Efficient Call-by-value Evaluation Strategy of Primitive Recursive Program Schemes

Andrea Mößle*
Universität Ulm, Fakultät für Informatik, Abt. Programmiermethodik und Compilerbau
E-mail: andrea@bach.informatik.uni-ulm.de

Heiko Vogler
TU Dresden, Fakultät Informatik, Lehrstuhl Grundlagen der Programmierung
E-mail: vogler@inf.tu-dresden.de

Abstract

We consider primitive recursive program schemes with parameters together with the call-by-value computation rule. The schemes are finite systems of functions which are defined by primitive (or: structural) recursion; simultaneous recursion and nesting of function calls is allowed. We present a transformation strategy which replaces primitive recursion by iteration. The transformation strategy which is fully automatic, takes as input a primitive recursive program scheme M with parameters and it computes a program scheme M' as output. We prove that, for every argument tuple, M' is at least as (time) efficient as M. We also prove that there are infinitely many nontrivial primitive recursive program schemes M with parameters for which the transformation yields a program scheme M' such that there are infinitely many argument tuples for which M' is more efficient than M. Moreover, we provide an algorithm which decides for an arbitrary given primitive recursive program scheme M with parameters whether M' is more efficient than M.

1 Introduction

Algorithms can be often elegantly described in a recursive way and consequently, recursion is offered by every modern procedural programming language. Usually, such programming languages are implemented on runtime stack machines which create for every function call an activation block; in such blocks, the return address, static and dynamic links, and the variables of the procedure are stored (see e.g. [Dij60], [McC60]). However, the creation and administration of activation blocks cause a great amount of runtime overhead and hence, great efforts have been made to transform the programs such that recursion is replaced by iteration. Such transformations are quoted as recursion elimination (see, e.g., [Knu74], [Ric65], [Bir77a], [Bir77b], [AS78]).

In functional programming languages, recursion is even the most important way of defining objects. And in fact, during the investigation of functional programming languages, new facettes with respect to recursion elimination appeared [BD77, Bir80, Boi92]. One of the most well-known techniques is the unfold/fold technique of [BD77] which has been refined in many ways [PP93].

*The work of this author was supported by the Deutsche Forschungsgemeinschaft (DFG).

In the context of functional programming languages, the main goal of recursion elimination is to avoid multiple computations of function calls and multiple traversals of subcomponents of function arguments.

Also in the field of program transformation as a discipline to compose programs from an abstract specification by stepwise refinement, the question of producing efficient programs attracted attention [Fea82, BW82, PP86, Par90]. In particular, recursion elimination is addressed. According to [Boi92], the strategies of the cited papers can be divided into the following categories: accumulation, finite differencing (cf., e.g., [PK82]), algorithm theories, and inverting the order of evaluation.

In [Kla88], an alternative way of getting rid of structural recursion in functional programming languages is suggested. Roughly speaking, there the recursion elimination is integrated into the compiler and the resulting iteration is supported by a few new machine components and instructions (also cf. [IK87a, IK87b, IKV90, Thi91a, Thi91b]).

Many of the techniques (e.g., the Unfold/Fold technique [BD77]) are interactive in the sense that they need the help of the user. Often the transformation techniques exploit additional assumptions either about the form of the function definitions or about the domain and the range of the functions (cf., e.g., [Bir80]). Also in some of the proposed solutions the resulting formalism is of completely different nature than the original formalism (cf., e.g., the intelligent compilers or the method to find procedural solutions for recursive function definition).

For a long time, the necessity of proving the correctness of transformation strategies was neglected. Recently, this topic has been dicussed in [San95a, San95b]. There, he presents a tool for proving the correctness of existing transformation methods for higher-order functional programs and a simple syntactic method for guiding and constraining the unfold/fold method.

In this paper we present a transformation strategy which eliminates primitive recursion. The transformation strategy is fully automatic, it does not require any additional assumptions about the domain or the range of the defined functions, it is presented in a formal way, and its correctness is proved. As input the transformation strategy considers primitive recursive program schemes (with parameters) [CF82] equipped with the call-by-value computation rule. Given a primitive recursive program scheme M, the transformation strategy transforms M into a program scheme M'; the transformation is performed by means of a well-defined sequence of transformation relations called splitting, sharing, and tupling denoted by $\vdash_{split,M}$, $\vdash_{share,M}$, and $\vdash_{tuple,M}$, respectively. In fact, the transformation eliminates some of the recursive computations of function values by turning them into iterative computations.

We prove that every call-by-value computation of M' is at least as efficient as the corresponding call-by-value computation of M. Also we prove that there are infinitely many primitive recursive program schemes and infinitely many arguments for such schemes such that the transformation strategy yields more efficient (i.e., shorter) call-by-value computations. Moreover, we present an algorithm which decides for a given primitive recursive program scheme M whether there is an argument for which the call-by-value computation of M' is more efficient than the call-by-value computation of M. We note that the strategy called "safe tupling" which is proposed in [Chi93] and extended in [CK93, CH95], yields similar results. However, there the authors do not show a decision algorithm with the mentioned behaviour.

Since the recursion elimination deals with the concept of computations, it is reasonable to formalize our transformation strategy on formal models for the reduction semantics of primitive recursive program schemes. An appropriate formal model is the macro tree transducer [Eng80] (also cf. [EV85]). Macro tree transducers are particular convergent (i.e., confluent and terminating), left-linear, non-overlapping, constructor based term rewriting systems (cf. [Klo92] for a survey on term rewriting systems). Macro tree transducers also fit into our requirement not to use additional assumptions about the domain and range of the defined functions: the semantic domain of macro tree transducers is the free term algebra.

Now let us explain in more detail how the transformation strategy is defined on tree transducers (cf. Figure 1). In fact, the transformation strategy takes a macro tree transducer M as input

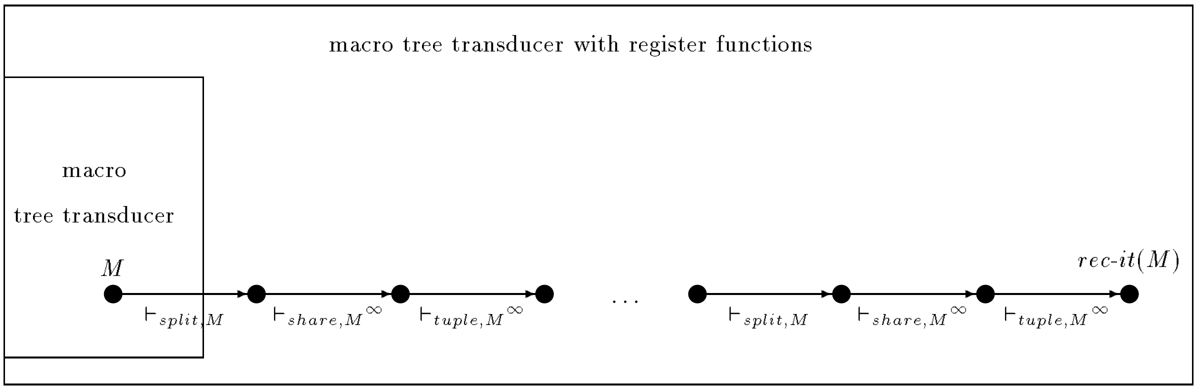


Figure 1: Transformation of a macro tree transducer.

and it considers M as a macro tree transducer with register functions. In such transducers a new type of function may occur which is called register function. Intuitively, the arguments of such functions can serve as registers in which the values of the original functions can be computed iteratively (in the same spirit as, e.g., in the register programs of [AE79] values are accumulated in registers). In fact, all the three mentioned transformation relations which constitute the transformation strategy, i.e., $\vdash_{split,M}$, $\vdash_{share,M}$, and $\vdash_{tuple,M}$, are defined as binary relations over the class of macro tree transducers with register functions. We prove that $\vdash_{split,M}$, $\vdash_{share,M}$, and $\vdash_{tuple,M}$ are semantic preserving, confluent, and noetherian. Then, roughly speaking, our transformation strategy is defined as computing normalforms of a macro tree transducer with respect to the composed relation

$$\vdash_{split,M} \circ \vdash_{share,M}^{\infty} \circ \vdash_{tuple,M}^{\infty},$$

where, e.g., $\vdash_{share,M}^{\infty}$ denotes the relation with all pairs (N, N') such that N' is the normalform of N with respect to $\vdash_{share,M}$ (and similarly for $\vdash_{tuple,M}$). Then M is transformed into the macro tree transducer $rec\text{-}it(M)$ with register functions which is called the *recursive-iterative tree transducer of M*. This name is due to the fact that some of the function values are still computed recursively while others are computed iteratively.

In the rest of this introduction we will 1) show an example M_1 of a macro tree transducer and the corresponding recursive-iterative tree transducer $rec\text{-}it(M_1)$, 2) discuss the origins of our transformation strategy, and 3) outline the structure of this paper.

On first glance, the example might seem a bit too complicated to show the nature of macro tree transducers. However, it is composed such that we can discuss smoothly the origins of the transformation strategy also on the basis of this example.

$$
\begin{array}{rcl}
f_S(p_0(x_1)) & \rightarrow & f_A(x_1, \varepsilon, \varepsilon) \\
f_A(p_1(x_1, x_2), y_1, y_2) & \rightarrow & \#(f_A(x_1, 1(y_1), b(y_2)), f_C(x_2, c(f_A(x_1, 1(y_1), b(y_2))))) \\
f_A(p_2, y_1, y_2) & \rightarrow & \#(y_1, y_2) \\
f_C(p_3, y_1) & \rightarrow & c(y_1)
\end{array}
$$

Figure 2: Rules of the macro tree transducer M_1.

The macro tree transducer M_1 has three functions f_S, f_A, and f_C with arity 1, 3, and 2, respectively. It takes input trees over the ranked alphabet $\{p_0^{(1)}, p_1^{(2)}, p^{(0)}, p_3^{(0)}\}$. The output trees are built up over the ranked alphabet $\{\#^{(2)}, c^{(1)}, 1^{(1)}, b^{(1)}, \varepsilon^{(0)}\}$. Figure 2 shows the rules of M_1. A call-by-value computation of M_1 on the input tree $p_0(p_1(p_2, p_3))$ looks as follows:

$$
\begin{array}{rcl}
f_S(p_0(p_1(p_2, p_3))) & \Rightarrow & f_A(p_1(p_2, p_3), \varepsilon, \varepsilon) \\
& \Rightarrow & \#(f_A(p_2, 1(\varepsilon), b(\varepsilon)), f_C(p_3, c(f_A(p_2, 1(\varepsilon), b(\varepsilon))))) \\
& \Rightarrow^2 & \#(\#(1(\varepsilon), b(\varepsilon)), f_C(p_3, c(\#(1(\varepsilon), b(\varepsilon))))) \\
& \Rightarrow & \#(\#(1(\varepsilon), b(\varepsilon)), c(c(\#(1(\varepsilon), b(\varepsilon)))))
\end{array}
$$

Note that five derivation steps are necessary to compute the result and note that the function call $f_A(p_2, 1(\varepsilon), b(\varepsilon))$ has to be evaluated twice. Also note that the result is computed recursively in a demand driven way (or: top-down manner).

In this paper we will transform the macro tree transducer M_1 into the recursive-iterative tree transducer $\mathit{rec\text{-}it}(M)$ shown in Figure 3 (the function $\langle f_A, p_1, 1\rangle$ is a register function).

$$
\begin{array}{rcl}
f_S(p_0(x_1)) & \rightarrow & f_A(x_1, \varepsilon, \varepsilon) \\
f_A(p_1(x_1, x_2), y_1, y_2) & \rightarrow & \langle f_A, p_1, 1\rangle(x_1, x_2, y_1, y_2, f_A(x_1, 1(y_1), b(y_2))) \\
\langle f_A, p_1, 1\rangle(x_1, x_2, y_1, y_2, z_1) & \rightarrow & \#(z_1, f_C(x_2, c(z_1))) \\
f_A(p_2, y_1, y_2) & \rightarrow & \#(y_1, y_2) \\
f_C(p_3, y_1) & \rightarrow & c(y_1)
\end{array}
$$

Figure 3: Rules of the recursive-iterative tree transducer $\mathit{rec\text{-}it}(M_1)$.

Let us consider the computation of $f_S(p_0(p_1(p_2, p_3)))$ performed by $\mathit{rec\text{-}it}(M_1)$:

$$
\begin{array}{rcl}
f_S(p_0(p_1(p_2, p_3))) & \Rightarrow & f_A(p_1(p_2, p_3), \varepsilon, \varepsilon) \\
& \Rightarrow & \langle f_A, p_1, 1\rangle(p_2, p_3, \varepsilon, \varepsilon, f_A(p_2, 1(\varepsilon), b(\varepsilon))) \\
& \Rightarrow & \langle f_A, p_1, 1\rangle(p_2, p_3, \varepsilon, \varepsilon, \#(1(\varepsilon), b(\varepsilon))) \\
& \Rightarrow & \#(\#(1(\varepsilon), b(\varepsilon)), f_C(p_3, c(\#(1(\varepsilon), b(\varepsilon))))) \\
& \Rightarrow & \#(\#(1(\varepsilon), b(\varepsilon)), c(c(\#(1(\varepsilon), b(\varepsilon)))))
\end{array}
$$

Note that $rec\text{-}it(M_1)$, as M_1, computes the result in five steps. Also note that the result is computed sometimes demand driven and sometimes value driven. The function call $f_A(p_2, 1(\varepsilon), b(\varepsilon))$ is only evaluted once.

In fact, for every input tree the evaluation of $rec\text{-}it(M_1)$ is at least as efficient as the evaluation of M_1. To illustrate this, let us consider input trees t_n of the form $p_0(p_1(...p_1(p_1(p_2, p_3), p_3)..., p_3))$ in which, for some $n > 1$, n p_1 labeled nodes and n p_3 labeled nodes occur. Figure 4 shows a comparison of the lengths of call-by-value computations of M_1 and $rec\text{-}it(M_1)$. In general, the lengths of computations of M_1 on input trees of the form t_n can be described by the function a which is defined recursively as follows:

$$a(1) = 5$$

$$a(n+1) = 1 + 2 * a(n-1)$$

which is clearly an exponential function. In contrast, the lengths of computations of $rec\text{-}it(M_1)$ on input trees of the form t_n can be described by the function b which is defined linearly as follows:

$$b(n) = 2 + 3 * n$$

n	M_1	$rec\text{-}it(M_1)$
1	5	5
2	11	8
3	23	11
⋮	⋮	⋮

Figure 4: Comparison of efficiency of M_1 and $rec\text{-}it(M_1)$.

Now we will describe the origins of our approach. The idea of the underlying transformation strategy goes back to the fact that macro tree transducers are closely related to attribute grammars [Eng80, CF82] and to a technique [DPSS77] of computing output objects of an attribute grammar by means of IO macro grammars. Let us explain this chain of connections by applying the two involved transformations to M_1, and then compare the resulting IO macro grammar with $rec\text{-}it(M_1)$.

In the first step we apply the construction of [CF82] which takes a well-presented macro tree transducer and transforms it into an equivalent attribute grammar. In fact, M_1 is well-presented. The input symbols p_i of the macro tree transducer M_1 have to be considered as productions of some context-free grammar. Here we choose the interpretation $p_0 : S \to A$, $p_1 : A \to aACa$, $p_2 : A \to b$, and $p_3 : C \to c$. (Note that there is no reason why we have chosen this particular interpretation.) Figure 5 shows the resulting attribute grammar G_1 (with slight variations due to a better fitting in our context). For an easier understanding, the involved attribute dependencies are shown in Figures 6 and 7; there we have omitted the superscripts of the attributes, because the correspondence between attributes and nonterminals is clear.

Actually, the resulting attribute grammar can be considered as an instance of a particular type of attribute grammar: it is a *simple-L-attributed grammar*. A simple-L-attributed grammar is an attribute grammar [Knu68] for which the following additional conditions hold.

$$
\begin{array}{llll|lll}
p_0: & S & \rightarrow & A & s^0(S) & = & s^1(A) \\
 & & & & i_1^1(A) & = & \varepsilon \\
 & & & & i_2^1(A) & = & \varepsilon \\
p_1: & A & \rightarrow & aACa & s^0(A) & = & s^1(A)\#s^2(C) \\
 & & & & i_1^1(A) & = & 1i_1^0(A) \\
 & & & & i_2^1(A) & = & bi_2^0(A) \\
 & & & & i^1(C) & = & cs^1(A) \\
p_2: & A & \rightarrow & b & s^0(A) & = & i_1^0(A)\#i_2^0(A) \\
p_3: & C & \rightarrow & c & s^0(C) & = & ci^0(C)
\end{array}
$$

Figure 5: Productions and semantic equations of G_1.

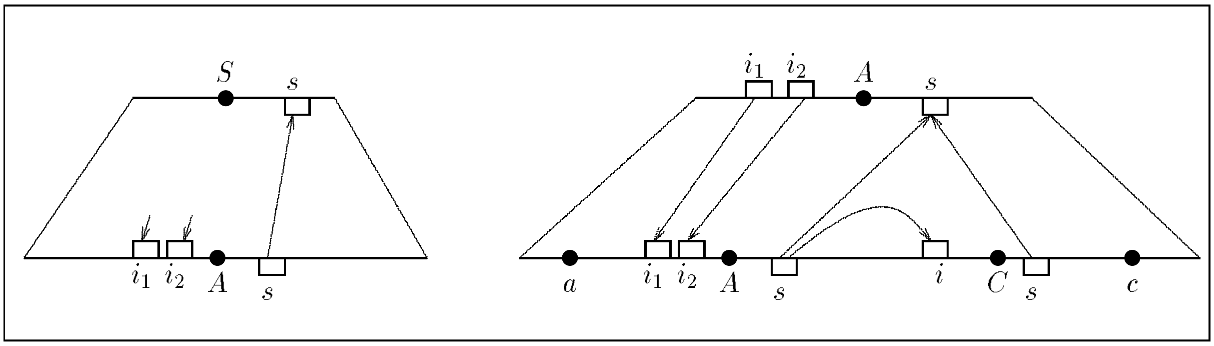


Figure 6: Dependency graphs of the productions $S \rightarrow A$ and $A \rightarrow aACa$, respectively.

- Every nonterminal has exactly one synthesized attribute (and an arbitrary number of inherited attributes; the startsymbol S has zero inherited attributes).
- It is an L-attributed grammar [Boc76], i.e., the attribute evaluation can be performed in a depth-first left-to-right tree traversal.
- The semantic domain is the set of words over some output terminal alphabet on which word functions operate; a word function is a function of the form

  $$\lambda u_1.\lambda u_2.\ldots\lambda u_r.w$$

  such that w is a word over the output terminal symbols and the variables $u_1, u_2, \ldots, u_r$; the variables range over output terminal words.

As usual, the semantics of an input word w in the language of the underlying context-free grammar is described as the value of the designated synthesized attribute at the root of the derivation tree of w. For instance, $abca$ is a word in the language of the context free grammar underlying G_1. The derivation tree of $abca$ is the tree $p_0(p_1(p_2, p_3))$ and the corresponding

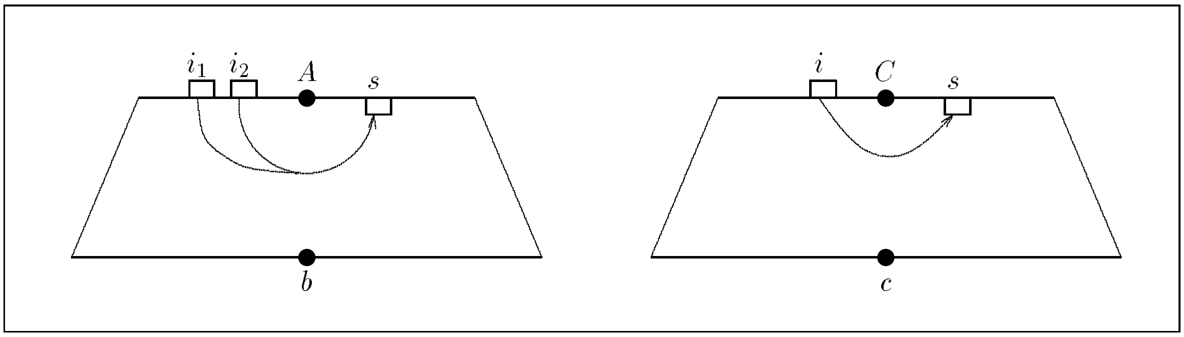


Figure 7: Dependency graphs of the productions $A \to b$ and $C \to c$, respectively.

dependency graph is shown in Figure 8. The semantics of $abca$ is the output terminal word $1\#b\#cc1\#b$ (recall that G_1 interprets its function symbols in semantic rules as word functions).

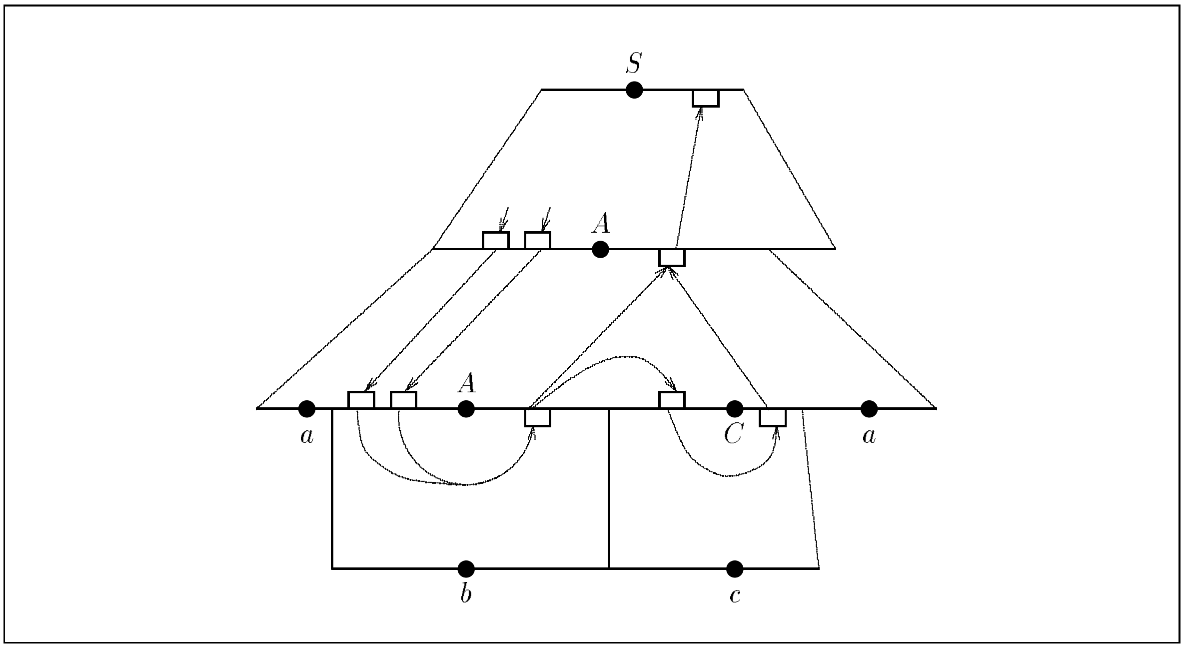


Figure 8: Dependency graph of the derivation tree.

Now we can apply the second step which is due to [DPSS77]. There, a method is described to associate with a simple-L-attributed grammar G an equivalent IO-macro grammar $io(G)$ [Fis68, ES78]. The nonterminals of G are taken over as nonterminals of $io(G)$; the output terminal symbols of G become the terminal symbols of $io(G)$. The rules of $io(G)$ are computed from the semantic equations of the simple-L-attributed grammar by imitating the depth-first left-to-right tree traversal locally to the current production. In fact, during this traversal attribute values are computed iteratively in a value driven way. The rules of the IO-macro grammar

$$
\begin{array}{lrcl}
\pi_0^1: & S & \to & H_0^1(A(\varepsilon,\varepsilon)) \\
\pi_0^2: & H_0^1(z_1) & \to & z_1 \\
\pi_1^1: & A(y_1,y_2) & \to & H_1^1(y_1,y_2,A(1y_1,by_2)) \\
\pi_1^2: & H_1^1(y_1,y_2,z_1) & \to & H_1^2(y_1,y_2,z_1,C(cz_1)) \\
\pi_1^3: & H_1^2(y_1,y_2,z_1,z_2) & \to & z_1\#z_2 \\
\pi_2^1: & A(y_1,y_2) & \to & y_1\#y_2 \\
\pi_3^1: & C(y_1) & \to & cy_1
\end{array}
$$

Figure 9: Rules of the IO-macro grammar $io(G_1)$.

$io(G_1)$ associated to G_1 are shown in Figure 9. We note that new nonterminals appear (here: H_0^1, H_1^1, and H_1^2).

Now let us compute the semantics of the input word $abca$ by means of $io(G_1)$. This is done by deriving an output terminal word by leftmost-innermost derivation relation.

$$
\begin{array}{lll}
S & \Rightarrow_{\pi_0^1} & H_0(A(\varepsilon,\varepsilon)) \\
 & \Rightarrow_{\pi_1^1} & H_0(H_1^1(\varepsilon,\varepsilon,A(1,b))) \\
 & \Rightarrow_{\pi_2^1} & H_0(H_1^1(\varepsilon,\varepsilon,1\#b)) \\
 & \Rightarrow_{\pi_1^2} & H_0(H_1^2(\varepsilon,\varepsilon,1\#b,C(c1\#b))) \\
 & \Rightarrow_{\pi_3^1} & H_0(H_1^2(\varepsilon,\varepsilon,1\#b,cc1\#b)) \\
 & \Rightarrow_{\pi_1^3} & H_0(1\#b\#cc1\#b) \\
 & \Rightarrow_{\pi_0^2} & 1\#b\#cc1\#b
\end{array}
$$

We note that the value of $A(1,b)$ only has to be computed once; it is stored in an argument position of nonterminal H_1^2 for later use. Moreover, we note that the computation of the final object, i.e., the output terminal string, is computed iteratively in a value-driven way.

If we compare $rec\text{-}it(M_1)$ and $io(G_1)$, then we realize that $rec\text{-}it(M_1)$ can be considered as a mixture of the rules of the original macro tree transducer M_1 (which still compute values in a demand driven, i.e., recursive way) and the IO macro grammar $io(G_1)$ (which compute values in a value-driven, i.e., iterative way). In particular, the register function $\langle f_A, p_1, 1\rangle$ corresponds to the nonterminal H_1^1. The connection between $rec\text{-}it(M_1)$ and $io(G_1)$ becomes even more clear if we slightly optimize $io(G_1)$ without changing its semantics. The resulting IO-macro grammar $io(G_1)'$ is shown in Figure 10. In fact, $io(G_1)'$ has the same structure as $rec\text{-}it(M_1)$ and the computations of $io(G_1)'$ and $rec\text{-}it(M_1)$ on input tree $p_0(p_1(p_2,p_3))$ are closely related:

$$
\begin{array}{lrcl}
\pi_0^1: & S & \to & A(\varepsilon,\varepsilon) \\
\pi_1^1: & A(y_1,y_2) & \to & H_1^1(y_1,y_2,A(1y_1,by_2)) \\
\pi_1^2: & H_1^1(y_1,y_2,z_1) & \to & z_1\#C(cz_1) \\
\pi_2^1: & A(y_1,y_2) & \to & y_1\#y_2 \\
\pi_3^1: & C(y_1) & \to & cy_1
\end{array}
$$

Figure 10: Rules of the IO-macro grammar $io(G_1)'$.

derivation of $rec\text{-}it(M_1)$	derivation of $io(G_1)'$
$f_S(p_0(p_1(p_2,p_3)))$	S
$\Rightarrow$ $f_A(p_1(p_2,p_3),\varepsilon,\varepsilon)$	$\Rightarrow_{\pi_0^1}$ $A(\varepsilon,\varepsilon)$
$\Rightarrow$ $\langle f_A,p_1,1\rangle(p_2,p_3,\varepsilon,\varepsilon,f_A(p_2,1(\varepsilon),b(\varepsilon)))$	$\Rightarrow_{\pi_1^1}$ $H_1^1(\varepsilon,\varepsilon,A(1,b))$
$\Rightarrow$ $\langle f_A,p_1,1\rangle(p_2,p_3,\varepsilon,\varepsilon,\#(1(\varepsilon),b(\varepsilon)))$	$\Rightarrow_{\pi_2^1}$ $H_1^1(\varepsilon,\varepsilon,1\#b)$
$\Rightarrow$ $\#(\#(1(\varepsilon),b(\varepsilon)),f_C(p_3,c(\#(1(\varepsilon),b(\varepsilon)))))$	$\Rightarrow_{\pi_1^2}$ $1\#b\#C(c1\#b)$
$\Rightarrow$ $\#(\#(1(\varepsilon),b(\varepsilon)),c(c(\#(1(\varepsilon),b(\varepsilon)))))$	$\Rightarrow_{\pi_3^1}$ $1\#b\#cc1\#b$

This paper is organized in six sections. Section 2 contains general notations and Section 3 recalls the concept of macro tree transducers, introduces the concept of macro tree transducers with register functions and determines some useful properties of the reduction relation associated to macro tree transducers with register functions. Besides the term "more efficient" is exactly defined. In Section 4 the three transformation relations splitting $\vdash_{split,M}$, sharing $\vdash_{share,M}$ and tupling $\vdash_{tuple,M}$ are presented. For each transformation relation the properties semantic-preserving, confluent, and noetherian are proved. In Section 5 the transformation strategy is defined. Furthermore, a decision algorithm is given which is able to determine whether or not, $rec\text{-}it(M)$ is more efficient than M.

2 Preliminaries

We recall some general notations and fundamental ideas which will be used in the rest of the paper.

2.1 General notations

The set of nonnegative integers is denoted by $\mathbb{N}$. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. Let S be an arbitrary set. The *set of strings* over S is denoted by S^*. The *empty string* is denoted by ε.

For a finite set A, the cardinality of A is denoted by $card(A)$.

Let $\Rightarrow$ be a binary relation on an arbitrary set S. Then, for every $n \geq 0$, the relation $\Rightarrow^n$ denotes the n-fold composition of $\Rightarrow$; the relation $\Rightarrow^+$ denotes the transitive closure of $\Rightarrow$; the relation $\Rightarrow^*$ denotes the reflexive, transitive closure of $\Rightarrow$. As usual, we write $\xi_1 \Rightarrow \xi_2$ rather than $(\xi_1, \xi_2) \in \Rightarrow$.

Let $\xi \in S$. If there is no $\xi' \in S$ such that $\xi \Rightarrow \xi'$, then ξ is called *irreducible with respect to* $\Rightarrow$. If ξ is irreducible with respect to $\Rightarrow$ and $\xi_0 \Rightarrow^* \xi$ for some $\xi_0 \in S$, then ξ is an *irreducible form of* ξ_0 *with respect to* $\Rightarrow$.

A *derivation with respect to* $\Rightarrow$ is a sequence $\xi_1 \Rightarrow \xi_2 \Rightarrow \xi_3 \Rightarrow \ldots$ such that for every $i \geq 1$, the elements $\xi_i \in S$ and $\xi_i \Rightarrow \xi_{i+1}$.

We say that $\Rightarrow$ is

- *confluent* if, for every $\xi, \xi_1, \xi_2 \in S$, the following implication holds: if $\xi \Rightarrow^* \xi_1$ and $\xi \Rightarrow^* \xi_2$, then there is a $\xi' \in S$ such that $\xi_1 \Rightarrow^* \xi'$ and $\xi_2 \Rightarrow^* \xi'$,
- *locally confluent* if, for every $\xi, \xi_1, \xi_2 \in S$, the following implication holds: if $\xi \Rightarrow \xi_1$ and $\xi \Rightarrow \xi_2$, then there is a $\xi' \in S$ such that $\xi_1 \Rightarrow^* \xi'$ and $\xi_2 \Rightarrow^* \xi'$, and
- *noetherian* if there are no infinite derivations.

We mention two results concerning a binary relation $\Rightarrow$ (cf., e.g., [Hue80]). If $\Rightarrow$ is locally confluent and noetherian, then $\Rightarrow$ is confluent. If $\Rightarrow$ is confluent and noetherian, then, for every $\xi \in S$, there is a unique irreducible form of ξ with respect to $\Rightarrow$, which is called *normalform of* ξ, denoted by $nf(\Rightarrow, \xi)$.

2.2 Lists and operations on lists

Let S be an arbitrary set. A *list over* S is a tuple $(s_1, \ldots, s_m)$ for some elements $s_1, \ldots, s_m \in S$; the empty list is denoted by $()$.

The *set of lists over* S is denoted by $\mathcal{L}(S)$. For an element $s \in S$ and a list $l \in \mathcal{L}(S)$, we abbreviate the fact that s occurs at least once in l by writing s *in* l. The append-operator $++$ on lists over S is defined as follows: if $(s_1, \ldots, s_m)$ and $(t_1, \ldots, t_n)$ are two lists over S, then $(s_1, \ldots, s_m) + +(t_1, \ldots, t_n) = (s_1, \ldots, s_m, t_1, \ldots, t_n)$ is a list over S. A non-empty list $l = (s_1, \ldots, s_m)$ with $m > 0$ is also denoted by $s_1 : l_1$ where $l_1 = (s_2, \ldots, s_m)$. The mapping $double : \mathcal{L}(S) \to \mathcal{L}(\mathbb{N})$ yields a list of positions at which elements of S occur repeatedly in the argument list. It is defined as follows:

$$
\begin{array}{lcl}
double(l) & = & f(l, (), 1, ()) \text{ where} \\
f(a : l_1, l_2, i, l_3) & = & f(l_1, l_2, i+1, l_3 + +(i)), \text{ if } a \text{ } in \text{ } l_2 \\
 & & f(l_1, l_2 + +(a), i+1, l_3), \text{ otherwise} \\
f((), l_2, i, l_3) & = & l_3
\end{array}
$$

For example, if $S = \{a, b, c\}$, then $double((a, b, b, a, c, a)) = (3, 4, 6)$. The function $delpos : \mathcal{L}(S) \times \mathcal{L}(\mathbb{N}) - \to \mathcal{L}(S)$ omits from the first argument list the elements, of which the positions are given in the second argument list, if this second list is ordered by $<$, otherwise it is undefined. It is defined as follows:

$$\begin{array}{lcl} delpos(l_1, l_2) & = & g(l_1, l_2, 1) \text{ where} \\ g(a : l_1, b : l_2, i) & = & g(l_1, l_2, i+1), \text{ if } b = i \\ & & a : g(l_1, b : l_2, i+1), \text{ otherwise} \\ g(a : l_1, (), i) & = & a : l_1 \\ g((), l_2, i) & = & () \end{array}$$

2.3 Ranked alphabets and trees

A *ranked alphabet* Γ is an alphabet in which to every symbol $\gamma \in \Gamma$ a nonnegative integer is associated; this integer is called the *rank of* γ and it is denoted by $rank_\Gamma(\gamma)$. For every $n \geq 0$, we denote the set of symbols of Γ with rank n by $\Gamma^{(n)}$. If γ has rank n, then we write also $\gamma^{(n)}$.

Let Γ be a ranked alphabet and let S be an arbitrary set. The *set of trees over* Γ *indexed by* S, denoted by $T\langle\Gamma\rangle(S)$, is the smallest set T such that the following two conditions hold.

(i) For every $s \in S$, the element $s \in T$.

(ii) For every $\gamma \in \Gamma^{(k)}$ with $k \geq 0$ and $t_1, \ldots, t_k \in T$, the tree $\gamma(t_1, \ldots, t_k) \in T$.

In the context of trees we prefer to write γ instead of $\gamma()$, if $\gamma \in \Gamma^{(0)}$. The set $T\langle\Gamma\rangle(\emptyset)$ is abbreviated by $T\langle\Gamma\rangle$. Let $t \in T\langle\Gamma\rangle(S)$. The set of *paths* of t, denoted by $path(t)$, and the *height of* t, denoted by $height(t)$, are defined inductively on the structure of t.

(i) If $t \in \Gamma^{(0)} \cup S$, then $path(t) = \{\varepsilon\}$ and $height(t) = 0$.

(ii) If $t = \gamma(t_1, \ldots, t_k)$ with $k > 0$, then $path(t) = \{\varepsilon\} \cup \{iw \mid i \in [k], w \in path(t_i)\}$ and $height(t) = 1 + max(\{height(t_i) \mid i \in [k]\})$.

The prefix ordering on $path(t)$ is denoted by $\leq$ and the lexicographical ordering on $path(t)$ is denoted by $\leq_{lex}$.

Let $w_1, w_2 \in path(t)$ be two paths of t. Then w_1 and w_2 are *incomparable*, iff neither $w_1 \leq w_2$ nor $w_2 \leq w_1$.

Let $t = \gamma(t_1, \ldots, t_k)$ with $k \geq 0$ be a tree and let $w \in path(t)$. We define the *label of* t *at* w, denoted by $label(t, w)$, and the *subtree of* t *at* w, denoted by $sub(t, w)$, inductively as follows:

(i) If $w = \varepsilon$, then $label(t, w) = \gamma$ and $sub(t, w) = t$.

(ii) If $w = iv$ for some $i \in [k]$, $v \in path(t_i)$, then $label(t, w) = label(t_i, v)$ and $sub(t, w) = sub(t_i, v)$.

The label of t at ε is also denoted by $root(t)$. The *set of subtrees of* t is denoted by $SUB(t)$.

Let $w_1, \ldots, w_n$ with $n \geq 0$ be paths of a tree t such that, for every $i, j \in [n]$ with $i \neq j$, w_i and w_j are incomparable. Then, for trees $s_1, \ldots, s_n$, we abbreviate by $t[w_1 \leftarrow s_1, \ldots, w_n \leftarrow s_n]$ the tree t in which we have replaced each subtree at w_i by the tree s_i. Let t be a tree and let $t_1, \ldots, t_n$ be subtrees of t such that, for every $i, j \in [n]$ with $i \neq j$, $t_i \notin SUB(t_j)$ and $t_j \notin SUB(t_i)$. The tree t in which we have replaced each occurrence of a subtree t_i by the tree s_i is abbreviated by $t[t_1/s_1, \ldots, t_n/s_n]$.

2.4 Variables and Substitutions

For the rest of the paper, we fix three sets of *variables*, viz., the set $X = \{x_1, x_2, x_3, \ldots\}$ of *recursion variables*, the set $Y = \{y_1, y_2, y_3, \ldots\}$ of *context parameters*, and the set $Z = \{z_1, z_2, z_3, \ldots\}$ of *result variables*. For every $k \geq 0$, we define $X_k = \{x_1, \ldots, x_k\}$ and similarly for Y_k and Z_k. The union $X \cup Y \cup Z$ is denoted by V. For a tree $t \in T\langle\Gamma\rangle(V)$, the set of variables which occur in t, is denoted by $V(t)$.

Let Γ be a ranked alphabet. A mapping $\varphi : V \to T\langle\Gamma\rangle(V)$, where the set $\{x \mid \varphi(x) \neq x, x \in V\}$ is finite, is called Γ-*substitution*. The set $\{x \mid \varphi(x) \neq x\}$ is denoted by $\mathcal{D}(\varphi)$ and is called the *domain of* φ. The *extension of* φ is the mapping $\tilde{\varphi} : T\langle\Gamma\rangle(V) \to T\langle\Gamma\rangle(V)$ defined inductively as follows.

(i) If $t \in V$, then $\tilde{\varphi}(t) = \varphi(t)$.

(ii) If $t = \gamma(t_1, \ldots, t_n)$, $\gamma \in \Gamma^{(n)}$, then $\tilde{\varphi}(t) = \gamma(\tilde{\varphi}(t_1), \ldots, \tilde{\varphi}(t_n))$.

In the following the extension of a substitution φ is also denoted by φ.
Let $t \in T\langle\Gamma\rangle(V)$. If $\mathcal{D}(\varphi) = \{x_1, \ldots, x_n\}$, then $\varphi(t)$ is represented as $t[x_1/\varphi(x_1), \ldots, x_n/\varphi(x_n)]$ or $t[x_i/\varphi(x_i); i \in [n]]$.

2.5 Principle of simultaneous induction

Here we use the principle of simultaneous induction which is a kind of double induction on trees [EV85]. Let Γ be a ranked alphabet, let A be an arbitrary set, and for every $k \geq 0$, let B_k be a set. The mapping

$$f : T\langle\Gamma\rangle \to A$$

and for every $k \geq 0$, the mapping

$$g_k : (T\langle\Gamma\rangle)^k \to B_k$$

is *defined by simultaneous induction* if the following holds.

(a) For every $\sigma \in \Gamma^{(k)}$ with $k \geq 0$ and $s_1, \ldots, s_k \in T\langle\Gamma\rangle$, $g_k((s_1, \ldots, s_k))$ is used to define $f(\sigma(s_1, \ldots, s_k))$.

(b) For every $s_1, \ldots, s_k \in T\langle\Gamma\rangle$ with $k \geq 0$, $f(s_1), \ldots f(s_k)$ are used to define $g_k((s_1, \ldots, s_k))$.

Similarly we can use the principle of simultaneous induction to prove properties. For every $k \geq 0$, let Q_k and P be two predicates which range over $(T\langle\Gamma\rangle)^k$ and $T\langle\Gamma\rangle$, respectively. Q_k and P are said to be *proved by simultaneous induction* if (a) and (b) are proved.

(a) For every $\sigma \in \Gamma^{(k)}$ with $k \geq 0$ and $s_1, \ldots, s_k \in T\langle\Gamma\rangle$, if $Q_k((s_1, \ldots, s_k))$ holds, then $P(\sigma(s_1, \ldots, s_k))$ holds.

(b) For every $s_1, \ldots, s_k \in T\langle\Gamma\rangle$ with $k \geq 0$, if $P(s_1)$ and $P(s_2)$ and $\ldots$ and $P(s_k)$ hold, then $Q_k((s_1, \ldots, s_k))$ holds.

3 Macro tree transducer and macro tree transducer with register functions

Before starting with the main topic of this paper – the transformation relations and the transformation strategy – we define the class of term rewriting systems which we want to transform, namely the class of macro tree transducers, and the class of term rewriting systems on which these transformations are based. This class is called the class of macro tree transducers with register functions.

Macro tree transducers are well-known from the literature and were introduced in [CF82, Eng80] (see also [EV85]). They are formal calculi to describe the computation of primitive recursive program schemes with parameters in which simultaneous recursion and nesting of function calls in parameters are possible.

For technical reasons we define the set of ground right-hand sides of a macro tree transducer separately.

Definition 3.1 Let F and Δ be two ranked alphabets. Let $k, n \geq 0$. The set of *ground right-hand sides over F, Δ, k and n*, denoted by $gr\text{-}RHS(F, \Delta, k, n)$, is the smallest set RHS such that the following properties hold:

(i) $Y_n \subseteq RHS$.

(ii) For every $\delta \in \Delta^{(m)}$ with $m \geq 0$ and $\zeta_1, \ldots, \zeta_m \in RHS$, the term $\delta(\zeta_1, \ldots, \zeta_m) \in RHS$.

(iii) For every $f \in F^{(m+1)}$ with $m \geq 0$, i with $1 \leq i \leq k$, and $\zeta_1, \ldots, \zeta_m \in RHS$, the term $f(x_i, \zeta_1, \ldots, \zeta_m) \in RHS$. □

Definition 3.2 A *macro tree transducer* is a tuple $M = (F, \Sigma, \Delta, R)$ such that

- F is the ranked alphabet of *functions*,
- Σ is the ranked alphabet of *input symbols*,
- Δ is the ranked alphabet of *output symbols* with $\Sigma \subseteq \Delta$ and $\Delta \cap F = \emptyset$, and
- R is a finite set of rules of the form $l \to \zeta$. For every $f \in F^{(n+1)}$ with $n \geq 0$ and $\sigma \in \Sigma^{(k)}$ with $k \geq 0$, there is exactly one rule in R of the form

$$f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n) \to \zeta$$

and ζ is in $gr\text{-}RHS(F, \Delta, k, n)$. No other rules are in R.

□

The class of macro tree transducers is denoted by $\mathcal{M}$. Let us give an example of a macro tree transducer which will also serve as running example in the next sections.

Example 3.3 The tuple $M_1 = (F_1, \Sigma_1, \Delta_1, R_1)$ with the components

- $F_1 = \{lift^{(2)}, extr^{(2)}\}$,
- $\Sigma_1 = \{\sigma^{(2)}, \alpha^{(0)}\}$,

- $\Delta_1 = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$, and
- R_1 is the set of the rules which are shown in Figure 11,

is a macro tree transducer. □

$$
\begin{array}{rcl}
lift(\alpha, y_1) & \rightarrow & y_1 \\
lift(\sigma(x_1, x_2), y_1) & \rightarrow & \sigma(lift(x_1, y_1), \sigma(extr(x_2, \gamma(y_1)), extr(x_1, y_1))) \\
extr(\alpha, y_1) & \rightarrow & \alpha \\
extr(\sigma(x_1, x_2), y_1) & \rightarrow & \sigma(lift(x_2, lift(x_1, y_1)), lift(x_1, y_1))
\end{array}
$$

Figure 11: Rules of the macro tree transducer M_1.

Since we will have to deal with some additional functions in the sequel, let us call the original functions of a macro tree transducer *simple functions*.

As mentioned before, the transformation strategy which we will define, transforms a macro tree transducer into a more general term rewriting system, called macro tree transducer with register functions. Macro tree transducer with register functions are a generalization of macro tree transducers in the sense that, to every pair (f, σ) consisting of a simple function f and an input symbol σ, a finite sequence of *register functions* is associated. Intuitively, the arguments of such register functions serve as registers in which trees over Δ can be built up and accumulated. This accumulation is done as follows: the simple function f calls the first register function in the corresponding sequence, and the i-th register function calls the $(i+1)$-st register function. Thus register functions are non recursive.

During the transformation process we will tuple simple functions into one function; such functions will be called *tuple functions*. Note that also to every pair (g, σ) consisting of a tuple function g and an input symbol σ, a finite sequence of register functions is associated. The next definition describes the functions involved in a macro tree transducer with register functions formally.

Definition 3.4 Let Σ be a ranked alphabet. A *system FS of functions over Σ* is a tuple $(sim(FS), tup(FS), reg(FS))$ of disjoint ranked alphabets $sim(FS)$, $tup(FS)$, and $reg(FS)$ of *simple functions*, *tuple functions*, and *register functions*, respectively, such that the following properties hold:

- $sim(FS)^{(0)} \cup tup(FS)^{(0)} \cup reg(FS)^{(0)} = \emptyset$, i.e., every simple function, tuple function, and register function has at least rank 1.
- $tup(FS) = \{(f_1, \ldots, f_m)^{(n)} \mid m > 1, n > 0, f_1, \ldots, f_m \in sim(FS)^{(n)}\}$
- $reg(FS) = \bigcup_{f \in sim(FS) \cup tup(FS), \sigma \in \Sigma} \ (f, \sigma)\text{-}reg(FS)$.
- For every $f \in sim(FS) \cup tup(FS)$ of rank $n+1$ and $\sigma \in \Sigma^{(k)}$ with $n, k \geq 0$, there is an $n_{f,\sigma} \geq 0$ such that the set $(f, \sigma)\text{-}reg(FS) = \{\langle f, \sigma, i\rangle | 1 \leq i \leq n_{f,\sigma}\}$ and $rank_{reg(FS)}(\langle f, \sigma, i\rangle) = n + k + r_{\langle f,\sigma,i\rangle}$ for some $r_{\langle f,\sigma,i\rangle} > 0$.

The set $sim(FS) \cup tup(FS)$ is called the set of *ground functions*, denoted by $gr(FS)$. □

In the following, given a system FS of functions over Σ, we also use FS as abbreviation for $sim(FS) \cup tup(FS) \cup reg(FS)$.

Definition 3.5 A *macro tree transducer with register functions* is a tuple $N = (FS, \Sigma, \Delta, R)$ such that

- $FS = (sim(FS), tup(FS), reg(FS))$ is a system of functions over Σ.
- Σ and Δ are ranked alphabets of *input symbols* and *output symbols*, respectively. For every tuple function $(f_1, \ldots, f_m)$ with $m > 1$ it holds that $comb_m \in \Delta^{(m)}$.
- R is a finite set of *rules* which is partitioned into the sets $gr(R)$ and $reg(R)$ of *ground rules* and *register rules*, respectively. Every rule in R is left linear, i.e., in the left-hand side of every rule no variable may occur more than once. The two sets have the following properties.
  - $gr(R)$: For every $f \in gr(FS)^{(n+1)}$ with $n \geq 0$ and $\sigma \in \Sigma^{(k)}$ with $k \geq 0$, there is exactly one rule of the form

    $$f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n) \to \zeta$$

    such that either
    - $\zeta \in gr\text{-}RHS(gr(FS), \Delta, k, n)$ or
    - ζ has the form $\langle f, \sigma, 1\rangle(x_1, \ldots, x_k, y_1, \ldots, y_n, \zeta_1, \ldots, \zeta_r)$ where r is determined by the equality $rank_{reg(FS)}(\langle f, \sigma, 1\rangle) = k + n + r$ and, for every j with $1 \leq j \leq r$, the term $\zeta_j \in gr\text{-}RHS(gr(FS), \Delta, k, n)$.
  - $reg(R)$:
    - For every $\langle f, \sigma, i\rangle \in reg(FS)^{(n+k+r)}$ with $f \in gr(FS)^{(n+1)}$ and $\sigma \in \Sigma^{(k)}$, and $1 \leq i \leq n_{f,\sigma} - 1$, there is exactly one rule in $reg(FS)$ of the form

      $$\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, u_1, \ldots, u_r) \to \langle f, \sigma, i+1\rangle(\tilde{x}, \tilde{y}, \zeta_1, \ldots, \zeta_p)$$

      where $\tilde{x}$ and $\tilde{y}$ abbreviate the sequences $x_1, \ldots, x_k$ and $y_1, \ldots, y_n$, respectively, and
      - for every j with $1 \leq j \leq r$, the term $u_j \in Z$ or there are $r(j) \geq 0$, $\delta \in \Delta^{(r(j))}$, and $z_{j,1}, \ldots, z_{j,r(j)} \in Z$ such that $u_j = \delta(z_{j,1}, \ldots, z_{j,r(j)})$ and
      - $p \geq 0$ and for every j with $1 \leq j \leq p$, the term $\zeta_j \in gr\text{-}RHS(gr(FS), \Delta \cup \bigcup_{1 \leq i \leq r} V(u_i), k, n)$.
    - For every $\langle f, \sigma, n_{f,\sigma}\rangle \in reg(FS)^{(n+k+r)}$ with $f \in gr(FS)^{(n+1)}$ and $\sigma \in \Sigma^{(k)}$, there is exactly one rule in $reg(FS)$ of the form

      $$\langle f, \sigma, n_{f,\sigma}\rangle(\tilde{x}, \tilde{y}, u_1, \ldots, u_r) \to \zeta$$

      where $\tilde{x}$ and $\tilde{y}$ abbreviate the sequences $x_1, \ldots, x_k$ and $y_1, \ldots, y_n$, respectively, and
      - for every j with $1 \leq j \leq r$, the term $u_j \in Z$ or there are $r(j) \geq 0$, $\delta \in \Delta^{(r(j))}$, and $z_{j,1}, \ldots, z_{j,r(j)} \in Z$ such that $u_j = \delta(z_{j,1}, \ldots, z_{j,r(j)})$ and

· $\zeta \in gr\text{-}RHS(gr(FS), \Delta \cup \bigcup_{1 \le i \le r} V(u_i), k, n)$. $\square$

A rule of which the left-hand side has the form $f(\sigma(\ldots),\ldots)$ or $\langle f, \sigma, i\rangle(\ldots)$ is called an *(f, σ)-rule*.

We denote the class of macro tree transducers with register functions by $\mathcal{N}$. Note that, in a rule of a macro tree transducer with register functions, the argument list of a register function $\langle f, \sigma, i\rangle$ with $rank(f) = n + 1$ and $rank(\sigma) = k$ always starts with the variables $x_1, \ldots, x_k, y_1, \ldots, y_n$. In the following we often abbreviate these sequences by $\tilde{x}$ and $\tilde{y}$.

Let us illustrate this definition by an example:

Example 3.6 Consider the macro tree transducer N_1 with register functions which is the tuple $(FS_1, \Sigma_1, \Delta_1, R_1)$ with the following components:

- $sim(FS_1) = \{fib^{(1)}, h^{(1)}\}$, $tup(FS_1) = \{(fib, h)^{(1)}\}$, and $reg(FS_1) = \{\langle fib, \sigma, 1\rangle^{(2)}, \langle (fib, h), \sigma, 1\rangle^{(2)}\}$,
- $\Sigma_1 = \{\sigma^{(1)}, \alpha^{(0)}\}$,
- $\Delta_1 = \{comb_2^{(2)}, +^{(2)}, \sigma^{(1)}, \alpha^{(0)}\}$, and
- the rules of R_1 are shown in Figure 12. Thus, $gr(R_1)$ contains the rules (1), (2), (4), (5), (6), and (7) and $reg(R_1)$ contains the rules (3) and (8). $\square$

$$
\begin{array}{rcll}
fib(\alpha) & \to & \sigma(\alpha) & (1)\\
fib(\sigma(x_1)) & \to & \langle fib, \sigma, 1\rangle(x_1, (fib, h)(x_1)) & (2)\\
\langle fib, \sigma, 1\rangle(x_1, comb_2(z_1, z_2)) & \to & +(z_1, z_2) & (3)\\[4pt]
h(\alpha) & \to & \alpha & (4)\\
h(\sigma(x_1)) & \to & fib(x_1) & (5)\\[4pt]
(fib, h)(\alpha) & \to & comb_2(\sigma(\alpha), \alpha) & (6)\\
(fib, h)(\sigma(x_1)) & \to & \langle (fib, h), \sigma, 1\rangle(x_1, (fib, h)(x_1)) & (7)\\
\langle (fib, h), \sigma, 1\rangle(x_1, comb_2(z_1, z_2)) & \to & comb_2(+(z_1, z_2), z_1) & (8)
\end{array}
$$

Figure 12: Rules of the macro tree transducer N_1 with register functions.

In the rest of this section, let $N = (FS, \Sigma, \Delta, R)$ be an arbitrary, but fixed macro tree transducer with register functions. Let us introduce some useful notations which we will often use in further sections.

Definition 3.7 1. The set of *right-hand sides of* N, denoted by $RHS(N)$, is the set $\{\zeta \mid l \to \zeta \in R\}$.

2. Let $t \in T\langle FS \cup \Delta\rangle(V)$. A subtree $t' = f(t_1, \ldots, t_n)$ of t with $f \in FS^{(n)}$ for some $n > 0$ is called *function call*. If $f \in gr(FS)^{(n)}$, then t' is called *ground function call*. The list of subtrees $(t_1, \ldots, t_n)$ of t' is called *list of arguments* of t', denoted by $arglist(t')$.

3. The *set of function calls* in t is denoted by $F_{call}(t)$; the *set of ground function calls* in t is denoted by $F_{call}^{gr}(t)$.

4. For every rule $l \to \zeta$ in $reg(R)$, the left-hand side l is called *flat*, if $arglist(l)$ *in* $\mathcal{L}(V)$. $\square$

Clearly, every macro tree transducer can be considered as a macro tree transducer with register functions in which the system of functions FS has the form $(F, \emptyset, \emptyset)$, i.e., $sim(FS) = F$, $tup(FS) = \emptyset$, and $reg(FS) = \emptyset$.

To assign a reduction relation to macro tree transducers with register functions we first define the syntactical structure of the intermediate results of the reduction relation.

Definition 3.8 The set of *syntax-directed expressions* over FS, Σ, and Δ, which is denoted by $sdExp(FS, \Sigma, \Delta)$, is the smallest set $sdExp$ such that the following conditions hold:

(i) For every $\delta \in \Delta^{(m)}$ with $m \geq 0$ and $\psi_1, \ldots, \psi_m \in sdExp$, the tree $\delta(\psi_1, \ldots, \psi_m) \in sdExp$.

(ii) For every $f \in gr(FS)^{(n+1)}$ with $n \geq 0$, $s \in T\langle\Sigma\rangle$, and $\psi_1, \ldots, \psi_n \in sdExp$, the function call $f(s, \psi_1, \ldots, \psi_n) \in sdExp$.

(iii) For every $g \in reg(FS)^{(k+n+r)}$ for some $k, n \geq 0$, and $r > 0$, $s_1, \ldots, s_k \in T\langle\Sigma\rangle$, $t_1, \ldots, t_n \in T\langle\Delta\rangle$, and $\psi_1, \ldots, \psi_r \in sdExp$, the function call $g(s_1, \ldots, s_k, t_1, \ldots, t_n, \psi_1, \ldots, \psi_r) \in sdExp$. $\square$

Until know we have only defined syntactical objects of macro tree transducers with register functions. Now we define a reduction relation for macro tree transducers with register functions.

Definition 3.9 The *call-by-value derivation relation induced by* N, denoted by $\overset{cbv}{\Rightarrow}_N$, is the binary relation on $sdExp(FS, \Sigma, \Delta)$ such that, for every $\psi_1, \psi_2 \in sdExp(FS, \Sigma, \Delta)$, $\psi_1 \overset{cbv}{\Rightarrow}_N \psi_2$ iff

- there is a path w in $path(\psi_1)$,
- there is a rule $l \to \zeta$ in R, and
- there is a Δ-substitution φ with $\mathcal{D}(\varphi) = V(l)$

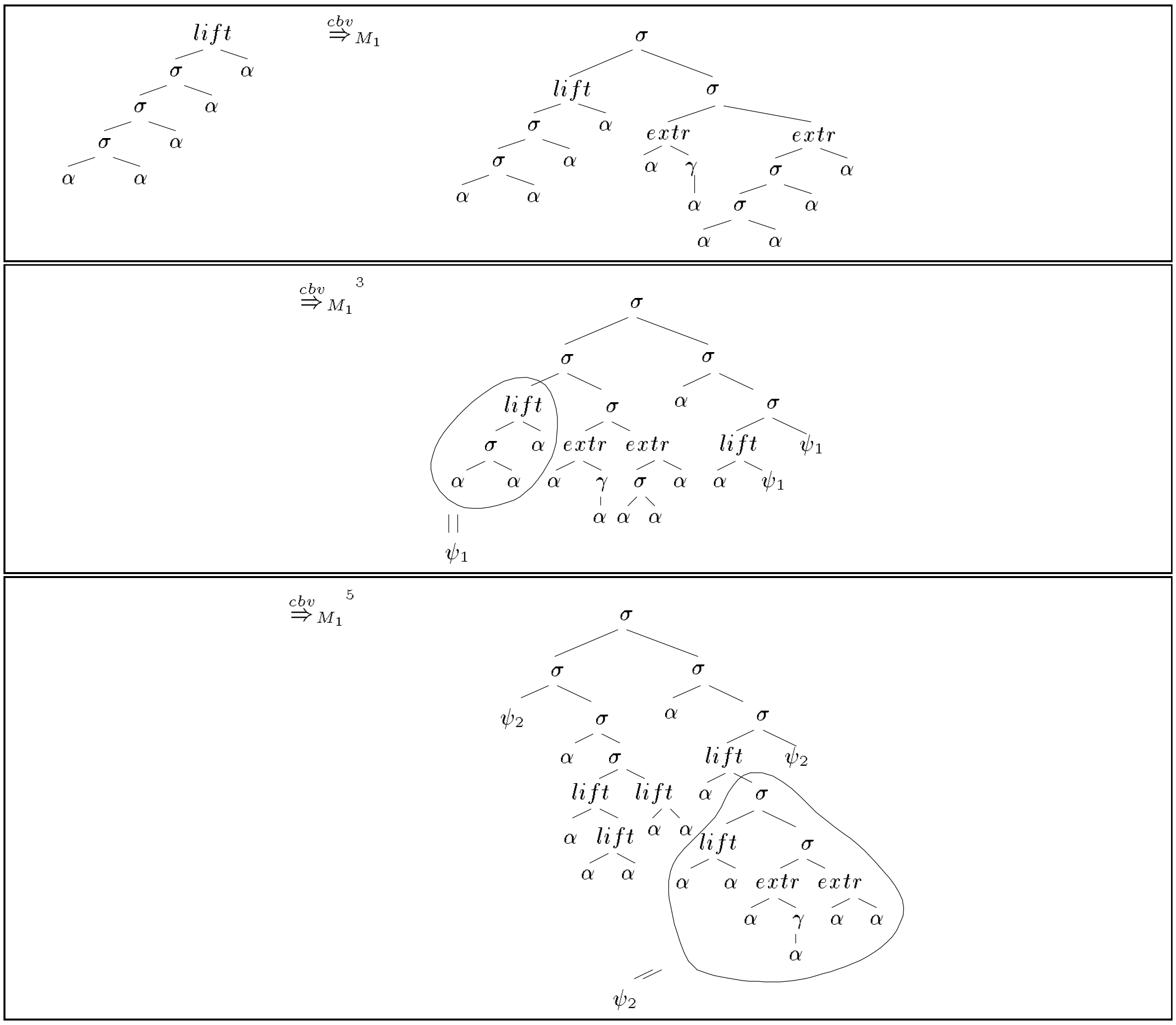
such that $\varphi(l) = sub(\psi_1, w)$ and $\psi_2 = \psi_1[w \leftarrow \varphi(\zeta)]$. The substitution φ is called *matching substitution* of l and $sub(\psi_1, w)$. $\square$

Example 3.10 First, we consider the macro tree transducer N_1 with register functions of Example 3.6 and show some steps of $\overset{cbv}{\Rightarrow}_N$ applied to the syntax-directed expression $fib(\sigma(\sigma(\sigma(\alpha))))$.

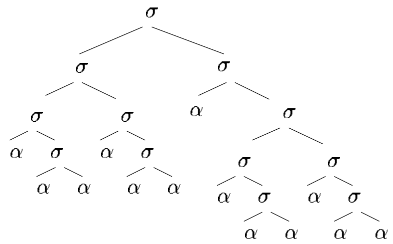
$$
\begin{array}{lll}
fib(\sigma(\sigma(\sigma(\alpha)))) & \overset{cbv}{\Rightarrow}_N & \langle fib, \sigma, 1\rangle(\sigma(\sigma(\alpha)), (fib, h)(\sigma(\sigma(\alpha)))) \\
 & & \text{with rule (2), path } w = \varepsilon\text{, and matching substitution } \varphi = [x_1/\sigma(\sigma(\alpha))] \\
 & \overset{cbv}{\Rightarrow}_N & \langle fib, \sigma, 1\rangle(\sigma(\sigma(\alpha)), \langle (fib, h), \sigma, 1\rangle(\sigma(\alpha), (fib, h)(\sigma(\alpha)))) \\
 & \overset{cbv}{\Rightarrow}_N & \langle fib, \sigma, 1\rangle(\sigma(\sigma(\alpha)), \langle (fib, h), \sigma, 1\rangle(\sigma(\alpha), \langle (fib, h), \sigma, 1\rangle(\alpha, (fib, h)(\alpha)))) \\
 & \overset{cbv}{\Rightarrow}_N & \langle fib, \sigma, 1\rangle(\sigma(\sigma(\alpha)), \langle (fib, h), \sigma, 1\rangle(\sigma(\alpha), \langle (fib, h), \sigma, 1\rangle(\alpha, comb_2(\sigma(\alpha), \alpha)))) \\
 & \overset{cbv}{\Rightarrow}_N & \langle fib, \sigma, 1\rangle(\sigma(\sigma(\alpha)), \langle (fib, h), \sigma, 1\rangle(\sigma(\alpha), comb_2(+(\sigma(\alpha), \alpha), \sigma(\alpha)))) \\
 & \overset{cbv}{\Rightarrow}_N & \langle fib, \sigma, 1\rangle(\sigma(\sigma(\alpha)), comb_2(+(+(\sigma(\alpha), \alpha), \sigma(\alpha)), +(\sigma(\alpha), \alpha))) \\
 & \overset{cbv}{\Rightarrow}_N & +(+(+(\sigma(\alpha), \alpha), \sigma(\alpha)), +(\sigma(\alpha), \alpha))
\end{array}
$$

It can be proved that the function fib computes the Fibonacci numbers assuming that α is interpreted by 0, σ is interpreted by the successor function $+1$, and $+$ is the addition of natural numbers.

Let M_1 be the macro tree transducer of Example 3.3. As second example we show a derivation of the function call $t = lift(\sigma(\sigma(\sigma(\alpha,\alpha),\alpha),\alpha),\alpha)$ by $\stackrel{cbv}{\Rightarrow}_{M_1}$ in the following boxes. For the sake of clearity, this time we use the graphical representation of the syntax-directed expressions. Note that the result is computed in 22 steps. Also note that during the derivation function calls arise multiple time as, e.g., the function call $\psi_1 = f(\sigma(\alpha,\alpha),\alpha)$. In further sections we will often fall back upon the given function call t and its derivation. $\square$



$\stackrel{cbv}{\Rightarrow}_{M_1}^{13}$



Let us list some properties of $\stackrel{cbv}{\Rightarrow}_N$.

Lemma 3.11 *The relation* $\stackrel{cbv}{\Rightarrow}_N$ *is locally confluent.*

Proof. By Lemma 3.1 of [Hue80] $\stackrel{cbv}{\Rightarrow}_N$ is locally confluent, because by definition of $\stackrel{cbv}{\Rightarrow}_N$ there does not exist any critical pair in R. In other words, the function calls which can be reduced in a step are always non-overlapping. $\square$

Lemma 3.12 *For every syntax-directed expression* ψ *there is a bound on the length of derivations starting from* ψ*.*

Proof. For the sake of brevity we coin the following notion. Let $\psi \in sdExp(FS, \Sigma, \Delta)$ and $a \geq 0$. Then we write $length\_der(\psi) \leq a$ if every derivation starting from ψ is not longer than a.

Now we prove the following statement by induction on the structure of ψ.

> For every $\psi \in sdExp(FS, \Sigma, \Delta)$ of the form $p(\psi_1, \ldots, \psi_m)$ for some $m \geq 0$, if for every $i \in [m]$, there is a number a_{ψ_i} such that $length\_der(\psi_i) \leq a_{\psi_i}$, then there is a number a_ψ such that $length\_der(\psi) \leq a_\psi$.

Consider an arbitrary call-by-value derivation starting from $p(\psi_1, \ldots, \psi_m)$. According to the inductive definition of syntax-directed expressions, we have to consider three cases: $p \in \Delta^{(m)}$, $p \in gr(FS)^{(m)}$, or $p \in reg(FS)^{(m)}$.

1. $p \in \Delta^{(m)}$: Then the derivation has the form

$$p(\psi_1, \ldots, \psi_m) \stackrel{cbv}{\Rightarrow}_N^{c} p(\xi_1, \ldots, \xi_m)$$

   where $c \leq \Sigma_{i=1}^m a_{\psi_i}$, because $length\_der(\psi_i) \leq a_{\psi_i}$. Then choose $a_\psi = \Sigma_{i=1}^m a_{\psi_i}$.

2. $p \in gr(FS)^{(m)}$: We prove the following two statements by simultaneous induction.

   P: For every $f \in gr(FS)^{(n+1)}$, $n \geq 0$, $s \in T\langle\Sigma\rangle$, $\theta_1, \ldots, \theta_n \in sdExp(FS, \Sigma, \Delta)$, if for every $i \in [n]$, there is a number $a_{\theta_i} \geq 0$ such that $length\_der(\theta_i) \leq a_{\theta_i}$, then there is a number $a_{f(s,\theta_1,\ldots,\theta_n)}$ such that $length\_der(f(s, \theta_1, \ldots, \theta_n)) \leq a_{f(s,\theta_1,\ldots,\theta_n)}$.

   Q: For every $k \geq 0$, $s_1, \ldots, s_k \in T\langle\Sigma\rangle$, $n \geq 0$, $\zeta \in gr\text{-}RHS(gr(FS), \Delta \cup Z, k, n)$ such that $V(\zeta) \cap Z = \{z_1, \ldots, z_r\}$ for some $r \geq 0$, $t_1, \ldots, t_r \in T\langle\Sigma\rangle$, and $\theta_1, \ldots, \theta_n \in sdExp(FS, \Sigma, \Delta)$, if for every $j \in [n]$, there is a number $b_{\theta_j} \geq 0$ such that $length\_der(\theta_j) \leq b_{\theta_j}$, then there is a number $b_{\zeta'}$ with $\zeta' = \zeta[x_i/s_i; i \in [k]][y_\mu/\theta_\mu; \mu \in [n]][z_\nu/t_\nu; \nu \in [r]]$ such that

$$length\_der(\zeta') \leq b_{\theta_j}.$$

P $\Rightarrow$ **Q:** This implication is proved by induction on the structure of ζ. Let $t_1, \dots, t_r \in T\langle\Sigma\rangle$, $\theta_1, \dots, \theta_n \in sdExp(FS, \Sigma, \Delta)$ and b_{θ_j} such that $length\_der(\theta_j) \leq b_{\theta_j}$.

(i) $\zeta = y_j \in Y$ with $1 \leq j \leq n$: Since $\zeta' = \theta_j$, we can choose $b_{\zeta'} = b_{\theta_j}$ and the statement follows trivially.

(ii) $\zeta = \delta(\zeta_1, \dots, \zeta_m)$ with $\delta \in \Delta^{(m)}$ and assume that **Q** holds for $\zeta_1, \dots, \zeta_m$. We can choose $b_{\zeta'} = \Sigma_{j=1}^{m} b_{\zeta_j}$ and the statement holds.

(iii) $\zeta = f(x_j, \zeta_1, \dots, \zeta_m)$ with $f \in gr(FS)^{n+1}$, $1 \leq j \leq k$ and assume that **Q** holds for $\zeta_1, \dots, \zeta_n$. Consider an arbitrary call-by-value derivation starting from $\zeta' = f(x_j, \zeta_1, \dots, \zeta_n)[x_i/s_i; i \in [k]][y_\mu/\theta_\mu; \mu \in [n]][z_\nu/t_\nu; \nu \in [r]] = f(s_j, \zeta_1', \dots, \zeta_n')$ with, for every $\rho \in [n]$, $\zeta_\rho' = \zeta_\rho[x_i/s_i; i \in [k]][y_\mu/\theta_\mu; \mu \in [n]][z_\nu/t_\nu; \nu \in [r]]$:

$$\zeta' \overset{cbv}{\Rightarrow}_N^{\,c} f(s_j, \bar{\zeta_1}, \dots, \bar{\zeta_n})$$

Then $\bar{\zeta_1}, \dots, \bar{\zeta_m} \in T\langle\Delta\rangle$ and $c \leq \Sigma_{\rho=1}^{n} b'_{\zeta_\rho}$. Since $length\_der(\bar{\zeta_i}) \leq 0$, **Q** follows from **P**.

Q $\Rightarrow$ **P:** Consider $f \in gr(FS)^{(n+1)}$, $s \in T\langle\Sigma\rangle$, and $\theta_1, \dots, \theta_n \in sdExp(FS, \Sigma, \Delta)$, and assume that for every $i \in [n]$ there is an a_{θ_i} such that $length\_der(\theta_i) \leq a_{\theta_i}$. Now consider an arbitrary call-by-value derivation starting from $f(s, \theta_1, \dots, \theta_n)$:

$$f(s, \theta_1, \dots, \theta_n) \overset{cbv}{\Rightarrow}_N^{\,c} f(s, \xi_1, \dots, \xi_n)$$

Then $\xi_1, \dots, \xi_n \in T\langle\Delta\rangle$ and $c \leq \Sigma_{i=1}^{n} a_{\theta_i}$.
Let $s = \sigma(s_1, \dots, s_k)$ and let $f(\sigma(x_1, \dots, x_k), y_1, \dots, y_n) \to \zeta$ be the rule in R.

case 1 $\zeta \in gr\text{-}RHS(gr(FS), \Delta, k, n)$: Then **P** follows immediately from **Q** and the assumption on the θ_i's.

case 2 $\zeta = \langle f, \sigma, 1\rangle(x_1, \dots, x_k, y_1, \dots, y_n, \zeta_1, \dots, \zeta_{r_{\langle f,\sigma,1\rangle}})$: Then an arbitrary call-by-value derivation starting from $\zeta[x_i/s_i][y_j/\xi_j]$ is a sequence

$$\begin{array}{lllll}
\zeta[x_i/s_i][y_j/\xi_j] = \varphi_1 & \overset{cbv}{\Rightarrow}_N^{\,c_1} & \varphi_2 & \overset{cbv}{\Rightarrow}_{N,reg(R)} & \varphi_2' \\
 & \overset{cbv}{\Rightarrow}_N^{\,c_2} & \varphi_3 & \overset{cbv}{\Rightarrow}_{N,reg(R)} & \varphi_3' \\
 & & \vdots & & \\
 & \overset{cbv}{\Rightarrow}_N^{\,c_{n_{f,\sigma}-1}} & \varphi_{n_{f,\sigma}} & \overset{cbv}{\Rightarrow}_{N,reg(R)} & \varphi_{n_{f,\sigma}}'
\end{array}$$

such that $\varphi_i \overset{cbv}{\Rightarrow}_{N,reg(R)} \varphi_i'$ is induced by the register rule

$$\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, u_1, \dots, u_{\langle f,\sigma,i\rangle}) \to \langle f, \sigma, i+1\rangle(\tilde{x}, \tilde{y}, \zeta_1, \dots, \zeta_{r_{\langle f,\sigma,i+1\rangle}}).$$

From **Q** we can calculate the numbers $c_1, c_2, \dots, c_{n_{f,\sigma}-1}$. Then we can define

$$a_{f(s,\theta_1,\dots,\theta_n)} = c + \Sigma_{i=1}^{n_{f,\sigma}-1} c_i + (n_{f,\sigma} - 1).$$

3. $p \in reg(FS)^{(m)}$: Then the derivation has the form

$$\langle f, \sigma, \nu\rangle(\psi_1, \dots, \psi_m) \overset{cbv}{\Rightarrow}_N^{\,c} \langle f, \sigma, \nu\rangle(\xi_1, \dots, \xi_m)$$

where $c \leq \Sigma_{i=1}^{m} a_{\psi_i}$ and $\xi_1, \dots, \xi_m \in T\langle\Delta\rangle$. The argumentation is similar to 2., case 2.

Therewith the lemma is proven. $\square$

Lemma 3.13 *The relation* $\overset{cbv}{\Rightarrow}_N$ *is noetherian.*

<u>Proof.</u> We prove the lemma by contradiction. We assume that $\overset{cbv}{\Rightarrow}_N$ is not noetherian, i.e., there exists an infinite derivation. Let

$$\psi_1 \overset{cbv}{\Rightarrow}_N \psi_2 \overset{cbv}{\Rightarrow}_N \psi_3 \dots$$

where, for every $i \in \mathbb{N}$, $\psi_i \in sdExp(FS, \Sigma, \Delta)$ be such an infinite derivation. This is in contradiction to Lemma 3.12. $\square$

Lemma 3.14 *The relation* $\overset{cbv}{\Rightarrow}_N$ *is confluent and noetherian.*

<u>Proof.</u> This result follows directly by the fact that $\overset{cbv}{\Rightarrow}_N$ is locally confluent (cf. Lemma 3.11) and noetherian (cf. Lemma 3.13) and by Lemma 2.4 of [Hue80]. $\square$

An important consequence of the fact that $\overset{cbv}{\Rightarrow}_N$ is confluent and noetherian is the existance of a unique normalform $nf(\overset{cbv}{\Rightarrow}_N, \psi)$ of every syntax-directed expression ψ.

Definition 3.15 The *call-by-value tree function* computed by N is the total function $\tau_{cbv}(N) : \bigcup_{n \geq 0}(gr(FS)^{(n)} \times T\langle\Sigma\rangle \times (T\langle\Delta\rangle)^n) \to sdExp(reg(FS), \Sigma, \Delta)$ defined as follows: for every $f \in gr(FS)^{(n+1)}$ with $n \geq 0$, $s \in T\langle\Sigma\rangle$ and $t_1, \dots, t_n \in T\langle\Delta\rangle$,

$$\tau_{cbv}(N)(f, s, t_1, \dots, t_n) = nf(\overset{cbv}{\Rightarrow}_N, f(s, t_1, \dots, t_n)).$$

$\square$

Note that the register rules of a macro tree transducer with register functions are in general not exhaustive. Hence, the $\overset{cbv}{\Rightarrow}_N$-normalform of a syntax-directed expression is a tree over $reg(FS)$ and Δ.

Definition 3.16 Let $M = (F_M, \Sigma_M, \Delta_M, R_M)$ be a macro tree transducer and let $N = (FS, \Sigma, \Delta, R)$ be a macro tree transducer with register functions. M and N are *semantically equivalent* if the following conditions hold:

- $F_M = sim(FS)$.
- $\Sigma_M = \Sigma$.
- $\Delta_M \subseteq \Delta$.
- For every $f \in F_M^{(n+1)}$ with $n \geq 0$, $s \in T\langle\Sigma\rangle$, $t_1, \dots, t_n \in T\langle\Delta_M\rangle$, $\tau_{cbv}(M)(f, s, t_1, \dots, t_n) = \tau_{cbv}(N)(f, s, t_1, \dots, t_n)$.
- For every $(f_1, \dots, f_m) \in tup(FS)^{(n+1)}$ for some $n \geq 0$, $s \in T\langle\Sigma\rangle$, $t_1, \dots, t_n \in T\langle\Delta_M\rangle$, if, for every $i \in [m]$, $\tau_{cbv}(M)(f_i, s, t_1, \dots, t_n) = \xi_i$, then

  $$\tau_{cbv}(N)((f_1, \dots, f_m), s, t_1, \dots, t_n)) = comb_m(\xi_1, \dots, \xi_m)$$

  and $comb_m \in \Delta^{(m)}$. $\square$

Of course, in general it may be undecidable, if a macro tree transducer M and a macro tree transducer N with register functions are semantically equivalent. But later, we will use this notion in a context in which the semantical equivalence is given automatically.

Observation 3.17 Let M be a macro tree transducer and let N be a macro tree transducer with register functions such that M and N are semantically equivalent. Then the range of the function $\tau_{cbv}(N)$ is $T\langle\Delta\rangle$. □

Especially in the field of transformation relations and transformation strategies it is important to be able to compare the source and the target of the transformation in order to know whether the transformation makes sense or not. Therefore an appropriate measure has to be defined. Here we consider the number of steps to compute the same normalform as complexity measure.

Definition 3.18 Let $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma, \Delta', R')$ be two macro tree transducers with register functions.

1. N' is *at least as efficient as* N if, for every syntax-directed expression $\psi \in sdExp(gr(FS) \cap gr(FS'), \Sigma, \Delta \cap \Delta')$ the following holds: if there is some $a \geq 0$ and an irreducible $\xi \in sdExp(gr(FS) \cap gr(FS'), \Sigma, \Delta \cap \Delta')$ such that
$$\psi \overset{cbv}{\Rightarrow}_N^{\ a} \xi,$$
then there is a b with $0 \leq b \leq a$ such that
$$\psi \overset{cbv}{\Rightarrow}_{N'}^{\ b} \xi.$$

2. N' is *sometimes better than* N if there are a syntax-directed expression $\psi \in sdExp(gr(FS) \cap gr(FS'), \Sigma, \Delta \cap \Delta')$, numbers a and b, and an irreducible $\xi \in sdExp(gr(FS) \cap gr(FS'), \Sigma, \Delta \cap \Delta')$ such that
$$\psi \overset{cbv}{\Rightarrow}_N^{\ a} \xi, \quad \psi \overset{cbv}{\Rightarrow}_{N'}^{\ b} \xi, \text{ and } b < a.$$

3. N' is *more efficient than* N if N' is at least as efficient as N and N' is sometimes better than N. □

4 Transformation relations

In this section we define three different transformation relations on macro tree transducers with register functions: splitting, sharing, and tupling.

4.1 Splitting

As before, let $N = (FS, \Sigma, \Delta, R)$ be an arbitrary, but fixed macro tree transducer with register functions. In this subsection we present a transformation on macro tree transducers with register functions which allows to introduce further register functions and therewith, further register rules. The task of this transformation, called *splitting transformation relation*, is to create a rule the right-hand side of which is a register function call and the argument list of this call contains ground function calls with equal argument lists. These ground function calls are candidates for the transformations sharing and tupling. Hence, the splitting transformation relation is only a preparation for the other transformation relations. We note that this splitting technique has been used in [Vog90].

We illustrate the principle of the splitting transformation relation by the macro tree transducer M_1 of Example 3.3. We consider the right-hand side of the $(extr, \sigma)$-rule in R_1:

$$extr(\sigma(x_1, x_2), y_1) \quad \rightarrow \quad \sigma(lift(x_2, lift(x_1, y_1)), lift(x_1, y_1)).$$

There, the ground function call $lift(x_1, y_1)$ occurs twice, i.e., we have two ground function calls with equal argument lists. This rule is splitted by the splitting transformation relation into the following two rules:

$$\begin{array}{rcl} extr(\sigma(x_1, x_2), y_1) & \rightarrow & \langle extr, \sigma, 1\rangle(x_1, x_2, y_1, lift(x_1, y_1), lift(x_1, y_1)) \\ \langle extr, \sigma, 1\rangle(x_1, y_1, z_1, z_2) & \rightarrow & \sigma(lift(x_2, z_1), z_2). \end{array}$$

Roughly speaking, the right-hand side ζ of the original rule is changed into a register function call where its argument list contains (besides the variables x_1, x_2, and y_1) some ground function calls of ζ. As a preparation for the sharing, we are in particular interested to extract ground function calls from ζ which have the same argument list. The second rule is necessary to compute the context of the extracted ground function calls. Hereby, every variable in Z denotes the value of a ground function call. This context is indeed nothing else but the old right-hand side in which the extracted ground function calls are replaced by variables in Z.

To determine the appropriate ground function calls which are to be extracted, we define the cut through a right-hand side.

Definition 4.1 Let $\zeta \in RHS(N)$. The *cut through* ζ, denoted by $Cut(\zeta)$, is the maximal subset $S \subseteq path(\zeta)$ such that the following conditions hold:

- If $root(\zeta) \in reg(FS)$, then $S = \emptyset$. (split only ground right-hand sides)
- For every $w \in S$, $label(\zeta, w) \in gr(FS)$. (labeled by ground function)
- For every $w \in S$, $arglist(sub(\zeta, w))$ *in* $\mathcal{L}(T\langle\Delta\rangle(V))$. (no nesting in arguments)

- If $S \neq \emptyset$, then there exist paths $w, v \in S$ with $w \neq v$ such that $arglist(sub(\zeta, w)) = arglist(sub(\zeta, v))$. (splitting makes sense) $\square$

The first condition makes sure that only rules with ground right-hand sides are considered. It would be possible to define splitting also on rules with register functions in their right-hand side, but in our context we do not need such a general form of splitting. As explained before, we take only ground function calls under consideration (cf. the second condition) which are closest to the leaves of the right-hand side (cf. the third condition). For the set of paths which determine the listed function calls, we require that there are at least two different paths in it which lead to function calls with equal argument list (cf. the fourth condition). We will see in the next sections, how this property is used.

Consider, e.g., the macro tree transducer M_1 of Example 3.3: there are two rules, namely the $(lift, \sigma)$-rule and the $(extr, \sigma)$-rule of which the cuts through their right-hand sides are not empty. More precisely, $\{1, 21, 22\}$ and $\{12, 2\}$ are the cuts through the right-hand sides of the $(lift, \sigma)$-rule and the $(extr, \sigma)$-rule, respectively.

Recall that the left-hand side of an (f, σ)-rule has the form either $f(\sigma(\ldots), \ldots)$ or $\langle f, \sigma, i\rangle(\ldots)$.

Definition 4.2 Let M be a macro tree transducer. The *splitting transformation relation* with respect to M is the binary relation $\vdash_{split,M} \subseteq \mathcal{N} \times \mathcal{N}$ defined as follows. Let $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma, \Delta, R')$ be two macro tree transducers with register functions. Then $N \vdash_{split,M} N'$ iff the following conditions hold.

- M and N are semantically equivalent.
- There is an (f, σ)-rule $l \to \zeta$ in R with $f \in gr(FS)^{(n+1)}$ and $\sigma \in \Sigma^{(k)}$ for some $k, n \geq 0$ such that $Cut(\zeta) = \{w_1, \ldots, w_r\}$ for some $r > 0$ and w.l.o.g. $w_1 \leq_{lex} \ldots \leq_{lex} w_r$.
- The variables in Z occurring in l can be written as list $(z_1, \ldots, z_p)$ for some $p \geq 0$.
- Let the number $i = n_{f,\sigma} + 1$. Then $FS' = FS \cup \{\langle f, \sigma, i\rangle^{(k+n+p+r)}\}$.
- R' is obtained from R by replacing the rule $l \to \zeta$ by the rules

  $$l \to \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_p, sub(\zeta, w_1), \ldots, sub(\zeta, w_r))$$

  and

  $$\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_p, z_{p+1}, \ldots, z_{p+r}) \to \zeta[w_1 \leftarrow z_{p+1}, \ldots, w_r \leftarrow z_{p+r}]$$

  where $\tilde{x}$ and $\tilde{y}$ denote the sequences $x_1, \ldots, x_k$ and $y_1, \ldots, y_n$, respectively.

$\square$

Note that the splitting transformation relation can only be applied, if there is at least one rule with a ground right-hand side which has a nonempty cut. Also note that, if l has the form $f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n)$, then we assume that $n_{f,\sigma} = 0$. Recall that otherwise $n_{f,\sigma}$ denotes the number of (f, σ)-register functions.

Now we illustrate the splitting transformation relation on an example.

Example 4.3 Consider the macro tree transducer M_1 of Example 3.3. As mentioned before, the cut through the right-hand side $\zeta_{extr,\sigma}$ of the $(extr, \sigma)$-ground rule is the set $\{12, 2\}$. We execute the step $M_1 \vdash_{split,M_1} N_1$.

Let us stepwise examine the conditions of Definition 4.2.

- Trivially, M_1 is semantically equivalent to itself.
- Consider the $(extr, \sigma)$-ground rule. The cut through its right-hand side is $\{12, 2\}$. Note that $r = 2$.
- There does not occur any variable in Z in the left-hand side of the $(extr, \sigma)$-ground rule, i.e., $p = 0$.
- Since $card((extr, \sigma)\text{-}reg(FS)) = 0$, it follows that $i = 1$ and $FS'_1 = \{lift^{(2)}, extr^{(2)}, \langle extr, \sigma, 1\rangle^{(5)}\}$
- R'_1 is the set of rules which are shown in Figure 13.

Therewith the conditions are fulfilled and $M_1 \vdash_{split, M_1} N_1$.

$$
\begin{array}{rcl}
lift(\alpha, y_1) & \rightarrow & y_1 \\
lift(\sigma(x_1, x_2), y_1) & \rightarrow & \sigma(lift(x_1, y_1), \sigma(extr(x_2, y_1), extr(x_1, y_1))) \\
extr(\alpha, y_1) & \rightarrow & \alpha \\
extr(\sigma(x_1, x_2), y_1) & \rightarrow & \langle extr, \sigma, 1\rangle(x_1, x_2, y_1, lift(x_1, y_1), lift(x_1, y_1)) \\
\langle extr, \sigma, 1\rangle(x_1, x_2, y_1, z_1, z_2) & \rightarrow & \sigma(lift(x_2, z_1), z_2)
\end{array}
$$

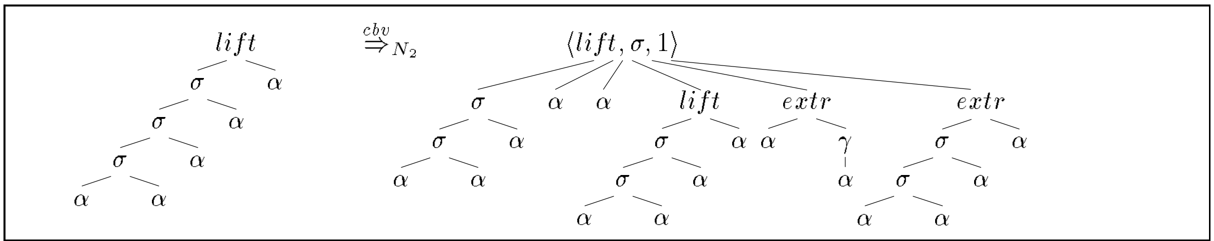
Figure 13: Rules of the macro tree transducer N_1 with register functions.

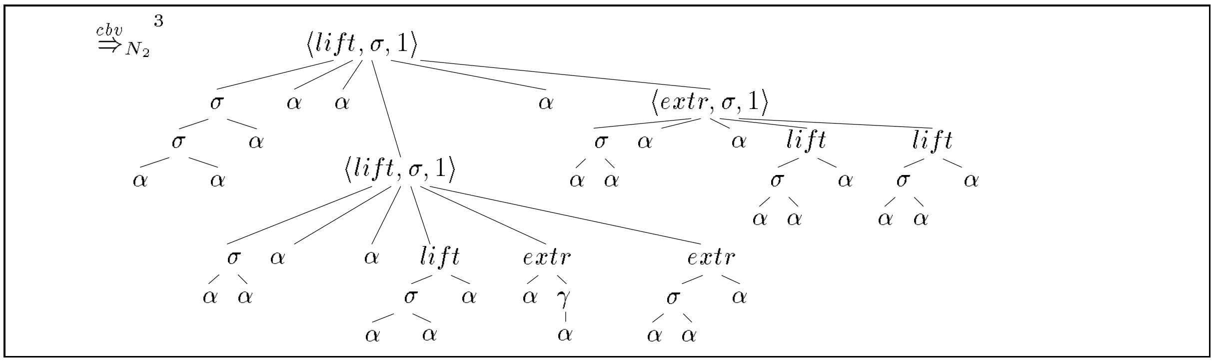
If we assume that N_1 and M_1 are semantically equivalent (this fact is proven in the sequel), then the splitting transformation relation can be again applied to N_1 by considering the $(lift, \sigma)$-rule with $\{1, 21, 22\}$ as cut through $\zeta_{lift,\sigma}$. Then $N_1 \vdash_{split, M_1} N_2$ with $N_2 = (FS_2, \Sigma_1, \Delta_1, R_2)$ where $FS_2 = \{lift^{(2)}, extr^{(2)}, \langle extr, \sigma, 1\rangle^{(5)}, \langle lift, \sigma, 1\rangle^{(6)}\}$ and R_2 is given in Figure 14.

$$
\begin{array}{rcl}
lift(\alpha, y_1) & \rightarrow & y_1 \\
lift(\sigma(x_1, x_2), y_1) & \rightarrow & \langle lift, \sigma, 1\rangle(x_1, x_2, y_1, lift(x_1, y_1), extr(x_2, y_1), extr(x_1, y_1)) \\
\langle lift, \sigma, 1\rangle(x_1, x_2, y_1, z_1, z_2, z_3) & \rightarrow & \sigma(z_1, \sigma(z_2, z_3)) \\
extr(\alpha, y_1) & \rightarrow & \alpha \\
extr(\sigma(x_1, x_2), y_1) & \rightarrow & \langle extr, \sigma, 1\rangle(x_1, x_2, y_1, lift(x_1, y_1), lift(x_1, y_1)) \\
\langle extr, \sigma, 1\rangle(x_1, x_2, y_1, z_1, z_2) & \rightarrow & \sigma(lift(x_2, z_1), z_2)
\end{array}
$$

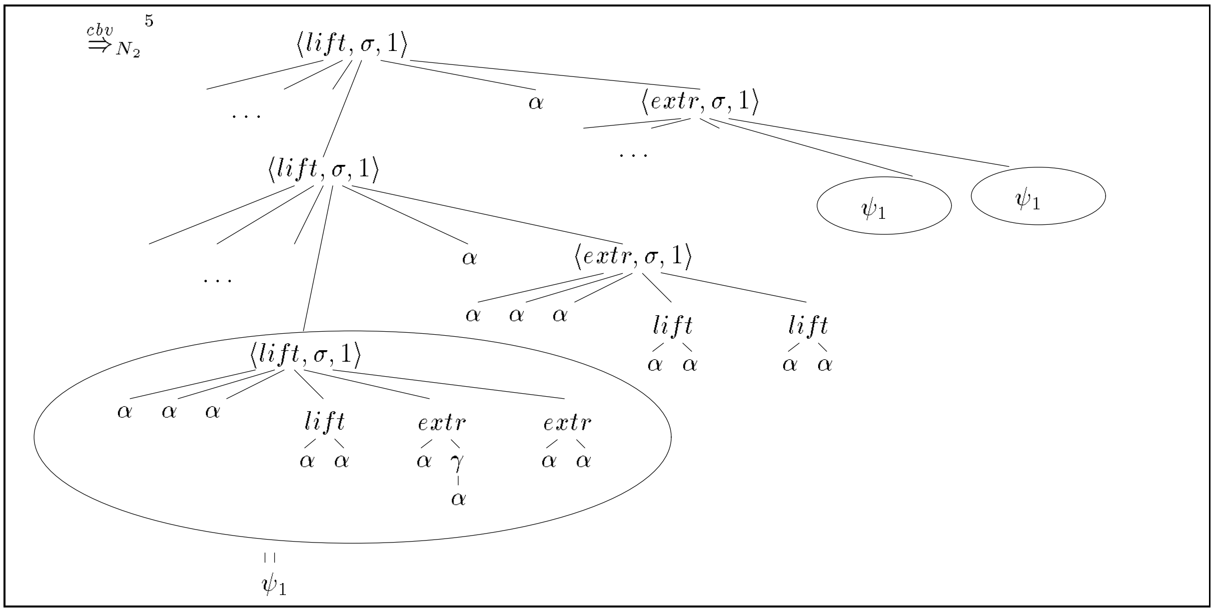
Figure 14: Rules of the macro tree transducer N_2 with register functions.

The following figures show a derivation of the function call $lift(\sigma(\sigma(\sigma(\alpha, \alpha), \alpha), \alpha), \alpha)$ by $\stackrel{cbv}{\Rightarrow}_{N_2}$. Note that the result is computed bottom-up in 29 steps.

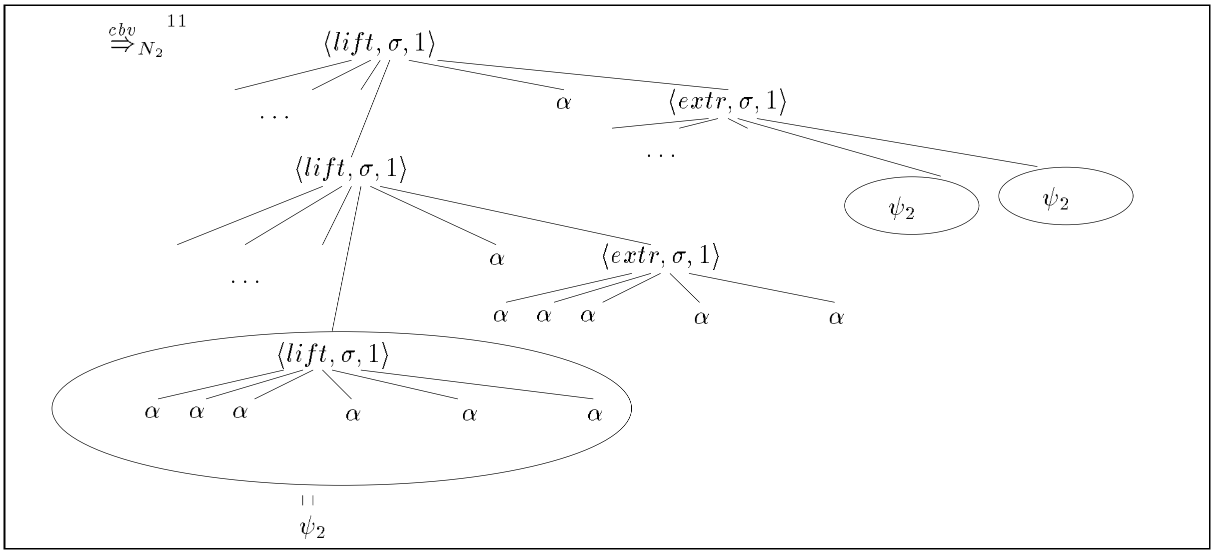




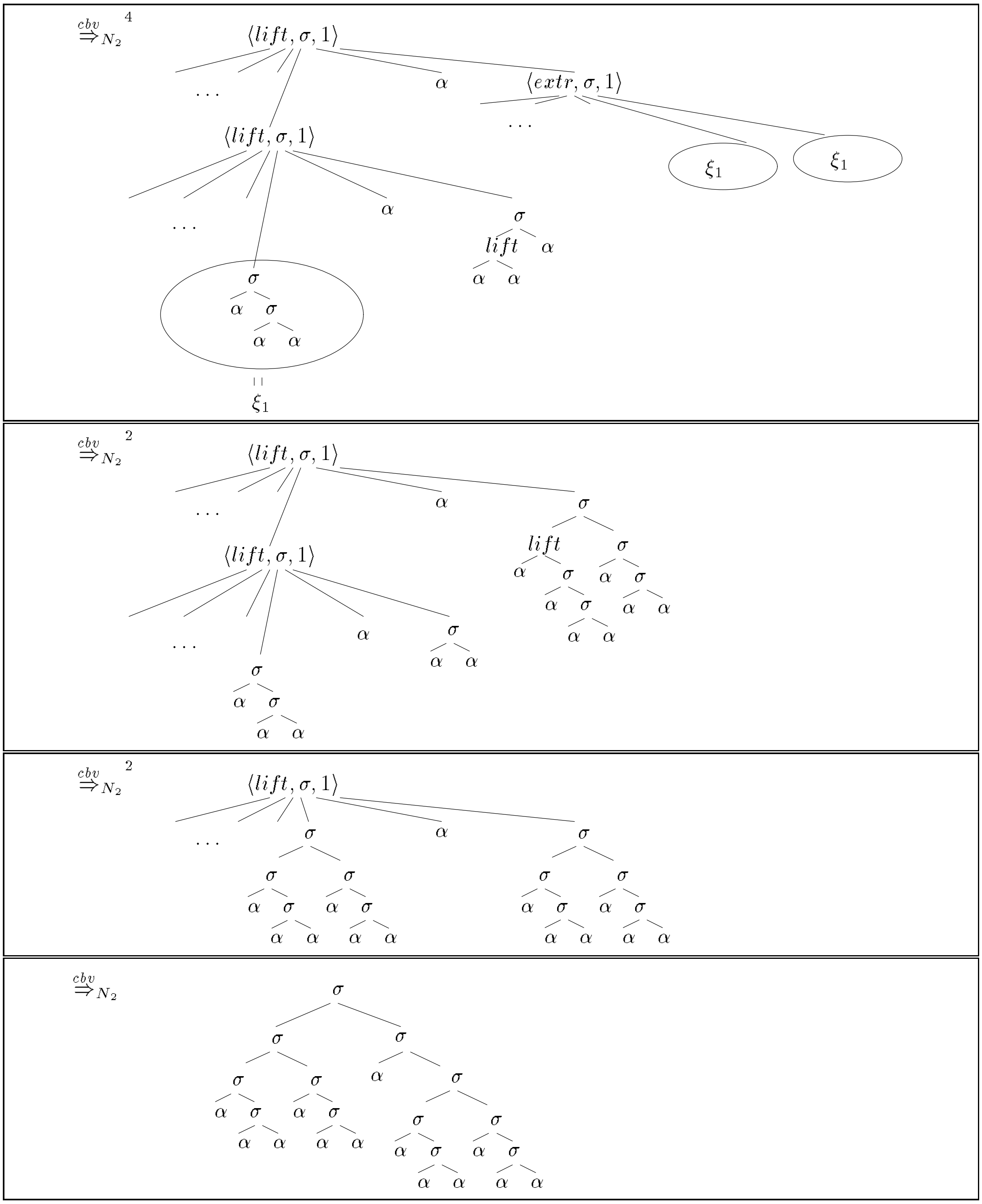
$\overset{cbv}{\Rightarrow}_{N_2}^{3}$
$\langle lift, \sigma, 1\rangle$
σ α α α $\langle extr, \sigma, 1\rangle$
σ α σ α α $lift$ $lift$
α α $\langle lift, \sigma, 1\rangle$ α α σ α σ α
α α α α
σ α α $lift$ $extr$ $extr$
α α σ α α γ σ α
α α α α α



$\overset{cbv}{\Rightarrow}_{N_2}^{5}$
$\langle lift, \sigma, 1\rangle$
$\dots$ α $\langle extr, \sigma, 1\rangle$
$\dots$
$\langle lift, \sigma, 1\rangle$ ψ_1 ψ_1
$\dots$ α $\langle extr, \sigma, 1\rangle$
α α α $lift$ $lift$
α α α α
$\langle lift, \sigma, 1\rangle$
α α α $lift$ $extr$ $extr$
α α α γ α α
α
ψ_1



$\overset{cbv}{\Rightarrow}_{N_2}^{11}$
$\langle lift, \sigma, 1\rangle$
$\dots$ α $\langle extr, \sigma, 1\rangle$
$\dots$
$\langle lift, \sigma, 1\rangle$ ψ_2 ψ_2
$\dots$ α $\langle extr, \sigma, 1\rangle$
α α α α α
$\langle lift, \sigma, 1\rangle$
α α α α α α
ψ_2



Remark 4.4 Let M be a macro tree transducer. If, for two macro tree transducers $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma, \Delta, R')$ with register functions, $N \vdash_{split,M} N'$, then the following holds:

1. $card(FS') = card(FS) + 1$, in particular, $card(reg(FS')) = card(reg(FS)) + 1$ and

$tup(FS) = tup(FS')$.

2. $gr(FS') = gr(FS)$.

3. $card(R') = card(R) + 1$.

4. There is exactly one rule $l \to \zeta$ in R such that there are two rules $l \to \zeta'$ and $l' \to \zeta''$ in R' and $R - \{l \to \zeta\} = R' - \{l \to \zeta', l' \to \zeta''\}$. Moreover, $Cut(\zeta) \neq \emptyset$ and $Cut(\zeta') = \emptyset$. The rule $l \to \zeta$ is called *splitted rule of* N *and* N'. $\Box$

Clearly, by splitting one rule into two, the efficiency of the resulting macro tree transducer with register functions decreases with respect to the original transducer. Later (cf. Theorem 5.13) we will see that this increment is compensated by the application of the other two transformation relations.

Corollary 4.5 *Let M be a macro tree transducer. Let $N, N' \in \mathcal{N}$ and $N \vdash_{split,M} N'$. N is more efficient than N'. In particular, assuming that $l \to \zeta$ is the splitted rule of N and N', then if the evaluation of a syntax-directed expression ψ by $\overset{cbv}{\Rightarrow}_N$ needs L derivation steps in which the rule $l \to \zeta$ is applied K times, then the evaluation of ψ by $\overset{cbv}{\Rightarrow}_{N'}$ needs $L + K$ derivation steps.*

Proof: Let $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma, \Delta, R')$ be macro tree transducers with register functions such that $N \vdash_{split,M} N'$. Let $l \to \zeta$ be the splitted rule of N and N' and let $R' = (R - \{l \to \zeta\}) \cup \{l \to \zeta', l' \to \zeta''\}$. To prove that N is more efficient than N' we have to prove the following statements: N is at least as efficient as N' and N is sometimes more efficient than N' (cf. Definition 3.18). In particular, we have to prove that, if for some syntax-directed expression $\psi \in sdExp(gr(FS), \Sigma, \Delta)$ there are an $a \geq 0$ and an irreducible $\psi' \in sdExp(FS \cap FS', \Sigma, \Delta)$ such that $\psi \overset{cbv}{\Rightarrow}_{N'}^{a} \psi'$, then there is a $b \leq a$ such that $\psi \overset{cbv}{\Rightarrow}_{N}^{b} \psi'$ (at least as efficient). The property "sometimes more efficient" corresponds to the second statement of the corollary: we have to prove that K applications of the rule $l \to \zeta'$ during the $\overset{cbv}{\Rightarrow}_{N'}$-derivation of a syntax-directed expression ψ lead to a $\overset{cbv}{\Rightarrow}_N$-derivation of ψ which is K steps shorter.

Let us consider an arbitrary derivation of a syntax-directed expression ψ by $\overset{cbv}{\Rightarrow}_{N'}$:

$$\psi = \psi_1 \overset{cbv}{\Rightarrow}_{N'} \psi_2 \overset{cbv}{\Rightarrow}_{N'} \ldots \overset{cbv}{\Rightarrow}_{N'} \psi_\nu \overset{cbv}{\Rightarrow}_{N'} \psi_{\nu+1} \overset{cbv}{\Rightarrow}_{N'} \ldots \overset{cbv}{\Rightarrow}_{N'} \psi_{\nu+\mu} \overset{cbv}{\Rightarrow}_{N'}^{*} \psi_m = \psi'.$$

Let $\nu \in [m]$ such that for every $i < \nu$, in the step $\psi_i \overset{cbv}{\Rightarrow}_{N'} \psi_{i+1}$ the rule $l \to \zeta'$ is *not* applied and $\psi_\nu \overset{cbv}{\Rightarrow}_{N'} \psi_{\nu+1}$ where $\psi_{\nu+1} = \psi_\nu[w \leftarrow \varphi(\zeta')]$ and $sub(\psi_\nu, w) = \varphi(l)$ for some path w and matching substitution φ. Furthermore, let $\psi_{\nu+\mu} = \psi_\nu[w \leftarrow \xi_w]$ and ξ_w is the normalform of $sub(\psi_\nu, w)$.

Since all other rules apart from the splitted rule are equal in N and N', it holds that there is an equal $\overset{cbv}{\Rightarrow}_N$-derivation until ψ_ν is reached:

$$\psi = \psi_1 \overset{cbv}{\Rightarrow}_N \psi_2 \overset{cbv}{\Rightarrow}_N \psi_3 \ldots \overset{cbv}{\Rightarrow}_N \psi_\nu$$

Note that by assumption ψ does not contain register function calls, but the syntax-directed expressions ψ_i with $i > 1$ may contain register function calls, because the applied rules may introduce register function calls.

By Definition 4.2 the splitted rule can be of one of the following form: either the root of the left-hand side l is a ground function or a register function. The right-hand side ζ has to be a ground right-hand side.

If we can prove that there is a $\stackrel{cbv}{\Rightarrow}_N$-derivation of ψ_ν to $\psi_{\nu+\mu}$ which is K steps shorter where K denotes the number of applications of the splitted rule, than the corresponding $\stackrel{cbv}{\Rightarrow}_{N'}$-derivation, then the statements are proven (for the rest of the derivation, the same argumentation as above holds).

Since $\psi_{\nu+\mu} = \psi_\nu[w \leftarrow \xi_w]$, it suffices to prove the following two statements.

1. Let $s = \sigma(s_1, \ldots, s_k) \in T\langle\Sigma\rangle$ and $t_1, \ldots, t_n \in T\langle\Delta\rangle$. If the left-hand side l of the splitted rule has the form $f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n)$ and there is an $a \geq 1$ and an irreducible $\xi \in sdExp(FS, \Sigma, \Delta)$ such that
$$f(s, t_1, \ldots, t_n) {\stackrel{cbv}{\Rightarrow}_{N'}}^{a} \xi,$$
then there is a $b \geq 1$ and a derivation
$$f(s, t_1, \ldots, t_n) {\stackrel{cbv}{\Rightarrow}_{N}}^{b} \xi$$
and $b < a$. If the splitted rule is applied K times in the derivation by $\stackrel{cbv}{\Rightarrow}_{N'}$, then $b = a - K$.

2. Let $s_1, \ldots, s_k \in T\langle\Sigma\rangle$, $t_1, \ldots, t_n \in T\langle\Delta\rangle$, and $\psi_1, \ldots, \psi_r \in sdExp(FS, \Sigma, \Delta)$. If the left-hand side l of the splitted rule has the form $\langle f, \sigma, i-1\rangle(\tilde{x}, \tilde{y}, u_1, \ldots, u_{r_{\langle f,\sigma,i-1\rangle}})$ and there is an $a \geq 1$ and an irreducible $\xi \in sdExp(FS, \Sigma, \Delta)$ such that
$$\langle f, \sigma, i-1\rangle(\tilde{s}, \tilde{t}, \psi_1, \ldots, \psi_{r_{\langle f,\sigma,i-1\rangle}}) {\stackrel{cbv}{\Rightarrow}_{N'}}^{a} \xi,$$
then there is a $b \geq 1$ and a derivation
$$\langle f, \sigma, i-1\rangle(\tilde{s}, \tilde{t}, \psi_1, \ldots, \psi_{r_{\langle f,\sigma,i-1\rangle}}) {\stackrel{cbv}{\Rightarrow}_{N}}^{b} \xi$$
and $b < a$. If the splitted rule is applied K times in the derivation by $\stackrel{cbv}{\Rightarrow}_{N'}$, then $b = a - K$.

No other forms of l are possible.

Proof.

1. Let $l = f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n)$ and let $Cut(\zeta) = \{w_1, \ldots, w_r\}$ for some $r > 0$ and let, w.l.o.g., $w_1 \leq_{lex} \ldots \leq_{lex} w_r$. Then by definition, for every $j \in [r]$, $sub(\zeta, w_j)$ is a ground function call of which the argument list is a list over $T\langle\Delta\rangle(X_k \cup Y_n)$. In particular, the first argument is a variable in X_k, the other arguments are elements of $T\langle\Delta\rangle(Y_n)$. Note that the rules $l \to \zeta'$ and $l' \to \zeta''$ have the forms $f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n) \to \langle f, \sigma, 1\rangle(\tilde{x}, \tilde{y}, sub(\zeta, w_1), \ldots, sub(\zeta, w_r))$ and $\langle f, \sigma, 1\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_r) \to \zeta[w_1 \leftarrow z_1, \ldots, w_r \leftarrow z_r]$, respectively.

   We prove the statement by induction over the height of the input tree. We have to start the proof with $height(s) = 1$, because in the case $height(s) = 0$, there is no (f, s)-rule which contains any ground function call in its right-hand side. Note that, for the same reason, the rules for the nullary constructors are equal in R and R'.

(i) Let $height(s) = 1$, i.e., $s = \sigma(\alpha_1, \ldots, \alpha_k)$ for some $k > 0$, $\sigma \in \Sigma^{(k)}$, and $\alpha_1, \ldots, \alpha_k \in \Sigma^{(0)}$. Let us consider the $\overset{cbv}{\Rightarrow}_{N'}$ derivation:

$$
\begin{array}{lll}
f(s, t_1, \ldots, t_n) & \overset{cbv}{\Rightarrow}_{N'} & \langle f, \sigma, 1\rangle(\alpha_1, \ldots, \alpha_k, t_1, \ldots, t_n, \varphi(sub(\zeta, w_1)), \ldots, \varphi(sub(\zeta, w_r))) \\
& & \text{where } \varphi = [x_1/\alpha_1, \ldots, x_k/\alpha_k, y_1/t_1, \ldots, y_n/t_n] \\
& \overset{cbv}{\Rightarrow}_{N'}{}^{r} & \langle f, \sigma, 1\rangle(\alpha_1, \ldots, \alpha_k, t_1, \ldots, t_n, \xi_1, \ldots, \xi_r) \\
& & \text{where, for every } j \in [r],\ sub(\varphi(\zeta), w_j) \overset{cbv}{\Rightarrow}_{N'} \xi_j \\
& & \text{and } \xi_j \in T\langle \Delta \rangle \\
& \overset{cbv}{\Rightarrow}_{N'} & \varphi(\zeta[w_1 \leftarrow z_1, \ldots, w_r \leftarrow z_r])[z_1/\xi_1, \ldots, z_r/\xi_r] \\
& & \text{and } \varphi(\zeta[w_1 \leftarrow z_1, \ldots, w_r \leftarrow z_r])[z_1/\xi_1, \ldots, z_r/\xi_r] \text{ is equal} \\
& & \text{to } \varphi(\zeta)[w_1 \leftarrow \xi_1, \ldots, w_r \leftarrow \xi_r] \\
& \overset{cbv}{\Rightarrow}_{N'}{}^{m} & \xi \\
& & \text{for some } m \geq 0 \text{ and } \xi \text{ is irreducible.}
\end{array}
$$

Since the rules for the nullary constructors are equal in R and in R', it follows that, for every $j \in [r]$, also $sub(\varphi(\zeta), w_j) \overset{cbv}{\Rightarrow}_N \xi_j$. Hence, the corresponding $\overset{cbv}{\Rightarrow}_N$-derivation has the following form:

$$
\begin{array}{lll}
f(s, t_1, \ldots, t_n) & \overset{cbv}{\Rightarrow}_{N} & \varphi(\zeta) \\
& & \text{where } \varphi = [x_1/\alpha_1, \ldots, x_k/\alpha_k, y_1/t_1, \ldots, y_n/t_n] \\
& \overset{cbv}{\Rightarrow}_{N}{}^{r} & \varphi(\zeta)[w_1 \leftarrow \xi_1, \ldots, w_r \leftarrow \xi_r] \\
& \overset{cbv}{\Rightarrow}_{N}{}^{m} & \xi \\
& & \text{because during the further derivation only function calls} \\
& & \text{have to be evaluated which have a nullary constructor } \alpha_j \\
& & \text{with } j \in [k] \text{ as recursion argument.}
\end{array}
$$

Define $a = 1 + r + 1 + m$ and $b = 1 + r + m$, then $a > b$ and we have seen that the splitted rule was applied exactly once, i.e., $K = 1$.

(ii) Let $height(s) = \rho > 1$, i.e., $s = \sigma(s_1, \ldots, s_k)$ for some $k > 0$, $s_1, \ldots, s_k \in T\langle\Sigma\rangle$ with $max\{height(s_i) \mid i \in [k]\} = \rho - 1$. Again, we consider the derivation by $\overset{cbv}{\Rightarrow}_{N'}$:

$$
\begin{array}{lll}
f(s, t_1, \ldots, t_n) & \overset{cbv}{\Rightarrow}_{N'} & \langle f, \sigma, 1\rangle(\tilde{s}, \tilde{t}, \varphi(sub(\zeta, w_1)), \ldots, \varphi(sub(\zeta, w_r))) \\
& & \text{where } \varphi = [x_\nu/s_\nu; \nu \in [k]][y_\mu/t_\mu; \mu \in [n]] \\
& \overset{cbv}{\Rightarrow}_{N'}{}^{\Sigma_{j=1}^{r} a_j} & \langle f, \sigma, 1\rangle(\tilde{s}, \tilde{t}, \xi_1, \ldots, \xi_r) \\
& & \text{where, for every } j \in [r],\ sub(\varphi(\zeta), w_j) \overset{cbv}{\Rightarrow}_{N'}{}^{a_j} \xi_j \text{ and} \\
& & \xi_j \text{ is irreducible. Note that the recursion argument} \\
& & \text{of } sub(\varphi(\zeta), w_j) \text{ is at most of height } \rho - 1. \\
& \overset{cbv}{\Rightarrow}_{N'} & \varphi(\zeta[w_1 \leftarrow z_1, \ldots, w_r \leftarrow z_r])[z_1/\xi_1, \ldots, z_r/\xi_r] \\
& & \text{and } \varphi(\zeta[w_1 \leftarrow z_1, \ldots, w_r \leftarrow z_r])[z_1/\xi_1, \ldots, z_r/\xi_r] \text{ is equal} \\
& & \text{to } \varphi(\zeta)[w_1 \leftarrow \xi_1, \ldots, w_r \leftarrow \xi_r] \\
& \overset{cbv}{\Rightarrow}_{N'}{}^{m} & \xi
\end{array}
$$

Let us consider the corresponding $\overset{cbv}{\Rightarrow}_N$-derivation:

$$
\begin{array}{lll}
f(s,t_1,\ldots,t_n) & \overset{cbv}{\Rightarrow}_N & \varphi(\zeta) \\
 & & \text{where } \varphi = [x_\nu/s_\nu;\nu\in[k]][y_\mu/t_\mu;\mu\in[n]] \\
 & \overset{cbv}{\Rightarrow}_N^{\Sigma_{i=1}^r b_i} & \varphi(\zeta)[w_1 \leftarrow \xi_1,\ldots,w_r \leftarrow \xi_r] \\
 & & \text{where by induction hypothesis, for every } j\in[r], \\
 & & sub(\varphi(\zeta),w_j)\overset{cbv}{\Rightarrow}_N^{b_j}\xi_j \text{ and } b_j \le a_j, \text{ more precisely,} \\
 & & a_j = b_j + K_j \text{ for some } K_j \ge 0 \text{ and } K_j \text{ denotes the} \\
 & & \text{number of applications of the splitted rule.} \\
 & \overset{cbv}{\Rightarrow}_N^{m} & \xi
\end{array}
$$

Define $a = 2+m+\Sigma_{j=1}^r a_j = 2+m+\Sigma_{j=1}^r(b_j+K_j)$ and $b = 1+m+\Sigma_{j=1}^r b_j$. Hence, $b<a$ and in particular, $b = a-(1+\Sigma_{j=1}^r K_j)$.

2. This case is similar to 1. □

Lemma 4.6 *Let N and N' be macro tree transducers with register functions. If $N \vdash_{split,M} N'$, then*

(a) $\tau_{cbv}(N) = \tau_{cbv}(N')$.

(b) N' and M are semantically equivalent.

Proof.
(a) Since $\tau_{cbv}(N)$ and $\tau_{cbv}(N')$ are total functions, it suffices to prove that $\tau_{cbv}(N') \subseteq \tau_{cbv}(N)$. Consider a derivation by N'. Then, since N is more efficient than N' (cf. Corollary 4.5), there is a corresponding derivation by N leading to the same normalform (cf. Definition 3.18). Thus $\tau_{cbv}(N') \subseteq \tau_{cbv}(N)$.

(b) Let $N = (FS,\Sigma,\Delta,R)$ and $N' = (FS',\Sigma,\Delta,R')$. M and N are semantically equivalent (cf. Definition 4.2). Since $tup(FS) = tup(FS')$ (cf. Remark 4.4 1.) and by Statement (a) it holds that M and N' are semantically equivalent. □

The previous facts are illustrated by comparing Examples 3.3, 3.10, and 4.3. In Example 4.3 the macro tree transducer N_2 with register functions is given such that $M_1 \vdash_{split,M_1} N_1$ and $N_1 \vdash_{split,M_1} N_2$. In the $\overset{cbv}{\Rightarrow}_{M_1}$-derivation of the function call $t = lift(\sigma(\sigma(\sigma(\alpha,\alpha),\alpha,)\alpha),\alpha)$ (cf. Example 3.10, Page 18) the result is computed in 22 steps, whereas the $\overset{cbv}{\Rightarrow}_{N_2}$-derivation of t needs 29 steps for the same result. In the $\overset{cbv}{\Rightarrow}_{M_1}$-derivation, the splitted $(extr,\sigma)$-rule of M_1 and N_1 is applied twice and the splitted $(lift,\sigma)$-rule of N_1 and N_2 is applied 5 times. This corresponds to the fact that the $\overset{cbv}{\Rightarrow}_{N_2}$-derivation needs 7 steps more. It follows that the $\overset{cbv}{\Rightarrow}_{N_1}$-derivation of t (which was not given) would need 24 steps.

Note that M_1 and N_2 are semantically equivalent (cf. Lemma 4.6).

Lemma 4.7 *The relation $\vdash_{split,M}$ is locally confluent.*

Proof. Let $N = (FS,\Sigma,\Delta,R)$, $N_1 = (FS_1,\Sigma,\Delta,R_1)$, and $N_2 = (FS_2,\Sigma,\Delta,R_2)$ be macro tree transducers with register functions such that $N \vdash_{split,M} N_1$ and $N \vdash_{split,M} N_2$ with $N_1 \ne N_2$. We show that there exists a macro tree transducer $N_3 = (FS_3,\Sigma,\Delta,R_3)$ with register functions with $N_1 \vdash_{split,M} N_3$ and $N_2 \vdash_{split,M} N_3$.

Let $l_1 \to \zeta_1$ be the splitted rule of N and N_1 and let $l_2 \to \zeta_2$ be the splitted rule of N and N_2. Since $N_1 \neq N_2$ and the cut through a right-hand side is unique, it holds that $l_1 \to \zeta_1$ and $l_2 \to \zeta_2$ are different rules.

Let us consider N_1 and N_2.

N_1: By definition of the splitting transformation relation, the rule $l_2 \to \zeta_2$ is a rule of R_1. Let $l_1 \to \langle f_1, \sigma_1, i_1\rangle(\tilde{x}, \tilde{y}, \theta_{1,1}, \ldots, \theta_{1,r_1})$ and $\langle f_1, \sigma_1, i_1\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_1}) \to \zeta_1'$ be the two rules in R_1 which have replaced the rule $l_1 \to \zeta_1$.

N_2: The rule $l_1 \to \zeta_1$ is in R_2.
Let $l_2 \to \langle f_2, \sigma_2, i_2\rangle(\tilde{x}, \tilde{y}, \theta_{2,1}, \ldots, \theta_{2,r_2})$ and $\langle f_2, \sigma_2, i_2\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_2}) \to \zeta_2'$ be the two rules in R_2 which have replaced the rule $l_2 \to \zeta_2$.

The splitting transformation can be applied to N_1 and N_2 because the rules $l_2 \to \zeta_2$ and $l_1 \to \zeta_1$ are in R_1 and R_2, respectively. Since the form of the new rules only depends on the given rules, $N_1 \vdash_{split,M} N_3$ and $N_2 \vdash_{split,M} N_3$ by trivial comparison. □

Lemma 4.8 *The relation $\vdash_{split,M}$ is noetherian.*

Proof. It can easily be seen that $\vdash_{split,M}$ is only applicable to macro tree transducers with register functions with rules of which at least one right-hand side has a nonempty cut. By one step of $\vdash_{split,M}$ such a rule is replaced by two rules. One of the new rules has an empty cut, the other rule has on its right-hand side function calls with a nesting depth of ground function calls decremented by one (in comparison to the original rule). Since the nesting depth is finite, the number of applications of $\vdash_{split,M}$ to these rules is also finite. Since the number of new rules is also finite, the number of steps of $\vdash_{split,M}$ is finite for every macro tree transducer with register functions. □

Lemma 4.9 *The relation $\vdash_{split,M}$ is confluent and noetherian.*

Proof. This result follows directly by the fact that $\vdash_{split,M}$ is locally confluent and noetherian (cf. previous lemmas) and by Lemma 2.4 of [Hue80]. □

Hence, for every macro tree transducer with register functions there exists a normalform with respect to $\vdash_{split,M}$. In Example 4.3, N_2 is the normalform of M_1 w.r.t. $\vdash_{split,M_1}$.

4.2 Sharing

In the field of graphs and graph rewriting techniques the notion of sharing is well-known. Instead of having several occurrences of the same subgraph, every subgraph can be represented exactly once and it is referred by multiple pointers to it, cf. Figure 15 ([SPvE93, HP91]).

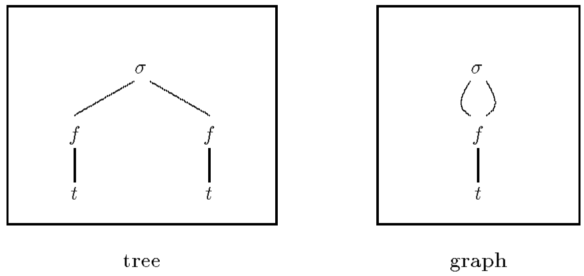


Figure 15: The term $\gamma(f(t), f(t))$ written as a tree and as a graph with shared subgraph.

In our context of trees and macro tree transducers with register functions the avoidance of multiple occurrences of the same subtree cannot be handeled in this way. In the literature, sharing is realized by introducing where-clauses (in the field of program transformation this method is often called abstraction rule) and by associating a kind of graph semantics to these clauses. If, e.g., $\gamma(f(t), f(t))$ is the right-hand side of a rule in a term rewriting system, then this right-hand side is replaced by the right-hand side $\gamma(z, z)$ *where* $z = f(t)$ (cf. e.g. [PP93]). The semantics associated to term rewriting systems with where-clauses has to guarantee that the where-clause is evaluated exactly once.

Since we only want to deal with macro tree transducers with register functions and we do not want to introduce where-clauses with an extra semantic treatment, we choose another way to realize sharing. As we have seen in the previous section, register functions have the property that they only occur in right-hand sides at their root. The following procedure can be applied to the argument list of such a register function: we evaluate each ground function call exactly once under a register function symbol and copy its value several times.

Consider, e.g., the part of the calling graph of the expression $lift(\sigma(\sigma(\sigma(\alpha, \alpha), \alpha), \alpha), \alpha)$ in Figure 16.

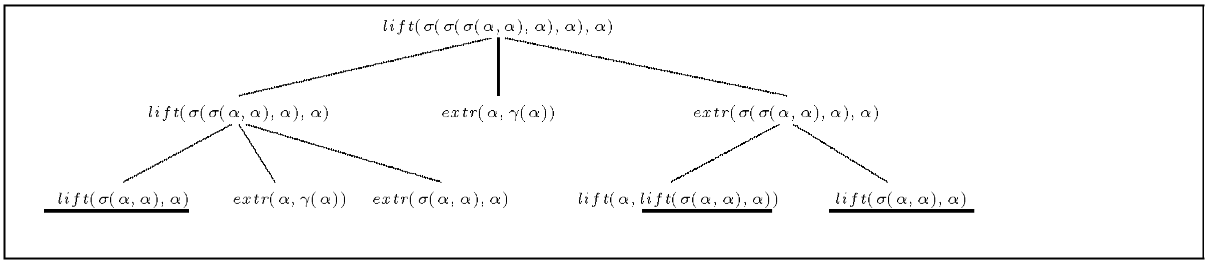


Figure 16: Part of a calling graph of ground functions.

The underlined function call occurs in multiple positions in the tree. In this section we are interested in these positions which have the same predecessor. First we need some technical notions.

Definition 4.10 Let S be an arbitrary set and let $B = (b_1, \ldots, b_r)$ be a list over S with $r > 0$. Furthermore let $B' = delpos(B, double(B))$ with $B' = (b'_1, \ldots, b'_{r'})$. The *index shift associated with* B is the mapping $\varphi_B : Z_r \to Z_r$ such that for every $i \in [r]$, $\varphi_B(z_i) = z_j$, where j is the unique element in $[r']$ such that $b_i = b'_j$. $\square$

Note that, if $B \neq B'$, then $r' < r$. We illustrate this definition by an example. In Figure 17, a list B is shown which contains the elements a and b in multiple positions. These positions can be retrieved by the function *double*. The index shift φ_B associated with B is determined by the

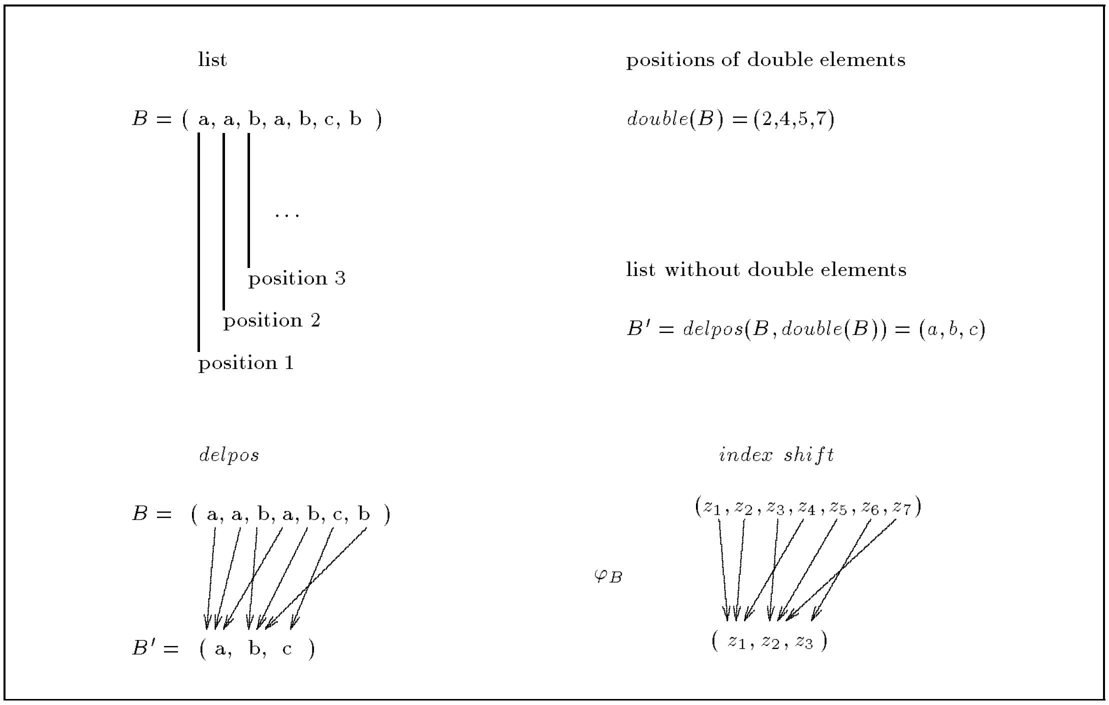


Figure 17: Index shift associated with a list B.

positions of the list B and by the positions of the list $B' = delpos(B, double(B))$. Consider, e.g., the element a which occurs in B in positions 1, 2, and 4. In B', the element a occurs exactly once in position 1. Hence, $\varphi_B(z_1) = \varphi_B(z_2) = \varphi_B(z_4) = z_1$.

We define a criterion which determines, if a macro tree transducer with register functions is ready for sharing in the sense of avoiding multiple evaluation of several occurrences of the same ground function call.

Definition 4.11 Let $N = (FS, \Sigma, \Delta, R)$ be a macro tree transducer with register functions and let $l \to \zeta$ be a rule in R. Then N is *ready for sharing in the rule* $l \to \zeta$ if the following conditions hold:

- $root(\zeta) \in reg(FS)$,
- there exist paths $w_1 \in \mathbb{N}$ and $w_2 \in \mathbb{N}$ with $w_1 \neq w_2$ such that $sub(\zeta, w_1) = sub(\zeta, w_2)$ and $sub(\zeta, w_i) \in F^{gr}_{call}(\zeta)$, and

- the left-hand side l' of the rule $l' \to \zeta'$ in R with $root(l') = root(\zeta)$ is flat. $\square$

Note that "ready for sharing in a rule r" is a property which is completely determined by the form of r and of the rule which has the same root in its left-hand side as r in its right-hand side.

Now we are able to define the sharing transformation relation of macro tree transducers with register functions.

Definition 4.12 Let M be a macro tree transducer. The *sharing transformation relation* with respect to M is the binary relation $\vdash_{share,M} \subseteq \mathcal{N} \times \mathcal{N}$ defined as follows. Let $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma, \Delta, R')$ be two macro tree transducers with register functions. It holds that $N \vdash_{share,M} N'$ iff the following conditions hold:

- M and N are semantically equivalent.
- There is a rule $l \to \zeta$ in R such that N is ready for sharing in $l \to \zeta$. Let $\zeta = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \psi_1, \ldots, \psi_r)$ with $\langle f, \sigma, i\rangle \in reg(FS)^{(k+n+r)}$, $k, n \geq 0$, $r > 0$, and $\tilde{x}$ and $\tilde{y}$ abbreviate the sequences $x_1, \ldots, x_k$ and $y_1, \ldots, y_n$, respectively.
- $FS' = (FS - \{\langle f, \sigma, i\rangle^{(n+k+r)}\}) \cup \{\langle f, \sigma, i\rangle^{(n+k+r')}\}$.
- R' is obtained from R by the following replacement of rules:

  - The rule $l \to \zeta$ is replaced by the rule
  $$l \to \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \psi'_1, \ldots, \psi'_{r'})$$
  where $(\psi'_1, \ldots, \psi'_{r'}) = delpos((\psi_1, \ldots, \psi_r), double((\psi_1, \ldots, \psi_r)))$.
  - The rule $\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_r) \to \zeta \in R$, is replaced by the rule
  $$\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r'}) \to \varphi_{(\psi_1,\ldots,\psi_r)}(\zeta)$$

$\square$

Note that we perform a maximal sharing, i.e., if in a right-hand side ζ the terms $\theta_1, \ldots, \theta_m$ with $m \geq 1$ occur several times, in particular, let each θ_i occur k_i times, then in the new right-hand side each θ_i occur exactly once.

Example 4.13 Let us consider the macro tree transducer $N_2 = (FS_2, \Sigma_2, \Delta_2, R_2)$ with register functions of Example 4.3. The set R_2 of rules is recalled in Figure 18 where we have underlined terms which are candidates for sharing.

N_2 is ready for sharing in the $(extr, \sigma)$-ground rule, because its right-hand side's root is labeled by the register function $\langle extr, \sigma, 1\rangle$, the function call $lift(x_1, y_1)$ occurs twice in its right-hand side at paths 4 and 5, and the left-hand side of the $\langle extr, \sigma, 1\rangle$ rule is flat. Furthermore N_2 and M_1 are semantically equivalent (cf. previous section). Hence, $N_2 \vdash_{share,M_1} N_3$ and N_3 is the macro tree transducer $(FS_3, \Sigma_2, \Delta_2, R_3)$ with register functions with $FS_3 = \{lift^{(2)}, \langle lift, \sigma, 1\rangle^{(6)}, extr^{(2)}, \langle extr, \sigma, 1\rangle^{(4)}\}$ and R_3 is the set of rules as shown in Figure 19.

$$
\begin{array}{rcl}
lift(\alpha, y_1) & \to & y_1 \\
lift(\sigma(x_1,x_2), y_1) & \to & \langle lift,\sigma,1\rangle(x_1,x_2,y_1,lift(x_1,y_1),extr(x_2,\gamma(y_1)),extr(x_1,y_1)) \\
\langle lift,\sigma,1\rangle(x_1,x_2,y_1,z_1,z_2,z_3) & \to & \sigma(z_1,\sigma(z_2,z_3)) \\
 & & \\
extr(\alpha, y_1) & \to & \alpha \\
extr(\sigma(x_1,x_2), y_1) & \to & \langle extr,\sigma,1\rangle(x_1,x_2,y_1,\underline{lift(x_1,y_1)},\underline{lift(x_1,y_1)}) \\
\langle extr,\sigma,1\rangle(x_1,x_2,y_1,z_1,z_2) & \to & \sigma(lift(x_2,z_1),z_2)
\end{array}
$$

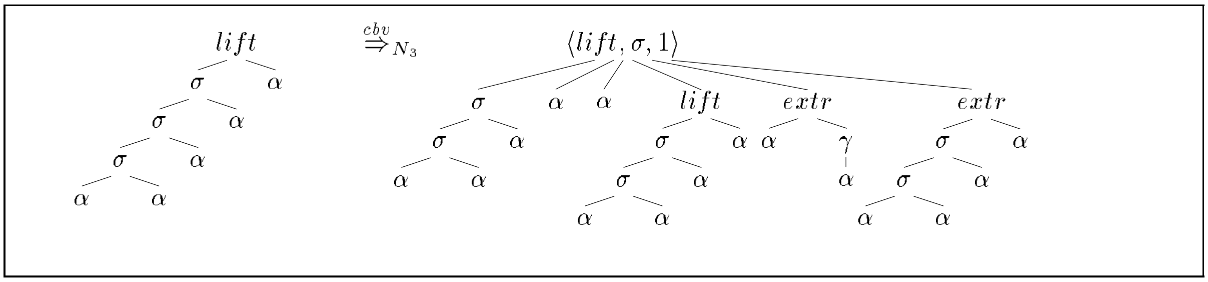
Figure 18: Rules of the macro tree transducer N_2 with register functions.

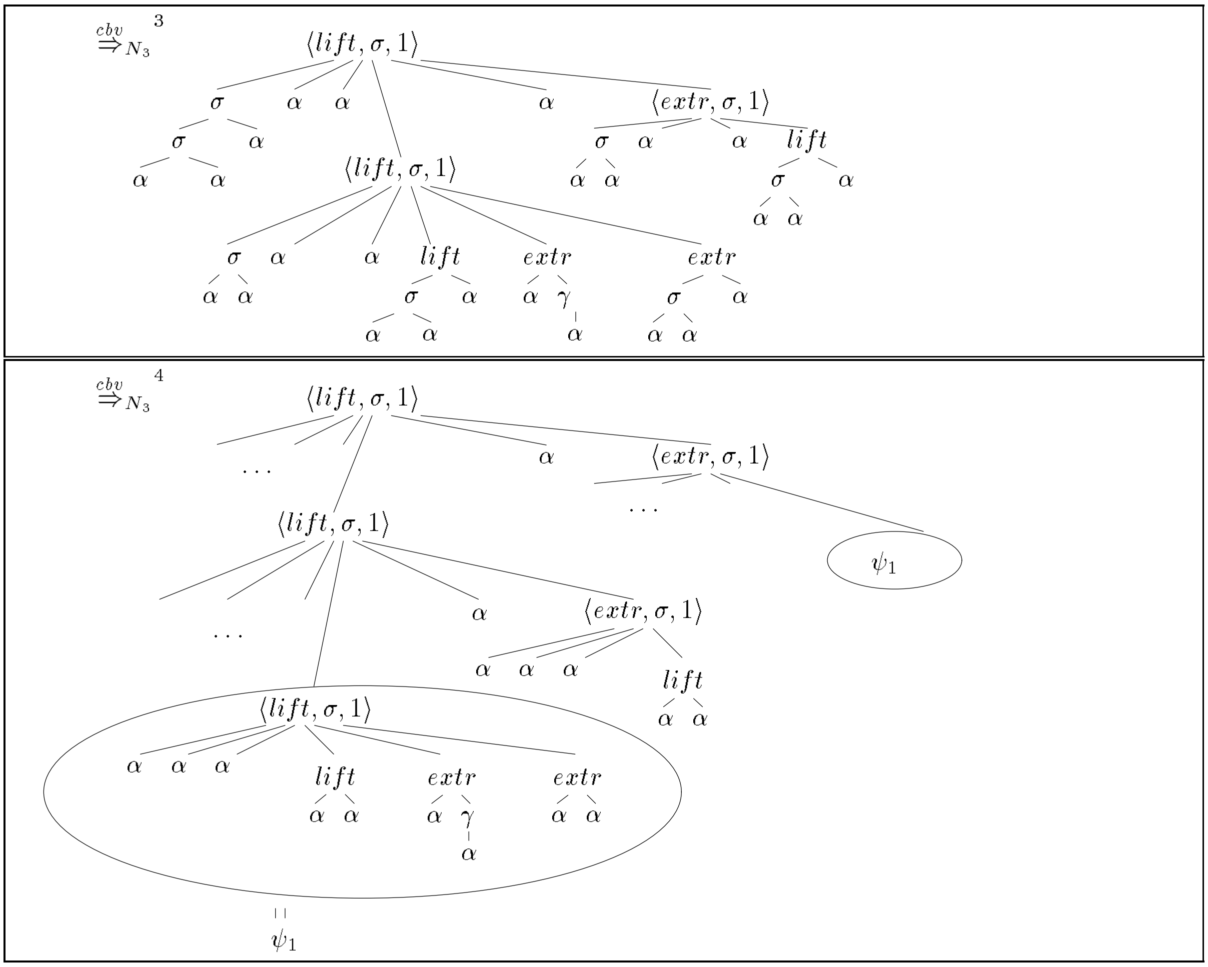
$$
\begin{array}{rcl}
lift(\alpha, y_1) & \to & y_1 \\
lift(\sigma(x_1,x_2), y_1) & \to & \langle lift,\sigma,1\rangle(x_1,x_2,y_1,lift(x_1,y_1),extr(x_2,\gamma(y_1)),extr(x_1,y_1)) \\
\langle lift,\sigma,1\rangle(x_1,x_2,y_1,z_1,z_2,z_3) & \to & \sigma(z_1,\sigma(z_2,z_3)) \\
 & & \\
extr(\alpha, y_1) & \to & \alpha \\
extr(\sigma(x_1,x_2), y_1) & \to & \langle extr,\sigma,1\rangle(x_1,x_2,y_1,lift(x_1,y_1)) \\
\langle extr,\sigma,1\rangle(x_1,x_2,y_1,z_1) & \to & \sigma(lift(x_2,z_1),z_1)
\end{array}
$$

Figure 19: Rules of the macro tree transducer N_3 with register functions.

Note that $double((lift(x_1,y_1),lift(x_1,y_1))) = (2)$ and $delpos((lift(x_1,y_1),lift(x_1,y_1)),2) = (lift(x_1,y_1))$ and the index shift associated with $B = (lift(x_1,y_1),lift(x_1,y_1))$ is the mapping φ_B with $\varphi_B(z_1) = z_1$ and $\varphi_B(z_2) = z_1$.

As before we show the derivation of the function call $lift(\sigma(\sigma(\sigma(\alpha,\alpha),\alpha),\alpha),\alpha)$ by $\stackrel{cbv}{\Rightarrow}_{N_3}$, but we show only the begin of the derivation. The derivation is very similar to the derivation by $\stackrel{cbv}{\Rightarrow}_{N_2}$ with the difference that some evaluations of $lift$-function calls are omitted. The result is computed in 23 steps.





$\square$

Remark 4.14 Let $N, N' \in \mathcal{N}$ with $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma, \Delta, R')$ and $N \vdash_{share,M} N'$.

1. It holds that $tup(FS) = tup(FS')$.

2. There is a unique rule $l \to \zeta$ in R such that N is ready for sharing in $l \to \zeta$ and N' is not ready for sharing in $l \to \zeta'$ which is the corresponding rule in R'. The rule $l \to \zeta$ is called *shared rule* of N and N' and $l \to \zeta'$ is called *result of the shared rule*.

3. For the shared rule $l \to \zeta$ and the result $l \to \zeta'$ of the shared rule, it holds that $F^{gr}_{call}(\zeta') = F^{gr}_{call}(\zeta)$.

4. By definition of the sharing transformation relation, N' differs from N only in the rank of a register function $\langle f, \sigma, i\rangle$ and in the rules where this function occurs. Hence, for every $\overset{cbv}{\Rightarrow}_{N'}$-derivation $\psi_1 \overset{cbv}{\Rightarrow}_{N'} \dots \overset{cbv}{\Rightarrow}_{N'} \psi_m$ with $m > 0$ and $\psi_1, \dots, \psi_m \in sdExp(FS' - \{\langle f, \sigma, i\rangle\}, \Sigma, \Delta)$ there exists an equal $\overset{cbv}{\Rightarrow}_N$-derivation and vice versa. $\square$

Clearly, we want to know if the defined sharing transformation relation is semantic preserving, i.e., if for every $N, N' \in \mathcal{N}$ with $N \vdash_{share,M} N'$ it holds that N' computes the same call-by-value tree function as N. Another important question is, whether N' is more efficient than N or not. The following corollary gives an answer to these questions.

Corollary 4.15 *Let $N, N' \in \mathcal{N}$ and $N \vdash_{share,M} N'$. It holds that N' is more efficient than N. In particular, assuming that $l \to \zeta$ is the shared rule of N and N', then if the evaluation of a syntax-directed expression ψ by $\overset{cbv}{\Rightarrow}_N$ needs L derivation steps and the rule $l \to \zeta$ is applied K times, then the evaluation of ψ by $\overset{cbv}{\Rightarrow}_{N'}$ needs less than $L - K$ derivation steps.*

Proof: Let $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma, \Delta, R')$ such that $N \vdash_{share,M} N'$.

By definition of $\vdash_{share,M}$ there exists a register function $\langle f, \sigma, i\rangle$ in both FS and FS' such that $rank_{FS}(\langle f, \sigma, i\rangle) \neq rank_{FS'}(\langle f, \sigma, i\rangle)$.

Let $k = rank_\Sigma(\sigma)$ and $n + 1 = rank_{FS}(f) = rank_{FS'}(f)$. For the sake of brevity we denote the function symbol $\langle f, \sigma, i\rangle$ of N of rank $k + n + r$ by g and the function symbol $\langle f, \sigma, i\rangle$ of N' of rank $k + n + r'$ by g'.

Let $l_1 \to \zeta_1$ and $l_2 \to \zeta_2$ be the uniquely determined rules in N such that $root(\zeta_1) = g$ and $root(l_2) = g$ and let $l_1 \to \zeta_1'$ and $l_2' \to \zeta_2'$ be the uniquely determined rules in N' such that $root(\zeta_1') = g'$ and $root(l_2) = g'$. (Note that $l_1 \to \zeta_1$ is the shared rule of N and N' and $l_1 \to \zeta_1'$ is the result of the shared rule.) These rules are the only rules which are different in N and N'.

As before (compare Corollary 4.5), to prove that N' is more efficient than N we have to prove the following statements: N' is at least as efficient as N and N' is sometimes more efficient than N (cf. Definition 3.18). In particular, we have to prove that, if for some syntax-directed expression $\psi \in sdExp(gr(FS), \Sigma, \Delta)$ (note that $gr(FS') = gr(FS)$) there are an $a \geq 0$ and an irreducible $\psi' \in sdExp(FS \cap FS', \Sigma, \Delta)$ such that $\psi \overset{cbv}{\Rightarrow}_N^a \psi'$, then there is a $b \leq a$ such that $\psi \overset{cbv}{\Rightarrow}_{N'}^b \psi'$ (at least as efficient). The property "sometimes more efficient" corresponds to the second statement of the corollary: we have to prove that K applications of the rule $l_1 \to \zeta_1$ during the $\overset{cbv}{\Rightarrow}_N$-derivation of a syntax-directed expression ψ lead to a $\overset{cbv}{\Rightarrow}_{N'}$-derivation of ψ which is at least K steps shorter.

Let us consider an arbitrary derivation of a syntax-directed expression ψ by $\overset{cbv}{\Rightarrow}_N$:

$$\psi = \psi_1 \overset{cbv}{\Rightarrow}_N \psi_2 \overset{cbv}{\Rightarrow}_N \ldots \overset{cbv}{\Rightarrow}_N \psi_\nu \overset{cbv}{\Rightarrow}_N \psi_{\nu+1} \overset{cbv}{\Rightarrow}_N \ldots \overset{cbv}{\Rightarrow}_N \psi_{\nu+\mu} \overset{cbv}{\Rightarrow}_N^* \psi_m = \psi'.$$

Let $\nu \in [m]$ such that for every $i < \nu$, in the step $\psi_i \overset{cbv}{\Rightarrow}_N \psi_{i+1}$ the shared rule $l_1 \to \zeta_1$ is *not* applied and $\psi_\nu \overset{cbv}{\Rightarrow}_N \psi_{\nu+1}$ where $\psi_{\nu+1} = \psi_\nu[w \leftarrow \varphi(\zeta_1)]$ and $sub(\psi_\nu, w) = \varphi(l_1)$ for some path w and matching substitution φ. Furthermore, let $\psi_{\nu+\mu} = \psi_\nu[w \leftarrow \xi_w]$ and ξ_w is the normalform of $sub(\psi_\nu, w)$.

Since all other rules apart from the shared rule and the result of the shared rule are equal in N and N', it holds that there is an equal $\overset{cbv}{\Rightarrow}_{N'}$-derivation until ψ_ν is reached:

$$\psi = \psi_1 \overset{cbv}{\Rightarrow}_{N'} \psi_2 \overset{cbv}{\Rightarrow}_{N'} \psi_3 \ldots \overset{cbv}{\Rightarrow}_{N'} \psi_\nu$$

Note that by assumption ψ does not contain register function calls, but the syntax-directed expressions ψ_i with $i > 1$ may contain register function calls, because the applied rules may introduce register function calls.

By Definition 4.12 the shared rule can be of one of the following form: either the root of the left-hand side l_1 is a ground function or a register function. The right-hand side ζ_1 has to be a register function call.

If we can prove that there is a $\overset{cbv}{\Rightarrow}_{N'}$-derivation of ψ_ν to $\psi_{\nu+\mu}$ which is at least about the number of applications of the shared rule shorter as the corresponding $\overset{cbv}{\Rightarrow}_N$-derivation, then the statements are proven (for the rest of the derivation, the same argumentation as above holds).

Since $\psi_{\nu+\mu} = \psi_\nu[w \leftarrow \xi_w]$, it suffices to prove the following two statements for $\zeta_1 = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \theta_1, \ldots, \theta_r)$ where $\tilde{x}$ and $\tilde{y}$ abbreviate the sequences $x_1, \ldots, x_k$ and $y_1, \ldots, y_n$.

1. Let $s = \sigma(s_1, \ldots, s_k) \in T\langle\Sigma\rangle$ and $t_1, \ldots, t_n \in T\langle\Delta\rangle$. If the left-hand side l_1 of the shared rule has the form $f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n)$ and there is an $a \geq 1$ and a $\xi \in sdExp(FS, \Sigma, \Delta)$ such that
$$f(s, t_1, \ldots, t_n) \overset{cbv}{\Rightarrow}_N{}^{a}\, \xi$$
and ξ is irreducible, then there is a $b \geq 1$ and a derivation
$$f(s, t_1, \ldots, t_n) \overset{cbv}{\Rightarrow}_{N'}{}^{b}\, \xi$$
and $b < a$. If the shared rule is applied K times in the derivation by $\overset{cbv}{\Rightarrow}_N$, then $b \leq a - K$.

2. Let $s_1, \ldots, s_k \in T\langle\Sigma\rangle$, $t_1, \ldots, t_n \in T\langle\Delta\rangle$, and $\psi_1, \ldots, \psi_r \in sdExp(FS, \Sigma, \Delta)$. If the left-hand side l_1 of the shared rule has the form $\langle f, \sigma, i-1\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_{\langle f,\sigma,i-1\rangle}})$ and there is an $a \geq 1$ and a $\xi \in sdExp(FS, \Sigma, \Delta)$ such that
$$\langle f, \sigma, i-1\rangle(\tilde{s}, \tilde{t}, \psi_1, \ldots, \psi_{r_{\langle f,\sigma,i-1\rangle}}) \overset{cbv}{\Rightarrow}_N{}^{a}\, \xi$$
and ξ is irreducible, then there is a $b \geq 1$ and a derivation
$$\langle f, \sigma, i-1\rangle(\tilde{s}, \tilde{t}, \psi_1, \ldots, \psi_{r_{\langle f,\sigma,i-1\rangle}}) \overset{cbv}{\Rightarrow}_{N'}{}^{b}\, \xi$$
and $b < a$. If the shared rule is applied K times in the derivation by $\overset{cbv}{\Rightarrow}_N$, then $b \leq a - K$.

In fact, these are the only two possible forms of the predecessor of ζ_1 with respect to $\overset{cbv}{\Rightarrow}_N$.

Proof.

1. Let $l_1 = f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n)$. Note that g is in this case $\langle f, \sigma, 1\rangle$. We prove this statement by induction on the height of s. We have to start the proof with $height(s) = 1$, because in the case $height(s) = 0$, the right-hand side ζ_1 does not contain any ground function call and hence, the assumptions above cannot hold.

   (i) $height(s) = 1$, i.e., $s = \sigma(\alpha_1, \ldots, \alpha_k)$ with $\alpha_i \in \Sigma^{(0)}$. The $\overset{cbv}{\Rightarrow}_N$-derivation of $f(s, t_1, \ldots, t_n)$ has the following form:

$$\begin{array}{ll}
 & f(s, t_1, \ldots, t_n) \\
\overset{cbv}{\Rightarrow}_N & \varphi_1(\zeta_1) = \langle f, \sigma, 1\rangle(\tilde{\alpha}, \tilde{t}, \psi_1, \ldots, \psi_r) \\
 & \text{where } \varphi_1 = [x_\nu/\alpha_\nu; \nu \in [k]][y_\mu/t_\mu; \mu \in [n]] \\
\overset{cbv}{\Rightarrow}_N{}^{a_1} & \langle f, \sigma, 1\rangle(\tilde{\alpha}, \tilde{t}, \xi_1, \ldots, \xi_r) \\
 & \text{for some } a_1 \geq 1 \text{ and, for every } j \in [r],\ \xi_j = nf(\overset{cbv}{\Rightarrow}_N, \psi_j) \\
\overset{cbv}{\Rightarrow}_N & \varphi_2(\zeta_2) \\
 & \text{where } \varphi_2 = [x_\nu/\alpha_\nu; \nu \in [k]][y_\mu/t_\mu; \mu \in [n]][z_\nu/\xi_\nu; \nu \in [r]] \\
 & \text{Note that this step is possible because } l_2 \text{ is flat.} \\
\overset{cbv}{\Rightarrow}_N{}^{a_2} & \xi \\
 & \text{for some } a_2 \geq 1.
\end{array}$$

Define $a = a_1 + a_2 + 2$. Since the rules for nullary constructors have not been changed by $\vdash_{share,M}$ (for the reason that no ground function calls occur in their right-hand sides) and, for every $j \in [r]$, ψ_j is by definition a tree over ground function calls with recursion argument of height 0 and constructors, it holds that

$$nf(\overset{cbv}{\Rightarrow}_N, \psi_j) = nf(\overset{cbv}{\Rightarrow}_{N'}, \psi_j)$$

and the lengths of the derivations are equal. The shared rule was applied exactly once. Hence,

$$
\begin{array}{ll}
 & f(s, t_1, \ldots, t_n) \\
\overset{cbv}{\Rightarrow}_{N'} & \varphi_1(\zeta_1') = \langle f, \sigma, 1\rangle(\tilde{\alpha}, \tilde{t}, \psi_1', \ldots, \psi_{r'}') \\
 & \text{where by definition of } \vdash_{share,M} \text{ it holds that} \\
 & (\psi_1', \ldots, \psi_{r'}') = delpos((\psi_1, \ldots, \psi_r), double((\theta_1, \ldots, \theta_r))) \\
 & \text{and, by definition of } delpos \text{ and } double \text{ and} \\
 & \text{by the fact that } \psi_j \text{ is an instance of } \theta_j, \\
 & \text{for every } j' \in [r'], \text{ there is a } j \in [r] \text{ such that } \psi_{j'}' = \psi_j. \\
\overset{cbv}{\Rightarrow}_{N'}^{b_1} & \langle f, \sigma, 1\rangle(\tilde{\alpha}, \tilde{t}, \xi_1', \ldots, \xi_{r'}') \\
 & \text{for some } b_1 \geq 1 \text{ and, for every } j \in [r'], \xi_j' = nf(\overset{cbv}{\Rightarrow}_{N'}, \psi_j'). \\
 & \text{With the remark above, it holds that for every} \\
 & j' \in [r'], \text{ there is a } j \in [r] \text{ such that } \xi_j' = \xi_j \\
 & \text{and it holds that } b_1 < a_1 \text{ because of } r' < r. \\
\overset{cbv}{\Rightarrow}_{N'} & \varphi'(\varphi_{(\theta_1,\ldots,\theta_r)}(\zeta_2)) \\
 & \text{where} \\
 & \varphi' = [x_\nu/\alpha_\nu; \nu \in [k]][y_\mu/t_\mu; \mu \in [n]][z_\nu/\xi_\nu'; \nu \in [r']] \\
 & \text{and } \varphi_{(\theta_1,\ldots,\theta_r)} \text{ is the index shift}
\end{array}
$$

We prove that

$$(*) \qquad \varphi'(\varphi_{(\theta_1,\ldots,\theta_r)}(\zeta_2)) = \varphi_2(\zeta_2).$$

The domain of the index shift is the set Z, the other variables remain unchanged, i.e., it suffices to prove that

$$(\varphi_{(\theta_1,\ldots,\theta_r)}(\zeta_2))[z_i/\xi_i'; i \in [r']] = \zeta_2[z_i/\xi_i; i \in [r]].$$

Let us consider a variable z_i with $i \in [r]$ which occurs in the left-hand side and in the right-hand side of the equation (*). By definition of the index shift this z_i is replaced by z_j with $j \in [r']$, if $\theta_i = \theta_j'$. Note that if $\theta_i = \theta_j'$ then $\xi_i = \xi_j'$. By the second substitution in the left-hand side of the equation (*) this z_j is replaced by ξ_j' whereas the variable z_i in the right-hand side of the equation is replaced by ξ_i. Since ξ_i and ξ_j' are equal, the equation holds. Therewith (*) is proved. During further $\overset{cbv}{\Rightarrow}_{N'}$-derivation steps of $\varphi_2(\zeta_2)$ there have only function symbols to be applied of which the rules were not changed. Hence,

$$\varphi_2(\zeta_2) \overset{cbv}{\Rightarrow}_{N'}^{b_2} \xi$$

and $b_2 = a_2$. Define $b = b_1 + b_2 + 2$. It holds that $b < a$, i.e., $b \leq a - 1$.

(ii) Let $height(s) = \rho$, i.e., $s = \sigma(s_1, \ldots, s_k)$. We argue in the same way as in (i) except that now for every $\overset{cbv}{\Rightarrow}_N$-derivation of function calls ψ_j there exists a $\overset{cbv}{\Rightarrow}_{N'}$-derivation of ψ_j which is either of equal length or shorter (if the shared rule is applied) because of the induction hypothesis and because of the fact that other rules have not been changed.

2. This part of the proof is similar to (a).

Now we have proved that a derivation in which the changed rules occur, is shorter w.r.t. $\overset{cbv}{\Rightarrow}_N$ than the one w.r.t. $\overset{cbv}{\Rightarrow}_{N'}$. Since no other rules are changed, it follows immediately that N' is more efficient than N. $\square$

In Example 4.13 a part of the derivation of the function call $t = lift(\sigma(\sigma(\sigma(\alpha, \alpha), \alpha), \alpha), \alpha)$ by $\overset{cbv}{\Rightarrow}_{N_3}$ is given. The derivation of t by $\overset{cbv}{\Rightarrow}_{N_2}$ is given in Example 4.3; its length is 29. Note that $N_2 \vdash_{share,M_1} N_3$ and the shared rule of N_2 and N_3 is the rule with left-hand side $extr(\sigma(x_1, x_2), y_1)$. During the $\overset{cbv}{\Rightarrow}_{N_2}$-derivation of t the shared rule is applied twice. Hence, in accordance with our statements the result has to be computed in at most 27 steps by $\overset{cbv}{\Rightarrow}_{N_3}$. In fact, it is computed in 23 steps.

Lemma 4.16 *Let N and N' be two macro tree transducers with register functions. If $N \vdash_{share,M}$ N', then*

(a) $\tau_{cbv}(N) = \tau_{cbv}(N')$.

(b) N' and M are semantically equivalent.

<u>Proof.</u>
(a) Since $\tau_{cbv}(N)$ and $\tau_{cbv}(N')$ are total functions, it suffices to prove that $\tau_{cbv}(N) \subseteq \tau_{cbv}(N')$. This follows again from the notion of more efficient and from Corollary 4.15.

(b) Let $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma, \Delta, R')$. M and N are semantically equivalent (cf. Definition 4.12). Since $tup(FS) = tup(FS')$ (cf. Remark 4.14 1.) and by Statement (a) it holds that M and N' are semantically equivalent. $\square$

Lemma 4.17 *The relation $\vdash_{share,M}$ is locally confluent.*

<u>Proof.</u> Let $N = (FS, \Sigma, \Delta, R)$, $N' = (FS', \Sigma, \Delta, R')$, and $N'' = (FS'', \Sigma, \Delta, R'')$ be macro tree transducers with register functions such that $N \vdash_{share,M} N'$ and $N \vdash_{share,M} N''$ and $N' \neq N''$.

Then by Definition 4.12 it follows that there are two different, uniquely determined rules $l_1 \to \zeta_1$ and $l_2 \to \zeta_2$ in R for which the following three conditions hold:

1. N is ready for sharing in $l_1 \to \zeta_1$ and in $l_2 \to \zeta_2$.
2. N' is not ready for sharing in the rule in R' with left-hand side l_1. (Remark 4.14).
3. N'' is not ready for sharing in the rule in R'' with left-hand side l_2. (Remark 4.14).

Note that the relation $\vdash_{share,M}$ only changes two rules: the rule $l \to \zeta$ in which N is ready for sharing and the rule of which the left-hand sides root is labeled by the register function $root(\zeta)$. Also note the the manner in which these rules are changed, is determined by the form of ζ.

Hence, in our case it holds that if these four rules, i.e., the rules $l_1 \to \zeta_1$, $l_2 \to \zeta_2$ and the rules the left-hand side's roots of which are labeled by $root(\zeta_1)$ and $root(\zeta_2)$, are pairwise different, then the following holds: $l_2 \to \zeta_2$ is a rule in R' and N' is ready for sharing in $l_2 \to \zeta_2$. $l_1 \to \zeta_1$ is a rule in R'' and N'' is ready for sharing in $l_1 \to \zeta_1$. Thus, it is easy to construct $\tilde{N}$ with $N' \vdash_{share,M} \tilde{N}$ and $N'' \vdash_{share,M} \tilde{N}$.

More interesting are the cases in which the rules are not pairwise different. By the assumption above ($N' \neq N''$) it holds that $l_1 \to \zeta_1$ and $l_2 \to \zeta_2$ are different. This implies that also the two rules with left-hand side's roots $root(\zeta_1)$ and $root(\zeta_2)$ are different.

Hence, it remains to consider the following cases: $root(l_2) = root(\zeta_1)$ or $root(l_1) = root(\zeta_2)$.

Let us consider the case $root(l_2) = root(\zeta_1)$ (the other case can be proved analogously). Then, ζ_1 is of the form

$$\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \theta_1, \ldots, \theta_{r_i})$$

for some $\langle f, \sigma, i\rangle \in reg(FS)$ of rank $n+k+r_i$ where n and k are the ranks of f and σ, respectively. By definition of macro tree transducers with register functions and by the fact that N is ready for sharing in $l_2 \to \zeta_2$, $l_2 \to \zeta_2$ is of the form

$$\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_i}) \to \langle f, \sigma, i+1\rangle(\tilde{x}, \tilde{y}, \delta_1, \ldots, \delta_{r_{i+1}})$$

and there is a rule $l_3 \to \zeta_3$ of the form

$$\langle f, \sigma, i+1\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_{i+1}}) \to \zeta_3$$

We can concentrate our attention to these three rules.

Let us consider the application of $\vdash_{share,M}$ to N': By definition of the sharing transformation relation, instead of the rules $l_1 \to \zeta_1$ and $l_2 \to \zeta_2$, the rules $l_1 \to \zeta_1'$ with $\zeta_1' = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \theta_1', \ldots, \theta_{r_i'}')$ and $\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_i'}) \to \varphi_{(\theta_1,\ldots,\theta_{r_i})}(\zeta_2)$ (denoted as $l_2' \to \zeta_2'$) are in N'. The list $(\theta_1', \ldots, \theta_{r_i'}')$ is the result of $delpos((\theta_1, \ldots, \theta_{r_i}), double((\theta_1, \ldots, \theta_{r_i})))$. Since the index shift $\varphi_{(\theta_1,\ldots,\theta_{r_i})}$ (in the following abbreviated by φ_1) maps variables in Z_{r_i} to variables in $Z_{r_i'}$ with $r_i' < r_i$, it holds that no equal ground function call can disappear, but additional equal ground function calls can arise (if, e.g., $f(x_1, z_1)$ and $f(x_1, z_2)$ are two function calls in the argument list of ζ_2 and the index shift maps z_2 to z_1). Note that from the fact that N is ready for sharing in $l_2 \to \zeta_2$ it follows that there are equal ground function calls in ζ_2. With the remark above it follows that N' is still ready for sharing in $l_2' \to \zeta_2'$.

Let $N' \vdash_{share,M} \tilde{N}$ such that $l_2' \to \zeta_2'$ is the shared rule of N' and $\tilde{N}$. Then the set of rules of $\tilde{N}$ contains instead of the rules $l_2' \to \zeta_2'$ and $l_3 \to \zeta_3$, the rules $\tilde{l_2'} \to \tilde{\zeta_2}$ and $\tilde{l_3} \to \tilde{\zeta_3}$. Note that comparing $\tilde{N}$ with N, additionally to the above named rules, $\tilde{N}$ contains the rule $l_1 \to \zeta_1'$ instead of the rule $l_1 \to \zeta_1$. The form of the considered rules in N and their changes in $\tilde{N}$ are listed below:

- $l_1 \to \zeta_1$ with $\zeta_1 = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \theta_1, \ldots, \theta_{r_i})$ is in N and
  $l_1 \to \zeta_1'$ with $\zeta_1' = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \theta_1', \ldots, \theta_{r_i'}')$ and
  $(\theta_1', \ldots, \theta_{r_i'}') = delpos((\theta_1, \ldots, \theta_{r_i}), double((\theta_1, \ldots, \theta_{r_i})))$ is in $\tilde{N}$.

- $\langle f,\sigma,i\rangle(\tilde{x},\tilde{y},z_1,\ldots,z_{r_i}) \to \langle f,\sigma,i+1\rangle(\tilde{x},\tilde{y},\delta_1,\ldots,\delta_{r_{i+1}})$ is in N and
$\langle f,\sigma,i\rangle(\tilde{x},\tilde{y},z_1,\ldots,z_{r'_i}) \to \langle f,\sigma,i+1\rangle(\tilde{x},\tilde{y},\widetilde{\delta}_1,\ldots,\widetilde{\delta}_{\widetilde{r_{i+1}}})$ with
$(\widetilde{\delta}_1,\ldots,\widetilde{\delta}_{\widetilde{r_{i+1}}}) = \mathit{delpos}((\varphi_1(\delta_1),\ldots\varphi_1(\delta_{r_{i+1}})),\mathit{double}((\varphi_1(\delta_1),\ldots,\varphi_1(\delta_{r_{i+1}})))$ is in $\tilde{N}$.
- $\langle f,\sigma,i+1\rangle(\tilde{x},\tilde{y},z_1,\ldots,z_{r_{i+1}}) \to \zeta_3$ is in N and
$\langle f,\sigma,i+1\rangle(\tilde{x},\tilde{y},z_1,\ldots,z_{\tilde{r}_{i+1}}) \to \varphi_2(\zeta_3)$ where φ_2 denotes the index shift $\varphi_{(\varphi_1(\delta_1),\ldots,\varphi_1(\delta_{r_{i+1}}))}$ is in $\tilde{N}$.

We have to show that $\tilde{N}$ can also be computed by $\vdash_{share,M}$-steps applied to N''.

Now, we consider N'': Instead of the rules $l_2 \to \zeta_2$ and $l_3 \to \zeta_3$, the rules $l_2 \to \zeta''_2$ with $\zeta''_2 = \langle f,\sigma,i\rangle(\tilde{x},\tilde{y},\delta'_1,\ldots,\delta'_{r'_{i+1}})$ and $l''_3 \to \zeta''_3$ are in R''. The list $(\delta'_1,\ldots,\delta'_{r'_{i+1}})$ is the result of $\mathit{delpos}((\delta_1,\ldots,\delta_{r_{i+1}}),\mathit{double}((\delta_1,\ldots,\delta_{r_{i+1}})))$. The rule $l_1 \to \zeta_1$ is still in R''. Hence, N'' is ready for sharing in this rule. Then, $N'' \vdash_{share,M} N'''$ and N''' contains instead of the rules $l_1 \to \zeta_1$ and $l_2 \to \zeta''_2$ the rules $l_1 \to \zeta'_1$ and $l'_2 \to \varphi_1(\zeta''_2)$. Let us again list, how the three considered rules looks like in N''':

- $l_1 \to \zeta'_1$ is in N''' and is equal to the rule in $\tilde{N}$
- $l'_2 \to \varphi_1(\langle f,\sigma,i+1\rangle(\tilde{x},\tilde{y},\delta'_1,\ldots,\delta'_{r'_{i+1}}))$ with
$(\delta'_1,\ldots,\delta'_{r'_{i+1}}) = \mathit{delpos}((\delta_1,\ldots,\delta_{r_{i+1}}),\mathit{double}((\delta_1,\ldots,\delta_{r_{i+1}})))$ is in N'''. This rule can also be written as $l'_2 \to \langle f,\sigma,i+1\rangle(\tilde{x},\tilde{y},\bar{\delta}_1,\ldots,\bar{\delta}_{r'_{i+1}}))$ with
$(\bar{\delta}_1,\ldots,\bar{\delta}_{r'_{i+1}}) = \mathit{delpos}((\varphi_1(\delta_1),\ldots,\varphi_1(\delta_{r_{i+1}})),\mathit{double}((\delta_1,\ldots,\delta_{r_{i+1}})))$
- $l''_3 \to \varphi_3(\zeta_3)$ is in N''' where φ_3 is the index shift $\varphi_{(\delta_1,\ldots,\delta_{r_{i+1}})}$.

Two cases are possible:
(a) $\mathit{double}((\delta_1,\ldots,\delta_{r_{i+1}})) = \mathit{double}((\varphi_1(\delta_1),\ldots,\varphi_1(\delta_{r_{i+1}})))$, i.e., no new additional ground function calls arise by applying φ_1. Then, the rules arised from $l_2 \to \zeta_2$ in $\tilde{N}$ and N''' are equal. Consequently, the same holds for the rules arised from $l_3 \to \zeta_3$. Therewith, $N' \vdash_{share,M} \tilde{N}$ and $N'' \vdash_{share,M} \tilde{N}$.

(b) $\mathit{double}((\delta_1,\ldots,\delta_{r_{i+1}})) \neq \mathit{double}((\varphi_1(\delta_1),\ldots,\varphi_1(\delta_{r_{i+1}})))$, i.e., new additional ground function calls arise by applying φ_1. Then N''' is ready for sharing in $l'_2 \to \varphi_1(\zeta''_2)$, i.e., $N''' \vdash_{share,M} \hat{N}$. As before, we consider the three rules in $\hat{N}$.

- The rule $l_1 \to \zeta'_1$ is in $\hat{N}$ and in $\tilde{N}$.
- $l'_2 \to \langle f,\sigma,i+1\rangle(\tilde{x},\tilde{y},\hat{\delta}_1,\ldots,\hat{\delta}_{\hat{r}_{i+1}})$ with
$(\hat{\delta}_1,\ldots,\hat{\delta}_{\hat{r}_{i+1}}) = \mathit{delpos}((\bar{\delta}_1,\ldots,\bar{\delta}_{r'_{i+1}})),\mathit{double}((\bar{\delta}_1,\ldots,\bar{\delta}_{r'_{i+1}})))$. By definition of *delpos* and *double* and by the fact that
$$(\bar{\delta}_1,\ldots,\bar{\delta}_{r'_{i+1}}) = \mathit{delpos}((\varphi_1(\delta_1),\ldots,\varphi_1(\delta_{r_{i+1}})),\mathit{double}((\delta_1,\ldots,\delta_{r_{i+1}})))$$
it follows immediately that $(\hat{\delta}_1,\ldots,\hat{\delta}_{\hat{r}_{i+1}}) = (\widetilde{\delta}_1,\ldots,\widetilde{\delta}_{\widetilde{r_{i+1}}})$. Hence $\hat{N} = \tilde{N}$.

We have proved that if $N \vdash_{share,M} N'$ and $N \vdash_{share,M} N''$, then there exists a $\tilde{N}$ such that $N' \vdash_{share,M} \tilde{N}$ and $N'' \vdash^*_{share,M} \tilde{N}$, i.e., $\vdash_{share,M}$ is locally confluent. □

Lemma 4.18 *The relation $\vdash_{share,M}$ is noetherian.*

Proof. We prove the following statement by induction on K.

> For every macro tree transducer N with register functions there exists a number $K \geq 0$ such that the length of the derivations by $\vdash_{share,M}$ starting from N is exactly equal to K.

Let $N = (FS, \Sigma, \Delta, R)$ be a macro tree transducer with register functions and let $K \geq 0$ denote the number of rules in which N is ready for sharing. We prove that this K fulfills the conditions by induction on K.

(i) $K = 0$. Then $N {\vdash_{share,M}}^0 N' = N$.

(ii) $K > 0$. Let the statement hold for $K - 1$.
By assumption there are rules $l_1 \to \zeta_1, \ldots, l_K \to \zeta_K$ in which N is ready for sharing. By Definition 4.11, for every $j \in [K]$, ζ_j is of the form $g_j(\tilde{x}, \tilde{y}, \psi_{j,1}, \ldots, \psi_{j,r_j})$ with $g_j \in reg(FS)$. Let, for every $j \in [K]$, $g_j(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_j}) \to \theta_j$ be the g_j-rules. (Note that by Definition 4.11 the left-hand side of the g_j-rule has exactly this form). Hence, $N \vdash_{share,M} N_1$ and $N_1 = (FS_1, \Sigma, \Delta, R_1)$ where $FS_1 = (FS - \{g_1^{(k+n+r_1)}\}) \cup \{g_1^{(k+n+r'_1)}\}$ and $R_1 = (R - \{l_1 \to \zeta_1, g_1(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_1})\}) \cup \{l_1 \to \zeta'_1, g_1(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r'_1}) \to \theta'_1\}$. Now, N_1 is not ready for sharing in rule $l_1 \to \zeta'_1$, because ζ'_1 does not contain any equal function calls by definition of *delpos* and *double*. Since the index shift only renames variables in Z, even if the rule $g_1(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_1}) \to \theta_1$ is equal to a rule in $\{l_2 \to \zeta_2, \ldots, l_K \to \zeta_K\}$, N_1 is still ready for sharing in the rules $l'_2 \to \zeta'_2, \ldots, l'_K \to \zeta'_K$ where, for every $i \in [2, K]$, $(l'_i \to \zeta'_i) = (l_i \to \zeta_i)$, if $l_i \neq g_1(\tilde{x}, \tilde{y}, z_1, \ldots, z_{r_1})$, and otherwise the rule has to be adapted. Hence, by induction hypothesis $N_1 {\vdash_{share,M}}^{K-1} N'$. □

Lemma 4.19 *The relation $\vdash_{share,M}$ is confluent and noetherian.*

Proof. This result follows directly by the fact that $\vdash_{share,M}$ is locally confluent and noetherian (cf. previous lemmas) and by Lemma 2.4 of [Hue80]. □

Hence, for every macro tree transducer N with register functions there is a unique macro tree transducer $nf(\vdash_{share,M}, N)$ with register functions which is irreducible with respect to $\vdash_{share,M}$. Note that N_3 (cf. Example 4.13) is the normalform of N_2 with respect to $\vdash_{share,M_1}$.

Finally, let us describe a connection between the splitting transformation relation and the sharing transformation relation.

Lemma 4.20 Let M be a macro tree transducer. It holds that

$$\vdash_{split,M} \circ \vdash_{share,M}^{\infty} \subseteq \vdash_{share,M}^{\infty} \circ \vdash_{split,M} \circ \vdash_{share,M}^{\infty} .$$

Proof. The following statement has to be proved: Let N and N' be two macro tree transducers with register functions. If $N \vdash_{split,M} \circ \vdash_{share,M}^{\infty} N'$, then $N \vdash_{share,M}^{\infty} \circ \vdash_{split,M} \circ \vdash_{share,M}^{\infty} N'$.

Let $N \vdash_{split,M} \circ \vdash_{share,M}^{\infty} N'$, i.e., there is a macro tree transducer N_1 with register functions such that $N \vdash_{split,M} N_1 \vdash_{share,M}^{\infty} N'$. We distinguish the following two cases:

1. If N is not ready for sharing, then $nf(\vdash_{share,M}, N) = N$ and the statement holds trivially.

2. If N is ready for sharing in the rules $l_1 \rightarrow \zeta_1, \ldots, l_m \rightarrow \zeta_m$, then N_1 is also ready for sharing in these rules, because the splitting transformation relation may only enable a further sharing, but it has no influence to rules of which the root of the right-hand side is labeled by a register function (note that this property holds for the rules $l_1 \rightarrow \zeta_1, \ldots, l_m \rightarrow \zeta_m$). Let N_2 be a macro tree transducer with register functions such that $N \vdash^{\infty}_{share,M} N_2$. By assumption ($\vdash_{split,M}$ is applicable to N) there is a rule $l \rightarrow \zeta$ in the set R of rules of N such that $Cut(\zeta) \neq \emptyset$. Note that ζ is a ground right-hand side (cf. definition of the splitting transformation relation). The sharing transformation steps applied to N in the derivation $N \vdash^{\infty}_{share,M} N_2$ may only rename variables of ground right-hand sides. Thus, the rule $l' \rightarrow \zeta'$ where $l = l'$, if $root(l)$ is a ground function, or $root(l) = root(l')$, if $root(l)$ is a register function, $\zeta' = \varphi(\zeta)$, and φ is a renaming of variables in Z, is in the set of rules of N_2 and $Cut(\zeta') \neq \emptyset$. Hence, there is a macro tree transducer N_3 with register functions such that $N_2 \vdash_{split,M} N_3$. By similar comparisons as in the proofs of the local confluence of the two transformation relations it holds that $nf(\vdash_{share,M}, N_3) = N'$. $\square$

4.3 Tupling

In the previous section we have introduced a transformation relation which allows to avoid the recomputation of values caused by equal function calls in a right-hand side. But also function calls with different ground functions at their root, but the same list of arguments can cause the effect that during a computation equal function calls arise which have to be evaluated. Note that by such function calls with the same list of arguments also multiple traversals of common inputs take place.

Consider, e.g., the part of the calling graph of $lift(\sigma(\sigma(\alpha,\alpha),\alpha),\alpha)$ on the basis of the rules of N_3 of Example 4.13 which is shown in Figure 20.

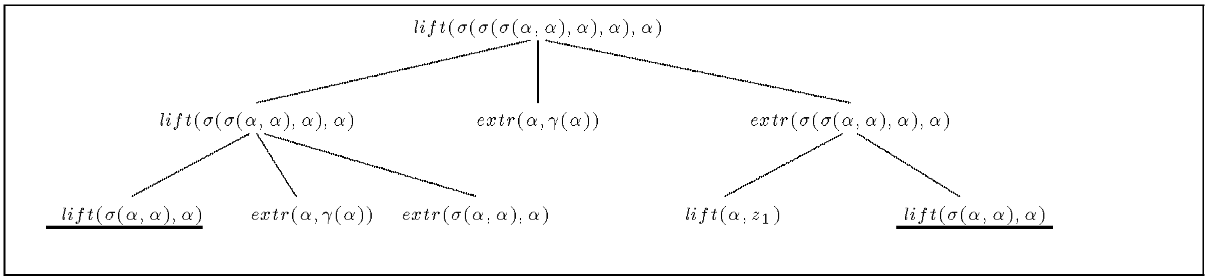


Figure 20: Part of a calling graph of ground functions.

We show a method to tuple ground function calls with different ground functions at their root, but the same list of arguments, as, e.g., $extr(t_1, t_2)$ and $lift(t_1, t_2)$. Before starting with the formal details, let us discuss this tupling informally.

The idea of the tupling is to create a new function which is a tuple of these functions with the same list of arguments, such that multiple traversals of common inputs are avoided. This idea on its own is not new, it is proposed in many other papers (cf., e.g., [PP93, CH95]). What is new is the algorithmical way in which this is done. For the created tuple function, new rules have to be defined as follows: for every input symbol σ, the right-hand sides of the rules of the simple functions from which the tuple function is build, are combined by a *comb*-symbol. For the sake of simplicity, we combine the rules of the original macro tree transducer. Taking the rules of a macro tree transducer with register functions where for every simple function register functions occurs, we would have to take care of a lot of technical details by merging the right-hand side of the register rules in an appropriate way.

With the described procedure the calling graph of Figure 20 is changed to the calling graph in Figure 21.

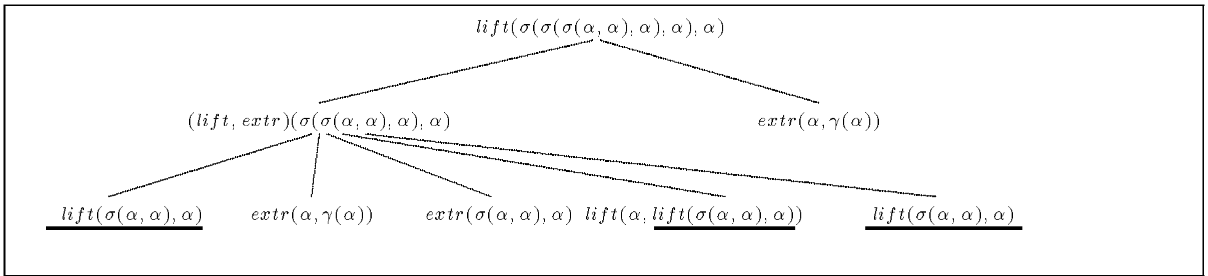


Figure 21: Part of a calling graph of ground functions.

Examining the tupling more carefully, we observe that the following three steps have to be executed:

1. We consider a right-hand side of a rule of a macro tree transducer with register functions and determine if there are candidates which can be tupled. Note that this requires that the right-hand side's root is labeled by a register function and the candidates occur directly under this function (with the same argumentation as in Section 4.2). These candidates are replaced by a new function which is the tuple of the simple functions.

2. By step 1.), the rank of the register function at the root shrinks. Hence, the rule for the register function has to be adapted in its left-hand side and in its right-hand side.

3. New rules have to be created for the new function symbol.

Before defining the tupling transformation relation, we define under which conditions a right-hand side of a macro tree transducer with register functions is called *ready for tupling.*

Definition 4.21 Let $N = (FS, \Sigma, \Delta, R)$ be a macro tree transducer with register functions and let $l \to \zeta$ be a rule in R with $root(\zeta) \in reg(FS)$. Let ψ be in $arglist(\zeta)$ and $\psi \in F^{gr}_{call}(\zeta)$. Define $F(\zeta, \psi) = \{\psi' \mid \psi'$ is in $arglist(\zeta)$, $\psi' \in F^{gr}_{call}(\zeta)$, $\psi' \neq \psi$, and $arglist(\psi) = arglist(\psi')\}$. If $F(\zeta, \psi) \neq \emptyset$, then we say that ζ *is ready for tupling the set* $F(\zeta, \psi) \cup \{\psi\}$ of ground function calls. □

Note that only right-hand sides can be ready for tupling S, of which the root is labeled by a register function. Also note that, if ζ is ready for tupling S, then the cardinality of S is at least 2. Finally note that ζ can be ready for tupling several different sets of ground function calls. Let, e.g., $f(\sigma(x_1), y_1) \to \zeta$ with $\zeta = \langle f, \sigma, 1\rangle(x_1, y_1, f(x_1, y_1), g(x_1, y_1), g(x_1, \gamma(y_1)), f(x_1, \gamma(y_1)))$ be a rule of a macro tree transducer with register functions. Then ζ is ready for tupling $\{f(x_1, y_1), g(x_1, y_1)\}$ and ζ is ready for tupling $\{f(x_1, \gamma(y_1)), g(x_1, \gamma(y_1))\}$.

Definition 4.22 Let M be a macro tree transducer. The *tupling transformation relation* with respect to M is the binary relation $\vdash_{tuple,M} \subseteq \mathcal{N} \times \mathcal{N}$ defined as follows. Let $N = (FS, \Sigma, \Delta, R)$ and N' be two macro tree transducers with register functions. Then $N \vdash_{tuple,M} N'$ iff the following conditions hold:

- N and M are semantically equivalent.
- There is a rule $l_1 \to \zeta_1$ in R such that ζ_1 is ready for tupling the set S of ground function calls and N is not ready for sharing in $l_1 \to \zeta_1$.
- $N' = (FS', \Sigma, \Delta', R')$ where the components FS', Δ', and R' are constructed by the algorithm TUPLE which is shown in the following box. The algorithm receives as input N, M, $l_1 \to \zeta_1$, and S. □

Algorithm TUPLE:

Let FS', R', and Δ' be program variables.

Input: macro tree transducer $N = (FS, \Sigma, \Delta, R)$ with register functions,
macro tree transducer $M = (sim(FS), \Sigma, \Delta_M, R_M)$,
rule $l_1 \rightarrow \zeta_1 \in R$,
set S of ground function calls in ζ_1.

Output: FS', Δ', R'

Initialization: Let $FS' := FS$, $R' := R$, and $\Delta' := \Delta$.

Let $\zeta_1 = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \psi_1, \ldots, \psi_r)$ with $\langle f, \sigma, i\rangle \in reg(FS)^{(k+n+r)}$ for some $k, n \geq 0$ and $r > 0$.
Let $S = \{\psi_{p_1}, \ldots, \psi_{p_m}\}$ where $p_1, \ldots, p_m \in [r]$ and w.l.o.g. $p_1 < \ldots < p_m$.
Let, for every $\nu \in [m]$, $\psi_{p_\nu} = f_\nu(\theta_0, \theta_1, \ldots, \theta_v)$ and let $f_{new} = (f_1, \ldots, f_m)$.
Furthermore, let $l_2 \rightarrow \zeta_2$ be the rule in R where $l_2 = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, u_1, \ldots, u_r)$.
Perform the following three steps COMB, ADAPT, and NEW (in this order).

COMB: Define $\zeta_1' = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \psi_1', \ldots, \psi_{r'}')$ with $r' = r - m + 1$, and
$(\psi_1', \ldots, \psi_{r'}') = delpos((\psi_1, \ldots, \psi_{p_1-1}, \psi_{p_1}', \psi_{p_1+1}, \ldots, \psi_r), (p_2, \ldots, p_m))$
with $\psi_{p_1}' = f_{new}(\theta_0, \theta_1, \ldots, \theta_v)$.
$FS' := (FS' - \{\langle f, \sigma, i\rangle\}^{(k+n+r)}) \cup \{\langle f, \sigma, i\rangle^{(k+n+r')}, f_{new}\}$
$R' := (R' - \{l_1 \rightarrow \zeta_1\}) \cup \{l_1 \rightarrow \zeta_1'\}$

ADAPT: Define $l_2' = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, u_1', \ldots, u_{r'}')$ and
$(u_1', \ldots, u_{r'}') = delpos((u_1, \ldots, u_{p_1-1}, u_{p_1'}, u_{p_1+1}, \ldots, u_r), (p_2, \ldots, p_m))$
with $u_{p_1}' = comb_m(z_{p_1}, \ldots, z_{p_m})$.
$\Delta' := \Delta' \cup \{comb_m^{(m)}\}$
$R' := (R' - \{l_2 \rightarrow \zeta_2\}) \cup \{l_2' \rightarrow \zeta_2\}$

NEW: **if** $f_{new} \notin FS$ **then**
$R' := R' \cup \{f_{new}(\delta(x_1, \ldots, x_q), y_1, \ldots, y_v) \rightarrow comb_m(\zeta_{f_1,\delta}, \ldots, \zeta_{f_m,\delta}) \mid$
for every $\delta \in \Sigma^{(q)}$ with $q \geq 0\}$
fi
(* Note that $\zeta_{f_i,\delta}$ is the right-hand side of the (f_i, δ)-ground rule in R_M.*)

By executing such a tupling, equal ground function calls can arise on the right-hand sides of the new rules. Therewith applying $\vdash_{tuple,M}$ to a macro tree transducer with register functions enables us often to apply $\vdash_{split,M}$ and $\vdash_{share,M}$ even if this was not possible before.

Example 4.23 Let us consider the macro tree transducer N_3 with register functions of Example 4.13. The right-hand side $\zeta_{lift,\sigma}$ of the $(lift, \sigma)$-ground rule has the form

$$\langle f, \sigma, 1\rangle(x_1, x_2, y_1, lift(x_1, y_1), extr(x_2, \gamma(y_1)), extr(x_1, y_1)).$$

In this right-hand side there are two function calls with the same argument list (x_1, y_1), namely $lift(x_1, y_1)$ and $extr(x_1, y_1)$. Hence, $\zeta_{lift,\sigma}$ is ready for tupling the set $\{lift(x_1, y_1), extr(x_1, y_1)\}$ of ground function calls.

Since N_3 was constructed by applications of $\vdash_{split,M_1}$ and $\vdash_{share,M_1}$ starting from the macro tree transducer M_1 of Example 3.3, it holds that N_3 and M_1 are semantically equivalent. Furthermore it holds that N_3 is not ready for sharing in the $(lift, \sigma)$-ground rule. Hence, $N_3 \vdash_{tuple,M_1} N_4$ and N_4 is the macro tree transducer $(FS_4, \Sigma_1, \Delta_4, R_4)$ with register functions defined as follows:

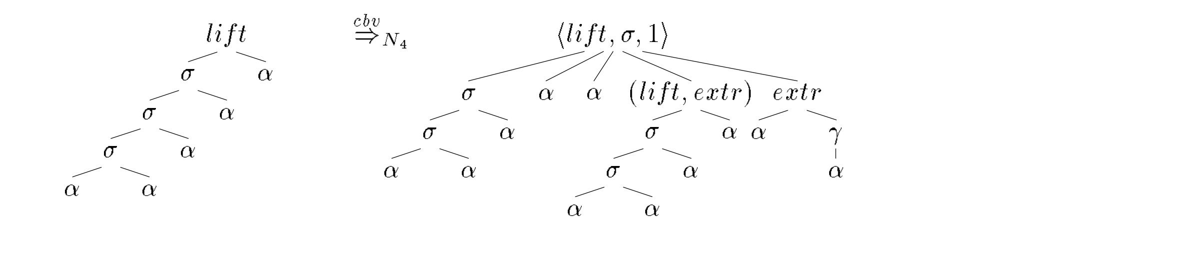
- $FS_4 = \{lift^{(2)}, \langle lift, \sigma, 1\rangle^{(5)}, (lift, extr)^{(2)}, extr^{(2)}, \langle extr, \sigma, 1\rangle^{(4)}\}$
- $\Delta_4 = \{\sigma^{(2)}, comb_2^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$
- R_4 is the set of rules as shown in Figure 22.

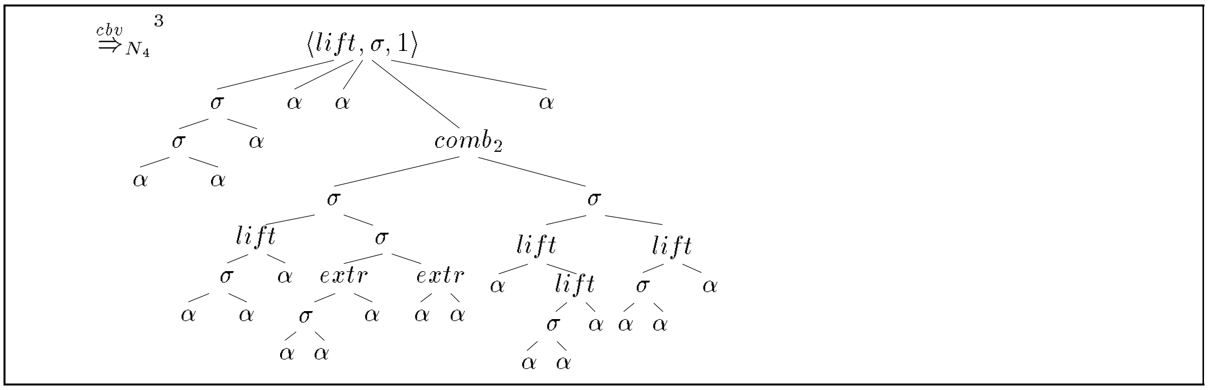
Note that N_4 is irreducible with respect to $\vdash_{tuple,M_1}$.

$$
\begin{array}{rcl}
lift(\alpha, y_1) & \to & y_1 \\
lift(\sigma(x_1, x_2), y_1) & \to & \langle lift, \sigma, 1\rangle(x_1, x_2, y_1, (lift, extr)(x_1, y_1), extr(x_2, \gamma(y_1))) \\
\langle lift, \sigma, 1\rangle(x_1, x_2, y_1, comb_2(z_1, z_3), z_2) & \to & \sigma(z_1, \sigma(z_2, z_3)) \\
 & & \\
(lift, extr)(\alpha, y_1) & \to & comb_2(y_1, \alpha) \\
(lift, extr)(\sigma(x_1, x_2), y_1) & \to & comb_2(\sigma(lift(x_1, y_1), \sigma(extr(x_2, \gamma(y_1)), extr(x_1, y_1))), \\
 & & \qquad \sigma(lift(x_2, lift(x_1, y_1)), lift(x_1, y_1))) \\
 & & \\
extr(\alpha, y_1) & \to & \alpha \\
extr(\sigma(x_1, x_2), y_1) & \to & \langle extr, \sigma, 1\rangle(x_1, x_2, y_1, lift(x_1, y_1)) \\
\langle extr, \sigma, 1\rangle(x_1, x_2, y_1, z_1) & \to & \sigma(lift(x_2, z_1), z_1)
\end{array}
$$

Figure 22: Rules of the macro tree transducer N_4 with register functions.

Let us show a part of the derivation by $\overset{cbv}{\Rightarrow}_{N_4}$ of the function call $t = lift(\sigma(\sigma(\sigma(\alpha, \alpha), \alpha), \alpha), \alpha)$ which is well-known from earlier derivation examples. The result is computed in 27 steps. In comparison to this derivation, the length of the derivation of t by $\overset{cbv}{\Rightarrow}_{N_3}$ was 23 (cf. Example 4.13). Note that during the $\overset{cbv}{\Rightarrow}_{N_4}$-derivation of t, function calls occur which were avoided in the derivations of the macro tree transducers with register functions N_2 and N_3. Hence, N_3 is sometimes better than N_4.





But there can also syntax-directed expressions found such that N_4 is sometimes better than N_3. Consider, e.g., the derivations of the function call $\psi = lift(\sigma(\alpha,\alpha),\alpha)$ by $\stackrel{cbv}{\Rightarrow}_{N_3}$ and $\stackrel{cbv}{\Rightarrow}_{N_4}$.

$$\begin{array}{lll} lift(\sigma(\alpha,\alpha),\alpha) & \stackrel{cbv}{\Rightarrow}_{N_3} & \langle lift,\sigma,1\rangle(\alpha,\alpha,\alpha,lift(\alpha,\alpha),extr(\alpha,\gamma(\alpha)),extr(\alpha,\alpha)) \\ & \stackrel{cbv}{\Rightarrow}_{N_3}^{3} & \langle lift,\sigma,1\rangle(\alpha,\alpha,\alpha,\alpha,\alpha,\alpha) \\ & \stackrel{cbv}{\Rightarrow}_{N_3} & \sigma(\alpha,\sigma(\alpha,\alpha)) \end{array}$$

The corresponding $\stackrel{cbv}{\Rightarrow}_{N_4}$-derivation has the following form:

$$\begin{array}{lll} lift(\sigma(\alpha,\alpha),\alpha) & \stackrel{cbv}{\Rightarrow}_{N_4} & \langle lift,\sigma,1\rangle(\alpha,\alpha,\alpha,(lift,extr)(\alpha,\alpha),extr(\alpha,\gamma(\alpha))) \\ & \stackrel{cbv}{\Rightarrow}_{N_4}^{2} & \langle lift,\sigma,1\rangle(\alpha,\alpha,\alpha,comb_2(\alpha,\alpha),\alpha) \\ & \stackrel{cbv}{\Rightarrow}_{N_4} & \sigma(\alpha,\sigma(\alpha,\alpha)) \end{array}$$

The normalform of ψ is computed in 5 steps by $\stackrel{cbv}{\Rightarrow}_{N_3}$ and in 4 steps by $\stackrel{cbv}{\Rightarrow}_{N_4}$.

This phenomenon is due to the construction of the rules for the tuple functions: the construction is based on the right-hand side of the macro tree transducer M_1. □

For the above mentioned reason, if $N \vdash_{tuple,M} N'$, then statements about the efficiency of N with respect to N' are not possible.

Let us determine some properties of $\vdash_{tuple,M}$.

Remark 4.24 Let $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma, \Delta', R')$ be macro tree transducers with register functions and let M be a macro tree transducer such that $N \vdash_{tuple,M} N'$.

1. There is exactly one rule $l \to \zeta$ in R such that $l \to \zeta'$ is in R' and $\zeta \neq \zeta'$. There is exactly one set S of ground function calls such that ζ is ready for tupling S and ζ' is not ready for tupling S. The rule $l \to \zeta$ is called *in S tupled rule of N and N'*.

2. For this set S it holds that $card(S) \geq 2$.

3. Either $card(FS) = card(FS')$ or $card(FS) = card(FS') - 1$.

4. If $card(FS) = card(FS')$, then $card(R) = card(R')$.

5. If $card(FS) = card(FS') - 1$, then $card(R) < card(R')$. More precisely, $card(R') = card(R) + card(\Sigma)$.

6. Let $l_1 \to \zeta_1$ be the rule in R such that $\zeta_1 = g(\tilde{x}, \tilde{y}, \psi_1, \ldots, \psi_r)$ is ready for tupling the set $\{\psi_{p_1}, \ldots, \psi_{p_m}\}$ of ground function calls with $p_1, \ldots, p_m \in [r]$ and $p_1 < \ldots < p_m$.

Furthermore let $l_2 \to \zeta_2$ be the rule in R with $root(l_2) = g = root(\zeta_1)$. Then it holds that in $l_2 = g(\tilde{x}, \tilde{y}, u_1, \ldots, u_r)$ the arguments $u_{p_1}, \ldots, u_{p_m}$ are variables in Z, more precisely, $(u_{p_1}, \ldots, u_{p_m}) = (z_{p_1}, \ldots, z_{p_m})$. $\square$

As usually, we examine if the defined relation $\vdash_{tuple,M}$ is semantic preserving, locally confluent, and terminating.

Lemma 4.25 *Let $N = (FS, \Sigma, \Delta, R)$ and $N' = (FS', \Sigma', \Delta', R')$ be macro tree transducers with register functions. If $N \vdash_{tuple,M} N'$, then*

(a) For every $f \in gr(FS)^{(n+1)}$ with $n \geq 0$, $s \in T\langle\Sigma\rangle$, $t_1, \ldots, t_n \in T\langle\Delta\rangle$,

$$\tau_{cbv}(N)(f, s, t_1, \ldots, t_n) = \tau_{cbv}(N')(f, s, t_1, \ldots, t_n).$$

(b) N' and M are semantically equivalent.

Proof. Let $M = (F_M, \Sigma_M, \Delta_M, R_M)$ be the macro tree transducer such that $N \vdash_{tuple,M} N'$. It holds that N and M are semantically equivalent.
(a) Since the function τ_{cbv} is total, it suffices to prove that $\tau_{cbv}(N) \subseteq \tau_{cbv}(N')$.

Before we start with the proof we make some preliminary considerations.

By definition of $\vdash_{tuple,M}$ there exists a rule $l_1 \to \zeta_1$ in R such that ζ_1 is ready for tupling the set S of ground function calls, and there is a rule $l_1 \to \zeta_1'$ in R' such that $\zeta_1' \neq \zeta_1$ and ζ_1' is not ready for tupling S (cf. Remark 4.24 1.). By definition, ζ_1 is of the form $\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \psi_1, \ldots, \psi_r)$ and $\zeta_1' = \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \psi_1', \ldots, \psi_{r'}')$ where $r' < r$. Furthermore there is a rule $\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, u_1, \ldots, u_r) \to \zeta_2$ in R and a rule $\langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, u_1', \ldots, u_{r'}') \to \zeta_2$ in R'.

No other rules are changed by $\vdash_{tuple,M}$, but there may be $card(\Sigma)$ new rules in R' which are not in R. Let $S = \{\psi_{p_1}, \ldots, \psi_{p_m}\}$ where $m > 1$, $p_1, \ldots, p_m \in [r]$ and $p_1 < \ldots < p_m$. Let, for every $\nu \in [m]$, $root(\psi_{p_\nu}) = f_\nu$.

By definition of $\vdash_{tuple,M}$ it holds that

$$(\psi_1', \ldots, \psi_{r'}') = delpos((\psi_1, \ldots, \psi_{p_1-1}, \psi_{p_1}', \psi_{p_1+1}, \ldots, \psi_r), (p_2, \ldots, p_r))$$

and $\psi_{p_1}' = \psi_{p_1}[\varepsilon \leftarrow (f_1, \ldots, f_m)]$.

It suffices to prove that, for every $s = \sigma(s_1, \ldots, s_k) \in T\langle\Sigma\rangle$ and $t_1, \ldots, t_n \in T\langle\Delta\rangle$, the following two statements hold.

1. If $l_1 = f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n)$ and there is an irreducible $\xi \in T\langle\Delta\rangle$ such that

$$f(s, t_1, \ldots, t_n) \overset{cbv}{\Rightarrow}_N^* \xi,$$

then there exists a derivation

$$f(s, t_1, \ldots, t_n) \overset{cbv}{\Rightarrow}_{N'}^* \xi.$$

2. Let $\lambda_1, \ldots, \lambda_{r_{i-1}} \in sdExp(FS, \Sigma, \Delta)$ for some $r_{i-1} \geq 0$. If $l_1 = \langle f, \sigma, i-1\rangle(\tilde{x}, \tilde{y}, u_1, \ldots, u_{r_{i-1}})$ and there exists an irreducible $\xi \in sdExp(reg(FS), \Sigma, \Delta)$ such that

$$\langle f, \sigma, i-1\rangle(\tilde{s}, \tilde{t}, \lambda_1, \ldots, \lambda_{r_{i-1}}) \overset{cbv}{\Rightarrow}_N^* \xi,$$

then there exists a derivation

$$\langle f, \sigma, i-1\rangle(\tilde{s}, \tilde{t}, \lambda_1, \ldots, \lambda_{r_{i-1}}) \overset{cbv}{\Rightarrow}_{N'}^* \xi.$$

Proof.

1. Let $l_1 = f(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n)$. We prove this fact by induction on the height of s. Also here we have to start the proof with $height(s) = 1$, because in the case $height(s) = 0$, the right-hand side ζ_1 does not fulfill the assumptions: by definition, no ground function call can occur.

   (i) $height(s) = 1$, i.e., $s = \sigma(\alpha_1, \ldots, \alpha_k)$. The $\overset{cbv}{\Rightarrow}_N$-derivation of $f(s, t_1, \ldots, t_n)$ has the following form:

   $$\begin{array}{lll} f(s, t_1, \ldots, t_n) & \overset{cbv}{\Rightarrow}_N & \langle f, \sigma, 1\rangle(\tilde{\alpha}, \tilde{t}, \theta_1, \ldots, \theta_r) \\ & \overset{cbv}{\Rightarrow}_N^{\ *} & \langle f, \sigma, 1\rangle(\tilde{\alpha}, \tilde{t}, \xi_1, \ldots, \xi_r) \\ & & \text{where, for every } j \in [r],\ \xi_j = nf(\overset{cbv}{\Rightarrow}_N, \theta_j) \\ & \overset{cbv}{\Rightarrow}_N & \varphi(\zeta_2) \\ & & \text{where } \varphi = [x_\nu/\alpha_\nu; \nu \in [k]][y_\mu/t_\mu; \mu \in [n]][z_\nu/\xi_\nu; \nu \in [r]] \\ & \overset{cbv}{\Rightarrow}_N^{\ \infty} & \xi \end{array}$$

   The rules for nullary constructors have not been changed by $\vdash_{tuple,M}$, because there are no function calls in the right-hand sides of these rules. Hence, it holds that

   $$\text{for every } j \in [r],\quad nf(\overset{cbv}{\Rightarrow}_N, \theta_j) = nf(\overset{cbv}{\Rightarrow}_{N'}, \theta_j).$$

   Let us consider the $\overset{cbv}{\Rightarrow}_{N'}$ derivation:

   $$f(s, t_1, \ldots, t_n) \overset{cbv}{\Rightarrow}_{N'} \langle f, \sigma, 1\rangle(\tilde{\alpha}, \tilde{t}, \theta'_1, \ldots, \theta'_{r'})$$

   where by definition of $\vdash_{tuple,M}$ and $delpos$ for every $j \in [r']$, $j \neq p_1$, there is an $i \in [r]$ such that $\theta'_j = \theta_i$ and $\theta'_{p_1} = (f_1, \ldots, f_m)(\alpha_d, \delta_1, \ldots, \delta_n)$, if $\theta_{p_1} = f_1(\alpha_d, \delta_1, \ldots, \delta_n)$ with $d \in [k]$.

   Hence, instead of having m function calls $f_1(\alpha_d, \delta_1, \ldots, \delta_n), \ldots, f_m(\alpha_d, \delta_1, \ldots, \delta_n)$ we have one function call $(f_1, \ldots, f_m)(\alpha_d, \delta_1, \ldots, \delta_n)$. Since $height(\alpha_d) = 0$, we have to consider the $((f_1, \ldots, f_m), \alpha_d)$-rule of R' which was constructed by the Algorithm TUPLE:

   $$(f_1, \ldots, f_m)(\alpha_d, y_1, \ldots, y_n) \to comb_m(\zeta_{f_1,\alpha_d}, \ldots \zeta_{f_m,\alpha_d})$$

   where, for every $i \in [m]$, ζ_{f_i,α_d} denotes the right-hand side of the (f_i, α_d)-ground rule in the macro tree transducer M. Hence by the fact that $\tau_{cbv}(M) = \tau_{cbv}(N)$ it holds that, for every $i \in [m]$, $f_i(\alpha_d, y_1, \ldots, y_n) \to \zeta_{f_i,\alpha_d}$ are rules in R. It follows directly by the fact that $\overset{cbv}{\Rightarrow}_{N'}$ is confluent that

   $$nf(\overset{cbv}{\Rightarrow}_{N'}, \theta'_{p_1}) = comb_m(nf(\overset{cbv}{\Rightarrow}_N, \theta_{p_1}), \ldots, nf(\overset{cbv}{\Rightarrow}_N, \theta_{p_m})).$$

   Hence,

   $$\langle f, \sigma, 1\rangle(\tilde{\alpha}, \tilde{t}, \theta'_1, \ldots, \theta'_{r'}) \overset{cbv}{\Rightarrow}_{N'}^{\ *} \langle f, \sigma, 1\rangle(\tilde{\alpha}, \tilde{t}, \xi'_1, \ldots, \xi'_{r'})$$

   where
   $(\xi'_1, \ldots, \xi'_{r'}) = delpos((\xi_1, \ldots, \xi_{p_1-1}, comb_m(\xi_1, \ldots, \xi_{p_m}), \xi_{p_1+1}, \ldots, \xi_r), (p_2, \ldots, p_m))$

By definition of $\vdash_{tuple,M}$ (part ADAPT in Algorithm TUPLE) there is a rule in R' which can be applied to the syntax-directed expression $\langle f,\sigma,1\rangle(\tilde{\alpha},\tilde{t},\xi'_1,\ldots,\xi'_{r'})$, namely the rule $\langle f,\sigma,1\rangle(\tilde{x},\tilde{y},u'_1,\ldots,u'_{r'}) \to \zeta_2$ and $u'_{p_1} = comb_m(u_{p_1},\ldots,u_{p_m})$.

$$\langle f,\sigma,1\rangle(\tilde{\alpha},\tilde{t},\xi'_1,\ldots,\xi'_{r'}) \overset{cbv}{\Rightarrow}{}_{N'}^{*} \varphi'(\zeta_2)$$

and (comparing this derivation with the derivation by $\overset{cbv}{\Rightarrow}_N$) it holds that $\varphi' = \varphi$, because

$$(u'_1,\ldots,u'_r) = delpos((u_1,\ldots,u_{p_1-1},comb_m(u_{p_1},\ldots,u_{p_m}),u_{p_1+1},\ldots,u_r),(p_2,\ldots,p_m)),$$

i.e., the arguments of the left-hand side l_2 are re-ordered in the same way as the arguments in ζ_1 and the evaluation of θ'_{p_1} created the symbol $comb_m$ at its root which is consumed while the application of the rule $l'_2 \to \zeta_2$.

During further $\overset{cbv}{\Rightarrow}_{N'}$-derivation steps of $\varphi(\zeta_2)$ there have only function symbols to be applied of which the rules were not changed. Hence,

$$\varphi(\zeta_2) \overset{cbv}{\Rightarrow}{}_{N'}^{\infty} \xi$$

(ii) Let $height(s) = \rho$, i.e., $s = \sigma(s_1,\ldots,s_k)$. We argue in the same way as in (i) except that now for every $\overset{cbv}{\Rightarrow}_N$-evaluation of function calls θ_j there exists a $\overset{cbv}{\Rightarrow}_{N'}$-evaluation of θ_j because of the induction hypothesis and because of the fact that M is semantically equivalent to N.

2. This part of the proof is similar to 1.

To prove (b) the following two conditions have to be proved:

(b_1) For every $f \in F_M^{(n+1)}$ with $n \geq 0$, $s \in T\langle\Sigma\rangle$, $t_1,\ldots,t_n \in T\langle\Delta_M\rangle$, $\tau_{cbv}(M)(f,s,t_1,\ldots,t_n) = \tau_{cbv}(N')(f,s,t_1,\ldots,t_n)$.

(b_2) For every $(f_1,\ldots,f_m) \in tup(FS)^{(n+1)}$ for some $n \geq 0$, $s \in T\langle\Sigma\rangle$, $t_1,\ldots,t_n \in T\langle\Delta_M\rangle$, if, for every $i \in [m]$, $\tau_{cbv}(M)(f_i,s,t_1,\ldots,t_n) = \xi_1$, then $\tau_{cbv}(N)((f_1,\ldots,f_m),s,t_1,\ldots,t_n)) = comb_m(\xi_1,\ldots,\xi_m)$ and $comb_m \in \Delta^{(m)}$.

Condition (b_1) follows directly by (a), because $F_M \subseteq gr(FS)$.

Since M and N are semantically equivalent (by definition), it follows that for every $(f_1,\ldots,f_m) \in tup(FS)$ with $f_1,\ldots,f_m \in sim(FS)^{(n+1)}$, $s \in T\langle\Sigma\rangle$, $t_1,\ldots,t_n \in T\langle\Delta\rangle$, if

$$f_1(s,t_1,\ldots,t_n) \overset{cbv}{\Rightarrow}{}_M^{\infty} \xi_1$$

$$\vdots$$

$$f_m(s,t_1,\ldots,t_n) \overset{cbv}{\Rightarrow}{}_M^{\infty} \xi_m,$$

then

$$(f_1,\ldots,f_m)(s,t_1,\ldots,t_n) \overset{cbv}{\Rightarrow}{}_N^{\infty} comb_m(\xi_1,\ldots,\xi_m)$$

With Part (a) and the fact that $tup(FS) \subseteq tup(FS')$, it follows that

$$(f_1, \ldots, f_m)(s, t_1, \ldots, t_n) \stackrel{cbv}{\Rightarrow}{}^{\infty}_{N'} comb_m(\xi_1, \ldots, \xi_m)$$

By Remark 4.24, if $tup(FS) = tup(FS')$, then the statement is proved. If $tup(FS) \neq tup(FS')$, then FS' contains exactly one tuple function more than N. Let $(g_1, \ldots, g_k)$ be this function. It follows directly by construction that if

$$g_1(s, t_1, \ldots, t_n) \stackrel{cbv}{\Rightarrow}{}^{\infty}_{M} \xi_1$$

$$\vdots$$

$$g_k(s, t_1, \ldots, t_n) \stackrel{cbv}{\Rightarrow}{}^{\infty}_{M} \xi_k,$$

then

$$(g_1, \ldots, g_k)(s, t_1, \ldots, t_n) \stackrel{cbv}{\Rightarrow}{}^{\infty}_{N} comb_m(\xi_1, \ldots, \xi_k).$$

□

Lemma 4.26 *The relation $\vdash_{tuple,M}$ is locally confluent.*

Proof. Let $N = (FS, \Sigma, \Delta, R)$, $N' = (FS', \Sigma, \Delta', R')$, and $N'' = (FS'', \Sigma, \Delta'', R'')$ be macro tree transducers with register functions and let M be a macro tree transducer such that $N \vdash_{tuple,M} N'$ and $N \vdash_{tuple,M} N''$ and $N \neq N'$.

We have to show that there exists a macro tree transducer $\tilde{N}$ with register functions such that $N' \vdash^*_{tuple,M} \tilde{N}$ and $N'' \vdash^*_{tuple,M} \tilde{N}$. By definition it follows that there are two rules $l_1 \rightarrow \zeta_1$ and $l_2 \rightarrow \zeta_2$ in R and two sets S and T of ground function calls such that the following conditions hold:

- ζ_1 is ready for tupling S, ζ_2 is ready for tupling T
- if $\zeta_1 = \zeta_2$, then $S \neq T$
- There is a rule $l_1 \rightarrow \zeta_1'$ in R' such that ζ_1' is not ready for tupling S.
- There is a rule $l_2 \rightarrow \zeta_2'$ in R'' such that ζ_2' is not ready for tupling T.

Let us consider the tupling transformation relation. If, for some macro tree transducers N_1 and N_2 with register functions, $N_1 \vdash_{tuple,M} N_2$, then two rules of the set of rules of N_1 occur modified in the set of rules of N_2: more precisely, the right-hand side of one rule of which the root is labeled by a register function $\langle f, \sigma, i\rangle$, and the left-hand side of the another rule of which the root is labeled by the same register function $\langle f, \sigma, i\rangle$, are modified. Furthermore, N_2 may contain some more rules of a new tuple function. These rules cannot be ready for tupling because their right-hand sides do not contain any register function. Their construction is based on the rules of the macro tree transducer M.

We have to distinguish the following two cases: (a) $l_1 \rightarrow \zeta_1$ and $l_2 \rightarrow \zeta_2$ are different, i.e., $l_1 \neq l_2$ and (b) $l_1 \rightarrow \zeta_1$ and $l_2 \rightarrow \zeta_2$ are equal.

(a) Here, three cases are possible.

1. The rule $l_2 \to \zeta_2$ is in N' and the rule $l_1 \to \zeta_1$ is in N''. Then ζ_2 is ready for tupling T and ζ_1 is ready for tupling S. Hence, $N' \vdash_{tuple,M} \tilde{N}'$ and $\tilde{N}'$ contains (in comparison with N') the same new rules as N''. Also $N'' \vdash_{tuple,M} \tilde{N}''$ and $\tilde{N}''$ contains (in comparison with N'') the same new rules as N'. By trivial comparison of the executed changes to the rules it holds that $\tilde{N}' = \tilde{N}'' = \tilde{N}$.
2. The rule $l_2 \to \zeta_2$ is in N' and instead of the rule $l_1 \to \zeta_1$, the rule $l'_1 \to \zeta_1$ is in N'', i.e., $root(l_1) = root(\zeta_2)$. As before, ζ_2 is ready for tupling T and ζ_1 is ready for tupling S. The rest of this case is similar to 1.
3. Instead of the rule $l_2 \to \zeta_2$, the rule $l'_2 \to \zeta_2$ is in N' and the rule $l_1 \to \zeta_1$ is in N'', i.e., $root(l_2) = root(\zeta_1)$. Compare case 2.

(b) If the rules are equal, then S and T are different (otherwise it would hold that $N' = N''$). By definition it holds that $S \cap T = \emptyset$. ζ'_1 is still ready for tupling T and ζ'_2 is ready for tupling S (note that $l_2 = l_1$, but $\zeta'_1 \neq \zeta'_2$). Hence, $N' \vdash_{tuple,M} \tilde{N}'$ and $N'' \vdash_{tuple,M} \tilde{N}''$. By comparing the rules of $\tilde{N}'$ and $\tilde{N}''$ it holds that $\tilde{N}' = \tilde{N}'' = \tilde{N}$. □

Lemma 4.27 *The relation $\vdash_{tuple,M}$ is noetherian.*

Proof. Let $N = (FS, \Sigma, \Delta, R)$ be a macro tree transducer with register functions and let M be a macro tree transducer such that M and N are semantically equivalent. For every rule $l \to \zeta$ in N there is a unique number $k(l \to \zeta)$ of different sets $S_1, \ldots, S_{k(l\to\zeta)}$ such that ζ is ready for tupling the set S_i of ground function calls. The rules which are newly created by the TUPLE-Algorithm are rules which have no register function calls in their right-hand sides. Hence, no such right-hand side can be ready for tupling. Since the number of rules is finite and to every rule a finite number is associated, it holds that the relation $\vdash_{tuple,M}$ is noetherian. □

Lemma 4.28 *The relation $\vdash_{tuple,M}$ is confluent and noetherian.*

Proof. This result follows directly by the fact that $\vdash_{tuple,M}$ is locally confluent and noetherian (cf. previous lemmas) and by Lemma 2.4 of [Hue80]. □

Lemma 4.29 *Let M be a macro tree transducer.*

1. $\vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \subseteq \vdash^{\infty}_{tuple,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$.

2. $\vdash_{split,M} \circ \vdash^{\infty}_{tuple,M} \subseteq \vdash^{\infty}_{tuple,M} \circ \vdash_{split,M} \circ \vdash^{\infty}_{tuple,M}$.

Proof. Let $N = (FS, \Sigma, \Delta, R)$ and N' be two macro tree transducers with register functions.

1. Let $N \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} N'$. We have to show that also $N \vdash^{\infty}_{tuple,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} N'$ holds. If $N \vdash^{\infty}_{tuple,M} N$ or $N \vdash^{\infty}_{share,M} N$, then the statement holds trivially. Otherwise there are rules $l_1 \to \zeta_1, \ldots, l_m \to \zeta_m$ with $m \geq 1$ in R such that, for every $i \in [m]$, ζ_i is ready for tupling $S_{i,1}, \ldots, S_{i,n}$ for some $n \geq 1$. It holds that N is not ready for sharing in some of these rules because this is a condition for the applicability of the tupling transformation relation. Note that this condition makes sure that rules which are both, ready for sharing and ready for tupling, are first changed by the sharing transformation relation. Let $l_{j_1} \to \zeta_{j_1}, \ldots, l_{j_k} \to \zeta_{j_k}$ be these rules. On the one hand the tupling transformation relation

enables or disables no further tupling and, on the other hand, no further sharing, i.e, $N \vdash^{\infty}_{tuple,M} N_1$ and N_1 is ready for sharing, because N is ready for sharing. Hence, there is a macro tree transducer N_2 with register functions such that $N_1 \vdash^{\infty}_{share,M} N_2$. By trivial comparison of the changes applied to the rules it follows that $nf(N_1, \vdash_{tuple,M}) = N'$.

Note that the sharing transformation relation may enable the applicability of the tupling transformation relation. For this reason, only the $\subseteq$ relation holds.

2. Let $N \vdash_{split,M} \circ \vdash^{\infty}_{tuple,M} N'$. We have to show that also $N \vdash^{\infty}_{tuple,M} \circ \vdash_{split,M} \circ \vdash^{\infty}_{tuple,M} N'$ holds. This proof is similar to the proof of statement 1.: if $N \vdash^{\infty}_{tuple,M} N$, then the statement holds trivially. The splitting transformation relation may enable but not disable the tupling transformation.

$\square$

5 Transformation strategy

As explained in the introduction we want to present a method how to transform a macro tree transducer M into a macro tree transducer N with register functions such that M and N are semantically equivalent and N is more efficient than M. Until now we have defined three different transformation relations. Now we define how they have to be combined such that the goal is reached.

Definition 5.1 Let $M = (F_M, \Sigma, \Delta_M, R_M)$ be a macro tree transducer. The *reducing transformation relation* with respect to M is the binary relation $\vdash_{red,M} \subseteq \mathcal{N} \times \mathcal{N}$ defined as follows. Let $N = (FS, \Sigma, \Delta, R)$ and N' be two macro tree transducers with register functions. It holds that $N \vdash_{red,M} N'$, if the following conditions hold:

- M and N are semantically equivalent.
- There is a rule $l \to \zeta$ in R such that $Cut(\zeta) \neq \emptyset$.

Then N' is determined by the transformation $N \vdash_{split,M} N_1 \vdash^{\infty}_{share,M} N_2 \vdash^{\infty}_{tuple,M} N'$. $\square$

Let us give a short example for the application of $\vdash_{red,M}$.

Example 5.2 Consider the macro tree transducer N_1 with register functions of Example 4.3. It is well-known that M_1 (cf. Example 3.3) and N_1 are semantically equivalent. For the right-hand side ζ of the $(lift, \sigma)$-rule of N_1 it holds that $Cut(\zeta) \neq \emptyset$ (cf. Example 4.3). By $N_1 \vdash_{red,M_1} N'$, it holds that N' is the result of the transformation $N_1 \vdash_{split,M_1} N_1' \vdash^{\infty}_{share,M_1} N_1'' \vdash^{\infty}_{tuple,M_1} N'$ for some macro tree transducers N_1' and N_1'' with register functions. By the previous Examples 4.3, 4.13, and 4.23, it holds that $N_1 \vdash_{split,M_1} N_2$, $N_2 \vdash^{\infty}_{share,M_1} N_3$, and $N_3 \vdash^{\infty}_{tuple,M_1} N_4$.

Hence, we can choose $N' = N_4$.

Remark 5.3 Let $N = (FS, \Sigma, \Delta, R)$ be a macro tree transducer with register functions and M be a macro tree transducer such that M and N are semantically equal.

1. If there is a macro tree transducer N' with register functions such that $N \vdash_{red,M} N'$, then $N' = (FS', \Sigma, \Delta', R')$ with $sim(FS) = sim(FS')$, $tup(FS) \subseteq tup(FS')$, $reg(FS) \subset reg(FS')$, and $\Delta \subseteq \Delta'$.
2. If $N \vdash_{red,M} N'$ with $N' = (FS', \Sigma, \Delta', R')$ and $\Delta \subset \Delta'$, then $tup(FS) \subset tup(FS')$.
3. If there are exactly k different rules $l_1 \to \zeta_1, \ldots, l_k \to \zeta_k$ in R such that, for every $i \in [k]$, $Cut(\zeta_i) \neq \emptyset$, then there are exactly k different macro tree transducers with register functions $N_1, \ldots, N_k$ such that, for every $i \in [k]$, $N \vdash_{red,M} N_i$. $\square$

Lemma 5.4 *Let M be a macro tree transducer. The following holds:*

1. $\vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \subseteq \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$
   (or equivalently: $\vdash_{red,M} \subseteq \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash_{red,M}$*)*
2. $\vdash_{split,M} \circ \vdash_{red,M} \subseteq \vdash_{red,M} \circ \vdash_{red,M}$

Proof.

1.

$$\begin{array}{lll} & \vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} & \text{by Lemma 4.20} \\ \subseteq & \vdash^{\infty}_{share,M} \circ \vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} & \text{by Lemma 4.29 1.} \\ \subseteq & \vdash^{\infty}_{share,M} \circ \vdash_{split,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} & \text{by Lemma 4.29 2.} \\ \subseteq & \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash_{split,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} & \end{array}$$

Now we analyse the transformation relation which results from the sequence of inclusions. Let N and N_1 be two macro tree transducers with register functions such that $N \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} N_1$. Since the tupling transformation relation does not introduce new sharing possibilities, $N_1 = nf(N_1, \vdash_{share,M})$. Applying the transformation relation $\vdash_{split,M}$ to N_1 introduces exactly one rule $l \to \zeta$ which may be ready for sharing or tupling, i.e., if $N_1 \vdash_{split,M} N_2$, then the transformation relation $\vdash^{\infty}_{tuple,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$ is concentrated to this rule $l \to \zeta$ and to the rule $l' \to \zeta'$ such that $root(\zeta) = root(l')$. Note that neither $\vdash_{tuple,M}$ nor $\vdash_{share,M}$ introduces rules which can be ready for sharing or tupling. By definition of the splitting transformation relation, it holds that ζ' is a ground right-hand side, i.e., $l' \to \zeta'$ cannot be a shared or tupled rule. Now two cases are possible:

- If $\vdash_{tuple,M}$ can be applied to N_2, then $l \to \zeta$ must be the tupled rule. By definition of the sharing transformation relation, ζ is not ready ready for sharing (this was a condition for the applicability of the tupling transformation relation). No other rule can be ready for sharing by the previous considerations. Hence, if $N_2 \vdash^{\infty}_{tuple,M} N_3$, then $N_3 = nf(N_3, \vdash_{share,M})$ and $\vdash_{split,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$ is simplified to $\vdash_{split,M} \circ \vdash^{\infty}_{tuple,M}$.
- If $\vdash_{tuple,M}$ cannot be applied to N_2, then $\vdash_{split,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$ is simplified to $\vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$. Note that, since $\vdash^{\infty}_{share,M}$ is local to the rule $l \to \zeta$, $\vdash^{\infty}_{share,M}$ is equal to $\vdash_{share,M}$, if sharing is applicable.

Altogether, the statement holds.

2.

$$\begin{array}{lll} & \vdash_{split,M} \circ \vdash_{red,M} & \text{by Definition 5.1} \\ = & \vdash_{split,M} \circ \vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} & \text{by 1.} \\ \subseteq & \vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} & \text{by Definition 5.1} \\ = & \vdash_{red,M} \circ \vdash_{red,M} & \end{array}$$

□

In the following, we call a $\vdash_{red,M}$-step *local*, if the involved $\vdash_{share,M}$ and $\vdash_{tuple,M}$ transformation relations only change the rules which were created by the splitting transformation relation. Note that for such a local $\vdash_{red,M}$-step the involved sharing transformation relation is applied at most once.

Lemma 5.5 *If $N \vdash_{red,M} N'$, then N and N' are semantically equivalent.*

Proof. This statement follows directly by the fact that $\vdash_{red,M}$ is a sequence of relations for which it is was proved that they are semantic-preserving. □

$$
\begin{array}{lllll}
 & & N'_1 \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \mathbf{N_1} \vdash_{split,M} \tilde{N}_1 \vdash^{\infty}_{share,M} \circ & & \\
 & \nearrow^{split,\,M} & & \searrow^{\infty}_{tuple,\,M} & \\
\mathbf{N} \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} N' & & & & \mathbf{N_3} \\
 & \searrow_{split,\,M} & & \nearrow^{\infty}_{tuple,\,M} & \\
 & & N'_2 \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \mathbf{N_2} \vdash_{split,M} \tilde{N}_2 \vdash^{\infty}_{share,M} \circ & &
\end{array}
$$

Figure 23: Proof of the local confluence of $\vdash_{red,M}$.

Lemma 5.6 *The relation $\vdash_{red,M}$ is locally confluent.*

<u>Proof.</u> Let N, N_1, and N_2 be macro tree transducers with register functions such that $N \vdash_{red,M} N_1$ and $N \vdash_{red,M} N_2$ where $N_1 \neq N_2$. We prove that there exists an N_3 such that $N_1 \vdash_{red,M} N_3$ and $N_2 \vdash_{red,M} N_3$.

By Lemma 5.4 it follows that

$$N \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash_{red,M} N_1 \quad \text{and}$$

$$N \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash_{red,M} N_2.$$

Since the normalforms are unique it holds that there exists a macro tree transducer N' with register functions such that

$$N \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} N' \quad \text{and} \quad N' \vdash_{red,M} N_1 \quad \text{and} \quad N' \vdash_{red,M} N_2$$

and thus (by Definition 5.1) there are N'_1 and N'_2 such that

$$N' \vdash_{split,M} N'_1 \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} N_1 \quad \text{and}$$

$$N' \vdash_{split,M} N'_2 \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} N_2.$$

It holds that $N'_1 \neq N'_2$ because of $N_1 \neq N_2$ and because of the fact that normalforms of $\vdash_{share,M}$ and $\vdash_{tuple,M}$ are unique. Hence, there exist two rules $l_1 \to \zeta_1$ and $l_2 \to \zeta_2$ in N' such that $Cut(\zeta_1) \neq \emptyset$ and $Cut(\zeta_2) \neq \emptyset$. As we have seen in the proof of Lemma 5.4, the relation $\vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$ is local to the rules which are constructed by splitting the rules $l_1 \to \zeta_1$ and $l_2 \to \zeta_2$. Hence, the rule $l_2 \to \zeta_2$ is still in the set of rules of N_1 and the rule $l_1 \to \zeta_1$ is still in the set of the rules of N_2. Therewith $\vdash_{split,M}$ can be applied to N_1 and N_2, i.e., $N_1 \vdash_{split,M} \tilde{N}_1$ and $N_2 \vdash_{split,M} \tilde{N}_2$ for some $\tilde{N}_1$ and $\tilde{N}_2$. As before, the relation $\vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$ is local to the new constructed rules. Let $\tilde{N}_1 \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} N_3$. By trivial comparison of the rules it follows that $\tilde{N}_2 \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} N_3$. Hence, the statement is proved. The proof is pictured in Figure 23. $\square$

Lemma 5.7 *The relation $\vdash_{red,M}$ is noetherian.*

<u>Proof.</u> Let $M = (F_M, \Sigma, \Delta_M, R_M)$ be a macro tree transducer and let N_0 be a macro tree transducer with register functions which is semantically equivalent to M. Consider an arbitrary derivation

$$N_0 \vdash_{red,M} N_1 \vdash_{red,M} N_2 \vdash_{red,M} \ldots \vdash_{red,M} N_r \vdash_{red,M} N_{r+1} \vdash_{red,M} \ldots$$

of $\vdash_{red,M}$. By Lemma 5.4 a sequence of $\vdash_{red,M}$-steps can be reordered as follows: if $N_0 \vdash_{red,M} N_1$, then $N_0 \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M} \circ \vdash_{red,M} N_1$ and the $\vdash_{red,M}$ step is local. The same holds for all other $\vdash_{red,M}$-steps, because, for every $i > 0$, N_i is the normalform of N_i with respect to the $\vdash_{share,M}$ and $\vdash_{tuple,M}$ transformation relation.

We observe that $\vdash_{tuple,M}$ is the only transformation relation (w.r.t. the three basic transformation relations) which can introduce new rules with the property that their right-hand sides can be splitted. In particular, these right-hand sides are combinations of right-hand sides of the macro tree transducer M (cf. TUPLE-Algorithm, Part NEW). However, the left-hand sides of such new rules have tuple functions as root label and, since there are only finitely many permutations over the set F_M and since no two rules start with the same tuple function and input symbol, this process of adding new rules eventually stops.

For a finite set of given rules, $\vdash_{split,M}$ is noetherian. Hence, assuming that no more new rules are added, $\vdash_{red,M}$ is also noetherian. □

Lemma 5.8 *The relation $\vdash_{red,M}$ is confluent and noetherian.*

Proof. This result follows directly by the fact that $\vdash_{red,M}$ is locally confluent and noetherian (cf. previous lemmas) and by Lemma 2.4 of [Hue80]. □

Definition 5.9 Let M be a macro tree transducer. The *recursive-iterative tree transducer* associated with M, denoted by *rec-it*(M), is the macro tree transducer $nf(\vdash_{red,M}, M)$ with register functions. □

Example 5.10 Let us continue our running example by showing the recursive-iterative tree transducer associated with M_1 (cf. Example 3.3). It can be computed by the following transformation sequence:

$$M_1 \underbrace{\vdash_{split,M_1} N_1 \vdash_{share,M_1}}_{\vdash_{red,M_1}} N_3' \underbrace{\vdash_{split,M_1} N_3 \vdash_{tuple,M_1}}_{\vdash_{red,M_1}} N_4 \underbrace{\vdash_{split,M_1} \circ \vdash_{share,M_1} \circ \vdash_{tuple,M_1}}_{\vdash_{red,M_1}} \textit{rec-it}(M_1)$$

where

- the macro tree transducer N_1 with register functions is the one from Example 4.3 (the rules are shown in Figure 13),
- the macro tree transducer N_3' with register functions is obtained from N_3 (cf. Example 4.13, Figure 19) as follows: the $extr$-rules of N_3' are equal to those of N_3 and the $lift$-rules are the ones of N_1, and
- the macro tree transducer N_4 with register functions can be found in Example 4.23.

The rules of *rec-it*(M_1) are shown in Figure 24. The following pictures Figures 25 and 26 show a derivation of the well-known function call by $\stackrel{cbv}{\Rightarrow}_{\textit{rec-it}(M_1)}$. It is remarkable that before computing the results of more complex subcalls, the input tree is analyzed and with the help of the register functions a computation plan is created. Note that this derivation only needs 12 steps.

$$
\begin{array}{rcl}
lift(\alpha, y_1) & \rightarrow & y_1 \\
lift(\sigma(x_1, x_2), y_1) & \rightarrow & \langle lift, \sigma, 1\rangle(x_1, x_2, y_1, (lift, extr)(x_1, y_1), \\
 & & \qquad extr(x_2, \gamma(y_1))) \\
\langle lift, \sigma, 1\rangle(x_1, x_2, y_1, comb_2(z_1, z_3), z_2) & \rightarrow & \sigma(z_1, \sigma(z_2, z_3)) \\
 & & \\
(lift, extr)(\alpha, y_1) & \rightarrow & comb_2(y_1, \alpha) \\
(lift, extr)(\sigma(x_1, x_2), y_1) & \rightarrow & \langle (lift, extr), \sigma, 1\rangle(x_1, x_2, y_1, (lift, extr)(x_1, y_1), \\
 & & \qquad extr(x_2, \gamma(y_1))) \\
\langle (lift, extr), \sigma, 1\rangle(x_1, x_2, y_1, comb_2(z_1, z_3), z_2) & \rightarrow & comb_2(\sigma(z_1, \sigma(z_3, z_2)), \sigma(lift(x_2, z_1), z_1)) \\
 & & \\
extr(\alpha, y_1) & \rightarrow & \alpha \\
extr(\sigma(x_1, x_2), y_1) & \rightarrow & \langle extr, \sigma, 1\rangle(x_1, x_2, y_1, lift(x_1, y_1)) \\
\langle extr, \sigma, 1\rangle(x_1, x_2, y_1, z_1) & \rightarrow & \sigma(lift(x_2, z_1), z_1)
\end{array}
$$

Figure 24: Rules of the macro tree transducer $rec\text{-}it(M_1)$ with register functions.

n	by $\stackrel{cbv}{\Rightarrow}_{M_1}$	by $\stackrel{cbv}{\Rightarrow}_{rec\text{-}it(M_1)}$
0	1	1
1	4	4
2	10	8
3	22	12
$\vdots$	$\vdots$	$\vdots$
k	$a(k-1) + 2 * a(k-2) + 4$	$4 * k$

Table 1: Comparison of a derivation by $\stackrel{cbv}{\Rightarrow}_{M_1}$ and $\stackrel{cbv}{\Rightarrow}_{rec\text{-}it(M_1)}$.

Consider a function call $lift(s, t)$ with $t \in T\langle\Delta_1\rangle$ and s is a tree over Σ_1 such that each second subtree is the nullary symbol α and n denotes the number of symbols σ occurring in s. The number of reduction steps $a(n)$ necessary to compute the normalform of the function call is given in Table 1. As it can be seen, there is a great gain in efficiency.

□

We make the following observation about the structure of the rules in $rec\text{-}it(R)$:

Observation 5.11 Let $M = (F, \Sigma, \Delta, R)$ be a macro tree transducer and $rec\text{-}it(M) = (rec\text{-}it(FS), \Sigma, rec\text{-}it(\Delta), rec\text{-}it(R))$. For every rule $l \rightarrow \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \psi_1, \ldots, \psi_r)$ in $rec\text{-}it(R)$ with $\langle f, \sigma, i\rangle \in reg(FS)^{(k+n+r)}$ it holds that, for every $j \in [r]$, ψ_j is either

- in Z or
- a ground function call. □

Now we can state the following important result.

Theorem 5.12 *Let $M = (F, \Sigma, \Delta, R)$ be a macro tree transducer. It holds that $rec\text{-}it(M)$ is at least as efficient as M.*

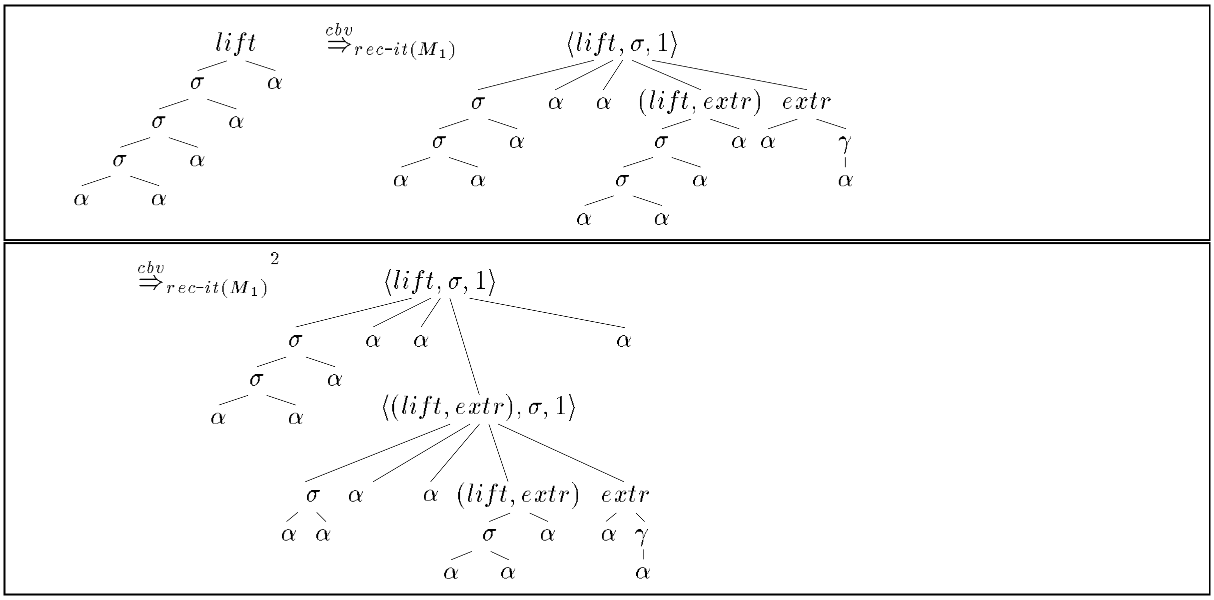


Figure 25: Derivation by $\stackrel{cbv}{\Rightarrow}_{rec\text{-}it(M_1)}$ (Part 1).

<u>Proof.</u> Since $\vdash_{red,M}$ is semantic-preserving, for every $\stackrel{cbv}{\Rightarrow}_M$-derivation of a syntax-directed expression ψ to a normalform ξ, there exists also a $\stackrel{cbv}{\Rightarrow}_{rec\text{-}it(M)}$-derivation of ψ to ξ. It has to be proved that the derivations by $\stackrel{cbv}{\Rightarrow}_{rec\text{-}it(M)}$ are at most as long as the derivations by $\stackrel{cbv}{\Rightarrow}_M$.

Since $\vdash_{red,M}$ is noetherian, there is an $L \geq 0$ such that

$$M \vdash_{red,M} N_1 \vdash_{red,M} \ldots \vdash_{red,M} N_L = rec\text{-}it(M)$$

M is a macro tree transducer, i.e., there are no register functions and register rules in the set of rules of M. By definition, $\vdash_{red,M} \; = \; \vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$ and the transformation relation $\vdash_{split,M}$ introduces a new register rule into the set of rules: the splitted rule of M and N_1 is replaced by two rules and only one of these rules can be changed by the transformation relations $\vdash_{share,M}$ and $\vdash_{tuple,M}$, namely the rule of which the root of the right-hand side is labeled by a register function. The relations $\vdash_{share,M}$ and $\vdash_{tuple,M}$ introduce no new rules with register functions: they only change an existing rule with a register function at the root of the right-hand side; $\vdash_{tuple,M}$ creates rules for tuple functions without register function calls.

It has been proved in Corollary 4.5 that, if $N \vdash_{split,M} N'$, then N is more efficient than N'. Every application of the splitted rule in a derivation by $\stackrel{cbv}{\Rightarrow}_N$ is simulated by the application of two rules in a derivation by $\stackrel{cbv}{\Rightarrow}_{N'}$. But, according to Definition 4.1, a rule is only splitted if its right-hand side contains at least two ground function calls with equal argument lists. This condition is exactly the one which is needed for the applicability of at least one of the transformation relations $\vdash_{share,M}$ and $\vdash_{tuple,M}$. The other conditions which are neccessary, are automatically fulfilled in this sequence of transformation steps.

In Corollary 4.15 it was proved that, for every macro tree transducer N with register functions, the macro tree transducer N' with register functions obtained by $N \vdash_{share,M} N'$, is more efficient than N in the following sense: Let $l \rightarrow \zeta$ be the shared rule of N and N'. Every

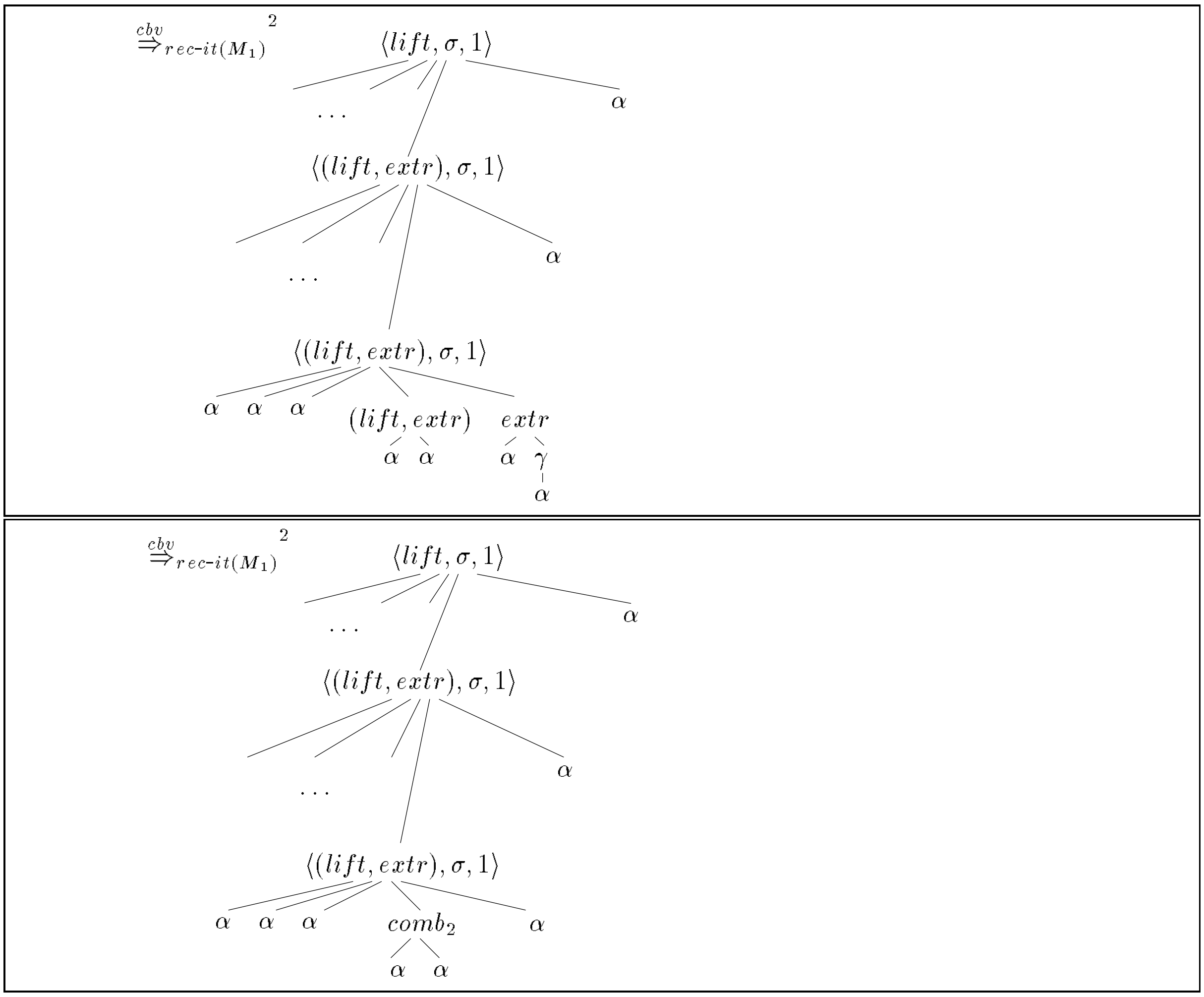


Figure 26: Derivation by $\stackrel{cbv}{\Rightarrow}_{rec\text{-}it(M_1)}$ (Part 2).

application of this rule during a derivation by $\stackrel{cbv}{\Rightarrow}_{N'}$ needs at least one step less than a derivation by $\stackrel{cbv}{\Rightarrow}_{N}$. Therewith the loss of efficiency by the relation $\vdash_{split,M}$ is compensated.

If no $\vdash_{share,M}$ step can be executed, then the loss of efficiency has to be compensated by the relation $\vdash_{tuple,M}$. There, different function calls with equal argument list are glued together and a new function call with the same argument list arises. The rules for the new function call are created by simply combining the right-hand sides of the rules for these functions of M. Hence, instead of computing, e.g., r different function calls which means r applications of rules, only one function call has to be computed with the same combined right-hand side. Altogether there are $r-1$ steps less if the right-hand side with register function which has arised by the application of $\vdash_{split,M}$, has to be evaluated.

The other steps $N_1 \vdash_{red,M} N_2$, ... can be justified as above: the splitting transformation relation is executed exactly once and the arising rule the right-hand side of which is a register function, is the only rule in which ground function calls in its right-hand side can be deleted or tupled.

Note that it cannot be stated that, for every $j \in [L]$, N_j is at least as efficient as N_{j-1} but in comparison with M, every N_j is at least as efficient as M. Altogether the statement holds with $b \leq a$. □

Theorem 5.13 *There are infinite many macro tree transducers M such that rec-it(M) is more efficient than M.*

<u>Proof.</u> We have seen in the proof of Theorem 5.12 that the loss of efficiency due to the $\vdash_{split,M}$ steps is compensated by the other transformation relations. We can formulate five cases in which the compensation leads even to a more efficient macro tree transducer with register functions. We state that *rec-it*(M) is sometimes better than M if, in the transformation from M into *rec-it*(M) by $\vdash_{red,M}$, there is at least one $\vdash_{red,M}$ step such that for its constituents $\vdash_{share,M}$ and $\vdash_{tuple,M}$ at least one of the following conditions holds:

1. the $\vdash_{red,M}$ step contains an application of $\vdash_{share,M}$ and an application of $\vdash_{tuple,M}$,
2. the $\vdash_{red,M}$ step contains at least two applications of $\vdash_{tuple,M}$,
3. in the $\vdash_{red,M}$ step at least one $\vdash_{share,M}$ step is applied which deletes at least two function calls,
4. the $\vdash_{red,M}$ step contains a $\vdash_{share,M}$ step which deletes a function call $f(\ldots)$ of which at least one rule has a function call in its right-hand side, or
5. in the $\vdash_{red,M}$ step at least one $\vdash_{tuple,M}$ step is applied which tuples at least three function calls.

The cases 1.-3. and 5. are clear from the proof of Theorem 5.12. Case 4. needs a more detailled explanation: Let us assume that the $\vdash_{red,M}$ step contains one application of $\vdash_{share,M}$ and no applications of $\vdash_{tuple,M}$ (otherwise case 1. would hold). Furthermore let us assume that $\vdash_{share,M}$ deletes only one function call $f(\ldots)$ from the argument list of a rule $l \to \zeta$. This means that during a $\stackrel{cbv}{\Rightarrow}$-derivation where the rule $l \to \zeta$ is applied, the evaluation of this function call (where the formal parameters are replaced by concrete instances of trees) is omitted. If this evaluation would need more than one derivation step, then the loss of efficiency due to splitting is overcompensated. In the case that there is an input symbol $\sigma \in \Sigma$ such that the (f, σ)-rule has a function call in its right-hand side, an input tree can be constructed such that the evaluation of the omitted function call needs more than one step. Hence also case 4. leads to a more efficient macro tree transducer with register functions.

One may ask which macro tree transducers fullfill the conditions above. Obviously, at least every macro tree transducer which has at least one rule of which the right-hand side contains three function calls with equal argument list. There are also infinitely many other macro tree transducers which fulfill the conditions above by simultaneous function calls as, e.g., the macro tree transducer version of the Fibonacci-function and, of course, our running example. □

Considering the transformation sequence from a macro tree transducer M to the recursive-iterative tree transducer *rec-it*(M), it can be exactly determined whether *rec-it*(M) is more efficient than M or not. The following decision algorithm accepts as input the macro tree transducer and it yields "yes", if *rec-it*(M) is more efficient than M, and "no", otherwise. The algorithm uses the instruction “stop” which terminates the execution of the algorithm at this spot. The terms “Case i” with $i \in [5]$ are concerned to the five cases which are enumerated in the proof of Theorem 5.13. The global while-statement performs a local $\vdash_{red,M}$-step. Note that the sharing transformation relation is embedded in an if-statement instead of a while-statement, because only local $\vdash_{red,M}$-steps are executed and such a local $\vdash_{red,M}$ step contains at most one application of $\vdash_{share,M}$.

Decision Algorithm:

Let *countsh*, *counttup*, N, and N' be program variables.

Input: macro tree transducer $M = (F, \Sigma, \Delta, R)$.

Output: either “yes, *rec-it*(M) is more efficient than M”
or “no, *rec-it*(M) is not more efficient than M”

Initialization: Let *countsh* $:= 0$, *counttup* $:= 0$, $N = M$.

while there is an N' such that $N \vdash_{split,M} N'$ **do**
  $N := N'$;
  if N is ready for sharing **then**
    $N \vdash_{share,M} N'$; *countsh* $:=$ *countsh* $+ 1$;
    Let $l \to \langle f, \sigma, i\rangle(\tilde{x}, \tilde{y}, \psi_1, \ldots, \psi_r)$ be the shared rule of N and N'.
    if $double(\psi_1, \ldots, \psi_r) = (l_1, \ldots, l_m)$ with $m > 1$ **then**
      return “yes” and stop; **fi** (* Case 3 *)
    if $double(\psi_1, \ldots, \psi_r) = (l_1)$ and l_1 is a function call of the form $f(\ldots)$ such that there is an (f, σ)-rule of which the right-hand side contains at least one function call **then**
      return “yes”and stop; **fi** (* Case 4 *)
    $N := N'$;
  fi
  while there is a rule $l \to \zeta$ in the set of rules of N
    such that ζ is ready for tupling the set S of ground function calls **do**
    $N \vdash_{tuple,M} N'$; *counttup* $:=$ *counttup* $+ 1$;
    Let $l \to \zeta$ be the in S tupled rule of N and N'
    if $card(S) > 2$, **then** return “yes” and stop; **fi** (* Case 5 *)
    $N := N'$;
    if *counttup* > 1 **then** return “yes” and stop; **fi** (* Case 2 *)
  od
  if *counttup* $+$ *countsh* > 1 **then** return “yes” and stop; **fi** (* Case 1 *)
  counttup $:= 0$; *countsh* $:= 0$;
od;
return “no”;

At the end of this section we present a macro tree transducer M such that *rec-it*(M) is not more efficient than M.

Example 5.14 Let us consider the macro tree transducer $M_2 = (F_2, \Sigma_2, \Delta_2, R_2)$ which is a modified version of M_1. The components of M_2 are given as follows:

- $F_2 = \{li^{(2)}, ex^{(2)}\}$,
- $\Sigma_2 = \{\sigma^{(2)}, \alpha^{(0)}\}$,
- $\Delta_2 = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$, and
- R_2 is the set of the rules which are shown in Figure 27,

Then $rec\text{-}it(M_2)$ is computed by the following transformation sequence:

$$M_2 \underbrace{\vdash_{split,M_2} \circ \vdash_{tuple,M_2}}_{\vdash_{red,M_2}} \circ \underbrace{\vdash_{split,M_2} \circ \vdash_{tuple,M_2}}_{\vdash_{red,M_2}} rec\text{-}it(M_2)$$

The rules of $rec\text{-}it(M_2)$ are shown in Figure 28. For every syntax-directed expression $\psi \in sdExp(F_2, \Sigma_2, \Delta_2)$ it holds that the derivations by $\stackrel{cbv}{\Rightarrow}_{rec\text{-}it(M_2)}$ and by $\stackrel{cbv}{\Rightarrow}_{M_2}$ are of equal length.

$$\begin{array}{rcl}
li(\alpha, y_1) & \to & y_1 \\
li(\sigma(x_1, x_2), y_1) & \to & \sigma(li(x_1, y_1), \sigma(ex(x_2, \gamma(y_1)), ex(x_1, y_1))) \\
ex(\alpha, y_1) & \to & \alpha \\
ex(\sigma(x_1, x_2), y_1) & \to & li(x_1, li(x_2, y_1))
\end{array}$$

Figure 27: Rules of the macro tree transducer M_2.

$$\begin{array}{rcl}
li(\alpha, y_1) & \to & y_1 \\
li(\sigma(x_1, x_2), y_1) & \to & \langle li, \sigma, 1\rangle(x_1, x_2, y_1, (li, ex)(x_1, y_1), \\
 & & \qquad ex(x_2, \gamma(y_1))) \\
\langle li, \sigma, 1\rangle(x_1, x_2, y_1, comb_2(z_1, z_3), z_2) & \to & \sigma(z_1, \sigma(z_2, z_3)) \\
 & & \\
(li, ex)(\alpha, y_1) & \to & comb_2(y_1, \alpha) \\
(li, ex)(\sigma(x_1, x_2), y_1) & \to & \langle (li, ex), \sigma, 1\rangle(x_1, x_2, y_1, (li, ex)(x_1, y_1), \\
 & & \qquad ex(x_2, \gamma(y_1)), li(x_2, y_1)) \\
\langle (li, ex), \sigma, 1\rangle(x_1, x_2, y_1, comb_2(z_1, z_3), z_2, z_4) & \to & comb_2(\sigma(z_1, \sigma(z_3, z_2)), li(x_1, z_4)) \\
 & & \\
ex(\alpha, y_1) & \to & \alpha \\
ex(\sigma(x_1, x_2), y_1) & \to & li(x_1, li(x_2, y_1))
\end{array}$$

Figure 28: Rules of the macro tree transducer $rec\text{-}it(M_2)$ with register functions.

6 Conclusion

We have defined particular classes of recursive program schemes, namely the class $\mathcal{M}$ of macro tree transducers and the class $\mathcal{N}$ of macro tree transducers with register functions which contains the first one. Our aim was to define a transformation from an arbitrary macro tree transducer into a macro tree transducer with register functions such that the resulting transducer is at least as efficient as the original transducer, and there exist macro tree transducers for which the transformation even yields a macro tree transducer with register functions which is more efficient than the original one. As measure of efficiency we have taken the number of call-by-value derivation steps necessary to compute the normalform of a syntax-directed expression built from the components of the underlying macro tree transducer.

For this purpose we have defined three transformation relations $\vdash_{split,M}$, $\vdash_{share,M}$, and $\vdash_{tuple,M}$ with the pleasant properties that they are semantic preserving, confluent, and noetherian. For two of these transformation relations, namely $\vdash_{split,M}$ and $\vdash_{share,M}$, exact statements about the efficiency were proven: an application of $\vdash_{split,M}$ decreases the efficiency whereas $\vdash_{share,M}$ increases the efficiency. For the tupling transformation relation no such statement could be proven.

With the help of these three transformation relations a transformation from a macro tree transducer M to a macro tree transducer $rec\text{-}it(M)$ with register functions was defined such that $rec\text{-}it(M)$ is at least as efficient as M. This transformation is the computation of the normalform of M with respect to the transformation relation $\vdash_{red,M}$ which is the composition $\vdash_{split,M} \circ \vdash^{\infty}_{share,M} \circ \vdash^{\infty}_{tuple,M}$. It was shown that, for every macro tree transducer, this normalform exists.

Finally a decision algorithm was given which is able to determine whether or not, $rec\text{-}it(M)$ is more efficient than M.

An important topic of this paper was to define the relations as precise and easy as possible to be able to proof all the statements we made. Hence, the transformation strategy itself is not very efficient and many optimations are possible to speed up the strategy and to optimize also $rec\text{-}it(M)$, e.g., the life time of the formal parameters which were saved in the argument list of register functions no matter if they are used or not, could be examined. Another obvious optimization would be to define an ordering on the simple functions to avoid that by the tupling transformation relation, e.g., the tuple functions (f, g) and (g, f) arise. But these optimizations have no effects to the efficiency of $rec\text{-}it(M)$ and only complicates the proof and the clearity of the presentation.

At the time being we are generalizing this strategy to more powerful program schemes.

References

[AE79] P. R. J. Asveld and J. Engelfriet. Extended linear macro grammars, iteration grammars, and register programs. *Acta Informatica*, 11:259–285, 1979.

[AS78] M.A. Auslander and H.R. Strong. Systematic recursion removal. *Communications of the ACM*, 21(2):127–134, 1978.

[BD77] R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.

[Bir77a] R.S. Bird. Improving programs by the introduction of recursion. *Communications of the ACM*, 20(11):856–863, 1977.

[Bir77b] R.S. Bird. Notes on recursion elimination. *Communications of the ACM*, 20(6):434–439, 1977.

[Bir80] R.S. Bird. Tabulation techniques for recursive programs. *ACM Computing Surveys*, 12(4):403–417, 1980.

[Boc76] G.V. Bochmann. Semantic evaluation from left to right. *Communications of the ACM*, 19:55–62, 1976.

[Boi92] E.A. Boiten. Improving recursive functions by inverting the order of evaluation. *Science of Computer Programming*, 18:139–179, 1992.

[BW82] F.L. Bauer and H. Wössner. *Algorithmic language and program development.* Springer, Berlin, 1982.

[CF82] B. Courcelle and P. Franchi-Zannettacci. Attribute grammars and recursive program schemes. *Theoretical Computer Science*, 17:163–191 and 253–257, 1982.

[CH95] W. Chin and M. Hagiya. A transformation method for dynamic-sized tabulation. *Acta Informatica*, 32:93–115, 1995.

[Chi93] W. Chin. Towards an automated tupling strategy. In *3rd ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, pages 119–132, Kopenhagen, 1993. ACM Press.

[CK93] W. Chin and S.C. Khoo. Tupling functions with multiple recursion parameters. *LNCS*, 724:124–140, 1993.

[Dij60] E.W. Dijkstra. Recursive programming. *Numerische Mathematik*, 2:312–318, 1960.

[DPSS77] J. Duske, R. Parchmann, M. Sedello, and J. Specht. IO-macrolanguages and attributed translations. *Information and Control*, 35:87–105, 1977.

[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.* New York, Academic Press, 1980.

[ES78] J. Engelfriet and E.M. Schmidt. IO and OI, Part I and II. *J. Comput. System Sci.*, 15, 16:328–353, 67–99, 1977, 1978.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. System Sci.*, 31(1):71–146, august 1985.

[Fea82] M.S. Feather. A system for assisting program transformation. *ACM Transactions on Programming Languages and Systems*, 4(1):1–20, 1982.

[Fis68] M.J. Fischer. *Grammars like macro-like productions.* PhD thesis, Harvard University, 1968 (see also: Proc. on 9th Symp. on SWAT, pp. 131–142, 1968).

[HP91] B. Hoffmann and D. Plump. Implementing term rewriting by jungle evaluation. *R.A.I.R.O. Informatique théorique et Applications*, 25(5):445–472, 1991.

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

[IK87a] K. Indermark and H. Klaeren. Compiling fibonacci-like recursion. *SIGPLAN Notices*, 22:101–107, 1987.

[IK87b] K. Indermark and H. Klaeren. Efficient implementation of structural recursion. In *LNCS 278, FCT 1987*, pages 204–217, 1987.

[IKV90] K. Indermark, H. Klaeren, and H. Vogler. Computational aspects of structural recursion. *invited lecture at STACS 1990 Rouen, France*, 1990.

[Kla88] H. Klaeren. *Ein algebraischer Ansatz zur Rekursionselimination.* RWTH Aachen, Fachgruppe Informatik, Ahornstr. 55, D-52056 Aachen, 1988. Habilitationsschrift.

[Klo92] J. W. Klop. Term rewriting systems. In *Handbook of Logic in Computer Science, Volume 2*, pages 1–116. Clarendon Press, Oxford, 1992.

[Knu68] D.E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2:125–145, 1968.

[Knu74] D.E. Knuth. Structured programming with go to statements. *ACM Computing Surveys*, 6(4):261–301, 1974.

[McC60] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Communications of the ACM*, 3:184–195, 1960.

[Par90] H.A. Partsch. *Specification and Transformation of Programs: a formal approach to software development.* Texts and monographs in computer science. Springer-Verlag, Berlin, 1990.

[PK82] R. Paige and S. Koenig. Finite differencing of computable expressions. *ACM Transactions on Programming Languages and Systems*, 4(3):402–454, 1982.

[PP86] H. Partsch and P. Pepper. Program transformations expressed by algebraic type manipulations. *Technique et Science Informatiques*, 03:197–212, 1986.

[PP93] A. Pettorossi and M. Proietti. Rules and strategies for program transformation. In B. Möller, H. Partsch, and S. Schumann, editors, *Formal Program Development, IFIPTC2/WG 2.1 State-of-the-Art Report*, LNCS 755, pages 263–304, 1993.

[Ric65] H.G. Rice. Recursion and iteration. *Communications of the ACM*, 8(2):114–115, 1965.

[San95a] D. Sands. Proving the correctness of recursion-based automatic program transformations. In P. Mosses, M. Nielsen, and M. Schwartzbach, editors, *Sixth International Joint Conference on Theory and Practice of Software Development (TAPSOFT)*, volume 915 of *Lecture Notes in Computer Science*, pages 681–695. Springer-Verlag, 1995.

[San95b] D. Sands. Total correctness by local improvement in the transformation of functional programs. Technical report, DIKU, University of Copenhagen, 1995. Extended Version of [Sands,POPL '95] (Submitted for Publication).

[SPvE93] M.R. Sleep, M.J. Plasmeijer, and M.C.J.D. van Eekelen, editors. *Term Graph Rewriting, Theory and Practice.* Wiley Professional Computing, 1993.

[Thi91a] P. Thiemann. Efficient implementation of structural recursive programs. Technical Report WSI-91-12, Wilhelm Schickard-Institut für Informatik, Universität Tübingen, 1991.

[Thi91b] P. Thiemann. Tabulating recursive functions without descent laws. Technical Report WSI-91-1, Wilhelm Schickard Institut für Informatik, Universität Tübingen, 1991.

[Vog90] H. Vogler. *Funktionale Programmierung mit primitiver Rekursion - formale Modelle zur Reduktionssemantik.* RWTH Aachen, Fachgruppe Informatik, Ahornstr. 55, D-52056 Aachen, 1990. Habilitationsschrift.