TECHNISCHE UNIVERSITÄT DRESDEN
FAKULTÄT INFORMATIK

Bakkalaureatsarbeit

Bachelor Thesis

von / by

Andreas Maletti

zum Thema / on the topic

„Efficiency analysis for the elimination of intermediate results in functional programs by compositions of attributed tree transducers"

Betreuender Hochschullehrer:	Prof. Dr.–Ing. habil. Heiko Vogler
Betreuer:	Dr. Armin Kühnemann
Institut:	Theoretische Informatik
Professur:	Grundlagen der Programmierung
Begonnen am:	02.04.2001
Eingereicht am:	31.07.2001

Erklärung

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

Bannewitz, den 31. Juli 2001

....
Andreas Maletti

Contents

Chapter 1

Introduction

Compositions of existing functions, i.e. the result of one function call is passed as a parameter to some other function, represent a clear, modular style of defining new functions. That is why function compositions are quite ubiquitous in functional programming. The functional programming language HASKELL (cf. [Tho99]), for example, even introduces a special operator (.) denoting function composition and also implements many standard library functions as a composition of others. On the one hand side this style provides developers with an efficient way of reusing existing functionality, but on the other hand side the composition might also introduce inefficiencies. These inefficiencies include construction and/or destruction of structured intermediate result objects as well as possibly inefficient computation (e.g. unnecessary tree traversals).

Several techniques are known to avoid the actual representation of intermediate results by transforming the function definitions [BD77, Mal89, GLP93], including composition results of tree transducer theory. In this thesis we consider a series of transformations proposed independently by [Küh98, CDPR99] based on attributed tree transducers [Fül81] applicable to restricted functional programs. Specifically we investigate the time-behaviour of the transformation result compared to the original sequential composition and we are aimed towards identifying classes of functional programs, where the transformation result outperforms the original sequential composition independent of the actual input.

Within this thesis we will use the unweighted number of reduction steps needed to compute the final result using a call-by-need evaluation strategy as our performance measure. This efficiency measure is very easy to determine and was also applied in [Höf99, Bor97], but in [Bor97] other performance measures were discussed as well.

To elucidate the involved inefficiencies, we consider the following program written in HASKELL.

```
revnot :: [Bool] -> [Bool]
revnot = reverse.(map not)
```

The function named `revnot` firstly negates every element of a list consisting of booleans and then reverses this list. This is stated in an obvious way by using a function composition of `(map not)` and `reverse`. So we effectively reused existing functionality, namely the functions `reverse`, `map` and `not`, making the program easier to understand, easier to validate and easy to maintain.

Contrasting these advantages are the inefficiencies experienced when computing with the above function definition. Firstly if we use lazy evaluation, then every constructor created (as output) by the function `(map not)` is immediately consumed by the function `reverse`, which results in unnecessary construction / destruction operations. Furthermore the list is iterated (traversed) twice, the first iteration creates the list containing the negated elements and the second iteration reverses the result list.

In order to apply our considered series of transformations, we necessarily need implementation details of the functions `reverse` and `(map not)` called `ex` in the following. Using the definitions of the functions, we (and a compiler as well) can determine, whether the transformations are applicable, and furthermore can also transform the definitions.

```
ex :: [Bool] -> [Bool]
ex []     = []
ex (x:xs) = not x : ex xs

reverse :: [Bool] -> [Bool]
reverse xs = let reverse' [] y     = y
                 reverse' (x:xs) y = reverse' xs (x:y)
             in
                 reverse' xs []
```

Both functions are primitive recursive and without going into detail at this stage, the proposed transformations are indeed applicable to this example yielding a function definition like (we modified the original result of the construction in order to improve readability):

```
revnot2 :: [Bool] -> [Bool]
revnot2 xs = let revnot2' [] y     = y
                 revnot2' (x:xs) y = revnot2' xs (not x:y)
             in
                 revnot2' xs []
```

It is easy to see that both functions (`revnot` and `revnot2`) are equivalent. The latter function avoids the construction/destruction of intermediate results completely, but the question remains which function performs better according to our performance measure (reduction steps). In this case it is obvious, that the latter function definition outperforms the original one, since it only iterates the list once. In general the problem cannot be decided that easily, so our aim is to find syntactic classes of functional programs for which the decision can be drawn. Given that we have such a decision procedure, a compiler could automatically opt for the more efficient function definition enabling the developer to stick to the readable, modular style of programming.

We omitted the exact requirements for the considered transformations to be applicable. First of all we only consider primitive-recursive functions, which we can model using macro tree transducers [EV85]. Furthermore we disallow multiple calls of the same function on the same subtree. This requirement is called weakly single-use and given a weakly single-use macro tree transducer, we can construct an equivalent attributed tree transducer (e.g. [Küh00a]), which defines the same function. Furthermore if the first macro tree transducer is linear in context parameters, then we can compose the gained attributed tree transducers using a composition result of [Gie88]. The result of the composition is an attributed tree transducer, which defines the same function as the sequential composition of the original function definitions. This attributed tree transducer can then be transformed back into a macro tree transducer using a transformation based on [FZ82] yielding a functional program.

The following illustration outlines the structure of this thesis. In addition it gives an overview of the series of transformations. The labels MAC, MAC_{wsu}, MAC_{su}, ATT and ATT_{su} shall denote the classes of all translations definable by arbitrary macro tree transducers, weakly single-use macro tree transducers, single-use macro tree transducers, arbitrary attributed tree transducers and single-use attributed tree transducers, respectively. Furthermore the symbol ; denotes the sequential composition (of tree transduction classes) and the arrows indicate transformations.

$$
\begin{tikzcd}
\mathrm{MAC}_{su} \arrow[dd] & ; & \mathrm{MAC}_{wsu} \arrow[dd] & \\
 & & & \text{Chapter 3} \\
\mathrm{ATT}_{su} \arrow[ddr] & ; & \mathrm{ATT} \arrow[ddl] & \\
 & & & \text{Chapter 4} \\
 & \mathrm{ATT} \arrow[dd] & & \\
 & & & \text{Chapter 5} \\
 & \mathrm{MAC} & &
\end{tikzcd}
$$

This thesis is divided into six chapters. Chapter two presents the basic notations as well as an introduction to tree transducer theory. In chapter three we firstly state the construction which transforms a weakly single-use macro tree transducer into an attributed tree transducer, and secondly we relate the efficiencies of the original macro tree transducer and the newly created attributed tree transducer. The next chapter introduces the construction needed to compose a single-use attributed tree transducer and an attributed tree transducer along with an efficiency relation between the original attributed tree transducers and the resulting attributed tree transducer. In chapter five we present the construction translating an attributed tree transducer back into a macro tree transducer as well as efficiency considerations. Finally in the last chapter we summarize the individual results gained in the previous chapters and outline future work in this field of study.

Chapter 2

Preliminaries

2.1 Mathematical Foundations

The set of natural numbers including 0 is designated by N in the following and $N_+ = N \setminus \{0\}$ shall denote the set of all positive integers. For arbitrary $n \in N$ let $[n] = \{\, x \mid x \leq n \,,\, x \in N_+ \,\} = \{1, \ldots, n\}$ be the subset of N_+, which contains all positive integers smaller than n along with n itself. Therefore $[0] = \emptyset$, where $\emptyset$ represents the empty set.

The number of elements of a finite set S is denoted $|S| \in N$ and there is only one set $S = \emptyset$ with $|S| = 0$. We will use the subset relation $S_1 \subseteq S_2$ between sets S_1 and S_2 to state that every element of S_1 is also an element of S_2 and the strict subset relation $S_1 \subset S_2$ to represent the fact, that $S_1 \subseteq S_2$ and $S_1 \neq S_2$. Furthermore we define $\mathcal{P}(S)$ to be the powerset of an arbitrary set S (i.e. $\mathcal{P}(S) = \{\, X \mid X \subseteq S \,\}$).

Let $n \in N$ and $S_1, \ldots, S_n$ be arbitrary non-empty sets, then $S_1 \times \cdots \times S_n = \{\, (x_1, \ldots, x_n) \mid x_1 \in S_1, \ldots, x_n \in S_n \,\}$ represents the n-fold cartesian product. Elements of such products are called n-tuples and pairs, quintuples in particular are 2-tuples, 5-tuples, respectively. We use the notation S^n with $n \in N$, instead of the n-fold cartesian product of some non-empty set S with itself. Therefore $S^0 = \emptyset$ and $S^1 = S$.

A binary relation $\rightarrow \subseteq S^2$ over a set S is a subset of S^2. We often prefer the infix notation $a \rightarrow b$ contrasting the prefix notation $\rightarrow (a, b)$. Let $\rightarrow^0$ be the identity relation over S, then $\rightarrow^{n+1} = \{\, (a, c) \mid a \in S \,,\, c \in S \,,\, (\exists b \in S)\, [a \rightarrow b \wedge b \rightarrow^n c] \,\}$ is the $(n+1)$-fold composition of relation $\rightarrow$ for arbitrary $n \in N$. Furthermore the transitive, respectively the reflexive and transitive, closure of $\rightarrow$ is denoted $\rightarrow^+ = \bigcup_{n \in N_+} (\rightarrow^n)$, respectively $\rightarrow^* = \bigcup_{n \in N} (\rightarrow^n)$.

Let $\rightarrow$ be a binary relation over a set S. A sequence of the form $t_0 \rightarrow t_1 \rightarrow \ldots \rightarrow t_n$ with $n \in N$ and $t_0, \ldots, t_n \in S$ is called reduction, if for every $i \in [n]$ it holds that $t_{i-1} \rightarrow t_i$. The (derivation) relation $\rightarrow$ is called terminating, if and only if there does not exist an infinite reduction, and $\rightarrow$ is called confluent, if and only if whenever $t \rightarrow^* t_1$ and $t \rightarrow^* t_2$ with $t, t_1, t_2 \in S$, then there also exists a $t' \in S$ with $t_1 \rightarrow^* t'$ and $t_2 \rightarrow^* t'$. Let $\rightarrow$ be terminating and confluent, then for every element $t \in S$ there exists a unique normal form $t' \in S$ (i.e. $t \rightarrow^* t'$ and there does not exist a $t'' \in S$, such that $t' \rightarrow t''$), which we will denote $nf_{\rightarrow}(t)$.

We will also make use of the short-hands X_n and Y_n for every $n \in N$ to represent the sets $X_n = \{\, x_i \mid i \in [n] \,\} = \{x_1, \ldots, x_n\}$ and $Y_n = \{\, y_i \mid i \in [n] \,\}$, so consequently $|X_n| = n$. In case S is a non-empty, finite set, then $\max_R(S) \in S$, respectively $\min_R(S) \in S$, shall return the maximal, respectively minimal, element of S according to the total ordering relation (reflexive, transitive, antisymmetric, linear) R over S, i.e. for every $x \in S$: $R\big(x, \max_R(S)\big)$, respectively for every $x \in S$: $R\big(\min_R(S), x\big)$. Especially in case of $S \subset N$ being a non-empty, finite subset of the natural numbers, then we will use $\max S = \max_{\leq}(S)$ and $\min S = \min_{\leq}(S)$.

2.2 Basics of formal language theory

A finite set of *symbols* is called *alphabet*. Let Σ be an arbitrary alphabet, then a finite sequence $s_1 \dots s_m$ of symbols, where $m \in N_+$ and $s_1, \dots, s_m \in \Sigma$, is a *word* over Σ. The set of all such words over an alphabet Σ is designated Σ^+ and the associative binary operator $\cdot$ denoting concatenation of words shall be defined over elements of $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$ using the neutral element $\varepsilon \notin \Sigma$. To simplify notation we omit unnecessary parentheses and we will usually write $w_1 w_2$ instead of $w_1 \cdot w_2$. The n-fold concatenation of some word $w \in \Sigma^*$ shall be denoted by w^n for $n \in N$, such that consequently $w^0 = \varepsilon$.

The associative binary set concatenation operator $\circ$ performs concatenation over non-empty alphabets of words as follows $L_1 \circ L_2 = \{\, w_1 w_2 | w_1 \in L_1, w_2 \in L_2 \,\}$, where the sets $L_1, L_2 \subseteq \Sigma^*$ are subsets of Σ^*. Let Σ^n denote the n-fold concatenation of Σ with itself for every $n \in N$, where $\Sigma^0 = \{\varepsilon\}$ is the set containing only the neutral element and consequently $\Sigma^* = \bigcup_{n \in N} (\Sigma^n)$ and $\Sigma^+ = \bigcup_{n \in N_+} (\Sigma^n)$.

A *language* $\mathcal{L} \subseteq \Sigma^*$ over some alphabet Σ is an arbitrary subset of all *words* (including ε) over Σ. In particular the neutral element ε is called *empty word*. Let $|w| \in N$ denote the length of a word w (the underlying alphabet must be deducable from the context), such that $|\varepsilon| = 0$.

2.2.1 Definition (ranked alphabet)
Let Γ be an alphabet and in addition let rank $: \Gamma \to N$ *be a total function, which associates a rank (arity) to a given symbol of* Γ. *A set*

$$\Sigma = \{\, \sigma^{(\mathrm{rank}(\sigma))} \mid \sigma \in \Gamma \,\}$$

is then called ranked alphabet. *For a ranked alphabet Σ we will use the notation* rank_Σ *to designate the corresponding* rank*-function and we define the restriction $\Sigma^{(n)}$ of a ranked alphabet Σ to n-ary symbols (symbols with rank n) for every $n \in N$ to be*

$$\Sigma^{(n)} = \{\, \sigma \mid \sigma^{(n)} \in \Sigma \,\}.$$

Given a ranked alphabet Σ, the underlying alphabet will be denoted $\underline{\Sigma} = \bigcup_{n \in N} \Sigma^{(n)}$.

We define the set $T_\Sigma(Y) \subseteq (\underline{\Sigma} \cup Y \cup \{(,)\})^*$ of all *terms* build of constructors (symbols) of the ranked alphabet Σ possibly indexed with elements of the set Y to be the smallest set T such that

- $Y \subseteq T$ and
- for every $n \in N$, $\sigma \in \Sigma^{(n)}$ and $t_1, \dots, t_n \in T$ also $(\sigma\, t_1 \dots t_n) \in T$, where the term t_i, $i \in [n]$ is called i-th successor of $(\sigma\, t_1 \dots t_n)$.

We also introduce the shorthand $T_\Sigma = T_\Sigma(\emptyset)$ and for the sake of simplicity we will write α instead (α), if $\alpha \in \Sigma^{(0)}$ and we will omit unnecessary parentheses, especially in case of monadic alphabets Σ ($\Sigma = \Sigma^{(0)} \cup \Sigma^{(1)}$).

Let $T_\Sigma(Y)$ be a set of terms (deducible from the context) as defined above, an element $t \in T_\Sigma(Y)$ and in addition a symbol $\alpha \in (\underline{\Sigma} \cup Y)$, then $|t|_\alpha$ shall denote the number of occurrences of α in t. So for every $\beta \in (\underline{\Sigma} \cup Y)$ and $t_1, \ldots, t_n \in T_\Sigma(Y)$

$$|\beta\, t_1 \ldots t_n|_\alpha = \sum_{i \in [n]} |t_i|_\alpha + \begin{cases} 1 & \text{, if } \alpha = \beta \\ 0 & \text{, else} \end{cases} \quad \text{, where } n = \begin{cases} 0 & \text{, if } \beta \in Y \\ \mathrm{rank}_\Sigma(\beta) & \text{, otherwise} \end{cases}.$$

The set of all paths *Paths* is defined as $Paths = \mathcal{P}(\{\varepsilon\} \cup N \circ (\{.\} \circ N)^*)$. Furthermore we define the following functions and notations:

1. $\mathrm{paths}(t)$ computes the set of all paths of a tree $t \in T_\Sigma(Y)$.

$$\begin{array}{lcl} \mathrm{paths} & : & T_\Sigma(Y) \rightarrow Paths \\ \mathrm{paths}' & : & T_\Sigma(Y) \rightarrow (Paths \setminus \{\varepsilon\}) \end{array}$$

   For every $n \in N$, $\sigma \in \Sigma^{(n)}$, $y \in Y$ and $t_1, \ldots, t_n \in T_\Sigma(Y)$

$$\begin{array}{lcl} \mathrm{paths}'(y) & = & \emptyset \\ \mathrm{paths}'(\sigma\, t_1 \ldots t_n) & = & [n] \cup \big(\, [n] \circ \{.\} \circ \mathrm{paths}'(t_j)\, \big) \end{array}$$

   and finally $\mathrm{paths}(t) = \mathrm{paths}'(t) \cup \{\varepsilon\}$ for every $t \in T_\Sigma(Y)$.

2. $\mathrm{label}_t(p)$ shall compute the label at the node reachable by path $p \in \mathrm{paths}(t)$ in the tree $t \in T_\Sigma(Y)$.

$$\mathrm{label}_t : \quad \mathrm{paths}(t) \rightarrow (\underline{\Sigma} \cup Y)$$

   For every $n \in N$, $\sigma \in \Sigma^{(n)}$, $y \in Y$ and $t_1, \ldots, t_n \in T_\Sigma(Y)$

$$\begin{array}{lcl} \mathrm{label}_{(\sigma\, t_1 \ldots t_n)}(p) & = & \begin{cases} \sigma & \text{, if } p = \varepsilon \\ \mathrm{label}_{t_p}(\varepsilon) & \text{, if } p \in [n] \\ \mathrm{label}_{t_j}(p') & \text{, if } p = j.p' \text{ with } j \in [n] \end{cases} \\ \mathrm{label}_y(\varepsilon) & = & y \end{array}$$

3. $t[x/y]$ with $t, x, y \in T_\Sigma(Y)$ denotes the substitution of each and every occurrence of x by y in t.

Let F and Π be sets, then we define the set $Fun_{F,\Pi}$ of all attribute instances over attributes of F and paths of Π to be

$$Fun_{F,\Pi} = \{\, \langle f, \pi \rangle \mid f \in F\, ,\, \pi \in \Pi\, \}.$$

2.3 Macro tree transducers

Within this section we will define macro tree transducers [EV85], which we use to represent functional programs. Therefore we firstly define the apparatus macro tree transducer, then associate formal semantics and finally introduce syntactic properties. The first two definitions establishing the set of right hand sides and the components of a macro tree transducer are taken out of [Voi01].

2.3.1 Definition (right hand sides)
Let $k, n \in N$ be natural numbers and F, Δ be ranked alphabets. Then the set $RHS_{\Delta,F,k,n}$ of right hand sides *over function symbols of F and output symbols of Δ indexed with recursion variables $x_1, \ldots, x_k$ and context parameters $y_1, \ldots, y_n$ is defined to be the smallest set $RHS \subseteq T_{\Delta \cup F}(Y_n \cup X_k)$ such that*

- $Y_n \subseteq RHS$,
- *for every $r \in N$, $\delta \in \Delta^{(r)}$ and $\varrho_1, \ldots, \varrho_r \in RHS$ it holds that $(\delta\, \varrho_1 \ldots \varrho_r) \in RHS$ and*
- *for every $r \in N$, $f \in F^{(r+1)}$, $x \in X_k$ and $\varrho_1, \ldots, \varrho_r \in RHS$ it holds that $(f\, x\, \varrho_1 \ldots \varrho_r) \in RHS$.*

□

Having defined the shape of the right hand sides, we can now give a compact definition of the apparatus macro tree transducer.

2.3.2 Definition (macro tree transducer)
A macro tree transducer M is a quintuple $M = (\Sigma, \Delta, F, e, R)$, where

- *Σ is a ranked alphabet of* input symbols *with $\Sigma^{(0)} \neq \emptyset$,*
- *Δ is a ranked alphabet of* output symbols *with $\Delta^{(0)} \neq \emptyset$,*
- *F is a ranked alphabet of* function symbols *with $\underline{F} \cap (\underline{\Sigma} \cup \underline{\Delta}) = \emptyset$ and $F^{(0)} = \emptyset$,*
- *$e \in RHS_{\Delta,F,1,0}$ is an* initial expression *and*
- *R is a non-empty, finite set of* rules *of the form $f\,(\sigma\, x_1 \ldots x_k)\, y_1 \ldots y_n = rhs_{M,\sigma,f}$ with $k, n \in N$, $\sigma \in \Sigma^{(k)}$, $f \in F^{(n)}$ and $rhs_{M,\sigma,f} \in RHS_{\Delta,F,k,n}$, such that there is exactly one rule for every combination of f and σ.*

□

We will try to illustrate the above definitions by stating an example.

2.3.3 Example (reverse for lists)
We define the macro tree transducer $M_{rev} = (\Sigma, \Sigma, \{rev^{(2)}\}, (rev\, x_1\, N), R)$ with $\Sigma = \{A^{(1)}, B^{(1)}, N^{(0)}\}$ and

$$R = \{ \begin{array}{lcll} rev\,(A\, x_1)\, y_1 & = & rev\, x_1\, (A\, y_1) & , \\ rev\,(B\, x_1)\, y_1 & = & rev\, x_1\, (B\, y_1) & , \\ rev\, N\, y_1 & = & y_1 & \}. \end{array}$$

□

Intuitively the above stated macro tree transducer reverses an input list consisting of As, Bs and the end delimiter N by accumulating the list in the context parameter y_1. To associate formal semantics to our macro tree transducers, we will define the set of sentential forms and afterwards a derivation relation.

2.3.4 Definition (sentential forms)
Let Σ, Δ and F be ranked alphabets. The set $SF_{\Sigma,\Delta,F}$ of sentential forms *over Σ, Δ and F is defined to be the smallest subset $SF \subseteq T_{\Sigma\cup\Delta\cup F}$, such that*

- *for every $k \in N$, $\delta \in \Delta^{(k)}$ and $t_1, \ldots, t_k \in SF$ it holds that $(\delta\, t_1 \ldots t_k) \in SF$ and*
- *for every $n \in N$, $f \in F^{(n+1)}$, $e \in T_\Sigma$ and $t_1, \ldots, t_n \in SF$ it holds that $(f\ e\ t_1 \ldots t_n) \in SF$.*

□

The following definition of a derivation relation for macro tree transducers is taken out of [Höf99], where it was also shown that this derivation relation represents a call-by-name (leftmost-outermost) derivation relation.

2.3.5 Definition (call-by-name derivation relation)
Let $M = (\Sigma, \Delta, F, e, R)$ be a macro tree transducer. Then the derivation relation *of M denoted $\Rightarrow_M$ is the smallest binary relation $\Rightarrow$ over $SF_{\Sigma,\Delta,F}$, such that*

- *for every $k, n \in N$, $\sigma \in \Sigma^{(k)}$, $f \in F^{(n+1)}$, $e_1, \ldots, e_k \in T_\Sigma$ and $t_1, \ldots, t_n \in SF_{\Sigma,\Delta,F}$ with $\big(f\,(\sigma\, x_1 \ldots x_k)\, y_1 \ldots y_n = \varrho\big) \in R$ holds*

$$\big(f\,(\sigma\, e_1 \ldots e_k)\, t_1 \ldots t_n\big) \;\Rightarrow\; \varrho[x_1/e_1, \ldots, x_k/e_k\,,\, y_1/t_1, \ldots, y_n/t_n]$$

- *and for every $k \in N$, $\sigma \in \Sigma^{(k)}$, $i \in [k]$, $t_1, \ldots, t_{i-1} \in T_\Delta$ and $t'_i, t_i, \ldots, t_k \in SF_{\Sigma,\Delta,F}$ with $t_i \Rightarrow t'_i$ holds*

$$(\sigma\, t_1 \ldots t_i \ldots t_k) \;\Rightarrow\; (\sigma\, t_1 \ldots t_{i-1}\, t'_i\, t_{i+1} \ldots t_k).$$

□

In [EV85] it was shown that the non-deterministic derivation relation for macro tree transducers is terminating and confluent, consequently the derivation relation $\Rightarrow_M$ is also terminating and confluent. The following example shall illustrate the reduction process with the derivation relation defined above.

2.3.6 Example (reduction using M_{rev})
In this example we will reduce the input tree $t = A(B(B\,N)) = ABBN$ using the macro tree transducer M_{rev} defined in Example 2.3.3.

$$\begin{array}{rl} & rev\,(ABBN)\ N \\ \Rightarrow_{M_{rev}} & rev\,(BBN)\,(AN) \\ \Rightarrow_{M_{rev}} & rev\,(BN)\,(BAN) \\ \Rightarrow_{M_{rev}} & rev\,N\,(BBAN) \\ \Rightarrow_{M_{rev}} & BBAN \end{array}$$

□

The sentential form $(BBAN) \in T_\Delta$ is the normal form. This result represents the reversed input list and we observe that the reduction process involved four reduction steps in total. Having defined the derivation relation, we can finally associate semantics to macro tree transducers. A macro tree transducer defines a translation (a total mapping) of an input tree to an output tree.

2.3.7 Definition (induced translation)
Let $M = (\Sigma, \Delta, F, e, R)$ be a macro tree transducer. The translation induced by M, denoted $\tau(M) \subseteq T_\Sigma \times T_\Delta$, is defined to be

$$\tau(M) = \{\, (t_\Sigma, t_\Delta) \mid t_\Sigma \in T_\Sigma\,,\, t_\Delta \in T_\Delta\,,\, t_\Delta = nf_{\Rightarrow_M}(e\,[x_1/t_\Sigma])\,\}.$$

□

Occasionally we will interpret this relation as a mapping, e.g. $\tau(M)(t_\Sigma) = t_\Delta$.

2.3.8 Example (induced translation of M_{rev})
Let M_{rev} be the macro tree transducer which was defined in Example 2.3.3. The translation induced by M_{rev} is

$$\tau(M_{rev}) = \{\, (wN, w^R N) \mid wN \in T_\Sigma\,\},$$

where w^R denotes the reversed word (i.e. $(ABB)^R = BBA$). □

In the following macro tree transducers are considered equivalent, if and only if their induced translations are equal. In addition we introduce the class MAC of all translations computable by macro tree transducers. As preceeded our series of transformations is only applicable to certain subclasses of macro tree transducers. These subclasses as well as some other important syntactic restrictions are defined below.

2.3.9 Definition (weakly single-use macro tree transducer)
Let $M = (\Sigma, \Delta, F, e, R)$ be a macro tree transducer. If and only if for every input symbol $\sigma \in \underline{\Sigma}$, $i \in [\mathrm{rank}_\Sigma(\sigma)]$ and $f \in \underline{F}$, a call like $(f\,x_i \ldots)$ occurs in a right hand side of at most one rule at σ, and there and additionally also in e only one such call, then the macro tree transducer M is called weakly single-use*. The class of all translations computable by weakly single-use macro tree transducers will be denoted* MAC_{wsu}. □

2.3.10 Definition (at least single-use macro tree transducer)
Let $M = (\Sigma, \Delta, F, e, R)$ be a macro tree transducer. If and only if for every input symbol $\sigma \in \underline{\Sigma}$, $i \in [\mathrm{rank}_\Sigma(\sigma)]$ and $f \in \underline{F}$, a call like $(f\,x_i \ldots)$ occurs in a right hand side of at least one rule at σ and a call like $(f\,x_1 \ldots)$ occurs in e at least once as well, then the macro tree transducer M is called at least single-use*. The class of all translations computable by at least single-use macro tree transducers will be denoted* MAC_{lsu}. □

2.3.11 Definition (syntactic single-use macro tree transducer)
Let $M = (\Sigma, \Delta, F, e, R)$ be a macro tree transducer. M is called syntactic single-use, if and only if M is both weakly single-use and at least single-use. We identify the class of all translations computable by syntactic single-use macro tree transducers by MAC_{ssu}. □

The above definitions intuitively state that all suitable calls like $(f\,x_i \ldots)$ appear at most once in the case of weakly single-use, or at least once in the case of at least single-use in all right hand sides at an input symbol σ. The same restriction is also imposed on the initial expression. The definition of syntactic single-use, adapted from [Gie88], is the combination of both at least single-use and weakly single-use.

2.3.12 Definition (non-copying, preserving, non-deleting)
Let $M = (\Sigma, \Delta, F, e, R)$ be a macro tree transducer. If and only if for every input symbol $\sigma \in \underline{\Sigma}$, every function symbol $f \in \underline{F}$ and every context parameter y_i with $i \in [\mathrm{rank}_F(f)]$

$$|rhs_{M,\sigma,f}|_{y_i} \begin{cases} \leq 1 \\ = 1 \\ \geq 1 \end{cases} \text{then } M \text{ is called} \begin{cases} \textit{non-copying (in context parameters)} \\ \textit{preserving in context parameters} \\ \textit{non-deleting in context parameters} \end{cases} .$$

The classes of all translations computable by non-copying, preserving in context parameters and non-deleting in context parameters macro tree transducers are denoted MAC_{nc}, MAC_{prc} and MAC_{ndc}, respectively. □

Similarly this represents the fact that every context parameter occurs at most once (for non-copying) or at least once (for non-deleting in context parameters) in every right hand side. Taking the properties weakly single-use and non-copying together, we gain the property single-use.

2.3.13 Definition (single-use macro tree transducer)
A macro tree transducer M is called single-use*, if and only if M is both non-copying and weakly single-use. Furthermore we identify the class of all translations computable by single-use macro tree transducers by MAC_{su}.* □

If we reconsider our running example, it is easy to see that M_{rev} is in fact a syntactic single-use and preserving in context parameters macro tree transducer, hence it is also single-use.

Finally we will also introduce the concept of top-down tree transducers, which are macro tree transducers without context parameters. Consequently all function symbols are unary. We will use TOP, TOP_{wsu}, TOP_{ssu} and TOP_{lsu} to identify the corresponding classes of translations.

2.4 Attributed tree transducers

In this section we will define attributed tree transducers, which were introduced in [Fül81]. The definitions are mostly taken out of [Küh00a], but they are slightly adapted to use established notations of the field of attribute grammars (cf. [KV97]). We firstly define inner an outer attribute occurrences.

2.4.1 Definition (Inner and outer attribute occurrences)
We define the functions $in_{S,I}$ and $out_{S,I}$ both typed $N \to \mathcal{P}(Fun_{S\cup I,N})$, that return the alphabets of inner *and* outer attribute occurrences *according to alphabets S and I of synthesized and inherited attributes as*

$$\begin{aligned} \mathrm{in}_{S,I}(k) &= Fun_{S,\{0\}} \cup Fun_{I,[k]} \\ \mathrm{out}_{S,I}(k) &= Fun_{I,\{0\}} \cup Fun_{S,[k]}. \end{aligned}$$

□

With the above definition we can now define the syntax of attributed tree transducers. We will also define some shorthands for attributed tree transducers. It will always be clear from the context to which attributed tree transducer these shorthands refer. Occasionally we will also decorate these shorthands with subscripts to express that they refer to the attributed tree transducer with the same subscript.

2.4.2 Definition (attributed tree transducer)
A attributed tree transducer M is a 7-tuple $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$, where

- Σ *is a ranked alphabet of* input symbols *with* $\Sigma^{(0)} \neq \emptyset$,
- Δ *is a ranked alphabet of* output symbols *with* $\Delta^{(0)} \neq \emptyset$,
- S *is an alphabet of* synthesized attributes,
- I *is an alphabet of* inherited attributes,
- $\widehat{s} \notin S$ *is a distinguished synthesized attribute called* initial synthesized attribute,
- $\widehat{\sigma} \notin (\underline{\Sigma} \cup \underline{\Delta})$ *is a distinguished unary input symbol called* root symbol,
- $\Sigma_+ = \Sigma \cup \{\widehat{\sigma}^{(1)}\}\,, \quad S_+ = S \cup \{\widehat{s}\}\,, \quad F = S \cup I\,, \quad F_+ = S_+ \cup I,$
- *we define the functions in_M and out_M both typed $\underline{\Sigma}_+ \to \mathcal{P}(Fun_{F_+,N})$ to be*

$$\mathrm{in}_M(\sigma) = \begin{cases} \mathrm{in}_{\{\widehat{s}\},I}(1) & \text{, } \textit{if } \sigma = \widehat{\sigma} \\ \mathrm{in}_{S,I}(n) & \text{, } \textit{otherwise} \end{cases} \qquad \mathrm{out}_M(\sigma) = \begin{cases} \mathrm{out}_{S,\emptyset}(1) & \text{, } \textit{if } \sigma = \widehat{\sigma} \\ \mathrm{out}_{S,I}(n) & \text{, } \textit{otherwise} \end{cases}$$

- $R = (rhs_{M,\sigma} \mid \sigma \in \underline{\Sigma}_+)$ *is a non-empty, finite family of* right hand side functions, *where for every $\sigma \in \underline{\Sigma}_+$ the function $rhs_{M,\sigma}$ is a total function of the type*

$$\mathrm{in}_M(\sigma) \;\to\; T_\Delta\big(\mathrm{out}_M(\sigma)\big).$$

□

To counteract misunderstandings we assume, that I , S_+ , $\Sigma_+ \cup \Delta$ and N are pairwise disjoint. The following example shall illustrate, how an attributed tree transducer is defined.

2.4.3 Example (reverse for lists)
We define the attributed tree transducer M_{a-rev} to be

$$M_{a-rev} = (\Sigma, \Sigma, \{rev\}, \{i\}, \widehat{s}, \widehat{\sigma}, \{rhs_{M,\widehat{\sigma}}, rhs_{M,A}, rhs_{M,B}, rhs_{M,N}\})$$

with $\Sigma = \{A^{(1)}, B^{(1)}, N^{(0)}\}$ and

$$\begin{array}{lcl|lcl} rhs_{M,\widehat{\sigma}}(\langle \widehat{s}, 0\rangle) & = & \langle rev, 1\rangle & rhs_{M,A}(\langle rev, 0\rangle) & = & \langle rev, 1\rangle \\ rhs_{M,\widehat{\sigma}}(\langle i, 1\rangle) & = & N & rhs_{M,A}(\langle i, 1\rangle) & = & A\,\langle i, 0\rangle \\ rhs_{M,N}(\langle rev, 0\rangle) & = & \langle i, 0\rangle & rhs_{M,B}(\langle rev, 0\rangle) & = & \langle rev, 1\rangle \\ & & & rhs_{M,B}(\langle i, 1\rangle) & = & B\,\langle i, 0\rangle. \end{array}$$

□

Since attributed tree transducers are just special attribute grammars and attribute grammars can be circular (cf. [KV97]), attributed tree transducers can be circular as well. With the help of the following definitions we will exclude circular attributed tree transducers.

2.4.4 Definition (direct dependency relation)
Let M be an attributed tree transducer and furthermore let $\sigma \in \underline{\Sigma}_+$. The direct dependency relation $\rightarrow_{M,\sigma} \subseteq \mathrm{in}_M(\sigma) \times \mathrm{out}_M(\sigma)$ is the smallest relation such that for every $\xi \in \mathrm{in}_M(\sigma)$ and every $\chi \in \mathrm{out}_M(\sigma)$ the relation $\xi \rightarrow_{M,\sigma} \chi$ holds, if $|rhs_{M,\sigma}(\xi)|_\chi \geq 1$. This relation is extended to attribute instances in a tree $t \in T_\Sigma$ in the obvious way yielding the direct dependency relation $\rightarrow_{M,t}$ of the tree t. □

2.4.5 Definition (circular attributed tree transducer)
An attributed tree transducer $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ is called circular*, if and only if there exists an input tree $t = (\widehat{\sigma}\, t')$ with $t' \in T_\Sigma$ and an attribute instance $\chi \in Fun_{F,\mathrm{paths}(t)}$ such that $\chi \rightarrow^+_{M,t} \chi$.* □

In the following we will always assume non-circular attributed tree transducers, when speaking about attributed tree transducers. Again we will associate semantics to our (non-circular) attributed tree transducers by stating a derivation relation.

2.4.6 Definition (path relabeling)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be an attributed tree transducer and furthermore let $m = \max\{\, \mathrm{rank}_\Sigma(\sigma) \mid \sigma \in \underline{\Sigma}\,\}$. For every $p \in Paths$ the function $\mathrm{relabel}_{M,p}$ *typed*

$$\mathrm{relabel}_{M,p} \;:\; T_\Delta\big(\mathrm{out}_{S,I}(m)\big) \;\rightarrow\; T_\Delta\big(Fun_{F,Paths}\big)$$

shall return the sentential form corresponding to a given right hand side. Therefore the function replaces all attribute occurrences by attribute instances using path p. For every $j, k \in N$, $\sigma \in \Sigma^{(k)}$, $f \in F$ and $t_1, \ldots, t_n \in T_\Delta\big(\mathrm{out}_{S,I}(m)\big)$ we define $\mathrm{relabel}_{M,p}$ *to be*

$$\begin{aligned}
\mathrm{relabel}_{M,p}\big(\sigma\, t_1 \ldots t_n\big) &= \sigma\left(\mathrm{relabel}_{M,p}(t_1)\ \ldots\ \mathrm{relabel}_{M,p}(t_n)\right)\\
\mathrm{relabel}_{M,p}\big(\langle f,j\rangle\big) &= \begin{cases} \langle f,p\rangle & \text{if } j = 0\\ \langle f,j\rangle & \text{if } p = \varepsilon\,,\ j \neq 0\\ \langle f,p.j\rangle & \text{if } p \neq \varepsilon\,,\ j \neq 0\end{cases}
\end{aligned}$$

□

2.4.7 Definition (call-by-name derivation relation)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be an attributed tree transducer and in addition let $t \in T_{\Sigma_+}$ be an input tree. The derivation relation of M according to the input tree t is the following binary relation $\Rightarrow_{M,t} \subseteq T^2_\Delta(Fun_{F_+,\mathrm{paths}(t)})$. For every $t_1, t_2 \in T_\Delta(Fun_{F_+,\mathrm{paths}(t)})$ the sentential form t_1 can be reduced *to t_2, i.e. $t_1 \Rightarrow_{M,t} t_2$, if and only if there exists $j \in N$, $\sigma \in \underline{\Sigma}_+$, $f \in F_+$, $p' \in \mathrm{paths}(t)$ and $t' \in T_{\Delta\cup\{u^{(0)}\}}(Fun_{F_+,\mathrm{paths(t)}})$ with $u \notin (F \cup \underline{\Delta} \cup Paths)$ such that*

- $|t'|_u = 1, \quad \mathrm{label}_t(p') = \sigma, \quad \langle f,j\rangle \in \mathrm{in}_M(\sigma),$
- $p = \begin{cases} p' & \text{if } j = 0\\ j & \text{if } j \neq 0\,,\ p' = \varepsilon\,,\\ p'.j & \text{otherwise}\end{cases}$
- $t_1 = t'\left[u/\langle f,p\rangle\right]$ *and*

- $t_2 = t'\Big[u/\mathrm{relabel}_{M,p}\big(rhs_{M,\sigma}(\langle f,j\rangle)\big)\Big]$.

A redex *is such an occurrence of a term* $\langle f,p\rangle$*.* □

The following example illustrates the application of the derivation relation and the process of computing a normal form using an attributed tree transducer.

2.4.8 Example (reduction using M_{a-rev})
Let $M = M_{a-rev}$ be the attributed tree transducer which is defined in Example 2.4.3 and furthermore let $t = \widehat{\sigma}\,(A\,(B\,(B\,N)))$.

$$\begin{array}{ll}
 & \langle \widehat{s}, \varepsilon\rangle \\
\Rightarrow_{M,t} & \langle rev, 1\rangle \\
\Rightarrow_{M,t} & \langle rev, 1.1\rangle \\
\Rightarrow_{M,t} & \langle rev, 1.1.1\rangle \\
\Rightarrow_{M,t} & \langle rev, 1.1.1.1\rangle \\
\Rightarrow_{M,t} & \langle i, 1.1.1.1\rangle \\
\Rightarrow_{M,t} & B\,\langle i, 1.1.1\rangle \\
\Rightarrow_{M,t} & BB\,\langle i, 1.1\rangle) \\
\Rightarrow_{M,t} & BBA\,\langle i, 1\rangle)) \\
\Rightarrow_{M,t} & BBAN
\end{array}$$

□

Finally we associate translation semantics to a given attributed tree transducer.

2.4.9 Definition (induced translation)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be an attributed tree transducer. The translation induced by M, denoted $\tau(M)$, is defined to be

$$\tau(M) = \{\, (t_\Sigma, t_\Delta) \mid t_\Sigma \in T_\Sigma \,,\, t_\Delta \in T_\Delta \,,\, t_\Delta = nf_{\Rightarrow_{M,(\widehat{\sigma}\, t_\Sigma)}}(\langle \widehat{s}, \varepsilon\rangle) \,\}.$$

□

Note that the translation is independend of the chosen root symbol, since it is only used as an auxillary device. In the following attributed tree transducers are considered equivalent, if and only if their induced translations are equal. In addition we introduce the class ATT of all translations computable by attributed tree transducers.

2.4.10 Definition (single-use, syntactic single-use, at least single-use)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be an attributed tree transducer. If and only if for every input symbol $\sigma \in \underline{\Sigma}_+$ and every outer attribute occurrence $\chi \in \mathrm{out}_M(\sigma)$

$$\sum_{\xi \in \mathrm{in}_M(\sigma)} |rhs_{M,\sigma}(\xi)|_\chi \begin{cases} \leq 1 \\ = 1 \\ \geq 1 \end{cases} \textit{then } M \textit{ is called} \begin{cases} \textit{single-use} \\ \textit{syntactic single-use} \\ \textit{at least single-use} \end{cases}.$$

The classes of all translations computable by single-use, syntactic single-use and at least single-use attributed tree transducers are denoted ATT_{su}, ATT_{ssu} and ATT_{lsu}, respectively. □

The above definition intuitively states that every outer attribute occurrence appears at most once (in the case of single-use) or at least once (in the case of at least single-use) in all right hand sides of every right hand side function. The definition of syntactic single-use, taken out of [Gie88], is the combination of both at least single-use and single-use.

2.4.11 Definition (weakly single-use attributed tree transducer)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be an attributed tree transducer. If and only if for every input symbol $\sigma \in \underline{\Sigma}_+$ and every outer synthesized attribute occurrence $\chi \in \big(\mathrm{out}_M(\sigma) \cap Fun_{S_+,N_+}\big)$

$$\sum_{\xi \in \mathrm{in}_M(\sigma)} |rhs_{M,\sigma}(\xi)|_\chi \;\leq 1$$

then M is called weakly single-use. *The class of all translations computable by weakly single-use attributed tree transducers is denoted* ATT_{wsu}. □

If we reconsider our running example, it is easy to see that M_{a-rev} is in fact a syntactic single-use attributed tree transducer, hence it is also single-use, weakly single-use and at least single-use.

2.4.12 Definition (translation composition)
Let τ_1 and τ_2 be translations. Then the sequential composition $\tau_1;\tau_2$ is defined to be

$$\tau_1;\tau_2 = \{\, (t_1, t_2) \mid (t_1, t') \in \tau_1 \,,\, (t', t_2) \in \tau_2 \,\}.$$

The sequential composition of two suitable tree transducers shall express the sequential composition of their induced translations. The sequential composition is also extended to classes of translations in the obvious way.

Whenever we compose two translations, tree transducers or classes of tree transductions, then we will always assume that they are suitable, i.e. that their typing allows composition.

A thorough introduction to tree transducer theory can be found in [FV98].

Chapter 3

Incorporate attributed tree transducers

The first step in our series of transformations is the construction of an equivalent attributed tree transducer given a macro tree transducer (functional program). Since ATT $\subset$ MAC, this is not possible in general, but for several subclasses it was shown that an equivalent attributed tree transducer can be constructed. Specifically we will consider weakly single-use macro tree transducers, because for them an equivalent attributed tree transducer can be constructed ($\mathrm{MAC}_{wsu} \subseteq \mathrm{ATT}$).

The first section introduces the construction and provides an example of its application. In addition we also present an import lemma, which establishes a relation between properties of the original macro tree transducer and properties of the gained attributed tree transducer. The following section will establish a new derivation relation for attributed tree transducers, modelling a call-by-need evaluation strategy more closely. Finally in the last section we will investigate the efficiency relation between the original macro tree transducer and the newly created attributed tree transducer.

3.1 The construction

In this section we will state the construction taken from [Küh00a]. It transforms a weakly single-use macro tree transducer into an equivalent attributed tree transducer (tree transducers are considered equivalent, if their induced translations are equal).

3.1.1 Construction ($\mathbf{MAC}_{wsu} \subseteq$ ATT)
Let $M = (\Sigma, \Delta, F, e, R)$ be a weakly single-use macro tree transducer and furthermore let $\alpha \in \Delta^{(0)}$ be arbitrary. We then construct an attributed tree transducer $M' = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R')$ with arbitrary $\widehat{\sigma}$ and $\widehat{s}$, such that $\tau(M') = \tau(M)$, as follows:

- $S = \underline{F}$
- $I = \{\, y_{f,l} \mid f \in \underline{F}\,,\, l \in [\mathrm{rank}_F(f) - 1] \,\}$
- $R' = (\, rhs_{M',\sigma} \mid \sigma \in \underline{\Sigma}_+ \,)$

Let $m = \max\{ \operatorname{rank}_\Sigma(\sigma) \mid \sigma \in \underline{\Sigma}_+ \}$ be the maximum arity of the input symbols, then we define the functions top_f and dec_f for every $n \in N$ and $f \in F^{(n+1)}$ to be:

$$\begin{array}{lcll} \mathrm{top}_f & : & RHS_{\Delta,F,m,n} & \to T_\Delta(\mathrm{out}_{S,I}(m)) \\ \mathrm{dec}_f & : & RHS_{\Delta,F,m,n} & \to \mathcal{P}(\mathrm{in}_{S,I}(m) \times T_\Delta(\mathrm{out}_{S,I}(m))) \end{array}$$

- for every $l \in [n]$
  $$\begin{array}{ll} \mathrm{top}_f(y_l) & = \langle y_{f,l}, 0\rangle \\ \mathrm{dec}_f(y_l) & = \emptyset \end{array}$$
- for every $k \in N$, $\delta \in \Delta^{(k)}$ and $\varrho_1, \ldots, \varrho_k \in RHS_{\Delta,F,m,n}$
  $$\begin{array}{ll} \mathrm{top}_f(\delta\, \varrho_1 \ldots \varrho_k) & = \delta\,\big(\mathrm{top}_f(\varrho_1)\big) \ldots \big(\mathrm{top}_f(\varrho_k)\big) \\ \mathrm{dec}_f(\delta\, \varrho_1 \ldots \varrho_k) & = \bigcup_{i\in[n]} \mathrm{dec}_f(\varrho_i) \end{array}$$
- for every $r \in N$, $g \in F^{(r)}$, $j \in [m]$ and $\varrho_1, \ldots, \varrho_r \in RHS_{\Delta,F,m,n}$
  $$\begin{array}{ll} \mathrm{top}_f(g\, x_j\, \varrho_1 \ldots \varrho_r) & = \langle g, j\rangle \\ \mathrm{dec}_f(g\, x_j\, \varrho_1 \ldots \varrho_r) & = \bigcup_{i\in[r]} \mathrm{dec}_f(\varrho_i) \cup \{ \big(\langle y_{g,l}, j\rangle, \mathrm{top}_f(\varrho_l)\big) \mid l \in [r] \} \end{array}$$

Using the two previously defined functions, we can now construct the right hand side functions $rhs_{M',\sigma}$ by stating that for every $\sigma \in \underline{\Sigma}$ and $\xi \in \mathrm{in}_M(\sigma)$

$$rhs_{M',\sigma}(\xi) = \begin{cases} \mathrm{top}_f(rhs_{M,\sigma,f}) & \text{, if } \xi = \langle f, 0\rangle \text{ with } f \in S \\ \varrho & \text{, if } (\xi, \varrho) \in \bigcup_{f\in S} \mathrm{dec}_f(rhs_{M,\sigma,f}) \\ \alpha & \text{, otherwise} \end{cases} .$$

The right hand side function $rhs_{M',\widehat{\sigma}}$ is defined for every $\xi \in \mathrm{in}_M(\widehat{\sigma})$ by stating that

$$rhs_{M',\widehat{\sigma}}(\xi) = \begin{cases} \mathrm{top}_f(e) & \text{, if } \xi = \langle \widehat{s}, 0\rangle \\ \varrho & \text{, if } (\xi, \varrho) \in \mathrm{dec}_f(e) \text{ for arbitrary } f \\ \alpha & \text{, otherwise} \end{cases} .$$

□

3.1.2 Example (transform M_{rev})

M_{rev} of Example 2.3.3 is a weakly single-use macro tree transducer as we have seen and therefore we can apply Construction 3.1.1 to gain the following attributed tree transducer, which induces the same translation.

$$M' = (\Sigma, \Sigma, \{rev\}, \{y_{rev,1}\}, \widehat{s}, \widehat{\sigma}, (rhs_{M',\widehat{\sigma}}, rhs_{M',A}, rhs_{M',B}, rhs_{M',N}))$$

where $\Sigma = \{A^{(1)}, B^{(1)}, N^{(0)}\}$ and the right hand side functions are defined by

$$\begin{array}{lcl|lcl} rhs_{M',\widehat{\sigma}}(\langle \widehat{s}, 0\rangle) & = & \langle rev, 1\rangle & rhs_{M',A}(\langle rev, 0\rangle) & = & \langle rev, 1\rangle \\ rhs_{M',\widehat{\sigma}}(\langle y_{rev,1}, 1\rangle) & = & N & rhs_{M',A}(\langle y_{rev,1}, 1\rangle) & = & A\,\langle y_{rev,1}, 0\rangle \\ rhs_{M',N}(\langle rev, 0\rangle) & = & \langle y_{rev,1}, 0\rangle & rhs_{M',B}(\langle rev, 0\rangle) & = & \langle rev, 1\rangle \\ & & & rhs_{M',B}(\langle y_{rev,1}, 1\rangle) & = & B\,\langle y_{rev,1}, 0\rangle. \end{array}$$

□

We clearly see that this attributed tree transducer M' is almost syntactically equal to the attributed tree transducer M_{a-rev} of Example 2.4.3. By consistent renaming of the attributes we could gain syntactic equivalence. Therefore we can conclude that M_{a-rev} computes the same translation as M_{rev}.

In the following we will try to find sufficient restrictions of the original macro tree transducer M such that the attributed tree transducers M' gained by Construction 3.1.1 has a particular property. The first two results were already shown in [Küh00a] and are just stated for completeness.

$$\begin{aligned} \mathrm{MAC}_{wsu} &\subseteq \mathrm{ATT}_{wsu} && (3.1)\\ \mathrm{MAC}_{su} &\subseteq \mathrm{ATT}_{su} && (3.2)\\ \mathrm{MAC}_{ssu}\cap\mathrm{MAC}_{ndc} &\subseteq \mathrm{ATT}_{lsu} && (3.3)\\ \mathrm{MAC}_{ssu}\cap\mathrm{MAC}_{prc} &\subseteq \mathrm{ATT}_{ssu} && (3.4) \end{aligned}$$

3.1.3 Lemma (statement 3.3)
Let $M = (\Sigma, \Delta, F, e, R)$ be macro tree transducer, which is syntactic single-use and non-deleting in context parameters, and let $M' = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R')$ be the resulting attributed tree transducer of Construction 3.1.1 applied to M. Then M' is at least single-use.

Proof: Let $\sigma \in \underline{\Sigma}$ be an arbitrary input symbol. M' is at least single-use, if we can prove that for every $\chi \in \mathrm{out}_{M'}(\sigma)$ the condition

$$\sum_{\xi\in\mathrm{in}_M(\sigma)} |rhs_{M,\sigma}(\xi)|_\chi \;\geq\; 1$$

holds. Since $\mathrm{out}_{M'}(\sigma) = Fun_{I,\{0\}} \cup Fun_{S,[\mathrm{rank}_\Sigma(\sigma)]}$ we prove the lemma in two stages. In the first half of the proof, we will show that the property holds for outer inherited attribute occurrences and in the second part we will consider the remaining outer synthesized attribute occurrences.

- Let $y_{f,l} \in I$ be an arbitrary inherited attribute. Then $\langle y_{f,l}, 0\rangle \in \mathrm{out}_{M'}(\sigma)$. Such an occurrence can only be created by the top_f function, if the context parameter y_l is met. This necessarily happens, since M is non-deleting in context parameters and therefore $|rhs_{M,\sigma,f}|_{y_l} \geq 1$.
- Let $f \in S$ and $j \in [\mathrm{rank}_\Sigma(\sigma)]$ be arbitrary. Then $\langle f, j\rangle \in \mathrm{out}_{M'}(\sigma)$ and such an occurrence can only be created by some top_g function with $g \in S$, if the right hand side $rhs_{M,\sigma,g}$ contains a call like $(f\,x_j\ldots)$. We can conclude that such a g must exist, since M is syntactic single-use and hence a call like $(f\,x_j\ldots)$ occurs in exactly one right hand side.

The very same kind of reasoning can be applied for the input symbol $\widehat{\sigma}$, where the first half is trivial since there are no outer inherited attributes. So we have proven the result for arbitrary $\sigma \in \underline{\Sigma}_+$ and therefore completed the proof. □

3.1.4 Lemma (statement 3.4)

$$MAC_{ssu} \cap MAC_{prc} \subseteq ATT_{ssu}$$

Proof: Since $\mathrm{ATT}_{ssu} = \mathrm{ATT}_{lsu} \cap \mathrm{ATT}_{su}$, we simply combine the results (3.2) and (3.3).

$$\begin{aligned} \mathrm{MAC}_{su}\cap\mathrm{MAC}_{ssu}\cap\mathrm{MAC}_{ndc} &\subseteq \mathrm{ATT}_{lsu}\cap\mathrm{ATT}_{su} & \\ \mathrm{MAC}_{wsu}\cap\mathrm{MAC}_{nc}\cap\mathrm{MAC}_{ssu}\cap\mathrm{MAC}_{ndc} &\subseteq \mathrm{ATT}_{ssu} & \text{by Def. 2.3.13}\\ \mathrm{MAC}_{ssu}\cap\mathrm{MAC}_{prc} &\subseteq \mathrm{ATT}_{ssu} & \text{by Def. 2.3.12} \end{aligned}$$

□

3.2 Establishing a new derivation relation

Since the aim of this thesis is to establish an efficiency relation between functional programs using the call-by-need evaluation strategy, we will characterize a call-by-need evaluation strategy in the following and then we will adjust the derivation relations of Definitions 2.3.5 and 2.4.7 to partially modell a derivation enforcing the call-by-need evaluation strategy.

The call-by-need evaluation strategy is used in lazy functional programming languages (e.g. Haskell as a prominent member) and therefore shall be used for macro tree transducers as well. Basically a call-by-need evaluation strategy is just a better call-by-name (OI, outermost, normal order) evaluation strategy, since the call-by-need strategy only corrects a deficiency which might occur when parameter positions are copied while reducing using the call-by-name strategy. A comprehensive introduction into evaluation strategies can be found, for example, in [BW88].

The following two properties characterize a call-by-need derivation relation:

- Whenever nested redeces occur in some sentential form, then some outermost redex (i.e. a redex which is not contained in some other redex) is reduced first. Imposing this restriction yields two major advantages, firstly if a normal-form exists it will be reached after finitely many reduction steps and secondly only redeces which actually contribute to the final result (normal form) will be reduced. Basically this characterizes the call-by-name evaluation strategy and the derivation relations of Definition 2.3.5 and Definition 2.4.7 already implement this property, since Definition 2.3.5 explicitly states it in the second item and in case of attributed tree transducers (Definition 2.4.7) nested redeces cannot appear.
- In addition common subtrees appearing at parameter positions (either input trees at recursion arguments or output trees at context parameters) are shared via references, such that the above mentioned subtrees are not copied and separately reduced. Consequently only deficiencies resulting out of copying context parameters might actually occur in the stated derivation relations, since the input trees do not contain redeces.

3.2.1 Example (copying macro tree transducer)
We define a macro tree transducer M by stating:

$$M = (\{\sigma^{(2)}, \alpha^{(0)}\}, \{\sigma^{(2)}, \alpha^{(0)}\}, \{f^{(2)}, g^{(1)}\}, (f\, x_1\, (g\, x_1)), R)$$

and

$$\begin{array}{llll} R = \{ & f\,(\sigma\, x_1\, x_2)\, y_1 & = & \sigma\,(f\, x_1\, y_1)\,(f\, x_2\, y_1), \\ & f\,\alpha\, y_1 & = & y_1, \\ & g\,(\sigma\, x_1\, x_2) & = & \sigma\,(g\, x_1)\,(g\, x_2), \\ & g\,\alpha & = & \alpha \qquad\qquad \} . \end{array}$$

Obviously the right hand side $\sigma\,(f\, x_1\, y_1)\,(f\, x_2\, y_1)$ contains two occurrences of the context parameter y_1. Consequently the macro tree transducer M is copying (not non-copying). Nevertheless the macro tree transducer M has many beneficial properties, namely M is syntactic single-use and non-deleting in context parameters.

We will outline two possible computations of the normal form, one of them using the introduced derivation relation and the other one using a call-by-need evaluation strategy. Therefore we will (in both cases) use the input tree $t = (\sigma\,\alpha\,\alpha)$.

call-by-name evaluation: (nine reduction steps in total)

$$\begin{array}{ll}
 & f(\sigma\,\alpha\,\alpha)\;(g\,(\sigma\,\alpha\,\alpha)) \\
\Rightarrow_M & \sigma\left(f\,\alpha\,(g\,(\sigma\,\alpha\,\alpha))\right)\left(f\,\alpha\,(g\,(\sigma\,\alpha\,\alpha))\right) \\
\Rightarrow_M & \sigma\left(g\,(\sigma\,\alpha\,\alpha)\right)\left(f\,\alpha\,(g\,(\sigma\,\alpha\,\alpha))\right) \\
\Rightarrow_M & \sigma\left(\sigma\,(g\,\alpha)\,(g\,\alpha)\right)\left(f\,\alpha\,(g\,(\sigma\,\alpha\,\alpha))\right) \\
\Rightarrow_M & \sigma\left(\sigma\,\alpha\,(g\,\alpha)\right)\left(f\,\alpha\,(g\,(\sigma\,\alpha\,\alpha))\right) \\
\Rightarrow_M & \sigma\left(\sigma\,\alpha\,\alpha\right)\left(f\,\alpha\,(g\,(\sigma\,\alpha\,\alpha))\right) \\
\Rightarrow_M & \sigma\,(\sigma\,\alpha\,\alpha)\;(g\,(\sigma\,\alpha\,\alpha)) \\
\Rightarrow_M & \sigma\,(\sigma\,\alpha\,\alpha)\;(\sigma\,(g\,\alpha)\,(g\,\alpha)) \\
\Rightarrow_M & \sigma\,(\sigma\,\alpha\,\alpha)\;(\sigma\,\alpha\,(g\,\alpha)) \\
\Rightarrow_M & \sigma\,(\sigma\,\alpha\,\alpha)\,(\sigma\,\alpha\,\alpha)
\end{array}$$

In the following reduction associated output trees (referenced output trees) are underlined in order to illustrate the reduction process.

call-by-need evaluation: (six reduction steps in total)

$$\begin{array}{ll}
 & f(\sigma\,\alpha\,\alpha)\;(g\,(\sigma\,\alpha\,\alpha)) \\
\stackrel{e}{\Rightarrow} & \sigma\left(f\,\alpha\,(\underline{g\,(\sigma\,\alpha\,\alpha)})\right)\left(f\,\alpha\,(\underline{g\,(\sigma\,\alpha\,\alpha)})\right) \\
\stackrel{e}{\Rightarrow} & \sigma\left(\underline{g\,(\sigma\,\alpha\,\alpha)}\right)\left(f\,\alpha\,(\underline{g\,(\sigma\,\alpha\,\alpha)})\right) \\
\stackrel{e}{\Rightarrow} & \sigma\,(\underline{g\,(\sigma\,\alpha\,\alpha)})\;(\underline{g\,(\sigma\,\alpha\,\alpha)}) \\
\stackrel{e}{\Rightarrow} & \sigma\,(\underline{\sigma\,(g\,\alpha)\,(g\,\alpha)})\;(\underline{\sigma\,(g\,\alpha)\,(g\,\alpha)}) \\
\stackrel{e}{\Rightarrow} & \sigma\,(\underline{\sigma\,\alpha\,(g\,\alpha)})\;(\underline{\sigma\,\alpha\,(g\,\alpha)}) \\
\stackrel{e}{\Rightarrow} & \sigma\,(\underline{\sigma\,\alpha\,\alpha})\,(\underline{\sigma\,\alpha\,\alpha})
\end{array}$$

So the call-by-need evaluation strategy avoids unnecessary reductions, which result out of copying a nested redex. □

3.2.2 Definition (call-by-need derivation relation)
Let M be a macro tree transducer. In the following we will denote the outlined call-by-need derivation relation of M by $\stackrel{e}{\Rightarrow}_M$. □

Copying of context parameters might result in a major deficiency as we have seen, since the copied trees usually contain redeces. Of course those redeces will be copied as well, which will result in several reductions of the same subterm.

Since we want to perform an efficiency analysis using macro tree transducers at the level of attributed tree transducers, we must ascertain, that the efficiency measure we apply is comparable, such that the efficiency of the attributed tree transducer can somehow be related to the efficiency of the original program (macro tree transducer). The main difference (seen from the perspective of an attributed tree transducer) is the explicit handling of context parameters using inherited attributes and the absence of nested redeces.

In the following we present a new derivation relation especially for attributed tree transducers, which –to some extend– models a call-by-need evaluation strategy.

3.2.3 Definition (derivation relation)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be an attributed tree transducer and in addition let $t = (\widehat{\sigma}\, t')$ with $t' \in T_\Sigma$ be an input tree. Then we define the derivation relation $\overset{e}{\Rightarrow}_{M,t}$ of M according to the input tree t to be the following binary relation

$$\overset{e}{\Rightarrow}_{M,t} \subset T^2_\Delta(Fun_{F_+,\mathrm{paths(t)}})$$

such that for every $t_1, t_2 \in T_\Delta(Fun_{F_+,\mathrm{paths(t)}})$ the relation $t_1 \overset{e}{\Rightarrow} t_2$ holds, if and only if there exists $j, k \in N$ and $\sigma \in \Sigma_+^{(k)}$, $f \in F_+$ and $p' \in \mathrm{paths}(t)$ with $\mathrm{label}_t(p') = \sigma$ such that

- *for synthesized attributes $f \in S_+$, $p = p'$ and $j = 0$,*
- *for inherited attributes $f \in I$, $p = \begin{cases} j & \text{, if } p' = \varepsilon \\ p'.j & \text{, otherwise} \end{cases}$*

and in both cases we observe that for every attribute instance χ with $\chi \rightarrow^+_{M,t} \langle f, p\rangle$ the number of occurrences in the sentential form t_1 is exactly zero ($|t_1|_\chi = 0$) and

$$t_2 = t_1 \Big[\langle f, p\rangle \,/\, \mathrm{relabel}_{M,p'} \left(rhs_{M,\sigma}(\langle f, j\rangle)\right) \Big] .$$

□

So basically we search for permissible redeces as we did in the derivation relation defined in Definition 2.4.7. In addition we only reduce an attribute instance $\langle f, p\rangle$, if there is no dependent attribute instance χ present in the sentential form t_1. The use of references is modelled by globally substituting the current attribute instance, effectively reducing all occurrences of attribute instances $\langle f, p\rangle$ in parallel. In the following we will provide an essential lemma, which we will use later on. An example illustrating the above definition will be provided after the next lemma.

3.2.4 Proposition (termination and confluency)
Let M be an attributed tree transducer. It is easy to see that the above defined derivation relation is indeed terminating and confluent. Furthermore let s be a sentential form and t be an input tree. Then both derivation relations compute the same normal form $nf_{\overset{e}{\Rightarrow}_{M,t}}(s) = nf_{\Rightarrow_{M,t}}(s)$. □

3.2.5 Lemma (single attribute instance derivation)
The derivation relation defined in Definition 3.2.3 reduces each attribute instance at most once.

Proof: Given an attributed tree transducer $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$, an input tree $t = (\widehat{\sigma}\, t')$, $t' \in T_\Sigma$ and an attribute instance $\langle f, p\rangle$ with $f \in F_+$, $p \in \mathrm{paths}(t)$ and $|t_1|_{\langle f,p\rangle} \geq 1$, which appears in some sentential form $t_1 \in SF_{\Delta, F_+, t}$, then the attribute instance $\langle f, p\rangle$ is only reduced, if

$$\neg(\exists f' \in F_+)\,(\exists p' \in \mathrm{paths}(t))\,\Big(\big(|t_1|_{\langle f',p'\rangle} \geq 1\big) \wedge \big(\langle f', p'\rangle \rightarrow^+_{M,t} \langle f, p\rangle\big)\Big).$$

So, reducing the sentential form t_1 using the redex $\langle f, p\rangle$ to t_2 ($t_1 \overset{e}{\Rightarrow}_{M,t} t_2$), subsequent reductions cannot introduce the attribute instance $\langle f, p\rangle$ to some sentential form (contradicts the given condition) and furthermore the reduction substitutes every occurrence of the attribute instance, such that the attribute instance $\langle f, p\rangle$ cannot appear in the resulting sentential form ($|t_2|_{\langle f,p\rangle} = 0$). Thus we successfully showed, that an attribute instance is never reduced more than once. □

3.2.6 Example (single attribute instance derivation)
To illustrate Lemma 3.2.5 and Definition 3.2.3, we will apply Construction 3.1.1 to the macro tree transducer M defined in Example 3.2.1 to gain the attributed tree transducer

$$M' = (\{\sigma^{(2)}, \alpha^{(0)}\}, \{\sigma^{(2)}, \alpha^{(0)}\}, \{f, g\}, \{y_{f,1}\}, \widehat{s}, \widehat{\sigma}, (rhs_{M',\widehat{\sigma}}, rhs_{M',\sigma}, rhs_{M',\alpha}))$$

with right hand side functions

$$\begin{array}{lcl}
rhs_{M',\widehat{\sigma}}(\langle \widehat{s}, 0\rangle) & = & \langle f, 1\rangle \\
rhs_{M',\widehat{\sigma}}(\langle y_{f,1}, 1\rangle) & = & \langle g, 1\rangle \\
rhs_{M',\sigma}(\langle f, 0\rangle) & = & \sigma\,\langle f, 1\rangle\,\langle f, 2\rangle \\
rhs_{M',\sigma}(\langle g, 0\rangle) & = & \sigma\,\langle g, 1\rangle\,\langle g, 2\rangle \\
rhs_{M',\sigma}(\langle y_{f,1}, 1\rangle) & = & \langle y_{f,1}, 0\rangle \\
rhs_{M',\sigma}(\langle y_{f,1}, 2\rangle) & = & \langle y_{f,1}, 0\rangle \\
rhs_{M',\alpha}(\langle f, 0\rangle) & = & \langle y_{f,1}, 0\rangle \\
rhs_{M',\alpha}(\langle g, 0\rangle) & = & \alpha.
\end{array}$$

Again we select $t = \widehat{\sigma}(\sigma\,\alpha\,\alpha)$ as input tree and perform a computation of the normal form using the new derivation relation.

$$\begin{array}{rll}
 & & \langle \widehat{s}, \varepsilon\rangle \\
 & \overset{e}{\Rightarrow}_{M',t} & \langle f, 1\rangle \\
 & \overset{e}{\Rightarrow}_{M',t} & \sigma\,\langle f, 1.1\rangle\,\langle f, 1.2\rangle \\
 & \overset{e}{\Rightarrow}_{M',t} & \sigma\,\langle y_{f,1}, 1.1\rangle\,\langle f, 1.2\rangle \\
(*) & \overset{e}{\Rightarrow}_{M',t} & \sigma\,\langle y_{f,1}, 1\rangle\,\langle f, 1.2\rangle \\
 & \overset{e}{\Rightarrow}_{M',t} & \sigma\,\langle y_{f,1}, 1\rangle\,\langle y_{f,1}, 1.2\rangle \\
(**) & \overset{e}{\Rightarrow}_{M',t} & \sigma\,\langle y_{f,1}, 1\rangle\,\langle y_{f,1}, 1\rangle \\
 & \overset{e}{\Rightarrow}_{M',t} & \sigma\,\langle g, 1\rangle\,\langle g, 1\rangle \\
 & \overset{e}{\Rightarrow}_{M',t} & \sigma\,\big(\sigma\,\langle g, 1.1\rangle\,\langle g, 1.2\rangle\big)\,\big(\sigma\,\langle g, 1.1\rangle\,\langle g, 1.2\rangle\big) \\
 & \overset{e}{\Rightarrow}_{M',t} & \sigma\,\big(\sigma\,\alpha\,\langle g, 1.2\rangle\big)\,\big(\sigma\,\alpha\,\langle g, 1.2\rangle\big) \\
 & \overset{e}{\Rightarrow}_{M',t} & \sigma\,(\sigma\,\alpha\,\alpha)\,(\sigma\,\alpha\,\alpha)
\end{array}$$

We clearly see, that in fact every attribute instance is reduced at most once and that the reference effect is achieved by using the global substitution. Nevertheless the attributed tree transducer M' requires more reduction steps (ten in total), but this is only due to the explicit handling of context parameters. Though the attribute instance $\langle y_{f,1}, 1\rangle$ in the sentential form marked with $(*)$ seems to be a suitable redex, the new derivation relation prohibits reduction, since there is still an attribute instance (namely $\langle f, 1.2\rangle$) which depends on it. Finally at $(**)$ this attribute instance can be reduced and both occurrences of $\langle y_{f,1}, 1\rangle$ are reduced in parallel by global substitution. □

3.3 Efficiency considerations

Within this section we will try to establish an efficiency relation between the original weakly single-use macro tree transducer and the resulting attributed tree transducer created by Construction 3.1.1, when applied to the macro tree transducer. Therefore we firstly define a performance measure, which will simply be the plain number of reduction steps, as counted in the previous examples. This measure is particularly easy to determine, since it only counts function calls / attribute instance reductions. Of course there are more elaborate measures available, but they tend to be either highly machine-dependent or hard to determine.

3.3.1 Definition (number of reduction steps)
Let M be a tree transducer with input ranked alphabet Σ, let t be an input tree and let $\Rightarrow$ be a derivation relation of that tree transducer. We denote by $\text{count}_{M,\Rightarrow}(t)$ *the number of reduction steps necessary to compute the normal form using the derivation relation $\Rightarrow$ and the input tree t.*

- *Given an attributed tree transducer $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ this number is defined to be $n \in N$, if and only if*

$$\langle \widehat{s}, \varepsilon \rangle \Rightarrow^n \; nf_{\Rightarrow}(\langle \widehat{s}, \varepsilon \rangle)$$

- *and given a macro tree transducer $M = (\Sigma, \Delta, F, e, R)$ the number of reduction steps is n, if and only if*

$$e[x_1/t] \Rightarrow^n \; nf_{\Rightarrow}(e[x_1/t]).$$

□

Obviously attributed tree transducers need at least one reduction step, which is necessary to reduce the initial synthesized attribute instance. In the following this reduction step is often considered separately.

3.3.2 Lemma (number of reduction steps for attributed tree transducers)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be an attributed tree transducer, $t = (\widehat{\sigma}\, t')$ with $t' \in T_\Sigma$ be an input tree and let $\rightarrow_{M,t}$ be the direct dependency relation of the attributed tree transducer M between attribute instances of the input tree t.

$$\text{count}_{M,\overset{e}{\Rightarrow}_{M,t}} = |\{ \langle f,p \rangle \mid \langle \widehat{s}, \varepsilon \rangle \rightarrow^*_{M,t} \langle f,p \rangle \}|$$

Then the number of reduction steps using the newly established derivation relation $\overset{e}{\Rightarrow}_{M,t}$ and input tree t is equal to the number of attribute instances, on which the distinguished attribute instance $\langle \widehat{s}, \varepsilon \rangle$ depends.

Proof: Starting with the initial synthesized attribute instance $\langle \widehat{s}, \varepsilon \rangle$ as the initial sentential form (according to Definition 3.3.1), exactly the attribute instances $\langle f,p \rangle$, $f \in F$, $p \in \text{paths}(t)$ with $\langle \widehat{s}, \varepsilon \rangle \rightarrow^*_{M,t} \langle f,p \rangle$ will appear in some sentential form during the reduction process of $\langle \widehat{s}, \varepsilon \rangle$ into its normal form $nf_{\Rightarrow_{M,t}}(\langle \widehat{s}, \varepsilon \rangle)$. Since the normal form does not contain attribute instances, every such attribute instance $\langle f,p \rangle$ needs to be reduced at least once and according to Lemma 3.2.5 at most once. Consequently the number of reduction steps is exactly the number of such attribute instances. □

3.3.3 Example (reduction for M_{a-rev})
In this example we will apply the attributed tree transducer $M = M_{a-rev}$ of Example 2.4.3 to the input tree $t = (\widehat{\sigma}\, ABBN)$ and demonstrate a step-wise reduction

starting with $\langle \widehat{s}', \varepsilon\rangle$ using the derivation relation $\overset{e}{\Rightarrow}_{M,t}$.

$$\begin{array}{ll} & \langle \widehat{s}', \varepsilon\rangle \\ \overset{e}{\Rightarrow}_{M,t} & \langle s', 1\rangle \\ \overset{e}{\Rightarrow}_{M,t} & \langle s', 1.1\rangle \\ \overset{e}{\Rightarrow}_{M,t} & \langle s', 1.1.1\rangle \\ \overset{e}{\Rightarrow}_{M,t} & \langle s', 1.1.1.1\rangle \\ \overset{e}{\Rightarrow}_{M,t} & \langle i', 1.1.1.1\rangle \\ \overset{e}{\Rightarrow}_{M,t} & B\,\langle i', 1.1.1\rangle \\ \overset{e}{\Rightarrow}_{M,t} & BB\,\langle i', 1.1\rangle \\ \overset{e}{\Rightarrow}_{M,t} & BBA\,\langle i', 1\rangle \\ \overset{e}{\Rightarrow}_{M,t} & BBAN \end{array}$$

□

Within the reduction nine different attribute instances appeared and consequently (by Lemma 3.3.2) nine steps were necessary to compute the normal form. If we reconsider Example 3.2.1, then we easily see that the stated Lemma 3.3.2 holds there as well.

In the following two theorems we finally relate the number of reduction steps of the attributed tree transducer to the number of reduction steps of the original macro tree transducer. The first theorem states this for the call-by-name evaluation strategy, whereas the second theorem considers call-by-need.

3.3.4 Theorem (relating the number of reduction steps (call-by-name))
Let $M = (\Sigma, \Delta, F, e, R)$ be a weakly single-use macro tree transducer, let $t \in T_\Sigma$ be an input tree and furthermore let $M' = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R')$ be the attributed tree transducer resulting from Construction 3.1.1 using M as input. Then

$$\mathrm{count}_{M,\Rightarrow_M}(t) = \mathrm{count}_{M',\Rightarrow_{M',t}}(t) - \imath_{M',\Rightarrow_{M',t}}(t) - 1$$

where $\imath_{M',\Rightarrow_{M',t}}(t)$ is exactly the number of reduction steps invested (during computation of the normal form) to reduce inherited attribute instances of t with M'.

Proof: Let $\Rightarrow = \Rightarrow_M$, $\Rightarrow_t = \Rightarrow_{M',t}$, $\overline{\Rightarrow} = \Rightarrow_{\overline{M}}$ and $\overline{\Rightarrow}_t = \Rightarrow_{\overline{M}',t}$. In order to prove this result, we will make extensive use of the property, that the attributed tree transducer M' defines the same translation as M. We will exploit the following trick, also mentioned in [Voi01], a second time in Theorem 5.2.1. Firstly we select an arbitrary output symbol $\diamond^{(1)} \notin \Delta$ and define a new macro tree transducer $\overline{M} = (\Sigma, \Delta \cup \{\diamond^{(1)}\}, F, e, \overline{R})$, where the rule-set $\overline{R}$ contains for every $k, n \in N$, $\sigma \in \Sigma^{(k)}$ and $f \in F^{(n+1)}$ a rule

$$f(\sigma\, x_1 \dots x_k)\, y_1 \dots y_n = \diamond\,(rhs_{M,\sigma,f}).$$

We modify the original rules of M to output the special output symbol $\diamond$ and afterwards behave like the original rules do. Then obviously the number of reduction steps is

$$\mathrm{count}_{M,\Rightarrow}(t) = \mathrm{count}_{\overline{M},\overline{\Rightarrow}}(t) = |nf_{\overline{\Rightarrow}}(e[x_1/t])|_\diamond,$$

since the macro tree transducers M and $\overline{M}$ effectively behave equally just that the macro tree transducer $\overline{M}$ outputs the special symbol $\diamond$ at every reduction step.

The macro tree transducer $\overline{M}$ is also weakly single-use, since the original macro tree transducer$M$ is, and therefore we can apply Construction 3.1.1 to $\overline{M}$ to gain the attributed tree transducer $\overline{M}' = (\Sigma, \Delta \cup \{\diamond^{(1)}\}, S, I, \widehat{s}, \widehat{\sigma}, \overline{R}')$, which is only a slight modification of M' with $\overline{R}' = (\,\overline{rhs}'_{\overline{M}',\sigma} \,|\, \sigma \in \underline{\Sigma}_+\,)$, such that for every $\sigma \in \underline{\Sigma}_+$ and $\xi \in \mathrm{in}_{\overline{M}'}(\sigma)$ the right hand side function $\overline{rhs}'_{\overline{M}',\sigma}$ is defined to be

$$\overline{rhs}'_{\overline{M}',\sigma}(\xi) = \begin{cases} \diamond\left(rhs_{M',\sigma}(\xi)\right) & \text{, if } \xi \in Fun_{S,\{0\}} \\ rhs_{M',\sigma}(\xi) & \text{, otherwise} \end{cases}.$$

Again we can conclude that

$$\mathrm{count}_{M',\Rightarrow_t}(t) = \mathrm{count}_{\overline{M}',\overline{\Rightarrow}_t}(t) \qquad \text{and} \qquad \imath_{M',\Rightarrow_t}(t) = \jmath_{\overline{M}',\overline{\Rightarrow}_t}(t),$$

where $\jmath_{\overline{M}',\overline{\Rightarrow}_t}$ is exactly the number of reduction steps invested to reduce inherited attribute instances (during computation of the normal form) using $\overline{M}'$. Consequently

$$\mathrm{count}_{M',\Rightarrow_t}(t) - \imath_{M',\Rightarrow_t}(t) = \mathrm{count}_{\overline{M}',\overline{\Rightarrow}_t}(t) - \jmath_{\overline{M}',\overline{\Rightarrow}_t}(t) = |nf_{\overline{\Rightarrow}_t}(\langle\widehat{s},\varepsilon\rangle)|_\diamond + 1$$

This result is due to the strong resemblance of the attributed tree transducers M' and $\overline{M}'$. They effectively exhibit equal behaviour just that the attributed tree transducer $\overline{M}'$ outputs the special symbol $\diamond$ every time a synthesized attribute instance different from the initial synthesized attribute instance $\langle\widehat{s},\varepsilon\rangle$ is reduced. Since the reduction of the initial synthesized attribute instance $\langle\widehat{s},\varepsilon\rangle$ also counts as one reduction step, we added one.

Due to Construction 3.1.1 the induced translations of $\overline{M}$ and $\overline{M}'$ are equal ($\tau(\overline{M}) = \tau(\overline{M}')$) and accordingly $nf_{\overline{\Rightarrow}}(e[x_1/t]) = nf_{\overline{\Rightarrow}_t}(\langle\widehat{s},\varepsilon\rangle)$, so we can conclude

$$|nf_{\overline{\Rightarrow}}(e[x_1/t])|_\diamond = |nf_{\overline{\Rightarrow}_t}(\langle\widehat{s},\varepsilon\rangle)|_\diamond$$

and with the previous observations

$$\mathrm{count}_{M,\Rightarrow}(t) = \mathrm{count}_{M',\Rightarrow_t}(t) - \imath_{M',\Rightarrow_t}(t) - 1.$$

3.3.5 Theorem (relating the number of reduction steps (call-by-need))
Let $M = (\Sigma, \Delta, F, e, R)$ be a weakly single-use macro tree transducer, let $t \in T_\Sigma$ be an input tree and furthermore let $M' = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R')$ be the attributed tree transducer resulting from Construction 3.1.1 using M as input. Then

$$\mathrm{count}_{M,\overset{e}{\Rightarrow}_M}(t) = \mathrm{count}_{M',\overset{e}{\Rightarrow}_{M',t}}(t) - \imath_{M',\overset{e}{\Rightarrow}_{M',t}}(t) - 1$$

where $\imath_{M',\overset{e}{\Rightarrow}_{M',t}}(t)$ is exactly the number of reduction steps invested (during computation of the normal form) to reduce inherited attribute instances of t with M'.

Proof: First of all, if no context parameter is ever shared in the reduction, then the derivation relations ($\overset{e}{\Rightarrow}_M$ and $\overset{e}{\Rightarrow}_{M',t}$) behave exactly like their corresponding call-by-name derivation relations ($\Rightarrow_M$ and $\Rightarrow_{M',t}$). This only holds, since M' is weakly single-use. Let J' be a family consisting of all reduced synthesized attribute instances (excluding the initial synthesized attribute instance) that appeared during computation of the normal form $nf_{\Rightarrow_{M',t}}(\langle\widehat{s},\varepsilon\rangle)$ using the call-by-name derivation relation. Furthermore let J be the family containing exactly the pairs of function symbols and input subtrees, such that if a redex of the form $(f\, t_\Sigma \ldots)$ with $f \in \underline{F}$

and $t_\Sigma \in T_\Sigma$ was reduced in the computation of the normal form $nf_{\Rightarrow_M}(e[x_1/t])$, then $(f, t_\Sigma) \in J$. It is easy to see (with the help of Theorem 3.3.4 and Construction 3.1.1) that there is a bijection between those two families, i.e. if $\langle s, 1.p\rangle \in J'$ and t' is the subtree of t reachable by path p, then the pair $(s, t') \in J$.

Furthermore we observe that the number of elements of such a family determines the number of reduction steps. So

$$|J| = \mathrm{count}_{M,\Rightarrow_M}(t) \qquad |J'| = \mathrm{count}_{M',\Rightarrow_{M',t}}(t) - \imath_{M',\Rightarrow_{M',t}}(t) - 1$$

Corresponding to J and J' let K and K' denote the families gained in the very same way just using the call-by-need derivation relations. Then it is easy to see that both families K and K' are effectively the families J and J' with duplicates eliminated. This is due to the fact, that both call-by-need derivation relations essentially resemble the corresponding call-by-name derivation relation, but avoid multiple reductions of a redex. Consequently neither K nor K' can contain duplicates, since M is weakly single-use and therefore duplicates in J are necessarily shared and K' cannot contain duplicates due to Lemma 3.2.5. Together with the bijection we can conclude that

$$\mathrm{count}_{M,\overset{e}{\Rightarrow}_M}(t) = |K| = |K'| = \mathrm{count}_{M',\overset{e}{\Rightarrow}_{M',t}}(t) - \imath_{M',\overset{e}{\Rightarrow}_{M',t}}(t) - 1$$

□

The previous theorem intuitively states that if the macro tree transducer M needs k reduction steps to compute $\tau(M)(t)$ for a given t, then the attributed tree transducer M' also needs k reduction steps to compute $\tau(M')(t)$, if we disregard the reduction steps used to reduce inherited attribute instances as well as the reduction step used to reduce the initial synthesized attribute instance. We will illustrate this result in the following examples.

3.3.6 Example (M_{rev} and M_{a-rev})

We restate the reduction of the the input tree $(\widehat{\sigma}\, ABBN)$ using the attributed tree transducer $M = M_{a-rev}$ of Example 2.4.3.

$$\begin{array}{rll}
 & & \langle \widehat{s}', \varepsilon\rangle \\
 & \Rightarrow_{M',t} & \langle s', 1\rangle \\
 & \Rightarrow_{M',t} & \langle s', 1.1\rangle \\
 & \Rightarrow_{M',t} & \langle s', 1.1.1\rangle \\
 & \Rightarrow_{M',t} & \langle s', 1.1.1.1\rangle \\
 & \Rightarrow_{M',t} & \langle i', 1.1.1.1\rangle \\
(*) & \Rightarrow_{M',t} & B\,\langle i', 1.1.1\rangle \\
(*) & \Rightarrow_{M',t} & BB\,\langle i', 1.1\rangle) \\
(*) & \Rightarrow_{M',t} & BBA\,\langle i', 1\rangle)) \\
(*) & \Rightarrow_{M',t} & BBAN
\end{array}$$

Obviously if we do not count the reduction steps invested to reduce inherited attribute instances (the last four, marked with asterisks $(*)$), then we gain five reduction steps, which is one more than the original macro tree transducer. (cf. Example 2.3.6) □

3.3.7 Example (Example 3.2.1 and Example 3.2.6)

In Example 3.2.1 we showed, using the call-by-need evaluation strategy, that six reduction steps are necessary for the macro tree transducer M to compute the normal form. In Example 3.2.6 we stated the corresponding attributed tree transducer M' and also illustrated that the attributed tree transducer actually needed ten reduction steps. Again the efficiency relation of Theorem 3.3.5 holds, since exactly three reduction steps are used to reduce inherited attribute instances. □

Chapter 4

Compose attributed tree transducers

In this chapter we will consider the construction for composing a single-use attributed tree transducer and an attributed tree transducer. Therefore we state the construction in the first section along with examples. We will also present sufficient properties of the input attributed tree transducers, such that the resulting attributed tree transducer has a particular property. In the next section we investigate the efficiency of the composition result related to the efficiency of the original attributed tree transducers.

4.1 The construction

The following construction is based on results by [Gan83, Gie88]. We present the version of [Küh97]. A good introduction can be found in [Küh00a].

4.1.1 Construction ($\mathbf{ATT}_{su}$; ATT $\subseteq$ ATT)
Let $M_1 = (\Sigma, \Omega, S_1, I_1, \widehat{s}_1, \widehat{\sigma}, R_1)$ be a single-use attributed tree transducer and let $M_2 = (\Omega, \Delta, S_2, I_2, \widehat{s}_2, \widehat{\sigma}, R_2)$ be an attributed tree transducer. Furthermore let $m = \max\{\, \mathrm{rank}_{\Sigma_+}(\sigma) \mid \sigma \in \underline{\Sigma}_+ \,\}$ and let $\alpha \in \Delta^{(0)}$ be an arbitrary output symbol of M_2, then we define the attributed tree transducer $M_2' = (\Omega', \Delta', S_2, I_2, \widehat{s}_2, \widehat{\sigma}, R_2')$, which is able to process right hand sides of the attributed tree transducer M_1, to be:

- $\Omega' = \Omega \cup \mathrm{out}_{S_1,I_1}(m)$
- $\Delta' = \Delta \cup \mathrm{out}_{S_1 \times S_2, I_1 \times S_2}(m)$
- $R_2' = (\, rhs_{M_2',\omega} \mid \omega \in \underline{\Omega}'_+ \,)$ with right hand side functions such that
  - $rhs_{M_2',\omega} = rhs_{M_2,\omega}$ for every $\omega \in \underline{\Omega}_+$ and
  - for every $f_1 \in F_1$ and $j \in N$ with $\langle f_1, j\rangle \in \mathrm{out}_{S_1,I_1}(m)$

$$rhs_{M_2',\langle f_1,j\rangle}(\xi) = \begin{cases} \langle (f_1, s_2), j\rangle & \text{, if } \xi = \langle s_2, 0\rangle \text{ with } s_2 \in S_2 \\ \alpha & \text{, otherwise} \end{cases}$$

We can construct the attributed tree transducer $M_{1;2} = (\Sigma, \Delta, S, I, (\widehat{s}_1, \widehat{s}_2), \widehat{\sigma}, R)$ with $\tau(M_{1;2}) = \tau(M_1); \tau(M_2)$ as follows

- $S = (S_1 \times S_2) \cup (I_1 \times I_2)$,
- $I = (S_1 \times I_2) \cup (I_1 \times S_2)$,
- let $I_2 = \{i_1, \ldots, i_a\}$, $a = |I_2|$, then for every $f \in F_1$ and $j \in [m]$ the substitution $\kappa_{f,j}$ is defined to be $\kappa_{f,j} = [\langle i_1, \varepsilon\rangle / \langle (f, i_1), j\rangle \, , \, \ldots \, , \, \langle i_a, \varepsilon\rangle / \langle (f, i_a), j\rangle]$ and
- $R = (\, rhs_{M_{1;2},\sigma} \mid \sigma \in \underline{\Sigma}_+ \,)$ where
  - for every $k \in N$, $\sigma \in \Sigma_+^{(k)}$ and $\langle f_1, j\rangle \in \mathrm{in}_{M_1}(\sigma)$ with $f_1 \in F_1$ and $j \in N$ let $\varrho = rhs_{M_1,\sigma}(\langle f_1, j\rangle)$, then the right hand side function $rhs_{M_{1;2},\sigma}$ is defined as follows
    - for every $s_2 \in S_2$
    $$rhs_{M_{1;2},\sigma}(\langle (f_1, s_2), j\rangle) = nf_{\Rightarrow_{M_2',\varrho}}(\langle s_2, \varepsilon\rangle)\kappa_{f_1,j}$$
    - for every $p \in \mathrm{paths}(\varrho)$, $i_2 \in I_2$, $f_1' \in F_1$ and $j' \in N$ with $\langle f_1', j'\rangle \in \mathrm{out}_{M_1}(\sigma)$ and $\mathrm{label}_\varrho(p) = \langle f_1', j'\rangle$
    $$rhs_{M_{1;2},\sigma}(\langle (f_1', i_2), j'\rangle) = nf_{\Rightarrow_{M_2',\varrho}}(\langle i_2, p\rangle)\kappa_{f_1,j}$$
  - let $\varrho = rhs_{M_1,\widehat{\sigma}}(\langle \widehat{s}_1, 0\rangle)$, then the right hand side function $rhs_{M_{1;2},\widehat{\sigma}}$ is defined by stating
    - for every $p \in \mathrm{paths}(\widehat{\sigma}\ \varrho)$, $s_1' \in S_1$ and $i_2 \in I_2$ with $\mathrm{label}_{(\widehat{\sigma}\ \varrho)}(p) = \langle s_1', 1\rangle$
    $$rhs_{M_{1;2},\widehat{\sigma}}(\langle (s_1', i_2), 1\rangle) = nf_{M_2',(\widehat{\sigma}\ \varrho)}(\langle i_2, p\rangle)$$
    - and
    $$rhs_{M_{1;2},\widehat{\sigma}}(\langle (\widehat{s}_1, \widehat{s}_2), 0\rangle) = nf_{M_2',(\widehat{\sigma}\ \varrho)}(\langle \widehat{s}_2, \varepsilon\rangle).$$
- For every $k \in N$, $\sigma \in \Sigma_+^{(k)}$ and $\xi \in \mathrm{in}_{M_{1;2}}(\sigma)$, if the right hand side function $rhs_{M_{1;2},\sigma}$ is still undefined at ξ, then $rhs_{M_{1;2},\sigma}(\xi) = \alpha$.

□

Let M_1 be a single-use attributed tree transducer and M_2 be an attributed tree transducer. Then with the help of Construction 4.1.1 we can construct an attributed tree transducer $M_{1;2}$, which computes the same translation as the composition of M_1 and M_2. The new attributed tree transducer has the advantage of removing the intermediate result altogether. Therefore the attributed tree transducer $M_{1;2}$ avoids unnecessary memory allocations / deallocations, which were only needed to store the intermediate result or parts of it in the original sequential composition. Nevertheless the attributed tree transducer $M_{1;2}$ might require more reduction steps than the sequential execution of the attributed tree transducers M_1 and M_2. The following examples shall demonstrate Construction 4.1.1 and the possible effects concerning efficiency.

4.1.2 Example ($M_{palin}; M_{palin}$)
We define the attributed tree transducer $M_{palin} = (\Sigma, \Sigma, \{s\}, \{i\}, \widehat{s}, \widehat{\sigma}, R)$ with $\Sigma = \{A^{(1)}, B^{(1)}, N^{(0)}\}$ and $R = (rhs_{\widehat{\sigma}}, rhs_A, rhs_B, rhs_N)$ where

$$
\begin{array}{lcl}
rhs_{\widehat{\sigma}}(\langle \widehat{s}, 0\rangle) & = & \langle s, 1\rangle \\
rhs_{\widehat{\sigma}}(\langle i, 1\rangle) & = & N \\
rhs_A(\langle s, 0\rangle) & = & A\ \langle s, 1\rangle \\
rhs_A(\langle i, 1\rangle) & = & A\ \langle i, 0\rangle \\
rhs_B(\langle s, 0\rangle) & = & B\ \langle s, 1\rangle \\
rhs_B(\langle i, 1\rangle) & = & B\ \langle i, 0\rangle \\
rhs_N(\langle s, 0\rangle) & = & \langle i, 0\rangle
\end{array}
$$

The attributed tree transducer M_{palin} transforms the input list (considered as word) into a palindrom by appending the reversed list (the reversed word) to the end of the input word. Since M_{palin} is a single-use attributed tree transducer, we can apply Construction 4.1.1, which yields the attributed tree transducer $M_{palin;palin} = (\Sigma, \Sigma, \overline{S}, \overline{I}, (\widehat{s}, \widehat{s}), \widehat{\sigma}, \overline{R})$ with $\overline{R} = (\overline{rhs_{\widehat{\sigma}}}, \overline{rhs_A}, \overline{rhs_B}, \overline{rhs_N})$,

$$\overline{S} = \{(s,s)\,,\,(i,i)\}, \qquad \overline{I} = \{(s,i)\,,\,(i,s)\}$$

and

$$
\begin{array}{lcl}
\overline{rhs_{\widehat{\sigma}}}(\langle (\widehat{s},\widehat{s}), 0\rangle) & = & \langle (s,s), 1\rangle \\
\overline{rhs_{\widehat{\sigma}}}(\langle (i,s), 1\rangle) & = & \langle (i,i), 1\rangle \\
\overline{rhs_{\widehat{\sigma}}}(\langle (s,i), 1\rangle) & = & N \\
\overline{rhs_A}(\langle (s,s), 0\rangle) & = & A\ \langle (s,s), 1\rangle \\
\overline{rhs_A}(\langle (i,i), 0\rangle) & = & A\ \langle (i,i), 1\rangle \\
\overline{rhs_A}(\langle (i,s), 1\rangle) & = & A\ \langle (i,s), 0\rangle \\
\overline{rhs_A}(\langle (s,i), 1\rangle) & = & A\ \langle (s,i), 0\rangle \\
\overline{rhs_B}(\langle (s,s), 0\rangle) & = & B\ \langle (s,s), 1\rangle \\
\overline{rhs_B}(\langle (i,i), 0\rangle) & = & B\ \langle (i,i), 1\rangle \\
\overline{rhs_B}(\langle (i,s), 1\rangle) & = & B\ \langle (i,s), 0\rangle \\
\overline{rhs_B}(\langle (s,i), 1\rangle) & = & B\ \langle (s,i), 0\rangle \\
\overline{rhs_N}(\langle (s,s), 0\rangle) & = & \langle (i,s), 0\rangle \\
\overline{rhs_N}(\langle (i,i), 0\rangle) & = & \langle (s,i), 0\rangle
\end{array}
$$

If we assume that the input list has length n, then the attributed tree transducer M_{palin} needs $2n + 3$ reduction steps to generate the output list of length $2n$. If we reconsider this list as the input of M_{palin}, then the number of reduction steps will be $4n + 3$ this time. To sum up, we need $6(n + 1)$ reduction steps to execute the sequential composition.

In contrast to that the attributed tree transducer $M_{palin;palin}$ applied to the same input list of length n, only needs $4n + 5$ reduction steps and consequently is more efficient. This example out of [Bor97] shows, that the composition result might be more efficient than the original sequential composition. □

4.1.3 Example ($M_{a-rev}; M_{a-rev}; M_{a-rev}$)
The attributed tree transducer M_{a-rev} defined in Example 2.4.3 is single-use and consequently we can compose it with itself using Construction 4.1.1. This yields an attributed tree transducer, which computes the identity in a rather long-winded way. The resulting attributed tree transducer $M_{a-rev\,;\,a-rev} = (\Sigma, \Sigma, \overline{S}, \overline{I}, (\widehat{s}, \widehat{s}), \widehat{\sigma}, \overline{R})$ with $\overline{R} = (\overline{rhs_{\widehat{\sigma}}}, \overline{rhs_A}, \overline{rhs_B}, \overline{rhs_N})$,

$$\overline{S} = \{(rev,rev)\,,\,(i,i)\}, \qquad \overline{I} = \{(rev,i)\,,\,(i,rev)\}$$

and

$$\begin{array}{lcl}
\overline{rhs}_{\widehat{\sigma}}(\langle(\widehat{s},\widehat{s}),0\rangle) & = & \langle(rev,rev),1\rangle \\
\overline{rhs}_{\widehat{\sigma}}(\langle(i,rev),1\rangle) & = & \langle(i,i),1\rangle \\
\overline{rhs}_{\widehat{\sigma}}(\langle(rev,i),1\rangle) & = & N \\[4pt]
\overline{rhs}_{A}(\langle(rev,rev),0\rangle) & = & \langle(rev,rev),1\rangle \\
\overline{rhs}_{A}(\langle(i,i),0\rangle) & = & A\ \langle(i,i),1\rangle \\
\overline{rhs}_{A}(\langle(i,rev),1\rangle) & = & \langle(i,rev),0\rangle \\
\overline{rhs}_{A}(\langle(rev,i),1\rangle) & = & \langle(rev,i),0\rangle \\[4pt]
\overline{rhs}_{B}(\langle(rev,rev),0\rangle) & = & \langle(rev,rev),1\rangle \\
\overline{rhs}_{B}(\langle(i,i),0\rangle) & = & B\ \langle(i,i),1\rangle \\
\overline{rhs}_{B}(\langle(i,rev),1\rangle) & = & \langle(i,rev),0\rangle \\
\overline{rhs}_{B}(\langle(rev,i),1\rangle) & = & \langle(rev,i),0\rangle \\[4pt]
\overline{rhs}_{N}(\langle(rev,rev),0\rangle) & = & \langle(i,rev),0\rangle \\
\overline{rhs}_{N}(\langle(i,i),0\rangle) & = & \langle(rev,i),0\rangle
\end{array}$$

If we again assume that the length of the input list is n, then the attributed tree transducer M_{a-rev} needs $2n+3$ reduction steps to generate the output list of length n this time. This output list is treated as the input list of another incarnation of M_{a-rev}, which then needs again $2n+3$ reduction steps to compute the final result. To sum up, we need $4n+6$ reduction steps to execute the sequential composition.

The composition result $M_{a-rev\,;\,a-rev}$ needs $4n+5$ reduction steps given an input list of length n. The composition result is again more efficient, but this time the difference is only marginal (exactly one reduction step) and applying Construction 4.1.1 to the attributed tree transducers $M_{a-rev\,;\,a-rev}$ and M_{a-rev} will yield an attributed tree transducer $M_{a-rev;a-rev;a-rev}$ which suffers from the explosion of attributes. Then things are reversed, so that the sequential composition needs only $6n+8$ reduction steps, while the composition result needs $8n+9$ reduction steps to compute the final result. These effects were also observed in [Bor97]. □

In [Gan83, Gie88] it was shown that the class of syntactic single-use attributed tree transducers ATT_{ssu} is closed with respect to composition. In [Küh97] this result was generalized to single-use attributed tree transducers.

$$\begin{array}{lclcll}
\mathrm{ATT}_{su} & ; & \mathrm{ATT}_{su} & \subseteq & \mathrm{ATT}_{su} & \text{[Küh97]} \\
\mathrm{ATT}_{ssu} & ; & \mathrm{ATT}_{lsu} & \subseteq & \mathrm{ATT}_{lsu} & \\
\mathrm{ATT}_{ssu} & ; & \mathrm{ATT}_{ssu} & \subseteq & \mathrm{ATT}_{ssu} & \text{[Gan83, Gie88]}
\end{array}$$

4.1.4 Lemma ($\mathbf{ATT}_{ssu}$; $\mathbf{ATT}_{lsu} \subseteq \mathbf{ATT}_{lsu}$)

$$ATT_{ssu}\,;\,ATT_{lsu} \subseteq ATT_{lsu}$$

Proof sketch: *The result is a straightforward generalization of the result presented in [Gan83]. The very same kind of reasoning that was applied there to show the composition result, also proves this result. The main difference is that for the second attributed tree transducer a unique right hand side containing a certain attribute occurrence no longer exists. But since the second attributed tree transducer is at least single-use there is at least one such right hand side. Consequently the second part of the proof in [Gan83] cannot be established, yielding that an outer attribute occurrence might appear several times in all right hand sides at some input symbol, but at least once.* □

4.2 Efficiency considerations

Let us assume, that we have two attributed tree transducers M_1 and M_2 along with an attributed tree transducer $M_{1;2}$, which represents the composition of M_1 and M_2. Then we want to relate efficiency (to be precise the number of reduction steps using a call-by-need derivation relation) of M_1 and M_2 to the efficiency of $M_{1;2}$ and this relation should be independent of the concrete input tree. Therefore we firstly define a helper function, namely a function which computes the size of a given tree.

4.2.1 Definition (size-function)
Let a set of terms $T_\Sigma(Y)$ be given (deducible from the context). Then the function size *shall compute the size of a term.*

$$\text{size} : \quad T_\Sigma(Y) \rightarrow N$$

$$\text{size}(t) = \sum_{\alpha \in \underline{\Sigma} \cup Y} |t|_\alpha$$

□

For at least single-use attributed tree transducers computing the number of reduction steps is particularly easy using the equation below. This is due to the fact that every attribute instance of a certain input tree is reduced.

4.2.2 Lemma ($\text{count}_{M, \overset{e}{\Rightarrow}_{M,t}}(t)$ for at least single-use attributed tree transducer)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be an at least single-use attributed tree transducer and furthermore let $t = (\widehat{\sigma}\, t')$, $t' \in T_\Sigma$ be an input tree. The number of reduction steps $\text{count}_{M, \overset{e}{\Rightarrow}_{M,t}}(t)$ can then be computed using the following equation.

$$\text{count}_{M, \overset{e}{\Rightarrow}_{M,t}}(t) = \text{size}(t')|F| + 1$$

Proof: Let $\sigma \in \underline{\Sigma}_+$ be arbitrary and let $p \in \text{paths}(t)$ such that $\text{label}_t(p) = \sigma$. In the first part, we prove that for every outer attribute occurrence $\chi \in \text{out}_M(\sigma)$ there exists a $s \in S$ such that

$$\langle s, p \rangle \rightarrow^+_{M,t} \text{relabel}_{M,p}(\chi).$$

This statement is proven by induction over trees. We will use nullary input symbols $\alpha \in \Sigma^{(0)}$ as induction basis. In this case the statement simplifies to

$$(\forall i \in I)(\exists s \in S) \quad \langle s, p \rangle \rightarrow^+_{M,t} \langle i, p \rangle,$$

since there are no successors of α. Given the at least single-use requirement we can conclude that every inherited attribute occurrence $\langle i, 0 \rangle$ appears at least once in some right hand side $rhs_{M,\alpha}(\psi)$. Consequently ψ needs to be an inner synthesized attribute occurrence, since other inner attribute occurrences do not exist. So setting $\langle s, p \rangle = \text{relabel}_{M,p}(\psi)$ proves the statement immediately.

Having done the induction basis, we will now prove the statement for an arbitrary $\sigma \in \underline{\Sigma}_+$, assuming that the property holds for every successor of σ. Therefore we select an arbitrary outer attribute occurrence χ, then by the at least single-use requirement we conclude that this attribute occurrence appears in some right hand side $rhs_{M,\sigma}(\psi)$. We can now perform a case analysis on ψ to get:

1. ψ is some inner synthesized attribute occurrence. To fulfill the given statement we set $\langle s, p \rangle = \text{relabel}_{M,p}(\psi)$.

2. ψ is an inherited attribute occurrence at some successor. Then by induction hypothesis for $\text{relabel}_{M,p}(\psi)$ we gain a synthesized attribute instance $\langle s', p.j\rangle$, which depends on $\text{relabel}_{M,p}(\psi)$, such that $\chi' = \langle s', j\rangle$ is a synthesized attribute occurrence at exactly this successor of σ. This occurrence is again an outer attribute occurrence of σ, so consequently we perform the same kind of reasoning (including the case-analysis) again, knowing that this process must eventually lead to case (1), since the input attributed tree transducer is non-circular and there are only finitely many attribute occurrences.

The proven statement shows, that if every synthesized attribute instance $\langle s, p\rangle$ with $s \in S$ and $\text{label}_t(p) = \sigma$ is needed, then all attribute instances $\langle s', p.j\rangle$ and if $p \neq \varepsilon$ then also all attribute instances $\langle i, p\rangle$ with $s' \in S$, $i \in I$ and $j \in [\text{rank}_\Sigma(\sigma)]$ are needed as well. Furthermore, since the distinguished attribute instance $\langle \widehat{s}, \varepsilon\rangle$ is used during reduction, all attribute instances occur in the reduction. Together with the property that every attribute instance is evaluated exactly once (Lemma 3.2.5), the above statement is proven. □

4.2.3 Example (M_{a-rev} and Lemma 4.2.2)
If we reconsider the input tree $t = (\widehat{\sigma}\, ABBN)$ and the attributed tree transducer $M = M_{a-rev}$ of Example 2.4.3 which is at least single-use, then we can compute the number of reduction steps using Lemma 4.2.2. This yields

$$\text{count}_{M,\overset{e}{\Rightarrow}_{M,t}}(\widehat{\sigma}\, ABBN) = \text{size}(ABBN)|F| + 1,$$

where $\text{size}(ABBN) = 4$ and $|F| = 2$. So finally $\text{count}_{M,\overset{e}{\Rightarrow}_{M,t}}(t) = 4 * 2 + 1 = 9$, which is exactly the result we gained when reducing the term into its normal form in Example 3.3.6. □

Certainly it is easy to compute the required number of reduction steps given the concrete input tree, since the number of required attribute instances can easily be determined. But since the concrete input tree will not be known at compile time, the efficiency considerations should be independent of the actual input tree.

At this stage we opt for the composition hierarchy $\text{ATT}_{ssu}\, ;\, \text{ATT}_{ssu} \subseteq \text{ATT}_{ssu}$. On the one hand side it is easy to determine the number of reduction steps using Lemma 4.2.2 given a syntactic single-use attributed tree transducer, but on the other hand side this is quite a severe restriction. Effectively the evaluation strategies call-by-need and call-by-name coincide for single-use attributed tree transducers.

In chapter five we will show, why it is important for our considerations that the resulting attributed tree transducer is single-use. One might argue that our call-by-need derivation relation $\overset{e}{\Rightarrow}_{M,t}$ of an attributed tree transducer M and an input tree t, does not accurately model a call-by-need evaluation strategy, if the attributed tree transducer M is not weakly single-use, but this not the reason, since it is rather easy to develop a derivation relation which accurately models a call-by-need evaluation strategy.

If we restrict the attributed tree transducers to be syntactic single-use, then consequently the number of reduction steps only depends on the size of the input tree and the amount of attributes of an attributed tree transducer. If we reconsider the result of Lemma 4.2.2, then we gain the following Theorem.

4.2.4 Theorem (composing syntactic single-use attributed tree transducers)
Let $M_1 = (\Sigma, \Omega, S_1, I_1, \widehat{s}_1, \widehat{\sigma}, R_1)$ and $M_2 = (\Omega, \Delta, S_2, I_2, \widehat{s}_2, \widehat{\omega}, R_2)$ be syntactic single-use attributed tree transducers. Furthermore let $t_1 \in T_\Sigma$ and

$t_2 \in T_\Omega$, **such that** $t_2 = nf_{\Rightarrow_{M_1,(\widehat{\sigma}\, t_1)}}(\langle \widehat{s}_1, \varepsilon\rangle)$. **Then the syntactic single-use attributed tree transducer** $M_{1;2}$**, which results out of Construction 4.1.1 using** M_1 **and** M_2 **as input, is more efficient with respect to the plain number of reduction steps (using a call-by-need evaluation strategy), if and only if**

$$\mathrm{size}(t_2)|F_2| \;>\; \mathrm{size}(t_1)|F_1|(|F_2|-1)-1,$$

where $F_1 = S_1 \cup I_1$ **and** $F_2 = S_2 \cup I_2$**.**

Proof: Let $t_1' = (\widehat{\sigma}\, t_1)$ and $t_2' = (\widehat{\omega}\, t_2)$. According to Lemma 4.2.2 we get:

$$\mathrm{count}_{M_1, \overset{\varepsilon}{\Rightarrow}_{M_1,t_1'}}(t_1') = \mathrm{size}(t_1)|F_1| + 1$$

$$\mathrm{count}_{M_2, \overset{\varepsilon}{\Rightarrow}_{M_2,t_2'}}(t_2') = \mathrm{size}(t_2)|F_2| + 1$$

Consequently the number of reduction steps of the sequential composition of M_1 and M_2 is

$$\mathrm{count}_{M_1, \overset{\varepsilon}{\Rightarrow}_{M_1,t_1'}}(t_1') + \mathrm{count}_{M_2, \overset{\varepsilon}{\Rightarrow}_{M_2,t_2'}}(t_2') = \mathrm{size}(t_1)|F_1| + \mathrm{size}(t_2)|F_2| + 2.$$

The attributed tree transducer $M_{1;2}$, which is the result of Construction 4.1.1, has $|S_1||S_2| + |I_1||I_2|$ synthesized and $|S_1||I_2| + |I_1||S_2|$ inherited attributes. The input tree for the attributed tree transducer $M_{1;2}$ is t_1' and therefore

$$\begin{aligned} \mathrm{count}_{M_{1;2}, \overset{\varepsilon}{\Rightarrow}_{M_{1;2},t_1'}}(t_1') &= \mathrm{size}(t_1)(|S_1||S_2| + |I_1||I_2| + |S_1||I_2| + |I_1||S_2|) + 1 \\ &= \mathrm{size}(t_1)|F_1||F_2| + 1. \end{aligned}$$

To ascertain efficiency we let

$$\mathrm{count}_{M_{1;2}, \overset{\varepsilon}{\Rightarrow}_{M_{1;2},t_1'}}(t_1') < \mathrm{count}_{M_1, \overset{\varepsilon}{\Rightarrow}_{M_1,t_1'}}(t_1') + \mathrm{count}_{M_2, \overset{\varepsilon}{\Rightarrow}_{M_2,t_2'}}(t_2')$$

and consequently (by simple arithmetics)

$$\begin{array}{rrcl} & \mathrm{size}(t_1)|F_1||F_2| + 1 & < & \mathrm{size}(t_1)|F_1| + \mathrm{size}(t_2)|F_2| + 2 \\ \Longleftrightarrow & \mathrm{size}(t_1)(|F_1||F_2| - |F_1|) & < & \mathrm{size}(t_2)|F_2| + 1 \\ \Longleftrightarrow & \mathrm{size}(t_2)|F_2| & > & \mathrm{size}(t_1)|F_1|(|F_2|-1) - 1. \end{array}$$

□

The difference $\mathrm{size}(t_2)|F_2| - \mathrm{size}(t_1)|F_1|(|F_2|-1)+1$ exactly characterizes the number of reduction steps that were cut down by the composition result. Consequently if this difference is positive, then the attributed tree transducer $M_{1;2}$ should be prefered, whereas if the difference is negative, then the attributed tree transducer $M_{1;2}$ should be avoided, since it introduces inefficiencies. In case the difference is actually zero, then $M_{1;2}$ should be applied, since it avoids the computation of an intermediate result.

The last representation was favoured, since it evidently shows that an increase of the size of the intermediate result also increases the above mentioned difference. So obviously the size of the intermediate result seems to play an important role. Still this result is not fully satisfactory, because actually the decision cannot be drawn independent of the input tree (e.g. the size of the input tree is needed). This matter will be discussed in chapter five, when we put all the pieces together.

In the following we will present some examples illustrating the Theorem 4.2.4 and justifying the efficiency considerations of the previous section.

4.2.5 Example (M_{palin} ; M_{palin})
Let M_{palin} be the attributed tree transducer defined in Example 4.1.2. M_{palin} is syntactic single-use, consequently Construction 4.1.1 and Theorem 4.2.4 are applicable. Since $S = \{s\}$, $I = \{i\}$ and for arbitrary input trees $(\widehat{\sigma}\, t_1)$ with $t_1 \in T_\Sigma$ $\mathrm{size}(nf_{\Rightarrow_{M_{a-rev}},(\widehat{\sigma}\, t)}(\langle \widehat{s}, \varepsilon\rangle)) = 2\,\mathrm{size}(t_1) - 1 = 2n - 1$, we can conclude that

$$2(2n-1) > 2n(2-1) - 1.$$

So consequently $M_{palin\,;\,palin}$ is more efficient than the original sequential composition as stated in the previous section. The performance gain is exactly $2n - 1$ reduction steps. □

4.2.6 Example (M_{a-rev} ; M_{a-rev})
Let M_{a-rev} be the attributed tree transducer defined in Example 2.4.3. We already stated that M_{a-rev} is syntactic single-use. Consequently Construction 4.1.1 and Theorem 4.2.4 are again applicable. Since $S = \{rev\}$, $I = \{i\}$ and for arbitrary input trees $(\widehat{\sigma}\, t_1)$ with $t_1 \in T_\Sigma$ $\mathrm{size}(nf_{\Rightarrow_{M_{a-rev}},(\widehat{\sigma}\, t)}(\langle \widehat{s}, \varepsilon\rangle)) = \mathrm{size}(t_1) = n$, we can deduce

$$2n > 2n(2-1) - 1.$$

Again $M_{a-rev\,;\,a-rev}$ is more efficient than the original sequential composition. The performance benefit is exactly 1 reduction step. □

4.2.7 Example (M_{a-rev} ; M_{a-rev} ; M_{a-rev})
Let M_{a-rev} be the attributed tree transducer defined in Example 2.4.3. The composition result $M_{a-rev;a-rev;a-rev}$ of M_{a-rev} and $M_{a-rev\,;\,a-rev}$ is less efficient than the sequential composition, since

$$4n > 2n(4-1) - 1.$$

The performance loss is characterized by exactly $2n - 1$ reduction steps. □

At this stage a construction named *copy-rules elimination* can be applied to the composition result to eliminate the so-called *copy-rules*. The complete construction can be found in [CDPR98]. A copy-rule is a rule, which does not produce output symbols. Apparently the elimination of the copy-rules can seriously alter an attributed tree transducer.

In order to benefit from this construction we need to compute the composition result and then apply the copy-rules elimination to it. This yields another attributed tree transducer and we can then administer our performance analysis using this optimized attributed tree transducer. This is possible since the resulting attributed tree transducer will be syntactic single-use, whenever the inputted attributed tree transducer is syntactic single-use. We will demonstrate this using $M_{a-rev\,;\,a-rev}$.

4.2.8 Example (copy-rules elimination)
Let $M_{a-rev\,;\,a-rev}$ be the attributed tree transducer defined in Example 4.1.3 and furthermore let M' be the resulting attributed tree transducer, when copy-rules elimination [CDPR98] is applied to $M_{a-rev\,;\,a-rev}$. Then

$$M' = (\{A^{(1)}, B^{(1)}, N^{(0)}\}, \{A^{(1)}, B^{(1)}, N^{(0)}\}, \{(i,i)\}, \emptyset, \widehat{s}, \widehat{\sigma}, R')$$

with $R' = (rhs_{M',\widehat{\sigma}}, rhs_{M',A}, rhs_{M',B}, rhs_{M',N})$ and right hand side functions as follows:

$$\begin{array}{lcl} rhs_{M',\widehat{\sigma}}(\langle \widehat{s}, 0\rangle) & = & \langle (i,i), 1\rangle \\ rhs_{M',A}(\langle (i,i), 0\rangle) & = & A\,\langle (i,i), 1\rangle \\ rhs_{M',B}(\langle (i,i), 0\rangle) & = & B\,\langle (i,i), 1\rangle \\ rhs_{M',N}(\langle (i,i), 0\rangle) & = & N. \end{array}$$

We use a partial result out of the proof of Theorem 4.2.4 to perform an efficiency analysis:

$$\text{size}(t_1)|F'| + 1 \; < \; \text{size}(t_1)|F_1| + \text{size}(t_2)|F_2| + 2,$$

where in our running example $F_1 = F_2$ (they both correspond to M_{a-rev}) and $F' = \{(i,i)\}$ corresponds to the optimized attributed tree transducer M'. Again let $n = \text{size}(t_1)$, then

$$n + 1 \; < \; 2n + 2n + 2$$

and clearly M' is much more efficient, since we managed to cut down $3n+1$ reduction steps. □

We saw that copy-rules elimination might greatly improve the performance of the composition result. Although the attributed tree transducer $M_{1;2}$ is less efficient than the sequential composition of the attributed tree transducers M_1 and M_2, it might happen that the attributed tree transducer M' with copy-rules eliminated outperforms the sequential composition. This happens, for example, with $M_{a-rev;a-rev;a-rev}$ of Example 4.1.3.

Chapter 5

Back to macro tree transducers

5.1 The construction

Within this section we will present the last construction needed to transform the gained attributed tree transducer back to a macro tree transducer. Therefore we will state the construction of [Küh00a], based on [FZ82].

5.1.1 Construction (ATT $\subseteq$ MAC)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be an attributed tree transducer with $q = |I|$ and $I = \{i_1, \ldots, i_q\}$. Furthermore let $\alpha \in \Delta^{(0)}$ be an arbitrary output symbol. We construct a macro tree transducer $M' = (\Sigma, \Delta, F, e, R')$ with $\tau(M') = \tau(M)$ where

- $F = \{\, s^{(q+1)} \mid s \in S \,\}$,
- $e = trans_{\widehat{\sigma},\emptyset}\big(rhs_M(\langle \widehat{s}, 0\rangle, \widehat{\sigma}\big)$ and
- for every $k \in N$, $\sigma \in \Sigma^{(k)}$ and $s \in S$ the rule-set R' contains the rule

$$s\,(\sigma\, x_1 \ldots x_k)\, y_1 \ldots y_q = \mathrm{trans}_{\sigma,\emptyset}\big(rhs_M(\langle s, 0\rangle, \sigma)\big).$$

For every $k \in N, \sigma \in \Sigma_+^{(k)}$ and $V \subseteq \mathrm{out}_M(\sigma)$ we define the function

$$\mathrm{trans}_{\sigma,V} \;:\; T_\Delta\big(\mathrm{out}_M(\sigma)\big) \;\rightarrow\; \mathrm{RHS}_{\Delta,F,k,q}$$

- for every $l \in [q]$
  $\mathrm{trans}_{\sigma,V}(\langle i_l, 0\rangle) \quad = \quad y_l$
- for every $n \in N$, $\delta \in \Delta^{(n)}$ and $\varrho_1 \ldots \varrho_n \in T_\Delta\big(\mathrm{out}_M(\sigma)\big)$
  $\mathrm{trans}_{\sigma,V}(\delta\,\varrho_1 \ldots \varrho_n) \quad = \quad \delta\,\big(\mathrm{trans}_{\sigma,V}(\varrho_1)\big) \ldots \big(\mathrm{trans}_{\sigma,V}(\varrho_n)\big)$
- for every $s \in S$ and $j \in [k]$
  $$\mathrm{trans}_{\sigma,V}(\langle s, j\rangle) \quad = \quad \begin{cases} \alpha & \text{if } \langle s, j\rangle \in V \\ s\,x_j\,\psi_1 \ldots \psi_q & \text{otherwise} \end{cases}$$
  where $\psi_n = \mathrm{trans}_{\sigma,V\cup\{\langle s,j\rangle\}}\big(rhs_M(\langle i_n, j\rangle, \sigma)\big)$ for every $n \in [q]$.

□

Having stated the construction we will now turn to an example. We will use the attributed tree transducer $M_{a-rev\,;\,a-rev}$ of Example 4.1.3.

5.1.2 Example ($M_{rev;rev}$)
Let the attributed tree transducer $M_{a-rev\,;\,a-rev}$ be given as in Example 4.1.3. Then the result of applying Construction 5.1.1 to this attributed tree transducer is the macro tree transducer $M_{rev;rev} = (\Sigma, \Delta, F, e, R)$ with

- $\Sigma = \Delta = \{A^{(1)}\,,\, B^{(0)}\,,\, N^{(0)}\}$,
- $F = \{(rev, rev)^{(3)}\,,\, (i,i)^{(3)}\}$,
- $e = (rev, rev)\, x_1\, \big((i,i)\, x_1\, N\, N\big)\, N$ and

$$\begin{array}{llll} R = \{ & (rev,rev)\,(A\,x_1)\,y_1\,y_2 & = & (rev,rev)\,x_1\,y_1\,y_2\;, \\ & (rev,rev)\,(B\,x_1)\,y_1\,y_2 & = & (rev,rev)\,x_1\,y_1\,y_2\;, \\ & (rev,rev)\,N\,y_1\,y_2 & = & y_1\;, \\ & (i,i)\,(A\,x_1)\,y_1\,y_2 & = & A\,\big((i,i)\,x_1\,y_1\,y_2\big)\;, \\ & (i,i)\,(B\,x_1)\,y_1\,y_2 & = & B\,\big((i,i)\,x_1\,y_1\,y_2\big)\;, \\ & (i,i)\,N\,y_1\,y_2 & = & y_2 \qquad\qquad \}. \end{array}$$

□

As we can see in this example the gained macro tree transducer is still far from being optimal for computing the identity. This is where optimizations like *removal of superflous context parameters* and *copy state elimination* step in. Both techniques are introduced in [Voi01]. These techniques might remove function symbols completely (copy state elimination) or change their arity (removal of superflous context parameters). Similar to copy-rule elimination we simply apply the construction and consider efficiency again afterwards.

5.1.3 Example ($M'_{rev;rev}$)
Let the macro tree transducer $M_{rev;rev}$ be given as in the previous Example. Then the result of applying removal of superflous context parameters and copy state elimination will be the macro tree transducer $M'_{rev;rev} = (\Sigma, \Delta, F, e, R)$ with

- $\Sigma = \Delta = \{A^{(1)}\,,\, B^{(0)}\,,\, N^{(0)}\}$,
- $F = \{(i,i)^{(2)}\}$,
- $e = (i,i)\, x_1\, N\, N$ and

$$\begin{array}{llll} R = \{ & (i,i)\,(A\,x_1)\,y_1 & = & A\,\big((i,i)\,x_1\,y_1\big)\;, \\ & (i,i)\,(B\,x_1)\,y_1 & = & B\,\big((i,i)\,x_1\,y_1\big)\;, \\ & (i,i)\,N\,y_1 & = & y_1 \qquad\qquad \}. \end{array}$$

The state (function symbol) (rev, rev) was eliminated, since it only projected on its first context parameter, and the first context parameter (originally) y_1 was removed, since it is never outputted. □

5.2 Efficiency considerations

The final step is to relate the efficiency of the composed attributed tree transducer to the efficiency of the macro tree transducer gained by Construction 5.1.1. Since we restricted ourselves to syntactic single-use attributed tree transducers in Chapter four, our main focus will be the call-by-name evaluation strategy. Since call-by-need and call-by-name coincide for single-use macro tree transducers, we do not need to consider them separately.

5.2.1 Theorem (relating the number of derivation steps)
Let $M = (\Sigma, \Delta, S, I, \widehat{s}, \widehat{\sigma}, R)$ be a single-use attributed tree transducer and let $(\widehat{\sigma}\, t)$ with $t \in T_\Sigma$ be an input tree. Furthermore let $M' = (\Sigma, \Delta, F, e, R')$ be the macro tree transducer resulting from Construction 5.1.1 using M as input. Then

$$\mathrm{count}_{M',\Rightarrow_{M'}}(t) = \mathrm{count}_{M,\Rightarrow_{M,t}}(t) - \imath_{M,\Rightarrow_{M,t}}(t) - 1$$

where $\imath_{M,\Rightarrow_{M,t}}(t)$ denotes the number of reduction steps invested to reduce inherited attribute instances (during computation of the normal form).

Proof: In order to prove this result, we will again make use of the property, that the macro tree transducer M' defines the same translation as M. The proof is very similar to the one found in Theorem 3.3.5. Firstly we select an arbitrary output symbol $\diamond^{(1)} \notin \Delta$ and we define a new attributed tree transducer $\overline{M} = (\Sigma, \Delta \cup \{\diamond^{(1)}\}, S, I, \widehat{s}, \widehat{\sigma}, \overline{R})$, which is only a slight modification of M with $\overline{R} = (\overline{rhs}_{\overline{M},\sigma} \mid \sigma \in \underline{\Sigma}_+)$, such that for every $\sigma \in \underline{\Sigma}_+$ and $\xi \in \mathrm{in}_{\overline{M}}(\sigma)$ the right hand side function $\overline{rhs}_{\overline{M},\sigma}$ is defined to be

$$\overline{rhs}_{\overline{M},\sigma}(\xi) = \begin{cases} \diamond\left(rhs_{M,\sigma}(\xi)\right) & \text{, if } \xi \in Fun_{S,\{0\}} \\ rhs_{M,\sigma}(\xi) & \text{, otherwise} \end{cases} .$$

This yields a well-defined attributed tree transducer and we can immediately conclude that

$$\mathrm{count}_{M,\Rightarrow_{M,t}}(t) = \mathrm{count}_{\overline{M},\Rightarrow_{\overline{M},t}}(t) \qquad \text{and} \qquad \imath_{M,\Rightarrow_{M,t}}(t) = \jmath_{\overline{M},\Rightarrow_{\overline{M},t}}(t),$$

where $\jmath_{\overline{M},\Rightarrow_{\overline{M},t}}(t)$ is exactly the number of reduction steps invested to reduce inherited attribute instances of t using $\overline{M}$. Consequently

$$\mathrm{count}_{M,\Rightarrow_{M,t}}(t) - \imath_{M,\Rightarrow_{M,t}}(t) = \mathrm{count}_{\overline{M},\Rightarrow_{\overline{M},t}}(t) - \jmath_{\overline{M},\Rightarrow_{\overline{M},t}}(t) = |nf_{\Rightarrow_{\overline{M},t}}(\langle\widehat{s},\varepsilon\rangle)|_\diamond + 1$$

This result is due to the strong resemblance of the attributed tree transducers M and $\overline{M}$. They effectively exhibit the same behaviour just that the attributed tree transducer $\overline{M}$ outputs the symbol $\diamond$ every time a synthesized attribute instance is reduced. So consequently the number of $\diamond$ symbols in the normal form represents the number of reduced synthesized attributes instances. Since the original attributed tree transducer M is single-use, the attributed tree transducer $\overline{M}$ is single-use as well. This effectively excludes that the substitution affects more than one attribute instance, hence the number of $\diamond$ symbols corresponds to the number of reduction steps invested to reduce synthesized attribute instances.

Similar to the proof of Theorem 3.3.5 we will apply Construction 5.1.1 to $\overline{M}$ to gain a macro tree transducer $\overline{M}' = (\Sigma, \Delta \cup \{\diamond^{(1)}\}, F, e, \overline{R}')$, where for every $k, n \in N$, $\sigma \in \Sigma^{(k)}$ and every $f \in F^{(n+1)}$ the rule-set $\overline{R}'$ contains the rule

$$f\,(\sigma\, x_1 \ldots x_k)\, y_1 \ldots y_n = \diamond\,(rhs_{M',\sigma,f}).$$

Again these rules (as opposed to the original rules of M') output the special output symbol $\diamond$ and afterwards behave like the original rule does. Then obviously the number of reduction steps is

$$\mathrm{count}_{M',\Rightarrow_{M'}}(t) = \mathrm{count}_{\overline{M}',\Rightarrow_{\overline{M}'}}(t) = |nf_{\Rightarrow_{\overline{M}'}}(e[x_1/t])|_\diamond,$$

since the output symbol $\diamond$, which the macro tree transducer $\overline{M}'$ outputs at every reduction step, is not counted and sharing of context parameters cannot appear.

Although the macro tree transducers are not necessarily non-copying, sharing cannot occur. If a context parameter occurs several times in some right hand side, then only one appearance will be evaluated (reached). This is due to the fact that the attributed tree transducer was single-use.

Due to Construction 5.1.1 the induced translations of $\overline{M}$ and $\overline{M}'$ are equal (i.e. $\tau(\overline{M}) = \tau(\overline{M}'))$ and also $nf_{\Rightarrow_{\overline{M},t}}(\langle\widehat{s},\varepsilon\rangle) = nf_{\Rightarrow_{\overline{M}'}}(e[x_1/t])$. We conclude

$$|nf_{\Rightarrow_{\overline{M}'}}(e[x_1/t])|_\diamond = |nf_{\Rightarrow_{\overline{M},t}}(\langle\widehat{s},\varepsilon\rangle)|_\diamond$$

and finally

$$\mathrm{count}_{M',\Rightarrow_{M'}}(t) = \mathrm{count}_{M,\Rightarrow_{M,t}}(t) - \imath_{M,\Rightarrow_{M,t}}(t) - 1.$$

□

After having established this relation, we will give an example.

5.2.2 Example (relating the number of reduction steps)
Let $M = M_{rev;rev}$ of Example 5.1.2. Furthermore let $t = ABBN$ be the input tree. We demonstrate the step-wise reduction using the call-by-name derivation relation.

$$\begin{array}{ll}
 & (rev,rev)\,(ABBN)\,\big((i,i)\,(ABBN)\;N\,N\big)\;N \\
\Rightarrow_M & (rev,rev)\,(BBN)\,\big((i,i)\,(ABBN)\;N\,N\big)\;N \\
\Rightarrow_M & (rev,rev)\,(BN)\,\big((i,i)\,(ABBN)\;N\,N\big)\;N \\
\Rightarrow_M & (rev,rev)\,N\,\big((i,i)\,(ABBN)\;N\,N\big)\;N \\
\Rightarrow_M & (i,i)\,(ABBN)\;N\,N \\
\Rightarrow_M & A\,\big((i,i)\,(BBN)\;N\,N\big) \\
\Rightarrow_M & AB\,\big((i,i)\,(BN)\;N\,N\big) \\
\Rightarrow_M & ABB\,\big((i,i)\;N\;N\,N\big) \\
\Rightarrow_M & ABBN
\end{array}$$

So effectively we needed eight reduction steps to compute the normal form. In order to compare this result, we demonstrate the reduction process using the attributed tree transducer $M = M_{a-rev\,;\,a-rev}$ of Example 4.1.3.

$$\begin{array}{llll}
 & \langle(\widehat{s},\widehat{s}),\varepsilon\rangle & \Rightarrow_{M,(\widehat{\sigma}\,t)} & \langle(rev,rev),1\rangle \\
\Rightarrow_{M,(\widehat{\sigma}\,t)} & \langle(rev,rev),1.1\rangle & \Rightarrow_{M,(\widehat{\sigma}\,t)} & \langle(rev,rev),1.1.1\rangle \\
\Rightarrow_{M,(\widehat{\sigma}\,t)} & \langle(rev,rev),1.1.1.1\rangle & \Rightarrow_{M,(\widehat{\sigma}\,t)} & \langle(i,rev),1.1.1.1\rangle \\
\Rightarrow_{M,(\widehat{\sigma}\,t)} & \langle(i,rev),1.1.1\rangle & \Rightarrow_{M,(\widehat{\sigma}\,t)} & \langle(i,rev),1.1\rangle \\
\Rightarrow_{M,(\widehat{\sigma}\,t)} & \langle(i,rev),1\rangle & \Rightarrow_{M,(\widehat{\sigma}\,t)} & \langle(i,i),1\rangle \\
\Rightarrow_{M,(\widehat{\sigma}\,t)} & A\,\langle(i,i),1.1\rangle & \Rightarrow_{M,(\widehat{\sigma}\,t)} & AB\,\langle(i,i),1.1.1\rangle \\
\Rightarrow_{M,(\widehat{\sigma}\,t)} & ABB\,\langle(i,i),1.1.1.1\rangle & \Rightarrow_{M,(\widehat{\sigma}\,t)} & ABB\,\langle(rev,i),1.1.1.1\rangle \\
\Rightarrow_{M,(\widehat{\sigma}\,t)} & ABB\,\langle(rev,i),1.1.1\rangle & \Rightarrow_{M,(\widehat{\sigma}\,t)} & ABB\,\langle(rev,i),1.1\rangle \\
\Rightarrow_{M,(\widehat{\sigma}\,t)} & ABB\,\langle(rev,i),1\rangle & \Rightarrow_{M,(\widehat{\sigma}\,t)} & ABBN
\end{array}$$

The attributed tree transducer needs 17 reduction steps in total, nine of which are used to reduce synthesized attribute instances. So the given relation holds for this example. □

We postponed the problem concerning copying macro tree transducers. This example shall illustrate the effects that might occur.

5.2.3 Example (copying macro tree transducer)
Let $M_1 = M_{rev}$ of Example 2.3.3 and $M_2 = (\Sigma, \Delta, F, e, R)$ be macro tree transducers with

- $\Sigma = \{A^{(1)}, B^{(1)}, N^{(0)}\}$,
- $\Delta = \{\alpha^{(0)}, \sigma^{(2)}\}$,
- $F = \{s^{(2)}\}$,
- $e = (s\, x_1\, \alpha)$ and

$$R = \{ \begin{array}{lcl} s\,(A\,x_1)\,y_1 & = & \sigma\,(s\,x_1\,y_1)\,y_1\ , \\ s\,(B\,x_1)\,y_1 & = & s\,x_1\,y_1\ , \\ s\,N\,y_1 & = & y_1 \end{array} \}.$$

Although the macro tree transducer M_2 has many beneficial syntactic properties (syntactic single-use and non-deleting in context parameters), it is not non-copying and therefore also not single-use. The macro tree transducer M' shall be result of our series of transformations. We will only state the final result:

$$M' = (\Sigma, \Delta, \{(r,s), (y_r, y_s)\}, e', R')$$

with $e' = (r,s)\,x_1\,((y_r,y_s)\,x_1\,\alpha\,\alpha)\,\alpha$ and

$$R' = \{ \begin{array}{lcl} (r,s)\,(A\,x_1)\,y_1\,y_2 & = & (r,s)\,x_1\,(\sigma\,y_1\,((y_r,y_s)\,x_1\,(\sigma\,y_1\,\alpha)\,y_2))\,y_2\ , \\ (r,s)\,(B\,x_1)\,y_1\,y_2 & = & (r,s)\,x_1\,y_1\,y_2\ , \\ (r,s)\,N\,y_1\,y_2 & = & y_1\ , \\ (y_r,y_s)\,(A\,x_1)\,y_1\,y_2 & = & (y_r,y_s)\,x_1\,(\sigma\,y_1\alpha)\,y_2\ , \\ (y_r,y_s)\,(B\,x_1)\,y_1\,y_2 & = & (y_r,y_s)\,x_1\,y_1\,y_2\ , \\ (y_r,y_s)\,N\,y_1\,y_2 & = & y_2 \end{array} \}$$

Let $t_1 = BBN$ be an input tree. We assume that a call-by-need derivation relation computes the results. It is easy to see that M_1 needs three reduction steps to compute $\tau(M_1)(t_1) = BBN$ and M_2 needs another three reduction steps to compute $\tau(M_2)(BBN) = \alpha$. In total the sequential composition needs six reduction steps. Next we consider the composition result:

$$\begin{array}{ll} & (r,s)\,(BBN)\,((y_r,y_s)\,(BBN)\,\alpha\,\alpha)\,\alpha \\ \stackrel{e}{\Rightarrow}_{M'} & (r,s)\,(BN)((y_r,y_s)\,(BBN)\,\alpha\,\alpha)\,\alpha \\ \stackrel{e}{\Rightarrow}_{M'} & (r,s)\,N((y_r,y_s)\,(BBN)\,\alpha\,\alpha)\,\alpha \\ \stackrel{e}{\Rightarrow}_{M'} & (y_r,y_s)\,(BBN)\,\alpha\,\alpha \\ \stackrel{e}{\Rightarrow}_{M'} & (y_r,y_s)\,(BN)\,\alpha\,\alpha \\ \stackrel{e}{\Rightarrow}_{M'} & (y_r,y_s)\,N\,\alpha\,\alpha \\ \stackrel{e}{\Rightarrow}_{M'} & \alpha \end{array}$$

The composition result also needs six reduction steps to compute the normal form. Let $t_2 = ABN$ be an input tree of the same size, then M_1 needs again three reduction steps to compute $\tau(M_1)(t_2) = BAN$. M_2 also needs three reduction steps to compute $\tau(M_2)(BAN) = (\sigma\,\alpha\,\alpha)$. So the sequential composition again needs six reduction steps. In general the number of reduction steps for the sequential composition only depends on the size of the input tree. Now let us consider M' (shared redeces are underlined):

$$
\begin{array}{ll}
 & (r,s)\,(ABN)\,((y_r,y_s)\,(ABN)\,\alpha\,\alpha)\,\alpha \\
\stackrel{e}{\Rightarrow}_{M'} & (r,s)\,(BN)\,(\sigma\,(\underline{(y_r,y_s)\,(ABN)\,\alpha\,\alpha})\,((y_r,y_s)\,(BN) \\
 & \quad (\sigma\,(\underline{(y_r,y_s)\,(ABN)\,\alpha\,\alpha})\,\alpha)\,\alpha))\alpha \\
\stackrel{e}{\Rightarrow}_{M'} & (r,s)\,N\,(\sigma\,(\underline{(y_r,y_s)\,(ABN)\,\alpha\,\alpha})\,((y_r,y_s)\,(BN) \\
 & \quad (\sigma\,(\underline{(y_r,y_s)\,(ABN)\,\alpha\,\alpha})\,\alpha)\,\alpha))\alpha \\
\stackrel{e}{\Rightarrow}_{M'} & \sigma\,(\underline{(y_r,y_s)\,(ABN)\,\alpha\,\alpha})\,((y_r,y_s)\,(BN) \\
 & \quad (\sigma\,(\underline{(y_r,y_s)\,(ABN)\,\alpha\,\alpha})\,\alpha)\,\alpha) \\
\stackrel{e}{\Rightarrow}_{M'} & \sigma\,(\underline{(y_r,y_s)\,(BN)\,(\sigma\,\alpha\,\alpha)\,\alpha})\,((y_r,y_s)\,(BN)\,(\sigma\,(\underline{(y_r,y_s)\,(BN)\,(\sigma\,\alpha\,\alpha)\,\alpha})\,\alpha)\,\alpha) \\
\stackrel{e}{\Rightarrow}_{M'} & \sigma\,(\underline{(y_r,y_s)\,N\,(\sigma\,\alpha\,\alpha)\,\alpha})\,((y_r,y_s)\,(BN)\,(\sigma\,(\underline{(y_r,y_s)\,N\,(\sigma\,\alpha\,\alpha)\,\alpha})\,\alpha)\,\alpha) \\
\stackrel{e}{\Rightarrow}_{M'} & \sigma\,\alpha\,((y_r,y_s)\,(BN)\,(\sigma\,\alpha\,\alpha)\,\alpha) \\
\stackrel{e}{\Rightarrow}_{M'} & \sigma\,\alpha\,((y_r,y_s)\,N\,(\sigma\,\alpha\,\alpha)\,\alpha) \\
\stackrel{e}{\Rightarrow}_{M'} & \sigma\,\alpha\,\alpha
\end{array}
$$

Surprisingly the macro tree transducer M' needs eigth reduction steps. So we see that the number of reduction steps (for M') depends on the position of the A in the input tree. If $t_3 = (BAN)$ is the input tree, then M' needs seven reduction steps, whereas the sequential composition still needs only six. □

This is mainly due to Construction 4.1.1, where two former inherited attributes i_1, i_2 become a synthesized attribute (i_1, i_2). The problem is that those inherited attributes correspond to context parameters, which can be shared, whereas the synthesized attribute corresponds to a function call and will not be shared. Specifically the non-shared (y_r, y_s)-call is exactly the source of this problem.

5.3 Putting the pieces together

Let M_1 and M_2 be syntactic single-use, preserving in context parameters macro tree transducers. Then by Construction 3.1.1 we gain attributed tree transducers M_1' and M_2', which will both be syntactic single-use. If M_1 needs k_1 reduction steps to compute the normal form given an arbitrary input tree, then $k_1 + 1$ synthesized attribute instances are reduced in the process of computing the same normal form using M_1' (cf. Theorem 3.3.5). The same applies of course for M_2 and M_2'. Using M_1' and M_2' as input for Construction 4.1.1 yields another syntactic single-use attributed tree transducer M', which induces the same translation like $M_1'; M_2'$ and thereby $M_1; M_2$. With the help of Construction 5.1.1 we finally gain a macro tree transducer M, which also induces the same translation.

The macro tree transducer M needs l reduction steps to compute some normal form given the input tree t, if $l + 1$ synthesized attribute instances were reduced by M' in order to compute the normal form given the input tree $(\widehat{\sigma}\,t)$ (cf. Theorem 5.2.1).

A relation between k_1, k_2 and l was established in Theorem 4.2.4, which we will restate here for macro tree transducers.

5.3.1 Theorem (decision theorem)
Let M_1, M_2 be syntactic single-use and preserving in context parameters

macro tree transducers and let M with $\tau(M) = \tau(M_1); \tau(M_2)$ be the macro tree transducer constructed as explained above. Furthermore let t_1 be an input tree for M_1 and consequently $t_2 = \tau(M_1)(t_1)$. Then M is more efficient, with respect to the plain number of reduction steps, if and only if

$$\mathrm{size}(t_2)|F_2| > \mathrm{size}(t_1)\Big(|F_1|(|F_2|-1) + (\sum_{f\in \underline{F_1}} \mathrm{rank}_{F_1}(f) - 1)(\sum_{f\in \underline{F_2}} \mathrm{rank}_{F_2}(f) - 1)\Big)$$

□

In general the transformation seems to suffer heavily from the explosion in the number of attributes caused by Construction 4.1.1. Furthermore there is a strong presentiment that this behaviour applies to less restricted classes as well.

In the following M_1 and M_2 shall be syntactic single-use attributed tree transducers as defined in Theorem 4.2.4. We try to find subclasses such that we can guarantee a performance benefit.

1. Let M_2 be a top-down tree transducer with $|F_2| = 1$. Then Theorem 4.2.4 yields

   $$\mathrm{size}(t_2) \; > \; 0.$$

   Consequently performing the composition is beneficial. A similar result was shown in [Höf99]. Actually the motivating example presented in the introduction is an example for this class.

2. M_1 is producing, i.e. every right hand side must contain at least one output symbol. Then every function call also introduces an output symbol and since the macro tree transducer M_1 is syntactic single-use and non-deleting in context parameters, $\mathrm{size}(t_2) \geq \mathrm{size}(t_1)|F_1|$ and consequently if

   $$|F_1| \quad > \quad (\sum_{f\in \underline{F_1}} \mathrm{rank}_{F_1}(f) - 1)(\sum_{f\in \underline{F_2}} \mathrm{rank}_{F_2}(f) - 1)$$

   then performing the transformation is in fact beneficial.

Since $\mathrm{ATT}_{ssu}\, ; \mathrm{ATT}_{lsu} \subseteq \mathrm{ATT}_{lsu}$ and our performance measure applies for at least single-use attributed tree transducers as well (cf. Lemma 4.2.2), we can conclude that the resulting attributed tree transducer will reduce every attribute instance of a given input tree at least once (using a true call-by-need derivation relation for attributed tree transducers), because it might reduce synthesized attribute instances several times (non-weakly single-use behaviour). Then the macro tree transducer, resulting out of Construction 5.1.1 will also reduce the corresponding function calls more than once, so that we can pessimistically approximate the number of reduction steps.

5.3.2 Observation

Let M_1 be a syntactic single-use and preserving in context parameters macro tree transducer, M_2 be a syntactic single-use and non-deleting in context parameters macro tree transducer and let M with $\tau(M) = \tau(M_1); \tau(M_2)$ be the macro tree transducer constructed as explained above. Furthermore let t_1 be an input tree for M_1 and consequently $t_2 = \tau(M_1)(t_1)$. Then the sequential composition $M_1; M_2$ is more efficient, with respect to the plain number of reduction steps, if

$$\mathrm{size}(t_2)|F_2| < \mathrm{size}(t_1)\Big(|F_1|(|F_2|-1) + (\sum_{f\in \underline{F_1}} \mathrm{rank}_{F_1}(f) - 1)(\sum_{f\in \underline{F_2}} \mathrm{rank}_{F_2}(f) - 1)\Big)$$

□

Chapter 6

Conclusions

The main focus of this thesis was the identification of classes of functional programs, where we could guarantee (independent of the actual input) that the composition result (as computed by our series of transformations) always outperforms the original sequential composition. An input-size dependent characterization was established. There one also noticed that the series of transformations seems to suffer heavily from the explosion in the number of attributes initiated by the composition of the attributed tree transducer. Some small subclasses were identified and a pessimistic approximation of the performance was given, when dealing with copying behaviour.

6.1 Future work

Within this thesis we presented an efficiency analysis for the composition of macro tree transducers with the help of attributed tree transducers. Therefore we needed to restrict the macro tree transducers (and accordingly the functional programs) to be syntactic single-use and non-deleting in context parameters.

It could be worthwhile to further investigate this series of transformations trying to relax some of our imposed restrictions, especially the restriction single-use. To establish efficiency analysis results for general macro tree transducers, more elaborate techniques are necessary to drop the restrictions non-deleting in context parameters or at least single-use.

Furthermore other transformations, especially the direct composition of macro tree transducers presented in [Voi01], should be analysized as well. By investigating these transformations one might be able to gain decision procedures for more general classes of functional programs, since these transformations not necessarily impose conditions such as weakly single-use.

Also intermediate result elimination techniques which are not based on tree transducer theory should be taken into consideration, in particular the short-cut deforestation technique [GLP93] since it is already implemented (for example in efficient implementations of Haskell).

Bibliography

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

[Bor97] E. Bormann. Effizienzanalyse für Transformationen von primitiv-rekursiven Programmschemata. Großer Beleg, Dresden University of Technology, September 1997.

[BW88] R. Bird and P. Wadler. *Introduction to functional programming.* International Series in Computer Science. Prentice Hall, 1988.

[CDPR98] L. Correnson, E. Duris, D. Parigot, and G. Roussel. Symbolic composition. Technical report 3348, Unité de recherche INRIA, Rocquencourt (France), 1998.

[CDPR99] L. Correnson, E. Duris, D. Parigot, and G. Roussel. Declarative program transformation: a deforestation case-study. In *International Conference on Principles and Practice of Declarative Programming 1999 Paris (France)*, volume 1702 of *Lecture Notes in Computer Science*, pages 360–377. Springer, September / October 1999.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *Journal of Computing System Science*, 31:71–146, 1985.

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

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

[FZ82] P. Franchi-Zannettacci. *Attributs sémantiques et schémas de programmes.* Ph.D. thesis, Université de Bordeaux I, 1982.

[Gan83] H. Ganzinger. Increasing modularity and language-independency in automatically generated compilers. *Science of Computer Programming*, 3:223–278, 1983.

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

[GLP93] A. Gill, J. Launchbury, and S.L. Peyton Jones. A short cut to deforestation. In *Conference on functional programming languages and computer architecture*, pages 223–232. ACM Press, Denmark, 1993.

[God00] M. Godisch. Komposition mit Hilfe von Attributgrammatiken. Hauptseminarbeleg, Dresden University of Technology, July 2000.

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

[Küh97] A. Kühnemann. *Berechnungsstärken von Teilklassen primitiv-rekursiver Programmschemata*. Ph.D. thesis, Dresden University of Technology, 1997.

[Küh98] A. Kühnemann. Benefits of tree transducers for optimizing functional programs. In *Foundations of Software Technology & Theoretical Computer Science 1998 Chennai (India)*, volume 1530 of *Lecture Notes in Computer Science*, pages 146–157. Springer, 1998.

[Küh99] A. Kühnemann. Comparison of deforestation techniques for functional programs and for tree transducers. In *International Symposium on Functional and Logic Programming 1999 Tsukuba (Japan)*, volume 1722 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 1999.

[Küh00a] A. Kühnemann. Attribute grammars and program optimization. Lecture script, Dresden University of Technology, July 2000.

[Küh00b] A. Kühnemann. Benefits of tree transducers for optimizing functional programs. In *International Colloquium 'Partial Evaluation and Program Transformation'*, pages 61–82. Waseda University, Tokyo (Japan), 2000.

[KV97] A. Kühnemann and H. Vogler. *Attributgrammatiken - Eine grundlegende Einführung*. Vieweg, 1997.

[Mal89] G. Malcolm. Homomorphisms and promotability. In *Mathematics of program construction*, volume 375 of *Lecture Notes in Computer Science*, pages 335–347. Springer, 1989.

[Rou94] G. Roussel. *Algorithmes de base pour la modularité et la réutilisabilité des grammaires attribuées*. Ph.D. thesis, Université de Paris 6, 1994.

[Tho99] S. Thompson. *Haskell - the craft of functional programming*. International Computer Science Series. Addison-Wesley, 1999.

[Voi01] J. Voigtländer. Composition of restricted macro tree transducers. Master's thesis, Dresden University of Technology, March 2001.

[Wad90] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248, 1990.