Technische Universität Dresden
Fakultät Informatik
Diplomarbeit / Masters Thesis

zum Thema / on the subject

Composition of restricted Macro Tree Transducers

zur Erlangung des akademischen Grades / to receive the academic degree

Diplom-Informatiker / Master of Computer Science

von / by

Janis Voigtländer
Matrikelnummer 2482099

Verantwortlicher Hochschullehrer: Prof. Dr.-Ing. habil. Heiko Vogler
Lehrstuhl Grundlagen der Programmierung
Institut für Theoretische Informatik
Betreuer: Dr.-Ing. Armin Kühnemann

begonnen am 27.09.2000
abgegeben am 26.03.2001

Erklärung

Hiermit versichere ich, daß ich die vorliegende schriftliche Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Dresden, den 26. März 2001

Janis Voigtländer

Contents

1 Introduction **1**

2 Preliminaries **6**

2.1 Functions on Trees 7

2.2 Macro Tree Transducers 8

2.3 Proving a Lemma 12

3 The Construction **14**

3.1 The Idea 14

3.2 Formal Development 20

3.3 The symmetric Result does not hold 31

4 Tidying up **33**

4.1 Removing superfluous Context Parameters 34

4.2 Eliminating Copy-states 38

4.3 Turning non-terminating Programs into terminating ones 43

5 Useful Things **46**

5.1 A Size-Height Property for non-copying Macro Tree Transducers 46

5.2 Counting Reduction Steps of non-copying Macro Tree Transducers 50

5.3 An Implementation 53

6 Conclusions **55**

Bibliography **57**

A Proving the Correctness of Construction 3.2 **59**

Chapter 1

Introduction

An important style of writing programs in a functional language is to define new functions by composition of existing functions. This means that the result of a function application is passed as argument to another function. This technique of solving an overall problem by combining solutions of partial problems, which might be solved by predefined or self-written functions, allows to write simple and modular programs, but can cause inefficiencies. If the created intermediate results are structured objects—for example lists or trees—their creation and eventual destruction will consume time and memory space. Furthermore, it is possible that, e.g., elements of a list are traversed more often, than would really be necessary for solving the overall problem. Often, other function definitions can be given that solve the same problem, but avoid the mentioned inefficiencies. Thus, we would like to have transformation techniques, which enable us to optimize functions written in the modular style, by eliminating intermediate data structures. Several such techniques have been studied in the literature, e.g., the *fold/unfold-technique* from [BD77], its algorithmic instances *supercompilation* [Tur86] and *deforestation* [Wad90], *catamorphism fusion* [Mal89, MFP91] and *shortcut deforestation* [GLP93].

In this thesis, we follow an approach for eliminating intermediate results, which is based on the theory of tree transducers. Specifically, we consider *Macro Tree Transducers* [Eng80], which can represent a large class of functional programs[1]. Composition of functions then corresponds to consecutive execution of Macro Tree Transducers, that is, the output of one transducer is given as input to the second one, and so on (we will only consider the composition of two Macro Tree Transducers). The goal of avoiding intermediate data structures corresponds to the problem of replacing such a composition of Macro Tree Transducers by a single Macro Tree Transducer, which computes the same function. It is well known that this is not possible in general, because the class of Macro Tree Transductions is not closed under composition (cf. [EV85]). For subclasses, however, such constructions can be found.

For illustration of the problem of intermediate results and as informal introduction to the concept of Macro Tree Transducers, consider the following example. Assume given a representation of arithmetic terms, built from two variables and the binary operations addition and multiplication, as trees with nullary symbols A and B and binary symbols $+$ and $*$ (this representation corresponds to an algebraic data type in a functional language, e.g., `data Term = A | B | + Term Term | * Term Term` in Haskell). Further, assume given a function for computing the prefix-notation of such a term as monadic tree of now

[1]Macro Tree Transducers are extended schemes of primitive recursion, allowing simultaneous definition of several functions and nested function calls in parameter positions.

unary symbols A, B, $+$, $*$, and a nullary symbol ϵ for denoting the end of the list[2]. As a Macro Tree Transducer, this function *pre* can be given by the following rules:

$$\begin{array}{lll}
\text{(i)} \ : \ pre(+(x_1,x_2),y) & \rightarrow & +(pre(x_1,pre(x_2,y))) \\
\text{(ii)} \ : \ pre(*\,(x_1,x_2),y) & \rightarrow & *\,(pre(x_1,pre(x_2,y))) \\
\text{(iii)} : \ pre(A,y) & \rightarrow & A(y) \\
\text{(iv)} : \ pre(B,y) & \rightarrow & B(y)
\end{array}$$

These rules can be seen as just another notation for the defining equations in a functional language. In order to compute the prefix-notation of a term t, the rules (i)–(iv) are used to exhaustively rewrite the initial expression $pre(t,\epsilon)$, as shown in Figure 1.1.

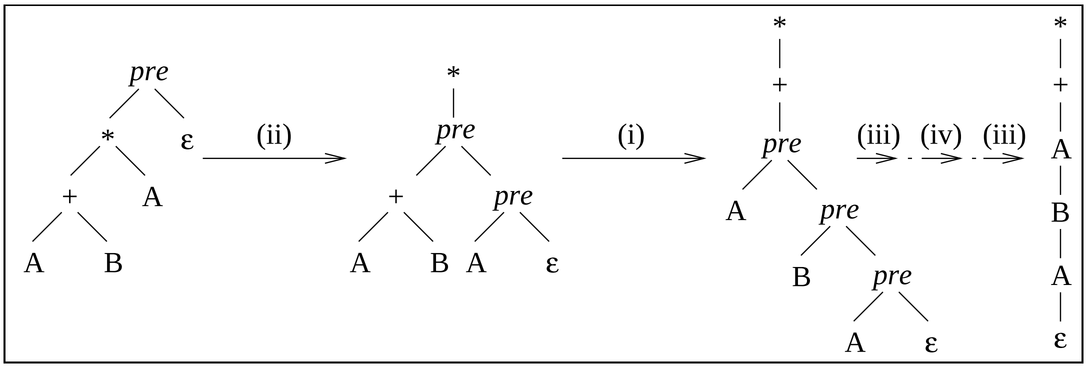

Figure 1.1: Computation by rewriting

Now, consider the problem of computing, for a given term as above, a sequence of instructions for a stack-machine with two registers and instructions for addition and multiplication (represented as monadic tree, labelled with instructions $LOAD_A$, $LOAD_B$, ADD and MUL). Instead of writing a program for this problem from scratch, we might realize that we can solve the task by reversing the prefix-notation of the given term and replacing labels A, B, $+$ and $*$ by $LOAD_A$, $LOAD_B$, ADD and MUL, respectively. Thus, we define the following function aux[3]:

$$\begin{array}{lll}
\text{(v)} \ \ : \ aux(A(x),y) & \rightarrow & aux(x,LOAD_A(y)) \\
\text{(vi)} \ : \ aux(B(x),y) & \rightarrow & aux(x,LOAD_B(y)) \\
\text{(vii)} : \ aux(+(x),y) & \rightarrow & aux(x,ADD(y)) \\
\text{(viii)} : \ aux(*\,(x),y) & \rightarrow & aux(x,MUL(y)) \\
\text{(ix)} \ : \ aux(\epsilon,y) & \rightarrow & y
\end{array}$$

The instruction sequence for a given term t can now be computed by composing the existing function *pre* and our new auxiliary function, namely by rewriting the composite expression $aux(pre(t,\epsilon),\epsilon)$ with rules (i)–(ix), as shown in Figure 1.2.

This modular solution, however, is inefficient, because it creates and consumes an intermediate result. Depending on the used rewriting strategy, respectively the evaluation order of our functional language, this intermediate data structure might never exist as a whole, but nevertheless, for all of its nodes, memory cells have to be allocated and later deallocated. Also, even with a lazy evaluation strategy, this program performs a superfluous traversal on the intermediate list. To avoid these inefficiencies, we could instead have given the

[2]In most functional languages, one would use a built-in type of lists instead, but for our illustration this representation of lists as monadic trees is more handy. In principle, our techniques are also transferable to, e.g., polymorphic lists.

[3]On a built-in list type, this auxiliary function might be obtained from fusion of a `map`-function and the standard list `reverse`-function.

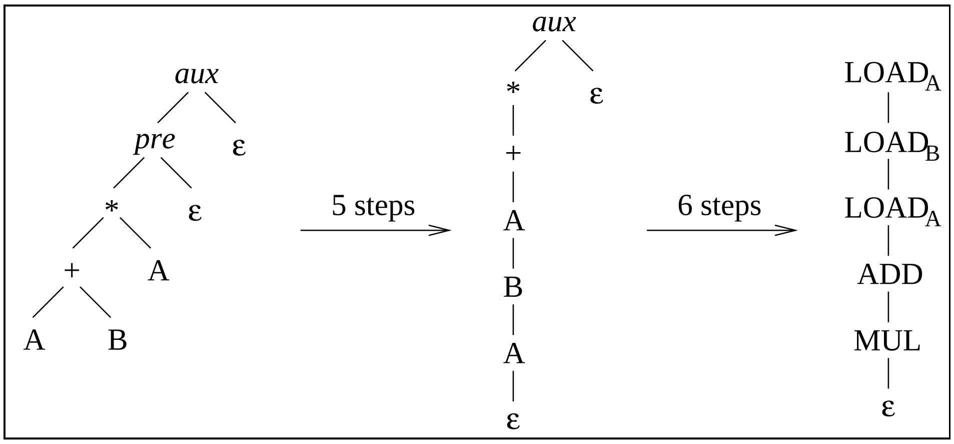

Figure 1.2: Creating and consuming an intermediate data structure

following function definition:

$$
\begin{array}{lllll}
\text{(x)} & : & ins(+(x_1, x_2), y) & \rightarrow & ins(x_2, ins(x_1, ADD(y))) \\
\text{(xi)} & : & ins(*\,(x_1, x_2), y) & \rightarrow & ins(x_2, ins(x_1, MUL(y))) \\
\text{(xii)} & : & ins(A, y) & \rightarrow & LOAD_A(y) \\
\text{(xiii)} & : & ins(B, y) & \rightarrow & LOAD_B(y)
\end{array}
$$

If now, for a given term t, we use rules (x)–(xiii) on $ins(t, \epsilon)$, we will calculate the same instruction sequence as before with the modular solution, but without creating and traversing the intermediate list (see Figure 1.3).

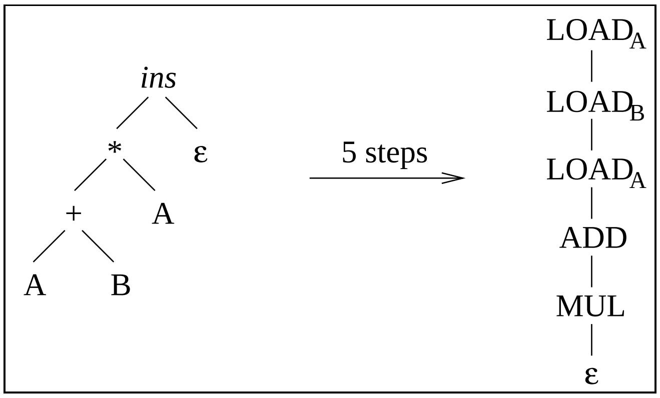

Figure 1.3: Avoiding the intermediate data structure

For the programmer, however, it would probably be more difficult (and, arguably, bad non-modular style) to come up with this more efficient solution to the problem, instead of reusing the already existing function for the prefix-notation of a term together with a simple list processing function.

Consequently, it would be worthwhile to have a technique, which can transform the modular solution (i)–(ix) into the efficient solution (x)–(xiii). To the best of our knowledge, techniques such as deforestation and catamorphism fusion cannot perform the optimization that we want to achieve here. An approach that is applicable to our example is based on *attribute grammars* (cf. [Knu68]) and was proposed independently in [Küh98] and [CDPR99]. The idea is to transform the two functions (in our formalism represented by two restricted Macro Tree Transducers), which we want to compose, into attribute grammars, respectively *Attributed Tree Transducers* [Fül81], which are an abstraction of attribute grammars in the same sense as Macro Tree Transducers are used as primitive recursive program schemes. If the first Attributed Tree Transducer fulfils the *single-use* restriction (cf. the *syntactic single-use requirement* in [GG84, Gie88]), which essentially means that every attribute occurrence in a tree may be used at most once in calculating the values of other attribute occurrences, the two transducers can be composed into a single Attributed Tree Transducer (based on composition results from [Gan83, GG84, Gie88]). Applying a construction based on [Fra82], this Attributed Tree Transducer can then be transformed into a Macro Tree Transducer, thus giving a functional program for the composition of the two original functions, but without producing and consuming the intermediate result.

This approach does not work for all Macro Tree Transducers, because only restricted Macro Tree Transducers can be transformed into Attributed Tree Transducers, which can then be composed. One such restriction is the property of a Macro Tree Transducer to be *weakly single-use*, which means that at every occurrence of the input tree, recursive computations of functions on subtrees are restricted to be used at most once. Moreover, in order to obtain a single-use Attributed Tree Transducer—such that the composition result for Attributed Tree Transducers becomes applicable—, the first Macro Tree Transducer has to be further restricted to be *single-use*, which in addition to the restriction of being weakly single-use means that context parameters cannot be copied. Figure 1.4 shows the transformation steps we have to take, where arrows indicate transformations and the semicolon stands for composition of tree transduction classes. Here, the class of functions computable by Macro Tree Transducers is denoted as MAC, the class of Attributed Tree Transductions as ATT, and the restrictions single-use and weakly single-use are indicated by subscripts su and wsu, respectively.

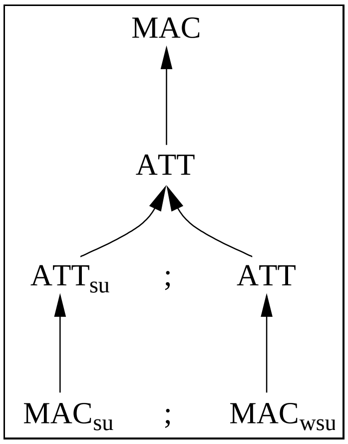

Figure 1.4: Indirect composition

This composition of restricted Macro Tree Transducers by indirection via Attributed Tree Transducers has been implemented in the Haskell$^+$ program transformation system [Les99, HVM$^+$01]. There are several reasons, why we are interested in a direct construction for composing Macro Tree Transducers, without the above indirection. First, there would be benefits for the implementation, both, because a direct construction could be implemented more efficiently, and because the implementor—e.g., a compiler constructor—could work directly on functional programs, instead of having to consider the formalism of attribute grammars, just for optimization's sake[4]. Second, there is a chance that a direct construction would produce better program code than the indirection by several transformations, because every single program transformation tends to introduce a certain "ballast" into the program, like, e.g., superfluous function parameters. Third, we

[4]This argument does not apply to the Haskell$^+$ system, which was designed as extension to the Haskell language and allows the definition of attribute grammars.

want to broaden the applicability of Macro Tree Transducer composition by generalizing the result $MAC_{su}; MAC_{wsu} \subseteq MAC$. As we already noted, we cannot expect to be able to compose two arbitrary Macro Tree Transducers, because the class of Macro Tree Transductions is known not to be closed under composition. But, we can aim for weaker restrictions on the Macro Tree Transducers than necessary so far, which still allow a composition. The success of this intention is limited, if we stick to the indirect construction via Attributed Tree Transducers, because then we can only compose Macro Tree Transducers that can be transformed into Attributed Tree Transducers[5], which is not always possible, because we have a strict inclusion $ATT \subset MAC$ (cf. [Eng80, Fra82]). A direct composition construction has no such a priori limitation. In fact, we will prove the characterization $MAC_{nc}; MAC_{wsu} = MAC$, which is a generalization of the result quoted above, because every single-use Macro Tree Transducer is, by definition, *non-copying*.

Another problem that we want to consider in this thesis is the already mentioned "ballast", which can be introduced by our program transformations. Besides the problem of introducing superfluous function parameters (which will be detected by a fixpoint construction), another phenomenon is of special interest. In [CDPR99] it was called the problem of *copy-rules* and refers to the introduction—by the composition of attribute grammars—of rules that lead to superfluous traversals through the input tree. Thus, it can happen that the intermediate data structure is eliminated, but the resulting program is no more efficient than the original one (with respect to the number of reduction steps). In [CDPR99] it is stated that in most cases, a static analysis of the Attributed Tree Transducer can detect these copy-rules, such that they can be eliminated before the transformation into the final Macro Tree Transducer. This analysis is not used in our construction, because we do not use the indirection via Attributed Tree Transducers. However, after having introduced our direct construction for $MAC_{nc}; MAC_{wsu} \subseteq MAC$, we will note that the problem of superfluous traversals through the input tree is not specific to the composition of attribute grammars, but also arises from our construction. We will generalize the concept of copy-rules to the concept of *copy-states* on the level of our functional programs and present a construction, which detects and removes all copy-states of a given Macro Tree Transducer, thus eliminating superfluous traversals. Our direct composition construction together with the presented post-processing constructions will then be able to transform the modular program—rules (i) to (ix)—from the introductory example into the more efficient program corresponding to rules (x)–(xiii).

This thesis is divided into six chapters and an appendix. In Chapter 2 we define necessary notations and introduce the basic concepts of Macro Tree Transducers and a proof principle. In Chapter 3 we develop our main construction for composing two Macro Tree Transducers—the first of which is non-copying and the second one weakly single-use—to a single Macro Tree Transducer. We will also show that a symmetric result—composing a weakly single-use Macro Tree Transducer and a non-copying one in this order—is not possible in general. In Chapter 4 we develop two fixpoint constructions for post-processing Macro Tree Transducers obtained by the composition construction, eliminating inefficiencies caused by superfluous context parameters and by copy-states, respectively. In Chapter 5 we show two interesting applications of the composition result to problems about non-copying Macro Tree Transducers. We will also present an implementation of our composition construction in the Haskell$^+$ system by Andreas Maletti. In Chapter 6 we draw some conclusions. The correctness proof for our main construction was moved to the Appendix.

[5] Another possibility is to extend results from [Küh98] by first decomposing tree transducers, thus *introducing* new intermediate results, but creating new opportunities for compositions, which altogether can still lead to an optimization.

Chapter 2

Preliminaries

We denote by $\mathbf{N}$ the set of natural numbers including 0, and for $n \in \mathbf{N}$, by $[n]$ the set $\{1, \ldots, n\}$. For a finite, non-empty set S of natural numbers, we denote by $max(S)$ the maximum of all its elements.
We will use several sets of lowercase variables. As a convention, we denote for $k \in \mathbf{N}$, by X_k the finite set $\{x_1, \ldots, x_k\}$ of variables, analogous for Y and Z.
For a set S, we denote by S^* the set of finite sequences of elements of S, where ϵ will denote the empty sequence. $\mathcal{P}(S)$ will denote the power set of a given set S. $|S|$ will denote the number of elements of a finite set S.
For simplicity, we define substitution as substitution over strings (elements of S^* for some finite set S). For a string v and two lists of strings $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ (with $n \in \mathbf{N}$), such that no two strings u_i and u_j $(i \neq j)$ overlap in v, we denote by $v[u_i \leftarrow v_i, 1 \leq i \leq n]$ the string obtained from v by replacing all occurrences of every u_i in v by v_i. We will also use the alternative notation $v[u_1, \ldots, u_n \leftarrow v_1, \ldots, v_n]$.

A *ranked alphabet* is a pair $(\Sigma, rank_\Sigma)$, where Σ is a finite set of symbols and $rank_\Sigma$ assigns to each of these symbols a natural number, its *rank*. By convention, we will assume that ranked alphabets are disjoint from $\mathbf{N}$. For every $k \in \mathbf{N}$, we define $\Sigma^{(k)} = \{\sigma \in \Sigma \mid rank_\Sigma(\sigma) = k\}$. The rank k of a symbol σ will also be denoted by writing $\sigma^{(k)}$. For a ranked alphabet Σ, we denote the set of all its ranks as $rank(\Sigma) = \{k \in \mathbf{N} \mid \exists \sigma \in \Sigma : rank_\Sigma(\sigma) = k\}$. A *ranked alphabet of states* is a ranked alphabet Q with $Q^{(0)} = \emptyset$. A *monadic* ranked alphabet is a ranked alphabet, where all symbols have rank zero or one.
For a ranked alphabet Σ and a set A disjoint from Σ, we define the set $T_\Sigma(A)$ of *trees over* Σ *indexed by* A as the smallest set T such that (i) $A \subseteq T$ and (ii) if $\sigma \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T$, then also $\sigma(t_1, \ldots, t_k) \in T$. For nullary symbols, we simply write α instead of $\alpha()$. By T_Σ we denote the set $T_\Sigma(\emptyset)$.

For a set S and a binary relation $\lhd \subseteq S \times S$, we denote by $\lhd^+$ the transitive and by $\lhd^*$ the reflexive, transitive closure of $\lhd$, respectively.
Let Σ be a ranked alphabet. A *rewrite rule* over Σ is a rule of the form $lhs \rightarrow rhs$ with $lhs, rhs \in T_\Sigma(V)$ for some set of variables V, such that lhs does not contain two occurrences of the same variable and every variable occurring in rhs is also contained in lhs. A set R of such rules induces a *term rewriting system* (see for example [Hue80]), denoted (Σ, R)[1], with a binary *reduction relation* $\Rightarrow_R \subseteq T_\Sigma \times T_\Sigma$, such that $s \Rightarrow_R t$, iff R contains a rule $lhs \rightarrow rhs$, there is a tree $c \in T_\Sigma(\{v\})$ (with $v \notin V$), which contains v exactly once and there exist trees $l, r, t_1, \ldots, t_n \in T_\Sigma$ and variables $v_1, \ldots, v_n \in V$, such that:

[1]Usually Σ, as well as V, will be clear from the context and we will only mention R.

$$\begin{aligned}
s &= c \quad [v/l] \\
l &= lhs\,[v_i \leftarrow t_i, 1 \le i \le n] \\
r &= rhs[v_i \leftarrow t_i, 1 \le i \le n] \\
t &= c \quad [v/r]
\end{aligned}$$

A reduction relation $\Rightarrow_R$ is called *confluent*, if for every $s, s_1, s_2 \in T_\Sigma$ with $s \Rightarrow_R^* s_1$ and $s \Rightarrow_R^* s_2$, there exists a so-called confluence element $s' \in T_\Sigma$ with $s_1 \Rightarrow_R^* s'$ and $s_2 \Rightarrow_R^* s'$. A reduction relation $\Rightarrow_R$ is called *terminating*, if there is no infinite chain $s_1 \Rightarrow_R s_2 \Rightarrow_R s_3 \Rightarrow_R \ldots$. If $s \Rightarrow_R^* t$ and there is no t' with $t \Rightarrow_R t'$, then t is called a *normal form* of s with respect to $\Rightarrow_R$. If $\Rightarrow_R$ is confluent and terminating, then every tree $s \in T_\Sigma$ has a unique normal form, denoted as $nf(\Rightarrow_R, s)$.

Let S be a finite set. A set operator $\Phi : \mathcal{P}(S) \to \mathcal{P}(S)$ is called *monotonic*, if for every $s_1 \subseteq s_2 \subseteq S$ holds: $\Phi(s_1) \subseteq \Phi(s_2)$. Define $\Phi^0 = \emptyset$ and $\Phi^{n+1} = \Phi(\Phi^n)$, for every $n \in \mathbf{N}$. If Φ is monotonic, then we have $\Phi^0 \subseteq \Phi^1 \subseteq \Phi^2 \subseteq \ldots$, and since S is finite, this infinite chain cannot be strictly increasing. Thus, there exists the *least fixpoint* $\Phi^* = \bigcup_{n \in \mathbf{N}} \Phi^n$, which is reached after finitely many iterations of Φ.

Let I be a finite set and for every $i \in I$, let be given some set $S(i)$. We write the *indexed product* $\prod_{i \in I} S(i)$ for the type of dependently typed functions $f : I \to \bigcup_{i \in I} S(i)$, such that for every $i \in I$, we have $f(i) \in S(i)$. Such a function $f \in \prod_{i \in I} S(i)$ will be denoted as $[i_1 \mapsto f(i_1), \ldots, i_n \mapsto f(i_n)]$ for some ordering $i_1, \ldots, i_n$ of the elements in I. Further, for a ranked alphabet of states Q, we will write $\prod_{q \in Q^{(r+1)}} S(r)$ instead of $\prod_{q \in Q} S'(q)$, where for every $q \in Q$, we have $S'(q) = S(rank_Q(q) - 1)$.

Example 2.1 For $Q = \{q^{(2)}, p^{(3)}\}$, the indexed product $\prod_{q \in Q^{(r+1)}} [r]$ contains the function $f_1 = [q \mapsto 1, p \mapsto 2]$, but not the function $f_2 = [q \mapsto 2, p \mapsto 1]$, because $2 \notin [(rank_Q(q) - 1)]$. $\square$

2.1 Functions on Trees

Let Σ, Δ and Ω be ranked alphabets. We call a total function $\tau : T_\Sigma \to T_\Delta$ a *tree transduction from T_Σ to T_Δ*. We define the composition of two tree transductions $\tau_1 : T_\Sigma \to T_\Delta$ and $\tau_2 : T_\Delta \to T_\Omega$, denoted by $\tau_1; \tau_2$, as $(\tau_1; \tau_2)(t) = \tau_2(\tau_1(t))$, for every $t \in T_\Sigma$. Further, we denote the composition of two classes $\mathcal{T}_1$ and $\mathcal{T}_2$ of tree transductions by $\mathcal{T}_1; \mathcal{T}_2 = \{\tau_1; \tau_2 \mid \tau_1 \in \mathcal{T}_1, \tau_2 \in \mathcal{T}_2\}$.

Let A be a set disjoint from Σ. We will need the *set of occurrences* in a tree over Σ indexed by A, given by the function $occ : T_\Sigma(A) \to \mathcal{P}(\mathbf{N}^*)$, which is defined by structural recursion as follows: (i) if $t \in \Sigma^{(0)} \cup A$, then $occ(t) = \{\epsilon\}$ and (ii) if $t = \sigma(t_1, \ldots, t_k)$ with $k \in \mathbf{N}$, $0 < k$, $t_1, \ldots, t_k \in T_\Sigma(A)$, then $occ(t) = \{\epsilon\} \cup \{i\nu \mid 1 \le i \le k,\ \nu \in occ(t_i)\}$.

We will also need the *label at an occurrence in a tree*, given by the mapping $lab : \{(t, o) \mid t \in T_\Sigma(A),\ o \in occ(t)\} \to \Sigma \cup A$, defined by: (i) if $t \in \Sigma^{(0)} \cup A$, then $lab(t, \epsilon) = t$ and (ii) if $t = \sigma(t_1, \ldots, t_k)$ with $k \in \mathbf{N}$, $0 < k$, $t_1, \ldots, t_k \in T_\Sigma(A)$, then $lab(t, \epsilon) = \sigma$ and $lab(t, i\nu) = lab(t_i, \nu)$, for $i \in [k]$ and $\nu \in occ(t_i)$. The *subtree at an occurrence in a tree* is given by the function $sub : \{(t, o) \mid t \in T_\Sigma(A),\ o \in occ(t)\} \to T_\Sigma(A)$, defined by: (i) $sub(t, \epsilon) = t$, for $t \in T_\Sigma(A)$, and (ii) if $t = \sigma(t_1, \ldots, t_k)$ with $k \in \mathbf{N}$, $0 < k$, $t_1, \ldots, t_k \in T_\Sigma(A)$, then $sub(t, i\nu) = sub(t_i, \nu)$, for $i \in [k]$ and $\nu \in occ(t_i)$.

For some symbol $s \in \Sigma \cup A$, we obtain the *number of occurrences of s in a tree* by the function $\#_s : T_\Sigma(A) \to \mathbf{N}$, defined as follows: (i) if $t \in \Sigma^{(0)} \cup A$, then $\#_s(t) =$ (if $t = s$ then 1 else 0) and (ii) if $t = \sigma(t_1, \ldots, t_k)$ with $k \in \mathbf{N}$, $0 < k$, $t_1, \ldots, t_k \in T_\Sigma(A)$, then $\#_s(t) =$ (if $\sigma = s$ then 1 else 0) $+ \sum_{i\in[k]} \#_s(t_i)$.

We define the *height of a tree* by the function $height : T_\Sigma(A) \to \mathbf{N}$ as follows: (i) if $t \in \Sigma^{(0)} \cup A$, then $height(t) = 0$ and (ii) if $t = \sigma(t_1, \ldots, t_k)$ with $k \in \mathbf{N}$, $0 < k$, $t_1, \ldots, t_k \in T_\Sigma(A)$, then $height(t) = 1 + max(\{height(t_i) \mid i \in [k]\})$. Finally, we have the notion of *size of a tree*, defined by the mapping $size : T_\Sigma(A) \to \mathbf{N}$ with: (i) if $t \in \Sigma^{(0)} \cup A$, then $size(t) = 1$ and (ii) if $t = \sigma(t_1, \ldots, t_k)$ with $k \in \mathbf{N}$, $0 < k$, $t_1, \ldots, t_k \in T_\Sigma(A)$, then $size(t) = 1 + \sum_{i\in[k]} size(t_i)$.

2.2 Macro Tree Transducers

We will define Macro Tree Transducers of a slightly more general form than in [FV98], in that we allow for a more general initial expression than just the call of some state with fixed environment[2]. First, we describe the possible shapes of right-hand sides of rules.

Definition 2.2 Let Q be a ranked alphabet of states, let Δ be a ranked alphabet, $k \in \mathbf{N}$ and V a set of variables. Then the set $RHS(Q, \Delta, k, V)$ of *right-hand sides over Q and Δ, with k recursion variables and set of context variables V*, is defined as the smallest set $RHS \subseteq T_{Q\cup\Delta}(X_k \cup V)$ satisfying the following conditions:

- **(1)** $V \subseteq RHS$
- **(2)** for every $\delta \in \Delta^{(a)}$ with $a \in \mathbf{N}$, and $\phi_1, \ldots, \phi_a \in RHS$ holds: $\delta(\phi_1, \ldots, \phi_a) \in RHS$
- **(3)** for every $q \in Q^{(a+1)}$ with $a \in \mathbf{N}$, $x_i \in X_k$ and $\phi_1, \ldots, \phi_a \in RHS$ holds: $q(x_i, \phi_1, \ldots, \phi_a) \in RHS$

□

Now, we can define the components of a Macro Tree Transducer.

Definition 2.3 A *Macro Tree Transducer* M is a tuple $(Q, \Sigma, \Delta, e, R)$ with:

- **(1)** a ranked alphabet Q of states, where $Q^{(0)} = \emptyset$
- **(2)** a ranked alphabet Σ of input symbols
- **(3)** a ranked alphabet Δ of output symbols, where $Q \cap (\Sigma \cup \Delta) = \emptyset$
- **(4)** an initial expression $e \in RHS(Q, \Delta, 1, \emptyset)$
- **(5)** a finite set R of rules of the form $q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r) \to rhs_{q,\sigma}$ with $k, r \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$, $q \in Q^{(r+1)}$ and $rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$, such that there is exactly one rule for every combination of q and σ

□

Of course, the actual variable names used in left- and right-hand sides of Macro Tree Transducer rules are not fixed to come from X_k and Y_r for some $k, r \in \mathbf{N}$; consistent renaming is allowed, and we will also write examples using different variable names.

[2]A similar generalization is used in [Küh00], the alternative would have been to introduce an explicit root-symbol as in [Küh98].

Example 2.4 Let $Q_{rev} = \{rev^{(2)}\}$ and $\Sigma_{mon} = \{A^{(1)}, B^{(1)}, E^{(0)}\}$. Then we define the Macro Tree Transducer $M_{rev} = (Q_{rev}, \Sigma_{mon}, \Sigma_{mon}, e_{rev}, R_{rev})$ with set of rules R_{rev}:

$$\begin{array}{lll} \text{(i)} : & rev(A(x), z) & \rightarrow rev(x, A(z)) \\ \text{(ii)} : & rev(B(x), z) & \rightarrow rev(x, B(z)) \\ \text{(iii)} : & rev(E, z) & \rightarrow z \end{array}$$

and $e_{rev} = rev(x_1, E)$. □

The semantics of a Macro Tree Transducer is a function from trees over the input ranked alphabet to trees over the output ranked alphabet. It can be given either by substituting the input tree for x_1 in the initial expression e of a Macro Tree Transducer, and then calculating the normal form of this expression with respect to the reduction relation induced by the set R of rules, or by the following definition.

Definition 2.5 We define the *tree transduction induced by a Macro Tree Transducer* $M = (Q, \Sigma, \Delta, e, R)$ as the total function $\tau(M) : T_\Sigma \rightarrow T_\Delta$, which assigns to every tree $t \in T_\Sigma$ the value of $\tau_{M,1,0,\emptyset}(e, t)$. For a given set A disjoint from Δ, the following set of functions is defined by simultaneous recursion:

$$\begin{gathered} \{\tau_{M,k,r,A} : RHS(Q, \Delta, k, Y_r) \underbrace{\times \cdots \times T_\Sigma}_{k-times} \underbrace{\times \cdots \times T_\Delta(A)}_{r-times} \rightarrow T_\Delta(A) \\ \mid k \in \mathbf{N}, (r+1) \in rank(Q) \cup \{1\}\} \\ \cup \\ \{\tau_{M,q,A} : T_\Sigma \underbrace{\times \cdots \times T_\Delta(A)}_{r-times} \rightarrow T_\Delta(A) \mid r \in \mathbf{N}, q \in Q^{(r+1)}\} \end{gathered}$$

For every $k \in \mathbf{N}$, $(r+1) \in rank(Q) \cup \{1\}$, the function $\tau_{M,k,r,A}$ is defined as:

- $\tau_{M,k,r,A}(y_h, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r) = \theta_h$,
  for $y_h \in Y_r$, $t_1, \ldots, t_k \in T_\Sigma$ and $\theta_1, \ldots, \theta_r \in T_\Delta(A)$

- $\tau_{M,k,r,A}(\delta(\phi_1, \ldots, \phi_a), t_1, \ldots, t_k, \theta_1, \ldots, \theta_r) =$
  $\delta(\tau_{M,k,r,A}(\phi_1, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r),$
  $\ldots,$
  $\tau_{M,k,r,A}(\phi_a, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r)),$
  for $a \in \mathbf{N}$, $\delta \in \Delta^{(a)}$, $\phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$, $t_1, \ldots, t_k \in T_\Sigma$ and $\theta_1, \ldots, \theta_r \in T_\Delta(A)$

- $\tau_{M,k,r,A}(q(x_i, \phi_1, \ldots, \phi_a), t_1, \ldots, t_k, \theta_1, \ldots, \theta_r) =$
  $\tau_{M,q,A}(t_i, \tau_{M,k,r,A}(\phi_1, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r),$
  $\ldots,$
  $\tau_{M,k,r,A}(\phi_a, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r)),$
  for $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $x_i \in X_k$, $\phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$, $t_1, \ldots, t_k \in T_\Sigma$ and $\theta_1, \ldots, \theta_r \in T_\Delta(A)$

Additionally, for every $r, k \in \mathbf{N}$, $q \in Q^{(r+1)}$, $\sigma \in \Sigma^{(k)}$, $t_1, \ldots, t_k \in T_\Sigma$ and $\theta_1, \ldots, \theta_r \in T_\Delta(A)$ holds:

$\tau_{M,q,A}(\sigma(t_1, \ldots, t_k), \theta_1, \ldots, \theta_r) = \tau_{M,k,r,A}(rhs_{q,\sigma}, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r)$,
if R contains the rule $q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r) \rightarrow rhs_{q,\sigma}$ with $rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$.

We denote the class of all *tree transductions computable by Macro Tree Transducers* as *MAC*. □

Although the Macro Tree Transducers presented here have a more general initial expression than those used in [FV98], our tree transduction class MAC coincides with the corresponding one in [FV98], because Macro Tree Transducers in the two representations can be transformed to each other. Note, that the same need not be true for restricted classes of tree transductions, because our representation is more flexible.

We now give an example for the calculation of a tree transduction using our recursively defined semantics.

Example 2.6 We apply the Macro Tree Transducer from the previous example to the input tree $A(B(E))$, that is, we calculate $\tau(M_{rev})(A(B(E)))$:

$$\begin{aligned}
&\tau(M_{rev})(A(B(E))) = \\
&\tau_{M_{rev},1,0,\emptyset}(e_{rev}, A(B(E))) = \\
&\tau_{M_{rev},rev,\emptyset}(A(B(E)), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E)))) = \\
&\tau_{M_{rev},1,1,\emptyset}(rev(x_1, A(y_1)), B(E), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E)))) = \\
&\tau_{M_{rev},rev,\emptyset}(B(E), \tau_{M_{rev},1,1,\emptyset}(A(y_1), B(E), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E))))) = \\
&\tau_{M_{rev},1,1,\emptyset}(rev(x_1, B(y_1)), E, \tau_{M_{rev},1,1,\emptyset}(A(y_1), B(E), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E))))) = \\
&\tau_{M_{rev},rev,\emptyset}(E, \tau_{M_{rev},1,1,\emptyset}(B(y_1), E, \tau_{M_{rev},1,1,\emptyset}(A(y_1), B(E), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E)))))) = \\
&\tau_{M_{rev},0,1,\emptyset}(y_1, \tau_{M_{rev},1,1,\emptyset}(B(y_1), E, \tau_{M_{rev},1,1,\emptyset}(A(y_1), B(E), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E)))))) = \\
&\tau_{M_{rev},1,1,\emptyset}(B(y_1), E, \tau_{M_{rev},1,1,\emptyset}(A(y_1), B(E), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E))))) = \\
&B(\tau_{M_{rev},1,1,\emptyset}(y_1, E, \tau_{M_{rev},1,1,\emptyset}(A(y_1), B(E), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E)))))) = \\
&B(\tau_{M_{rev},1,1,\emptyset}(A(y_1), B(E), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E))))) = \\
&B(A(\tau_{M_{rev},1,1,\emptyset}(y_1, B(E), \tau_{M_{rev},1,0,\emptyset}(E, A(B(E)))))) = \\
&B(A(\tau_{M_{rev},1,0,\emptyset}(E, A(B(E))))) = \\
&B(A(E))
\end{aligned}$$

□

Note, that the functions, which are given in Definition 2.5, are parameterized over a set A disjoint from Δ. For now, this set is just the empty set, but later we will need the generalization for technical reasons.

We already noted that it is also possible to define the semantics of a Macro Tree Transducer by the rewriting system of its rules. The following proposition, which is a consequence of results in Chapter 4 of [FV98], states the equivalence between the recursively defined semantics and reduction semantics.

Proposition 2.7 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer.

1. $\Rightarrow_R$ is confluent and terminating
2. for every $t \in T_\Sigma$ holds: $\tau(M)(t) = nf(\Rightarrow_R, e[x_1 \leftarrow t])$

□

Example 2.8 Using the reduction semantics, we compute $\tau(M_{rev})(A(B(E)))$ as the normal form of $e_{rev}[x_1 \leftarrow A(B(E))]$ with respect to $\Rightarrow_{R_{rev}}$:

$$rev(A(B(E)), E) \Rightarrow_{R_{rev}} rev(B(E), A(E)) \Rightarrow_{R_{rev}} rev(E, B(A(E))) \Rightarrow_{R_{rev}} B(A(E))$$ □

We introduce two important syntactic properties of Macro Tree Transducers (cf. [Küh98]) and the corresponding computability classes.

Definition 2.9 A Macro Tree Transducer is called *non-copying*, if in every right-hand side, there is no more than one occurrence of every context variable. We denote the class of all *tree transductions computable by non-copying Macro Tree Transducers* as MAC_{nc}. □

Definition 2.10 A Macro Tree Transducer is called *weakly single-use*, if for every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$, $i \in [k]$ and $q \in Q$, a call $q(x_i, \cdots)$ occurs in a right-hand side of at most one rule at σ and there only one such call.We denote the class of all *tree transductions computable by weakly single-use Macro Tree Transducers* as MAC_{wsu}. □

We also identify Macro Tree Transducers, which have both of the above properties.

Definition 2.11 A Macro Tree Transducer is called *single-use*, if it is both non-copying and weakly single-use. We denote the class of all *tree transductions computable by single-use Macro Tree Transducers* as MAC_{su}. □

Another important restriction of Macro Tree Transducers is that of having no context parameters at all, which gives us Top-Down Tree Transducers (cf. [Rou70]).

Definition 2.12 A Macro Tree Transducer is called a *Top-Down Tree Transducer*, if all its states have rank one, that is, there are no context parameters present. We denote the class of all *tree transductions computable by Top-Down Tree Transducers* as TOP. □

Example 2.13

- M_{rev} is both non-copying and weakly single-use, thus it is also single-use.

- $M_{top} = (\{p^{(1)}, q^{(1)}\}, \Sigma_{mon}, \Sigma_{mon}, p(x_1), R_{top})$ with set of rules R_{top}:

$$\begin{array}{lll} \text{(i)} & : p(A(x_1)) & \rightarrow A(p(x_1)) \\ \text{(ii)} & : p(B(x_1)) & \rightarrow q(x_1) \\ \text{(iii)} & : p(E) & \rightarrow E \\ \text{(iv)} & : q(A(x_1)) & \rightarrow p(x_1) \\ \text{(v)} & : q(B(x_1)) & \rightarrow B(q(x_1)) \\ \text{(vi)} & : q(E) & \rightarrow E \end{array}$$

  is a Top-Down Tree Transducer and thus trivially non-copying, but it is not weakly single-use, because a call $p(x_1)$ occurs twice in right-hand sides of rules at A.

- The Macro Tree Transducer $M' = (\{q^{(2)}\}, \Sigma_{mon}, \{\alpha^{(2)}, \beta^{(2)}, \epsilon^{(0)}\}, q(x_1, \epsilon), R')$ with set of rules R':

$$\begin{array}{lll} \text{(i)} & : q(A(x_1), y_1) & \rightarrow q(x_1, \alpha(y_1, y_1)) \\ \text{(ii)} & : q(B(x_1), y_1) & \rightarrow q(x_1, \beta(y_1, y_1)) \\ \text{(iii)} & : q(E, y_1) & \rightarrow y_1 \end{array}$$

  is not non-copying, because the right-hand sides are not linear in context variables.

□

The following proposition will later be used to identify Macro Tree Transducers with above properties.

Proposition 2.14 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer. M is non-copying, iff for every $\sigma \in \Sigma$, $r \in \mathbf{N}$, $q \in Q^{(r+1)}$ and $h \in [r]$ holds: $\#_{y_h}(rhs_{q,\sigma}) \leq 1$. If Q is a singleton set $\{q\}$, then M is weakly single-use, iff for every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $i \in [k]$ holds: $\#_{x_i}(rhs_{q,\sigma}) \leq 1$. □

2.3 Proving a Lemma

In this section, we introduce the principle of proof by simultaneous induction and illustrate its application by proving a substitution lemma, which we will need later.

Definition 2.15 Let Σ be a ranked alphabet. Given two statements **(I)** and **(II)**, where **(I)** has a free variable $\mathbf{t} \in \mathbf{T_\Sigma}$ and **(II)** has free variables $\mathbf{k} \in \mathbf{N}$ and $\mathbf{t_1}, \ldots, \mathbf{t_k} \in \mathbf{T_\Sigma}$, we say that **(I)** and **(II)** are *proven by simultaneous induction*, if we can prove the following two induction steps:

(I) ⇐ (II) : for every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T_\Sigma$, such that **(II)** holds, we can show that **(I)** holds for $t = \sigma(t_1, \ldots, t_k)$

(II) ⇐ (I) : for every $k \in \mathbf{N}$ and $t_1, \ldots, t_k \in T_\Sigma$, such that **(I)** holds for each of the $t_1, \ldots, t_k$, we can show that **(II)** holds

By proving these two induction steps, we have proven that **(I)** holds for every $t \in T_\Sigma$ and that **(II)** holds for every $k \in \mathbf{N}$ and $t_1, \ldots, t_k \in T_\Sigma$. □

As an example, we prove the following lemma by using simultaneous induction.

Lemma 2.16 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer. For every set A disjoint from Δ, $t \in T_\Sigma$, $r \in \mathbf{N}$, $q \in Q^{(r+1)}$ and $\theta_1, \ldots, \theta_r \in T_\Delta(A)$, we have:
$\tau_{M,q,A}(t, \theta_1, \ldots, \theta_r) = (\tau_{M,q,[r]}(t, 1, \ldots, r))[l \leftarrow \theta_l, 1 \leq l \leq r]$. □

Proof: We prove the following two statements by simultaneous induction.

(I) For $\mathbf{t} \in \mathbf{T_\Sigma}$: for every set A with $A \cap \Delta = \emptyset$, $r \in \mathbf{N}$, $q \in Q^{(r+1)}$ and $\theta_1, \ldots, \theta_r \in T_\Delta(A)$ holds:
$\tau_{M,q,A}(t, \theta_1, \ldots, \theta_r) = (\tau_{M,q,[r]}(t, 1, \ldots, r))[l \leftarrow \theta_l, 1 \leq l \leq r]$

(II) For $\mathbf{k} \in \mathbf{N}, \mathbf{t_1}, \ldots, \mathbf{t_k} \in \mathbf{T_\Sigma}$: for every $A \cap \Delta = \emptyset$, $r + 1 \in rank(Q)$, $\theta_1, \ldots, \theta_r \in T_\Delta(A)$ and $\phi \in RHS(Q, \Delta, k, Y_r)$ holds:
$\tau_{M,k,r,A}(\phi, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r) = (\tau_{M,k,r,[r]}(\phi, t_1, \ldots, t_k, 1, \ldots, r))[l \leftarrow \theta_l, 1 \leq l \leq r]$

(I) ⇐ (II) : Let $t = \sigma(t_1, \ldots, t_k)$ with $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T_\Sigma$, then:

$\tau_{M,q,A}(\sigma(t_1, \ldots, t_k), \theta_1, \ldots, \theta_r)$
$\quad = \;$ (by definition of $\tau_{M,q,A}$)
$\tau_{M,k,r,A}(rhs_{q,\sigma}, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r)$
$\quad = \;$ (by induction hypothesis **(II)** for $t_1, \ldots, t_k$ and $\phi = rhs_{q,\sigma}$)
$(\tau_{M,k,r,[r]}(rhs_{q,\sigma}, t_1, \ldots, t_k, 1, \ldots, r))[l \leftarrow \theta_l, 1 \leq l \leq r]$
$\quad = \;$ (by definition of $\tau_{M,q,[r]}$)
$(\tau_{M,q,[r]}(\sigma(t_1, \ldots, t_k), 1, \ldots, r))[l \leftarrow \theta_l, 1 \leq l \leq r]$

(II) $\Leftarrow$ (I) : We prove the claim by structural induction on $\phi \in RHS(Q,\Delta,k,Y_r)$.

$\phi = y_h \in Y_r$:

$$
\begin{aligned}
&\tau_{M,k,r,A}(y_h, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r)\\
&\quad = \text{(by definition of } \tau_{M,k,r,A})\\
&\theta_h\\
&\quad = \text{(by substitution)}\\
&h[l \leftarrow \theta_l, 1 \le l \le r]\\
&\quad = \text{(by definition of } \tau_{M,k,r,[r]})\\
&(\tau_{M,k,r,[r]}(y_h, t_1, \ldots, t_k, 1, \ldots, r))[l \leftarrow \theta_l, 1 \le l \le r]
\end{aligned}
$$

$\phi = \delta(\phi_1, \ldots, \phi_a), a \in \mathbf{N}, \delta \in \Delta^{(a)}, \phi_1, \ldots, \phi_a \in RHS(Q,\Delta,k,Y_r)$:

$$
\begin{aligned}
&\tau_{M,k,r,A}(\delta(\phi_1, \ldots, \phi_a), t_1, \ldots, t_k, \theta_1, \ldots, \theta_r)\\
&\quad = \text{(by definition of } \tau_{M,k,r,A})\\
&\delta(\tau_{M,k,r,A}(\phi_1, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r),\\
&\quad \ldots,\\
&\quad \tau_{M,k,r,A}(\phi_a, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r))\\
&\quad = \text{(by induction hypothesis for } \phi_1, \ldots, \phi_a)\\
&\delta((\tau_{M,k,r,[r]}(\phi_1, t_1, \ldots, t_k, 1, \ldots, r))[l \leftarrow \theta_l, 1 \le l \le r],\\
&\quad \ldots,\\
&\quad (\tau_{M,k,r,[r]}(\phi_a, t_1, \ldots, t_k, 1, \ldots, r))[l \leftarrow \theta_l, 1 \le l \le r])\\
&\quad = \text{(by substitution)}\\
&\delta(\tau_{M,k,r,[r]}(\phi_1, t_1, \ldots, t_k, 1, \ldots, r),\\
&\quad \ldots,\\
&\quad \tau_{M,k,r,[r]}(\phi_a, t_1, \ldots, t_k, 1, \ldots, r))[l \leftarrow \theta_l, 1 \le l \le r]\\
&\quad = \text{(by definition of } \tau_{M,k,r,[r]})\\
&(\tau_{M,k,r,[r]}(\delta(\phi_1, \ldots, \phi_a), t_1, \ldots, t_k, 1, \ldots, r))[l \leftarrow \theta_l, 1 \le l \le r]
\end{aligned}
$$

$\phi = q(x_i, \phi_1, \ldots, \phi_a), a \in \mathbf{N}, q \in Q^{(a+1)}, x_i \in X_k, \phi_1, \ldots, \phi_a \in RHS(Q,\Delta,k,Y_r)$:

$$
\begin{aligned}
&\tau_{M,k,r,A}(q(x_i, \phi_1, \ldots, \phi_a), t_1, \ldots, t_k, \theta_1, \ldots, \theta_r)\\
&\quad = \text{(by definition of } \tau_{M,k,r,A})\\
&\tau_{M,q,A}(t_i, \tau_{M,k,r,A}(\phi_1, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r),\\
&\qquad\qquad \ldots,\\
&\qquad\qquad \tau_{M,k,r,A}(\phi_a, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r))\\
&\quad = \text{(by induction hypothesis } \mathbf{(I)} \text{ for } t_i)\\
&(\tau_{M,q,[a]}(t_i, 1, \ldots, a))[h \leftarrow \tau_{M,k,r,A}(\phi_h, t_1, \ldots, t_k, \theta_1, \ldots, \theta_r), 1 \le h \le a]\\
&\quad = \text{(by induction hypothesis for } \phi_h, h \in [a])\\
&(\tau_{M,q,[a]}(t_i, 1, \ldots, a))[h \leftarrow (\tau_{M,k,r,[r]}(\phi_h, t_1, \ldots, t_k, 1, \ldots, r))[l \leftarrow \theta_l, 1 \le l \le r],\\
&\qquad\qquad\qquad 1 \le h \le a]\\
&\quad = \text{(by } [r] \cap \Delta = \emptyset \text{ and properties of substitution)}\\
&((\tau_{M,q,[a]}(t_i, 1, \ldots, a))[h \leftarrow \tau_{M,k,r,[r]}(\phi_h, t_1, \ldots, t_k, 1, \ldots, r), 1 \le h \le a])\\
&\quad [l \leftarrow \theta_l, 1 \le l \le r]\\
&\quad = \text{(by induction hypothesis } \mathbf{(I)} \text{ for } t_i)\\
&(\tau_{M,q,[r]}(t_i, \tau_{M,k,r,[r]}(\phi_1, t_1, \ldots, t_k, 1, \ldots, r),\\
&\qquad\qquad \ldots,\\
&\qquad\qquad \tau_{M,k,r,[r]}(\phi_a, t_1, \ldots, t_k, 1, \ldots, r)))[l \leftarrow \theta_l, 1 \le l \le r]\\
&\quad = \text{(by definition of } \tau_{M,k,r,[r]})\\
&(\tau_{M,k,r,[r]}(q(x_i, \phi_1, \ldots, \phi_a), t_1, \ldots, t_k, 1, \ldots, r))[l \leftarrow \theta_l, 1 \le l \le r]
\end{aligned}
$$

□

Chapter 3

The Construction

The heart of this thesis is the development of an algorithm, which, given two Macro Tree Transducers—the first of which is non-copying and the second one weakly single-use—, constructs a single new Macro Tree Transducer that implements the composition of the two original ones. We can give an a priori justification[1] of the feasibility of this aim by semantic reasoning with the help of other tree transduction classes, which are not covered in this thesis [Fül81, Küh97].

Theorem 3.1 $MAC_{nc}; MAC_{wsu} = MAC$ □

Proof:

$$\begin{array}{lll} & MAC_{nc}; MAC_{wsu} & \\ \subseteq & TOP; ATT_{su}; ATT & \text{(an analogue of Lemma 5.3 and Theorem 7.1 of [Küh98])} \\ \subseteq & TOP; ATT & \text{(Lemma 6.4 of [Küh97])} \\ \subseteq & TOP; MAC & \text{(cf. [Fra82])} \\ \subseteq & MAC & \text{(Corollary 4.10 of [EV85])} \\ \subseteq & TOP; YIELD & \text{(Theorem 4.8 of [EV85])} \\ \subseteq & MAC_{nc}; MAC_{wsu} & (TOP \subseteq MAC_{nc} \text{ trivially; Lemma 6.10 of [Küh97], cf. [EV85])} \end{array}$$

The inclusion $MAC_{nc}; MAC_{wsu} \subseteq MAC$ is also illustrated in Figure 3.1. □

The same kind of semantic reasoning can be applied to obtain a result on well-presented Macro Tree Transducers (cf. [CF82, FV98])—namely, $MAC_{nc}; MAC_{wp} = MAC$[2]—, as well as on classes of Macro Attributed Tree Transducers, which are not subject of the present thesis (see [KV94]).

3.1 The Idea

Before giving our formal construction, we will try to provide an intuitive explanation of the underlying ideas:

The development of our construction first goes along the same lines as the classical construction for $MAC; TOP \subseteq MAC$ from [EV85]. Given two Macro Tree Transducers

[1] Although we arrived at it only *after* establishing our construction; the original aim was to give a direct construction for the weaker result $MAC_{su}; MAC_{wsu} \subseteq MAC$.

[2] $MAC_{nc}; MAC_{wp} \subseteq TOP; ATT_{su}; ATT \subseteq MAC \subseteq TOP; YIELD \subseteq MAC_{nc}; MAC_{wp}$, cf. Theorem 5.10 of [CF82] and the construction from Example 4.5 of [EV85]

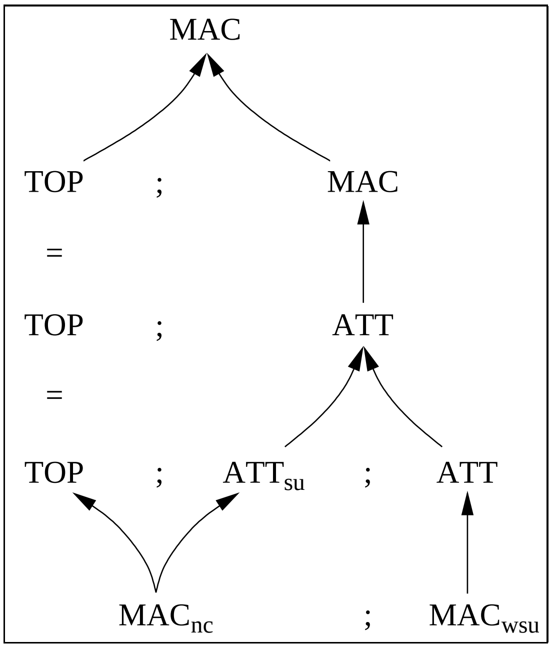

Figure 3.1: Argument for $MAC_{nc}; MAC_{wsu} \subseteq MAC$

M_1 (from T_Σ to T_Δ, with rules R_1) and M_2 (from T_Δ to T_Ω, with rules R_2) with respective sets of states $Q = \{q_1, \ldots, q_n\}$ and $P = \{p_1, \ldots, p_m\}$, we want to create a Macro Tree Transducer $M_{1;2}$ (from T_Σ to T_Ω) with paired states $\{(q,p) \mid q \in Q, p \in P\}$. The basic idea behind this pairing construction is that a nested call of the form $p(q(t, y_1, \ldots, y_r), z_1, \ldots, z_s)$—with concrete trees for the $y_1, \ldots, y_r$ and $z_1, \ldots, z_s$—, which might arise from running M_1 and M_2 independently (having started with a composite expression, like $aux(pre(t, \epsilon), \epsilon)$ in the introductory example), should correspond to a construct $(q,p)(t, ?, \ldots, ?)$ during the computation of $M_{1;2}$. Now, how to fill the question marks, respectively decide what rank the new states should have? Clearly, we will keep the context parameters $z_1, \ldots, z_s$ of the outer call to $p \in P^{(s+1)}$. The situation is more involved for the context parameters of $q \in Q^{(r+1)}$, because they are not just copied into the final output. Rather, they will occur in the intermediate result after reducing $q(t, y_1, \ldots, y_r)$ and during the computation of $p(\bullet, z_1, \ldots, z_s)$ on that intermediate result—substituted for the bullet—, they are likely to be met by some state of the second transducer. Following the construction from [EV85], we will thus, for every context parameter y of q, hold available its m translations for all the states $p_1, \ldots, p_m$ of M_2. Accordingly, $p(q(t, y_1, \ldots, y_r), z_1, \ldots, z_s)$ will be replaced by $(q,p)(t, y_{1,p_1}, \ldots, y_{r,p_m}, z_1, \ldots, z_s)$, assuming that in the $y_{1,p_1}, \ldots, y_{r,p_m}$-positions, we are provided with all necessary translations of context parameters.

Now, that we settled the question of the ranks of our new states, we have to think about how to build the rules for them. Namely, given some input symbol $\sigma \in \Sigma^{(k)}$, what will be the right-hand side for $(q,p)(\sigma(x_1, \ldots, x_k), y_{1,p_1}, \ldots, y_{r,p_m}, z_1, \ldots, z_s)$? According to our idea, this should be corresponding to $p(q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r), z_1, \ldots, z_s)$, so we will take the right-hand side of the first transducer's rule $q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r) \to rhs_{q,\sigma}$ and translate it with p by reducing $p(rhs_{q,\sigma}, z_1, \ldots, z_s)$. Of course, in general $rhs_{q,\sigma}$ will *not* only consist of output symbols of the first transducer—which are readily processed by states of M_2—, so it is not obvious how to do this reduction. In fact, while reducing $p(rhs_{q,\sigma}, z_1, \ldots, z_s)$, we have to take care of three different cases:

1. some state $p' \in P$ meets an output symbol δ of M_1 (which therefore is an input symbol for the second transducer)
2. some state $p' \in P$ meets a context parameter y_h of q
3. some state $p' \in P$ meets a call to some state $q' \in Q$ of the first transducer

In the first case, we will just apply the rule given by M_2 for p' at δ, and it will be exactly this application, which leads to the elimination of intermediate data structures. In the second case, we remember that the call to (q, p) was provided with translations of the context parameters with all possible states of P, so we will just select the right one and output $y_{h,p'}$. In the third case, we will have met some situation $p'(q'(x_i, \phi_1, \ldots, \phi_a), \eta_1, \ldots, \eta_{s'})$ with $p' \in P^{(s'+1)}$, $q' \in Q^{(a+1)}$, furthermore $\phi_1, \ldots, \phi_a$ being part of the first transducer's right-hand side $rhs_{q,\sigma}$ and $\eta_1, \ldots, \eta_{s'}$ being parts of right-hand side expressions for $M_{1;2}$, which have been built up during the reduction of $rhs_{q,\sigma}$ with p. By our underlying idea of pairing states, we know that we should replace this with something like $(q', p')(x_i, ?, \ldots, ?, \eta_1, \ldots, \eta_{s'})$, where for the question marks, we have to fill in the translations of the context parameters of q'—namely $\phi_1, \ldots, \phi_a$—with all possible states of M_2 (such that the state (q', p') can later use these provided translations). So we get $(q', p')(x_i, p_1(\phi_1, ?, \ldots, ?), \ldots, p_m(\phi_a, ?, \ldots, ?), \eta_1, \ldots, \eta_{s'})$. In the construction for the result $MAC; TOP \subseteq MAC$, we would be fine now, because then all the states of the second transducer would have rank 1, so there would be no question marks left in our expression and we could go on by translating the $\phi_1, \ldots, \phi_a$ with states $p_1, \ldots, p_m$, just in the same way as we started processing the $rhs_{q,\sigma}$ with p.

Instead, as we are dealing with general Macro Tree Transducers, we have to fill the question mark slots in the following expression:

(*) $$(q', p')(x_i, p_1(\phi_1, ?, \ldots, ?), \ldots, p_m(\phi_a, ?, \ldots, ?), \eta_1, \ldots, \eta_{s'}).$$

Now, what are we supposed to provide in the places of the question marks? It should be the context parameter values, with which the states $p_1, \ldots, p_m$ are expected to "arrive" at occurrences of $\phi_1, \ldots, \phi_a$ during computation of the $p'(q'(x_i, \phi_1, \ldots, \phi_a), \eta_1, \ldots, \eta_{s'})$ met above (and thus will clearly depend on the tree substituted for x_i). However, it is far from obvious, whether we can always provide this information. So, we better first answer the following question in general (see Figure 3.2):

> "Given two states q' and p' of Macro Tree Transducers M_1 and M_2 as above, and some input tree t for M_1, can we determine, for any state p'' of M_2, and any context parameter y_h from $y_1, \ldots, y_a$, which will be the values in context parameter positions in every occurrence of a call to p'' on y_h during the reduction of $p'(q'(t, y_1, \ldots, y_a), \eta_1, \ldots, \eta_{s'})$?"

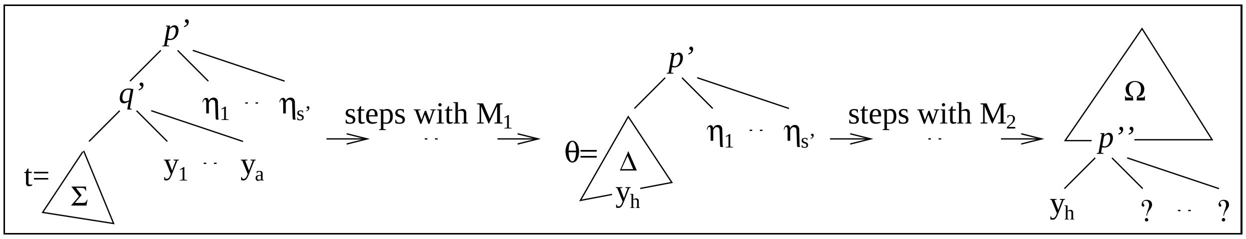

Figure 3.2: Can we uniquely determine the values in question mark positions?

The following two examples show that in general the answer to this question is *no*!

- consider a rule $q'(\alpha, y_1) \to \delta(y_1, y_1)$ of M_1 and rule $p'(\delta(x_1, x_2)) \to \omega(p''(x_1, \alpha), p''(x_2, \beta))$ of M_2, giving for $t = \alpha$ the reduction:
  $p'(q'(\alpha, y_1)) \Rightarrow_{R_1} p'(\delta(y_1, y_1)) \Rightarrow_{R_2} \omega(p''(y_1, \alpha), p''(y_1, \beta))$

- consider a rule $q'(\alpha, y_1) \to \gamma(y_1)$ of M_1 and rule $p'(\gamma(x_1)) \to p''(x_1, p''(x_1, \alpha))$ of M_2, giving for $t = \alpha$ the reduction:
  $p'(q'(\alpha, y_1)) \Rightarrow_{R_1} p'(\gamma(y_1)) \Rightarrow_{R_2} p''(y_1, p''(y_1, \alpha))$

In both cases, the reduction leads to occurrences of calls of p'' on y_1 with different values in context parameter positions. However, we claim that the above question can be answered positively, if M_1 is non-copying and M_2 is weakly single-use[3]—or, trivially, if one of them is a Top-Down Tree Transducer—, by reasoning as follows:

If M_1 is non-copying, then for every input tree t of M_1, the normal form θ of $q'(t, y_1, \ldots, y_a)$ will contain at most one occurrence of every $y_1, \ldots, y_a$ (can be shown by induction over the reduction sequence). If, moreover, M_2 is weakly single-use, then we can find out the values of context parameters in any occurrence of a call to some state p'' on, e.g., y_h during the reduction of $p'(\theta, \eta_1, \ldots, \eta_{s'})$, by "walking upwards in the tree θ" from the unique—if existing—occurrence of y_h in θ in the following manner:

1. If y_h is at the root of θ, then either $p'' = p'$ and the sought context parameter values are $\eta_1, \ldots, \eta_{s'}$, or $p'' \neq p'$ and we can take some arbitrary dummy context parameters, because there will be no call to p'' on y_h during reduction of $p'(\theta, \eta_1, \ldots, \eta_{s'})$.

2. If y_h is not at the root of θ, then it must occur below some output symbol of M_1, i.e., there is some context C and symbol δ (which is an input symbol for M_2), such that $\theta = C[\delta(\cdots, y_h, \cdots)]$, say y_h occurs as ith child of the occurrence of δ. Then there is at most one possible way, how y_h can be reached by a call to state p'', namely by the unique (since the second transducer is weakly single-use) occurrence of a $p''(x_i, \cdots)$-call in some right-hand side of a rule of M_2 at δ. If we assume that this unique occurrence is in the right-hand side of some rule
   $$p'''(\delta(\cdots), z_1, \ldots, z_{s'''}) \to \cdots p''(x_i, \psi_1, \ldots, \psi_{s''}) \cdots,$$
   we could from $\psi_1, \ldots, \psi_{s''}$ determine our sought context parameters of calls to p'' on reaching the occurrence of y_h, if we knew the context parameters of calls to p''' on reaching the occurrence of δ above y_h (because the $\psi_1, \ldots, \psi_{s''}$ might contain occurrences of the $z_1, \ldots, z_{s'''}$). This occurrence can now again be either the root of θ or below some output symbol of M_1 in θ, so we either are in case 1 or again "walk upwards" in θ using this second case, until finally we reach the root.

This informal and incomplete explanation (the $\psi_1, \ldots, \psi_{s''}$ might also contain calls to other children of the δ-occurrence, or calls to the same child with another state, and we did not say, how to handle these) seems to suggest that the above question can be answered with *yes*. However, this does not immediately help our construction of $M_{1;2}$, because a Macro Tree Transducer has to work locally and cannot follow our proposed procedure of "reduce $q'(t, y_1, \ldots, y_a)$ to its normal form θ, check where the context variables $y_1, \ldots, y_a$ are in θ, and walk your way up from there". Instead, we claim that the procedure suggested above *can* be implemented with a Macro Tree Transducer, and exactly this will be the "difficult part" of our construction.

[3]Note, that this excludes our two counterexamples.

For every two states q and p of M_1 and M_2, respectively, and every pair of context parameter positions h and l of q and p, respectively, we will introduce a new state (h_q, l_p)[4], which on a given tree t shall compute, what will be the lth context parameter in a call to p on the hth context variable of q, resulting from some reduction of $p'(q(t, \ldots, y_h, \ldots), \cdots)$, where p' is unknown, but unique (due to our reasoning of "walking upwards in the intermediate tree, using the weakly single-use property").

Coming back to **(*)**, we can now fill the question mark positions, using these new states:

(**)
$$\begin{aligned}(q',p')(x_i, & p_1(\phi_1, (1_{q'}, 1_{p_1})(x_i, ?, \ldots, ?), \ldots, (1_{q'}, s_{1p_1})(x_i, ?, \ldots, ?)),\\ & \ldots,\\ & p_m(\phi_a, (a_{q'}, 1_{p_m})(x_i, ?, \ldots, ?), \ldots, (a_{q'}, s_{mp_m})(x_i, ?, \ldots, ?)),\\ & \eta_1, \ldots, \eta_{s'}),\end{aligned}$$

assuming that the states $p_1, \ldots, p_m$ of M_2 have $s_1, \ldots, s_m$ context parameter positions, respectively.

This, however, produced new question marks, and before we can fill them, we have to think about what context parameters such a new state (h_q, l_p) requires, respectively, how its rules can be constructed. For every input symbol $\sigma \in \Sigma^{(k)}$ of M_1, we have to construct a rule $(h_q, l_p)(\sigma(x_1, \ldots, x_k), \cdots) \to ?$, aimed at computing the lth context parameter in a call to p on y_h, resulting from a reduction of $p'(q(\sigma(x_1, \ldots, x_k), \ldots, y_h, \ldots), \cdots)$ for unknown p'. Clearly, we should use our knowledge of the first transducer's rule $q(\sigma(x_1, \ldots, x_k), \ldots, y_h, \ldots) \to rhs_{q,\sigma}$. If $rhs_{q,\sigma}$ does not contain y_h, then there cannot result a call to any p on y_h from the reduction, so we are safe to choose some dummy right-hand side for (h_q, l_p) at σ. For the more general case that $rhs_{q,\sigma}$ *does* contain y_h, we will consider only an example here, and hope it is somewhat illuminating our idea.

Consider the case that $rhs_{q,\sigma} = \delta(y_2, y_1)$—for some input symbol $\delta \in \Delta^{(2)}$ of the second transducer—and we are interested in the first context parameter in a call to $p^{(2)}$ on y_2 (that is, we want to construct the right-hand side for $(2_q, 1_p)$ at σ). We know that the only way to have a call to p on the unique occurrence of y_2 is via a call to p on x_1 in some right-hand side of M_2 at δ. Due to the weakly single-use property of M_2, there can only be one such call in the collection of all right-hand sides at δ. Assume, it occurs in a rule $p'(\delta(x_1, x_2), z_1) \to \cdots p(x_1, \omega(p''(x_2), z_1)) \cdots$. Now, we know, what the first context parameter of p on y_2 will be—namely $\omega(p''(x_2), z_1)$—, but there is still x_2 present, for which we have to substitute the second child of the δ-occurrence in $rhs_{q,\sigma}$. Thus, we get $\omega(p''(y_1), z_1)$. This depends both on the p''-translation of y_1 and on the first context parameter of p' at the root of $rhs_{q,\sigma}$. Since, in general, we do not know, which states p' and p'' will be concerned, we will need to have as context parameters of $(2_q, 1_p)$: (i) all translations of context parameters of $q \in Q^{(r+1)}$ with some state of M_2, and (ii) all context parameters for every possible state of M_2 at the occurrence of $q(\sigma(x_1, \ldots, x_k), \ldots, y_2, \ldots)$ (respectively, at the root of $rhs_{q,\sigma}$). Thus, the above rule will indeed have left-hand side $(h_q, l_p)(\sigma(x_1, \ldots, x_k), y_{1,p_1}, \ldots, y_{r,p_m}, z_{p_1,1}, \ldots, z_{p_m,s_m})$.

Note, that—as argued above—for every input tree t, there will be at most one state p' of M_2, such that reducing $p'(q(t, \ldots, y_h, \ldots), \cdots)$ leads to a call of p on y_h. The result of reducing $(h_q, l_p)(t, y_{1,p_1}, \ldots, y_{r,p_m}, z_{p_1,1}, \ldots, z_{p_m,s_m})$, which shall compute the lth context parameter of this call of p on y_h, will then of the $z_{p_1,1}, \ldots, z_{p_m,s_m}$ only contain those parameters, which are associated with this unique p'. However, since for different input trees t also this p' might differ, it is unavoidable for the state (h_q, l_p) to have all the context parameters $z_{p_1,1}, \ldots, z_{p_m,s_m}$, though only a subset of them will be relevant

[4] This shall not denote indexed integers, rather it is used as a compacter notation for the pair of pairs $((h, q), (l, p))$.

in every concrete computation. Intuitively, in order to get the lth context parameter of a call $p(y_h, \cdots)$ resulting from reduction of $p'(q(t, \ldots, y_h, \ldots), \cdots)$, we have to compute $(h_q, l_p)(t, y_{1,p_1}, \ldots, y_{r,p_m}, z_{p_1,1}, \ldots, z_{p_m,s_m})$ and replace the $z_{p',1}, \ldots, z_{p',s'}$ (no other variables from $z_{p_1,1}, \ldots, z_{p_m,s_m}$ will occur in the result) by the context parameters of p' at the occurrence of $q(t, \ldots, y_h, \ldots)$.

Continuing our example of finding the first context parameter in a call to $p^{(2)}$ on y_2, we note that we did not really examine a very general form of right-hand side. What, if it were more complicated, say $rhs_{q,\sigma} = q'(x_b, \delta(y_2, y_1))$ instead? Again, we have to start from the unique occurrence of y_2 in this right-hand side, which is the first subtree of the δ-occurrence. By the reasoning given above—assuming the same rule for p' at δ—we get $\omega(p''(y_1), z_1)$, which contains the p''-translation of y_1 and the first context parameter of p' on reaching the occurrence of δ in $rhs_{q,\sigma}$. Since we said that $(2_q, 1_p)$ will have as context parameters the translations of $y_1, \ldots, y_r$ with all states of M_2, we can replace $p''(y_1)$ with $y_{1,p''}$. For z_1, however, we need to do further calculation, because it now does not stand for the context parameter of p' at the root of $rhs_{q,\sigma}$, but rather for the one on reaching the δ-occurrence in $rhs_{q,\sigma}$. But this δ occurs in a context parameter position of q', so we know that we *can* compute our needed value for z_1 by calling $(1_{q'}, 1_{p'})$ on x_b (thus computing the first context parameter of p' on reaching the first context parameter position of q' during a reduction on $q'(x_b, \cdots)$ with some state of M_2). Thus, we get $\omega(y_{1,p''}, (1_{q'}, 1_{p'})(x_b, ?, \ldots, ?))$. How to fill the context parameter positions of this call to $(1_{q'}, 1_{p'})$, is another problem, which we do not want to discuss at this point (rather, we postpone this discussion until having formally introduced our construction).

Coming back to **(**)**, we know a bit better, how to fill the remaining question marks in:

$$
\begin{aligned}
(q', p')(x_i, p_1(&\phi_1, (1_{q'}, 1_{p_1})(x_i, ?, \ldots, ?), \ldots, (1_{q'}, s_{1p_1})(x_i, ?, \ldots, ?)),\\
&\ldots,\\
p_m(&\phi_a, (a_{q'}, 1_{p_m})(x_i, ?, \ldots, ?), \ldots, (a_{q'}, s_{mp_m})(x_i, ?, \ldots, ?)),\\
\eta_1, &\ldots, \eta_{s'}).
\end{aligned}
$$

Recall, that we introduced the calls to $(1_{q'}, 1_{p_1}), \ldots, (a_{q'}, s_{mp_m})$ on x_i, in order to compute the context parameter values, with which the states $p_1, \ldots, p_m$ are expected to "arrive" at occurrences of $\phi_1, \ldots, \phi_a$ during the reduction of $p'(q'(x_i, \phi_1, \ldots, \phi_a), \eta_1, \ldots, \eta_{s'})$ for instantiated x_i. Further, recall that—by generalization of the above discussion on the example rule for state $(2_q, 1_p)$—every state $(h_{q'}, l_p) \in \{(1_{q'}, 1_{p_1}), \ldots, (a_{q'}, s_{mp_m})\}$ expects in the $z_{p_1,1}, \ldots, z_{p_m,s_m}$-positions of its call on x_i, the context parameters of all possible states of M_2 at the occurrence of $q'(x_i, \phi_1, \ldots, \phi_a)$. However, of the translations $p_1(\phi_1, \cdots), \ldots, p_m(\phi_a, \cdots)$ only those will be needed during the above computation of state (q', p') on x_i, which actually occur during the reduction of $p'(q'(x_i, \phi_1, \ldots, \phi_a), \eta_1, \ldots, \eta_{s'})$ for instantiated x_i. For such a p and ϕ_h, we know (by the discussion below the explanation of left-hand sides for (h_q, l_p)-states) that a call of state $(h_{q'}, l_p)$ on the concrete tree substituted for x_i, will of its $z_{p_1,1}, \ldots, z_{p_m,s_m}$-positions only depend on those, which are associated with p'. We also gave the intuition that for these $z_{p',1}, \ldots, z_{p',s'}$-positions, we have to provide the context parameters of p' at the occurrence of $q'(x_i, \ldots, \phi_h, \ldots)$. Thus, we will fill the $z_{p',1}, \ldots, z_{p',s'}$-positions with $\eta_1, \ldots, \eta_{s'}$, and the others with a dummy nullary output symbol *nil* (these positions will *not* influence the computation, as we will illustrate later in Example 3.3).

This leaves us with the following expression:

$$(***)\quad \begin{aligned}&(q',p')(x_i, p_1(\phi_1, (1_{q'}, 1_{p_1})(x_i, ?, \ldots, ?, nil, \ldots, nil, \eta_1, \ldots, \eta_{s'}, nil, \ldots, nil),\\ &\qquad\qquad \ldots,\\ &\qquad\qquad (1_{q'}, s_{1p_1})(x_i, ?, \ldots, ?, nil, \ldots, nil, \eta_1, \ldots, \eta_{s'}, nil, \ldots, nil)),\\ &\quad \ldots,\\ &\quad p_m(\phi_a, (a_{q'}, 1_{p_m})(x_i, ?, \ldots, ?, nil, \ldots, nil, \eta_1, \ldots, \eta_{s'}, nil, \ldots, nil),\\ &\qquad\qquad \ldots,\\ &\qquad\qquad (a_{q'}, s_{mp_m})(x_i, ?, \ldots, ?, nil, \ldots, nil, \eta_1, \ldots, \eta_{s'}, nil, \ldots, nil)),\\ &\quad \eta_1, \ldots, \eta_{s'}),\end{aligned}$$

still having to fill the $y_{1,p_1}, \ldots, y_{a,p_m}$ positions of calls to $(1_{q'}, 1_{p_1}), \ldots, (a_{q'}, s_{mp_m})$. Here, we have to provide $p_1, \ldots, p_m$-translations of the $\phi_1, \ldots, \phi_a$, which might be needed during the computations of these states. This looks suspiciously like a circular construction, because we would use $p_1(\phi_1, \cdots)$ within $p_1(\phi_1, \cdots)$ and the like. However, we claim that we can "cut" these cycles by keeping track of the nesting (similar to the unfolding in the construction for $ATT \subseteq MAC$, cf. [Fra82, FV98]), because the positions, where they occur, can for no possible input tree influence the computation. The intuitive reason for this is that otherwise we would meet a situation corresponding to the second "counterexample" on page 17, like $p''(y_1, p''(y_1, \alpha))$, which we claimed cannot occur, because all occurrences of $p''(y_1, \cdots)$ must have identical context parameters, if M_1 is non-copying and M_2 is weakly single-use.

3.2 Formal Development

We give the construction as a set of mutually recursive function definitions. For some state p of M_2, the $trans_p$- and $thru$-functions perform the task of translating the right-hand sides of M_1 with state p, much in the same way as the $\tau_{M_2,p,\emptyset}$- and $\tau_{M_2,k,r,\emptyset}$-functions from Definition 2.5 would process T_Δ-trees, when the tree transduction induced by M_2 is called on some input. Interesting is especially the third case for the $trans_p$-function, where the pairing of states takes place.

For some state q of M_1 and input tree variable x_i, the $unfold_{q(x_i)}$-function initiates the preparation of translated context parameters of q, necessary for the paired state as discussed in the last section. Its first argument is the position of the context parameter to be translated, its second argument is the state of M_2, with which to translate. The third argument is a set, which keeps track of nested parameter-state-combinations, in order to perform the "cutting" of cycles during the unfolding process, as mentioned at the end of last section.

For some state p of M_2 and a right-hand side ϕ of M_1, the $ctx_{p,\phi}$-function performs the "walking upwards" in the right-hand side ϕ, necessary for giving rules to the (h_q, l_p)-states. The intuitive reading is as follows: $ctx_{p,\phi}(o, l)$ gives us a right-hand side expression for $M_{1;2}$, which computes the lth context parameter in a call to state p, reaching the occurrence o in the first transducer's right-hand side expression ϕ (having started at the root of ϕ with some unknown state), as illustrated in Figure 3.3. We will give an explanation of the defining cases of $ctx_{p,\phi}$-functions immediately below the construction.

The introduced functions are parameterized by additional subscripts k, r and V, which are only necessary to determine correct types for them.

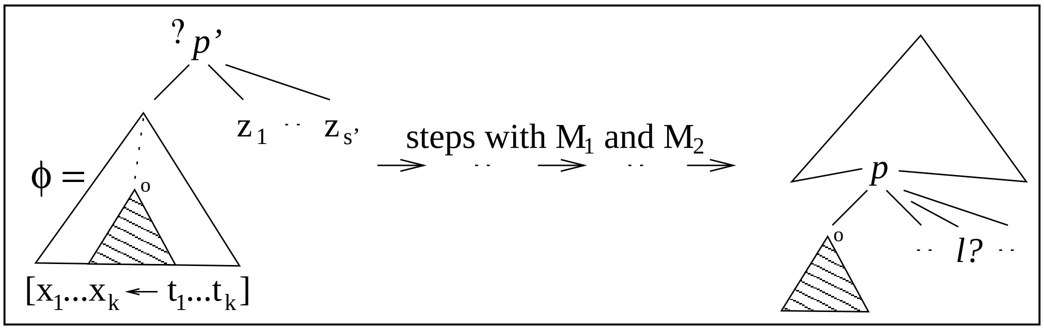

Figure 3.3: $ctx_{p,\phi}(o,l)$ gives an expression for $M_{1;2}$, computing the "l?"-position.

Construction 3.2 Let $M_1 = (Q, \Sigma, \Delta, e_1, R_1)$ and $M_2 = (P, \Delta, \Omega, e_2, R_2)$ be Macro Tree Transducers, such that M_1 is non-copying and M_2 is weakly single-use. We construct the Macro Tree Transducer $M_{1;2} = (S, \Sigma, \Omega, e_{1;2}, R_{1;2})$ as follows:

Let $m = |P|$ and fix some ordering of states in P, such that $P = \{p_1, \ldots, p_m\}$. For $1 \le i \le m$, let $s_i \in \mathbf{N}$ be such that $p_i \in P^{(s_i+1)}$. Additionally, let $nil \in \Omega^{(0)}$ be some dummy output symbol and $Z_P = \{z_{p_1,1}, \ldots, z_{p_1,s_1}, \ldots, z_{p_m,1}, \ldots, z_{p_m,s_m}\}$. Then the components of $M_{1;2}$ are obtained as follows:

- $S = \{(q,p)^{(r*m+s+1)} \mid r,s \in \mathbf{N}, q \in Q^{(r+1)}, p \in P^{(s+1)}\}$
  $\cup \{(h_q, l_p)^{(r*m+|Z_P|+1)} \mid r,s \in \mathbf{N}, q \in Q^{(r+1)}, h \in [r], p \in P^{(s+1)}, l \in [s]\}$
- $e_{1;2} = thru_{1,0,1,0,\emptyset}(e_2, e_1)$
- $R_{1;2}$ contains:
  - for $r,s,k \in \mathbf{N}$, $q \in Q^{(r+1)}$, $p \in P^{(s+1)}$ and $\sigma \in \Sigma^{(k)}$, the rule:
    $(q,p)(\sigma(x_1, \ldots, x_k), y_{1,p_1}, \ldots, y_{r,p_m}, z_1, \ldots, z_s) \to trans_{p,k,r,Z_s}(rhs_{q,\sigma}, z_1, \ldots, z_s)$,
    if R_1 contains the rule $q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r) \to rhs_{q,\sigma}$
  - for $r,s,k \in \mathbf{N}$, $q \in Q^{(r+1)}$, $h \in [r]$, $p \in P^{(s+1)}$, $l \in [s]$ and $\sigma \in \Sigma^{(k)}$, the rule:
    $(h_q, l_p)(\sigma(x_1, \ldots, x_k), y_{1,p_1}, \ldots, y_{r,p_m}, z_{p_1,1}, \ldots, z_{p_m,s_m}) \to \psi$,
    if R_1 contains the rule $q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r) \to rhs_{q,\sigma}$,
    where $\psi = nil$, if $rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$ does not contain the context variable y_h; otherwise $\psi = ctx_{p,rhs_{q,\sigma},k,r}(o,l)$ with $o \in occ(rhs_{q,\sigma})$ as the unique occurrence of y_h in $rhs_{q,\sigma}$ (M_1 is non-copying, thus $\#_{y_h}(rhs_{q,\sigma}) \le 1$!)

For every $k, r \in \mathbf{N}$ and finite set of variables V, the following set of functions is defined by simultaneous recursion:

$$\{trans_{p,k,r,V} \mid p \in P\} \cup \{unfold_{q(x_i),k,r,V} \mid q \in Q, x_i \in X_k\}$$
$$\cup \{thru_{a,s,k,r,V} \mid a \in rank(\Delta), s+1 \in rank(P) \text{ or } a = 1, s = 0\}$$

With $RHS_{k,r,V} = RHS(S, \Omega, k, \{y_{1,p_1}, \ldots, y_{r,p_m}\} \cup V)$, these have the following types:

for $s \in \mathbf{N}$, $p \in P^{(s+1)}$:
$trans_{p,k,r,V} : RHS(Q, \Delta, k, Y_r) \underbrace{\times \cdots \times RHS_{k,r,V}}_{s-times} \to RHS_{k,r,V}$

for $a \in rank(\Delta)$, $s+1 \in rank(P)$ or $a = 1$, $s = 0$:
$thru_{a,s,k,r,V} : RHS(P, \Omega, a, Y_s) \underbrace{\times \cdots \times RHS(Q, \Delta, k, Y_r)}_{a-times} \underbrace{\times \cdots \times RHS_{k,r,V}}_{s-times} \to RHS_{k,r,V}$

for $a \in \mathbf{N}$, $q \in Q^{(a+1)}$ and $x_i \in X_k$:
$unfold_{q(x_i),k,r,V} : [a] \times P \times \mathcal{P}([a] \times P) \underbrace{\times \cdots \times RHS(Q,\Delta,k,Y_r)}_{a-times} \underbrace{\times \cdots \times RHS_{k,r,V}}_{|Z_P|-times} \to RHS_{k,r,V}$

For every $s \in \mathbf{N}$ and $p \in P^{(s+1)}$, the function $trans_{p,k,r,V}$ is defined (with $\eta_1, \ldots, \eta_s \in RHS_{k,r,V}$) as follows:

- $trans_{p,k,r,V}(y_h, \eta_1, \ldots, \eta_s) = y_{h,p}$,
  for $y_h \in Y_r$
- $trans_{p,k,r,V}(\delta(\phi_1, \ldots, \phi_a), \eta_1, \ldots, \eta_s) = thru_{a,s,k,r,V}(rhs_{p,\delta}, \phi_1, \ldots, \phi_a, \eta_1, \ldots, \eta_s)$,
  for $a \in \mathbf{N}$, $\delta \in \Delta^{(a)}$, $\phi_1, \ldots, \phi_a \in RHS(Q,\Delta,k,Y_r)$ and rule
  $p(\delta(x_1, \ldots, x_a), y_1, \ldots, y_s) \to rhs_{p,\delta}$ in R_2, with $rhs_{p,\delta} \in RHS(P,\Omega,a,Y_s)$
- $trans_{p,k,r,V}(q(x_i, \phi_1, \ldots, \phi_a), \eta_1, \ldots, \eta_s) =$
  $(q,p)(x_i, unfold_{q(x_i),k,r,V}(1, p_1, \emptyset, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}),$
  $\ldots,$
  $unfold_{q(x_i),k,r,V}(a, p_m, \emptyset, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}),$
  $\eta_1, \ldots, \eta_s)$
  for $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $x_i \in X_k$ and $\phi_1, \ldots, \phi_a \in RHS(Q,\Delta,k,Y_r)$, where for every $s' \in \mathbf{N}$, $p' \in P^{(s'+1)}$ and $l' \in [s']$, we have $\eta_{p',l'} =$ (if $p' = p$ then $\eta_{l'}$ else nil).

For every $a \in rank(\Delta)$, $s+1 \in rank(P)$ or $a = 1$, $s = 0$, the function $thru_{a,s,k,r,V}$ is defined (with $\phi_1, \ldots, \phi_a \in RHS(Q,\Delta,k,Y_r)$ and $\eta_1, \ldots, \eta_s \in RHS_{k,r,V}$) as follows:

- $thru_{a,s,k,r,V}(y_l, \phi_1, \ldots, \phi_a, \eta_1, \ldots, \eta_s) = \eta_l$,
  for $y_l \in Y_s$
- $thru_{a,s,k,r,V}(\omega(\psi_1, \ldots, \psi_n), \phi_1, \ldots, \phi_a, \eta_1, \ldots, \eta_s) =$
  $\omega(thru_{a,s,k,r,V}(\psi_1, \phi_1, \ldots, \phi_a, \eta_1, \ldots, \eta_s),$
  $\ldots,$
  $thru_{a,s,k,r,V}(\psi_n, \phi_1, \ldots, \phi_a, \eta_1, \ldots, \eta_s)),$
  for $n \in \mathbf{N}$, $\omega \in \Omega^{(n)}$ and $\psi_1, \ldots, \psi_n \in RHS(P,\Omega,a,Y_s)$
- $thru_{a,s,k,r,V}(p(x_i, \psi_1, \ldots, \psi_n), \phi_1, \ldots, \phi_a, \eta_1, \ldots, \eta_s) =$
  $trans_{p,k,r,V}(\phi_i, thru_{a,s,k,r,V}(\psi_1, \phi_1, \ldots, \phi_a, \eta_1, \ldots, \eta_s),$
  $\ldots,$
  $thru_{a,s,k,r,V}(\psi_n, \phi_1, \ldots, \phi_a, \eta_1, \ldots, \eta_s)),$
  for $n \in \mathbf{N}$, $p \in P^{(n+1)}$, $x_i \in X_a$ and $\psi_1, \ldots, \psi_n \in RHS(P,\Omega,a,Y_s)$

For every $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $x_i \in X_k$, the function $unfold_{q(x_i),k,r,V}$ is defined (with $h \in [a]$, $p \in P$, $\mathcal{S} \subseteq [a] \times P$, $\phi_1, \ldots, \phi_a \in RHS(Q,\Delta,k,Y_r)$ and $\eta_{p_1,1}, \ldots, \eta_{p_m,s_m} \in RHS_{k,r,V}$) as follows:

- $unfold_{q(x_i),k,r,V}(h, p, \mathcal{S}, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}) = nil$,
  if $(h,p) \in \mathcal{S}$

- $unfold_{q(x_i),k,r,V}(h, p, \mathcal{S}, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}) =$
$$trans_{p,k,r,V}(\phi_h, (h_q, 1_p)(x_i, unfold_{q(x_i),k,r,V}(1, p_1, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}), \ldots, unfold_{q(x_i),k,r,V}(a, p_m, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}), \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}), \ldots, (h_q, s_p)(x_i, unfold_{q(x_i),k,r,V}(1, p_1, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}), \ldots, unfold_{q(x_i),k,r,V}(a, p_m, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}), \eta_{p_1,1}, \ldots, \eta_{p_m,s_m})),$$
if $(h,p) \notin \mathcal{S}$, $s \in \mathbf{N}$ and $p \in P^{(s+1)}$

Using the functions defined above, we additionally define for every $k, r \in \mathbf{N}$ and $\phi \in RHS(Q, \Delta, k, Y_r)$, the following set of functions by simultaneous recursion:
$$\{ctx_{p,\phi,k,r} : \{o \in occ(\phi) \mid lab(\phi, o) \notin X_k\} \times [s] \to RHS_{k,r,Z_P} \mid s \in \mathbf{N}, p \in P^{(s+1)}\}$$
For every $s \in \mathbf{N}$, $p \in P^{(s+1)}$, the function $ctx_{p,\phi,k,r}$ is defined (with $l \in [s]$) as follows:

- $ctx_{p,\phi,k,r}(\epsilon, l) = z_{p,l}$
- $ctx_{p,\phi,k,r}(oi, l) =$
  - in case $lab(\phi, o) = \delta$ with $o \in occ(\phi)$, $a \in \mathbf{N}$, $\delta \in \Delta^{(a)}$ and $i \in [a]$:
    If, with $s' \in \mathbf{N}$, $p' \in P^{(s'+1)}$ and $\psi_1, \ldots, \psi_s \in RHS(P, \Omega, a, Y_{s'})$, the only occurrence of a $p(x_i)$-call in the δ-rules of the weakly single-use M_2 looks as follows:
    $$p'(\delta(x_1, \ldots, x_a), y_1, \ldots, y_{s'}) \to \cdots p(x_i, \psi_1, \ldots, \psi_s) \cdots,$$
    then the right-hand side of the equation is:
    $thru_{a,s',k,r,Z_P}(\psi_l, sub(\phi, o1), \ldots, sub(\phi, oa), ctx_{p',\phi,k,r}(o, 1), \ldots, ctx_{p',\phi,k,r}(o, s'))$.
    If no such call exists in the δ-rules of M_2, then the right-hand side of the equation is nil.
  - in case $lab(\phi, o) = q$ with $o \in occ(\phi)$, $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $lab(\phi, o1) = x_b \in X_k$ and $2 \leq i \leq a+1$:
    $$((i-1)_q, l_p)(x_b, unfold_{q(x_b),k,r,Z_P}(1, p_1, \{(i-1, p)\}, sub(\phi, o2), \ldots, sub(\phi, o(a+1)), ctx_{p_1,\phi,k,r}(o, 1), \ldots, ctx_{p_m,\phi,k,r}(o, s_m)), \ldots, unfold_{q(x_b),k,r,Z_P}(a, p_m, \{(i-1, p)\}, sub(\phi, o2), \ldots, sub(\phi, o(a+1)), ctx_{p_1,\phi,k,r}(o, 1), \ldots, ctx_{p_m,\phi,k,r}(o, s_m)), ctx_{p_1,\phi,k,r}(o, 1), \ldots, ctx_{p_m,\phi,k,r}(o, s_m)).$$

Then we claim: $\tau(M_1); \tau(M_2) = \tau(M_{1;2})$. □

We will show the correctness of the previous claim in the Appendix.

As promised, we now want to explain the crucial part of the construction, namely the working of the $ctx_{p,\phi}$-functions. Recall, that—for appropriate k and r—$ctx_{p,\phi,k,r}(o, l)$ shall give a right-hand side expression for $M_{1;2}$, which computes the lth context parameter in a call to state p, reaching the occurrence o in the first transducer's right-hand side expression ϕ. In constructing this right-hand side expression, we are allowed to use the context parameters $z_{p_1,1},\ldots,z_{p_m,s_m}$ of all possible states of M_2 at the root of ϕ, as well as $y_{1,p_1},\ldots,y_{r,p_m}$, denoting translations of the context parameters $y_1,\ldots,y_r$—possibly occurring in ϕ—with states of M_2. For every $k,r,s\in\mathbf{N}$, $\phi\in RHS(Q,\Delta,k,Y_r)$ and $p\in P^{(s+1)}$, the function $ctx_{p,\phi,k,r}$ is defined by case analysis on its first argument, an occurrence in the tree ϕ (the second argument is some $l\in[s]$). This occurrence can be:

- the root of ϕ,
  then the result is the appropriate choice from the $z_{p,1},\ldots,z_{p,s}$-parameters

- the ith child position of the occurrence of an output symbol $\delta\in\Delta^{(a)}$ in ϕ,
  then we have to look for an occurrence of a $p(x_i)$-call in the δ-rules of M_2, because this is the only way, how a call to state p can reach the ith child position of the occurrence of δ. If there is no such call, we can safely output some dummy symbol, because then we are in a position, which cannot be relevant for the computation. If however, there is such a—then necessarily unique—call in a rule $p'(\delta(x_1,\ldots,x_a),y_1,\ldots,y_{s'})\to\cdots p(x_i,\psi_1,\ldots,\psi_s)\cdots$, then our sought lth context parameter of a call to state p, on reaching the ith child position of δ, is essentially $\psi_l\in RHS(P,\Omega,a,Y_{s'})$, except that it might contain references to the children of the occurrence of δ and to context parameters of the p'-call at δ. The former are obtained as the direct subtrees of the occurrence of δ in ϕ, while the latter can be computed by using the $ctx_{p',\phi,k,r}$-function on the occurrence of δ in ϕ. Everything is plugged together by applying the appropriate *thru*-function, and we are done with this case.

- the $(i-1)$th context parameter position of the occurrence of a call to some state $q\in Q^{(a+1)}$ on some $x_b\in X_k$ in ϕ,
  then the result can be computed by calling the state $((i-1)_q,l_p)$ on x_b, because this state was *designed* to compute the lth context parameter in a call to state p, reaching the $(i-1)$th context parameter position of a call to state q. In order to do so, the state $((i-1)_q,l_p)$ has to be provided with translations of q's context parameters—at the particular occurrence of the call to q in ϕ—with states of M_2, and furthermore, with the context parameter values of the second transducer's states at this occurrence. The latter ones are computed by $ctx_{p_1,\phi,k,r},\ldots,ctx_{p_m,\phi,k,r}$-function calls on the occurrence of q in ϕ. The translations of q's context parameters with states of M_2 are obtained by calling the $unfold_{q(x_b),k,r,Z_P}$-function on the subtrees of ϕ in context parameter positions of the occurrence of the call to q. The $unfold_{q(x_b),k,r,Z_P}$-function takes care of proper (and finite!) nesting of such translations. Note, that the unfolding is not started with the empty set of "cut-positions", but with $\{(i-1,p)\}$, because we know that the context parameters in a call to state p, reaching the $(i-1)$th context parameter position of a call to state q cannot depend on the p-translation of this $(i-1)$th context parameter.

Now, we give an example of applying the presented construction to two Macro Tree Transducers. In order to make explicit the positions, where the dummy symbol *nil* is used—indicating that these positions will not influence the computation—, we add *nil* to the output ranked alphabet, instead of using some existing nullary symbol, as we usually would do. The fact that these positions indeed cannot influence the computation, is then verified by arguing that for every possible input, the output tree computed by $M_{1;2}$ will contain no *nil*-symbols.

Example 3.3 Let $\Sigma'_{mon} = \{\alpha^{(1)}, \beta^{(1)}, \epsilon^{(0)}\}$ and $M_1 = (\{q_1^{(2)}\}, \Sigma_{mon}, \Sigma'_{mon}, q_1(x_1, \epsilon), R_1)$ with set of rules R_1:

$$\begin{array}{lll}
\text{(i)} & : q_1(A(x_1), y_1) & \rightarrow \alpha(q_1(x_1, y_1)) \\
\text{(ii)} & : q_1(B(x_1), y_1) & \rightarrow q_1(x_1, \beta(y_1)) \\
\text{(iii)} & : q_1(E, y_1) & \rightarrow y_1
\end{array}$$

Further, let $M_2 = (\{p_1^{(2)}, p_2^{(2)}\}, \Sigma'_{mon}, Nat, p_2(x_1, zero), R_2)$ with set of rules R_2:

$$\begin{array}{lll}
\text{(iv)} & : p_1(\alpha(x_1), y_1) & \rightarrow p_1(x_1, p_2(x_1, y_1)) \\
\text{(v)} & : p_1(\beta(x_1), y_1) & \rightarrow succ(y_1) \\
\text{(vi)} & : p_1(\epsilon, y_1) & \rightarrow y_1 \\
\text{(vii)} & : p_2(\alpha(x_1), y_1) & \rightarrow y_1 \\
\text{(viii)} & : p_2(\beta(x_1), y_1) & \rightarrow p_2(x_1, p_1(x_1, y_1)) \\
\text{(ix)} & : p_2(\epsilon, y_1) & \rightarrow y_1
\end{array}$$

Obviously, M_1 is non-copying and M_2 is weakly single-use, thus we can apply Construction 3.2 to obtain $M_{1;2} = (S, \Sigma_{mon}, Nat \cup \{nil\}, e_{1;2}, R_{1;2})$ with components as follows (where $m = 2$, $s_1 = s_2 = 1$ and $Z_P = \{z_{p_1,1}, z_{p_2,1}\}$):

- $S = \{(q_1, p_1)^{(4)}, (q_1, p_2)^{(4)}\} \cup \{(1_{q_1}, 1_{p_1})^{(5)}, (1_{q_1}, 1_{p_2})^{(5)}\}$
- $$\begin{array}{l}
e_{1;2} = thru_{1,0,1,0,\emptyset}(p_2(x_1, zero), q_1(x_1, \epsilon)) \\
\quad = \\
trans_{p_2,1,0,\emptyset}(q_1(x_1, \epsilon), thru_{1,0,1,0,\emptyset}(zero, q_1(x_1, \epsilon))) \\
\quad = \\
(q_1, p_2)(x_1, unfold_{q_1(x_1),1,0,\emptyset}(1, p_1, \emptyset, \epsilon, nil, zero), \\
\qquad\qquad unfold_{q_1(x_1),1,0,\emptyset}(1, p_2, \emptyset, \epsilon, nil, zero), \\
\qquad\qquad zero) \\
\quad = \\
(q_1, p_2)(x_1, trans_{p_1,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_1})(x_1, \\
\qquad\qquad\qquad unfold_{q_1(x_1),1,0,\emptyset}(1, p_1, \{(1, p_1)\}, \epsilon, nil, zero), \\
\qquad\qquad\qquad unfold_{q_1(x_1),1,0,\emptyset}(1, p_2, \{(1, p_1)\}, \epsilon, nil, zero), \\
\qquad\qquad\qquad nil, zero)), \\
\qquad\qquad trans_{p_2,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_2})(x_1, \\
\qquad\qquad\qquad unfold_{q_1(x_1),1,0,\emptyset}(1, p_1, \{(1, p_2)\}, \epsilon, nil, zero), \\
\qquad\qquad\qquad unfold_{q_1(x_1),1,0,\emptyset}(1, p_2, \{(1, p_2)\}, \epsilon, nil, zero), \\
\qquad\qquad\qquad nil, zero)), \\
\qquad\qquad zero) \\
\quad = \\
(q_1, p_2)(x_1, trans_{p_1,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_1})(x_1, \\
\qquad\qquad\qquad nil, \\
\qquad\qquad\qquad trans_{p_2,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_2})(x_1, \\
\qquad\qquad\qquad\quad unfold_{q_1(x_1),1,0,\emptyset}(1, p_1, \{(1, p_1), (1, p_2)\}, \epsilon, nil, zero), \\
\qquad\qquad\qquad\quad unfold_{q_1(x_1),1,0,\emptyset}(1, p_2, \{(1, p_1), (1, p_2)\}, \epsilon, nil, zero), nil, zero)), \\
\qquad\qquad\qquad nil, zero)), \\
\qquad\qquad trans_{p_2,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_2})(x_1, \\
\qquad\qquad\qquad trans_{p_1,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_1})(x_1, \\
\qquad\qquad\qquad\quad unfold_{q_1(x_1),1,0,\emptyset}(1, p_1, \{(1, p_2), (1, p_1)\}, \epsilon, nil, zero), \\
\qquad\qquad\qquad\quad unfold_{q_1(x_1),1,0,\emptyset}(1, p_2, \{(1, p_2), (1, p_1)\}, \epsilon, nil, zero), nil, zero)), \\
\qquad\qquad\qquad nil, \\
\qquad\qquad\qquad nil, zero)), \\
\qquad\qquad zero)
\end{array}$$

$$
\begin{aligned}
&= \\
&(q_1, p_2)(x_1, trans_{p_1,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_1})(x_1, \\
&\qquad\qquad nil, \\
&\qquad\qquad trans_{p_2,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_2})(x_1, nil, nil, nil, zero)), \\
&\qquad\qquad nil, zero)), \\
&\qquad trans_{p_2,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_2})(x_1, \\
&\qquad\qquad trans_{p_1,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_1})(x_1, nil, nil, nil, zero)), \\
&\qquad\qquad nil, \\
&\qquad\qquad nil, zero)), \\
&\qquad zero) \\
&= \\
&(q_1, p_2)(x_1, trans_{p_1,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_1})(x_1, \\
&\qquad\qquad nil, thru_{0,1,1,0,\emptyset}(y_1, (1_{q_1}, 1_{p_2})(x_1, nil, nil, nil, zero)), nil, zero)), \\
&\qquad trans_{p_2,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_2})(x_1, \\
&\qquad\qquad thru_{0,1,1,0,\emptyset}(y_1, (1_{q_1}, 1_{p_1})(x_1, nil, nil, nil, zero)), nil, nil, zero)), \\
&\qquad zero) \\
&= \\
&(q_1, p_2)(x_1, trans_{p_1,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_1})(x_1, nil, (1_{q_1}, 1_{p_2})(x_1, nil, nil, nil, zero), nil, zero)), \\
&\qquad trans_{p_2,1,0,\emptyset}(\epsilon, (1_{q_1}, 1_{p_2})(x_1, (1_{q_1}, 1_{p_1})(x_1, nil, nil, nil, zero), nil, nil, zero)), \\
&\qquad zero) \\
&= \\
&(q_1, p_2)(x_1, thru_{0,1,1,0,\emptyset}(y_1, (1_{q_1}, 1_{p_1})(x_1, nil, (1_{q_1}, 1_{p_2})(x_1, nil, nil, nil, zero), nil, zero)), \\
&\qquad thru_{0,1,1,0,\emptyset}(y_1, (1_{q_1}, 1_{p_2})(x_1, (1_{q_1}, 1_{p_1})(x_1, nil, nil, nil, zero), nil, nil, zero)), \\
&\qquad zero) \\
&= \\
&(q_1, p_2)(x_1, (1_{q_1}, 1_{p_1})(x_1, nil, (1_{q_1}, 1_{p_2})(x_1, nil, nil, nil, zero), nil, zero), \\
&\qquad (1_{q_1}, 1_{p_2})(x_1, (1_{q_1}, 1_{p_1})(x_1, nil, nil, nil, zero), nil, nil, zero), \\
&\qquad zero)
\end{aligned}
$$

- The rules in $R_{1;2}$ are obtained as:
  - $(q_1, p_1)(A(x_1), y_{1,p_1}, y_{1,p_2}, z_1) \to trans_{p_1,1,1,Z_1}(rhs_{q_1,A}, z_1)$, with:
    $$\begin{aligned}&trans_{p_1,1,1,Z_1}(\alpha(q_1(x_1, y_1)), z_1)\\ &\quad=\\ &thru_{1,1,1,1,Z_1}(rhs_{p_1,\alpha}, q_1(x_1, y_1), z_1)\\ &\quad=\\ &trans_{p_1,1,1,Z_1}(q_1(x_1, y_1), trans_{p_2,1,1,Z_1}(q_1(x_1, y_1), z_1))\\ &\quad=\\ &(q_1, p_1)(x_1, y_{1,p_1}, y_{1,p_2}, (q_1, p_2)(x_1, y_{1,p_1}, y_{1,p_2}, z_1))\end{aligned}$$
  - $(q_1, p_1)(B(x_1), y_{1,p_1}, y_{1,p_2}, z_1) \to trans_{p_1,1,1,Z_1}(rhs_{q_1,B}, z_1)$, with:
    $$\begin{aligned}&trans_{p_1,1,1,Z_1}(q_1(x_1, \beta(y_1)), z_1)\\ &\quad=\\ &(q_1, p_1)(x_1, succ((1_{q_1}, 1_{p_1})(x_1, nil, y_{1,p_2}, z_1, nil)), y_{1,p_2}, z_1)\end{aligned}$$
  - $(q_1, p_1)(E, y_{1,p_1}, y_{1,p_2}, z_1) \to trans_{p_1,0,1,Z_1}(rhs_{q_1,E}, z_1)$, with:
    $$\begin{aligned}&trans_{p_1,0,1,Z_1}(y_1, z_1)\\ &\quad=\\ &y_{1,p_1}\end{aligned}$$

- $(q_1,p_2)(A(x_1),y_{1,p_1},y_{1,p_2},z_1) \to trans_{p_2,1,1,Z_1}(rhs_{q_1,A},z_1)$, with:
  $$\begin{aligned}
  & trans_{p_2,1,1,Z_1}(\alpha(q_1(x_1,y_1)),z_1)\\
  & \quad = \\
  & z_1
  \end{aligned}$$
- $(q_1,p_2)(B(x_1),y_{1,p_1},y_{1,p_2},z_1) \to trans_{p_2,1,1,Z_1}(rhs_{q_1,B},z_1)$, with:
  $$\begin{aligned}
  & trans_{p_2,1,1,Z_1}(q_1(x_1,\beta(y_1)),z_1)\\
  & \quad = \\
  & (q_1,p_2)(x_1,succ((1_{q_1},1_{p_1})(x_1,nil,y_{1,p_2},nil,z_1)),y_{1,p_2},z_1)
  \end{aligned}$$
- $(q_1,p_2)(E,y_{1,p_1},y_{1,p_2},z_1) \to trans_{p_2,0,1,Z_1}(rhs_{q_1,E},z_1)$, with:
  $$\begin{aligned}
  & trans_{p_2,0,1,Z_1}(y_1,z_1)\\
  & \quad = \\
  & y_{1,p_2}
  \end{aligned}$$
- $(1_{q_1},1_{p_1})(A(x_1),y_{1,p_1},y_{1,p_2},z_{p_1,1},z_{p_2,1}) \to ctx_{p_1,rhs_{q_1,A},1,1}(12,1)$, with:
  $$\begin{aligned}
  & ctx_{p_1,\alpha(q_1(x_1,y_1)),1,1}(12,1)\\
  & \quad = \\
  & (1_{q_1},1_{p_1})(x_1, unfold_{q_1(x_1),1,1,Z_P}(1,p_1,\{(1,p_1)\},y_1,\eta_{p_1,1},\eta_{p_2,1}),\\
  & \qquad\qquad unfold_{q_1(x_1),1,1,Z_P}(1,p_2,\{(1,p_1)\},y_1,\eta_{p_1,1},\eta_{p_2,1}),\\
  & \qquad\qquad \eta_{p_1,1},\eta_{p_2,1}),\\
  & \quad \text{where } \eta_{p_1,1} = ctx_{p_1,\alpha(q_1(x_1,y_1)),1,1}(1,1)\\
  & \qquad\qquad \eta_{p_2,1} = ctx_{p_2,\alpha(q_1(x_1,y_1)),1,1}(1,1)\\
  & \quad = \text{ (because of rule } p_1(\alpha(x_1),y_1) \to p_1(x_1,p_2(x_1,y_1)) \text{ in } R_2)\\
  & (1_{q_1},1_{p_1})(x_1,nil,\\
  & \qquad\qquad trans_{p_2,1,1,Z_P}(y_1,(1_{q_1},1_{p_2})(x_1,\\
  & \qquad\qquad\quad unfold_{q_1(x_1),1,1,Z_P}(1,p_1,\{(1,p_1),(1,p_2)\},y_1,\eta_{p_1,1},\eta_{p_2,1}),\\
  & \qquad\qquad\quad unfold_{q_1(x_1),1,1,Z_P}(1,p_2,\{(1,p_1),(1,p_2)\},y_1,\eta_{p_1,1},\eta_{p_2,1}),\\
  & \qquad\qquad\quad \eta_{p_1,1},\eta_{p_2,1})),\\
  & \qquad\qquad \eta_{p_1,1},\eta_{p_2,1}),\\
  & \quad \text{where } \eta_{p_1,1} = thru_{1,1,1,1,Z_P}(p_2(x_1,y_1),q_1(x_1,y_1),ctx_{p_1,\alpha(q_1(x_1,y_1)),1,1}(\epsilon,1))\\
  & \qquad\qquad \eta_{p_2,1} = thru_{1,1,1,1,Z_P}(y_1,q_1(x_1,y_1),ctx_{p_1,\alpha(q_1(x_1,y_1)),1,1}(\epsilon,1))\\
  & \quad = \\
  & (1_{q_1},1_{p_1})(x_1,nil,y_{1,p_2},\eta_{p_1,1},\eta_{p_2,1}),\\
  & \quad \text{where } \eta_{p_1,1} = trans_{p_2,1,1,Z_P}(q_1(x_1,y_1),thru_{1,1,1,1,Z_P}(y_1,q_1(x_1,y_1),z_{p_1,1}))\\
  & \qquad\qquad \eta_{p_2,1} = z_{p_1,1}\\
  & \quad = \\
  & (1_{q_1},1_{p_1})(x_1,nil,y_{1,p_2},\eta_{p_1,1},z_{p_1,1}),\\
  & \quad \text{where } \eta_{p_1,1} = (q_1,p_2)(x_1,unfold_{q_1(x_1),1,1,Z_P}(1,p_1,\emptyset,y_1,nil,z_{p_1,1}),\\
  & \qquad\qquad\qquad\qquad unfold_{q_1(x_1),1,1,Z_P}(1,p_2,\emptyset,y_1,nil,z_{p_1,1}),z_{p_1,1})\\
  & \quad = \\
  & (1_{q_1},1_{p_1})(x_1,nil,y_{1,p_2},(q_1,p_2)(x_1,y_{1,p_1},y_{1,p_2},z_{p_1,1}),z_{p_1,1})
  \end{aligned}$$
- $(1_{q_1},1_{p_1})(B(x_1),y_{1,p_1},y_{1,p_2},z_{p_1,1},z_{p_2,1}) \to ctx_{p_1,rhs_{q_1,B},1,1}(21,1)$, with:
  $$\begin{aligned}
  & ctx_{p_1,q_1(x_1,\beta(y_1)),1,1}(21,1)\\
  & \quad = \text{ (because of rule } p_2(\beta(x_1),y_1) \to p_2(x_1,p_1(x_1,y_1)) \text{ in } R_2)\\
  & thru_{1,1,1,1,Z_P}(y_1,y_1,ctx_{p_2,q_1(x_1,\beta(y_1)),1,1}(2,1))\\
  & \quad = \\
  & ctx_{p_2,q_1(x_1,\beta(y_1)),1,1}(2,1)
  \end{aligned}$$

$$=$$

$$\begin{aligned}(1_{q_1}, 1_{p_2})(x_1, &\mathit{unfold}_{q_1(x_1),1,1,Z_P}(1, p_1, \{(1,p_2)\}, \beta(y_1), ctx_{p_1,q_1(x_1,\beta(y_1)),1,1}(\epsilon,1),\\ &\qquad ctx_{p_2,q_1(x_1,\beta(y_1)),1,1}(\epsilon,1)),\\ &\mathit{unfold}_{q_1(x_1),1,1,Z_P}(1, p_2, \{(1,p_2)\}, \beta(y_1), ctx_{p_1,q_1(x_1,\beta(y_1)),1,1}(\epsilon,1),\\ &\qquad ctx_{p_2,q_1(x_1,\beta(y_1)),1,1}(\epsilon,1)),\\ &ctx_{p_1,q_1(x_1,\beta(y_1)),1,1}(\epsilon,1), ctx_{p_2,q_1(x_1,\beta(y_1)),1,1}(\epsilon,1))\end{aligned}$$

$$=$$

$$\begin{aligned}(1_{q_1}, 1_{p_2})(x_1, &\mathit{trans}_{p_1,1,1,Z_P}(\beta(y_1),\\ &\quad (1_{q_1}, 1_{p_1})(x_1, \mathit{unfold}_{q_1(x_1),1,1,Z_P}(1, p_1, \{(1,p_2),(1,p_1)\}, \beta(y_1),\\ &\qquad\qquad z_{p_1,1}, z_{p_2,1}),\\ &\qquad \mathit{unfold}_{q_1(x_1),1,1,Z_P}(1, p_2, \{(1,p_2),(1,p_1)\}, \beta(y_1),\\ &\qquad\qquad z_{p_1,1}, z_{p_2,1}),\\ &\qquad z_{p_1,1}, z_{p_2,1})),\\ &nil,\\ &z_{p_1,1}, z_{p_2,1})\end{aligned}$$

$$=$$

$$(1_{q_1}, 1_{p_2})(x_1, succ((1_{q_1}, 1_{p_1})(x_1, nil, nil, z_{p_1,1}, z_{p_2,1})), nil, z_{p_1,1}, z_{p_2,1})$$

- $(1_{q_1}, 1_{p_1})(E, y_{1,p_1}, y_{1,p_2}, z_{p_1,1}, z_{p_2,1}) \to ctx_{p_1,rhs_{q_1,E},0,1}(\epsilon, 1)$, with:

  $$ctx_{p_1,y_1,0,1}(\epsilon,1)$$
  $$=$$
  $$z_{p_1,1}$$

- $(1_{q_1}, 1_{p_2})(A(x_1), y_{1,p_1}, y_{1,p_2}, z_{p_1,1}, z_{p_2,1}) \to ctx_{p_2,rhs_{q_1,A},1,1}(12, 1)$, with:

  $$ctx_{p_2,\alpha(q_1(x_1,y_1)),1,1}(12,1)$$
  $$=$$
  $$(1_{q_1}, 1_{p_2})(x_1, y_{1,p_1}, nil, ctx_{p_1,\alpha(q_1(x_1,y_1)),1,1}(1,1), ctx_{p_2,\alpha(q_1(x_1,y_1)),1,1}(1,1))$$
  $$=$$
  $$(1_{q_1}, 1_{p_2})(x_1, y_{1,p_1}, nil, (q_1, p_2)(x_1, y_{1,p_1}, y_{1,p_2}, z_{p_1,1}), z_{p_1,1})$$

- $(1_{q_1}, 1_{p_2})(B(x_1), y_{1,p_1}, y_{1,p_2}, z_{p_1,1}, z_{p_2,1}) \to ctx_{p_2,rhs_{q_1,B},1,1}(21, 1)$, with:

  $$ctx_{p_2,q_1(x_1,\beta(y_1)),1,1}(21,1)$$
  $$=$$
  $$thru_{1,1,1,1,Z_P}(p_1(x_1,y_1), y_1, ctx_{p_2,q_1(x_1,\beta(y_1)),1,1}(2,1))$$
  $$=$$
  $$y_{1,p_1}$$

- $(1_{q_1}, 1_{p_2})(E, y_{1,p_1}, y_{1,p_2}, z_{p_1,1}, z_{p_2,1}) \to ctx_{p_2,rhs_{q_1,E},0,1}(\epsilon, 1)$, with:

  $$ctx_{p_2,y_1,0,1}(\epsilon,1)$$
  $$=$$
  $$z_{p_2,1}$$

Thus, we get the following set of rules:

$$
\begin{array}{ll}
(q_1,p_1)(A(x_1),y_{1,p_1},y_{1,p_2},z_1) & \to (q_1,p_1)(x_1,y_{1,p_1},y_{1,p_2},(q_1,p_2)(x_1,y_{1,p_1},y_{1,p_2},z_1))\\
(q_1,p_1)(B(x_1),y_{1,p_1},y_{1,p_2},z_1) & \to (q_1,p_1)(x_1,succ((1_{q_1},1_{p_1})(x_1,nil,y_{1,p_2},z_1,nil)),y_{1,p_2},z_1)\\
(q_1,p_1)(E,y_{1,p_1},y_{1,p_2},z_1) & \to y_{1,p_1}\\
(q_1,p_2)(A(x_1),y_{1,p_1},y_{1,p_2},z_1) & \to z_1\\
(q_1,p_2)(B(x_1),y_{1,p_1},y_{1,p_2},z_1) & \to (q_1,p_2)(x_1,succ((1_{q_1},1_{p_1})(x_1,nil,y_{1,p_2},nil,z_1)),y_{1,p_2},z_1)\\
(q_1,p_2)(E,y_{1,p_1},y_{1,p_2},z_1) & \to y_{1,p_2}\\
(1_{q_1},1_{p_1})(A(x_1),y_{1,p_1},y_{1,p_2},z_{p_1,1},z_{p_2,1}) & \to (1_{q_1},1_{p_1})(x_1,nil,y_{1,p_2},(q_1,p_2)(x_1,y_{1,p_1},y_{1,p_2},z_{p_1,1}),z_{p_1,1})\\
(1_{q_1},1_{p_1})(B(x_1),y_{1,p_1},y_{1,p_2},z_{p_1,1},z_{p_2,1}) & \to (1_{q_1},1_{p_2})(x_1,succ((1_{q_1},1_{p_1})(x_1,nil,nil,z_{p_1,1},z_{p_2,1})),\\
 & \qquad\qquad\qquad\qquad\qquad\qquad nil,z_{p_1,1},z_{p_2,1})\\
(1_{q_1},1_{p_1})(E,y_{1,p_1},y_{1,p_2},z_{p_1,1},z_{p_2,1}) & \to z_{p_1,1}\\
(1_{q_1},1_{p_2})(A(x_1),y_{1,p_1},y_{1,p_2},z_{p_1,1},z_{p_2,1}) & \to (1_{q_1},1_{p_2})(x_1,y_{1,p_1},nil,(q_1,p_2)(x_1,y_{1,p_1},y_{1,p_2},z_{p_1,1}),z_{p_1,1})\\
(1_{q_1},1_{p_2})(B(x_1),y_{1,p_1},y_{1,p_2},z_{p_1,1},z_{p_2,1}) & \to y_{1,p_1}\\
(1_{q_1},1_{p_2})(E,y_{1,p_1},y_{1,p_2},z_{p_1,1},z_{p_2,1}) & \to z_{p_2,1}
\end{array}
$$

□

Now, we want to use the previous example to illustrate some points that have been discussed abstractly in the last section.

1. First, we focus on the rule constructed for (q_1,p_1) at B, with right-hand side:
$$(q_1,p_1)(x_1,succ((1_{q_1},1_{p_1})(x_1,nil,y_{1,p_2},z_1,\underline{nil})),y_{1,p_2},z_1),$$
containing two occurrences of nil. The second one—in the $z_{p_2,1}$-position of state $(1_{q_1},1_{p_1})$—is of the kind introduced into the expression **(***)** on page 20, where we promised the example given now. We can by no means assume that state $(1_{q_1},1_{p_1})$ will never need its last context parameter position, in fact we have that $(1_{q_1},1_{p_1})(B(E),y_{1,p_1},y_{1,p_2},z_{p_1,1},z_{p_2,1})$ reduces to $z_{p_2,1}$. Instead, we have to argue according to the place, where the $(1_{q_1},1_{p_1})$-call occurs, namely in the first context parameter position of a (q_1,p_1)-call on the same tree argument. Thus, we need to explain why there can be no input tree t, such that (q_1,p_1) called on t depends on its y_{1,p_1}-position and $(1_{q_1},1_{p_1})$ called on t depends on its $z_{p_2,1}$-position[5]. This could be proven formally, we reason intuitively as follows. If the normal form of $(q_1,p_1)(t,y_{1,p_1},y_{1,p_2},z_1)$ contains y_{1,p_1}, this means that reducing $p_1(q_1(t,y_1),z_1)$ will lead to a call of p_1 on y_1. By the weakly single-use property of M_2 we have argued that there can be only one state p' such that reducing $p'(q_1(t,y_1),\cdots)$ leads to a p_1-call on y_1. For our particular t, we know that this p' is p_1. But then, $(1_{q_1},1_{p_1})$ called on t (which computes the first context parameter in a call to p_1 on the first context variable of q_1, resulting from reduction of $p'(q_1(t,y_1),\cdots)$) will not depend on $z_{p_2,1}$, because $p'\neq p_2$, and—as indicated on page 18—the state $(1_{q_1},1_{p_1})$ called on t will of its $z_{p_1,1},z_{p_2,1}$-positions only depend on those, which are associated with p'. Note, that the situation is analogous for the nil-symbol in the $z_{p_1,1}$-position of the $(1_{q_1},1_{p_1})$-call in the right-hand side for state (q_1,p_2) at B: $(q_1,p_2)(x_1,succ((1_{q_1},1_{p_1})(x_1,nil,y_{1,p_2},\underline{nil},z_1)),y_{1,p_2},z_1)$.

2. Now, we want to consider the nil-symbols, which occur as first context parameter of $(1_{q_1},1_{p_1})$-calls in several rules and the initial expression:
$$\begin{array}{l}(q_1,p_2)(x_1,(1_{q_1},1_{p_1})(x_1,\underline{nil},(1_{q_1},1_{p_2})(x_1,nil,nil,nil,zero),nil,zero),\\ \qquad\qquad (1_{q_1},1_{p_2})(x_1,(1_{q_1},1_{p_1})(x_1,\underline{nil},nil,nil,zero),nil,nil,zero),\\ \qquad\qquad zero)\end{array}$$

[5]Note, that $(q_1,p_1)(B(E),y_{1,p_1},y_{1,p_2},z_1)$ reduces to $succ(z_1)$ and thus does not depend on its y_{1,p_1}-position.

These are justified by the fact that for no input tree t, the state $(1_{q_1}, 1_{p_1})$ depends on its y_{1,p_1}-position (analogous for state $(1_{q_1}, 1_{p_2})$ and y_{1,p_2}). This proposition holds, because $(1_{q_1}, 1_{p_1})$ on t computes the first context parameter position of a p_1-call on y_1, resulting from reduction of some $p'(q_1(t, y_1), \cdots)$. If this value would depend on the p_1-translation of y_1, we would have met during the reduction a situation like $p_1(y_1, \cdots p_1(y_1, \cdots) \cdots)$, which we argued—at the end of Section 3.1—to be impossible for non-copying M_1 and weakly single-use M_2.

3. We come to the explanation for the nil-symbol in the y_{1,p_2}-position of the second $(1_{q_1}, 1_{p_1})$-call in the initial expression:

$$\begin{aligned}(q_1, p_2)(x_1, &(1_{q_1}, 1_{p_1})(x_1, nil, (1_{q_1}, 1_{p_2})(x_1, nil, nil, nil, zero), nil, zero),\\ &(1_{q_1}, 1_{p_2})(x_1, (1_{q_1}, 1_{p_1})(x_1, nil, \underline{nil}, nil, zero), nil, nil, zero),\\ &zero),\end{aligned}$$

which stems from the "cutting" of circular nesting during the unfolding process, described at the end of last section. Note, that in general $(1_{q_1}, 1_{p_1})$ might very well depend on its second context parameter, e.g., $(1_{q_1}, 1_{p_1})(A(E), y_{1,p_1}, y_{1,p_2}, z_{p_1,1}, z_{p_2,1})$ reduces to y_{1,p_2}. But, we can argue that there can be no input tree t, such that $(1_{q_1}, 1_{p_2})$ called on t depends on its y_{1,p_1}-position and $(1_{q_1}, 1_{p_1})$ called on t depends on its y_{1,p_2}-position[6], by reasoning intuitively as follows. If the normal form of $(1_{q_1}, 1_{p_2})(t, y_{1,p_1}, y_{1,p_2}, z_{p_1,1}, z_{p_2,1})$ contains y_{1,p_1}, this means that the first context parameter of every p_2-call on y_1, resulting from reduction of $p'(q_1(t, y_1), \cdots)$ for some p', depends on the p_1-translation of y_1, in fact a situation like $p_2(y_1, \cdots p_1(y_1, ?) \cdots)$ is met during this reduction. The call of $(1_{q_1}, 1_{p_1})$ on t computes exactly the context parameter of this call of p_1 on y_1, i.e., the value in the question mark position. But, if this value would depend on the y_{1,p_2}-position of $(1_{q_1}, 1_{p_1})$, then the question mark position would in fact contain some value depending on the p_2-translation of y_1, giving as result of the above reduction a normal form containing a situation like $p_2(y_1, \cdots p_1(y_1, \cdots p_2(y_1, \cdots) \cdots) \cdots)$. This clearly contradicts the statement that—if M_1 is non-copying and M_2 weakly single-use—every two calls of p_2 on y_1 will have identical context parameters.

4. Finally, we turn to the right-hand side for state $(1_{q_1}, 1_{p_1})$ at B and the nil-symbol, which occurs there in the second context parameter position of the recursive call to $(1_{q_1}, 1_{p_1})$:

$$(1_{q_1}, 1_{p_2})(x_1, succ((1_{q_1}, 1_{p_1})(x_1, nil, \underline{nil}, z_{p_1,1}, z_{p_2,1})), nil, z_{p_1,1}, z_{p_2,1})$$

This right-hand side was calculated as $ctx_{p_2, q_1(x_1, \beta(y_1)), 1, 1}(2, 1)$, as defined in the last case for the computation of ctx-functions on page 23. The nil-symbol is created, because the unfolding is started with the set of "cut-positions" $\{(1, p_2)\}$, as justified in the last case of explaining the working of $ctx_{p,\phi}$-functions on page 24. It will never appear in the output, because—as argued above in point 3.—there is no input tree t, such that $(1_{q_1}, 1_{p_2})$ called on t depends on its first context parameter and $(1_{q_1}, 1_{p_1})$ called on t depends on its second context parameter.

After having discussed the previous rather artificial example at such length, let us conclude this section with a more practical example, namely the composition of M_{rev} with itself.

[6] Note, that $(1_{q_1}, 1_{p_2})(A(E), y_{1,p_1}, y_{1,p_2}, z_{p_1,1}, z_{p_2,1})$ reduces to $z_{p_1,1}$ and thus does not depend on its y_{1,p_1}-position.

Example 3.4 The composition of Macro Tree Transducers M_{rev} and M_{rev} according to Construction 3.2 yields $M_{rev;rev} = (S_{rev;rev}, \Sigma_{mon}, \Sigma_{mon}, e_{rev;rev}, R_{rev;rev})$ with $S_{rev;rev} = \{(rev, rev)^{(3)}, (1_{rev}, 1_{rev})^{(3)}\}$, set of rules $R_{rev;rev}$:

$$
\begin{array}{lll}
\text{(i)} & : (rev,rev)(A(x_1), y_{1,rev}, z_1) & \rightarrow (rev,rev)(x_1, y_{1,rev}, z_1) \\
\text{(ii)} & : (rev,rev)(B(x_1), y_{1,rev}, z_1) & \rightarrow (rev,rev)(x_1, y_{1,rev}, z_1) \\
\text{(iii)} & : (rev,rev)(E, y_{1,rev}, z_1) & \rightarrow y_{1,rev} \\
\text{(iv)} & : (1_{rev},1_{rev})(A(x_1), y_{1,rev}, z_{rev,1}) & \rightarrow A((1_{rev},1_{rev})(x_1, E, z_{rev,1})) \\
\text{(v)} & : (1_{rev},1_{rev})(B(x_1), y_{1,rev}, z_{rev,1}) & \rightarrow B((1_{rev},1_{rev})(x_1, E, z_{rev,1})) \\
\text{(vi)} & : (1_{rev},1_{rev})(E, y_{1,rev}, z_{rev,1}) & \rightarrow z_{rev,1}
\end{array}
$$

and $e_{rev;rev} = (rev, rev)(x_1, (1_{rev}, 1_{rev})(x_1, E, E), E)$. □

The resulting Macro Tree Transducer $M_{rev;rev}$ looks rather complicated. In Chapter 4, we will present constructions for post-processing and optimizing Macro Tree Transducers obtained from the composition construction.

3.3 The symmetric Result does not hold

In Section 3.2, we have presented a construction, which composes a non-copying Macro Tree Transducer and a weakly single-use one, to a single Macro Tree Transducer. We might try to intuitively understand that this is possible, because the first Macro Tree Transducer forbids the copying of context information, whereas the second one forbids multiple use of recursive computations on subtrees. It is thus naturally to ask, whether these two restrictions of controlling the copying of context information and recursive computations on subtrees, respectively, also work the other way round, that is, whether we also have $MAC_{wsu}; MAC_{nc} \subseteq MAC$. In this section, we will show that this is not the case, by giving a counterexample. In fact, we will even show the stronger result that $MAC_{su}; MAC_{nc} \not\subseteq MAC$.

Example 3.5

- Let $\Sigma_{bin} = \{\sigma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$, $Nat = \{succ^{(1)}, zero^{(0)}\}$ and $S_{\Sigma_{bin}}$ be the single-use Macro Tree Transducer $(\{s^{(2)}\}, \Sigma_{bin}, Nat, s(x_1, zero), R_{\Sigma_{bin}})$ with set of rules $R_{\Sigma_{bin}}$:

$$
\begin{array}{lll}
\text{(i)} & : s(\sigma(x_1, x_2), y_1) & \rightarrow succ(s(x_1, s(x_2, y_1))) \\
\text{(ii)} & : s(\alpha, y_1) & \rightarrow succ(y_1) \\
\text{(iii)} & : s(\beta, y_1) & \rightarrow succ(y_1)
\end{array}
$$

  We claim that for every $t \in T_{\Sigma_{bin}}$, we have: $height(\tau(S_{\Sigma_{bin}})(t)) = size(t)$.

- Consider the following non-copying Macro Tree Transducer $M_{exp} = (\{exp^{(2)}\}, Nat, Nat, exp(x_1, zero), R_{exp})$ with set of rules R_{exp}:

$$
\begin{array}{lll}
\text{(i)} & : exp(succ(x_1), y_1) & \rightarrow exp(x_1, exp(x_1, y_1)) \\
\text{(ii)} & : exp(zero, y_1) & \rightarrow succ(y_1)
\end{array}
$$

  We claim that for every $s \in T_{Nat}$, we have: $height(\tau(M_{exp})(s)) = 2^{height(s)}$.

□

The claim about $S_{\Sigma_{bin}}$ will be verified in Section 5.1 in a more general construction for arbitrary ranked alphabets. The claim about M_{exp} is folklore and might easily be proven by induction.

Here, we quote a classical result on Macro Tree Transducers (cf. [EV85]), namely that the heights of their output trees are exponentially bounded by the heights of their input trees.

Lemma 3.6 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer. There exists a constant $c \in \mathbf{N}$, such that for every input tree $t \in T_\Sigma$ holds: $height(\tau(M)(t)) \leq c^{height(t)}$. □

Now, we can show by counterexample that we do not have $MAC_{su}; MAC_{nc} \subseteq MAC$.

Theorem 3.7 There exist Macro Tree Transducers M_1 and M_2, which are single-use and non-copying, respectively, such that their composition cannot be implemented by a Macro Tree Transducer. □

Proof: Take $S_{\Sigma_{bin}}$ and M_{exp} as M_1 and M_2, respectively, and assume there exists a Macro Tree Transducer M, such that $\tau(M) = \tau(S_{\Sigma_{bin}}); \tau(M_{exp})$.
Then, for every $t \in T_{\Sigma_{bin}}$, we would have $\tau(M)(t) = \tau(M_{exp})(\tau(S_{\Sigma_{bin}})(t))$. According to Lemma 3.6, there must exist a constant $c \in \mathbf{N}$, such that $height(\tau(M)(t)) \leq c^{height(t)}$ for every such t.
We calculate:

$$\begin{aligned}
&height(\tau(M)(t)) = \\
&height(\tau(M_{exp})(\tau(S_{\Sigma_{bin}})(t))) = \\
&2^{height(\tau(S_{\Sigma_{bin}})(t))} = \\
&2^{size(t)}
\end{aligned}$$

Thus, there has to be a constant $c \in \mathbf{N}$, such that for every $t \in T_{\Sigma_{bin}}$, we have $2^{size(t)} \leq c^{height(t)}$. Taking for t some fully balanced binary tree of height h over the ranked alphabet Σ_{bin}, we get $size(t) = 2^{h+1} - 1$ and thus $2^{(2^{h+1}-1)} \leq c^h$ for every $h \in \mathbf{N}$. But, such a c cannot exist, due to standard function analysis. □

Chapter 4

Tidying up

The Macro Tree Transducer $M_{rev;rev}$, as constructed in Example 3.4, looks quite complicated. Clearly, we would expect the composition of two `reverse`-functions to be an identity function. Indeed, $M_{rev;rev}$ *does* implement the identity function, but in a very inefficient way. The reason is twofold.
First, we observe that the construction introduces context parameters, which are *superfluous*, in the sense that they will never (for no possible input tree) influence the output generated by a state. This is the case for parameter z_1 of state (rev, rev) and parameter $y_{1,rev}$ of state $(1_{rev}, 1_{rev})$.
Second, the state (rev, rev) never really performs any meaningful computation. No matter, on which input tree it is called, it will always just project on its first context parameter. We call a state showing such a behavior a *copy-state*[1].

The first problem is caused by the need for our construction to be as general as possible. There might actually be more complicated Macro Tree Transducers, where the additionally created context parameters are really needed. However, the presence of these superfluous context parameters does not influence the efficiency of our constructed Macro Tree Transducers, if lazy evaluation is used and we take the number of reduction steps as efficiency measure. Nevertheless, we still want to get rid of them, because they obscure the programs and, moreover, *do* influence the efficiency, if more detailed measures are used, such as taking into account the cost of allocating memory for representing function closures.
The second problem is more serious, as it leads to superfluous traversals through the input data structure and so contradicts our aim of *optimizing* functional programs by eliminating intermediate data structures.
In our example $M_{rev;rev}$, it is pretty obvious how to solve the two problems, but in general, both, superfluous context parameters and copy-states, are more difficult to detect. In this chapter, we are going to present two constructions (based on finding finite fixpoints), which allow to detect and remove superfluous context parameters and copy-states, respectively.

Example 4.1 Using our two proposed constructions on $M_{rev;rev}$, we will obtain the Macro Tree Transducer $M_{rev^2} = (S_{rev^2}, \Sigma_{mon}, \Sigma_{mon}, e_{rev^2}, R_{rev^2})$ with $S_{rev^2} = \{(1_{rev}, 1_{rev})^{(2)}\}$, set of rules R_{rev^2}:

$$\begin{array}{lll}
\text{(i)} : & (1_{rev}, 1_{rev})(A(x_1), z_{rev,1}) & \to A((1_{rev}, 1_{rev})(x_1, z_{rev,1})) \\
\text{(ii)} : & (1_{rev}, 1_{rev})(B(x_1), z_{rev,1}) & \to B((1_{rev}, 1_{rev})(x_1, z_{rev,1})) \\
\text{(iii)} : & (1_{rev}, 1_{rev})(E, z_{rev,1}) & \to z_{rev,1}
\end{array}$$

and $e_{rev^2} = (1_{rev}, 1_{rev})(x_1, E)$. □

[1] [CDPR99] introduced the concept of *copy-rules* for attribute grammars, our notion is more general.

4.1 Removing superfluous Context Parameters

Let us first consider the problem of detecting, if a state of a Macro Tree Transducer does not need all its context parameters. In order to give a formal description of the problem, we need a notion of how computation on a tree depends on its context. For this purpose, we introduce the concept of so-called cd-functions[2].

Definition 4.2 A *cd-function* (over a ranked alphabet Q of states) is any element of the indexed product $\mathcal{CD} = \prod_{q \in Q^{(r+1)}} \mathcal{P}([r])$. □

Example 4.3 For $Q = \{q^{(2)}, p^{(3)}\}$, the function $[q \mapsto \emptyset, p \mapsto \{1,2\}] \in \mathcal{CD}$ is a cd-function. □

Thus, a cd-function determines for every state q of rank $r+1$ a subset of $[r]$, intuitively meant to describe, which of its context parameters the state will actually need in a computation. Note, that there are only finitely many cd-functions—because Q is a finite ranked alphabet—and that the use of an indexed product allows to effectively represent cd-functions.

We are interested in describing, whether a certain context parameter of a state is ever needed during any computation. That is, we need to check this property for all possible input trees. Therefore, we will first define the cd-function of a particular input tree t.

Definition 4.4 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer and $t \in T_\Sigma$. The *cd-function of t for M*, denoted by $cd_M(t)$, is defined, for every $r \in \mathbf{N}$ and $q \in Q^{(r+1)}$, by: $cd_M(t)(q) = \{\nu \in [r] \mid \#_\nu(\tau_{M,q,[r]}(t, 1, \ldots, r)) > 0\}$. □

Rather than computing cd-functions with this definition, we would like to express the cd-function of a tree inductively in terms of the cd-functions of its immediate subtrees. Therefore, we define a function $cd_{M,\sigma}$, for every $\sigma \in \Sigma$, as follows.

Definition 4.5 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer. For every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$, the function $cd_{M,\sigma} : (\mathcal{CD})^k \to \mathcal{CD}$ is defined as follows: for every $cd_1, \ldots, cd_k \in \mathcal{CD}$, $r \in \mathbf{N}$ and $q \in Q^{(r+1)}$, we have $cd_{M,\sigma}(cd_1, \ldots, cd_k)(q) = C_{M,k,r}(rhs_{q,\sigma}, cd_1, \ldots, cd_k)$, if R contains the rule $q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r) \to rhs_{q,\sigma}$, with $rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$. For every $k \in rank(\Sigma)$ and $r+1 \in rank(Q)$, the following function is defined. $C_{M,k,r} : RHS(Q, \Delta, k, Y_r) \underbrace{\times \cdots \times \mathcal{CD}}_{k-times} \to \mathcal{P}([r])$, given by:

- $C_{M,k,r}(y_h, cd_1, \ldots, cd_k) = \{h\}$,
  for $y_h \in Y_r$ and $cd_1, \ldots, cd_k \in \mathcal{CD}$
- $C_{M,k,r}(\delta(\phi_1, \ldots, \phi_a), cd_1, \ldots, cd_k) = \bigcup_{l \in [a]} C_{M,k,r}(\phi_l, cd_1, \ldots, cd_k)$,
  for $a \in \mathbf{N}$, $\delta \in \Delta^{(a)}$, $\phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$ and $cd_1, \ldots, cd_k \in \mathcal{CD}$
- $C_{M,k,r}(q(x_b, \phi_1, \ldots, \phi_a), cd_1, \ldots, cd_k) = \bigcup_{l \in cd_b(q)} C_{M,k,r}(\phi_l, cd_1, \ldots, cd_k)$,
  for $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $x_b \in X_k$, $\phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$ and $cd_1, \ldots, cd_k \in \mathcal{CD}$

□

[2] cf. *context dependency*; the corresponding concept on Attributed Tree Transducers is that of an *is-graph* (see [FV98]).

The intuitive idea behind this recursive computation is that for some $q \in Q^{(r+1)}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \dots, t_k \in T_\Sigma$, the normal form of $q(\sigma(t_1, \dots, t_k), y_1, \dots, y_r)$ will contain exactly those context parameters, which are contained in the normal form of $rhs_{q,\sigma}[x_b \leftarrow t_b, 1 \leq b \leq k]$. Thus, we can simply collect all the y_h (with $1 \leq h \leq r$) in $rhs_{q,\sigma}$, where in occurrences of function calls on some x_b, only those context parameter positions are considered, which are determined—by the cd-function of t_b—to actually appear in the output.

Example 4.6 Recall the Macro Tree Transducer (after renaming of context variables) $M_{rev;rev} = (S_{rev;rev}, \Sigma_{mon}, \Sigma_{mon}, e_{rev;rev}, R_{rev;rev})$ with $S_{rev;rev} = \{(rev, rev)^{(3)}, (1_{rev}, 1_{rev})^{(3)}\}$ and set of rules $R_{rev;rev}$:

$$\begin{array}{lll}
\text{(i)} & : (rev, rev)(A(x_1), y_1, y_2) & \rightarrow (rev, rev)(x_1, y_1, y_2) \\
\text{(ii)} & : (rev, rev)(B(x_1), y_1, y_2) & \rightarrow (rev, rev)(x_1, y_1, y_2) \\
\text{(iii)} & : (rev, rev)(E, y_1, y_2) & \rightarrow y_1 \\
\text{(iv)} & : (1_{rev}, 1_{rev})(A(x_1), y_1, y_2) & \rightarrow A((1_{rev}, 1_{rev})(x_1, E, y_2)) \\
\text{(v)} & : (1_{rev}, 1_{rev})(B(x_1), y_1, y_2) & \rightarrow B((1_{rev}, 1_{rev})(x_1, E, y_2)) \\
\text{(vi)} & : (1_{rev}, 1_{rev})(E, y_1, y_2) & \rightarrow y_2
\end{array}$$

and $e_{rev;rev} = (rev, rev)(x_1, (1_{rev}, 1_{rev})(x_1, E, E), E)$.

Let $cd_1 = [(rev, rev) \mapsto \{1\}, (1_{rev}, 1_{rev}) \mapsto \{2\}] \in \mathcal{CD}$ and calculate $cd_{M,A}(cd_1)$ as follows. We have:

$$cd_{M,A}(cd_1)((rev, rev)) = C_{M,1,2}((rev, rev)(x_1, y_1, y_2), cd_1) = C_{M,1,2}(y_1, cd_1) = \{1\}$$

and:

$$\begin{array}{l}
cd_{M,A}(cd_1)((1_{rev}, 1_{rev})) = C_{M,1,2}(A((1_{rev}, 1_{rev})(x_1, E, y_2)), cd_1) = \\
C_{M,1,2}((1_{rev}, 1_{rev})(x_1, E, y_2), cd_1) = C_{M,1,2}(y_2, cd_1) = \{2\},
\end{array}$$

consequently: $cd_{M,A}(cd_1) = [(rev, rev) \mapsto \{1\}, (1_{rev}, 1_{rev}) \mapsto \{2\}] \in \mathcal{CD}$. Similarly, we can show that $cd_{M,B}(cd_1) = [(rev, rev) \mapsto \{1\}, (1_{rev}, 1_{rev}) \mapsto \{2\}]$. □

The following lemma shows that the $cd_{M,\sigma}$-functions indeed allow the recursive computation of cd-functions of trees.

Lemma 4.7 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer. For every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \dots, t_k \in T_\Sigma$, we have $cd_M(\sigma(t_1, \dots, t_k)) = cd_{M,\sigma}(cd_M(t_1), \dots, cd_M(t_k))$. □

Proof:

- First, we show an auxiliary proposition. We prove for every $k \in rank(\Sigma)$, $t_1, \dots, t_k \in T_\Sigma$ and $r + 1 \in rank(Q)$, by structural induction on $\phi \in RHS(Q, \Delta, k, Y_r)$, that the following holds:
  $\{\nu \in [r] \mid \#_\nu(\tau_{M,k,r,[r]}(\phi, t_1, \dots, t_k, 1, \dots, r)) > 0\} = C_{M,k,r}(\phi, cd_M(t_1), \dots, cd_M(t_k))$.

  $\phi = y_h \in Y_r$:

  $$\begin{array}{l}
  \{\nu \in [r] \mid \#_\nu(\tau_{M,k,r,[r]}(y_h, t_1, \dots, t_k, 1, \dots, r)) > 0\} \\
  \quad = \text{(by definition of } \tau_{M,k,r,[r]}) \\
  \{\nu \in [r] \mid \#_\nu(h) > 0\} \\
  \quad = \text{(by definition of } \#_\nu \text{ and set theory)} \\
  \{h\} \\
  \quad = \text{(by definition of } C_{M,k,r}) \\
  C_{M,k,r}(y_h, cd_M(t_1), \dots, cd_M(t_k))
  \end{array}$$

$\phi = \delta(\phi_1, \ldots, \phi_a), a \in \mathbf{N}, \delta \in \Delta^{(a)}, \phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$:

$$
\begin{aligned}
&\{\nu \in [r] \mid \#_\nu(\tau_{M,k,r,[r]}(\delta(\phi_1, \ldots, \phi_a), t_1, \ldots, t_k, 1, \ldots, r)) > 0\}\\
&\quad = \text{(by definition of } \tau_{M,k,r,[r]})\\
&\{\nu \in [r] \mid \#_\nu(\delta(\tau_{M,k,r,[r]}(\phi_1, t_1, \ldots, t_k, 1, \ldots, r),\\
&\qquad\qquad \ldots,\\
&\qquad\qquad \tau_{M,k,r,[r]}(\phi_a, t_1, \ldots, t_k, 1, \ldots, r))) > 0\}\\
&\quad = \text{(by definition of } \#_\nu)\\
&\{\nu \in [r] \mid \sum_{l \in [a]} \#_\nu(\tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r)) > 0\}\\
&\quad = \text{(by arithmetics)}\\
&\{\nu \in [r] \mid \bigvee_{l \in [a]} (\#_\nu(\tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r)) > 0)\}\\
&\quad = \text{(by set theory)}\\
&\bigcup_{l \in [a]} \{\nu \in [r] \mid \#_\nu(\tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r)) > 0\}\\
&\quad = \text{(by induction hypothesis for } \phi_l, l \in [a])\\
&\bigcup_{l \in [a]} C_{M,k,r}(\phi_l, cd_M(t_1), \ldots, cd_M(t_k))\\
&\quad = \text{(by definition of } C_{M,k,r})\\
&C_{M,k,r}(\delta(\phi_1, \ldots, \phi_a), cd_M(t_1), \ldots, cd_M(t_k))
\end{aligned}
$$

$\phi = q(x_b, \phi_1, \ldots, \phi_a), a \in \mathbf{N}, q \in Q^{(a+1)}, x_b \in X_k, \phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$:

$$
\begin{aligned}
&\{\nu \in [r] \mid \#_\nu(\tau_{M,k,r,[r]}(q(x_b, \phi_1, \ldots, \phi_a), t_1, \ldots, t_k, 1, \ldots, r)) > 0\}\\
&\quad = \text{(by definition of } \tau_{M,k,r,[r]})\\
&\{\nu \in [r] \mid \#_\nu(\tau_{M,q,[r]}(t_b, \tau_{M,k,r,[r]}(\phi_1, t_1, \ldots, t_k, 1, \ldots, r),\\
&\qquad\qquad \ldots,\\
&\qquad\qquad \tau_{M,k,r,[r]}(\phi_a, t_1, \ldots, t_k, 1, \ldots, r))) > 0\}\\
&\quad = \text{(by Lemma 2.16)}\\
&\{\nu \in [r] \mid \#_\nu((\tau_{M,q,[a]}(t_b, 1, \ldots, a))\\
&\qquad\qquad [l \leftarrow \tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r), 1 \le l \le a]) > 0\}\\
&\quad = \text{(by } [r] \cap \Delta = \emptyset\text{, properties of } \#_\nu \text{ and substitution)}\\
&\{\nu \in [r] \mid \bigvee_{l \in [a]} (\#_l(\tau_{M,q,[a]}(t_b, 1, \ldots, a)) > 0\\
&\qquad\qquad \wedge\ \#_\nu(\tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r)) > 0)\}\\
&\quad = \text{(by propositional logic)}\\
&\{\nu \in [r] \mid \bigvee_{l \in [a]\ \wedge\ \#_l(\tau_{M,q,[a]}(t_b, 1, \ldots, a)) > 0} (\#_\nu(\tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r)) > 0)\}\\
&\quad = \text{(by set theory)}\\
&\{\nu \in [r] \mid \bigvee_{l \in \{h \in [a] \mid \#_h(\tau_{M,q,[a]}(t_b, 1, \ldots, a)) > 0\}} (\#_\nu(\tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r)) > 0)\}\\
&\quad = \text{(by definition of } cd_M(t_b))\\
&\{\nu \in [r] \mid \bigvee_{l \in cd_M(t_b)(q)} (\#_\nu(\tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r)) > 0)\}\\
&\quad = \text{(by set theory)}\\
&\bigcup_{l \in cd_M(t_b)(q)} \{\nu \in [r] \mid \#_\nu(\tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r)) > 0\}\\
&\quad = \text{(by induction hypothesis for } \phi_l, l \in [a])\\
&\bigcup_{l \in cd_M(t_b)(q)} C_{M,k,r}(\phi_l, cd_M(t_1), \ldots, cd_M(t_k))\\
&\quad = \text{(by definition of } C_{M,k,r})\\
&C_{M,k,r}(q(x_b, \phi_1, \ldots, \phi_a), cd_M(t_1), \ldots, cd_M(t_k))
\end{aligned}
$$

- Now, we show for every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T_\Sigma$, that
$$cd_M(\sigma(t_1, \ldots, t_k)) = cd_{M,\sigma}(cd_M(t_1), \ldots, cd_M(t_k)),$$
by proving that for every $r \in \mathbf{N}$ and $q \in Q^{(r+1)}$, the following holds (using the auxiliary proposition, which was proven in the first point):

$$\begin{aligned}
& cd_M(\sigma(t_1, \ldots, t_k))(q) \\
& \quad = \text{(by definition of } cd_M(\sigma(t_1, \ldots, t_k))) \\
& \{\nu \in [r] \mid \#_\nu(\tau_{M,q,[r]}(\sigma(t_1, \ldots, t_k), 1, \ldots, r)) > 0\} \\
& \quad = \text{(by definition of } \tau_{M,q,[r]}) \\
& \{\nu \in [r] \mid \#_\nu(\tau_{M,k,r,[r]}(rhs_{q,\sigma}, t_1, \ldots, t_k, 1, \ldots, r)) > 0\} \\
& \quad = \text{(by the auxiliary proposition shown above, with } \phi = rhs_{q,\sigma}) \\
& C_{M,k,r}(rhs_{q,\sigma}, cd_M(t_1), \ldots, cd_M(t_k)) \\
& \quad = \text{(by definition of } cd_{M,\sigma}) \\
& cd_{M,\sigma}(cd_M(t_1), \ldots, cd_M(t_k))(q)
\end{aligned}$$

□

In order to detect superfluous context parameters, we need to examine the cd-functions of *all* possible input trees. Of course, there are infinitely many input trees, but as we remarked above, there are only finitely many different cd-functions. Moreover, since we have the means to compute them recursively, using the $cd_{M,\sigma}$-functions, we can actually effectively construct the set of all possible cd-functions for a given Macro Tree Transducer.

Construction 4.8 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer. Let $S = \mathcal{P}(\mathcal{CD})$. Let the function $\Phi : S \to S$ be defined by: $\Phi(s) = \{cd_{M,\sigma}(cd_1, \ldots, cd_k) \mid k \in \mathbf{N}, \sigma \in \Sigma^{(k)}, cd_1, \ldots, cd_k \in s\}$, for every $s \in S$.
Let Φ^* be the least fixpoint reached after finitely many iterations of Φ, starting with the empty set. We claim: $\Phi^* = \{cd_M(t) \mid t \in T_\Sigma\}$. □

Proof: First, note that Φ is a monotonic operator, and thus the least fixpoint $\Phi^* = \bigcup_{n \in \mathbf{N}} \Phi^n$ indeed exists and is reached after finitely many iterations of Φ.
Now, we show by induction on $n \in \mathbf{N}$ that we have: $\Phi^n = \{cd_M(t) \mid t \in T_\Sigma, height(t) < n\}$.

$n = 0$:
$$\Phi^0 = \emptyset = \{cd_M(t) \mid t \in T_\Sigma, height(t) < 0\}$$

$n > 0$:
$$\begin{aligned}
& \Phi^n \\
& \quad = \text{(by definition of } \Phi^n \text{ for } n > 0) \\
& \Phi(\Phi^{n-1}) \\
& \quad = \text{(by definition of } \Phi) \\
& \{cd_{M,\sigma}(cd_1, \ldots, cd_k) \mid \sigma \in \Sigma^{(k)}, cd_1, \ldots, cd_k \in \Phi^{n-1}\} \\
& \quad = \text{(by induction hypothesis for } n-1) \\
& \{cd_{M,\sigma}(cd_1, \ldots, cd_k) \mid \sigma \in \Sigma^{(k)}, cd_1, \ldots, cd_k \in \{cd_M(t) \mid t \in T_\Sigma, height(t) < n-1\}\} \\
& \quad = \text{(by } cd_1, \ldots, cd_k = cd_M(t_1), \ldots, cd_M(t_k) \text{ for} \\
& \qquad\quad t_1, \ldots, t_k \in \{t \mid t \in T_\Sigma, height(t) < n-1\}) \\
& \{cd_{M,\sigma}(cd_M(t_1), \ldots, cd_M(t_k)) \mid \sigma \in \Sigma^{(k)}, t_1, \ldots, t_k \in \{t \mid t \in T_\Sigma, height(t) < n-1\}\} \\
& \quad = \text{(by Lemma 4.7)} \\
& \{cd_M(\sigma(t_1, \ldots, t_k)) \mid \sigma \in \Sigma^{(k)}, t_1, \ldots, t_k \in \{t \mid t \in T_\Sigma, height(t) < n-1\}\} \\
& \quad = \text{(by set theory and property of } height) \\
& \{cd_M(t) \mid t \in T_\Sigma, height(t) < n\}
\end{aligned}$$

Thus, we have:
$\Phi^* = \bigcup_{n\in\mathbf{N}} \Phi^n = \bigcup_{n\in\mathbf{N}} \{cd_M(t) \mid t \in T_\Sigma, height(t) < n\} = \{cd_M(t) \mid t \in T_\Sigma\}.$

□

Example 4.9 Let M be the Macro Tree Transducer $M_{rev;rev}$ as above in Example 4.6. We perform the following iteration (referring to the cd-function cd_1 and calculation steps from Example 4.6):
$\Phi^0 = \emptyset$
$\Phi^1 = \{cd_{M,E}\} = \{[(rev, rev) \mapsto C_{M,0,2}(y_1), (1_{rev}, 1_{rev}) \mapsto C_{M,0,2}(y_2)]\} = \{cd_1\}$
$\Phi^2 = \{cd_{M,E}, cd_{M,A}(cd_{M,E}), cd_{M,B}(cd_{M,E})\} = \{[(rev, rev) \mapsto \{1\}, (1_{rev}, 1_{rev}) \mapsto \{2\}]\}$
Thus, $\Phi^* = \{[(rev, rev) \mapsto \{1\}, (1_{rev}, 1_{rev}) \mapsto \{2\}]\}$. □

The construction on $M_{rev;rev}$ shows us that a computation of state (rev, rev) never depends on its second context parameter, while a computation of state $(1_{rev}, 1_{rev})$ never depends on its first context parameter. This information will be used to simplify the Macro Tree Transducer. First, we formalize the intuition given here.

Definition 4.10 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer. For a state $q \in Q^{(r+1)}$ (with $r \in \mathbf{N}$), its νth context parameter (with $\nu \in [r]$), is said to be *superfluous for* q, iff for every cd-function $cd \in \Phi^*$, we have: $\nu \notin cd(q)$. □

Since superfluous context parameters do not influence the computations of a Macro Tree Transducer, we can always perform simplifications as follows. First, we replace in every right-hand side of a state $q \in Q^{(r+1)}$ of which we know that it never depends on $y_\nu \in Y_r$, every occurrence of y_ν with some nullary output symbol[3]. Then, we discard in every occurrence of some state q—in left- as well as in right-hand sides and in the initial expression—those context parameter positions, which were detected to be superfluous for q (thus potentially changing the rank of q).

Example 4.11 Given $M_{rev;rev}$, we use the information that y_2 is superfluous for (rev, rev), and y_1 is superfluous for $(1_{rev}, 1_{rev})$, to obtain the simplified Macro Tree Transducer $M'_{rev;rev} = (S'_{rev;rev}, \Sigma_{mon}, \Sigma_{mon}, e'_{rev;rev}, R'_{rev;rev})$ with $S'_{rev;rev} = \{(rev, rev)^{(2)}, (1_{rev}, 1_{rev})^{(2)}\}$, set of rules $R'_{rev;rev}$:

$$
\begin{array}{lll}
\text{(i)} : & (rev, rev)(A(x_1), y_1) & \to (rev, rev)(x_1, y_1) \\
\text{(ii)} : & (rev, rev)(B(x_1), y_1) & \to (rev, rev)(x_1, y_1) \\
\text{(iii)} : & (rev, rev)(E, y_1) & \to y_1 \\
\text{(iv)} : & (1_{rev}, 1_{rev})(A(x_1), y_2) & \to A((1_{rev}, 1_{rev})(x_1, y_2)) \\
\text{(v)} : & (1_{rev}, 1_{rev})(B(x_1), y_2) & \to B((1_{rev}, 1_{rev})(x_1, y_2)) \\
\text{(vi)} : & (1_{rev}, 1_{rev})(E, y_2) & \to y_2
\end{array}
$$

and $e'_{rev;rev} = (rev, rev)(x_1, (1_{rev}, 1_{rev})(x_1, E))$. □

4.2 Eliminating Copy-states

Now, we turn to the problem of detecting so-called copy-states. We will follow the same course as in the last section, by defining a notion, which allows us to formally capture our concept of copy-states, developing a fixpoint construction to effectively calculate the

[3]This step is necessary, in order to guarantee that the resulting Macro Tree Transducer will have valid right-hand sides.

necessary information, and giving examples and an informal procedure, how to apply the obtained information to optimize a Macro Tree Transducer.

We say that a state is a copy-state, if for every tree, on which it is called, it will simply project on one—and always the same—of its context parameters. In order to describe this formally, we introduce the concept of a pr-function.

Definition 4.12 A *pr-function* (over a ranked alphabet Q of states) is any element of the indexed product $\mathcal{PR} = \prod_{q \in Q^{(r+1)}} ([r] \cup \{*\})$. □

Thus, a pr-function determines for every state q of rank $r+1$, either an element of $[r]$—intuitively meant to express, on which one of its context parameters the state projects—or the special symbol $*$—meaning that the state does not project on one of its context parameters. Just as with cd-functions, note that there are finitely many pr-functions, and that $\mathcal{PR}$ gives an effective representation for them.

We need to know the potential copying behavior of a state on all possible input trees. Therefore, we start by defining the pr-function of a particular input tree t.

Definition 4.13 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer and $t \in T_\Sigma$. The *pr-function of t for M*, denoted by $pr_M(t)$, is defined, for every $r \in \mathbf{N}$, $q \in Q^{(r+1)}$, by: $pr_M(t)(q) = (\text{if } \nu \in [r] \text{ then } \nu \text{ else } *)$, where $\nu = \tau_{M,q,[r]}(t, 1, \ldots, r)$. □

Note, that—using Lemma 2.16—from $pr_M(t)(q) = \nu \in [r]$ we can deduce that for every $\theta_1, \ldots, \theta_r \in T_\Delta$, we will have: $\tau_{M,q,\emptyset}(t, \theta_1, \ldots, \theta_r) = \theta_\nu$. Consequently, the above definition captures exactly the intended description of copying behavior.

Again, instead of depending on the definition above, we seek for a way to compute the pr-function of a tree in terms of the pr-functions of its immediate subtrees. We define a function $pr_{M,\sigma}$, for every $\sigma \in \Sigma$, as follows.

Definition 4.14 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer. For every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$, the function $pr_{M,\sigma} : (\mathcal{PR})^k \to \mathcal{PR}$ is defined as follows: for every $pr_1, \ldots, pr_k \in \mathcal{PR}$, $r \in \mathbf{N}$ and $q \in Q^{(r+1)}$, we have $pr_{M,\sigma}(pr_1, \ldots, pr_k)(q) = P_{M,k,r}(rhs_{q,\sigma}, pr_1, \ldots, pr_k)$, if R contains the rule $q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r) \to rhs_{q,\sigma}$, with $rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$. For every $k \in rank(\Sigma)$ and $r+1 \in rank(Q)$, the following function is defined.
$P_{M,k,r} : RHS(Q, \Delta, k, Y_r) \underbrace{\times \cdots \times \mathcal{PR}}_{k-times} \to ([r] \cup \{*\})$, given by:

- $P_{M,k,r}(y_h, pr_1, \ldots, pr_k) = h$,
  for $y_h \in Y_r$ and $pr_1, \ldots, pr_k \in \mathcal{PR}$
- $P_{M,k,r}(\delta(\phi_1, \ldots, \phi_a), pr_1, \ldots, pr_k) = *$,
  for $a \in \mathbf{N}$, $\delta \in \Delta^{(a)}$, $\phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$ and $pr_1, \ldots, pr_k \in \mathcal{PR}$
- $P_{M,k,r}(q(x_b, \phi_1, \ldots, \phi_a), pr_1, \ldots, pr_k) = (\text{if } pr_b(q) = * \text{ then } *$
  $\text{else } P_{M,k,r}(\phi_{pr_b(q)}, pr_1, \ldots, pr_k))$,
  for $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $x_b \in X_k$, $\phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$ and $pr_1, \ldots, pr_k \in \mathcal{PR}$

□

Here, the intuitive idea is that for some $q \in Q^{(r+1)}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T_\Sigma$, the normal form of $q(\sigma(t_1, \ldots, t_k), y_1, \ldots, y_r)$ will be obtained as normal form of $rhs_{q,\sigma}[x_b \leftarrow t_b, 1 \leq b \leq k]$. Thus, if this right-hand side is just constituted by a context parameter y_h, we will indeed have a projection on the h-th context parameter. If some output symbol is generated, there will obviously be no simple projection behavior, thus the answer is the special symbol $*$. Finally, if the right-hand side is some function call on some x_b, then the pr-function of t_b will tell us, whether a projection takes place, and if so, which context parameter position has to be considered further.

Example 4.15 Recall from Example 4.11 the Macro Tree Transducer $M_{rev;rev}$ after the simplification—by removing superfluous context parameters—and renaming of context variables to $M'_{rev;rev} = (S'_{rev;rev}, \Sigma_{mon}, \Sigma_{mon}, e'_{rev;rev}, R'_{rev;rev})$ with set of states $S'_{rev;rev} = \{(rev, rev)^{(2)}, (1_{rev}, 1_{rev})^{(2)}\}$, set of rules $R'_{rev;rev}$:

(i) :	$(rev, rev)(A(x_1), y_1)$	$\rightarrow (rev, rev)(x_1, y_1)$
(ii) :	$(rev, rev)(B(x_1), y_1)$	$\rightarrow (rev, rev)(x_1, y_1)$
(iii) :	$(rev, rev)(E, y_1)$	$\rightarrow y_1$
(iv) :	$(1_{rev}, 1_{rev})(A(x_1), y_1)$	$\rightarrow A((1_{rev}, 1_{rev})(x_1, y_1))$
(v) :	$(1_{rev}, 1_{rev})(B(x_1), y_1)$	$\rightarrow B((1_{rev}, 1_{rev})(x_1, y_1))$
(vi) :	$(1_{rev}, 1_{rev})(E, y_1)$	$\rightarrow y_1$

and $e'_{rev;rev} = (rev, rev)(x_1, (1_{rev}, 1_{rev})(x_1, E))$.

Let $pr_1 = [(rev, rev) \mapsto 1, (1_{rev}, 1_{rev}) \mapsto 1] \in \mathcal{PR}$ and calculate $pr_{M,A}(pr_1)$ as follows.
We have:

$$pr_{M,A}(pr_1)((rev, rev)) = P_{M,1,1}((rev, rev)(x_1, y_1), pr_1) = P_{M,1,1}(y_1, pr_1) = 1$$

and:

$$pr_{M,A}(pr_1)((1_{rev}, 1_{rev})) = P_{M,1,1}(A((1_{rev}, 1_{rev})(x_1, y_1)), pr_1) = *,$$

consequently: $pr_{M,A}(pr_1) = [(rev, rev) \mapsto 1, (1_{rev}, 1_{rev}) \mapsto *] \in \mathcal{PR}$. Similarly, we can show that $pr_{M,B}(pr_1) = pr_{M,A}(pr_{M,B}(pr_1)) = pr_{M,B}(pr_{M,B}(pr_1)) = [(rev, rev) \mapsto 1, (1_{rev}, 1_{rev}) \mapsto *]$. □

The intended recursive computation of pr-functions of trees with the help of the $pr_{M,\sigma}$-functions, is validated by the following lemma.

Lemma 4.16 Let $M = (Q, \Sigma, \Delta, e, R)$ be a Macro Tree Transducer. For every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T_\Sigma$, we have $pr_M(\sigma(t_1, \ldots, t_k)) = pr_{M,\sigma}(pr_M(t_1), \ldots, pr_M(t_k))$. □

Proof:

- As auxiliary proposition, we prove for every $k \in rank(\Sigma)$, $t_1, \ldots, t_k \in T_\Sigma$ and $r+1 \in rank(Q)$, by structural induction on $\phi \in RHS(Q, \Delta, k, Y_r)$, the following:

$$\begin{aligned}&(\text{if } \nu \in [r] \text{ then } \nu \text{ else } *), \text{ where } \nu = \tau_{M,k,r,[r]}(\phi, t_1, \ldots, t_k, 1, \ldots, r)\\ &= \\ &P_{M,k,r}(\phi, pr_M(t_1), \ldots, pr_M(t_k)).\end{aligned}$$

$\phi = \delta(\phi_1, \ldots, \phi_a), a \in \mathbf{N}, \delta \in \Delta^{(a)}, \phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$:

$$\begin{aligned}
&(\text{if } \nu \in [r] \text{ then } \nu \text{ else } *), \text{ where } \nu = \tau_{M,k,r,[r]}(\delta(\phi_1, \ldots, \phi_a), t_1, \ldots, t_k, 1, \ldots, r)\\
&\quad = (\text{by definition of } \tau_{M,k,r,[r]})\\
&(\text{if } \nu \in [r] \text{ then } \nu \text{ else } *), \text{ where } \nu = \delta(\tau_{M,k,r,[r]}(\phi_1, t_1, \ldots, t_k, 1, \ldots, r),\\
&\qquad\qquad \ldots,\\
&\qquad\qquad \tau_{M,k,r,[r]}(\phi_a, t_1, \ldots, t_k, 1, \ldots, r))\\
&\quad = (\text{by } \delta(\ldots) \notin [r])\\
&*\\
&\quad = (\text{by definition of } P_{M,k,r})\\
&P_{M,k,r}(\delta(\phi_1, \ldots, \phi_a), pr_M(t_1), \ldots, pr_M(t_k))
\end{aligned}$$

$\phi = y_h \in Y_r$:
(if $\nu \in [r]$ then ν else $*$), where $\nu = \tau_{M,k,r,[r]}(y_h, t_1, \ldots, t_k, 1, \ldots, r)$
= (by definition of $\tau_{M,k,r,[r]}$)
h
= (by definition of $P_{M,k,r}$)
$P_{M,k,r}(y_h, pr_M(t_1), \ldots, pr_M(t_k))$

$\phi = q(x_b, \phi_1, \ldots, \phi_a), a \in \mathbf{N}, q \in Q^{(a+1)}, x_b \in X_k, \phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$:
(if $\nu \in [r]$ then ν else $*$), where $\nu = \tau_{M,k,r,[r]}(q(x_b, \phi_1, \ldots, \phi_a), t_1, \ldots, t_k, 1, \ldots, r)$
= (by definition of $\tau_{M,k,r,[r]}$)
(if $\nu \in [r]$ then ν else $*$), where $\nu = \tau_{M,q,[r]}(t_b, \tau_{M,k,r,[r]}(\phi_1, t_1, \ldots, t_k, 1, \ldots, r),$
$\ldots,$
$\tau_{M,k,r,[r]}(\phi_a, t_1, \ldots, t_k, 1, \ldots, r))$
= (by Lemma 2.16)
(if $\nu \in [r]$ then ν else $*$),
where $\nu = (\tau_{M,q,[a]}(t_b, 1, \ldots, a))[l \leftarrow \tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r), 1 \le l \le a]$
= (by properties of substitution)
(if ($\mu \in [a]$ and $\tau_{M,k,r,[r]}(\phi_\mu, t_1, \ldots, t_k, 1, \ldots, r) \in [r]$) then ν else $*$),
where $\mu = \tau_{M,q,[a]}(t_b, 1, \ldots, a)$
$\nu = \mu[l \leftarrow \tau_{M,k,r,[r]}(\phi_l, t_1, \ldots, t_k, 1, \ldots, r), 1 \le l \le a]$
= (by propositional logic)
(if $\mu \notin [a]$ then $*$ else (if $\nu \in [r]$ then ν else $*$)),
where $\mu = \tau_{M,q,[a]}(t_b, 1, \ldots, a)$
$\nu = \tau_{M,k,r,[r]}(\phi_\mu, t_1, \ldots, t_k, 1, \ldots, r)$
= (by induction hypothesis for ϕ_μ)
(if $\mu \notin [a]$ then $*$ else $P_{M,k,r}(\phi_\mu, pr_M(t_1), \ldots, pr_M(t_k))$),
where $\mu = \tau_{M,q,[a]}(t_b, 1, \ldots, a)$
= (by: $\mu \notin [a]$ iff (if $\mu \in [a]$ then μ else $*$) $= *$)
(if $\eta = *$ then $*$ else $P_{M,k,r}(\phi_\mu, pr_M(t_1), \ldots, pr_M(t_k))$),
where $\mu = \tau_{M,q,[a]}(t_b, 1, \ldots, a)$
$\eta =$ (if $\mu \in [a]$ then μ else $*$)
= (by: $\eta = \mu$ if $\eta \neq *$)
(if $\eta = *$ then $*$ else $P_{M,k,r}(\phi_\eta, pr_M(t_1), \ldots, pr_M(t_k))$),
where $\mu = \tau_{M,q,[a]}(t_b, 1, \ldots, a)$
$\eta =$ (if $\mu \in [a]$ then μ else $*$)
= (by definition of $pr_M(t_b)$)
(if $\eta = *$ then $*$ else $P_{M,k,r}(\phi_\eta, pr_M(t_1), \ldots, pr_M(t_k))$),
where $\eta = pr_M(t_b)(q)$
= (by definition of $P_{M,k,r}$)
$P_{M,k,r}(q(x_b, \phi_1, \ldots, \phi_a), pr_M(t_1), \ldots, pr_M(t_k))$

- Now, we show for every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T_\Sigma$, that
$$pr_M(\sigma(t_1, \ldots, t_k)) = pr_{M,\sigma}(pr_M(t_1), \ldots, pr_M(t_k)),$$
by proving that for every $r \in \mathbf{N}$ and $q \in Q^{(r+1)}$, the following holds (using the auxiliary proposition, which was proven in the first point):

$$\begin{aligned}
& pr_M(\sigma(t_1,\ldots,t_k))(q) \\
& \quad = \text{ (by definition of } pr_M(\sigma(t_1,\ldots,t_k))) \\
& (\text{if } \nu \in [r] \text{ then } \nu \text{ else } *), \text{ where } \nu = \tau_{M,q,[r]}(\sigma(t_1,\ldots,t_k),1,\ldots,r) \\
& \quad = \text{ (by definition of } \tau_{M,q,[r]}) \\
& (\text{if } \nu \in [r] \text{ then } \nu \text{ else } *), \text{ where } \nu = \tau_{M,k,r,[r]}(rhs_{q,\sigma},t_1,\ldots,t_k,1,\ldots,r) \\
& \quad = \text{ (by the auxiliary proposition shown above, with } \phi = rhs_{q,\sigma}) \\
& P_{M,k,r}(rhs_{q,\sigma},pr_M(t_1),\ldots,pr_M(t_k)) \\
& \quad = \text{ (by definition of } pr_{M,\sigma}) \\
& pr_{M,\sigma}(pr_M(t_1),\ldots,pr_M(t_k))(q)
\end{aligned}$$

□

In order to detect copy-states, we need to abstract from the infinite set of input trees to the finite set of their pr-functions. For a given Macro Tree Transducer, we can do this by using the $pr_{M,\sigma}$-functions in a fixpoint construction, analogous to that used in the last section for computing all possibly occurring cd-functions.

Construction 4.17 Let $M = (Q,\Sigma,\Delta,e,R)$ be a Macro Tree Transducer. Let $S' = \mathcal{P}(\mathcal{PR})$. Let the function $\Psi : S' \to S'$ be defined by: $\Psi(s) = \{pr_{M,\sigma}(pr_1,\ldots,pr_k) \mid \sigma \in \Sigma^{(k)}, pr_1,\ldots,pr_k \in s\}$, for every $s \in S'$.
Let Ψ^* be the least fixpoint reached after finitely many iterations of Ψ, starting with the empty set. We claim: $\Psi^* = \{pr_M(t) \mid t \in T_\Sigma\}$. □

Proof: Analogous to the proof for Φ^* from Construction 4.8. □

Example 4.18 Let M be the Macro Tree Transducer $M'_{rev;rev}$ as above in Example 4.15. We perform the following iteration (referring to the pr-function pr_1 and calculation steps from Example 4.15):

$$\begin{aligned}
\Psi^0 &= \emptyset \\
\Psi^1 &= \{pr_{M,E}\} = \{[(rev,rev) \mapsto P_{M,0,1}(y_1),(1_{rev},1_{rev}) \mapsto P_{M,0,1}(y_1)]\} = \{pr_1\} \\
\Psi^2 &= \{pr_{M,E}, pr_{M,A}(pr_{M,E}), pr_{M,B}(pr_{M,E})\} \\
&= \{[(rev,rev) \mapsto 1,(1_{rev},1_{rev}) \mapsto 1],[(rev,rev) \mapsto 1,(1_{rev},1_{rev}) \mapsto *]\} \\
\Psi^3 &= \Psi^2
\end{aligned}$$

Thus, $\Psi^* = \{[(rev,rev) \mapsto 1,(1_{rev},1_{rev}) \mapsto 1],[(rev,rev) \mapsto 1,(1_{rev},1_{rev}) \mapsto *]\}$. □

The construction on $M'_{rev;rev}$ shows us that the state (rev,rev) always projects on its only context parameter, whereas nothing like that can be said about $(1_{rev},1_{rev})$. This information will allow us, to further simplify the Macro Tree Transducer. But first, we formalize our intuition.

Definition 4.19 Let $M = (Q,\Sigma,\Delta,e,R)$ be a Macro Tree Transducer. A state $q \in Q^{(r+1)}$ (with $r \in \mathbf{N}$) is called a *copy-state, projecting on its* ν*th context parameter,* (with $\nu \in [r]$), iff for every pr-function $pr \in \Psi^*$, we have: $pr(q) = \nu$. □

Since a copy-state never performs any useful computation, we can always simplify a Macro Tree Transducer as follows. First, we remove every state q of which we know that it always projects on one and the same of its context parameters, together with all of its rules, from the Macro Tree Transducer. Then, in every occurrence of such a state q—in right-hand sides as well as in the initial expression—, we perform the corresponding projection.

Example 4.20 Given $M'_{rev;rev}$, we use the information that (rev,rev) always projects on its only context parameter, to obtain the further simplified Macro Tree Transducer $M''_{rev;rev} = (S''_{rev;rev},\Sigma_{mon},\Sigma_{mon},e''_{rev;rev},R''_{rev;rev})$ with $S''_{rev;rev} = \{(1_{rev},1_{rev})^{(2)}\}$, set of rules $R''_{rev;rev}$:

$$\begin{array}{lll} (i) & : (1_{rev}, 1_{rev})(A(x_1), y_1) & \to A((1_{rev}, 1_{rev})(x_1, y_1)) \\ (ii) & : (1_{rev}, 1_{rev})(B(x_1), y_1) & \to B((1_{rev}, 1_{rev})(x_1, y_1)) \\ (iii) & : (1_{rev}, 1_{rev})(E, y_1) & \to y_1 \end{array}$$

and $e''_{rev;rev} = (1_{rev}, 1_{rev})(x_1, E)$. □

Note, that $M''_{rev;rev}$ indeed corresponds to the optimized Macro Tree Transducer for computing the identity function, which we promised in Example 4.1.

Now, we want to give an example, which shows that the construction proposed here is more powerful than a naive version of copy-state elimination we previously implemented in the Haskell$^+$ system[4].

Example 4.21 For $Q = \{q^{(2)}, p^{(3)}\}$ and $e = q(x_1, zero)$, let be given the Macro Tree Transducer $M_{tricky} = (Q, Nat, Nat, e, R_{tricky})$ with set of rules R_{tricky}:

$$\begin{array}{lll} (i) & : q(succ(x_1), y_1) & \to p(x_1, p(x_1, y_1, zero), p(x_1, zero, y_1)) \\ (ii) & : q(zero, y_1) & \to y_1 \\ (iii) & : p(succ(x_1), y_1, y_2) & \to p(x_1, y_2, y_1) \\ (iv) & : p(zero, y_1, y_2) & \to y_1 \end{array}$$

We perform the following iteration:

$$\begin{array}{l} \Psi^0 = \emptyset \\ \Psi^1 = \{pr_{M,zero}\} = \{[q \mapsto 1, p \mapsto 1]\} \\ \Psi^2 = \{pr_{M,zero}, pr_{M,succ}(pr_{M,zero})\} = \{[q \mapsto 1, p \mapsto 1], [q \mapsto 1, p \mapsto 2]\} \\ \Psi^3 = \Psi^2 \cup \{pr_{M,succ}([q \mapsto 1, p \mapsto 2])\} \\ \quad = \Psi^2 \cup \{[q \mapsto P_{M,1,1}(p(x_1, p(x_1, y_1, zero), p(x_1, zero, y_1)), [q \mapsto 1, p \mapsto 2]), \\ \qquad\qquad p \mapsto P_{M,1,2}(p(x_1, y_2, y_1), [q \mapsto 1, p \mapsto 2])]\} \\ \quad = \Psi^2 \cup \{[q \mapsto P_{M,1,1}(p(x_1, zero, y_1), [q \mapsto 1, p \mapsto 2]), \\ \qquad\qquad p \mapsto P_{M,1,2}(y_1, [q \mapsto 1, p \mapsto 2])]\} \\ \quad = \Psi^2 \cup \{[q \mapsto P_{M,1,1}(y_1, [q \mapsto 1, p \mapsto 2]), p \mapsto 1]\} \\ \quad = \Psi^2 \cup \{[q \mapsto 1, p \mapsto 1]\} = \Psi^2 \end{array}$$

Thus, $\Psi^* = \{[q \mapsto 1, p \mapsto 1], [q \mapsto 1, p \mapsto 2]\}$. □

In this example, our new construction allows us to conclude that q will always project on its only context parameter, whereas the version implemented in the Haskell$^+$ system would not recognize this. However, the construction is meant mainly for post-processing Macro Tree Transducers, in order to eliminate copy-states that were introduced by composition transformations, and it is not obvious, whether such tricky patterns as in the previous example may occur in this setting.

4.3 Turning non-terminating Programs into terminating ones

In this section, we want to point out a potential problem of using the suggested copy elimination construction—presented in the last section—for the optimization of functional programs. Specifically, it occurs with lazy functional languages, where we can define infinite data structures. Such an infinite data structure might result from a circular definition like $list = A(B(list))$ over the ranked alphabet Σ_{mon}[5]. Now, consider the Macro

[4] The naive version implemented earlier does only consider situations, where the copying behavior does not involve more than one state.

[5] This corresponds to an infinite list $A(B(A(B(A(\cdots)))))$.

Tree Transducer M_{rev} from Example 2.4 and another function $head$, partially defined by the following rules:

(i) : $head(A(x_1)) \rightarrow \alpha$
(ii) : $head(B(x_1)) \rightarrow \beta$

If we now use a lazy reduction strategy to compute the expression $head(rev(rev(list, E), E))$, we obtain the reduction sequence shown in Figure 4.1.

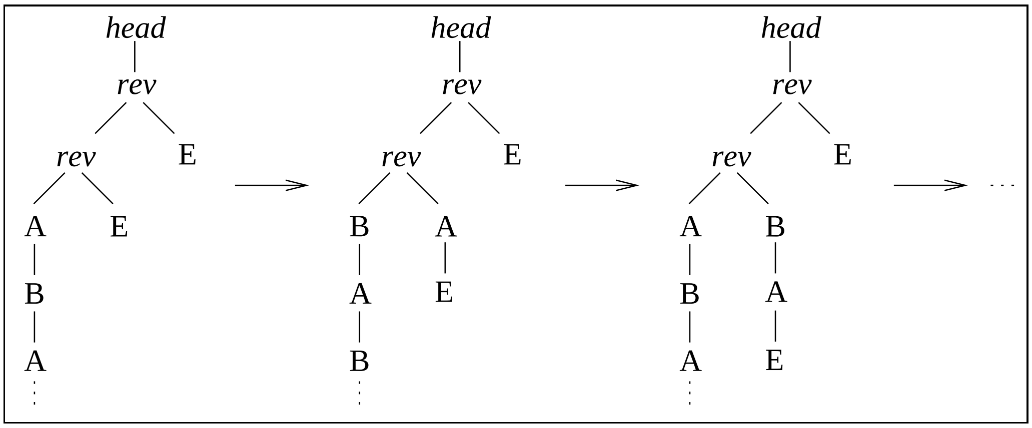

Figure 4.1: A non-terminating program

Note, that this reduction sequence is obviously non-terminating. In the expression, which we are trying to evaluate, there occurs a composition of two Macro Tree Transducers, namely the subterm $rev(rev(list, E), E)$ corresponds to the twofold application of M_{rev} on the input tree $list$. The composition construction presented in Chapter 3 together with the post-processing optimizations presented in this chapter, tell us that the composition of M_{rev} with itself can also be realized by the Macro Tree Transducer $M''_{rev;rev}$ given in Example 4.20. As new expression to be evaluated, an optimizer would thus obtain: $head((1_{rev}, 1_{rev})(list, E))$. Reducing this expression leads to the sequence shown in Figure 4.2.

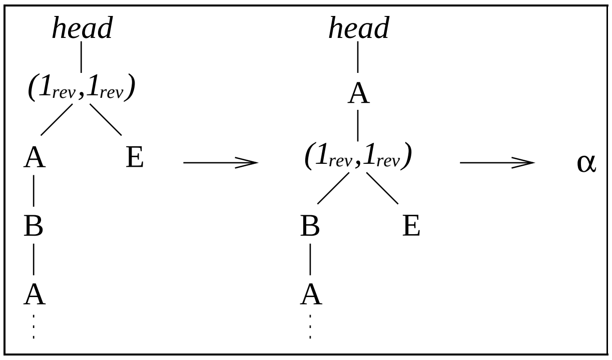

Figure 4.2: A terminating program

But, this means that we have turned a non-terminating program into a terminating one. Before considering the implications, this behavior has for a program transformation system, we want to point out, where the "strange" thing takes place. First, note that the problem does not occur in the composition construction itself. The Macro Tree Transducer $M_{rev;rev}$ constructed in Example 3.4 is really equivalent to the composition of M_{rev} with itself, which means that replacing the subterm $rev(rev(list, E), E)$ by the term $(rev, rev)(list, (1_{rev}, 1_{rev})(list, E, E), E)$ in the above expression would still lead to an infinite reduction sequence. The same is true for $M'_{rev;rev}$ from Example 4.11, obtained by eliminating superfluous context parameters, as shown in Figure 4.3.

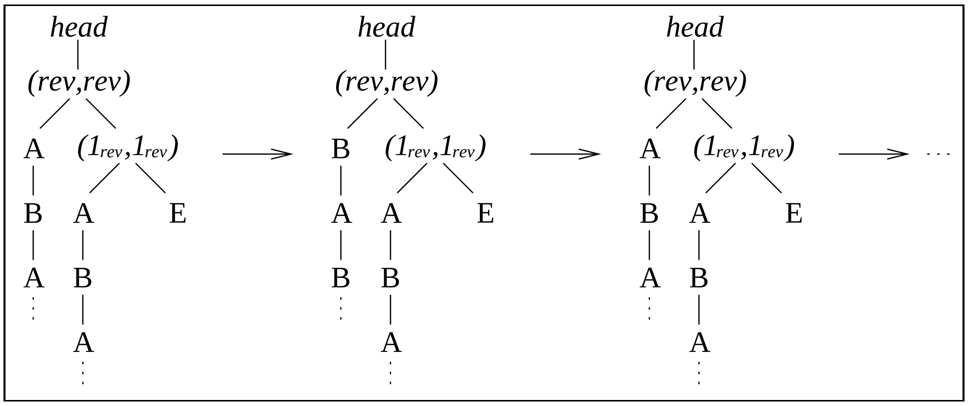

Figure 4.3: The intermediate program

The crucial point, where we suddenly get a terminating program, is the elimination of copy-states. The information that the state (rev, rev) always projects on its context parameter (which was used to obtain $M''_{rev;rev}$) is simply not true for the infinite tree $list$. This, however, does not mean that the proofs in the last section are wrong. They just do not apply here, because they prove statements about finite trees over ranked alphabets by structural induction. In particular, our fixpoint construction for Ψ^* (Construction 4.17) only computes the pr-functions of all *finite* trees. If only those are allowed as input trees, the transformation of eliminating copy-states in fact yields a program, which is equivalent to the original one.

But, since we are motivating our transformations as optimizations for lazy functional languages with the possibility of using infinite data structures, we have to answer the question, how to cope with the change of termination behavior by our program transformations. First, the good news: it cannot happen that a terminating program is turned into a non-terminating one. Thus, we might ignore the problem or even be happy that the termination behavior can be improved in some cases. But, this is not wholly satisfactory, because an important advantage of functional programming is the ease, with which we can apply equational reasoning for proving properties of programs. If we allow transformations that change the termination behavior and thus the semantics of programs, we are in danger of losing this advantage. We suggest the following pragmatic approach.

If transformations, which might turn non-terminating programs into terminating ones, are integrated into a language system—e.g., a compiler—the user should be given the possibility to control the application of these transformations. During development and testing of programs, these transformations should be suppressed, e.g., by a compiler flag. This might lead to performance losses, because potential optimizations are omitted, but in this phase the preservation of the termination behavior of programs is more important, because a non-termination usually indicates an algorithmic mistake, which should not be "hidden" by the compiler. When, however, the development of a program is finished and a final version is compiled (production run), all transformations should be enabled, taking advantage of all possible optimizations. In this phase, there should be no danger of changing the semantics of the program, because the developer has a strong responsibility—if not a proof obligation—to guarantee that the program indeed *is* terminating.

Chapter 5

Useful Things

This chapter is divided into three sections. The first uses Theorem 3.1 to show that for every non-copying Macro Tree Transducer, the size of the output tree associated to an input tree is exponentially bounded by the input tree's height. The second section shows, how Construction 3.2 can be applied to obtain, for every non-copying Macro Tree Transducer M_{nc}, an effective representation of the function, which computes the number of reduction steps that M_{nc} performs depending on its input tree. Finally, we will show an example application of an implementation of Construction 3.2 in the Haskell^{+} system.

5.1 A Size-Height Property for non-copying Macro Tree Transducers

In Section 3.3 we quoted the classical result that for every tree transduction induced by a Macro Tree Transducer, the height of the output tree is exponentially bounded by the height of the input tree. Lemmata of the same style—relating input- and output-height or -size—are available for other classes of Macro Tree Transductions (and will be summarized in a table below), but not yet for the class of non-copying Macro Tree Transductions. It would be useful to know such a lemma, because—as demonstrated in the proof of Theorem 3.7—these kinds of results can help in reasoning about the expressive power of various classes of tree transductions.

By using our construction for $MAC_{nc}; MAC_{wsu} \subseteq MAC$, we are going to show that for non-copying Macro Tree Transducers, the output-*size* is exponentially bounded by the input-*height*. For doing so, we will construct, for every ranked alphabet Δ, a weakly single-use Macro Tree Transducer S_Δ, which computes the size of trees over Δ as monadic natural numbers over the ranked alphabet Nat.

Construction 5.1 Let Δ be a ranked alphabet. We construct the Macro Tree Transducer $S_\Delta = (\{s^{(2)}\}, \Delta, Nat, s(x_1, zero), R_\Delta)$, where for every $k \in \mathbf{N}$, $\delta \in \Delta^{(k)}$, the set R_Δ contains the following rule:

$$s(\delta(x_1, \ldots, x_k), y_1) \to succ(\zeta_k(k, y_1)).$$

For $k \in \mathbf{N}$, the function $\zeta_k : (\{0\} \cup [k]) \times RHS(\{s^{(2)}\}, Nat, k, Y_1) \to RHS(\{s^{(2)}\}, Nat, k, Y_1)$ is defined as follows:

$$\begin{aligned} \zeta_k(0, \phi) &= \phi \\ \zeta_k(n, \phi) &= \zeta_k(n-1, s(x_n, \phi)), \qquad \text{for } 0 < n \le k \end{aligned}$$

We claim that S_Δ is weakly single-use (it is also non-copying and thus single-use, but we do not need this fact here) and for every $t \in T_\Delta$, we have: $height(\tau(S_\Delta)(t)) = size(t)$.

□

Proof:

- For proving S_Δ to be weakly single-use, we want to use Proposition 2.14 and first show for every $k \in \mathbf{N}$, $\delta \in \Delta^{(k)}$ and $i \in [k]$, by induction on $0 \leq n \leq k$, that the following holds: for every $\phi \in RHS(\{s^{(2)}\}, Nat, k, Y_1)$, we have:
  $\#_{x_i}(\zeta_k(n, \phi)) =$ (if $i \leq n$ then 1 else 0) $+ \#_{x_i}(\phi)$.

  $n = 0$:
  $\#_{x_i}(\zeta_k(0, \phi))$
  $=$ (by definition of ζ_k)
  $\#_{x_i}(\phi)$
  $=$ (by arithmetics)
  (if $i \leq 0$ then 1 else 0) $+ \#_{x_i}(\phi)$

  $0 < n \leq k$:
  $\#_{x_i}(\zeta_k(n, \phi))$
  $=$ (by definition of ζ_k)
  $\#_{x_i}(\zeta_k(n-1, s(x_n, \phi)))$
  $=$ (by induction hypothesis for $n-1$)
  (if $i \leq n-1$ then 1 else 0) $+ \#_{x_i}(s(x_n, \phi))$
  $=$ (by definition of $\#_{x_i}$)
  (if $i \leq n-1$ then 1 else 0) $+ \#_{x_i}(x_n) + \#_{x_i}(\phi)$
  $=$ (by definition of $\#_{x_i}$)
  (if $i \leq n-1$ then 1 else 0) $+$ (if $i = n$ then 1 else 0) $+ \#_{x_i}(\phi)$
  $=$ (by arithmetics)
  (if $i \leq n$ then 1 else 0) $+ \#_{x_i}(\phi)$

  For $n = k$ and $\phi = y_1$, it follows that:
  $\#_{x_i}(rhs_{s,\delta})$
  $=$ (by definition of S_Δ)
  $\#_{x_i}(succ(\zeta_k(k, y_1)))$
  $=$ (by definition of $\#_{x_i}$)
  $\#_{x_i}(\zeta_k(k, y_1))$
  $=$ (by the proposition shown above)
  (if $i \leq k$ then 1 else 0) $+ \#_{x_i}(y_1)$
  $=$ (by $i \in [k]$ and definition of $\#_{x_i}$)
  1

- For proving $height(\tau(S_\Delta)(t)) = size(t)$, for every $t \in T_\Delta$, we show that the following two statements hold by simultaneous induction.

  (I) For $\mathbf{t} \in \mathbf{T_\Delta}$: for every $\theta \in T_{Nat}$ holds:
  $height(\tau_{S_\Delta,s,\emptyset}(t, \theta)) = size(t) + height(\theta)$

  (II) For $\mathbf{k} \in \mathbf{N}, \mathbf{t_1}, \ldots, \mathbf{t_k} \in \mathbf{T_\Delta}$: for every $\theta \in T_{Nat}$ holds:
  $height(\tau_{S_\Delta,k,1,\emptyset}(\zeta_k(k, y_1), t_1, \ldots, t_k, \theta)) = \sum_{i \in [k]} size(t_i) + height(\theta)$

(I) ⇐ (II) : For $t = \delta(t_1, \ldots, t_k)$ with $k \in \mathbf{N}$, $\delta \in \Delta^{(k)}$ and $t_1, \ldots, t_k \in T_\Delta$:

$$
\begin{aligned}
& height(\tau_{S_\Delta,s,\emptyset}(\delta(t_1, \ldots, t_k), \theta)) \\
& \quad = \text{(by definition of } \tau_{S_\Delta,s,\emptyset}) \\
& height(\tau_{S_\Delta,k,1,\emptyset}(rhs_{s,\delta}, t_1, \ldots, t_k, \theta)) \\
& \quad = \text{(by definition of } S_\Delta) \\
& height(\tau_{S_\Delta,k,1,\emptyset}(succ(\zeta_k(k, y_1)), t_1, \ldots, t_k, \theta)) \\
& \quad = \text{(by definition of } \tau_{S_\Delta,k,1,\emptyset}) \\
& height(succ(\tau_{S_\Delta,k,1,\emptyset}(\zeta_k(k, y_1), t_1, \ldots, t_k, \theta))) \\
& \quad = \text{(by definition of } height) \\
& 1 + height(\tau_{S_\Delta,k,1,\emptyset}(\zeta_k(k, y_1), t_1, \ldots, t_k, \theta)) \\
& \quad = \text{(by induction hypothesis } \textbf{(II)} \text{ for } k \text{ and } t_1, \ldots, t_k) \\
& 1 + \sum_{i \in [k]} size(t_i) + height(\theta) \\
& \quad = \text{(by definition of } size) \\
& size(\delta(t_1, \ldots, t_k)) + height(\theta)
\end{aligned}
$$

(II) ⇐ (I) : We show by induction on $0 \leq n \leq k$ that the following holds:
For every $\phi \in RHS(\{s^{(2)}\}, Nat, k, Y_1)$, we have:

$$height(\tau_{S_\Delta,k,1,\emptyset}(\zeta_k(n, \phi), t_1, \ldots, t_k, \theta)) = \sum_{i \in [n]} size(t_i) + height(\tau_{S_\Delta,k,1,\emptyset}(\phi, t_1, \ldots, t_k, \theta)).$$

$n = 0$:

$$
\begin{aligned}
& height(\tau_{S_\Delta,k,1,\emptyset}(\zeta_k(0, \phi), t_1, \ldots, t_k, \theta)) \\
& \quad = \text{(by definition of } \zeta_k) \\
& height(\tau_{S_\Delta,k,1,\emptyset}(\phi, t_1, \ldots, t_k, \theta)) \\
& \quad = \text{(by arithmetics)} \\
& \sum_{i \in [0]} size(t_i) + height(\tau_{S_\Delta,k,1,\emptyset}(\phi, t_1, \ldots, t_k, \theta))
\end{aligned}
$$

$0 < n \leq k$:

$$
\begin{aligned}
& height(\tau_{S_\Delta,k,1,\emptyset}(\zeta_k(n, \phi), t_1, \ldots, t_k, \theta)) \\
& \quad = \text{(by definition of } \zeta_k) \\
& height(\tau_{S_\Delta,k,1,\emptyset}(\zeta_k(n-1, s(x_n, \phi)), t_1, \ldots, t_k, \theta)) \\
& \quad = \text{(by induction hypothesis for } n-1) \\
& \sum_{i \in [n-1]} size(t_i) + height(\tau_{S_\Delta,k,1,\emptyset}(s(x_n, \phi), t_1, \ldots, t_k, \theta)) \\
& \quad = \text{(by definition of } \tau_{S_\Delta,k,1,\emptyset}) \\
& \sum_{i \in [n-1]} size(t_i) + height(\tau_{S_\Delta,s,\emptyset}(t_n, \tau_{S_\Delta,k,1,\emptyset}(\phi, t_1, \ldots, t_k, \theta))) \\
& \quad = \text{(by induction hypothesis } \textbf{(I)} \text{ for } t_n) \\
& \sum_{i \in [n-1]} size(t_i) + size(t_n) + height(\tau_{S_\Delta,k,1,\emptyset}(\phi, t_1, \ldots, t_k, \theta)) \\
& \quad = \text{(by arithmetics)} \\
& \sum_{i \in [n]} size(t_i) + height(\tau_{S_\Delta,k,1,\emptyset}(\phi, t_1, \ldots, t_k, \theta))
\end{aligned}
$$

For $n = k$ and $\phi = y_1$, it follows that:

$$
\begin{aligned}
& height(\tau_{S_\Delta,k,1,\emptyset}(\zeta_k(k, y_1), t_1, \ldots, t_k, \theta)) = \\
& \sum_{i \in [k]} size(t_i) + height(\tau_{S_\Delta,k,1,\emptyset}(y_1, t_1, \ldots, t_k, \theta)) = \\
& \sum_{i \in [k]} size(t_i) + height(\theta)
\end{aligned}
$$

Finally, from **(I)**, we get for every $t \in T_\Delta$:

$$\begin{aligned}
& height(\tau(S_\Delta)(t)) \\
& \quad = \text{(by definition of } \tau(S_\Delta)) \\
& height(\tau_{S_\Delta,1,0,\emptyset}(s(x_1, zero), t)) \\
& \quad = \text{(by definition of } \tau_{S_\Delta,1,0,\emptyset}) \\
& height(\tau_{S_\Delta,s,\emptyset}(t, zero)) \\
& \quad = \text{(by statement } \mathbf{(I)}) \\
& size(t) + height(zero) \\
& \quad = \text{(by definition of } height) \\
& size(t)
\end{aligned}$$

□

Example 5.2 For the ranked alphabet Σ_{bin}, we obtain the Macro Tree Transducer $S_{\Sigma_{bin}}$, used in Example 3.5. □

Now, we can give our size-height property for non-copying Macro Tree Transducers.

Lemma 5.3 Let $M_{nc} = (Q, \Sigma, \Delta, e, R)$ be a non-copying Macro Tree Transducer. There exists a constant $c \in \mathbf{N}$, such that for every input tree $t \in T_\Sigma$ holds:
$size(\tau(M_{nc})(t)) \leq c^{height(t)}$. □

Proof: Since M_{nc} is non-copying and S_Δ is weakly single-use, there exists—by Theorem 3.1, respectively, Construction 3.2—a Macro Tree Transducer M, such that we have $\tau(M_{nc}); \tau(S_\Delta) = \tau(M) : T_\Sigma \to Nat$. For M we know, by Lemma 3.6, that there exists a constant $c \in \mathbf{N}$, such that for every $t \in T_\Sigma$ holds:

$$height(\tau(M)(t)) \leq c^{height(t)}.$$

This gives us, by $\tau(M)(t) = \tau(S_\Delta)(\tau(M_{nc})(t))$:

$$height(\tau(S_\Delta)(\tau(M_{nc})(t))) \leq c^{height(t)}.$$

And, by $height(\tau(S_\Delta)(s)) = size(s)$ for every $s \in T_\Delta$:

$$size(\tau(M_{nc})(t)) \leq c^{height(t)}.$$

□

Thus, we get the following summary of input-output boundedness for classes of Macro Tree Transductions:

MAC	: output-**height**	**exponentially**	bounded by input-**height**
MAC_{nc}	: output-**size**	**exponentially**	bounded by input-**height**
MAC_{wsu}	: output-**height**	**linearly**	bounded by input-**size**
MAC_{su}	: output-**size**	**linearly**	bounded by input-**size**
TOP	: output-**height**	**linearly**	bounded by input-**height**

The results for TOP and MAC are known from [EV85]. The bound for MAC_{wsu} follows from Theorem 7.1 of [Küh98] and the corresponding boundedness property for Attributed Tree Transducers [Fül81]. The result for MAC_{su} is a consequence of Theorem 7.2 of [Küh98] and a boundedness property for single-use Attributed Tree Transducers (cf. [Gie88, Küh97]). Finally, the bound for non-copying Macro Tree Transducers has been proven in this section.

5.2 Counting Reduction Steps of non-copying Macro Tree Transducers

A recurring task is to calculate, for a given Macro Tree Transducer, the number of reduction steps that this transducer performs on a particular input. This is needed, e.g., when giving examples such as in the Introduction (see Figures 1.2 and 1.3), but also, when we want to formally compare two Macro Tree Transducers with respect to their efficiency (e.g., because the second one was obtained from the first one by applying some program transformation). The typical approach is to count the reduction steps "by hand" for some small input trees, then to conjecture some law or arithmetic expression describing the number of reduction steps in dependence on the given input, and then to try to prove this conjecture for all input trees by structural induction. It would be nice to have a construction, which gives us, for some Macro Tree Transducer, an effective representation of the function that describes the dependency of number of reduction steps on the input tree. In this section, we are going to give a partial solution to this problem. Namely, we will present a construction, which works for every non-copying Macro Tree Transducer and represents the function under question not as an arithmetic expression, but as a Macro Tree Transducer with output ranked alphabet Nat.

The basic idea is almost trivial. In order to count the reduction steps, we will output a special symbol *tick* with every rule application. That is, the output tree for a given input will look as before, except that it contains additional occurrences of the *tick* symbol. We will illustrate this with an example.

Example 5.4

- Let be given $Q_{list} = \{list^{(1)}, exp^{(2)}\}$, $\Delta_{list} = \{:^{(2)}, []^{(0)}, succ^{(1)}, zero^{(0)}\}$ and the non-copying Macro Tree Transducer $M_{list} = (Q_{list}, Nat, \Delta_{list}, list(x_1), R_{list})$ with set of rules R_{list}:

$$\begin{array}{lll}
\text{(i)} : list(succ(x_1)) & \rightarrow & :(exp(x_1, zero), list(x_1)) \\
\text{(ii)} : list(zero) & \rightarrow & [] \\
\text{(iii)} : exp(succ(x_1), y_1) & \rightarrow & exp(x_1, exp(x_1, y_1)) \\
\text{(iv)} : exp(zero, y_1) & \rightarrow & succ(y_1)
\end{array}$$

- By adding a *tick* on every right-hand side, we get the non-copying Macro Tree Transducer $M'_{list} = (Q_{list}, Nat, \Delta'_{list}, list(x_1), R'_{list})$ with $\Delta'_{list} = \Delta_{list} \cup \{tick^{(1)}\}$ and set of rules R'_{list}:

$$\begin{array}{lll}
\text{(i)} : list(succ(x_1)) & \rightarrow & tick(:(exp(x_1, zero), list(x_1))) \\
\text{(ii)} : list(zero) & \rightarrow & tick([]) \\
\text{(iii)} : exp(succ(x_1), y_1) & \rightarrow & tick(exp(x_1, exp(x_1, y_1))) \\
\text{(iv)} : exp(zero, y_1) & \rightarrow & tick(succ(y_1))
\end{array}$$

□

The number of performed reduction steps can then easily be obtained by counting the occurrences of *tick* in the output tree (see Figure 5.1). Note, that the number we obtain thus is the number of reduction steps with lazy evaluation, because reductions that result from function calls in parameter positions, which are then discarded, will not be counted, as the *tick*'s resulting from these computations will never occur in the output tree. Also,

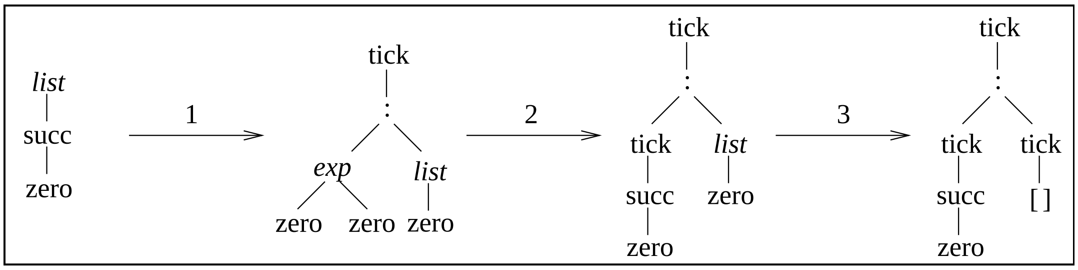

Figure 5.1: Counting reduction steps of M'_{list} by "ticking"

we do not need to consider sharing here, because in non-copying Macro Tree Transducers parameters cannot be copied[1].

Of course, we are not satisfied with counting the number of $tick$-symbols in the output tree "by hand". Instead, this counting will be done by a second Macro Tree Transducer, which outputs the number of $tick$-symbols as natural number over the monadic ranked alphabet Nat. For a ranked alphabet Δ, this Macro Tree Transducer is constructed similarly to S_Δ in the last section.

Construction 5.5 Let Δ be a ranked alphabet with $tick^{(1)} \in \Delta$. We construct the Macro Tree Transducer $C_\Delta = (\{s^{(2)}\}, \Delta, Nat, s(x_1, zero), R_\Delta)$, where R_Δ contains the rule:

$$s(tick(x_1), y_1) \to succ(s(x_1, y_1)),$$

and for every $k \in \mathbf{N}$, $\delta \in \Delta^{(k)}$ with $\delta \neq tick$, the set R_Δ contains the following rule:

$$s(\delta(x_1, \ldots, x_k), y_1) \to \zeta_k(k, y_1).$$

The functions ζ_k for $k \in \mathbf{N}$ are defined as in Construction 5.1. We claim that C_Δ is weakly single-use and for every $t \in T_\Delta$, we have: $height(\tau(C_\Delta)(t)) = \#_{tick}(t)$. □

The proof of our claim would be very similar to the proof of Construction 5.1 for S_Δ. We omit it here.

Example 5.6 For the extended ranked alphabet Δ'_{list}, we construct the weakly single-use Macro Tree Transducer $C_{\Delta'_{list}} = (\{s^{(2)}\}, \Delta'_{list}, Nat, s(x_1, zero), R_{\Delta'_{list}})$ with set of rules $R_{\Delta'_{list}}$:

$$\begin{array}{lll}
\text{(i)} : & s(tick(x_1), y_1) & \to succ(s(x_1, y_1)) \\
\text{(ii)} : & s(:(x_1, x_2), y_1) & \to s(x_1, s(x_2, y_1)) \\
\text{(iii)} : & s([\,], y_1) & \to y_1 \\
\text{(iv)} : & s(succ(x_1), y_1) & \to s(x_1, y_1) \\
\text{(v)} : & s(zero, y_1) & \to y_1
\end{array}$$

□

Since the adding of $tick$-symbols on the right-hand sides does not change the property of a Macro Tree Transducer to be non-copying, and since C_Δ is always weakly single-use, we can apply the composition construction presented in Chapter 3, in order to obtain a single Macro Tree Transducer, which directly computes the number of reduction steps.

Construction 5.7 Let $M_{nc} = (Q, \Sigma, \Delta, e, R)$ be a non-copying Macro Tree Transducer with $tick \notin \Delta$. Construct the—trivially—non-copying Macro Tree Transducer $M'_{nc} = (Q, \Sigma, \Delta', e, R')$ with $\Delta' = \Delta \cup \{tick^{(1)}\}$, where for every $k, r \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $q \in Q^{(r+1)}$, R' contains the rule $q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_r) \to tick(rhs)$, if R contains

[1]There is no difference between the call-by-name and call-by-need evaluation strategies in this situation.

the rule $q(\sigma(x_1,\ldots,x_k),y_1,\ldots,y_r) \to rhs$ with $rhs \in RHS(Q,\Delta,k,Y_r)$.
Now, construct the weakly single-use Macro Tree Transducer $C_{\Delta'}$. The Macro Tree Transducer $C_{M_{nc}}$ is obtained by composing M'_{nc} and $C_{\Delta'}$ with Construction 3.2. $\square$

As an example, we will perform the construction for the Macro Tree Transducer M_{list}. This will also serve as an additional illustration of Construction 3.2 and the post-processing steps presented in Chapter 4.

Example 5.8

- The Macro Tree Transducers M_{list}, M'_{list} and $C_{\Delta'_{list}}$ have already been given in previous examples.

- By applying Construction 3.2 on M'_{list} and $C_{\Delta'_{list}}$, we obtain the Macro Tree Transducer $C_{M_{list}} = (\{(list,s)^{(2)},(exp,s)^{(3)},(1_{exp},1_s)^{(3)}\},Nat,Nat,(list,s)(x_1,zero),R)$ with set of rules R:

$$\begin{array}{lll}
\text{(i)} : (list,s)(succ(x_1),z_1) & \to & succ((exp,s)(x_1, \\
 & & \qquad (1_{exp},1_s)(x_1,zero,(list,s)(x_1,z_1)), \\
 & & \qquad (list,s)(x_1,z_1))) \\
\text{(ii)} : (list,s)(zero,z_1) & \to & succ(z_1) \\
\text{(iii)} : (exp,s)(succ(x_1),y_{1,s},z_1) & \to & succ((exp,s)(x_1, \\
 & & \quad (exp,s)(x_1,y_{1,s},(1_{exp},1_s)(x_1,zero,z_1)),z_1)) \\
\text{(iv)} : (exp,s)(zero,y_{1,s},z_1) & \to & succ(y_{1,s}) \\
\text{(v)} : (1_{exp},1_s)(succ(x_1),y_{1,s},z_{s,1}) & \to & (1_{exp},1_s)(x_1,zero,(1_{exp},1_s)(x_1,zero,z_{s,1})) \\
\text{(vi)} : (1_{exp},1_s)(zero,y_{1,s},z_{s,1}) & \to & z_{s,1}
\end{array}$$

  Note, that here the first Macro Tree Transducer is non-copying, but not single-use[2]. Thus, this example really requires our new construction and cannot be handled by appealing to the weaker result $MAC_{su};MAC_{wsu} \subseteq MAC$.

- By analyzing the set of cd-functions for $C_{M_{list}}$—as described in Section 4.1—, we detect that state (exp,s) never needs its second context parameter and state $(1_{exp},1_s)$ never its first. Consequently, we can simplify the Macro Tree Transducer to $C'_{M_{list}} = (\{(list,s)^{(2)},(exp,s)^{(2)},(1_{exp},1_s)^{(2)}\},Nat,Nat,(list,s)(x_1,zero),R')$ with set of rules R':

$$\begin{array}{lll}
\text{(i)} : (list,s)(succ(x_1),z_1) & \to & succ((exp,s)(x_1,(1_{exp},1_s)(x_1,(list,s)(x_1,z_1)))) \\
\text{(ii)} : (list,s)(zero,z_1) & \to & succ(z_1) \\
\text{(iii)} : (exp,s)(succ(x_1),y_{1,s}) & \to & succ((exp,s)(x_1,(exp,s)(x_1,y_{1,s}))) \\
\text{(iv)} : (exp,s)(zero,y_{1,s}) & \to & succ(y_{1,s}) \\
\text{(v)} : (1_{exp},1_s)(succ(x_1),z_{s,1}) & \to & (1_{exp},1_s)(x_1,(1_{exp},1_s)(x_1,z_{s,1})) \\
\text{(vi)} : (1_{exp},1_s)(zero,z_{s,1}) & \to & z_{s,1}
\end{array}$$

- Analyzing the set of pr-functions for $C'_{M_{list}}$—as described in Section 4.2—tells us that the state $(1_{exp},1_s)$ always projects on its context parameter. Thus, we can further simplify to the Macro Tree Transducer $C''_{M_{list}} = (\{(list,s)^{(2)},(exp,s)^{(2)}\},Nat,Nat,(list,s)(x_1,zero),R'')$ with set of rules R'' as follows:

[2] In fact, the tree transduction $\tau(M'_{list})$ cannot be realized by any weakly single-use Macro Tree Transducer (or any Attributed Tree Transducer), due to the linear bound (in the input-size) of the output-height of weakly single-use Macro Tree Transducers, which was quoted in the last section.

$$\begin{array}{lll} \text{(i)} : & (list, s)(succ(x_1), z_1) & \to succ((exp, s)(x_1, (list, s)(x_1, z_1))) \\ \text{(ii)} : & (list, s)(zero, z_1) & \to succ(z_1) \\ \text{(iii)} : & (exp, s)(succ(x_1), y_{1,s}) & \to succ((exp, s)(x_1, (exp, s)(x_1, y_{1,s}))) \\ \text{(iv)} : & (exp, s)(zero, y_{1,s}) & \to succ(y_{1,s}) \end{array}$$

□

5.3 An Implementation

The Construction 3.2 has been implemented in the Haskell$^+$ system by Andreas Maletti [HVM$^+$01]. As an example for applying the system, we consider the following program.

Example 5.9

```
begindata Data
  data List = A List | B List | E
enddata

beginmag App [Mac,Mat,Su,Swp,Tl,Wp,Wsu,Xlin,Xnd,Ylin,Ynd]
  input Data
  syn app :: List -> List -> List
  app (A x) y = A (app x y)
  app (B x) y = B (app x y)
  app E     y = y
endmag

{- COMPOSITION: replace function calls as follows
  app (app x1 y1) z1 = (comp_app_o_app x1 (app y1 (comp_1app_o_1app x1 E z1)) z1)
-}
{- REMOVE SUPERFLUOUS CONTEXT PARAMETERS: replace function calls as follows
  comp_app_o_app   t y1 y2 = comp_app_o_app_   t y1
  comp_1app_o_1app t y1 y2 = comp_1app_o_1app_ t y2
-}
{- ELIMINATE COPY-STATES: replace function calls as follows
  comp_1app_o_1app_ t y = y
-}
beginmag App_o_App [Mac]
  input Data
  syn comp_app_o_app_ :: List -> List -> List
  comp_app_o_app_ (A x1) y1 = A (comp_app_o_app_ x1 y1)
  comp_app_o_app_ (B x1) y1 = B (comp_app_o_app_ x1 y1)
  comp_app_o_app_ E      y1 = y1
endmag
```

□

The input to the system consisted just of the first eleven lines, containing the definition of a Haskell data type `List` and a function `app`, which computes concatenation on the `List` type. The program also contains special keywords like **`begindata`**, **`enddata`**, **`input`** and **`syn`** that are used in the Haskell$^+$ language to specify ranked alphabets and tree transducers[3].

The system was instructed to analyze the input program and to compose the Macro Tree Transducer `App` with itself, followed by post-processing steps as presented in Chapter 4.

[3]Actually, a **`beginmag`**–**`endmag`** block specifies a *macro attribute grammar* (cf. [AJKV98]). A Macro Tree Transducer is just a special case of this concept.

The analyzing phase outputs a list of properties behind the "`beginmag App`" statement. In particular, the list items `Mac`, `Ylin` and `Wsu` tell us that we have a Macro Tree Transducer, which is non-copying and weakly single-use, respectively. Thus, the composition construction presented in Chapter 3 was indeed applicable and lead to the introduction of a new Macro Tree Transducer `App_o_App` into the output program. Also, the Haskell+ system outputs—in comments above the resulting Macro Tree Transducer—a list of equations that can be used as rewrite rules in order to take advantage of the composition and post-processing transformations[4].

Applying these equations from left to right, we get the following calculation:

```
app (app x1 y1) z1
   = (by the comment COMPOSITION)
(comp_app_o_app x1 (app y1 (comp_1app_o_1app x1 E z1)) z1)
   = (by the comment REMOVE SUPERFLUOUS CONTEXT PARAMETERS)
(comp_app_o_app_ x1 (app y1 (comp_1app_o_1app_ x1 z1)))
   = (by the comment ELIMINATE COPY-STATES)
(comp_app_o_app_ x1 (app y1 z1))
```

Since the equations, which were constructed for `comp_app_o_app_`, are essentially the same as those for `app`, this corresponds to a well known optimizing transformation, namely making use of the associativity of the concatenation function.

[4] In an optimizing compiler, the system would have to detect appropriate places in the program, where these equations should be applied.

Chapter 6

Conclusions

In this thesis, we have developed a direct construction that composes a non-copying Macro Tree Transducer and a weakly single-use Macro Tree Transducer, which can be regarded as special functional programs. Since every Macro Tree Transduction can also be realized by the composition of a non-copying Macro Tree Transducer and a weakly single-use one, we get the characterization $MAC = MAC_{nc}; MAC_{wsu}$. As a corollary of this result, we obtain a size-height property for non-copying Macro Tree Transducers.

Further, we have developed two constructions for post-processing Macro Tree Transducers, in order to eliminate inefficiencies in the program resulting from the composition process. In particular, we have solved, for every Macro Tree Transducer, the problem of superfluous traversals through the input tree caused by copy-states.

Thus, we have broadened the applicability of a technique—first proposed in [Küh97, Küh98]—for eliminating intermediate data structures in functional programs. However, we have not formally considered efficiency concerns in this thesis. In fact, there are examples, for which a lazy evaluation strategy on the original program—using an intermediate result—performs less reduction steps than on the program obtained by eliminating the intermediate result with the proposed techniques. It is thus important for future work to develop decision procedures that determine, when a composition technique should be applied, or to identify subclasses of Macro Tree Transducers, for which the application of a certain composition technique is guaranteed to improve the efficiency of programs. For the composition results $TOP; MAC \subseteq MAC$ and $MAC; TOP \subseteq MAC$, such a systematic study has begun in [Höf99], where also a formal quantitative comparison between these techniques and deforestation (cf. [Wad90]) is drawn. Also other transformation techniques, such as shortcut deforestation (see [GLP93]) and program calculation techniques based on recursion operators (see [MFP91]), are interesting for a comparison. A new approach that might be applicable to analyzing efficiency issues in composition transformations, is followed in [FV01], where Top-Down Tree Series Transducers (a generalization of Top-Down Tree Transducers obtained by associating a cost to computations) are introduced and composition results for them are studied.

Another possibility of further investigation is to further relax the restrictions needed to compose two Macro Tree Transducers or to find other restrictions that enable such a composition. We have already noted that for well-presented Macro Tree Transducers (cf. [CF82, FV98]), we get the characterization $MAC_{nc}; MAC_{wp} = MAC$. We believe that a direct composition construction for this setting can be given based on a similar idea as the one exploited in the main construction of the present thesis. Also, we consider it fruitful to further investigate composition constructions for Macro Attributed Tree Transducers (see [KV94, Küh98]), which we believe will give us interesting transformation techniques for higher-order functional programs.

Acknowledgements

This thesis has been written as part of my studies in the International Masters Programme Computational Logic in Dresden. It is a great programme and I would like to thank all the people involved in making it thus.

I want to thank my supervisor Armin Kühnemann, who found the patience to struggle through my explanations and proofs, and made good comments and suggestions, which I believe improved the readability a lot. Heiko Vogler and Armin Kühnemann introduced me to the fascinating field of tree transducers and program transformations. I am grateful for their support and for them to give me challenging tasks and problems during the last two years.

I would also like to thank Andreas Maletti for providing an implementation of my construction in the Haskell$^+$ system.

Part of this thesis has been written while I was a visiting student at the Computing Laboratory in Oxford. Many thanks to Richard Bird and the Algebra of Programming group for welcoming me there, it was a very interesting time.

The “Deutscher Akademischer Austauschdienst” and “Bundesstiftung Rosa Luxemburg” supported me during my studies. In particular, their financial support enabled me to make the great experience of going to Oxford for a term.

Last, but not least, I want to thank Marina and my family for being there.

Bibliography

[AJKV98] I. Adamski, B. Jokubeit, A. Kühnemann, and H. Vogler. Incremental evaluators for ordered macro attribute grammars. Technical Report TUD/FI98/10, Dresden University of Technology, 1998.

[BD77] R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24:44–67, 1977.

[CDPR99] L. Correnson, E. Duris, D. Parigot, and G. Roussel. Declarative program transformation: a deforestation case-study. In G. Nadathur, editor, *International Conference on Principles and Practice of Declarative Programming, Paris, France, Proceedings*, volume 1702 of *LNCS*, pages 360–377. Springer-Verlag, September 1999.

[CF82] B. Courcelle and P. Franchi–Zannettacci. Attribute grammars and recursive program schemes. *Theoret. Comput. Sci.*, 17:163–191, 235–257, 1982.

[Eng80] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory; perspectives and open problems*, pages 241–286. New York, Academic Press, 1980.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31:71–145, 1985.

[Fra82] P. Franchi–Zannettacci. *Attributs sémantiques et schémas de programmes.* PhD thesis, Université de Bordeaux I, 1982. Thèse d'Etat.

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

[FV98] Z. Fülöp and H. Vogler. *Syntax-directed semantics — Formal models based on tree transducers.* Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, 1998.

[FV01] Z. Fülöp and H. Vogler. Bottom-up and top-down tree series transformations. March 2001. Submitted for publication.

[Gan83] H. Ganzinger. Increasing modularity and language–independency in automatically generated compilers. *Sci. of Comput. Prog.*, 3:223–278, 1983.

[GG84] H. Ganzinger and R. Giegerich. Attribute coupled grammars. In *Proceedings of the ACM SIGPLAN'84, Symposium on Compiler Construction, SIGPLAN Notices Vol.19, No.6*, pages 157–170, June 1984.

[Gie88] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25:355–423, 1988.

[GLP93] A. Gill, J. Launchbury, and S.L. Peyton Jones. A short cut to deforestation. In *Conference on Functional Programming Languages and Computer Architecture, Copenhagen, Denmark, Proceedings*, pages 223–232. ACM Press, 1993.

[Höf99] M. Höff. Vergleich von Verfahren zur Elimination von Zwischenergebnissen bei funktionalen Programmen. Master's thesis, Dresden University of Technology, September 1999.

[Hue80] G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27:797–821, 1980.

[HVM$^+$01] M. Höff, R. Vater, A. Maletti, A. Kühnemann, and J. Voigtländer. Tree transducer based program transformations for Haskell$^+$. Progress report of the Haskell$^+$ group, Dresden University of Technology, 2001.

[Knu68] D.E. Knuth. Semantics of context–free languages. *Math. Syst. Theory*, 2:127–145, 1968. Corrections *Ibid.*, 5:95–96, 1971.

[Küh97] A. Kühnemann. *Berechnungsstärken von Teilklassen primitiv–rekursiver Programmschemata*. PhD thesis, Dresden University of Technology, 1997. Shaker Verlag, Aachen.

[Küh98] A. Kühnemann. Benefits of tree transducers for optimizing functional programs. In V. Arvind and R. Ramanujam, editors, *Foundations of Software Technology & Theoretical Computer Science, Chennai, India, Proceedings*, volume 1530 of *LNCS*, pages 146–157. Springer-Verlag, December 1998.

[Küh00] A. Kühnemann. Attribute grammars and program optimization. Script, Dresden University of Technology, Department of Computer Science, 2000.

[KV94] A. Kühnemann and H. Vogler. Synthesized and inherited functions — a new computational model for syntax–directed semantics. *Acta Informatica*, 31:431–477, 1994.

[Les99] C. Lescher. Entwurf und Implementierung einer Eingabesprache für ein System zur Erzeugung syntaxgesteuerter Editoren. Großer Beleg, Dresden University of Technology, 1999.

[Mal89] G. Malcolm. Homomorphisms and promotability. In J.L.A. van de Snepscheut, editor, *Mathematics of Program Construction*, volume 375 of *LNCS*, pages 335–347. Springer-Verlag, 1989.

[MFP91] E. Meijer, M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *Conference on Functional Programming Languages and Computer Architecture, Cambridge, Massachusetts, Proceedings*, pages 124–144, August 1991.

[Rou70] W.C. Rounds. Mappings and grammars on trees. *Math. Syst. Theory*, 4:257–287, 1970.

[Tur86] V.F. Turchin. The concept of a supercompiler. *ACM Trans. on Prog. Lang. and Systems*, 8:292–325, 1986.

[Wad90] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoret. Comput. Sci.*, 73:231–248, 1990.

Appendix A

Proving the Correctness of Construction 3.2

In this appendix, we want to show that Construction 3.2 indeed produces a Macro Tree Transducer, which is equivalent to the composition of the two given ones. We proceed in two stages, similar to the way we introduced the construction. First, we will formalize the idea of "walking upwards" in the intermediate result to obtain the context parameters of calls of the second transducer's states on the context parameters of the first tree transducer, as presented in Section 3.1. Therefore, we introduce a function that gives answers to the question marks in Figure 3.2. We will prove this formally and present some auxiliary notions and properties of this function. We will also introduce an ordering relation, which will prove useful in the "cutting" of potential cycles mentioned at the end of Section 3.1. As second step, we will then use the introduced notions and properties to show that the underlying idea, presented in the informal discussion before and now formalized exactly, can in fact be implemented as a Macro Tree Transducer and that Construction 3.2 is correct in that it constructs a Macro Tree Transducer $M_{1;2}$, which implements the composition of the two functions implemented by the given transducers M_1 and M_2.

In the following, let M_1 and M_2 be two fixed Macro Tree Transducers as in Construction 3.2. We will also use the notations and names introduced there. For technical reasons, we assume without loss of generality that there are no name conflicts between M_1 and M_2, i.e., all involved ranked alphabets are pairwise disjoint. This can always be achieved by renaming and guarantees, e.g., that rewriting rules of M_1 and M_2 can be applied in arbitrary order ($\Rightarrow_{R_1 \cup R_2}$ is confluent).
First, we introduce a set of suspended forms that can occur, when on some input tree a state of M_1 is called with unknown context parameters and the result of this is processed with some state of M_2.

Definition A.1 For every $r \in \mathbf{N}$ and set of variables V (with $Y_r \cap V = \emptyset$), we define the set $SUS(r, V)$ of *suspended forms with r open positions and context variables V* as the smallest set $SUS \subseteq T_{P \cup \Omega}(Y_r \cup V)$ satisfying the following conditions:

(1) $V \subseteq SUS$
(2) for every $\omega \in \Omega^{(n)}$ with $n \in \mathbf{N}$, and $\xi_1, \ldots, \xi_n \in SUS$ holds: $\omega(\xi_1, \ldots, \xi_n) \in SUS$
(3) for every $p \in P^{(n+1)}$ with $n \in \mathbf{N}$, $y_h \in Y_r$ and $\xi_1, \ldots, \xi_n \in SUS$ holds: $p(y_h, \xi_1, \ldots, \xi_n) \in SUS$

□

Note, that elements of $SUS(r,V)$ are always normal forms with respect to $\Rightarrow_{R_1 \cup R_2}$.

Now, we introduce functions that can be used to answer questions of the kind posed in Figure 3.2.

Definition A.2 We define the following set of functions by simultaneous recursion:

$$\{\kappa_{h,q,l,p} \mid r,s \in \mathbf{N}, q \in Q^{(r+1)}, p \in P^{(s+1)}, h \in [r], l \in [s]\}$$
$$\cup \{\pi_{p,\phi,k,r} \mid k,r \in \mathbf{N}, \phi \in RHS(Q,\Delta,k,Y_r), p \in P\}$$

These functions have the following types:

for $r,s \in \mathbf{N}$, $q \in Q^{(r+1)}$, $p \in P^{(s+1)}$, $h \in [r]$ and $l \in [s]$:
$\kappa_{h,q,l,p} : T_\Sigma \to SUS(r, Z_P)$

for $k,r,s \in \mathbf{N}$, $\phi \in RHS(Q,\Delta,k,Y_r)$ and $p \in P^{(s+1)}$:
$\pi_{p,\phi,k,r} : \underbrace{T_\Sigma \times \cdots \times T_\Sigma}_{k-times} \times \{o \in occ(\phi) \mid lab(\phi,o) \notin X_k\} \times [s] \to SUS(r, Z_P)$

For every $r,s \in \mathbf{N}$, $q \in Q^{(r+1)}$, $p \in P^{(s+1)}$, $h \in [r]$ and $l \in [s]$, the function $\kappa_{h,q,l,p}$ is defined as follows. For every $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $t_1,\ldots,t_k \in T_\Sigma$, we have:

$\kappa_{h,q,l,p}(\sigma(t_1,\ldots,t_k)) =$

- if $rhs_{q,\sigma}$ contains y_h at the unique occurrence o (M_1 is non-copying!), then:
  $\pi_{p,rhs_{q,\sigma},k,r}(t_1,\ldots,t_k,o,l)$
- if $rhs_{q,\sigma}$ does not contain y_h, then:
  nil

For every $k,r,s \in \mathbf{N}$, $\phi \in RHS(Q,\Delta,k,Y_r)$ and $p \in P^{(s+1)}$, the function $\pi_{p,\phi,k,r}$ is defined (with $l \in [s]$) as follows:

- $\pi_{p,\phi,k,r}(t_1,\ldots,t_k,\epsilon,l) = z_{p,l}$
- $\pi_{p,\phi,k,r}(t_1,\ldots,t_k,oi,l) =$
  - in case $lab(\phi,o) = \delta$ with $o \in occ(\phi)$, $a \in \mathbf{N}$, $\delta \in \Delta^{(a)}$ and $i \in [a]$:
    If, with $s' \in \mathbf{N}$, $p' \in P^{(s'+1)}$ and $\psi_1,\ldots,\psi_s \in RHS(P,\Omega,a,Y_{s'})$, the only occurrence of a $p(x_i)$-call in the δ-rules of M_2 looks as follows:
    $$p'(\delta(x_1,\ldots,x_a),y_1,\ldots,y_{s'}) \to \cdots p(x_i,\psi_1,\ldots,\psi_s)\cdots,$$
    then the right-hand side of the equation is:
    $$nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi,ou))[x_v \leftarrow t_v, 1 \le v \le k]), 1 \le u \le a,$$
    $$y_w \leftarrow \pi_{p',\phi,k,r}(t_1,\ldots,t_k,o,w), 1 \le w \le s']).$$
    If no such call exists in the δ-rules of M_2, then the right-hand side of the equation is nil.
  - in case $lab(\phi,o) = q$ with $o \in occ(\phi)$, $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $lab(\phi,o1) = x_b \in X_k$ and $2 \le i \le a+1$:
    $$(nf(\Rightarrow_{R_2}, (\kappa_{i-1,q,l,p}(t_b))[y_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi,o(u+1)))[x_v \leftarrow t_v, v \in [k]]),$$
    $$1 \le u \le a]))$$
    $$[z_{p',w} \leftarrow \pi_{p',\phi,k,r}(t_1,\ldots,t_k,o,w),\ s' \in \mathbf{N}, p' \in P^{(s'+1)}, 1 \le w \le s']$$

□

Note the similarities between the definitions of $\kappa_{h,q,l,p}$-functions and right-hand sides for states (h_q, l_p), and between the definitions of $\pi_{p,\phi,k,r}$-functions and $ctx_{p,\phi,k,r}$-functions, respectively. These correspondences will be made precise in Lemma A.14.

Example A.3 We recall the Macro Tree Transducers M_1 and M_2 from Example 3.3 and compute $\kappa_{1,q_1,1,p_2}(A(B(E)))$ as follows:

$$
\begin{array}{l}
\kappa_{1,q_1,1,p_2}(A(B(E))) \\
\quad = \text{ (by definition of } \kappa_{1,q_1,1,p_2}) \\
\pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),12,1) \\
\quad = \text{ (by definition of } \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}) \\
(nf(\Rightarrow_{R_2},(\kappa_{1,q_1,1,p_2}(B(E)))[y_1 \leftarrow nf(\Rightarrow_{R_1}, y_1[x_1 \leftarrow B(E)])])) \\
\quad [z_{p_1,1}, z_{p_2,1} \leftarrow \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1), \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1)] \\
\quad = \text{ (by definition of } \kappa_{1,q_1,1,p_2}) \\
(nf(\Rightarrow_{R_2},(\pi_{p_2,q_1(x_1,\beta(y_1)),1,1}(E,21,1))[y_1 \leftarrow y_1])) \\
\quad [z_{p_1,1}, z_{p_2,1} \leftarrow \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1), \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1)] \\
\quad = \text{ (by definition of } \pi_{p_2,q_1(x_1,\beta(y_1)),1,1}) \\
(nf(\Rightarrow_{R_2}, nf(\Rightarrow_{R_2}, p_1(x_1,y_1)[x_1 \leftarrow nf(\Rightarrow_{R_1}, y_1[x_1 \leftarrow E]), \\
\qquad\qquad\qquad\qquad\qquad\qquad y_1 \leftarrow \pi_{p_2,q_1(x_1,\beta(y_1)),1,1}(E,2,1)]))) \\
\quad [z_{p_1,1}, z_{p_2,1} \leftarrow \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1), \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1)] \\
\quad = \text{ (by substitution and normal form)} \\
(nf(\Rightarrow_{R_2}, p_1(y_1, \pi_{p_2,q_1(x_1,\beta(y_1)),1,1}(E,2,1)))) \\
\quad [z_{p_1,1}, z_{p_2,1} \leftarrow \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1), \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1)] \\
\quad = \text{ (by definition of } \pi_{p_2,q_1(x_1,\beta(y_1)),1,1}) \\
(nf(\Rightarrow_{R_2}, p_1(y_1,(nf(\Rightarrow_{R_2},(\kappa_{1,q_1,1,p_2}(E))[y_1 \leftarrow nf(\Rightarrow_{R_1}, \beta(y_1)[x_1 \leftarrow E])])) \\
\qquad\qquad\qquad [z_{p_1,1}, z_{p_2,1} \leftarrow \pi_{p_1,q_1(x_1,\beta(y_1)),1,1}(E,\epsilon,1), \pi_{p_2,q_1(x_1,\beta(y_1)),1,1}(E,\epsilon,1)]))) \\
\quad [z_{p_1,1}, z_{p_2,1} \leftarrow \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1), \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1)] \\
\quad = \text{ (by definition of } \pi_{p_1,q_1(x_1,\beta(y_1)),1,1} \text{ and } \pi_{p_2,q_1(x_1,\beta(y_1)),1,1}) \\
(nf(\Rightarrow_{R_2}, p_1(y_1,(nf(\Rightarrow_{R_2},(\kappa_{1,q_1,1,p_2}(E))[y_1 \leftarrow \beta(y_1)])) \\
\qquad\qquad\qquad [z_{p_1,1}, z_{p_2,1} \leftarrow z_{p_1,1}, z_{p_2,1}]))) \\
\quad [z_{p_1,1}, z_{p_2,1} \leftarrow \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1), \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1)] \\
\quad = \text{ (by definition of } \kappa_{1,q_1,1,p_2}) \\
(nf(\Rightarrow_{R_2}, p_1(y_1, nf(\Rightarrow_{R_2},(\pi_{p_2,y_1,0,1}(\epsilon,1))[y_1 \leftarrow \beta(y_1)])))) \\
\quad [z_{p_1,1}, z_{p_2,1} \leftarrow \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1), \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1)] \\
\quad = \text{ (by definition of } \pi_{p_2,y_1,0,1}) \\
(nf(\Rightarrow_{R_2}, p_1(y_1, nf(\Rightarrow_{R_2}, z_{p_2,1}[y_1 \leftarrow \beta(y_1)])))) \\
\quad [z_{p_1,1}, z_{p_2,1} \leftarrow \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1), \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1)] \\
\quad = \text{ (by normal form and substitution)} \\
p_1(y_1, \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}(B(E),1,1)) \\
\quad = \text{ (by definition of } \pi_{p_2,\alpha(q_1(x_1,y_1)),1,1}) \\
p_1(y_1, nf(\Rightarrow_{R_2}, y_1[x_1 \leftarrow nf(\Rightarrow_{R_1}, q_1(x_1,y_1)[x_1 \leftarrow B(E)]), \\
\qquad\qquad\qquad y_1 \leftarrow \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}(B(E),\epsilon,1)])) \\
\quad = \text{ (by definition of } \pi_{p_1,\alpha(q_1(x_1,y_1)),1,1}) \\
p_1(y_1, nf(\Rightarrow_{R_2}, y_1[x_1 \leftarrow nf(\Rightarrow_{R_1}, q_1(x_1,y_1)[x_1 \leftarrow B(E)]), \\
\qquad\qquad\qquad y_1 \leftarrow z_{p_1,1}])) \\
\quad = \text{ (by substitution and normal form)} \\
p_1(y_1, z_{p_1,1})
\end{array}
$$

□

The following theorem establishes the key property of the functions introduced in Definition A.2.

Theorem A.4 For every $t \in T_\Sigma$, $r, s, s' \in \mathbf{N}$, $q \in Q^{(r+1)}$, $p \in P^{(s+1)}$, $p' \in P^{(s'+1)}$, $h \in [r]$ and $l \in [s]$ holds, if during a reduction of $p'(q(t, y_1, \ldots, y_r), z_1, \ldots, z_{s'})$ with $\Rightarrow_{R_1 \cup R_2}$ an instance of a call $p(y_h, \xi_1, \ldots, \xi_s)$ occurs, then we have $\kappa_{h,q,l,p}(t) \in SUS(r, \{z_{p',1}, \ldots, z_{p',s'}\})$ and $nf(\Rightarrow_{R_1 \cup R_2}, \xi_l) = (\kappa_{h,q,l,p}(t))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$. □

Proof: In order to describe the context parameters, with which states of M_2 reach certain occurrences in a tree, we need the notion of *marking* a tree. Therefore, we define a function $mark$ on arbitrary trees, which adds to every symbol as a superscript its occurrence in the tree. For example, we have $mark(\delta(q(x, y), \alpha)) = \delta^\epsilon(q^1(x^{11}, y^{12}), \alpha^2)$. All operations on marked trees work in principle like on ordinary trees. Markings are ignored, e.g., in matching a tree against the left-hand side of a rewriting rule, but markings are preserved by, e.g., rewriting, substitutions or the sub-function. If there is, e.g., a rule $p(\delta(x_1, x_2)) \to \gamma(p(x_2))$, then this induces the rewriting $p(\delta^\epsilon(q^1(x^{11}, y^{12}), \alpha^2)) \Rightarrow \gamma(p(\alpha^2))$. We will use the placeholder $\bullet^o$ for an arbitrary tree, which has a root symbol marked with occurrence o. Now, we prove the following two statements by simultaneous induction.

(I) For $\mathbf{t \in T_\Sigma}$: for every $r, s, s' \in \mathbf{N}$, $q \in Q^{(r+1)}$, $p \in P^{(s+1)}$, $p' \in P^{(s'+1)}$, $h \in [r]$ and $l \in [s]$ holds, if during a reduction of $p'(q(t, y_1, \ldots, y_r), z_1, \ldots, z_{s'})$ with $\Rightarrow_{R_1 \cup R_2}$ an instance of $p(y_h, \xi_1, \ldots, \xi_s)$ occurs, then $\kappa_{h,q,l,p}(t) \in SUS(r, \{z_{p',1}, \ldots, z_{p',s'}\})$ and:
$nf(\Rightarrow_{R_1 \cup R_2}, \xi_l) = (\kappa_{h,q,l,p}(t))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$

(II) For $\mathbf{k \in N, t_1, \ldots, t_k \in T_\Sigma}$: for every $r \in \mathbf{N}$, $\phi \in RHS(Q, \Delta, k, Y_r)$, $o \in occ(\phi)$, such that $lab(\phi, o) \notin X_k$, $s, s' \in \mathbf{N}$, $p \in P^{(s+1)}$, $p' \in P^{(s'+1)}$ and $l \in [s]$ holds, if during a reduction of $p'((mark(\phi))[x_v \leftarrow t_v, v \in [k]], z_1, \ldots, z_{s'})$ with $\Rightarrow_{R_1 \cup R_2}$ an instance of a call $p(\bullet^o, \xi_1, \ldots, \xi_s)$ occurs, then we have
$\pi_{p,\phi,k,r}(t_1, \ldots, t_k, o, l) \in SUS(r, \{z_{p',1}, \ldots, z_{p',s'}\})$ and:
$nf(\Rightarrow_{R_1 \cup R_2}, \xi_l) = (\pi_{p,\phi,k,r}(t_1, \ldots, t_k, o, l))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$

(I) $\Leftarrow$ (II) : Let $t = \sigma(t_1, \ldots, t_k)$ with $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T_\Sigma$, then:

If during a reduction of $p'(q(\sigma(t_1, \ldots, t_k), y_1, \ldots, y_r), z_1, \ldots, z_{s'})$ with $\Rightarrow_{R_1 \cup R_2}$ an instance of $p(y_h, \xi_1, \ldots, \xi_s)$ occurs, then it must be the case that this instance occurs during a reduction of $p'(rhs_{q,\sigma}[x_v \leftarrow t_v, v \in [k]], z_1, \ldots, z_{s'})$ with $rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$. This is only possible, if there is a—then unique, because M_1 is non-copying—occurrence o of $rhs_{q,\sigma}$ with $lab(rhs_{q,\sigma}, o) = y_h$. Consequently, the $\xi_1, \ldots, \xi_s$ appear in an occurrence $p({y_h}^o, \xi_1, \ldots, \xi_s)$, resulting from a reduction of $p'((mark(rhs_{q,\sigma}))[x_v \leftarrow t_v, v \in [k]], z_1, \ldots, z_{s'})$. By induction hypothesis **(II)**, this means that $\pi_{p,rhs_{q,\sigma},k,r}(t_1, \ldots, t_k, o, l) \in SUS(r, \{z_{p',1}, \ldots, z_{p',s'}\})$ and:
$nf(\Rightarrow_{R_1 \cup R_2}, \xi_l) = (\pi_{p,rhs_{q,\sigma},k,r}(t_1, \ldots, t_k, o, l))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$
By the definition of $\kappa_{h,q,l,p}$ for $t = \sigma(t_1, \ldots, t_k)$, this amounts to $\kappa_{h,q,l,p}(\sigma(t_1, \ldots, t_k)) \in SUS(r, \{z_{p',1}, \ldots, z_{p',s'}\})$ and:
$nf(\Rightarrow_{R_1 \cup R_2}, \xi_l) = (\kappa_{h,q,l,p}(\sigma(t_1, \ldots, t_k)))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$

(II) $\Leftarrow$ (I) : For fixed r and ϕ, we prove the claim by induction on the prefix-order of occurrences.

In the base case ϵ, we have an instance of a call $p(\bullet^\epsilon, \xi_1, \ldots, \xi_s)$ during reduction of $p'((mark(\phi))[x_v \leftarrow t_v, v \in [k]], z_1, \ldots, z_{s'})$. This is only possible, if $p = p'$ and $\xi_1, \ldots, \xi_s = z_1, \ldots, z_{s'}$, in which case $\pi_{p,\phi,k,r}(t_1, \ldots, t_k, \epsilon, l) = z_{p,l} \in SUS(r, \{z_{p',1}, \ldots, z_{p',s'}\})$ and:

$nf(\Rightarrow_{R_1 \cup R_2}, \xi_l)$
$\quad = \text{(by } \xi_1, \ldots, \xi_s = z_1, \ldots, z_{s'})$
z_l
$\quad = \text{(by } p = p' \text{ and substitution)}$
$z_{p,l}[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$
$\quad = \text{(by definition of } \pi_{p,\phi,k,r})$
$(\pi_{p,\phi,k,r}(t_1, \ldots, t_k, \epsilon, l))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$

In the inductive case $oi \in occ(\phi)$—with $o \in occ(\phi)$, $i \in \mathbf{N}$ and $lab(\phi, oi) \notin X_k$—the reasoning is as follows. If during a reduction of $p'((mark(\phi))[x_v \leftarrow t_v, v \in [k]], z_1, \ldots, z_{s'})$ with $\Rightarrow_{R_1 \cup R_2}$ an instance of a call $p(\bullet^{oi}, \xi_1, \ldots, \xi_s)$ occurs, then the only chance for this to happen is that there exist $s'' \in \mathbf{N}$, $p'' \in P^{(s''+1)}$ and $\rho_1, \ldots, \rho_{s''}$, such that reducing $p'((mark(\phi))[x_v \leftarrow t_v, v \in [k]], z_1, \ldots, z_{s'})$ leads to an occurrence of $p''((sub(mark(\phi), o))[x_v \leftarrow t_v, v \in [k]], \rho_1, \ldots, \rho_{s''})$ and from this call of p'' on $(sub(mark(\phi), o))[x_v \leftarrow t_v, v \in [k]]$ results—by reduction—an instance $p(\bullet^{oi}, \xi_1, \ldots, \xi_s)$. This situation is shown in Figure A.1.

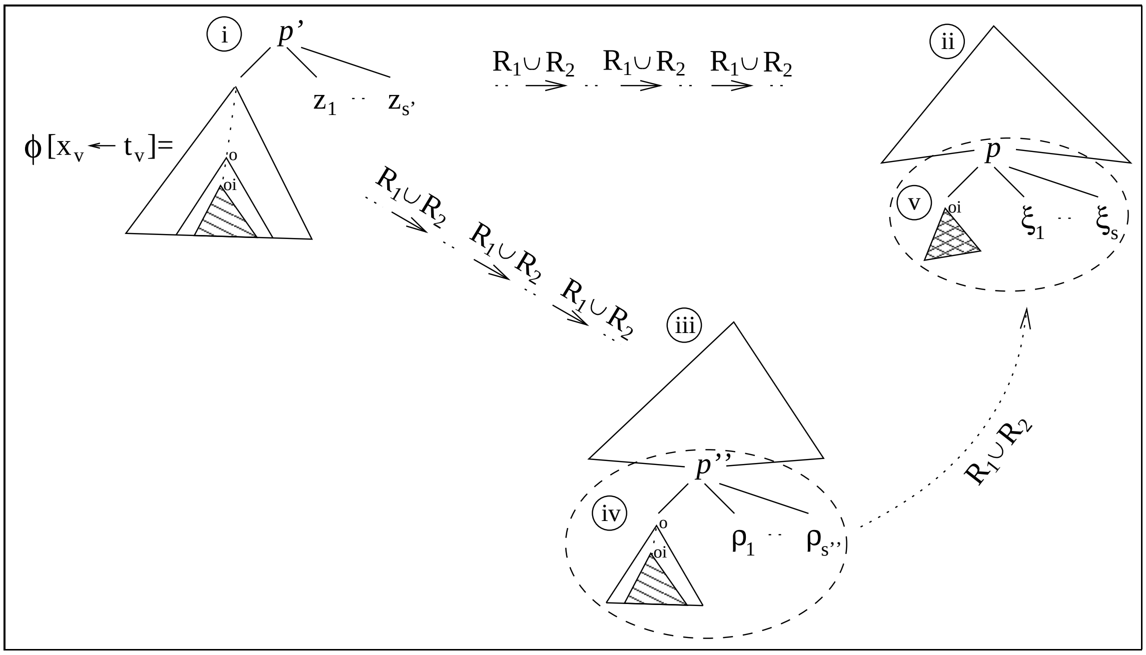

Figure A.1: If **i** reduces to **ii**, then exists **iii**, such that a reduction of **iv** contains a **v**.

By applying the induction hypothesis for o, we know that for every $w \in [s'']$, we have $\pi_{p'',\phi,k,r}(t_1, \ldots, t_k, o, w) \in SUS(r, \{z_{p',1}, \ldots, z_{p',s'}\})$ and:
$nf(\Rightarrow_{R_1 \cup R_2}, \rho_w) = (\pi_{p'',\phi,k,r}(t_1, \ldots, t_k, o, w))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$

We proceed by case distinction on $lab(\phi, o)$:

- $lab(\phi, o) = \delta$ with $a \in \mathbf{N}$, $\delta \in \Delta^{(a)}$ and $i \in [a]$:
  We know that from reduction of the call
  $p''(\delta^o(sub(mark(\phi), o1), \ldots, sub(mark(\phi), oa))[x_v \leftarrow t_v, v \in [k]], \rho_1, \ldots, \rho_{s''})$ results an instance of $p(\bullet^{oi}, \xi_1, \ldots, \xi_s)$. Consequently, there must be a rule
  $$p''(\delta(x_1, \ldots, x_a), y_1, \ldots, y_{s''}) \rightarrow \cdots p(x_i, \psi_1, \ldots, \psi_s) \cdots$$
  in R_2 (with $\psi_1, \ldots, \psi_s \in RHS(P, \Omega, a, Y_{s''})$) and:

$nf(\Rightarrow_{R_1 \cup R_2}, \xi_l) = nf(\Rightarrow_{R_1 \cup R_2}, \psi_l[x_u \leftarrow (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]], 1 \leq u \leq a,$
$y_w \leftarrow \rho_w, 1 \leq w \leq s''])$
$=$ (by confluence considerations)
$nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]]), 1 \leq u \leq a,$
$y_w \leftarrow nf(\Rightarrow_{R_1 \cup R_2}, \rho_w), 1 \leq w \leq s''])$
$=$ (by the information about the normal forms of $\rho_1, \ldots, \rho_{s''}$ gained above)
$nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]]), 1 \leq u \leq a,$
$y_w \leftarrow (\pi_{p'',\phi,k,r}(t_1, \ldots, t_k, o, w))$
$[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}], 1 \leq w \leq s''])$
$=$ (since the only occurrences of $z_{p',1}, \ldots, z_{p',s'}$ are in the expressions substituted for the y_w)
$(nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]]), 1 \leq u \leq a,$
$y_w \leftarrow \pi_{p'',\phi,k,r}(t_1, \ldots, t_k, o, w), 1 \leq w \leq s'']))$
$[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$

Taking into account that the rule from R_2 given above must be the unique occurrence of a $p(x_i)$-call in δ-rules of the weakly single-use Macro Tree Transducer M_2, we have by definition of $\pi_{p,\phi,k,r}$:
$\pi_{p,\phi,k,r}(t_1, \ldots, t_k, oi, l) =$
$nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]]), 1 \leq u \leq a,$
$y_w \leftarrow \pi_{p'',\phi,k,r}(t_1, \ldots, t_k, o, w), 1 \leq w \leq s''])$
Since we have $\pi_{p'',\phi,k,r}(t_1, \ldots, t_k, o, w) \in SUS(r, \{z_{p',1}, \ldots, z_{p',s'}\})$, for every $w \in [s'']$ (see induction hypothesis for o), this implies $\pi_{p,\phi,k,r}(t_1, \ldots, t_k, oi, l) \in SUS(r, \{z_{p',1}, \ldots, z_{p',s'}\})$. Together with the previously computed equivalent of $nf(\Rightarrow_{R_1 \cup R_2}, \xi_l)$, we also have:
$(\pi_{p,\phi,k,r}(t_1, \ldots, t_k, oi, l))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$
$=$
$(nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]]), 1 \leq u \leq a,$
$y_w \leftarrow \pi_{p'',\phi,k,r}(t_1, \ldots, t_k, o, w), 1 \leq w \leq s'']))$
$[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$
$=$
$nf(\Rightarrow_{R_1 \cup R_2}, \xi_l)$
This proves the claim.

- $lab(\phi, o) = q$ with $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $lab(\phi, o1) = x_b \in X_k$ and $2 \leq i \leq a+1$:
We know that from reduction of the call
$p''(q^o({x_b}^{o1}, sub(mark(\phi), o2), \ldots, sub(mark(\phi), o(a+1)))[x_v \leftarrow t_v, v \in [k]],$
$\rho_1, \ldots, \rho_{s''})$
results an instance of $p(\bullet^{oi}, \xi_1, \ldots, \xi_s)$. This means that a call of p reaches the $(i-1)$th context parameter position of the call of q^o on t_b. By the induction hypothesis **(I)** for t_b, we know that for an instance $p(y_{i-1}, {\xi'}_1, \ldots, {\xi'}_s)$, resulting from a reduction of $p''(q(t_b, y_1, \ldots, y_a), z_1, \ldots, z_{s''})$, we have $\kappa_{i-1,q,l,p}(t_b) \in SUS(a, \{z_{p'',1}, \ldots, z_{p'',s''}\})$ and:
$nf(\Rightarrow_{R_1 \cup R_2}, \xi'_l) = (\kappa_{i-1,q,l,p}(t_b))[z_{p'',1}, \ldots, z_{p'',s''} \leftarrow z_1, \ldots, z_{s''}]$
For our instance $p(\bullet^{oi}, \xi_1, \ldots, \xi_s)$, resulting from reducing
$p''(q^o(t_b, (sub(mark(\phi), o2))[x_v \leftarrow t_v, v \in [k]],$
$\ldots,$
$(sub(mark(\phi), o(a+1)))[x_v \leftarrow t_v, v \in [k]]),$
$\rho_1, \ldots, \rho_{s''}),$
this implies the following calculation:

$$nf(\Rightarrow_{R_1\cup R_2},\xi_l) = nf(\Rightarrow_{R_1\cup R_2},\xi'_l[y_u \leftarrow (sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]], u\in[a], z_w \leftarrow \rho_w, 1\le w\le s''])$$

$= $ (by confluence considerations)

$$nf(\Rightarrow_{R_1\cup R_2},(nf(\Rightarrow_{R_1\cup R_2},\xi'_l))[y_u \leftarrow (sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]], u\in[a], z_w \leftarrow \rho_w, 1\le w\le s''])$$

$= $ (by $nf(\Rightarrow_{R_1\cup R_2},\xi'_l) = (\kappa_{i-1,q,l,p}(t_b))[z_{p'',1},\ldots,z_{p'',s''} \leftarrow z_1,\ldots,z_{s''}]$)

$$nf(\Rightarrow_{R_1\cup R_2},(\kappa_{i-1,q,l,p}(t_b))[z_{p'',1},\ldots,z_{p'',s''} \leftarrow z_1,\ldots,z_{s''}] [y_u \leftarrow (sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]], u\in[a], z_w \leftarrow \rho_w, 1\le w\le s''])$$

$= $ (by confluence considerations)

$$nf(\Rightarrow_{R_2},(\kappa_{i-1,q,l,p}(t_b))[z_{p'',1},\ldots,z_{p'',s''} \leftarrow z_1,\ldots,z_{s''}] [y_u \leftarrow nf(\Rightarrow_{R_1},(sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]]), u\in[a], z_w \leftarrow nf(\Rightarrow_{R_1\cup R_2},\rho_w), 1\le w\le s''])$$

$= $ (by the information about the normal forms of $\rho_1,\ldots,\rho_{s''}$ gained above)

$$nf(\Rightarrow_{R_2},(\kappa_{i-1,q,l,p}(t_b))[z_{p'',1},\ldots,z_{p'',s''} \leftarrow z_1,\ldots,z_{s''}] [y_u \leftarrow nf(\Rightarrow_{R_1},(sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]]), u\in[a], z_w \leftarrow (\pi_{p'',\phi,k,r}(t_1,\ldots,t_k,o,w)) [z_{p',1},\ldots,z_{p',s'} \leftarrow z_1,\ldots,z_{s'}], 1\le w\le s''])$$

$= $ (by composing substitutions)

$$nf(\Rightarrow_{R_2},(\kappa_{i-1,q,l,p}(t_b))[y_u \leftarrow nf(\Rightarrow_{R_1},(sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]]), u\in[a], z_{p'',w} \leftarrow (\pi_{p'',\phi,k,r}(t_1,\ldots,t_k,o,w)) [z_{p',1},\ldots,z_{p',s'} \leftarrow z_1,\ldots,z_{s'}], 1\le w\le s''])$$

We obtained above that $\kappa_{i-1,q,l,p}(t_b) \in SUS(a,\{z_{p'',1},\ldots,z_{p'',s''}\})$. Thus, $\kappa_{i-1,q,l,p}(t_b)$ does not contain variables from $Z_P\backslash\{z_{p'',1},\ldots,z_{p'',s''}\}$, and we can transform the previous expression into the following equivalent[i]:

$$nf(\Rightarrow_{R_2},(\kappa_{i-1,q,l,p}(t_b))[y_u \leftarrow nf(\Rightarrow_{R_1},(sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]]), u\in[a], z_{p''',w} \leftarrow (\pi_{p''',\phi,k,r}(t_1,\ldots,t_k,o,w)) [z_{p',1},\ldots,z_{p',s'} \leftarrow z_1,\ldots,z_{s'}], s'''\in\mathbf{N}, p'''\in P^{(s'''+1)}, 1\le w\le s'''])$$

$= $ (by substitution and confluence considerations)

$$(nf(\Rightarrow_{R_2},(\kappa_{i-1,q,l,p}(t_b)) [y_u \leftarrow nf(\Rightarrow_{R_1},(sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]]), u\in[a]])) [z_{p''',w} \leftarrow nf(\Rightarrow_{R_2},(\pi_{p''',\phi,k,r}(t_1,\ldots,t_k,o,w))[z_{p',1},\ldots,z_{p',s'} \leftarrow z_1,\ldots,z_{s'}]), s'''\in\mathbf{N}, p'''\in P^{(s'''+1)}, 1\le w\le s''']$$

$= $ (since the $\pi_{p''',\phi,k,r}(t_1,\ldots,t_k,o,w) \in SUS(r,Z_P)$ are $\Rightarrow_{R_2}$-normal forms)

$$(nf(\Rightarrow_{R_2},(\kappa_{i-1,q,l,p}(t_b)) [y_u \leftarrow nf(\Rightarrow_{R_1},(sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]]), u\in[a]])) [z_{p''',w} \leftarrow (\pi_{p''',\phi,k,r}(t_1,\ldots,t_k,o,w))[z_{p',1},\ldots,z_{p',s'} \leftarrow z_1,\ldots,z_{s'}], s'''\in\mathbf{N}, p'''\in P^{(s'''+1)}, 1\le w\le s''']$$

$= $ (by substitution)

$$(nf(\Rightarrow_{R_2},(\kappa_{i-1,q,l,p}(t_b)) [y_u \leftarrow nf(\Rightarrow_{R_1},(sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]]), u\in[a]])) [z_{p''',w} \leftarrow \pi_{p''',\phi,k,r}(t_1,\ldots,t_k,o,w),\ s'''\in\mathbf{N}, p'''\in P^{(s'''+1)}, 1\le w\le s'''] [z_{p',1},\ldots,z_{p',s'} \leftarrow z_1,\ldots,z_{s'}]$$

[i] It does not matter, which values we replace for the variables from $Z_P\backslash\{z_{p'',1},\ldots,z_{p'',s''}\}$, because they anyway do not occur in $\kappa_{i-1,q,l,p}(t_b)$. We only need to ensure that for the $z_{p'',1},\ldots,z_{p'',s''}$ the same values are substituted as in the previous expression.

By definition of $\pi_{p,\phi,k,r}$, we have:
$\pi_{p,\phi,k,r}(t_1,\ldots,t_k,oi,l) =$
$(nf(\Rightarrow_{R_2},(\kappa_{i-1,q,l,p}(t_b))$
$[y_u \leftarrow nf(\Rightarrow_{R_1},(sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]]), u\in[a]]))$
$[z_{p''',w} \leftarrow \pi_{p''',\phi,k,r}(t_1,\ldots,t_k,o,w),\ s'''\in\mathbf{N}, p'''\in P^{(s'''+1)}, 1\le w\le s''']$
Since we reasoned above that $\kappa_{i-1,q,l,p}(t_b) \in SUS(a,\{z_{p'',1},\ldots,z_{p'',s''}\})$ and we have $\pi_{p'',\phi,k,r}(t_1,\ldots,t_k,o,w) \in SUS(r,\{z_{p',1},\ldots,z_{p',s'}\})$, for every $w\in[s'']$ (see induction hypothesis for o), this implies $\pi_{p,\phi,k,r}(t_1,\ldots,t_k,oi,l) \in SUS(r,\{z_{p',1},\ldots,z_{p',s'}\})$. Together with the previously computed equivalent of $nf(\Rightarrow_{R_1\cup R_2},\xi_l)$, we also have:
$(\pi_{p,\phi,k,r}(t_1,\ldots,t_k,oi,l))[z_{p',1},\ldots,z_{p',s'} \leftarrow z_1,\ldots,z_{s'}]$
$=$
$(nf(\Rightarrow_{R_2},(\kappa_{i-1,q,l,p}(t_b))$
$[y_u \leftarrow nf(\Rightarrow_{R_1},(sub(\phi,o(u+1)))[x_v \leftarrow t_v, v\in[k]]), u\in[a]]))$
$[z_{p''',w} \leftarrow \pi_{p''',\phi,k,r}(t_1,\ldots,t_k,o,w),\ s'''\in\mathbf{N}, p'''\in P^{(s'''+1)}, 1\le w\le s''']$
$[z_{p',1},\ldots,z_{p',s'} \leftarrow z_1,\ldots,z_{s'}]$
$=$
$nf(\Rightarrow_{R_1\cup R_2},\xi_l)$
This proves the claim.

□

On page 16, we posed the following question:

> "Given two states $q'\in Q^{(a+1)}$ and $p'\in P^{(s'+1)}$ and some input tree t for M_1, can we determine, for any state p'' of M_2, and any context parameter y_h from $y_1,\ldots,y_a$, which will be the values in context parameter positions in every occurrence of a call to p'' on y_h during the reduction of $p'(q'(t,y_1,\ldots,y_a),\eta_1,\ldots,\eta_{s'})$?"

Because of Theorem A.4, we can now answer this question positively by stating that the value in the lth context parameter position of every such call of p'' on y_h will be equal—modulo further reductions—to $(\kappa_{h,q',l,p''}(t))[z_{p',1},\ldots,z_{p',s'} \leftarrow \eta_1,\ldots,\eta_{s'}]$.

Example A.5 For the Macro Tree Transducers M_1 and M_2 from Example 3.3, we computed in Example A.3 the following:
$\kappa_{1,q_1,1,p_2}(A(B(E))) = p_1(y_1,z_{p_1,1}) \in SUS(1,\{z_{p_1,1}\})$

Theorem A.4 now says that if during a reduction of $p_1(q_1(A(B(E)),y_1),z_1)$ with $\Rightarrow_{R_1\cup R_2}$ an instance of a call $p_2(y_1,\xi_1)$ occurs, then we have:
$nf(\Rightarrow_{R_1\cup R_2},\xi_1) = p_1(y_1,z_{p_1,1})[z_{p_1,1}\leftarrow z_1]$

In fact, we have the following reduction:
$p_1(q_1(A(B(E)),y_1),z_1) \Rightarrow_{R_1} p_1(\alpha(q_1(B(E),y_1)),z_1) \Rightarrow_{R_2}$
$p_1(q_1(B(E),y_1),p_2(q_1(B(E),y_1),z_1)) \Rightarrow_{R_1} p_1(q_1(B(E),y_1),p_2(q_1(E,\beta(y_1)),z_1)) \Rightarrow_{R_1}$
$p_1(q_1(B(E),y_1),p_2(\beta(y_1),z_1)) \Rightarrow_{R_2} p_1(q_1(B(E),y_1),\underline{p_2(y_1,\underline{p_1(y_1,z_1)})})$

□

Now, we define for every state and every input tree of M_1, a binary relation on pairs, consisting of a context parameter position of the given state from M_1, and some state of the second transducer. This relation is intended to represent the nesting of occurrences of such pairs that can result from processing the intermediate result generated by M_1 (with the given state on the given input tree), with states of M_2.

Definition A.6 For every $r \in \mathbf{N}$, $q \in Q^{(r+1)}$ and $t \in T_\Sigma$, we define a binary relation $\prec_{q,t}$ on $[r] \times P$ as follows. For every $h_1, h_2 \in [r]$ and $p_1, p_2 \in P$, we have $(h_1, p_1) \prec_{q,t} (h_2, p_2)$ iff there exists an $l \in [rank_P(p_1) - 1]$, such that $\kappa_{h_1,q,l,p_1}(t)$ contains an occurrence of a $p_2(y_{h_2}, \cdots)$-call.
For some subset $\mathcal{S} \subseteq [r] \times P$, we write $(h_1, p_1) \prec^*_{q,t} \mathcal{S}$, if there exists an $(h', p') \in \mathcal{S}$ with $(h_1, p_1) \prec^*_{q,t} (h', p')$. Analogous for $\prec^+_{q,t}$. □

At the end of Section 3.1, we claimed that for non-copying M_1 and weakly single-use M_2, we cannot reach a situation, where two occurrences of one and the same such pair are nested, which would correspond to the second of the two "counterexamples" from page 17. Now, we can formalize this claim as follows.

Claim A.7 For every $r \in \mathbf{N}$, $q \in Q^{(r+1)}$ and $t \in T_\Sigma$, there are no $h \in [r]$ and $p \in P$ with $(h, p) \prec^+_{q,t} (h, p)$. □

We will not formally prove this claim here. Instead, we show the following slightly weaker lemma, which in fact covers all cases that are important for the correctness of the Macro Tree Transducer $M_{1;2}$.

Lemma A.8 For every $r, s \in \mathbf{N}$, $q \in Q^{(r+1)}$, $t \in T_\Sigma$, $p \in P^{(s+1)}$ and $h \in [r]$, the following holds: if there exist $s' \in \mathbf{N}$ and $p' \in P^{(s'+1)}$, such that during a reduction of $p'(q(t, y_1, \ldots, y_r), z_1, \ldots, z_{s'})$ with $\Rightarrow_{R_1 \cup R_2}$ an instance of a call $p(y_h, \cdots)$ occurs, then we do not have $(h, p) \prec^+_{q,t} (h, p)$.

Proof: The proof goes by contradiction. Let the instance of $p(y_h, \cdots)$ resulting from reducing $p'(q(t, y_1, \ldots, y_r), z_1, \ldots, z_{s'})$ be $p(y_h, \xi_1, \ldots, \xi_s)$, and assume, there were a natural number n and $(h_0, p_0), \ldots, (h_n, p_n) \in [r] \times P$ with $(h, p) \prec_{q,t} (h_0, p_0) \prec_{q,t} \ldots \prec_{q,t} (h_n, p_n) = (h, p)$. First, we show by induction that then for every $0 \leq k \leq n$ holds: $(h, p) \prec_{q,t} (h_k, p_k)$.

$k = 0$:
$(h, p) \prec_{q,t} (h_0, p_0)$ holds by assumption.

$0 < k \leq n$:
By induction hypothesis, we know that $(h, p) \prec_{q,t} (h_{k-1}, p_{k-1})$ and by assumption also $(h_{k-1}, p_{k-1}) \prec_{q,t} (h_k, p_k)$. With $p_{k-1} \in P^{(s_1+1)}$ and $p_k \in P^{(s_2+1)}$ (for $s_1, s_2 \in \mathbf{N}$), this means—by Definition A.6—that there exist $l_1 \in [s]$ and $l_2 \in [s_1]$, such that $\kappa_{h,q,l_1,p}(t)$ contains an occurrence of a $p_{k-1}(y_{h_{k-1}}, \cdots)$-call and $\kappa_{h_{k-1},q,l_2,p_{k-1}}(t)$ contains an occurrence of a $p_k(y_{h_k}, \cdots)$-call. By Theorem A.4, we have $nf(\Rightarrow_{R_1 \cup R_2}, \xi_{l_1}) = (\kappa_{h,q,l_1,p}(t))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$, consequently this normal form must contain an occurrence of a $p_{k-1}(y_{h_{k-1}}, \rho_1, \ldots, \rho_{s_1})$-call with $\rho_1, \ldots, \rho_{s_1} \in SUS(r, Z_{s'})$. But, this implies that there is a reduction of $p'(q(t, y_1, \ldots, y_r), z_1, \ldots, z_{s'})$ with $\Rightarrow_{R_1 \cup R_2}$ (namely by reducing the ξ_{l_1} in $p(y_h, \xi_1, \ldots, \xi_s)$) leading to an occurrence of a $p_{k-1}(y_{h_{k-1}}, \rho_1, \ldots, \rho_{s_1})$-call and thus again by Theorem A.4: $nf(\Rightarrow_{R_1 \cup R_2}, \rho_{l_2}) = (\kappa_{h_{k-1},q,l_2,p_{k-1}}(t))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$. Since $\rho_{l_2} \in SUS(r, Z_{s'})$ is a $\Rightarrow_{R_1 \cup R_2}$-normal form (by the remark below Definition A.1), we now know that it is equal to $(\kappa_{h_{k-1},q,l_2,p_{k-1}}(t))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$ and thus contains an occurrence of a $p_k(y_{h_k}, \cdots)$-call. Since ρ_{l_2} in turn occurred as a subtree of $nf(\Rightarrow_{R_1 \cup R_2}, \xi_{l_1}) = (\kappa_{h,q,l_1,p}(t))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$, we have that there is an occurrence of a $p_k(y_{h_k}, \cdots)$-call in $\kappa_{h,q,l_1,p}(t)$, which indeed gives us $(h, p) \prec_{q,t} (h_k, p_k)$.

Consequently, we have $(h, p) \prec_{q,t} (h, p)$. By Definition A.6, this means that there must be an $l \in [s]$, such that $\kappa_{h,q,l,p}(t)$ contains an occurrence of a $p(y_h, \cdots)$-call. By Theorem A.4, we have $nf(\Rightarrow_{R_1 \cup R_2}, \xi_l) = (\kappa_{h,q,l,p}(t))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$. This normal form now must contain a $p(y_h, {\xi'}_1, \ldots, {\xi'}_s)$-call with ${\xi'}_1, \ldots, {\xi'}_s \in SUS(r, Z_{s'})$. Since this call does occur in a $\Rightarrow_{R_1 \cup R_2}$-reduction of $p'(q(t, y_1, \ldots, y_r), z_1, \ldots, z_{s'})$, we have again by Theorem A.4: $nf(\Rightarrow_{R_1 \cup R_2}, \xi'_l) = (\kappa_{h,q,l,p}(t))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}]$. Since $\xi'_l \in SUS(r, Z_{s'})$ is a $\Rightarrow_{R_1 \cup R_2}$-normal form (by the remark below Definition A.1), this implies that $\xi'_l = (\kappa_{h,q,l,p}(t))[z_{p',1}, \ldots, z_{p',s'} \leftarrow z_1, \ldots, z_{s'}] = nf(\Rightarrow_{R_1 \cup R_2}, \xi_l)$. But ξ'_l was shown above to be a proper subtree of $nf(\Rightarrow_{R_1 \cup R_2}, \xi_l)$, which leads to a contradiction. $\square$

The previous lemma is weaker than Claim A.7, because it only tells something about instances $p(y_h, \cdots)$ that can actually result from reduction of $p'(q(t, y_1, \ldots, y_r), z_1, \ldots, z_{s'})$ for some state p' of M_2. The technique from the previous proof (using the semantics of $\kappa_{h,q,l,p}$-functions) could not be used for proving Claim A.7, because Theorem A.4 is not sufficient as complete characterization of the $\kappa_{h,q,l,p}$-functions, as it is also only concerned with constellations that actually can occur for given t and q. On the other hand, the proposition of Lemma A.8 is strong enough for our purposes, because for the Macro Tree Transducer $M_{1;2}$ under construction it is, of course, sufficient to simulate correctly the computations of M_1 and M_2 in all actually occurring constellations. In fact, we could replace all later uses of Claim A.7 by references to Lemma A.8 by imposing appropriate restrictions. This, however would require a considerable overhead of auxiliary notions and, more disturbingly, blur the already complex proof of our main Lemma A.14 beyond recognition, so we will instead use Claim A.7 and refer to the proof of Lemma A.8 for justification.

Next, we define an auxiliary rewriting system Pre, which simulates the use of provided translations of context parameters of M_1, as in the first defining equation of the $trans_p$-functions for states of M_2 on page 22.

Definition A.9 We define the rewriting system Pre, which contains for every $h, s \in \mathbf{N}$ with $p \in P^{(s+1)}$ the rule $p(y_h, z_1, \ldots, z_s) \rightarrow y_{h,p}$. Note, that here only the $z_1, \ldots, z_s$ are considered as variables of the rewriting system, whereas the y's are treated as ordinary symbols. $\square$

We will use the following property of the rewriting system Pre.

Proposition A.10 For every $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $t \in T_\Sigma$, $h_1, h_2 \in [a]$ and $p_1, p_2 \in P$, if for some $l \in [rank_P(p_1) - 1]$, the normal form $nf(\Rightarrow_{Pre}, \kappa_{h_1,q,l,p_1}(t))$ contains y_{h_2,p_2}, then $(h_1, p_1) \prec_{q,t} (h_2, p_2)$. $\square$

Now, we prove an important technical lemma about the correctness of precomputing translations of the first transducer's context parameter values with states of M_2 (given the correct context parameter values of these states on reaching M_1's context parameter positions), and then providing these translations in the appropriate positions of the result obtained by computing in ignorance of M_1's concrete context parameter values. The intuition, why this lemma holds, stems from the $\kappa_{h,q,1,p'}, \ldots, \kappa_{h,q,s',p'}$-functions computing the question mark positions as given in Figure A.2.

Lemma A.11 For every $a, s, r \in \mathbf{N}$, $q \in Q^{(a+1)}$, $p \in P^{(s+1)}$, $\theta_1, \ldots, \theta_a \in T_\Delta(Y_r)$, $t \in T_\Sigma$ and $\eta_{p_1,1}, \ldots, \eta_{p_m,s_m}$, such that for every $s' \in \mathbf{N}$, $p' \in P^{(s'+1)}$ and $l' \in [s']$, we have $\eta_{p',l'} = (\text{if } p' = p \text{ then } z_{l'} \text{ else } nil)$, the following holds:

$$
\begin{aligned}
&(nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t, y_1, \ldots, y_a)), z_1, \ldots, z_s))))\\
&\quad [y_{h,p'} \leftarrow (nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_h, \kappa_{h,q,1,p'}(t), \ldots, \kappa_{h,q,s',p'}(t))[y_1, \ldots, y_a \leftarrow \theta_1, \ldots, \theta_a])))\\
&\qquad\qquad [z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}],\\
&\qquad\quad 1 \leq h \leq a, s' \in \mathbf{N}, p' \in P^{(s'+1)}] =\\
&nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t, \theta_1, \ldots, \theta_a)), z_1, \ldots, z_s))) \qquad \square
\end{aligned}
$$

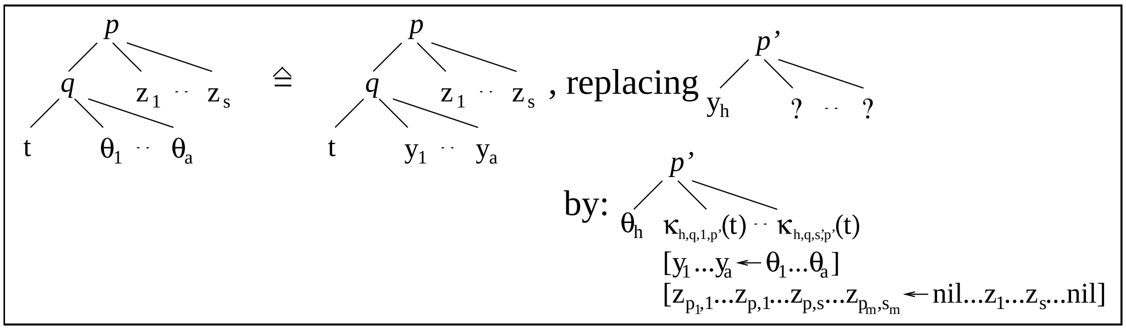

Figure A.2: The intuition behind Lemma A.11

Proof:

- First, consider the normal form $nf(\Rightarrow_{R_1\cup R_2}, p(q(t, y_1, \ldots, y_a), z_1, \ldots, z_s))$. We are going to prove by structural induction that for every subtree $\xi \in SUS(a, Z_s)$ of this normal form, the following holds:
$$(nf(\Rightarrow_{Pre}, \xi))[y_{h,p'} \leftarrow (nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_h, \kappa_{h,q,1,p'}(t), \ldots, \kappa_{h,q,s',p'}(t))[y_1, \ldots, y_a \leftarrow \theta_1, \ldots, \theta_a])))[z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}], 1 \le h \le a, s' \in \mathbf{N}, p' \in P^{(s'+1)}] =$$
$$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, \xi[y_1, \ldots, y_a \leftarrow \theta_1, \ldots, \theta_a]))$$
The cases $\xi \in Z_s$ and $\xi = \omega(\xi_1, \ldots, \xi_n)$ (for $n \in \mathbf{N}$, $\omega \in \Omega^{(n)}$ and $\xi_1, \ldots, \xi_n \in SUS(a, Z_s)$) are straightforward and thus omitted here.
In the case $\xi = p''(y_{h'}, \xi_1, \ldots, \xi_{s''})$ with $h' \in [a]$, $s'' \in \mathbf{N}$, $p'' \in P^{(s''+1)}$ and $\xi_1, \ldots, \xi_{s''} \in SUS(a, Z_s)$, we reason as follows. Since $p''(y_{h'}, \xi_1, \ldots, \xi_{s''})$ is a subtree of $nf(\Rightarrow_{R_1\cup R_2}, p(q(t, y_1, \ldots, y_a), z_1, \ldots, z_s))$, we can apply Theorem A.4 to obtain for every $l \in [s'']$ that $\kappa_{h',q,l,p''}(t) \in SUS(a, \{z_{p,1}, \ldots, z_{p,s}\})$ and $nf(\Rightarrow_{R_1\cup R_2}, \xi_l) = (\kappa_{h',q,l,p''}(t))[z_{p,1}, \ldots, z_{p,s} \leftarrow z_1, \ldots, z_s]$. As subtree of a normal form, ξ_l is itself a normal form, and we can calculate for every $l \in [s'']$:
$$(\kappa_{h',q,l,p''}(t))[y_1, \ldots, y_a \leftarrow \theta_1, \ldots, \theta_a][z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}]$$
$=$ (since the $\theta_1, \ldots, \theta_a$ do not contain any variables from Z_P, $\kappa_{h',q,l,p''}(t) \in SUS(a, \{z_{p,1}, \ldots, z_{p,s}\})$ contains no variables from $Z_P\backslash\{z_{p,1}, \ldots, z_{p,s}\}$, and $\eta_{p,1}, \ldots, \eta_{p,s} = z_1, \ldots, z_s$)
$$(\kappa_{h',q,l,p''}(t))[z_{p,1}, \ldots, z_{p,s} \leftarrow z_1, \ldots, z_s][y_1, \ldots, y_a \leftarrow \theta_1, \ldots, \theta_a]$$
$=$ (by $\xi_l = (\kappa_{h',q,l,p''}(t))[z_{p,1}, \ldots, z_{p,s} \leftarrow z_1, \ldots, z_s]$)
$$\xi_l[y_1, \ldots, y_a \leftarrow \theta_1, \ldots, \theta_a]$$
Now, we can calculate:
$$(nf(\Rightarrow_{Pre}, p''(y_{h'}, \xi_1, \ldots, \xi_{s''})))[y_{h,p'} \leftarrow (nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_h, \kappa_{h,q,1,p'}(t), \ldots, \kappa_{h,q,s',p'}(t))[y_1, \ldots, y_a \leftarrow \theta_1, \ldots, \theta_a])))[z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}], 1 \le h \le a, s' \in \mathbf{N}, p' \in P^{(s'+1)}]$$
$=$ (by $nf(\Rightarrow_{Pre}, p''(y_{h'}, \xi_1, \ldots, \xi_{s''})) = y_{h',p''}$)
$$(nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p''(y_{h'}, \kappa_{h',q,1,p''}(t), \ldots, \kappa_{h',q,s'',p''}(t))[y_1, \ldots, y_a \leftarrow \theta_1, \ldots, \theta_a])))[z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}]$$
$=$ (by substitution)

$$\begin{aligned}
& nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p''(\theta_{h'}, (\kappa_{h',q,1,p''}(t))[y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a] \\
& \qquad\qquad\qquad\qquad\qquad\qquad [z_{p_1,1},\ldots,z_{p_m,s_m} \leftarrow \eta_{p_1,1},\ldots,\eta_{p_m,s_m}], \\
& \qquad\qquad\qquad\qquad\qquad \ldots, \\
& \qquad\qquad\qquad\qquad\qquad (\kappa_{h',q,s'',p''}(t))[y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a] \\
& \qquad\qquad\qquad\qquad\qquad\qquad [z_{p_1,1},\ldots,z_{p_m,s_m} \leftarrow \eta_{p_1,1},\ldots,\eta_{p_m,s_m}]))) \\
& \quad = \text{(by the equivalence shown above for } \xi_l[y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a],\ l \in [s'']) \\
& nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p''(\theta_{h'}, \xi_1[y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a], \\
& \qquad\qquad\qquad\qquad\qquad \ldots, \\
& \qquad\qquad\qquad\qquad\qquad \xi_{s''}[y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a]))) \\
& \quad = \text{(by substitution)} \\
& nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p''(y_{h'}, \xi_1,\ldots,\xi_{s''})[y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a]))
\end{aligned}$$

- By confluence considerations, we know that:
  $nf(\Rightarrow_{R_1\cup R_2}, p(q(t,y_1,\ldots,y_a), z_1,\ldots,z_s)) =$
  $nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t,y_1,\ldots,y_a)), z_1,\ldots,z_s))$
  The lemma is now proven by the following calculation:
  $$\begin{aligned}
  & (nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t,y_1,\ldots,y_a)), z_1,\ldots,z_s)))) \\
  & \quad [y_{h,p'} \leftarrow (nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_h, \kappa_{h,q,1,p'}(t),\ldots,\kappa_{h,q,s',p'}(t)) \\
  & \qquad\qquad\qquad\qquad\qquad\qquad [y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a]))) \\
  & \qquad\qquad [z_{p_1,1},\ldots,z_{p_m,s_m} \leftarrow \eta_{p_1,1},\ldots,\eta_{p_m,s_m}], \\
  & \qquad\qquad 1 \le h \le a, s' \in \mathbf{N}, p' \in P^{(s'+1)}] \\
  & \quad = \text{(by the proposition proven in the first point,} \\
  & \qquad\quad \text{for } \xi = nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t,y_1,\ldots,y_a)), z_1,\ldots,z_s))) \\
  & nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, (nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t,y_1,\ldots,y_a)), z_1,\ldots,z_s))) \\
  & \qquad\qquad\qquad\qquad [y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a])) \\
  & \quad = \text{(by confluence considerations)} \\
  & nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t,\theta_1,\ldots,\theta_a)), z_1,\ldots,z_s)))
  \end{aligned}$$

□

The following claim states a similar property as the previous lemma, but for suspended forms obtained by $\kappa_{h,q,l,p}$-functions. A restricted version of this claim could be proven using the semantics of $\kappa_{h,q,l,p}$-functions given by Theorem A.4 (both the necessary restriction and the proof technique of repeatedly applying Theorem A.4—to determine the appropriate normal forms—are very similar to Lemma A.8).

Claim A.12 For every $a,s,r \in \mathbf{N}$, $q \in Q^{(a+1)}$, $p \in P^{(s+1)}$, $h \in [a]$, $l \in [s]$, $\theta_1,\ldots,\theta_a \in T_\Delta(Y_r)$ and $t \in T_\Sigma$ the following holds:

$$\begin{aligned}
& (nf(\Rightarrow_{Pre}, \kappa_{h,q,l,p}(t))) \\
& \quad [y_{h',p'} \leftarrow nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_{h'}, \kappa_{h',q,1,p'}(t),\ldots,\kappa_{h',q,s',p'}(t))[y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a])), \\
& \qquad\quad 1 \le h' \le a, s' \in \mathbf{N}, p' \in P^{(s'+1)}] = \\
& nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, (\kappa_{h,q,l,p}(t))[y_1,\ldots,y_a \leftarrow \theta_1,\ldots,\theta_a]))
\end{aligned}$$

□

The remarks given below the proof of Lemma A.8 also apply here, so we will use the unrestricted version of Claim A.12 henceforth.

Before we come to the main proof of this appendix, we need to establish three facts about the $trans_p$-, $thru$- and $unfold_{q(x_i)}$-functions, which are very similar to the substitution lemma proven in Section 2.3.

Proposition A.13 For every $k, r \in \mathbf{N}$ and finite set of variables V holds:

1. for every $s \in \mathbf{N}$, $p \in P^{(s+1)}$, $\phi \in RHS(Q, \Delta, k, Y_r)$ and $\eta_1, \ldots, \eta_s \in RHS_{k,r,V}$:
$trans_{p,k,r,V}(\phi, \eta_1, \ldots, \eta_s) =$
$(trans_{p,k,r,Z_s}(\phi, z_1, \ldots, z_s))[z_1, \ldots, z_s \leftarrow \eta_1, \ldots, \eta_s]$

2. for every $a \in rank(\Delta)$, $s+1 \in rank(P)$ or $a = 1$, $s = 0$, $\psi \in RHS(P, \Omega, a, Y_s)$, $\phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$ and $\eta_1, \ldots, \eta_s \in RHS_{k,r,V}$:
$thru_{a,s,k,r,V}(\psi, \phi_1, \ldots, \phi_a, \eta_1, \ldots, \eta_s) =$
$(thru_{a,s,k,r,Z_s}(\psi, \phi_1, \ldots, \phi_a, z_1, \ldots, z_s))[z_1, \ldots, z_s \leftarrow \eta_1, \ldots, \eta_s]$

3. for every $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $x_i \in X_k$, $h \in [a]$, $p \in P$, $\mathcal{S} \subseteq [a] \times P$, $\phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$ and $\eta_{p_1,1}, \ldots, \eta_{p_m,s_m} \in RHS_{k,r,V}$:
$unfold_{q(x_i),k,r,V}(h, p, \mathcal{S}, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}) =$
$(unfold_{q(x_i),k,r,Z_P}(h, p, \mathcal{S}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}))$
$[z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}]$

□

Now, we are going to present and prove the main lemma of this appendix. It consists of six statements that will be proven by nested simultaneous induction. In order to illustrate the proof structure, the statements are arranged in a nested fashion. We have a simultaneous induction of statements **(I)** and **(II)** for trees over the ranked alphabet Σ. Nested inside the first part of statement **(II)**, we have two statements **(i)** and **(ii)** that will also be proven by simultaneous induction, but for all possible right-hand sides built from states and output symbols of M_1 instead of all input trees[1]. The proof structure and dependencies used in the inductive argument are shown in Figure A.3.

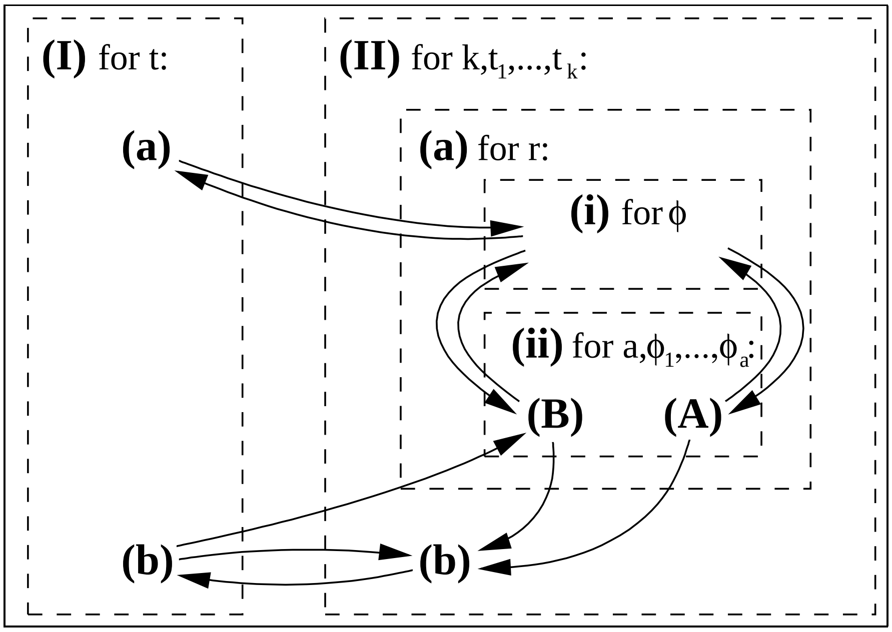

Figure A.3: Proof structure for Lemma A.14

[1]The principle of proof by simultaneous induction can be straightforwardly extended to this setting, namely by following the recursive structure of right-hand sides, as given in Definition 2.2.

Lemma A.14

(I) For $\mathbf{t \in T_\Sigma}$:

(a) for every $r, s \in \mathbf{N}$, $q \in Q^{(r+1)}$ and $p \in P^{(s+1)}$ holds:

$nf(\Rightarrow_{R_{1;2}}, (q,p)(t, y_{1,p_1}, \dots, y_{r,p_m}, z_1, \dots, z_s)) =$
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t, y_1, \dots, y_r)), z_1, \dots, z_s)))$

(b) for every $r, s \in \mathbf{N}$, $q \in Q^{(r+1)}$, $p \in P^{(s+1)}$, $h \in [r]$ and $l \in [s]$ holds:

$nf(\Rightarrow_{R_{1;2}}, (h_q, l_p)(t, y_{1,p_1}, \dots, y_{r,p_m}, z_{p_1,1}, \dots, z_{p_m,s_m})) = nf(\Rightarrow_{Pre}, \kappa_{h,q,l,p}(t))$

(II) For $\mathbf{k \in N, t_1, \dots, t_k \in T_\Sigma}$:

(a) for every $r \in \mathbf{N}$ holds:

(i) for $\phi \in \mathbf{RHS(Q, \Delta, k, Y_r)}$:

for every $s \in \mathbf{N}$ and $p \in P^{(s+1)}$ holds:

$nf(\Rightarrow_{R_{1;2}}, (trans_{p,k,r,Z_s}(\phi, z_1, \dots, z_s))[x_1, \dots, x_k \leftarrow t_1, \dots, t_k]) =$
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, \phi[x_1, \dots, x_k \leftarrow t_1, \dots, t_k]), z_1, \dots, z_s)))$

(ii) for $\mathbf{a \in N}, \phi_1, \dots, \phi_a \in \mathbf{RHS(Q, \Delta, k, Y_r)}$:

(A) for every $s \in \mathbf{N}$ ($a \in rank(\Delta)$, $s+1 \in rank(P)$ or $a = 1$, $s = 0$) and $\psi \in RHS(P, \Omega, a, Y_s)$ holds:

$nf(\Rightarrow_{R_{1;2}}, (thru_{a,s,k,r,Z_s}(\psi, \phi_1, \dots, \phi_a, z_1, \dots, z_s))[x_1, \dots, x_k \leftarrow t_1, \dots, t_k]) =$
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, \psi[x_u \leftarrow nf(\Rightarrow_{R_1}, \phi_u[x_1, \dots, x_k \leftarrow t_1, \dots, t_k]), 1 \le u \le a,$
$y_1, \dots, y_s \leftarrow z_1, \dots, z_s]))$

(B) for every $i \in [k]$, $q \in Q^{(a+1)}$, $\mathcal{S} \subseteq [a] \times P$, $h \in [a]$, $s \in \mathbf{N}$ and $p \in P^{(s+1)}$, if not $(h, p) \prec^*_{q,t_i} \mathcal{S}$, then:

$nf(\Rightarrow_{R_{1;2}}, (unfold_{q(x_i),k,r,Z_P}(h, p, \mathcal{S}, \phi_1, \dots, \phi_a, z_{p_1,1}, \dots, z_{p_m,s_m}))$
$[x_1, \dots, x_k \leftarrow t_1, \dots, t_k]) =$
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(y_h, \kappa_{h,q,1,p}(t_i), \dots, \kappa_{h,q,s,p}(t_i))$
$[y_u \leftarrow nf(\Rightarrow_{R_1}, \phi_u[x_1, \dots, x_k \leftarrow t_1, \dots, t_k]),$
$1 \le u \le a]))$

(b) for every $r \in \mathbf{N}$, $\phi \in RHS(Q, \Delta, k, Y_r)$, $o \in occ(\phi)$, such that $lab(\phi, o) \notin X_k$, $s \in \mathbf{N}$, $p \in P^{(s+1)}$ and $l \in [s]$:

$nf(\Rightarrow_{R_{1;2}}, (ctx_{p,\phi,k,r}(o, l))[x_1, \dots, x_k \leftarrow t_1, \dots, t_k]) =$
$nf(\Rightarrow_{Pre}, \pi_{p,\phi,k,r}(t_1, \dots, t_k, o, l))$

□

Proof: We prove **(I)** and **(II)** by simultaneous induction:

(I) $\Leftarrow$ (II) : Let $t = \sigma(t_1, \dots, t_k)$ with $k \in \mathbf{N}$, $\sigma \in \Sigma^{(k)}$ and $t_1, \dots, t_k \in T_\Sigma$, then:

(a) $nf(\Rightarrow_{R_{1;2}}, (q,p)(\sigma(t_1, \dots, t_k), y_{1,p_1}, \dots, y_{r,p_m}, z_1, \dots, z_s))$
= (by $\Rightarrow_{R_{1;2}}$)
$nf(\Rightarrow_{R_{1;2}}, rhs_{(q,p),\sigma}[x_1, \dots, x_k \leftarrow t_1, \dots, t_k])$
= (by definition of $rhs_{(q,p),\sigma}$ in Construction 3.2)
$nf(\Rightarrow_{R_{1;2}}, (trans_{p,k,r,Z_s}(rhs_{q,\sigma}, z_1, \dots, z_s))[x_1, \dots, x_k \leftarrow t_1, \dots, t_k])$
= (by induction hypothesis **(II)(a)(i)** for $\phi = rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$)
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, rhs_{q,\sigma}[x_1, \dots, x_k \leftarrow t_1, \dots, t_k]), z_1, \dots, z_s)))$
= (by $\Rightarrow_{R_1}$)
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(\sigma(t_1, \dots, t_k), y_1, \dots, y_r)), z_1, \dots, z_s)))$

(b) We have by $\Rightarrow_{R_{1;2}}$:

$nf(\Rightarrow_{R_{1;2}}, (h_q, l_p)(\sigma(t_1, \dots, t_k), y_{1,p_1}, \dots, y_{r,p_m}, z_{p_1,1}, \dots, z_{p_m,s_m})) =$
$nf(\Rightarrow_{R_{1;2}}, rhs_{(h_q,l_p),\sigma}[x_1, \dots, x_k \leftarrow t_1, \dots, t_k])$

Now, there are two possible cases:

- If $rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$ does *not* contain the context variable y_h, then by Construction 3.2, we have $rhs_{(h_q,l_p),\sigma} = nil$ and thus:
  $nf(\Rightarrow_{R_{1;2}}, rhs_{(h_q,l_p),\sigma}[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])$
  $\quad = \;$ (by $rhs_{(h_q,l_p),\sigma} = nil = nf(\Rightarrow_{Pre}, nil)$)
  $nf(\Rightarrow_{Pre}, nil)$
  $\quad = \;$ (by definition of $\kappa_{h,q,l,p}$)
  $nf(\Rightarrow_{Pre}, \kappa_{h,q,l,p}(\sigma(t_1, \ldots, t_k)))$

- If $rhs_{q,\sigma} \in RHS(Q, \Delta, k, Y_r)$ *does* contain the context variable y_h, and the unique such occurrence (M_1 is non-copying!) is $o \in occ(rhs_{q,\sigma})$, then we have:
  $nf(\Rightarrow_{R_{1;2}}, rhs_{(h_q,l_p),\sigma}[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])$
  $\quad = \;$ (by definition of $rhs_{(h_q,l_p),\sigma}$ in Construction 3.2)
  $nf(\Rightarrow_{R_{1;2}}, (ctx_{p,rhs_{q,\sigma},k,r}(o,l))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])$
  $\quad = \;$ (by induction hypothesis **(II)(b)** for $\phi = rhs_{q,\sigma}$)
  $nf(\Rightarrow_{Pre}, \pi_{p,rhs_{q,\sigma},k,r}(t_1, \ldots, t_k, o, l))$
  $\quad = \;$ (by definition of $\kappa_{h,q,l,p}$)
  $nf(\Rightarrow_{Pre}, \kappa_{h,q,l,p}(\sigma(t_1, \ldots, t_k)))$

(II) $\Leftarrow$ **(I) :** We first prove **(a)** by simultaneous induction of **(i)** and **(ii)**, then we can use **(a)** for proving **(b)**. In doing so, we can assume **(I)** for the fixed $t_1, \ldots, t_k$.

(a) simultaneous induction of **(i)** and **(ii)**

(i) $\Leftarrow$ **(ii) :** by cases of $\phi \in RHS(Q, \Delta, k, Y_r)$:

$\phi = y_h \in Y_r$:

$nf(\Rightarrow_{R_{1;2}}, (trans_{p,k,r,Z_s}(y_h, z_1, \ldots, z_s))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])$
$\quad = \;$ (by definition of $trans_{p,k,r,Z_s}$)
$y_{h,p}$
$\quad = \;$ (by definition of Pre and reduction)
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(y_h, z_1, \ldots, z_s)))$
$\quad = \;$ (by substitution and reduction)
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, y_h[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), z_1, \ldots, z_s)))$

$\phi = \delta(\phi_1, \ldots, \phi_a), a \in \mathbf{N}, \delta \in \Delta^{(a)}, \phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$:

$nf(\Rightarrow_{R_{1;2}}, (trans_{p,k,r,Z_s}(\delta(\phi_1, \ldots, \phi_a), z_1, \ldots, z_s))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])$
$\quad = \;$ (by definition of $trans_{p,k,r,Z_s}$)
$nf(\Rightarrow_{R_{1;2}}, (thru_{a,s,k,r,Z_s}(rhs_{p,\delta}, \phi_1, \ldots, \phi_a, z_1, \ldots, z_s))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])$
$\quad = \;$ (by induction hypothesis **(ii)(A)** for $\psi = rhs_{p,\delta} \in RHS(P, \Omega, a, Y_s)$)
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, rhs_{p,\delta}[x_u \leftarrow nf(\Rightarrow_{R_1}, \phi_u[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), 1 \leq u \leq a,$
$\qquad y_1, \ldots, y_s \leftarrow z_1, \ldots, z_s]))$
$\quad = \;$ (by reduction with $\Rightarrow_{R_2}$)
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(\delta(nf(\Rightarrow_{R_1}, \phi_1[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]),$
$\qquad \ldots,$
$\qquad nf(\Rightarrow_{R_1}, \phi_a[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])), z_1, \ldots, z_s)))$
$\quad = \;$ (by substitution and normal form)
$nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, \delta(\phi_1, \ldots, \phi_a)[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]),$
$\qquad z_1, \ldots, z_s)))$

$\phi = q(x_i, \phi_1, \ldots, \phi_a), a \in \mathbf{N}, q \in Q^{(a+1)}, x_i \in X_k, \phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$:
By definition of $trans_{p,k,r,Z_s}$, we have:

$$\begin{aligned}
&nf(\Rightarrow_{R_{1;2}}, (trans_{p,k,r,Z_s}(q(x_i, \phi_1, \ldots, \phi_a), z_1, \ldots, z_s))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) = \\
&nf(\Rightarrow_{R_{1;2}}, (q,p)(x_i, unfold_{q(x_i),k,r,Z_s}(1, p_1, \emptyset, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}), \\
&\qquad\qquad \ldots, \\
&\qquad\qquad unfold_{q(x_i),k,r,Z_s}(a, p_m, \emptyset, \phi_1, \ldots, \phi_a, \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}), \\
&\qquad z_1, \ldots, z_s)[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]),
\end{aligned}$$

where for every $s' \in \mathbf{N}$, $p' \in P^{(s'+1)}$ and $l' \in [s']$, we have $\eta_{p',l'} =$ (if $p' = p$ then $z_{l'}$ else nil). By Proposition A.13(3.) this is equivalent to:

$$\begin{aligned}
&nf(\Rightarrow_{R_{1;2}}, (q,p)(x_i, (unfold_{q(x_i),k,r,Z_P}(1, p_1, \emptyset, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
&\qquad\qquad [z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}], \\
&\qquad\qquad \ldots, \\
&\qquad\qquad (unfold_{q(x_i),k,r,Z_P}(a, p_m, \emptyset, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
&\qquad\qquad [z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}], \\
&\qquad z_1, \ldots, z_s)[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
&= \text{(by confluence considerations and definition of the } \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}) \\
&(nf(\Rightarrow_{R_{1;2}}, (q,p)(t_i, y_{1,p_1}, \ldots, y_{a,p_m}, z_1, \ldots, z_s))) \\
&\quad [y_{h,p'} \leftarrow (nf(\Rightarrow_{R_{1;2}}, (unfold_{q(x_i),k,r,Z_P}(h, p', \emptyset, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
&\qquad\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])) \\
&\qquad [z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}], \\
&\qquad 1 \le h \le a, s' \in \mathbf{N}, p' \in P^{(s'+1)}] \\
&= \text{(by induction hypothesis } \mathbf{(I)(a)} \text{ for } t_i \text{ and} \\
&\qquad \text{induction hypothesis } \mathbf{(ii)(B)} \text{ on the normal forms replaced for the } y_{h,p'}) \\
&(nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t_i, y_1, \ldots, y_a)), z_1, \ldots, z_s)))) \\
&\quad [y_{h,p'} \leftarrow (nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_h, \kappa_{h,q,1,p'}(t_i), \ldots, \kappa_{h,q,s',p'}(t_i)) \\
&\qquad\qquad [y_u \leftarrow nf(\Rightarrow_{R_1}, \phi_u[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
&\qquad\qquad 1 \le u \le a]))) \\
&\qquad [z_{p_1,1}, \ldots, z_{p_m,s_m} \leftarrow \eta_{p_1,1}, \ldots, \eta_{p_m,s_m}], \\
&\qquad 1 \le h \le a, s' \in \mathbf{N}, p' \in P^{(s'+1)}] \\
&= \text{(by Lemma A.11 with } \theta_u = nf(\Rightarrow_{R_1}, \phi_u[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), u \in [a]) \\
&nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(t_i, nf(\Rightarrow_{R_1}, \phi_1[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
&\qquad\qquad \ldots, \\
&\qquad\qquad nf(\Rightarrow_{R_1}, \phi_a[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]))), \\
&\qquad z_1, \ldots, z_s))) \\
&= \text{(by substitution and confluence of } \Rightarrow_{R_1}) \\
&nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, q(x_i, \phi_1, \ldots, \phi_a)[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
&\qquad z_1, \ldots, z_s)))
\end{aligned}$$

(ii) $\Leftarrow$ **(i) :**

(A) a straightforward structural induction on $\psi \in RHS(P, \Omega, a, Y_s)$, using induction hypotheses **(i)** on ϕ_i in the case $\psi = p(x_i, \cdots)$ for $1 \le i \le a$

(B) For fixed $i \in [k]$ and $q \in Q^{(a+1)}$, we proceed by proving the statement for every $\mathcal{S} \in \mathcal{P}([a] \times P)$, by induction on the reversed subset-order:

$\mathcal{S} = [a] \times P$:
In this case, we have $(h,p) \prec^*_{q,t_i} \mathcal{S}$ for every $h \in [a]$ and $p \in P$, thus nothing remains to prove.

$\mathcal{S} \subset [a] \times P$:
If not $(h,p) \prec^*_{q,t_i} \mathcal{S}$, then we know $(h,p) \notin \mathcal{S}$ and thus:

$$
\begin{array}{l}
nf(\Rightarrow_{R_{1;2}}, (\mathit{unfold}_{q(x_i),k,r,Z_P}(h, p, \mathcal{S}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
\quad = \text{ (by definition of } \mathit{unfold}_{q(x_i),k,r,Z_P}) \\
nf(\Rightarrow_{R_{1;2}}, (\mathit{trans}_{p,k,r,Z_P}(\phi_h, \\
\ (h_q, 1_p)(x_i, \mathit{unfold}_{q(x_i),k,r,Z_P}(1, p_1, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad\qquad \ldots, \\
\qquad\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(a, p_m, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\quad \ldots, \\
\ (h_q, s_p)(x_i, \mathit{unfold}_{q(x_i),k,r,Z_P}(1, p_1, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad\qquad \ldots, \\
\qquad\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(a, p_m, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad z_{p_1,1}, \ldots, z_{p_m,s_m})))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
\quad = \text{ (by Proposition A.13(1.))} \\
nf(\Rightarrow_{R_{1;2}}, (\mathit{trans}_{p,k,r,Z_s}(\phi_h, z_1, \ldots, z_s)) \\
\ \ [z_l \leftarrow (h_q, l_p)(x_i, \\
\qquad\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(1, p_1, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad\qquad \ldots, \\
\qquad\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(a, p_m, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad\quad z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad 1 \leq l \leq s] \\
\ \ [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
\quad = \text{ (by confluence considerations)} \\
(nf(\Rightarrow_{R_{1;2}}, (\mathit{trans}_{p,k,r,Z_s}(\phi_h, z_1, \ldots, z_s))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])) \\
\ \ [z_l \leftarrow nf(\Rightarrow_{R_{1;2}}, (h_q, l_p)(x_i, \\
\qquad\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(1, p_1, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad\qquad \ldots, \\
\qquad\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(a, p_m, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad\quad z_{p_1,1}, \ldots, z_{p_m,s_m})[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
\qquad 1 \leq l \leq s] \\
\quad = \text{ (by induction hypothesis } \mathbf{(i)} \text{ for } \phi_h) \\
(nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, \phi_h[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), z_1, \ldots, z_s)))) \\
\ \ [z_l \leftarrow nf(\Rightarrow_{R_{1;2}}, (h_q, l_p)(x_i, \\
\qquad\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(1, p_1, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad\qquad \ldots, \\
\qquad\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(a, p_m, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad\quad z_{p_1,1}, \ldots, z_{p_m,s_m})[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
\qquad 1 \leq l \leq s]
\end{array}
$$

We name this expression **(*)** and continue to calculate on the terms substituted for the z_l with $l \in [s]$:

$$
\begin{array}{l}
nf(\Rightarrow_{R_{1;2}}, (h_q, l_p)(x_i, \\
\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(1, p_1, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\qquad \ldots, \\
\quad \mathit{unfold}_{q(x_i),k,r,Z_P}(a, p_m, \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
\quad z_{p_1,1}, \ldots, z_{p_m,s_m})[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
\quad = \text{ (by substitution and confluence considerations)}
\end{array}
$$

$$
\begin{array}{l}
(nf(\Rightarrow_{R_{1;2}}, (h_q, l_p)(t_i, y_{1,p_1}, \ldots, y_{a,p_m}, z_{p_1,1}, \ldots, z_{p_m,s_m}))) \\
\quad [y_{h',p'} \leftarrow nf(\Rightarrow_{R_{1;2}}, (\mathit{unfold}_{q(x_i),k,r,Z_P}(h', p', \mathcal{S} \cup \{(h,p)\}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
\qquad\qquad\qquad\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
\qquad\quad 1 \leq h' \leq a, s' \in \mathbf{N}, p' \in P^{(s'+1)}] \\
\quad = \text{ (by induction hypothesis } \mathbf{(I)(b)} \text{ for } t_i) \\
(nf(\Rightarrow_{Pre}, \kappa_{h,q,l,p}(t_i))) \\
\quad [y_{h',p'} \leftarrow nf(\Rightarrow_{R_{1;2}}, (\mathit{unfold}_{q(x_i),k,r,Z_P}(h', p', \mathcal{S} \cup \{(h,p)\}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
\qquad\qquad\qquad\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
\qquad\quad 1 \leq h' \leq a, s' \in \mathbf{N}, p' \in P^{(s'+1)}]
\end{array}
$$

Now, we want to use the induction hypothesis on $\mathcal{S} \cup \{(h,p)\}$, as expressed below[i], in order to show that for every $l \in [s]$, this is equivalent to the following expression **(**)**:

$$
\begin{array}{l}
(nf(\Rightarrow_{Pre}, \kappa_{h,q,l,p}(t_i))) \\
\quad [y_{h',p'} \leftarrow nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_{h'}, \kappa_{h',q,1,p'}(t_i), \ldots, \kappa_{h',q,s',p'}(t_i)) \\
\qquad\qquad\qquad\qquad\qquad [y_u \leftarrow nf(\Rightarrow_{R_1}, \phi_u[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
\qquad\qquad\qquad\qquad\qquad\quad 1 \leq u \leq a])), \\
\qquad\quad 1 \leq h' \leq a, s' \in \mathbf{N}, p' \in P^{(s'+1)}]
\end{array}
$$

Therefore, we need to establish that for every $y_{h',p'}$ occurring in $nf(\Rightarrow_{Pre}, \kappa_{h,q,l,p}(t_i))$, we do not have $(h',p') \prec^*_{q,t_i} \mathcal{S} \cup \{(h,p)\}$. This can be shown by contradiction. Assume that for such a $y_{h',p'}$—for which by Proposition A.10 must hold $(h,p) \prec_{q,t_i} (h',p')$—we have $(h',p') \prec^*_{q,t_i} \mathcal{S} \cup \{(h,p)\}$. This would imply $(h,p) \prec^+_{q,t_i} \mathcal{S} \cup \{(h,p)\}$, which cannot be true, because by the assumption at the very beginning of the present case, we know that not $(h,p) \prec^*_{q,t_i} \mathcal{S}$, and modulo Claim A.7 we know that not $(h,p) \prec^+_{q,t_i} (h,p)$. Let us now return to expression **(*)**. Replacing the terms substituted there for z_l (with $l \in [s]$) by the terms **(**)** (shown above to be equivalent), gives us the following equivalent of **(*)**:

$$
\begin{array}{l}
(nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, \phi_h[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), z_1, \ldots, z_s)))) \\
\quad [z_l \leftarrow (nf(\Rightarrow_{Pre}, \kappa_{h,q,l,p}(t_i))) \\
\qquad\qquad [y_{h',p'} \leftarrow nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_{h'}, \kappa_{h',q,1,p'}(t_i), \ldots, \kappa_{h',q,s',p'}(t_i)) \\
\qquad\qquad\qquad\qquad\qquad [y_u \leftarrow nf(\Rightarrow_{R_1}, \phi_u[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
\qquad\qquad\qquad\qquad\qquad\quad 1 \leq u \leq a])), \\
\qquad\qquad\quad 1 \leq h' \leq a, s' \in \mathbf{N}, p' \in P^{(s'+1)}], \\
\qquad 1 \leq l \leq s] \\
\quad = \text{ (modulo Claim A.12 on the normal forms replaced for } z_1, \ldots, z_s, \\
\qquad \text{with } \theta_u = nf(\Rightarrow_{R_1}, \phi_u[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]),\ u \in [a]) \\
(nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(nf(\Rightarrow_{R_1}, \phi_h[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), z_1, \ldots, z_s)))) \\
\quad [z_l \leftarrow nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, (\kappa_{h,q,l,p}(t_i)) \\
\qquad\qquad\qquad [y_u \leftarrow nf(\Rightarrow_{R_1}, \phi_u[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), 1 \leq u \leq a])), \\
\qquad 1 \leq l \leq s] \\
\quad = \text{ (by confluence considerations and substitution)} \\
nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p(y_h, \kappa_{h,q,1,p}(t_i), \ldots, \kappa_{h,q,s,p}(t_i)) \\
\qquad\qquad\qquad [y_u \leftarrow nf(\Rightarrow_{R_1}, \phi_u[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), 1 \leq u \leq a]))
\end{array}
$$

---

[i]For every $h' \in [a]$, $s' \in \mathbf{N}$ and $p' \in P^{(s'+1)}$, if not $(h',p') \prec^*_{q,t_i} \mathcal{S} \cup \{(h,p)\}$, then:

$$
\begin{array}{l}
nf(\Rightarrow_{R_{1;2}}, (\mathit{unfold}_{q(x_i),k,r,Z_P}(h', p', \mathcal{S} \cup \{(h,p)\}, \phi_1, \ldots, \phi_a, z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) = \\
nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_{h'}, \kappa_{h',q,1,p'}(t_i), \ldots, \kappa_{h',q,s',p'}(t_i)) \\
\qquad\qquad\qquad [y_u \leftarrow nf(\Rightarrow_{R_1}, \phi_u[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), 1 \leq u \leq a]))
\end{array}
$$

(b) In proving **(II)(b)**, we may use the statements from **(II)(a)(ii)**, which have previously been proven for every $a \in \mathbf{N}$, $\phi_1, \ldots, \phi_a \in RHS(Q, \Delta, k, Y_r)$, but for the fixed $t_1, \ldots, t_k$ under consideration in **(II)(b)**. For fixed $r \in \mathbf{N}$ and $\phi \in RHS(Q, \Delta, k, Y_r)$, we proceed by induction on the prefix-order of occurrences. In the base case ϵ, we have for every $s \in \mathbf{N}$, $p \in P^{(s+1)}$ and $l \in [s]$:

$$
\begin{aligned}
& nf(\Rightarrow_{R_{1;2}}, (ctx_{p,\phi,k,r}(\epsilon, l))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
& \quad = \text{ (by definition of } ctx_{p,\phi,k,r}) \\
& z_{p,l} \\
& \quad = \text{ (by definition of } \pi_{p,\phi,k,r}) \\
& nf(\Rightarrow_{Pre}, \pi_{p,\phi,k,r}(t_1, \ldots, t_k, \epsilon, l))
\end{aligned}
$$

The inductive case $oi \in occ(\phi)$—with $o \in occ(\phi)$, $i \in \mathbf{N}$ and $lab(\phi, oi) \notin X_k$—is shown by case distinction on $lab(\phi, o)$.

- $lab(\phi, o) = \delta$ with $a \in \mathbf{N}$, $\delta \in \Delta^{(a)}$ and $i \in [a]$:
  - If there is no $p(x_i)$-call in the δ-rules of M_2, then we have:

    $$
    \begin{aligned}
    & nf(\Rightarrow_{R_{1;2}}, (ctx_{p,\phi,k,r}(oi, l))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
    & \quad = \text{ (by definition of } ctx_{p,\phi,k,r}) \\
    & nil \\
    & \quad = \text{ (by definition of } \pi_{p,\phi,k,r}) \\
    & nf(\Rightarrow_{Pre}, \pi_{p,\phi,k,r}(t_1, \ldots, t_k, oi, l))
    \end{aligned}
    $$

  - Otherwise, if the unique such call (remember that M_2 is weakly single-use) looks, with $s' \in \mathbf{N}$, $p' \in P^{(s'+1)}$ and $\psi_1, \ldots, \psi_s \in RHS(P, \Omega, a, Y_{s'})$, as follows:

    $$p'(\delta(x_1, \ldots, x_a), y_1, \ldots, y_{s'}) \to \cdots p(x_i, \psi_1, \ldots, \psi_s) \cdots,$$

    then we can calculate:

    $$
    \begin{aligned}
    & nf(\Rightarrow_{R_{1;2}}, (ctx_{p,\phi,k,r}(oi, l))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
    & \quad = \text{ (by definition of } ctx_{p,\phi,k,r}) \\
    & nf(\Rightarrow_{R_{1;2}}, (thru_{a,s',k,r,Z_P}(\psi_l, sub(\phi, o1), \ldots, sub(\phi, oa), \\
    & \qquad\qquad ctx_{p',\phi,k,r}(o, 1), \ldots, ctx_{p',\phi,k,r}(o, s'))) \\
    & \qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
    & \quad = \text{ (by Proposition A.13(2.))} \\
    & nf(\Rightarrow_{R_{1;2}}, (thru_{a,s',k,r,Z_{s'}}(\psi_l, sub(\phi, o1), \ldots, sub(\phi, oa), z_1, \ldots, z_{s'})) \\
    & \qquad [z_w \leftarrow ctx_{p',\phi,k,r}(o, w), 1 \leq w \leq s'] \\
    & \qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
    & \quad = \text{ (by confluence considerations)} \\
    & (nf(\Rightarrow_{R_{1;2}}, (thru_{a,s',k,r,Z_{s'}}(\psi_l, sub(\phi, o1), \ldots, sub(\phi, oa), z_1, \ldots, z_{s'})) \\
    & \qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])) \\
    & \quad [z_w \leftarrow nf(\Rightarrow_{R_{1;2}}, (ctx_{p',\phi,k,r}(o, w))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), w \in [s']] \\
    & \quad = \text{ (by the induction hypothesis on } o) \\
    & (nf(\Rightarrow_{R_{1;2}}, (thru_{a,s',k,r,Z_{s'}}(\psi_l, sub(\phi, o1), \ldots, sub(\phi, oa), z_1, \ldots, z_{s'})) \\
    & \qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])) \\
    & \quad [z_w \leftarrow nf(\Rightarrow_{Pre}, \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w)), 1 \leq w \leq s'] \\
    & \quad = \text{ (by the previously proven statement } \textbf{(a)(ii)(A)} \text{ for} \\
    & \qquad sub(\phi, o1), \ldots, sub(\phi, oa) \in RHS(Q, \Delta, k, Y_r), \\
    & \qquad s' + 1 \in rank(P) \text{ and } \psi_l \in RHS(P, \Omega, a, Y_{s'}))
    \end{aligned}
    $$

$$
\begin{aligned}
&(nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]]),\\
&\qquad 1 \le u \le a,\\
&\qquad y_1, \ldots, y_{s'} \leftarrow z_1, \ldots, z_{s'}])))\\
&\quad [z_w \leftarrow nf(\Rightarrow_{Pre}, \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w)), 1 \le w \le s']\\
&\quad = \text{(by confluence considerations)}\\
&nf(\Rightarrow_{Pre}, (nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]]),\\
&\qquad 1 \le u \le a,\\
&\qquad y_1, \ldots, y_{s'} \leftarrow z_1, \ldots, z_{s'}]))\\
&\qquad [z_w \leftarrow \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w), 1 \le w \le s'])\\
&\quad = \text{(since the } \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w) \in SUS(r, Z_P)\text{, for } w \in [s'] \text{, are}\\
&\qquad \Rightarrow_{R_2}\text{-normal forms, by the remark below Definition A.1)}\\
&nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]]),\\
&\qquad 1 \le u \le a,\\
&\qquad y_1, \ldots, y_{s'} \leftarrow z_1, \ldots, z_{s'}]\\
&\qquad [z_w \leftarrow \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w), 1 \le w \le s']))\\
&\quad = \text{(by composing substitutions)}\\
&nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, \psi_l[x_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, ou))[x_v \leftarrow t_v, v \in [k]]),\\
&\qquad 1 \le u \le a,\\
&\qquad y_w \leftarrow \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w), 1 \le w \le s']))\\
&\quad = \text{(by definition of } \pi_{p,\phi,k,r}\text{, taking into account the unique } p(x_i)\text{-call}\\
&\qquad \text{in the } \delta\text{-rules of } M_2 \text{ as given above)}\\
&nf(\Rightarrow_{Pre}, \pi_{p,\phi,k,r}(t_1, \ldots, t_k, oi, l))
\end{aligned}
$$

- $lab(\phi, o) = q$ with $a \in \mathbf{N}$, $q \in Q^{(a+1)}$, $lab(\phi, o1) = x_b \in X_k$ and $2 \le i \le a+1$: We calculate:

$$
\begin{aligned}
&nf(\Rightarrow_{R_{1;2}}, (ctx_{p,\phi,k,r}(oi, l))[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])\\
&\quad = \text{(by definition of } ctx_{p,\phi,k,r})\\
&nf(\Rightarrow_{R_{1;2}}, ((i-1)_q, l_p)(x_b, unfold_{q(x_b),k,r,Z_P}(1, p_1, \{(i-1, p)\},\\
&\qquad sub(\phi, o2), \ldots, sub(\phi, o(a+1)),\\
&\qquad ctx_{p_1,\phi,k,r}(o, 1),\\
&\qquad \ldots,\\
&\qquad ctx_{p_m,\phi,k,r}(o, s_m)),\\
&\qquad \ldots,\\
&\qquad unfold_{q(x_b),k,r,Z_P}(a, p_m, \{(i-1, p)\},\\
&\qquad sub(\phi, o2), \ldots, sub(\phi, o(a+1)),\\
&\qquad ctx_{p_1,\phi,k,r}(o, 1),\\
&\qquad \ldots,\\
&\qquad ctx_{p_m,\phi,k,r}(o, s_m)),\\
&\qquad ctx_{p_1,\phi,k,r}(o, 1), \ldots, ctx_{p_m,\phi,k,r}(o, s_m))\\
&\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k])\\
&\quad = \text{(by Proposition A.13(3.))}
\end{aligned}
$$

$$
\begin{aligned}
& nf(\Rightarrow_{R_{1;2}}, ((i-1)_q, l_p)(x_b, (unfold_{q(x_b),k,r,Z_P}(1, p_1, \{(i-1,p)\}, \\
&\qquad\qquad\qquad\qquad sub(\phi, o2), \ldots, sub(\phi, o(a+1)), \\
&\qquad\qquad\qquad\qquad z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
&\qquad\qquad [z_{p',w} \leftarrow ctx_{p',\phi,k,r}(o,w), s' \in \mathbf{N}, p' \in P^{(s'+1)}, \\
&\qquad\qquad\qquad 1 \le w \le s'], \\
&\qquad\qquad \ldots, \\
&\qquad\qquad (unfold_{q(x_b),k,r,Z_P}(a, p_m, \{(i-1,p)\}, \\
&\qquad\qquad\qquad\qquad sub(\phi, o2), \ldots, sub(\phi, o(a+1)), \\
&\qquad\qquad\qquad\qquad z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
&\qquad\qquad [z_{p',w} \leftarrow ctx_{p',\phi,k,r}(o,w), s' \in \mathbf{N}, p' \in P^{(s'+1)}, \\
&\qquad\qquad\qquad 1 \le w \le s'], \\
&\qquad\qquad ctx_{p_1,\phi,k,r}(o,1), \ldots, ctx_{p_m,\phi,k,r}(o,s_m)) \\
&\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
= &\ \text{(by substitution)} \\
& nf(\Rightarrow_{R_{1;2}}, ((i-1)_q, l_p)(x_b, unfold_{q(x_b),k,r,Z_P}(1, p_1, \{(i-1,p)\}, \\
&\qquad\qquad\qquad\qquad sub(\phi, o2), \ldots, sub(\phi, o(a+1)), \\
&\qquad\qquad\qquad\qquad z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
&\qquad\qquad \ldots, \\
&\qquad\qquad unfold_{q(x_b),k,r,Z_P}(a, p_m, \{(i-1,p)\}, \\
&\qquad\qquad\qquad\qquad sub(\phi, o2), \ldots, sub(\phi, o(a+1)), \\
&\qquad\qquad\qquad\qquad z_{p_1,1}, \ldots, z_{p_m,s_m}), \\
&\qquad\qquad z_{p_1,1}, \ldots, z_{p_m,s_m}) \\
&\qquad [z_{p',w} \leftarrow ctx_{p',\phi,k,r}(o,w), s' \in \mathbf{N}, p' \in P^{(s'+1)}, 1 \le w \le s'] \\
&\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]) \\
= &\ \text{(by substitution and confluence considerations)} \\
& (nf(\Rightarrow_{R_{1;2}}, ((i-1)_q, l_p)(t_b, y_{1,p_1}, \ldots, y_{a,p_m}, z_{p_1,1}, \ldots, z_{p_m,s_m}))) \\
&\quad [y_{h,p'} \leftarrow nf(\Rightarrow_{R_{1;2}}, (unfold_{q(x_b),k,r,Z_P}(h, p', \{(i-1,p)\}, \\
&\qquad\qquad\qquad\qquad sub(\phi, o2), \ldots, sub(\phi, o(a+1)), \\
&\qquad\qquad\qquad\qquad z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
&\qquad\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
&\qquad 1 \le h \le a, s' \in \mathbf{N}, p' \in P^{(s'+1)}] \\
&\quad [z_{p',w} \leftarrow nf(\Rightarrow_{R_{1;2}}, ctx_{p',\phi,k,r}(o,w)[x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
&\qquad s' \in \mathbf{N}, p' \in P^{(s'+1)}, 1 \le w \le s'] \\
= &\ \text{(by induction hypothesis } \mathbf{(I)(b)} \text{ for } t_b \text{ and} \\
&\quad \text{the induction hypothesis on } o) \\
& (nf(\Rightarrow_{Pre}, \kappa_{i-1,q,l,p}(t_b))) \\
&\quad [y_{h,p'} \leftarrow nf(\Rightarrow_{R_{1;2}}, (unfold_{q(x_b),k,r,Z_P}(h, p', \{(i-1,p)\}, \\
&\qquad\qquad\qquad\qquad sub(\phi, o2), \ldots, sub(\phi, o(a+1)), \\
&\qquad\qquad\qquad\qquad z_{p_1,1}, \ldots, z_{p_m,s_m})) \\
&\qquad\qquad [x_1, \ldots, x_k \leftarrow t_1, \ldots, t_k]), \\
&\qquad 1 \le h \le a, s' \in \mathbf{N}, p' \in P^{(s'+1)}] \\
&\quad [z_{p',w} \leftarrow nf(\Rightarrow_{Pre}, \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w)), s' \in \mathbf{N}, p' \in P^{(s'+1)}, 1 \le w \le s']
\end{aligned}
$$

Now, we claim that for all variables $y_{h,p'}$ that might occur in $nf(\Rightarrow_{Pre}, \kappa_{i-1,q,l,p}(t_b))$, we do not have $(h,p') \prec^*_{q,t_b} (i-1,p)$. The reasoning is by contradiction. If for such a $y_{h,p'}$—for which by Proposition A.10 must hold $(i-1,p) \prec_{q,t_b} (h,p')$—we would have $(h,p') \prec^*_{q,t_b} (i-1,p)$, this would imply $(i-1,p) \prec^+_{q,t_b} (i-1,p)$, which by Claim A.7 is impossible. This fact allows us to apply—on the normal forms replaced for the $y_{h,p'}$—the previously proven statement **(a)(ii)(B)** for $sub(\phi, o2), \ldots, sub(\phi, o(a+1)) \in RHS(Q, \Delta, k, Y_r)$ and obtain:

$$
\begin{aligned}
&(nf(\Rightarrow_{Pre}, \kappa_{i-1,q,l,p}(t_b)))\\
&\quad [y_{h,p'} \leftarrow nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, p'(y_h, \kappa_{h,q,1,p'}(t_b), \ldots, \kappa_{h,q,s',p'}(t_b))\\
&\qquad\qquad\qquad\qquad [y_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, o(u+1)))[x_v \leftarrow t_v, v \in [k]]),\\
&\qquad\qquad\qquad\qquad\quad 1 \leq u \leq a])),\\
&\qquad\quad 1 \leq h \leq a, s' \in \mathbf{N}, p' \in P^{(s'+1)}]\\
&\quad [z_{p',w} \leftarrow nf(\Rightarrow_{Pre}, \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w)), s' \in \mathbf{N}, p' \in P^{(s'+1)}, 1 \leq w \leq s']\\
&\quad = \text{(modulo Claim A.12 with}\\
&\qquad \theta_u = nf(\Rightarrow_{R_1}, (sub(\phi, o(u+1)))[x_v \leftarrow t_v, v \in [k]]),\ u \in [a])\\
&(nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, (\kappa_{i-1,q,l,p}(t_b))\\
&\qquad\qquad\qquad [y_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, o(u+1)))[x_v \leftarrow t_v, v \in [k]]),\\
&\qquad\qquad\qquad\quad 1 \leq u \leq a])))\\
&\quad [z_{p',w} \leftarrow nf(\Rightarrow_{Pre}, \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w)), s' \in \mathbf{N}, p' \in P^{(s'+1)}, 1 \leq w \leq s']\\
&\quad = \text{(by confluence considerations)}\\
&nf(\Rightarrow_{Pre}, (nf(\Rightarrow_{R_2}, (\kappa_{i-1,q,l,p}(t_b))\\
&\qquad\qquad\qquad [y_u \leftarrow nf(\Rightarrow_{R_1}, (sub(\phi, o(u+1)))[x_v \leftarrow t_v, v \in [k]]),\\
&\qquad\qquad\qquad\quad 1 \leq u \leq a]))\\
&\quad [z_{p',w} \leftarrow \pi_{p',\phi,k,r}(t_1, \ldots, t_k, o, w),\ s' \in \mathbf{N}, p' \in P^{(s'+1)}, 1 \leq w \leq s'])\\
&\quad = \text{(by definition of } \pi_{p,\phi,k,r})\\
&nf(\Rightarrow_{Pre}, \pi_{p,\phi,k,r}(t_1, \ldots, t_k, oi, l))
\end{aligned}
$$

□

Using the previous lemma, we can now prove the correctness of Construction 3.2.

Theorem A.15 $\tau(M_1); \tau(M_2) = \tau(M_{1;2})$ □

Proof: For every $t \in T_\Sigma$, we calculate:

$$
\begin{aligned}
&\tau(M_{1;2})(t)\\
&\quad = \text{(by Proposition 2.7 for } M_{1;2})\\
&nf(\Rightarrow_{R_{1;2}}, e_{1;2}[x_1 \leftarrow t])\\
&\quad = \text{(by definition of } e_{1;2} \text{ in Construction 3.2)}\\
&nf(\Rightarrow_{R_{1;2}}, (thru_{1,0,1,0,\emptyset}(e_2, e_1))[x_1 \leftarrow t])\\
&\quad = \text{(by statement } \mathbf{(II)(a)(ii)(A)} \text{ of Lemma A.14 for } k=1,\ t_1 = t \in T_\Sigma,\ r=0,\ a=1,\\
&\qquad \phi_1 = e_1 \in RHS(Q, \Delta, 1, \emptyset),\ s=0 \text{ and } \psi = e_2 \in RHS(P, \Omega, 1, \emptyset))\\
&nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, e_2[x_1 \leftarrow nf(\Rightarrow_{R_1}, e_1[x_1 \leftarrow t])]))\\
&\quad = \text{(by Proposition 2.7 for } M_1)\\
&nf(\Rightarrow_{Pre}, nf(\Rightarrow_{R_2}, e_2[x_1 \leftarrow \tau(M_1)(t)]))\\
&\quad = \text{(by Proposition 2.7 for } M_2)\\
&nf(\Rightarrow_{Pre}, \tau(M_2)(\tau(M_1)(t)))\\
&\quad = \text{(since } \tau(M_2)(\tau(M_1)(t)) \in T_\Omega \text{ is a normal form with respect to } \Rightarrow_{Pre})\\
&\tau(M_2)(\tau(M_1)(t))
\end{aligned}
$$

□