p.i, l) \\
&= \ (f, i-1, l)\ x_b \\
&\quad \mathrm{unfold}_{f(x_b),k,r,Z_{\mathrm{mx}}}(1, \{i-1\}, s|_{p.2}, \ldots, s|_{p.a+1}, \mathrm{ctx}_{s,k,r}(p,1), \ldots, \mathrm{ctx}_{s,k,r}(p,\mathrm{mx})) \\
&\quad \ldots \\
&\quad \mathrm{unfold}_{f(x_b),k,r,Z_{\mathrm{mx}}}(a, \{i-1\}, s|_{p.2}, \ldots, s|_{p.a+1}, \mathrm{ctx}_{s,k,r}(p,1), \ldots, \mathrm{ctx}_{s,k,r}(p,\mathrm{mx})) \\
&\quad \mathrm{ctx}_{s,k,r}(p,1)\ \ldots\ \mathrm{ctx}_{s,k,r}(p,\mathrm{mx})
\end{aligned}$$

□

Obviously the thru-mappings are no longer required in the construction above. Additionally, we want to drop the SUB-constructor, since it neither appears in any input of M_1, nor in the output of M_2. Thereby, we also translate all references to R_1 into references to R' by "undoing" the freezing step (cf. Lemma 6.11). We will highlight the spots, where this applies. Finally, we gain Construction 6.22.

Construction 6.22 (The extended construction). Let $M' = (F', m, \Delta, e', R') \in \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})$ with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ for some $\mathrm{mx} \in \mathbb{N}$. Let $F_1 = m^{-1}(1)$ and $F_2 = m^{-1}(2)$. We construct the macro tree transducer $M = (F, \Delta, e, R)$ as follows:

- $F = \{\, f^{(r+\mathrm{mx}+1)} \mid r \in \mathbb{N},\ f \in F_1^{(r+1)} \,\} \cup$
  $\cup \{\, (f, h, l)^{(r+\mathrm{mx}+1)} \mid r \in \mathbb{N},\ f \in F_1^{(r+1)},\ h \in [r],\ l \in [\mathrm{mx}] \,\}$,
- $e = \mathrm{trans}_{1,0,\emptyset}(e', \Pi_1, \ldots, \Pi_{\mathrm{mx}})$ and
- the set of equations R contains:
  - for every $k, r \in \mathbb{N}$, $f \in F_1^{(r+1)}$ and $\delta \in \Delta^{(k)}$ the equation [82]

    $$f\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r\ z_1\ \ldots\ z_{\mathrm{mx}} = \mathrm{trans}_{k,r,Z_{\mathrm{mx}}}(\mathrm{rhs}_{M'}(f, \delta), z_1, \ldots, z_{\mathrm{mx}}),$$

    if $(f\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r = \mathrm{rhs}_{M'}(f, \delta)) \in R'$, and
  - for every $k, r \in \mathbb{N}$, $f \in F_1^{(r+1)}$, $\delta \in \Delta^{(k)}$, $h \in [r]$ and $l \in [\mathrm{mx}]$ the equation

    $$(f, h, l)\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r\ z_1\ \ldots\ z_{\mathrm{mx}} = \psi,$$

    if $(f\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r = \mathrm{rhs}_{M'}(f, \delta)) \in R'$, where $\psi = \mathrm{nil}$ for some dummy output symbol $\mathrm{nil} \in \Delta^{(0)}$, if $\#_{y_h}(\mathrm{rhs}_{M'}(f, \delta)) = 0$, otherwise $\psi = \mathrm{ctx}_{\mathrm{rhs}_{M'}(f,\delta),k,r}(p, l)$ with $p \in \mathrm{occ}(\mathrm{rhs}_{M'}(f, \delta))$ such that $\mathrm{rhs}_{M'}(f, \delta)|_p = y_h$ [83].

[82] We supply the right hand side of M', since we will adjust the trans-mapping such that it can cope with substitution function symbols.

[83] p is unique, since the first module of M' is context-linear.

Let $\mathrm{RHS}_{k,r,V} = \mathrm{RHS}(F, \Delta, X_k, Y_r \cup V)$. We define the following mappings for every $k, r \in \mathbb{N}$ and finite set of variables V by simultaneous recursion:

$$\{\mathrm{trans}_{k,r,V}\} \cup \{ \mathrm{unfold}_{f(x_i),k,r,V} \mid f \in F_1,\ x_i \in X_k \}.$$

These are typed

- $\mathrm{trans}_{k,r,V} : \mathrm{RHS}_{F_1,F_2,\Delta}(X_k, Y_r) \times (\mathrm{RHS}_{k,r,V})^{\mathrm{mx}} \longrightarrow \mathrm{RHS}_{k,r,V}$ and
- $\mathrm{unfold}_{f(x),k,r,V} : [n] \times \mathfrak{P}([n]) \times (\mathrm{RHS}_{F_1,F_2,\Delta}(X_k, Y_r))^n \times (\mathrm{RHS}_{k,r,V})^{\mathrm{mx}} \longrightarrow \mathrm{RHS}_{k,r,V}$, where $n \in \mathbb{N}$ with $f \in F_1^{(n+1)}$.

The $\mathrm{trans}_{k,r,V}$-mapping is defined for every $\varrho_1, \ldots, \varrho_{\mathrm{mx}} \in \mathrm{RHS}_{k,r,V}$ by the following equations.

$$\mathrm{trans}_{k,r,V}(y_h, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) = y_h$$

for every $y_h \in Y_r$.

$$\mathrm{trans}_{k,r,V}(\Pi_l, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) = \varrho_l$$

for every $l \in [\mathrm{mx}]$.

$$\begin{aligned}\mathrm{trans}_{k,r,V}((f\ x\ s_1\ \ldots\ s_a), \varrho_1, \ldots, \varrho_{\mathrm{mx}}) = (f\ x\ &\mathrm{unfold}_{f(x),k,r,V}(1, \emptyset, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\ &\ldots \\ &\mathrm{unfold}_{f(x),k,r,V}(a, \emptyset, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\ &\varrho_1\ \ldots\ \varrho_{\mathrm{mx}})\end{aligned}$$

for every $a \in \mathbb{N}$, $f \in F_1^{(a+1)}$, $x \in X_k$ and $s_1, \ldots, s_a \in \mathrm{RHS}_{F_1,F_2,\Delta}(X_k, Y_r)$.

$$\begin{aligned}\mathrm{trans}_{k,r,V}((\delta\ s_1\ \ldots\ s_a), \varrho_1, \ldots, \varrho_{\mathrm{mx}}) = (\delta\ &\mathrm{trans}_{k,r,V}(s_1, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\ &\ldots \\ &\mathrm{trans}_{k,r,V}(s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}))\end{aligned}$$

for every $a \in \mathbb{N}_+$, $\delta \in \Delta^{(a)}$ and $s_1, \ldots, s_a \in \mathrm{RHS}_{F_1,F_2,\Delta}(X_k, Y_r)$.

$$\begin{aligned}\mathrm{trans}_{k,r,V}((\mathrm{sub}_a\ s\ s_1\ \ldots\ s_a), \varrho_1, \ldots, \varrho_{\mathrm{mx}}) = \mathrm{trans}_{k,r,V}(s, &\mathrm{trans}_{k,r,V}(s_1, \varrho_1, \ldots, \varrho_{\mathrm{mx}}), \ldots, \\ &\mathrm{trans}_{k,r,V}(s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}), \\ &\varrho_{a+1}, \ldots, \varrho_{\mathrm{mx}})\end{aligned}$$

for every $a \in \mathbb{N}$, $\mathrm{sub}_a \in F_2^{(a+1)}$ and $s, s_1, \ldots, s_a \in \mathrm{RHS}_{F_1,F_2,\Delta}(X_k, Y_r)$ [84].

For every $x \in X_k$, $a \in \mathbb{N}$, $f \in F_1^{(a+1)}$, $h \in [a]$, $S \subseteq [a]$, $s_1, \ldots, s_a \in \mathrm{RHS}_{F_1,F_2,\Delta}(X_k, Y_r)$ and $\varrho_1, \ldots, \varrho_{\mathrm{mx}} \in \mathrm{RHS}_{k,r,V}$ the $\mathrm{unfold}_{f(x),k,r,V}$-mapping is defined as

$$\mathrm{unfold}_{f(x),k,r,V}(h, S, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) = \mathrm{nil}$$

[84] An occurrence SUB-constructor is gained by freezing a sub-function symbol. By the definition of freezing 6.6, $\Pi_{a+1}, \ldots, \Pi_{\mathrm{mx}}$ are supplied directly at the missing argument positions. Translating those substitution variables leads to the occurrences of $\varrho_{a+1}, \ldots, \varrho_{\mathrm{mx}}$.

for some nil $\in \Delta^{(0)}$, if $h \in S$, and otherwise

$$\begin{aligned}
&\mathrm{unfold}_{f(x),k,r,V}(h, S, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\
= \;& \mathrm{trans}_{k,r,V}(s_h, (f, h, 1)\; x\; \mathrm{unfold}_{f(x),k,r,V}(1, S \cup \{h\}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \;\ldots \\
&\qquad \mathrm{unfold}_{f(x),k,r,V}(a, S \cup \{h\}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\
&\qquad \varrho_1 \;\cdots\; \varrho_{\mathrm{mx}}, \ldots, \\
&\quad (f, h, \mathrm{mx})\; x\; \mathrm{unfold}_{f(x),k,r,V}(1, S \cup \{h\}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \;\ldots \\
&\qquad \mathrm{unfold}_{f(x),k,r,V}(a, S \cup \{h\}, s_1, \ldots, s_a, \varrho_1, \ldots, \varrho_{\mathrm{mx}}) \\
&\qquad \varrho_1 \;\cdots\; \varrho_{\mathrm{mx}}),
\end{aligned}$$

Finally, for every $k, r \in \mathbb{N}$ and $s \in \mathrm{RHS}_{F_1,F_2,\Delta}(X_k, Y_r)$ the $\mathrm{ctx}_{s,k,r}$-mapping is typed

$$\mathrm{ctx}_{s,k,r} : \; \{\, p \in \mathrm{occ}(s) \mid \psi|_p \notin X_k \,\} \times [\mathrm{mx}] \longrightarrow \mathrm{RHS}_{k,r,Z_{\mathrm{mx}}}$$

and defined for every $l \in [\mathrm{mx}]$ and $p.i \in \mathrm{occ}(s)$ with $i \in \mathbb{N}_+$ and $s|_{p.i} \notin X_k$ by

$$\begin{aligned}
\mathrm{ctx}_{s,k,r}(\varepsilon, l) &= z_l. \\
\mathrm{ctx}_{s,k,r}(p.i, l) &= \mathrm{ctx}_{s,k,r}(p, l),
\end{aligned}$$

if $\mathrm{label}_\psi(p) = \delta$ for some $\delta \in \Delta$.

$$\mathrm{ctx}_{s,k,r}(p.i, l) = \mathrm{ctx}_{s,k,r}(p, l),$$

if $\mathrm{label}_s(p) = \mathrm{sub}_a$ for some $a \in \mathbb{N}$, $\mathrm{sub}_a \in F_2^{(a+1)}$ and $i > 1$.

$$\begin{aligned}
\mathrm{ctx}_{s,k,r}(p.i, l) = \;& (f, i-1, l)\; x\; \mathrm{unfold}_{f(x),k,r,Z_{\mathrm{mx}}}(1, \{i-1\}, s|_{p.2}, \ldots, s|_{p.(a+1)}, \\
&\qquad \mathrm{ctx}_{s,k,r}(p, 1), \ldots, \mathrm{ctx}_{s,k,r}(p, \mathrm{mx})) \;\ldots \\
&\quad \mathrm{unfold}_{f(x),k,r,Z_{\mathrm{mx}}}(a, \{i-1\}, s|_{p.2}, \ldots, s|_{p.(a+1)}, \\
&\qquad \mathrm{ctx}_{s,k,r}(p, 1), \ldots, \mathrm{ctx}_{s,k,r}(p, \mathrm{mx})) \\
&\quad \mathrm{ctx}_{s,k,r}(p, 1) \;\ldots\; \mathrm{ctx}_{s,k,r}(p, \mathrm{mx}),
\end{aligned}$$

if $\mathrm{label}_s(p) = f$ and $\mathrm{label}_s(p.1) = x \in X_k$ and $f \in F_1^{(a+1)}$ with $a \in \mathbb{N}$.

$$\mathrm{ctx}_{s,k,r}(p.1, l) = \mathrm{trans}_{k,r,Z_{\mathrm{mx}}}(s|_{p.l+1}, \mathrm{ctx}_{s,k,r}(p, 1), \ldots, \mathrm{ctx}_{s,k,r}(p, a), \Pi_{a+1}, \ldots, \Pi_{\mathrm{mx}})$$

if $\mathrm{label}_s(p) = \mathrm{sub}_a$ for some $a \in \mathbb{N}$ and $\mathrm{sub}_a \in F_2^{(a+1)}$ [85]. □

Theorem 6.23 (Correctness of Construction 6.22). *Let* $M \in \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})$ *and, furthermore, let* $M' = \mathfrak{C}_{\,6.22}(M)$*. Then*

$$\tau_M = \tau_{M'}.$$

Proof. By Theorem 6.18 this property holds for the extended indirect construction and in Construction 6.21 we computed the composition of the three steps of the indirect construction, thus, the property also holds for Construction 6.21. Construction 6.22, however, was derived (equivalence preserving) from Construction 6.21 and, thereby, we conclude the equivalence $\tau_M = \tau_{M'}$. □

[85] The first argument of a SUB-constructor corresponds to the recursion argument of a sub_a function call. The additional parameters, according to Equation 6.6 are $\Pi_{a+1}, \ldots, \Pi_{\mathrm{mx}}$, which remain invariant under the thru-mapping.

7 Efficiency analysis revisited

In this section we will study the efficiency implications of the extended Construction 6.22. Roughly speaking, we will first highlight a particular cause of inefficiency, which we will avoid by refining the construction. However, we will not prove the correctness of the refined construction, since the proof is tedious. Afterwards we are going to instantiate the refined construction to gain a construction suitable for restricted modular tree transducers; called leaf-accumulating; $M \in \mathrm{ModTT}(\mathrm{MAC_{nc}}, \mathrm{SUB})$, such that we can ensure efficiency non-deterioration.

Example 7.1 (A particular cause of inefficiency). Assume the modular tree transducer M_{usa} of Example 6.10 and the macro tree transducer M'_{usa} of Example 6.17 with function symbols renamed to coll and $(1,1)$. In Figure 24 and Figure 25 the computations

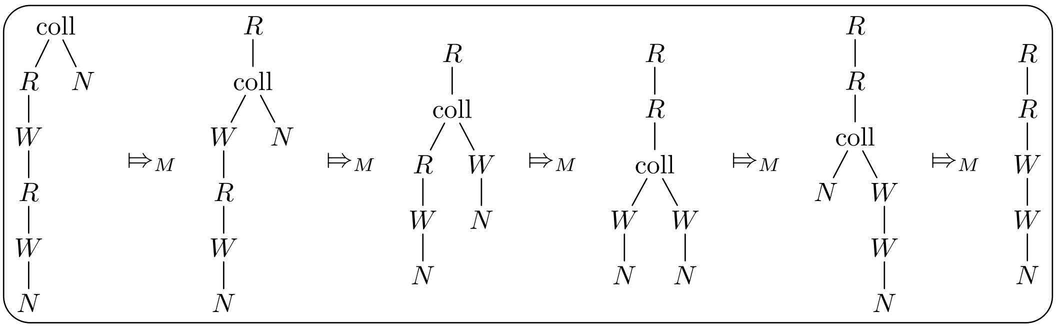

Figure 24: The particular cause of inefficiency illustrated, where $M = M_{\mathrm{usa}}$

of M_{usa} and M'_{usa}, respectively, are displayed. Hence $\mathrm{steps}_{\Rightarrow_{M_{\mathrm{usa}}}}(R\,(W\,(R\,(W\,N)))) = 5$ and $\mathrm{steps}_{\Rightarrow_{M'_{\mathrm{usa}}}}(R\,(W\,(R\,(W\,N)))) = 10$. $\square$

Intuitively, this inefficiency results out of the different setting. If two macro tree transducers are composed the whole output of the first macro tree transducer is processed by the second macro tree transducer, however, with a 2-modular tree transducer part of the output might be created solely by the first module without being processed by the second module. Therefore, we introduce a new set of function symbols with basically the original right hand sides [86]. So as long as there is no call to some substitution function symbol, we compute using the alternative function symbols and, thereby, avoid the illustrated inefficiency.

Through a series of modified constructions, we will refine the extended Construction 6.22. We will always justify the modifications, such that the correctness of the final construction is assured by the correctness of the extended Construction 6.22.

Construction 7.2 (Refined extended construction, step 1). Let

$$M = (F, m, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{MAC_{nc}}, \mathrm{SUB})$$

with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ for some $\mathrm{mx} \in \mathbb{N}$ and let $M' = \mathfrak{C}_{6.22}(M) = (F', \Delta, e', R')$. We construct the macro tree transducer $M'' = (F'', \Delta, e'', R'')$ defined by

[86] The context parameters containing the translations of the original context parameters remain, since they might be needed as soon as a substitution function symbol is encountered.

- $F'' = F' \cup \{\, \bar{f}^{(2r+\mathrm{mx}+1)} \mid r \in \mathbb{N},\ f \in m^{-1}(1)^{(r+1)} \,\}$,
- $e'' = (\bar{f} s_1 \dots s_r \Pi_1 \dots \Pi_{\mathrm{mx}} t_1 \dots t_r)$, if $e' = (f s_1 \dots s_r \Pi_1 \dots \Pi_{\mathrm{mx}})$ and $e = (f t_1 \dots t_r)$, and
- $R'' = R' \cup \bar{R}$, which contains for every $k, r \in \mathbb{N}$, $f \in m^{-1}(1)^{(r+1)}$ and $\delta \in \Delta^{(k)}$ the equation:

  $$\bar{f}\,(\delta\, x_1\ \dots\ x_k)\, y_1\ \dots\ y_r\ z_1\ \dots\ z_{\mathrm{mx}}\, y_{r+1}\ \dots\ y_{2r} = \overline{\mathrm{trans}}_{k,r,Z_{\mathrm{mx}}}(\mathrm{rhs}_M(f,\delta), z_1, \dots, z_{\mathrm{mx}}).$$

The $\overline{\mathrm{trans}}_{k,r,V}$-mappings are defined for every $k, r \in \mathbb{N}$ and finite set of variables V by structural recursion. With $\mathrm{RHS}_{k,r,V} = \mathrm{RHS}_{\mathrm{MAC}}(F'', \Delta, X_k, Y_r \cup V)$ they are typed

$$\overline{\mathrm{trans}}_{k,r,V} :\ \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r) \times (\mathrm{RHS}_{k,r,V})^{\mathrm{mx}} \longrightarrow \mathrm{RHS}_{k,r,V}.$$

The $\overline{\mathrm{trans}}_{k,r,V}$-mapping is defined for every $\varrho_1, \dots, \varrho_{\mathrm{mx}} \in \mathrm{RHS}_{k,r,V}$ by the following equations:

$$\overline{\mathrm{trans}}_{k,r,V}(y_h, \varrho_1, \dots, \varrho_{\mathrm{mx}}) = y_h$$

for every $y_h \in Y_r$.

$$\overline{\mathrm{trans}}_{k,r,V}(\Pi_j, \varrho_1, \dots, \varrho_{\mathrm{mx}}) = \varrho_j$$

for every $j \in [\mathrm{mx}]$.

$$\begin{aligned}\overline{\mathrm{trans}}_{k,r,V}((f\ x\ s_1\ \dots\ s_a), \varrho_1, \dots, \varrho_{\mathrm{mx}}) = \bar{f}\ x\ & \mathrm{unfold}_{f(x),k,r,V}(1, \emptyset, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\ & \dots \\ & \mathrm{unfold}_{f(x),k,r,V}(a, \emptyset, s_1, \dots, s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) \\ & \varrho_1\ \dots\ \varrho_{\mathrm{mx}} \\ & \overline{\mathrm{trans}}_{k,r,V}(s_1, \varrho_1, \dots, \varrho_{\mathrm{mx}})\ \dots \\ & \overline{\mathrm{trans}}_{k,r,V}(s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}})\end{aligned}$$

for every $a \in \mathbb{N}$, $f \in m^{-1}(1)^{(a+1)}$, $x \in X_k$ and $s_1, \dots, s_a \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r)$.

$$\begin{aligned}\overline{\mathrm{trans}}_{k,r,V}((\delta\ s_1\ \dots\ s_a), \varrho_1, \dots, \varrho_{\mathrm{mx}}) = (\delta\ & \overline{\mathrm{trans}}_{k,r,V}(s_1, \varrho_1, \dots, \varrho_{\mathrm{mx}})\ \dots \\ & \overline{\mathrm{trans}}_{k,r,V}(s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}))\end{aligned}$$

for every $a \in \mathbb{N}$, $\delta \in \Delta^{(a)} \setminus \Pi$ and $s_1, \dots, s_a \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r)$.

$$\begin{aligned}\overline{\mathrm{trans}}_{k,r,V}(\mathrm{sub}_a\ s\ s_1\ \dots\ s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}) = \mathrm{trans}_{k,r,V}(s,\ & \overline{\mathrm{trans}}_{k,r,V}(s_1, \varrho_1, \dots, \varrho_{\mathrm{mx}}), \dots, \\ & \overline{\mathrm{trans}}_{k,r,V}(s_a, \varrho_1, \dots, \varrho_{\mathrm{mx}}), \\ & \varrho_{a+1}, \dots, \varrho_{\mathrm{mx}})\end{aligned}$$

for every $a \in \mathbb{N}$, $\mathrm{sub}_a \in F_2^{(a+1)}$ and $s, s_1, \dots, s_a \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r)$.

Apparently, $\tau_M = \tau_{M'} = \tau_{M''}$, since the right hand sides of the $\bar{f}$-function symbols (for every $f \in m^{-1}(1)$) are essentially those of f with additional context parameters, which so far are not used. □

We note that for every call to a $\overline{\text{trans}}_{k,r,V}$-mapping the parameters $\varrho_1, \ldots, \varrho_{\text{mx}}$ will be $z_1, \ldots, z_{\text{mx}}$ [87], respectively. Consequently, we drop $\varrho_1, \ldots, \varrho_{\text{mx}}$ from the specification and replace every ϱ_i by z_i for every $i \in [\text{mx}]$ in the defining equations. Further, we note that in every right hand side $\text{rhs}_{M''}(f, \delta)$ with $k, r \in \mathbb{N}$, $f \in m^{-1}(1)^{(r+1)}$, $\delta \in \Delta^{(k)}$ and

$$(f\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots y_r\ z_1\ \ldots\ z_{\text{mx}}\ y_{r+1}\ \ldots\ y_{2r} = \text{rhs}_{M''}(f, \delta)) \in \bar{R}$$

every call to some $\bar{f}$ with $f \in m^{-1}(1)$ now looks like $(\bar{f}\, x\, \xi_1\, \ldots\, \xi_r\, z_1\, \ldots\, z_{\text{mx}}\, \xi_{r+1} \ldots \xi_{2r})$ for some $x \in X_k$ and $\xi_1, \ldots, \xi_{2r} \in \text{RHS}_{\text{MAC}}(F'', \Delta, X_k, Y_r \cup Z_{\text{mx}})$. Together with the initial expression, where the $z_1, \ldots, z_{\text{mx}}$ are set to $\Pi_1, \ldots, \Pi_{\text{mx}}$, respectively, this leads us to replace every z_i by Π_i for every $i \in [\text{mx}]$, and then drop the context parameters $z_1, \ldots, z_{\text{mx}}$ completely from the specification. Thereby, one of the defining equation changes to

$$\overline{\text{trans}}_{k,r,V}(\Pi_j, \varrho_1, \ldots, \varrho_{\text{mx}}) \ = \ \varrho_j = z_j = \Pi_j$$

and, thus, merges with the case where the first parameter is an element of $\Delta \setminus \Pi$. Altogether, we arrive at the following construction.

Construction 7.3 (Refined extended construction, step 2**).** Let

$$M = (F, m, \Delta, e, R) \in \text{ModTT}(\text{MAC}_{\text{nc}}, \text{SUB})$$

with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\text{mx}}\}$ for some $\text{mx} \in \mathbb{N}$, and let $M' = \mathfrak{C}_{6.22}(M) = (F', \Delta, e', R')$. We construct the macro tree transducer $M'' = (F'', \Delta, e'', R'')$ defined by

- $F'' = F' \cup \{\, \bar{f}^{(2r+1)} \mid r \in \mathbb{N},\ f \in m^{-1}(1)^{(r+1)} \,\}$,
- $e'' = (\bar{f}\ s_1\ \ldots\ s_r\ t_1\ \ldots\ t_r)$, if $e' = (f\ s_1\ \ldots\ s_r\ \Pi_1\ \ldots\ \Pi_{\text{mx}})$ and $e = (f\ t_1\ \ldots\ t_r)$, and
- $R'' = R' \cup \bar{R}$, where $\bar{R}$ contains for every $k, r \in \mathbb{N}$, $f \in m^{-1}(1)^{(r+1)}$ and $\delta \in \Delta^{(k)}$ the equation:

$$\bar{f}\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r\ y_{r+1}\ \ldots\ y_{2r} = \overline{\text{trans}}_{k,r}(\text{rhs}_M(f, \delta))$$

We define the $\overline{\text{trans}}_{k,r}$-mappings for every $k, r \in \mathbb{N}$ by structural recursion. They are typed

$$\overline{\text{trans}}_{k,r} :\ \text{RHS}_{m^{-1}(1), m^{-1}(2), \Delta}(X_k, Y_r) \longrightarrow \text{RHS}(F'', \Delta, X_k, Y_r).$$

The $\overline{\text{trans}}_{k,r}$-mapping is defined by the following equations.

$$\overline{\text{trans}}_{k,r}(y_h) \ = y_h$$

for every $y_h \in Y_r$.

$$\begin{aligned}\overline{\text{trans}}_{k,r}(f\ x\ s_1\ \ldots\ s_a) \ = (\bar{f}\ x\ &\text{unfold}_{f(x),k,r,\emptyset}(1, \emptyset, s_1, \ldots, s_a, \Pi_1, \ldots, \Pi_{\text{mx}})\ \ldots \\ &\text{unfold}_{f(x),k,r,\emptyset}(a, \emptyset, s_1, \ldots, s_a, \Pi_1, \ldots, \Pi_{\text{mx}}) \\ &\overline{\text{trans}}_{k,r}(s_1)\ \ldots\ \overline{\text{trans}}_{k,r}(s_a))\end{aligned}$$

[87] $z_1, \ldots, z_{\text{mx}}$ are supplied in the definition of the right hand sides for the new function symbols and they are copied unchanged to all calls to $\overline{\text{trans}}_{k,r,V}$ in the defining equations of $\overline{\text{trans}}_{k,r,V}$.

for every $a \in \mathbb{N}$, $f \in {m^{-1}(1)}^{(a+1)}$, $x \in X_k$ and $s_1, \ldots, s_a \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r)$.

$$\overline{\mathrm{trans}}_{k,r}(\delta\ s_1\ \ldots\ s_a) = (\delta\ \overline{\mathrm{trans}}_{k,r}(s_1)\ \ldots\ \overline{\mathrm{trans}}_{k,r}(s_a))$$

for every $a \in \mathbb{N}$, $\delta \in \Delta^{(a)}$ and $s_1, \ldots, s_a \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r)$.

$$\overline{\mathrm{trans}}_{k,r}(\mathrm{sub}_a\ s\ s_1\ \ldots\ s_a) = \mathrm{trans}_{k,r,\emptyset}(s, \overline{\mathrm{trans}}_{k,r}(s_1), \ldots, \overline{\mathrm{trans}}_{k,r}(s_a), \Pi_{a+1}, \ldots, \Pi_{\mathrm{mx}})$$

for every $a \in \mathbb{N}$, $\mathrm{sub}_a \in {m^{-1}(2)}^{(a+1)}$ and $s, s_1, \ldots, s_a \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r)$.
Still $\tau_M = \tau_{M'} = \tau_{M''}$. □

So far the additional parameters $y_{r+1}, \ldots, y_{2r}$ of a function symbol $\bar{f} \in F''^{(r+1)}$ with $r \in \mathbb{N}$ are not used, but in the final construction, we replace the defining equation

$$\overline{\mathrm{trans}}_{k,r}(y_h) = y_h \qquad \text{by} \qquad \overline{\mathrm{trans}}_{k,r}(y_h) = y_{h+r}.$$

Intuitively, this replacement is governed by the fact that we do not need to translate the context parameter (via the (f, h, l)-function symbols), since the substitution is never applied to y_h. Thus, we can output the original context parameter, of which we kept track in y_{h+r}.

Construction 7.4 (Refined extended construction). Let $M = (F, m, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})$ with substitution variables $\Pi = \{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ for some $\mathrm{mx} \in \mathbb{N}$ and let $M' = (F', \Delta, e', R') = \mathfrak{C}_{6.22}(M)$. We construct the macro tree transducer $M'' = (F'', \Delta, e'', R'')$ defined by

- $F'' = F' \cup \{\, \bar{f}^{(2r+1)} \mid r \in \mathbb{N},\ f \in {m^{-1}(1)}^{(r+1)} \,\}$,
- $e'' = (\bar{f}\ s_1\ \ldots\ s_r\ t_1\ \ldots\ t_r)$, if $e' = (f\ s_1\ \ldots\ s_r\ \Pi_1\ \ldots\ \Pi_{\mathrm{mx}})$ and $e = (f\ t_1\ \ldots\ t_r)$ and
- $R'' = R' \cup \bar{R}$, where $\bar{R}$ contains for every $k, r \in \mathbb{N}$, $f \in {m^{-1}(1)}^{(r+1)}$ and $\delta \in \Delta^{(k)}$ the equation:

$$\bar{f}\ (\delta\ x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r\ y_{r+1}\ \ldots\ y_{2r} = \overline{\mathrm{trans}}_{k,r}(\mathrm{rhs}_M(f, \delta))$$

We define the $\overline{\mathrm{trans}}_{k,r}$-mappings for every $k, r \in \mathbb{N}$ by structural recursion. They are typed

$$\overline{\mathrm{trans}}_{k,r} : \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r) \longrightarrow \mathrm{RHS}(F'', \Delta, X_k, Y_{2r}).$$

The $\overline{\mathrm{trans}}_{k,r}$-mapping is defined by the following equations.

$$\overline{\mathrm{trans}}_{k,r}(y_h) = y_{h+r}$$

for every $y_h \in Y_r$.

$$\begin{aligned}\overline{\mathrm{trans}}_{k,r}(f\ x\ s_1\ \ldots\ s_a) = (\bar{f}\ x\ & \mathrm{unfold}_{f(x),k,r,\emptyset}(1, \emptyset, s_1, \ldots, s_a, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\ & \ldots \\ & \mathrm{unfold}_{f(x),k,r,\emptyset}(a, \emptyset, s_1, \ldots, s_a, \Pi_1, \ldots, \Pi_{\mathrm{mx}}) \\ & \overline{\mathrm{trans}}_{k,r}(s_1)\ \ldots\ \overline{\mathrm{trans}}_{k,r}(s_a))\end{aligned}$$

for every $a \in \mathbb{N}$, $f \in m^{-1}(1)^{(a+1)}$, $x \in X_k$ and $s_1, \ldots, s_a \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r)$.

$$\overline{\mathrm{trans}}_{k,r}(\delta\; s_1\; \ldots\; s_a) = (\delta\; \overline{\mathrm{trans}}_{k,r}(s_1)\; \ldots\; \overline{\mathrm{trans}}_{k,r}(s_a))$$

for every $a \in \mathbb{N}$, $\delta \in \Delta^{(a)}$ and $s_1, \ldots, s_a \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r)$.

$$\overline{\mathrm{trans}}_{k,r}(\mathrm{sub}_a\; s\; s_1\; \ldots\; s_a) = \mathrm{trans}_{k,r,\emptyset}(s, \overline{\mathrm{trans}}_{k,r}(s_1), \ldots, \overline{\mathrm{trans}}_{k,r}(s_a), \Pi_{a+1}, \ldots, \Pi_{\mathrm{mx}})$$

for every $a \in \mathbb{N}$, $\mathrm{sub}_a \in m^{-1}(2)^{(a+1)}$ and $s, s_1, \ldots, s_a \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k, Y_r)$.
As usual the result M'' is denoted $M'' = \mathfrak{C}_{7.4}(M)$. □

Conjecture 7.5 (Correctness of Construction 7.4). *Let $M \in \mathrm{ModTT}(\mathrm{MAC}_{\mathrm{nc}}, \mathrm{SUB})$ and $M'' = \mathfrak{C}_{7.4}(M)$, then*

$$\tau_M = \tau_{M''}.$$

As already stated, we will not formally verify this claim, since the proof is pretty intense and tedious. □

We illustrate, how the refined Construction 7.4 avoids the particular inefficiency incurred by Construction 6.22, in the following example.

Example 7.6 (The refined construction applied to M_{usa}). Let M_{usa} be the modular tree transducer of Example 6.10. The macro tree transducer $M''_{\mathrm{usa}} = \mathfrak{C}_{7.4}(M) = (F'', \Delta, e'', P'')$ is

- $F'' = \{\mathrm{coll}^{(3)}, (\mathrm{coll}, 1, 1)^{(3)}, \overline{\mathrm{coll}}^{(3)}\}$,
- $e'' = \overline{\mathrm{coll}}\; x\; ((\mathrm{coll}, 1, 1)\; x\; N\; N)\; N$
- P'' contains

$$\begin{array}{llll}
\mathrm{coll}\; N & y_1\; z_1 = y_1 & \overline{\mathrm{coll}}\; N & y_1\; y_2 = y_2 \\
\mathrm{coll}\; (\,|\; x_1) & y_1\; z_1 = y_1 & \overline{\mathrm{coll}}\; (\,|\; x_1) & y_1\; y_2 = y_1 \\
\mathrm{coll}\; (R\; x_1) & y_1\; z_1 = R\; (\mathrm{coll}\; x_1\; y_1\; z_1) & \overline{\mathrm{coll}}\; (R\; x_1) & y_1\; y_2 = R\; (\overline{\mathrm{coll}}\; x_1\; y_1\; y_2) \\
\mathrm{coll}\; (W\; x_1) & y_1\; z_1 = \mathrm{coll}\; x_1\; (W\; y_1)\; z_1 & \overline{\mathrm{coll}}\; (W\; x_1) & y_1\; y_2 = \overline{\mathrm{coll}}\; x_1\; (W\; y_1)\; (W\; y_2)
\end{array}$$

$$\begin{array}{lll}
(\mathrm{coll}, 1, 1)\; N & y_1\; z_1 = & z_1 \\
(\mathrm{coll}, 1, 1)\; (\,|\; x_1) & y_1\; z_1 = & \mathrm{coll}\; x_1\; ((\mathrm{coll}, 1, 1)\; x_1\; N\; z_1)\; z_1 \\
(\mathrm{coll}, 1, 1)\; (R\; x_1) & y_1\; z_1 = & (\mathrm{coll}, 1, 1)\; x_1\; N\; z_1 \\
(\mathrm{coll}, 1, 1)\; (W\; x_1) & y_1\; z_1 = & (\mathrm{coll}, 1, 1)\; x_1\; N\; z_1.
\end{array}$$

Thus, $\mathrm{steps}_{\Rightarrow_{M''_{\mathrm{usa}}}}(R\; (W\; (R\; (W\; N)))) = 5$ as evidenced by Figure 26. □

As already pointed out in [Voi02], it is quite hard to relate the number of derivation steps invested to pre-translate the context parameters. Thus, we introduce a property, namely leaf-accumulating, that assures that such pre-translations are never used.

Definition 7.7 (Leaf-accumulating modular tree transducer). A modular tree transducer $M = (F, m, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{MAC_{nc}}, \mathrm{SUB})$ is called *leaf-accumulating*, if and only if there exists a partition of $m^{-1}(1) = F_1 \cup F_1'$ with $F_1 \cap F_1' = \emptyset$ and $F_1' \subseteq m^{-1}(1)^{(1)}$, such that for every $k, r \in \mathbb{N}$, $f \in m^{-1}(1)^{(r+1)}$ and $\delta \in \Delta^{(k)}$ the right hand side $\mathrm{rhs}_M(f, \delta)$ with $(f\ (\delta\, x_1\ \ldots\ x_k)\ y_1\ \ldots\ y_r = \mathrm{rhs}_M(f, \delta)) \in R$ the following condition holds:

$$\varrho \in \mathrm{RHS}_{F_1', m^{-1}(2), \Delta}(X_k, \emptyset)$$

for every $p \in \mathrm{occ}(\mathrm{rhs}_M(f, \delta))$ with $\mathrm{rhs}_M(f, \delta)|_p = (\mathrm{sub}_a\ \varrho\, \varrho_1\ \ldots\ \varrho_a)$ for some $a \in \mathbb{N}$, $\mathrm{sub}_a \in m^{-1}(2)^{(a+1)}$ and $\varrho, \varrho_1, \ldots, \varrho_a \in \mathrm{RHS}(1, F, m, \Delta, X_k, Y_r)$ and, additionally, if $f \in F_1'$, then also $\mathrm{rhs}_M(f, \delta) \in \mathrm{RHS}_{F_1', m^{-1}(2), \Delta}(X_k, \emptyset)$. □

Example 7.8 (Leaf-accumulating modular tree transducer). Let

$$M_{\mathrm{frev}} = (\{\mathrm{rev}^{(1)}, \mathrm{frev}^{(2)}, \mathrm{app}^{(2)}\}, m, \{A^{(1)}, B^{(1)}, C^{(1)}, N^{(0)}\}, (\mathrm{frev}\ x\ N), R)$$

with $m(\mathrm{frev}) = m(\mathrm{rev}) = 1$, $m(\mathrm{app}) = 2$ and R contains:

$$\begin{array}{llll}
\mathrm{frev}\ N\ y_1 & = y_1 & \mathrm{rev}\ N & = N \\
\mathrm{frev}\ (A\, x_1)\ y_1 & = \mathrm{app}\ (\mathrm{rev}\ x_1)\ (\mathrm{frev}\ x\ y_1) & \mathrm{rev}\ (A\, x_1) & = \mathrm{app}\ (\mathrm{rev}\ x_1)\ (A\ N) \\
\mathrm{frev}\ (B\, x_1)\ y_1 & = \mathrm{app}\ (\mathrm{rev}\ x_1)\ (\mathrm{frev}\ x\ y_1) & \mathrm{rev}\ (B\, x_1) & = \mathrm{app}\ (\mathrm{rev}\ x_1)\ (B\ N) \\
\mathrm{frev}\ (C\, x_1)\ y_1 & = \mathrm{app}\ (\mathrm{rev}\ x_1)\ (\mathrm{frev}\ x\ (C\ y_1)) & \mathrm{rev}\ (C\, x_1) & = \mathrm{rev}\ x_1
\end{array}$$

$$\begin{array}{ll}
\mathrm{app}\ N\ y_1 = & y_1 \\
\mathrm{app}\ (A\, x_1)\ y_1 = & A\ (\mathrm{app}\ x_1\ y_1) \\
\mathrm{app}\ (B\, x_1)\ y_1 = & B\ (\mathrm{app}\ x_1\ y_1) \\
\mathrm{app}\ (C\, x_1)\ y_1 = & C\ (\mathrm{app}\ x_1\ y_1).
\end{array}$$

M_{frev} is leaf-accumulating (the partition is $m^{-1}(1) = \{\mathrm{rev}\} \cup \{\mathrm{frev}\}$) and computes a concatenated inits-like function on the reversed list, which also filters C's and appends them to the end. For example $\tau_{M_{\mathrm{frev}}}(C\ (A\ (B\ N))) = B\ (A\ (B\ (C\ N)))$. However, M_{usa} of Example 6.10 is not leaf-accumulating. □

If a modular tree transducer $M \in \mathrm{ModTT}(\mathrm{MAC_{nc}}, \mathrm{SUB})$ is leaf-accumulating, then we can separate a 2-modular tree transducer $M' \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$, which contains the first module with function symbols restricted to the function symbols of F_1' according to Definition 7.7 and the whole second module. To this part we can then apply Construction 5.10 to gain a macro tree transducer $\bar{M}'$ which is at least as efficient as M'. The remaining right hand sides of the function symbols of F_1 (according to Definition 7.7) can be obtained by taking the variable shared term graph rewrite rules corresponding to the equations created by Construction 7.4. We claim the following:

Conjecture 7.9 (Refined extended construction). *Let M be a leaf-accumulating modular tree transducer. The macro tree transducer M' constructed according to the specification above is at least as efficient as M.* □

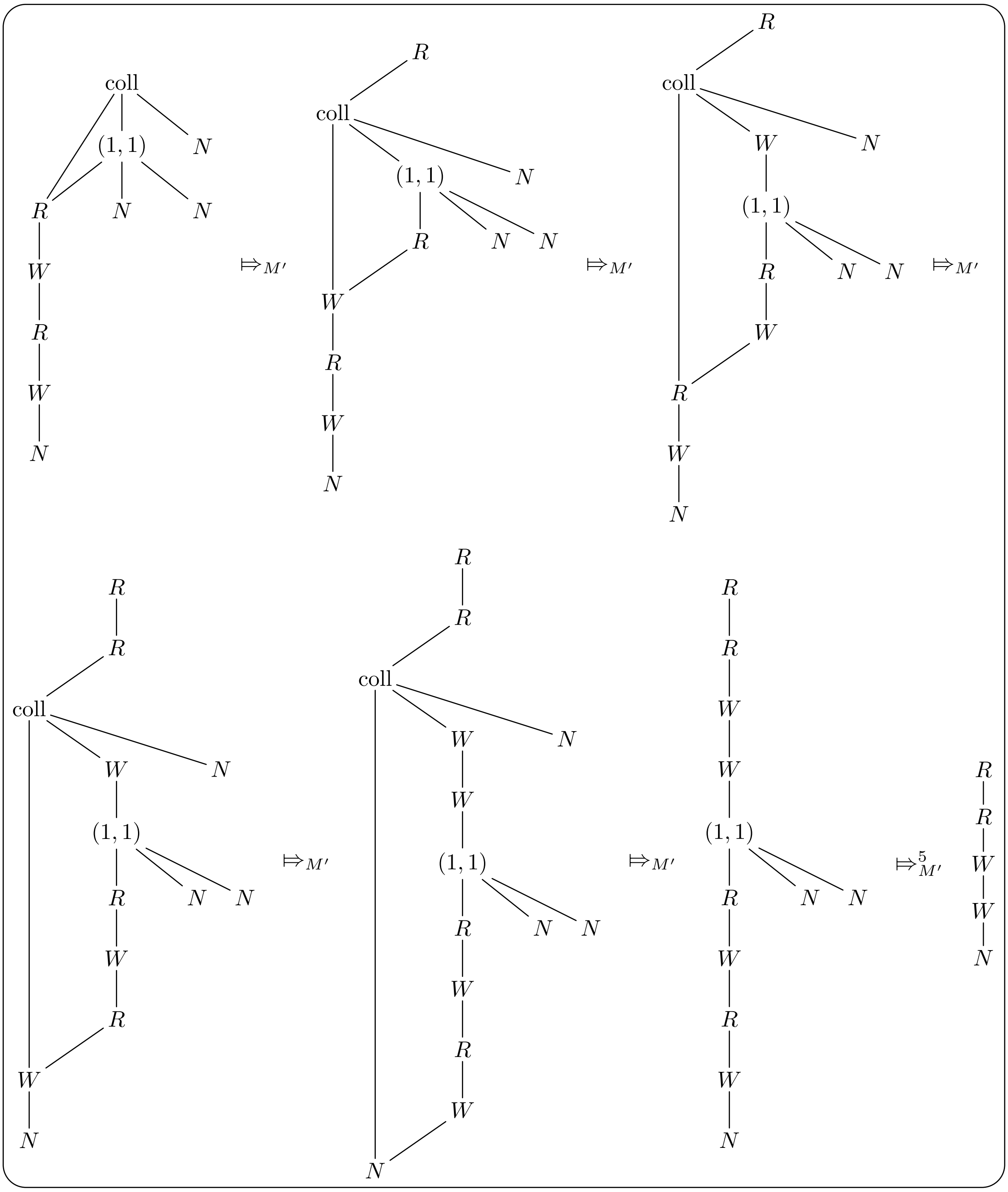

Figure 25: The particular cause of inefficiency illustrated, where $M' = \mathfrak{C}_{6.22}(M_{\text{usa}})$

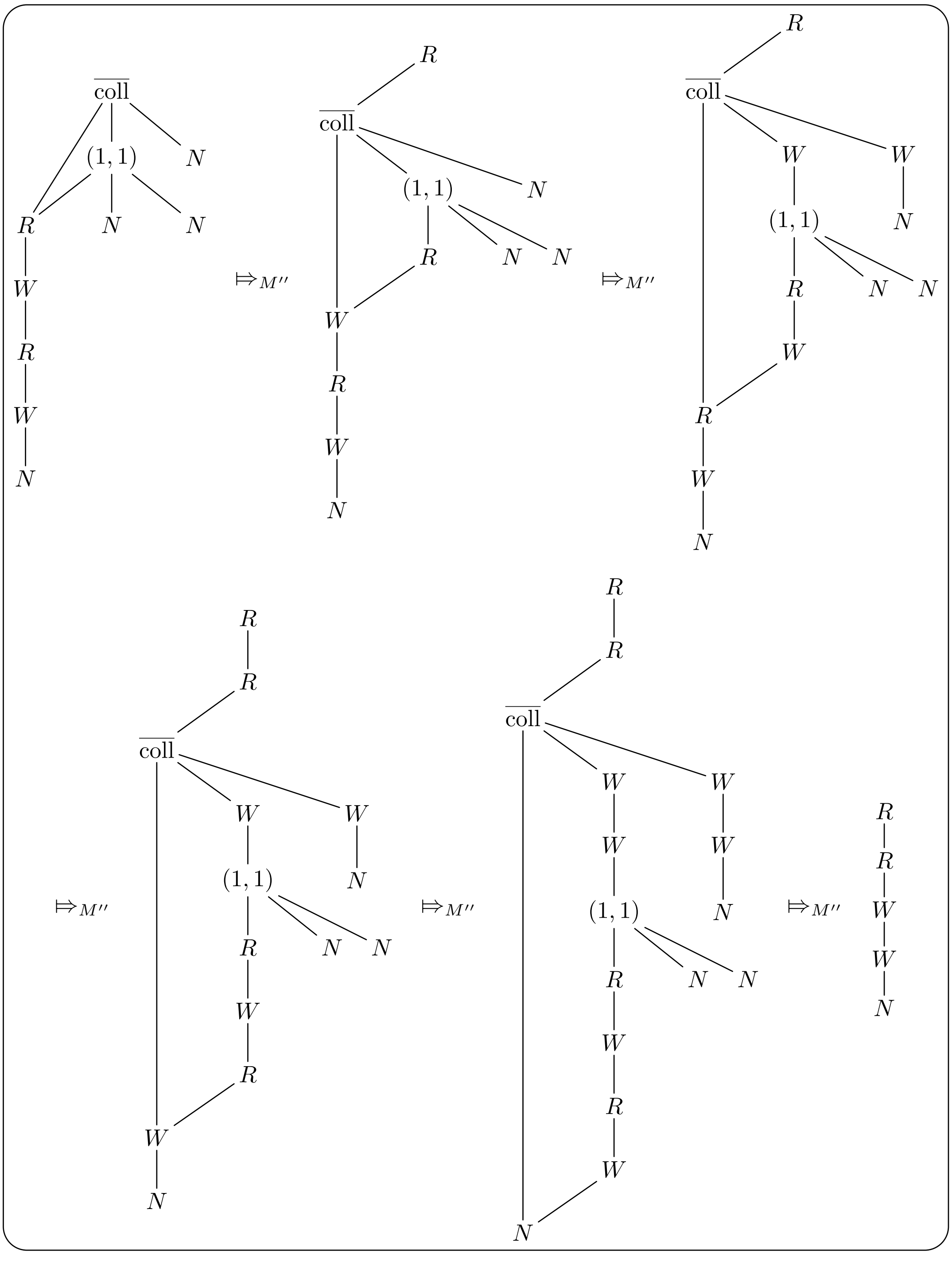

Figure 26: The particular cause of inefficiency resolved, where $M'' = M''_{\mathrm{usa}}$ and $(1,1) = (\mathrm{coll},1,1)$

8 Comparing the construction

Before proceeding with the efficiency analysis, we compare the given construction with similar ones found in the literature.

8.1 Comparison to the indirect construction of [KGK01]

As already pointed out, an indirect construction may be found in [KGK01]. Obviously both constructions produce semantically equivalent macro tree transducers (as shown in Theorem 4.16), else the constructions would not be correct.

However syntactic equality, in the sense that the constructed macro tree transducers are syntactically equivalent, is not achieved. This is due to the preservation of the original substitution function symbols until stage 3 (Composition) in [KGK01]. In our construction we implicitly freeze (cf. Freezing in [KGK01]) all substitution function symbol calls to one [88] substitution function symbol constructor. So we implicitly replace substitution function symbol calls like

$$\mathrm{sub_i}\; s\, s_1\; \ldots\; s_i \qquad \text{by} \qquad \mathrm{sub}\; s\, s_1\; \ldots\; s_i\, \Pi_{i+1}\; \ldots\; \Pi_{\mathrm{mx}},$$

where $\mathrm{sub}_i \in {F_2}^{(i+1)}$ with $i \in \mathbb{N}$, $\mathrm{mx} = \max(\mathrm{ar}(F_2)) - 1$ and (F_2, ar) as the function symbol signature of the substitution module and $\{\Pi_1, \ldots, \Pi_{\mathrm{mx}}\}$ as substitution variables.

This replacement is justified by Lemma 4.4, which immediately proves preservation of semantics, i.e.

$$s\{\,\Pi_1 \mapsto s_1, \ldots, \Pi_i \mapsto s_i\,\} = s\{\,\Pi_1 \mapsto s_1, \ldots, \Pi_i \mapsto s_i, \Pi_{i+1} \mapsto \Pi_{i+1}, \ldots, \Pi_{\mathrm{mx}} \mapsto \Pi_{\mathrm{mx}}\,\}.$$

So by replacing Lemma 11 of [KGK01] by Lemma 6.11 and deleting step 2 in Lemma 13 in [KGK01] we gain syntactic equality. The adjusted Lemmata can be found in the previous section (there for the extended setting) as Lemma 6.11 and 6.13.

We will not formally prove the claim that the constructed macro tree transducers (with respect to Construction 4.5 and the adjusted construction of [KGK01]) are syntactically equivalent (modulo renaming of the function symbols). We note, however, that, similar to the approach in the previous section, the steps can be formally composed, yielding in the first step a yield macro tree transducer [FV98] (cf. 6.20). Then our construction simply represents the composition of steps 1 and 3, thereby avoiding the explicit freezing.

Lastly, the question arises why we modified the original indirect construction to match with the construction presented herein arises. We could have adjusted Construction 4.5 as well, but we consider the simplification to one substitution function symbol to be beneficial, since it avoids awkward equations in the constructed macro tree transducers (cf. Example 8.1). Furthermore, direct constructions usually exhibit a better space and time behavior compared to indirect constructions. On the other hand an indirect construction usually provides a better insight into the process, thereby possibly admitting a straightforward lifting of the construction to less-restricted settings. In our case, however, the discussed simplification was proven to be essential for the extended construction to be applicable.

[88] and not many substitution function symbol constructors as in [KGK01]

Example 8.1 (Comparing the constructed macro tree transducers). In Example 4.2 we have already presented the resulting macro tree transducer M'_{rev}, when Construction 4.5 is applied to M_{rev}. In [KGK01] the same example is studied with respect to the construction presented there and the outcome is the macro tree transducer $M''_{\text{rev}} = (\{\text{rev}^{(1)}, \text{rev}'^{(2)}\}, \{A^{(1)}, B^{(1)}, N^{(0)}\}, (\text{rev } x), R)$ with equations

$$
\begin{array}{llcll}
\text{rev} & \text{N} & & = & \text{N} \\
\text{rev} & (\text{A } x_1) & & = & \text{rev}' \; x_1 \; (\text{A N}) \\
\text{rev} & (\text{B } x_1) & & = & \text{rev}' \; x_1 \; (\text{B N}) \\[4pt]
\text{rev}' & \text{N} & y_1 & = & y_1 \\
\text{rev}' & (\text{A } x_1) & y_1 & = & \text{rev}' \; x_1 \; (\text{A N}) \\
\text{rev}' & (\text{B } x_1) & y_1 & = & \text{rev}' \; x_1 \; (\text{B N})
\end{array}
$$

Thus, our construction avoids the unnecessary function symbol rev (with respect to the macro tree transducer presented in this example) by providing an initial value to the context parameters in the initial expression. Note that in this example the effect seems to be rather small, but in general for an arbitrary modular tree transducer $M = (F, m, \Delta, e, R)$ suitable for Construction 4.5, the original indirect construction outputs a macro tree transducer with at least $\text{card}(m^{-1}(1)) * \text{card}(m^{-1}(2))$ function symbols, whereas Construction 4.5 yields a macro tree transducer with only $\text{card}(m^{-1}(1))$ function symbols. □

8.2 Comparison to Wadler's transformation

In [Wad87] a transformation based on lists is presented. The setting of this transformation assumes a list result type, possibly polymorphic, and occurrences of the concatenate operator. This setting is incomparable to our modular tree transducer approach, since the general shape of equations of a program is less restricted and we cannot handle polymorphic types.

Construction 8.2 (Wadler's transformation [Wad87]). We assume a syntactically correct and well-typed functional program P, considered as a set of equations. For each equation

$$(f \; p_1 \; \ldots \; p_n \; = \; e) \in P$$

with a function symbol f with a list result type, patterns $p_1, \ldots, p_n$ with $n \in \mathbb{N}$ and an expression e, we introduce a new equation to the new program script P'

$$(f' \; p_1 \; \ldots \; p_n \; y \; = \; \text{append} \, e \, y) \in P',$$

where f' is a new function symbol and y is a new variable. Concatenation of lists is realized by append and the constructors : and [ ] are used to construct lists [89]. Then obviously $f \; p_1 \; \ldots \; p_n \; = \; f' \; p_1 \; \ldots \; p_n \; [\,]$. We then rewrite the right hand side to its normal form according to the following rewrite rules.

$$
\begin{array}{rcll}
\text{append} \, [\,] \, x & \to & x & (8.1) \\
\text{append} \, (x : y) \, z & \to & x : \, (\text{append} \, y \, z) & (8.2) \\
\text{append} \, (\text{append} \, x \, y) \, z & \to & \text{append} \, x \, (\text{append} \, y \, z) & (8.3) \\
\text{append} \, (f \; x_1 \; \ldots \; x_n) \, y & \to & f' \, x_1 \; \ldots \; x_n \, y & (8.4) \\
\text{append} \, (f' \; x_1 \; \ldots \; x_n \; y) \, z & \to & f' \, x_1 \; \ldots \; x_n \, (\text{append} \, y \, z) & (8.5)
\end{array}
$$

[89]See [BW88] for details on functional programs and lists.

The validity of this transformation can be shown with the techniques of [BD77]. □

Example 8.3 (Ackermann function). The ACKERMANN function can be computed by the following functional program.

$$\begin{array}{llll} \text{ack} & [\,] & y & = [\,] : y \\ \text{ack} & ([\,] : x) & [\,] & = \text{ack}\ x\ [\,[\,]\,] \\ \text{ack} & ([\,] : x) & ([\,] : y) & = \text{ack}\ x\ (\text{ack}\ ([\,] : x)\ y) \end{array}$$

This program script might seem rather awkward, but we needed to represent the natural numbers using a list data type in order for WADLER's transformation to be applicable. Intuitively, we use list of empty lists, where the number of empty lists encodes the natural number.

It is known that the ACKERMANN function is not primitive recursive [90], and thus cannot be computed by any modular tree transducer due to a result in [EV91]. Adding a concatenation to the program like

$$\text{douback}\ x\ y\ =\ \text{append}\ (\text{ack}\ x\ y)\ (\text{ack}\ x\ y)$$

with the usual equations for append and the list constructors. The function douback cannot be primitive recursive, else ack would be primitive recursive as well. Consequently Construction 4.5 or any construction which relies on modular tree transducers cannot be applied, but WADLER's transformation yields

$$\begin{array}{ll} & \text{append}\ (\text{append}\ (\text{ack}\ x\ y)\ (\text{ack}\ x\ y))\ z \\ \overset{(8.3)}{\to} & \text{append}\ (\text{ack}\ x\ y)\ (\text{append}\ (\text{ack}\ x\ y)\ z) \\ \overset{(8.4)}{\to} & \text{append}\ (\text{ack}\ x\ y)\ (\text{ack}'\ x\ y\ z) \\ \overset{(8.4)}{\to} & \text{ack}'\ x\ y\ (\text{ack}'\ x\ y\ z), \end{array}$$

thus $\text{douback}'\,x\,y\,z = \text{ack}'\,x\,y\,(\text{ack}'\,x\,y\,z)$ with initial expression $\text{douback}\,x\,y = \text{douback}'\,x\,y\,[\,]$ and

$$\begin{array}{lllll} \text{ack}' & [\,] & y & z & = [\,] : (\text{append}\ y\ z) \\ \text{ack}' & ([\,] : x) & [\,] & z & = \text{ack}'\ x\ [\,[\,]\,]\ z \\ \text{ack}' & ([\,] : x) & ([\,] : y) & z & = \text{ack}'\ x\ (\text{ack}\ ([\,] : x)\ y)\ z. \end{array}$$

The original function symbols still occur; as predicted by WADLER's characterization [91]; and the efficiency gain is minimal, but this example should only highlight the general incomparability. □

Example 8.4 (Inability to cope with trees). WADLER's transformation, as such, is not applicable to functions with result types (essentially) different from lists. Thus Example 5.4, though admissable for Construction 4.5, is not appropriate for WADLER's Transformation 8.2.

Hence by means of examples we have shown the general incomparability of the two constructions. Nevertheless, we want to consider both techniques on programs to which

[90] Gunter Dötzel. A function to end all functions. Algorithm: Recreational Programming 2.4, 16-17, October 1991.

[91] This is essentially different to the construction presented in the previous section, where the old function symbols are completely eliminated.

both are applicable. We immediately restrict ourselves to Construction 4.5, since the extended Construction 6.22 computes an essentially different output. Thus, we demand that the program specifies a restricted 2-modular tree transducer (a prerequisite of Construction 4.5) and, furthermore, the output signature Δ should obey $\Delta = \Delta^{(1)} \cup \Delta^{(0)}$ and $\mathrm{card}(\Delta^{(0)}) = 1$ (data structure should be list-like; a restriction of Construction 8.2).

Lemma 8.5 (Equivalence of Constructions 4.5 and 8.2). *Let*

$$M = (F, m, \Delta, e, R) \in \mathrm{ModTT}(\mathrm{TOP}, \mathrm{SUB})$$

with substitution variables Π*. Additionally, we require* $\Delta = \Delta^{(1)} \cup \Delta^{(0)}$ *and* $\mathrm{card}(\Delta^{(0)}) = 1$*. Then Constructions 4.5 and 8.2 produce equivalent results.*

Proof. Obviously the function symbol signature [92] and the initial expression coincide. Let $\rightarrow$ be the reduction relation induced by the rewrite rules of Construction 8.2. Hence the remaining proof needs to establish the equality

$$\mathrm{trans}_k(\mathrm{rhs}_M(f, \delta), \xi_1) \;=\; \mathrm{nf}_{\rightarrow}(\mathrm{append}\,\mathrm{rhs}_M(f, \delta)\,\xi_1)$$

for each $(f\ (\delta\ x_1\ \ldots\ x_k) = \mathrm{rhs}_M(f, \delta)) \in R$ with $k \in \mathbb{N}$, $f \in m^{-1}(1)$, $\delta \in \Delta^{(k)}$ and $\xi_1 \in \mathrm{RHS}(m^{-1}(1), \Delta, X_k, Y_1)$. We prove the statement via structural induction on $\mathrm{rhs}_M(f, \delta)$.

- **Induction base:**
  - Let $\Pi_1 = \mathrm{rhs}_M(f, \delta) \in \Pi$, then
  $$\mathrm{trans}_k(\Pi_1, \xi_1) \overset{(4.1)}{=} \xi_1 = \mathrm{nf}_{\rightarrow}(\xi_1) \overset{(8.1)}{=} \mathrm{nf}_{\rightarrow}(\mathrm{append}\,\Pi_1\,\xi_1).$$
  - Let $(f'\ x) = \mathrm{rhs}_M(f, \delta)$ for some $f' \in m^{-1}(1)$ and $x \in X_k$, then
  $$\mathrm{trans}_k((f'\ x), \xi_1) \overset{(4.2)}{=} (f'\ x\ \xi_1) = \mathrm{nf}_{\rightarrow}(f'\ x\ \xi_1) \overset{(8.5)}{=} \mathrm{nf}_{\rightarrow}(\mathrm{append}\ (f\ x)\ \xi_1).$$
- **Induction step:**
  - Let $(\delta'\, t_1) = \mathrm{rhs}_M(f, \delta)$ with $\delta' \in \Delta^{(1)}$ and $t_1 \in \mathrm{RHS}_{m^{-1}(1), m^{-1}(2), \Delta}(X_k, \emptyset)$, then by induction hypothesis
  $$\mathrm{trans}_k(t_1, \xi_1) \;=\; \mathrm{nf}_{\rightarrow}(\mathrm{append}\,t_1\,\xi_1).$$
  We conclude
  $$\begin{aligned} & \mathrm{trans}_k((\delta'\ t_1), \xi_1) \overset{(4.3)}{=} (\delta'\ \mathrm{trans}_k(t_1, \xi_1)) \\ \overset{\text{I.H.}}{=} \; & (\delta'\ \mathrm{nf}_{\rightarrow}(\mathrm{append}\,t_1\,\xi_1) \overset{(8.2)}{=} \mathrm{nf}_{\rightarrow}(\mathrm{append}\ (\delta'\ t_1)\ \xi_1). \end{aligned}$$

[92] The function symbols' names are different, but by the use of consistent renaming syntactic equivalence can be achieved.

– Let $(\mathrm{sub}_1\; t\; t_1) = \mathrm{rhs}_M(f,\delta)$ with $t, t_1 \in \mathrm{RHS}_{m^{-1}(1),m^{-1}(2),\Delta}(X_k,\emptyset)$ and $\mathrm{sub}_1 \in m^{-1}(2)^{(2)}$, then by induction hypothesis

$$\mathrm{trans}_k(t_1,\xi_1) \;=\; \mathrm{nf}_{\rightarrow}(\mathrm{append}\; t_1\; \xi_1)$$

and also

$$\mathrm{trans}_k(t,\mathrm{nf}_{\rightarrow}(\mathrm{append}\; t_1\; \xi_1)) \;=\; \mathrm{nf}_{\rightarrow}(\mathrm{append}\; t\; \mathrm{nf}_{\rightarrow}(\mathrm{append}\; t_1\; \xi_1)).$$

We conclude

$$\begin{aligned}
& \mathrm{trans}_k((\mathrm{sub}_1\; t\; t_1),\xi_1) \overset{(4.4)}{=} \mathrm{trans}_k(t,\mathrm{trans}_k(t_1,\xi_1)) \\
\overset{\text{I.H.}}{=}\; & \mathrm{trans}_k(t,\mathrm{nf}_{\rightarrow}(\mathrm{append}\; t_1\; \xi_1)) \overset{\text{I.H.}}{=} \mathrm{nf}_{\rightarrow}(\mathrm{append}\; t\; \mathrm{nf}_{\rightarrow}(\mathrm{append}\; t_1\; \xi_1)) \\
=\; & \mathrm{nf}_{\rightarrow}(\mathrm{append}\; t\; (\mathrm{append}\; t_1\; \xi_1)) \\
\overset{(8.3)}{=}\; & \mathrm{nf}_{\rightarrow}(\mathrm{append}\; (\mathrm{append}\; t\; t_1)\; \xi_1) = \mathrm{nf}_{\rightarrow}(\mathrm{append}\; (\mathrm{sub}_1\; t\; t_1)\; \xi_1).
\end{aligned}$$

As already stated in [Wad87] the rewrite rule (8.5) is strictly speaking not necessary, but assures confluency of $\rightarrow$. Thus, the statement is proven. □

We will illustrate this equivalence on Example 5.3, which fulfills the required restrictions of Lemma 8.5.

Example 8.6 (Example 5.3 and Construction 8.2). Immediate by Construction 8.2 we get the following equations [93]

$$\begin{array}{llll}
\mathrm{fib} & \mathrm{Z} & y_1 & = \mathrm{add}\ (\mathrm{S\ Z})\ y_1 \\
\mathrm{fib} & (\mathrm{S}\ x_1) & y_1 & = \mathrm{add}\ (\mathrm{add}\ (\mathrm{fib}'\ x_1)\ (\mathrm{fib}\ x_1))\ y_1 \\
\mathrm{fib}' & \mathrm{Z} & y_1 & = \mathrm{add\ Z}\ y_1 \\
\mathrm{fib}' & (\mathrm{S}\ x_1) & y_1 & = \mathrm{add}\ (\mathrm{fib}\ x_1)\ y_1.
\end{array}$$

and the original equations for add, which will remain unchanged. Applying rewriting (without renaming the function symbols) according to $\rightarrow$ of Construction 8.2 we get

$$\begin{array}{rcl}
\mathrm{add}\ (S\ Z)\ y_1 & \overset{(8.2)}{\rightarrow} & S\ (\mathrm{add}\ Z\ y_1) \overset{(8.1)}{\rightarrow} S\ y_1 \\
\mathrm{add}\ (\mathrm{add}\ (\mathrm{fib}'\ x_1)\ (\mathrm{fib}\ x_1))\ y_1 & \overset{(8.3)}{\rightarrow} & \mathrm{add}\ (\mathrm{fib}'\ x_1)\ (\mathrm{add}\ (\mathrm{fib}\ x_1)\ y_1) \\
& \overset{(8.4)}{\rightarrow} & \mathrm{add}\ (\mathrm{fib}'\ x_1)\ (\mathrm{fib}\ x_1\ y_1) \overset{(8.4)}{\rightarrow} \mathrm{fib}'\ x_1\ (\mathrm{fib}\ x_1\ y_1) \\
\mathrm{add}\ Z\ y_1 & \overset{(8.1)}{\rightarrow} & y_1 \\
\mathrm{add}\ (\mathrm{fib}\ x_1)\ y_1 & \overset{(8.4)}{\rightarrow} & \mathrm{fib}\ x_1\ y_1
\end{array}$$

and thereby the final equations are

$$\begin{array}{llll}
\mathrm{fib} & \mathrm{Z} & y_1 & = \mathrm{S}\ y_1 \\
\mathrm{fib} & (\mathrm{S}\ x_1) & y_1 & = \mathrm{fib}'\ x_1\ (\mathrm{fib}\ x_1\ y_1) \\
\mathrm{fib}' & \mathrm{Z} & y_1 & = y_1 \\
\mathrm{fib}' & (\mathrm{S}\ x_1) & y_1 & = \mathrm{fib}\ x_1\ y_1.
\end{array}$$

Obviously the equations are exactly those of M'_{fib}. □

[93] Here add = append, $(S\,u) = [\,] : u$ and $Z = [\,]$. Thereby, we again gain the representation already used in Example 8.3.

Currently, we cannot handle arbitrary context parameter use in the first module of the modular tree transducer in Construction 6.22 or 4.5, but WADLER's transformation 8.2 is able to tackle such modular tree transducers, e.g. the 2-modular tree transducer consisting of modules 2 and 3 of Example 3.4. Additionally, WADLER's note [Wad87] provides an efficiency analysis based on the notion of creativity. According to his definition all the function symbols of module 1 (the top-down tree transducer module in our restricted setting of Lemma 8.5) would necessarily be creative.

Following the argumentation of [Wad87], Construction 8.2 as well as Construction 4.5 by Lemma 8.5, will successfully eliminate all append-calls (i.e. turn the 2-modular tree transducer into a macro tree transducer) and, furthermore, will be non-deteriorating with respect to efficiency. Thus, we can already classify Examples 4.2 and 5.3, which suggests that the result of Construction 4.5 is always at least as efficient as its input with respect to the number of derivation steps of a call-by-name derivation relation (i.e. leftmost outermost reduction strategy or normal order reduction), since the imposed restrictions in Lemma 8.5 effectively outrule variable sharing and, thereby, degenerate call-by-need to call-by-name. In fact the generalization, that the construction is efficiency non-deteriorating with respect to a call-by-name derivation relation, is valid, although we will not formally prove it [94].

[94] The proof is very similar to the proof of Lemma 4.14 with additional effort to show that the resulting redex, if any, will be leftmost outermost. The key change is then reflected in the last sentence of the lemma, where $\kappa \Rightarrow_{M'}^{\mathrm{card}(P)} \kappa'$ turns into $\kappa \Rightarrow_{M'} \kappa'$, thus $\mathrm{card}(P) = 1$. With that additional condition the trans_M-mapping (as given there) will be surjective on the sentential forms of the derivation closures.

9 The implementation

The constructions of the thesis have been implemented in the HASKELL$^+$ system [Les99, HVM$^+$01]. The first subsection will explain the installation of the system on a particular architecture, which meets certain preconditions also mentioned there. Thereafter, the essential markups of HASKELL$^+$ are briefly introduced and example runs of the implementation are provided.

9.1 Installation

Please find the source code of the system on the enclosed disk. To install the system, please copy all files to a designated folder. If you have an installed HASKELL interpreter (e.g. HUGS [95] or GHCi [96]), then simply interpret the file "Main.hs" [97]. In case a HASKELL compilation system (e.g. GHC) is installed on your machine, invoke the compiler to create an executable of the provided system. Please consult your compiler manual, in order to find out how this is achieved. E.g. the invocation of the GHC 5 looks as follows:

```
ghc --make Main.hs
```

9.2 Running the system

Upon startup the system presents the following main interface [98].

```
*** Programs of the Haskell+ group ***

                                | 35|227|231| 55|107| 59|115| 47|255|547|...
                                |---|---|---|---|---|---|---|---|---|---|
    1 : hptree                  | x | x | x | x | x | x | x | x | x | x |
    2 : is_typ                  | x | x | x | x | x | x | x | x | x | x |
    4 : ATT              -> MAC |   |   | x |   |   |   |   |   |   |   |
    8 : MAC    o TOP     -> MAC |   |   |   |   | x |   |   |   |   |   |
   12 : Deforestation           |   |   |   |   |   |   |   | x |   |   |...
        ...
  256 : hptrans                 |   |   |   |   |   |   |   |   |   |   |...
  512 : MAC   -> TOP    o YIELD |   |   |   |   |   |   |   |   |   | x |
  516 : nc-MAC o wsu-MAC -> MAC |   |   |   |   |   |   |   |   |   |   |
 1024 : exportMacs              |   |   |   |   |   |   |   |   |   |   |
 2048 : to Haskell              |   |   |   |   |   |   |   |   |   |   |...

   -1 : Enter modular tree transducer section

Enter a number (0=end):
```

The constructions of this thesis are grouped in the modular tree transducer section. Entering `-1` at the prompt, yields the submenu to follow.

---

[95] See <http://www.haskell.org/hugs>

[96] See <http://www.haskell.org/ghc>

[97] Refer to the user manual of your interpreter for details on how to invoke the interpreter and how to interpret source files.

[98] Some parts of the tabular are omitted.

```
*** Programs of the modular tree transducer section ***

    1 : Recognize modular tree transducer
    2 : Check for top-down tree transducer module
    3 : Check for substitution module
    4 : Check for non-copying module
    --------------------------------------------------
   10 : Apply accumulation technique
   11 : Apply accumulation technique (fully automatic)

Enter a number (0=back):
```

1 Given a Haskell$^+$ input file and the function symbol occuring in the initial expression, try to recognize a modular tree transducer (cf. Definition 3.3), where this function symbol occurs in the first module. The output is either a list of function symbols partitioned into modules or the message that no modular tree transducer with this function symbol was found.

2 Given a Haskell$^+$ input file and a list of function symbols, detect whether the macro tree transducer module containing those function symbols is a top-down tree transducer module (cf. Definition 3.5). The output is either affirmative or rejecting.

3 Given a Haskell$^+$ input file and a list of function symbols, detect whether the macro tree transducer module containing those function symbols is a substitution module (cf. Definition 4.1) [99]. The output either confirms that the specified module is a substitution module by outputting the substitution variables or rejects.

4 Given a Haskell$^+$ input file and a list of function symbols, detect whether the macro tree transducer module containing those function symbols is non-copying (cf. Definition 6.6). The output is either affirmative or rejecting.

In general, if the supplied information is incorrect, then the behavior of the functions is unspecified, in particular, if the supplied function symbols do not constitute a valid macro tree transducer module. Next we present an example Haskell$^+$ input file and introduce the important markups.

```
begindata List
  data List = A List | B List | N
enddata

beginmag Rev
  input List
  syn rev :: List -> List

  rev   N   = N
  rev (A x) = app (rev x) (A N)
  rev (B x) = app (rev x) (B N)
endmag

beginmag App
  input List
  syn app :: List -> List -> List
```

```
  app   N   y = y
  app (A x) y = A (app x y)
  app (B x) y = B (app x y)
endmag
```

This input file implements the modular tree transducer M_{rev} of Example 4.2. Syntactically, the block inside the delimiters `begindata` and `enddata`, hereafter refer to as *data-block*, defines algebraic data structures in the syntax of regular HASKELL. This corresponds to the specification of the ranked alphabet. The keywords `beginmag` and `endmag` delimit a macro attribute grammar [AJKV98] also called *mag-block*, which we will use to define macro tree transducer modules. Note that each block (data-blocks and mag-blocks) is named. Several mag-blocks may form one macro tree transducer module, but one mag-block is never split into several macro tree transducer modules of the same modular tree transducer. The `input` keyword is followed by the data-block which defines the ranked alphabet. Thereafter, the type of each function symbols is given in standard HASKELL syntax prefixed by the keyword `syn`. Finally, the equations are stated in the syntax of the definition of modular tree transducers in Definition 3.3.

We will demonstrate the options on this example. First, we want the system to recognize the modular tree transducer M_{rev} [100] of Example 4.2 in the above script. Consequently, we select option `1` in the modular tree transducer section and proceed by inputting the file name of the input file (its suffix is usually `.hp`) and inputting `rev` as the function symbol occurring in the initial expression of M_{rev}. The dialogue is displayed below.

```
Enter the file name of the Haskell+ source code (without .hp): rev
Enter the name of the function in the initial expression (outermost): rev
```

The system answers:

```
Modular tree transducer:
        Module 1 :  ["rev"]
        Module 2 :  ["app"]
```

This is the partition of function symbols into modules. With this information at hand, we check whether Construction 4.5 is applicable by first selecting option `2`, supplying the input file name and the function symbols of the first module, thereafter selecting option `3` and, finally supplying the input file name again along with the function symbols of the second module. Below the important part of the interaction is displayed.

```
Enter a number (0=back): 2

Enter the file name of the Haskell+ source code (without .hp): rev
Enter the modules' functions in quotes separated by colons: "rev"

Module is a top-down tree transducer module.

...

Enter a number (0=back): 3

Enter the file name of the Haskell+ source code (without .hp): rev
```

[100] Henceforth, we will not distinguish between the syntactic specification of a modular tree transducer according to Definition 3.3 and the specification using the HASKELL^{+} syntax.

```
Enter the modules' functions in quotes separated by colons: "app"

Module is a substitution module with substitution variables  ["N"]
```

We will now apply Construction 4.5 to M_{rev} of Example 4.2 (strictly speaking to the HASKELL$^+$ input file above, which specifies M_{rev}) to gain M'_{rev} of Example 5.2. Therefore we select option `10` in the modular tree transducer section, supply the input file name, the function symbol occurring in the initial expression, the function symbols of the modules and the substitution variables. Finally, we output the result on stdout (standard output, usually the screen) by pressing `Return`.

```
Enter a number (0=back): 10

Enter the file name of the Haskell+ source code (without .hp): rev
Enter the name of the function in the initial expression (outermost): rev
Enter the first modules' functions in quotes separated by colons: "rev"
Enter the substitution modules' functions in quotes separated by colons: "app"
Enter the substitution variables in quotes separated by colons: "N"

Enter the Haskell+ output file name (without .hp, Return=stdout):

...
{-# RULES "COMPOSITION" forall u.
        rev u = rev_ u N
#-}
beginmag Acc_rev
  input List
  syn rev_ :: List -> List -> List
  rev_ (A x1) y1  =  rev_ x1 (A y1)
  rev_ (B x1) y1  =  rev_ x1 (B y1)
  rev_ N y1  =  y1
endmag
```

The left-out part in the dialogue above is the original program, which is outputted together with the new mag-block. In the comment above the mag-block, information on how to replace the initial expression is supplied. Altogether, we note that the result is M'_{rev} of Example 5.2. Since the whole procedure is rather annoying, we added an option, namely option `11`, which given an input file name and the function symbol occurring in the initial expression, applies Construction 4.5 automatically, if applicable. The program is not altered, if at some stage a precondition is not fulfilled.

The stated instructions also apply for Construction 6.22, since ultimately Construction 6.22 is employed to compute the resulting mag-block [101]. To check the preconditions of the extended construction, a test for non-copying modules was added. We illustrate this on M_{usa}.

```
Enter a number (0=back): 4

Enter the file name of the Haskell+ source code (without .hp): usa
Enter the modules' functions in quotes separated by colons: "coll"

Module is non-copying.
```

[101] Construction 4.5 is subsumed by Construction 6.22

```
...
Enter a number (0=back): 11

Enter the file name of the Haskell+ source code (without .hp): usa
Enter the name of the function in the initial expression (outermost): coll


Enter the Haskell+ output file name (without .hp, Return=stdout):

begindata Stripes
  data Str = Z | B Str | W Str | R Str
enddata

beginmag Coll [Mac,Mat,Su,Swp,Wp,Wsu,Xlin,Xnd,Ylin,Ynd]
  input Stripes
  syn coll :: Str -> Str -> Str
  coll Z y  =  y
  coll (B x) y  =  app y (coll x Z)
  coll (R x) y  =  R (coll x y)
  coll (W x) y  =  coll x (W y)
endmag

beginmag App [Mac,Mat,Su,Swp,Tl,Wp,Wsu,Xlin,Xnd,Ylin,Ynd]
  input Stripes
  syn app :: Str -> Str -> Str
  app Z y  =  y
  app (B x) y  =  B (app x y)
  app (R x) y  =  R (app x y)
  app (W x) y  =  W (app x y)
endmag

{-# RULES "COMPOSITION" forall u y1.
        coll u y1 = coll_ u (app y1 (par_1coll_1 u Z Z)) Z
#-}
beginmag Acc_coll
  input Stripes
  syn coll_ :: Str -> Str -> Str -> Str
  syn par_1coll_1 :: Str -> Str -> Str -> Str
  coll_ Z y1 y2  =  y1
  coll_ (B x1) y1 y2  =  y1
  coll_ (R x1) y1 y2  =  R (coll_ x1 y1 y2)
  coll_ (W x1) y1 y2  =  coll_ x1 (W y1) y2
  par_1coll_1 Z y1 y2  =  y2
  par_1coll_1 (B x1) y1 y2  =  coll_ x1 (par_1coll_1 x1 Z y2) y2
  par_1coll_1 (R x1) y1 y2  =  par_1coll_1 x1 Z y2
  par_1coll_1 (W x1) y1 y2  =  par_1coll_1 x1 Z y2
endmag
```

Finally, we want to illustrate the use of the refined constructions. Therefore, we input `11`, load an input file containing M_{doub} [102] and supply `doub` as the function symbol in the initial expression.

[102] The input program is contained in the output program.

```
Enter a number (0=back): 11

Enter the file name of the Haskell+ source code (without .hp): doub
Enter the name of the function in the initial expression (outermost): doub

Program is possibly deteriorating efficiency,
  output non-deteriorating Haskell code [Y/n]:

Enter the Haskell+ output file name (without .hp, Return=stdout):

begindata Tree
  data Tree = Sigma Tree Tree | Alpha
enddata

beginmag Doub [Att,Mac,Mat,Prod,Su,Swp,Tl,Top,Wp,Wsu,Xlin,Xnd,Ylin,Ynd]
  input Tree
  syn doub :: Tree -> Tree
  doub Alpha   =  Alpha
  doub (Sigma x1 x2)   =  sub (Sigma Alpha Alpha) (Sigma (doub x1) (doub x2))
endmag

beginmag Sub [Mac,Mat,Swp,Tl,Wp,Wsu,Xlin,Xnd,Ynd]
  input Tree
  syn sub :: Tree -> Tree -> Tree
  sub Alpha y1  =  y1
  sub (Sigma x1 x2) y1  =  Sigma (sub x1 y1) (sub x2 y1)
endmag

{-# RULES "COMPOSITION" forall u.
        doub u = doub_ u Alpha
#-}
doub_ :: Tree -> Tree -> Tree
doub_ (Sigma x1 x2) y1 = Sigma z1AT1_1 z1AT1_1
        where z1AT1_1 = Sigma (doub_ x1 y1) (doub_ x2 y1)
doub_ (Alpha) y1 = y1
```

Note that the system detected the possibility of a degradation of efficiency which can be avoided by introducing where-clauses. Accordingly, the system asks the user, whether the system shall introduce the where-clauses (thus return HASKELL code) [103]. The resulting HASKELL code fragment is also displayed above and, if we would have stored the resulting HASKELL$^+$ code in a file, then a transformation accessible from the main menu would have allowed us the translation of output into a HASKELL program, which could then be executed.

[103]The default value for this question is yes, so there is no need to supply a value in our case.

10 Conclusions

10.1 Future work

Since modular tree transducers exactly cover the class of primitive recursive tree functions, it would be worthwhile to study different constructions on modular tree transducers along with their efficiency impact.

In this thesis we extended the basic construction of [KGK01] to non-copying macro tree transducer modules as the first module. However, one could also try to lift or relax the restrictions on the second module. We can imagine relabelling modules, for example. Along with those extended constructions their efficiency impact has to be studied.

Even for the presented extended construction, further efficiency considerations could be enlightening. Especially, the results of [Voi02] should successfully be applied in the modular tree transducer setting. Eventually, one should also try to better approximate the efficiency of the output macro tree transducer of Construction 6.15 depending on the efficiency of its inputs. First approximations can be found in [Voi02].

Another interesting approach to efficiency analysis is the deduction of recurrences describing the efficiency of a program and, afterwards, try to solve the recurrences or at least establish relationships between them. A good introduction to efficiency analysis of functional programs is [San90].

Furthermore, tree series transducers [FV01], which internally capture costs could prove to be effective in the analysis of composition techniques. Finally, composition techniques and corresponding efficiency considerations should be lifted to higher order functional programs.

10.2 Acknowledgements

First of all, I would like to thank Armin Kühnemann for his superior tutelage and guidance. Moreover, he found the patience to scrutinize pre-final versions of this thesis, which; in my humble opinion; improved readability considerably. Together with Janis Voigtländer, he constantly checked the progress of the thesis and made priceless comments on work in progress.

I would also like to thank Heiko Vogler for spurring me to enter the field of theoretical computer science. It was not until I attended his lecture “Foundations of Programming”, that my interest in this field of study was awoken. In further lectures and seminars Prof. Vogler and his colleagues brought me up the tree concerning formal language theory.

In addition, I want to express my gratitude towards the DAAD (German Academic Exchange Service) and the NSC (National Science Council) for giving me the opportunity to study in Taiwan. It was truly an amazing experience, and I would like to thank all involved making it such. The DAAD also enabled me to study with Prof. Fülöp in Hungary for a week.

Finally, I feel grateful towards my friends and foremost my family for their constant support.

References

[AFM+95] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. A call-by-need lambda calculus. In *Proceedings of 22nd Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1995.

[AJKV98] Inge Adamski, Bernd Jokubeit, Armin Kühnemann, and Heiko Vogler. Incremental evaluators for ordered macro attribute grammars. Technical report TUD/FI98/10, Dresden University of Technology, 1998.

[Ave95] Jürgen Avenhaus. *Reduktionssysteme - Rechnen und Schließen in gleichungsdefinierten Strukturen*. Springer, Heidelberg/Berlin, first edition, 1995.

[Bar84] Hendrik P. Barendregt. *The Lambda Calculus - its syntax and semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science, Amsterdam, first revised edition, 1984.

[BD77] Rod M. Burstall and John Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.

[Bir84] Richard Bird. The promotion and accumulation strategies in transformational programming. *ACM Transactions on Programming Languages and Systems*, 6(4):487–504, 1984.

[BKdV+02] Marc Bezem, Jan Willem Klop, Roel de Vrijer, et al. *Term Rewriting Systems*. Cambridge University, Amsterdam/Bergen, 2002.

[Blo01] Stefan Blom. *Term Graph Rewriting - syntax and semantics*. PhD thesis, University of Amsterdam/Institute for Programming research and Algorithmics, Amsterdam, 2001.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University, Cambridge, first edition, 1998.

[Boi91] Eerke Boiten. The many disguises of accumulation. Technical Report 91-26, University of Nijmegen, December 1991.

[BS81] Stanley Burris and Hantamantagouda P. Sankappanavar. *A Course in Universal Algebra*, volume 78 of *Graduate Texts in Mathematics*. Springer, New York, first edition, 1981. Corrected version available at: http://www.thoralf.uwaterloo.ca/htdocs/ualg.html.

[Bün98] Reinhard Bündgen. *Termersetzungssysteme - Theorie, Implementierung, Anwendung*. Lehrbuch. Vieweg, Braunschweig/Wiesbaden, first edition, 1998.

[BvEG+87] Hendrik P. Barendregt, Marko C. J. D. van Eekelen, John R. W. Glauert, Richard Kennaway, Rinus Plasmeijer, and Ronan Sleep. Term graph rewriting. In *PARLE Parallel Architectures and Languages Europe, Volume II: Parallel Languages*, volume 259 of *Lecture Notes in Computer Science*, pages 141–158, Eindhoven, The Netherlands, June 1987. Springer.

[BW88] Richard Bird and Philip Wadler. *Introduction to Functional Programming.* International Series in Computer Science. Prentice Hall, Hemel Hempstead, Hertfordshire, first edition, 1988.

[Cla96] David Clark. *Term Graph Rewriting and Event Structures.* PhD thesis, Imperial College, London, 1996.

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

[Eng80] Joost Engelfriet. *Some open questions and recent results on tree transducers and tree languages*, chapter Formal language theory: perspectives and open problems, pages 241–286. Academic Press, New York, first edition, 1980.

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

[EV91] Joost Engelfriet and Heiko Vogler. Modular tree transducers. *Theoretical Computer Science*, 78:267–303, 1991.

[FV98] Zoltán Fülöp and Heiko Vogler. *Syntax-directed semantics – Formal models based on tree transducers.* Monographs in Theoretical Computer Science. Springer, Berlin/Heidelberg, first edition, 1998.

[FV01] Zoltán Fülöp and Heiko Vogler. Tree transducers with costs. available at: <http://www.orchid.inf.tu-dresden.de/gdp/forschung/publikation.html>, 2001.

[HVM$^+$01] Matthias Höff, Rainer Vater, Andreas Maletti, Armin Kühnemann, and Janis Voigtländer. Tree transducer based program transformations for Haskell$^+$. Progress report, Dresden University of Technology, March 2001.

[Jou93] Jean-Pierre Jouannaud. Introduction to rewriting. In Hubert Comon and Jean-Pierre Jouannaud, editors, *Term Rewriting - French Spring School of Theoretical Computer Science*, volume 909 of *Lecture Notes in Computer Science*, pages 1–15, Font Romeux, France, May 1993. Springer.

[KGK01] Armin Kühnemann, Robert Glück, and Kazuhiko Kakehi. Relating accumulative and non-accumulative functional programs. In *Rewriting Techniques and Applications*, volume 2051 of *Lecture Notes in Computer Science*, pages 154–168, Utrecht, The Netherlands, May 2001. Springer.

[Küh97] Armin Kühnemann. *Berechnungsstärken von Teilklassen primitiv-rekursiver Programmschemata.* PhD thesis, Dresden University of Technology, Dresden, 1997. Shaker, Aachen.

[Küh98] Armin Kühnemann. Benefits of tree transducers for optimizing functional programs. In *Foundations of Software Technology & Theoretical Computer Science*, volume 1530 of *Lecture Notes in Computer Science*, pages 146–157, Chennai (India), 1998. Springer.

[Küh01] Armin Kühnemann. Relating accumulative and non-accumulative functional programs. Talk, Dresden University of Technology, 2001.

[Les99] Christian Lescher. Entwurf und Implementierung einer Eingabesprache für ein System zur Erzeugung syntaxgesteuerter Editoren. Student thesis, Dresden University of Technology, 1999.

[Pot90] Michael D. Potter. *Sets: An Introduction.* Oxford University, New York, first edition, 1990.

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

[San90] David Sands. *Calculi for Time Analysis of Functional Programs.* PhD thesis, Imperial College of Science, University of London, September 1990.

[Str71] Raymond Strong. Translating recursion equations into flowcharts. *Journal of Computer and System Sciences*, 5:254–285, 1971.

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

[Tho99] Simon Thompson. *Haskell - the craft of functional programming.* International Computer Science Series. Addison-Wesley, first edition, 1999.

[VK01] Janis Voigtländer and Armin Kühnemann. Composition of functions with accumulating parameters. Technical report TUD-FI01-08, Dresden University of Technology, August 2001.

[Voi01] Janis Voigtländer. Composition of restricted macro tree transducers. Master's thesis, Dresden University of Technology, Dresden, March 2001.

[Voi02] Janis Voigtländer. Conditions for efficiency improvement by tree transducer composition. In S. Tison, editor, *13th International Conference on Rewriting Techniques and Applications*, volume 2378 of *Lecture Notes in Computer Science*, pages 222–236, Copenhagen, Denmark, July 2002. Springer.

[Wad87] Philip Wadler. The concatenate vanishes. Note, December 1987.

[Wad90] Philip Wadler. Deforestation: transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 1990.

[Wec92] Wolfgang Wechler. *Universal Algebra for Computer Scientists*, volume 25 of *Monographs on Theoretical Computer Science.* Springer, Heidelberg/Berlin, first edition, 1992.