Benefits of Tree Transducers for Optimizing Functional Programs

Armin Kühnemann

Grundlagen der Programmierung, Institut für Softwaretechnik I,
Fakultät Informatik, Technische Universität Dresden, D–01062 Dresden, Germany,
e–mail: kuehne@orchid.inf.tu–dresden.de

Abstract. We present a technique to prevent the construction of intermediate data structures in functional programs, which is based on results from the theory of tree transducers. We first decompose function definitions, which correspond to macro tree transducers, into smaller pieces. Under certain restrictions these pieces can be composed to an attributed tree transducer using a composition result for attribute grammars. The same construction can be used to compose the attributed tree transducers, which represent the initial function definitions.

1 Introduction

In functional programs compositions of functions are frequently used in order to support a modular style of programming. Thus, one function can produce an intermediate result, which is consumed by another function. If these intermediate results are structured objects, like lists or trees, then this programming style can cause inefficiencies. For instance, Wadler has tackled this problem by inventing the deforestation technique for transforming programs using intermediate data structures into programs that do not (cf. [18]). The original transformation terminates for treeless first-order programs, where essentially every occurrence of a function in the right-hand side of an equation is only applied to variables.

Attribute grammars have been introduced by Knuth in [12] to assign meaning to programs which are produced by context-free grammars. Johnsson even considers them as functional programming paradigms (cf. [11]). At least they suffer from the same problems, if they are used in a compositional programming style. Ganzinger has shown in [8] that restricted attribute grammars, which are called language morphisms, are closed under composition. Ganzinger and Giegerich have presented this composition in the framework of attribute coupled grammars (cf. [9, 10]). The imposed restriction is called *syntactic single use requirement*, which essentially means that every attribute value in a tree may be used at most once to calculate other attribute values.

Attributed tree transducers (for short *atts*) have been defined by Fülöp in [7] as formal models to study properties of attribute grammars. They abstract from attribute grammars in the sense that they translate trees into trees, which are constructed over ranked alphabets. Fülöp has shown that the class of atts is not closed under composition, but closed under right-composition with the class of *top-down tree transducers* (for short *tdtts*; cf. [15, 17]). Tdtts can be seen as atts without inherited attributes and thus as models for S-attribute grammars.

In this paper we present a technique to prevent intermediate data structures of primitive-recursive tree functions, which is based on the composition result for attribute grammars. For this purpose we consider *macro tree transducers* (for short *mtts*), which have been introduced by Engelfriet in [4] (also cf. [2, 6]). Mtts extend the scheme of primitive recursion, additionally allowing simultaneous definitions of functions and nesting of trees in parameter positions. In order to generalize our results, we actually present them for *macro attributed tree transducers* (for short *matts*), which we have invented in [14] as integration of atts and mtts. Since the classes of mtts and of matts generally are not closed under composition (cf. [6, 14]), we have to impose restrictions.

We project the syntactic single use requirement on matts. Since matts can copy context parameters, this projection still allows that an attribute value is used more than once to calculate other attribute values. Thus, we call our restriction *weakly single-use*. In this paper we additionally take advantage of the *noncopying* restriction for matts. This restriction means that context variables of functions may be used at most once in the right-hand sides of function definitions. Since atts have no context variables, the weakly single-use restriction for atts is called *single-use* restriction.

In [14] we have decomposed matts into atts and yield functions, which perform substitutions of parameter values. According to Engelfriet (cf. [5]), also yield functions can also be realized by atts. In [13] we have shown that the decomposition carries over the weakly single-use restriction from an matt to the first att. If the matt is noncopying, then the second att is single-use. Thus, under both restrictions, the single-use atts can be composed to one single-use att.

In consequence, also restricted matts, and in particular restricted primitive-recursive functions, can be composed to one single-use att. Thus, intermediate data structures are no more constructed!

With this method also compositions of functions can be handled, which deforestation does not optimize. An example which demonstrates this is the twofold reversal of monadic trees, where the accumulating parameter (cf. e.g. [16]) causes the trouble in deforestation. In [13] we have presented another example, in which three restricted mtts for the translation of binary numbers (cf. also [12]) into programs for an abstract machine were composed to one att.

Correnson et al. apply in [1] the same strategy as we did in [13]. They perform the transformation of functions into attribute grammars directly without decomposition and composition steps. But in contrast to our work, they do not specify the class of functions, which their strategy can treat. Further they state, that the strategy of composition via attribute grammars is more powerful than deforestation. We cannot agree with this statement, because there are also examples, which deforestation can handle, but not attribute grammar composition. An example is the composition of a tdtt, which produces for a monadic tree a full binary tree of the same height, with an mtt, which produces again a monadic tree with a height that is equal to the number of leaves of the binary tree.

2 Preliminaries

We denote the set of natural numbers including 0 by $\mathbb{N}$. For every $m \in \mathbb{N}$, the set $\{1, \ldots, m\}$ is denoted by $[m]$. We will use the set $Y = \{y_1, y_2, y_3, \ldots\}$ of *variables* and for every $k \in \mathbb{N}$ the abbreviation $Y_k = \{y_1, \ldots, y_k\}$. If Σ is an alphabet, then Σ^* denotes the set of *strings* over Σ. The empty string is denoted by ε. For a string v and two lists $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$ of strings such that no pair u_j and u_l overlaps in v, we abbreviate by $v[u_1/v_1, \ldots, u_n/v_n]$ (or by $v[u_j/v_j\,;\, j \in [n]]$) the string which is obtained from v by replacing every occurrence of u_j in v by v_j. Let $\Rightarrow$ be a binary relation on a set K. Then, $\Rightarrow^*$ denotes the transitive, reflexive closure of $\Rightarrow$. If $k \Rightarrow^* k'$ for $k, k' \in K$ and if there is no $k'' \in K$ such that $k' \Rightarrow k''$, then k' is called a *normal form of k with respect to* $\Rightarrow$, which is denoted by $nf(\Rightarrow, k)$, if it exists and if it is unique.

A *ranked alphabet* is a pair $(\Sigma, rank_\Sigma)$ where Σ is a finite set and $rank_\Sigma : \Sigma \longrightarrow \mathbb{N}$ is a mapping. For every $\sigma \in \Sigma$, $rank_\Sigma(\sigma)$ is called the *rank of* σ. If Σ is known and $rank_\Sigma(\sigma) = n$, then we write $\sigma^{(n)}$. $\Sigma^{(n)}$ denotes the set of elements with rank n. The set of *trees over* Σ, denoted by $T\langle\Sigma\rangle$, is the smallest set T such that for every $n \geq 0$, $\sigma \in \Sigma^{(n)}$, and $t_1, \ldots, t_n \in T$: $\sigma(t_1, \ldots, t_n) \in T$. Let $t \in T\langle\Sigma\rangle$. Then, $paths(t) \subseteq \mathbb{N}^*$ is the *set of paths of* t. If $t \in \Sigma^{(0)}$, then $paths(t) = \{\varepsilon\}$. If $t = \sigma(t_1, \ldots, t_n)$ with $\sigma \in \Sigma^{(n)}$, $n \geq 1$, and $t_1, \ldots, t_n \in T\langle\Sigma\rangle$, then $paths(t) = \{\varepsilon\} \cup \{jp \mid j \in [n], p \in paths(t_j)\}$. Every path $p \in paths(t)$ determines exactly one node of t. The *label* $\sigma \in \Sigma$ of the node of t which is reached by a path $p \in paths(t)$ is denoted by $label(t, p)$.

Let Σ, Δ, and Ω be ranked alphabets. A total function $\tau : T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ is called *tree transformation*. If $\tau_1 : T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ and $\tau_2 : T\langle\Delta\rangle \longrightarrow T\langle\Omega\rangle$ are tree transformations, then we denote the *composition* of τ_1 and τ_2 by $\tau_1 \circ \tau_2$. Thus, for every $t \in T\langle\Sigma\rangle$, $\tau_1 \circ \tau_2 : T\langle\Sigma\rangle \longrightarrow T\langle\Omega\rangle$ is defined by $(\tau_1 \circ \tau_2)(t) = \tau_2(\tau_1(t))$. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are two classes of tree transformations, then $\{\tau_1 \circ \tau_2 \mid \tau_1 \in \mathcal{T}_1, \tau_2 \in \mathcal{T}_2\}$ is denoted by $\mathcal{T}_1 \circ \mathcal{T}_2$.

A function $h : \Omega^{(0)} \longrightarrow T\langle\Delta\rangle(Y)$ such that $h(\omega)$ contains for every $\omega \in \Omega^{(0)}$ and $y_j \in Y$ at most one occurrence of y_j is called *yield base function*. The following *yield function* $yield_h : T\langle\Omega\rangle \longrightarrow T\langle\Delta\rangle(Y)$ interprets every occurrence of a symbol $\omega \in \Omega^{(n+1)}$ as substitution, which replaces every occurrence of a variable y_j with $j \in [n]$ in the interpretation of the first subtree of ω by the interpretation of the $(j+1)$-st subtree of ω: For every $\omega \in \Omega^{(0)}$, $yield_h(\omega) = h(\omega)$ and for every $n \geq 0$, $\omega \in \Omega^{(n+1)}$, and $t_0, t_1, \ldots, t_n \in T\langle\Omega\rangle$, $yield_h(\omega(t_0, t_1, \ldots, t_n)) = yield_h(t_0)[y_j/yield_h(t_j)\,;\, j \in [n]]$. The class of yield functions is denoted by $YIELD$.

3 Macro Attributed Tree Transducers

We recall the definition of a macro attributed tree transducer from [14]. Note that the first argument of every function is a path of a tree t. If the *path variable* z in the rules is instantiated by a path p to a node of t, then the path zj is instantiated by the path pj to the j-th successor of this node.

Definition 3.1 A *macro attributed tree transducer* (for short *matt*) is a tuple $M = (S, I, \Sigma, \Delta, s_{in}, root, R)$ where

- S is a ranked alphabet of *synthesized functions*; for every $s \in S$, $rank_S(s) \geq 1$.
- I is a ranked alphabet of *inherited functions*; for every $i \in I$, $rank_I(i) \geq 1$.
- Σ is a ranked alphabet of *input symbols*.
- Δ is a ranked alphabet of *output symbols*.
- s_{in} is a unary symbol, called the *initial synthesized function*.
- $root$ is a unary symbol, called the *root symbol*.

We assume that S, I, Σ, Δ, $\{s_{in}\}$, and $\{root\}$ are disjoint in pairs.
For every $k \geq 0$, $I' \in \{I, \emptyset\}$, and $n \geq 0$ the set of *right-hand sides over S, $[k]$, I', Δ, and Y_n*, denoted by $RHS(S, [k], I', \Delta, Y_n)$, is the smallest set RHS with:

1. For every $m \geq 0$, $s \in S^{(m+1)}$, $j \in [k]$, and $\varrho_1, \ldots, \varrho_m \in RHS$: $s(zj, \varrho_1, \ldots, \varrho_m) \in RHS$.
2. For every $m \geq 0$, $i \in I'^{(m+1)}$, and $\varrho_1, \ldots, \varrho_m \in RHS$: $i(z, \varrho_1, \ldots, \varrho_m) \in RHS$.
3. For every $r \geq 0$, $\delta \in \Delta^{(r)}$, and $\varrho_1, \ldots, \varrho_r \in RHS$: $\delta(\varrho_1, \ldots, \varrho_r) \in RHS$.
4. $Y_n \subseteq RHS$.

- $R = \bigcup_{\sigma \in \Sigma \cup \{root\}} R_\sigma$ is the set of *rules* with:
  - R_{root} contains
    - exactly one rule $s_{in}(z) \rightarrow \rho$ with $\rho \in RHS(S, [1], \emptyset, \Delta, Y_0)$ and
    - for every $n \geq 0$ and $i \in I^{(n+1)}$ exactly one rule $i(z1, y_1, \ldots, y_n) \rightarrow \varrho$ with $\varrho \in RHS(S, [1], \emptyset, \Delta, Y_n)$.
  - For every $k \geq 0$ and $\sigma \in \Sigma^{(k)}$, R_σ contains
    - for every $n \geq 0$ and $s \in S^{(n+1)}$ exactly one rule $s(z, y_1, \ldots, y_n) \rightarrow \varrho$ with $\varrho \in RHS(S, [k], I, \Delta, Y_n)$ and
    - for every $n \geq 0$, $i \in I^{(n+1)}$, and $j \in [k]$ exactly one rule $i(zj, y_1, \ldots, y_n) \rightarrow \varrho$ with $\varrho \in RHS(S, [k], I, \Delta, Y_n)$. □

Example 3.2 Let $\Sigma_{rv} = \{A^{(1)}, B^{(1)}, E^{(0)}\}$. We define the matts M_{rv} and M^*_{rv}:
$M_{rv} = (S_{rv}, \emptyset, \Sigma_{rv}, \Sigma_{rv}, s_{in}, root, R_{rv})$ with $S_{rv} = \{rv^{(2)}\}$ and
$(R_{rv})_{root} = \{s_{in}(z) \rightarrow rv(z1, E)\}$ $(R_{rv})_A = \{rv(z, y_1) \rightarrow rv(z1, A(y_1))\}$
$(R_{rv})_E = \{rv(z, y_1) \rightarrow y_1\}$ $(R_{rv})_B = \{rv(z, y_1) \rightarrow rv(z1, B(y_1))\}$
M_{rv} reverses the occurrences of As and Bs in a tree $t \in T\langle\Sigma_{rv}\rangle$ by accumulating them in the second parameter of rv. M_{rv} represents the four equations $s_{in}\ (root\ x) = rv\ x\ E$, $rv\ E\ y_1 = y_1$, $rv\ (A\ x)\ y_1 = rv\ x\ (A\ y_1)$, and $rv\ (B\ x)\ y_1 = rv\ x\ (B\ y_1)$ of a functional program in Haskell-like notation, where $root$, A, B, and E are data constructors, and x and y_1 are variables.
$M^*_{rv} = (S^*_{rv}, I^*_{rv}, \Sigma_{rv}, \Sigma_{rv}, s_{in}, root, R^*_{rv})$ with $S^*_{rv} = \{s^{(1)}\}$, $I^*_{rv} = \{i^{(1)}\}$, and
$(R^*_{rv})_{root} = \{s_{in}(z) \rightarrow s(z1),\ \ i(z1) \rightarrow E\}$
$(R^*_{rv})_A = \{s(z) \rightarrow s(z1),\ \ i(z1) \rightarrow A(i(z))\}$
$(R^*_{rv})_B = \{s(z) \rightarrow s(z1),\ \ i(z1) \rightarrow B(i(z))\}$
$(R^*_{rv})_E = \{s(z) \rightarrow i(z)\}$
M^*_{rv} reverses a tree $t \in T\langle\Sigma_{rv}\rangle$, collecting the labels of the nodes of t in reverse order by i and passing the resulting tree by s unchanged to the root of t. □

An matt translates *input trees* (of $T\langle\Sigma\rangle$) into *output trees* (of $T\langle\Delta\rangle$). For every $e \in T\langle\Sigma\rangle$, the tree $root(e)$, abbreviated by $\tilde{e}$, is called *control tree* and we choose the value of s_{in} at the root of $\tilde{e}$ as output tree (cf. Definition 3.8). We use the abbreviations $F = S \cup I$, $F_+ = F \cup \{s_{in}\}$, $S_+ = S \cup \{s_{in}\}$ and $\Sigma_+ = \Sigma \cup \{root\}$. A function $f \in F_+^{(1)}$ is called *attribute*. For every $k \geq 0$ and $\sigma \in \Sigma^{(k)}$ we define the set of *inside* and *outside function occurrences of* σ as $in_M(\sigma) = \{s(z) \mid s \in S\} \cup \{i(zj) \mid i \in I, j \in [k]\}$ and $out_M(\sigma) = \{i(z) \mid i \in I\} \cup \{s(zj) \mid s \in S, j \in [k]\}$, respectively. $in_M(root) = \{s_{in}(z)\} \cup \{i(z1) \mid i \in I\}$ and $out_M(root) = \{s(z1) \mid s \in S\}$ are the sets of *inside* and *outside function occurrences of root*. An *attribute occurrence* is a function occurrence with function component of rank 1.

Definition 3.3 Let $M = (S, I, \Sigma, \Delta, s_{in}, root, R)$ be an matt.

- M is a *macro tree transducer* (for short *mtt*), if $I = \emptyset$.
- M is an *attributed tree transducer* (for short *att*), if $S = S^{(1)}$ and $I = I^{(1)}$.
- M is a *top-down tree transducer* (for short *tdtt*), if $S = S^{(1)}$ and $I = \emptyset$. □

Example 3.4 M_{rv} is an mtt and M_{rv}^* is an att. □

The set of sentential forms of an matt M is the set of trees over Δ and functions which are applied to paths of a control tree $\tilde{e}$ in their first arguments. $\tilde{e}$ itself will parameterize the derivation relation. With regard to the derivation relation we define the set of sentential forms for an arbitrary superset $\Delta' \supseteq \Delta$.

Definition 3.5 Let $M = (S, I, \Sigma, \Delta, s_{in}, root, R)$ be an matt. Let $e \in T\langle\Sigma\rangle$ and let Δ' be a ranked alphabet with $\Delta \subseteq \Delta'$. The set $SF(F, s_{in}, paths(\tilde{e}), \Delta')$ of *sentential forms over* F, s_{in}, $paths(\tilde{e})$ *and* Δ' is the smallest set SF such that:

1. $s_{in}(\varepsilon) \in SF$.
2. For every $n \geq 0$, $f \in F^{(n+1)}$, $p \in (paths(\tilde{e}) - \{\varepsilon\})$, and $t_1, \ldots, t_n \in SF$: $f(p, t_1, \ldots, t_n) \in SF$.
3. For every $r \geq 0$, $\delta \in \Delta'^{(r)}$, and $t_1, \ldots, t_r \in SF$: $\delta(t_1, \ldots, t_r) \in SF$. □

For an matt M and for a control tree $\tilde{e}$, the set of *function instances of* $\tilde{e}$ is the set $\{s_{in}(\varepsilon)\} \cup \{f(p) \mid f \in F, p \in (paths(\tilde{e}) - \{\varepsilon\})\}$.

Definition 3.6 Let $M = (S, I, \Sigma, \Delta, s_{in}, root, R)$ be an matt. Let $e \in T\langle\Sigma\rangle$. The *derivation relation of* M *with respect to* $\tilde{e}$, denoted by $\Rightarrow_{M,\tilde{e}}$, is a binary relation on $SF(F, s_{in}, paths(\tilde{e}), \Delta)$ defined as follows:
For every $t_1, t_2 \in SF(F, s_{in}, paths(\tilde{e}), \Delta)$, $\quad t_1 \Rightarrow_{M,\tilde{e}} t_2$, $\quad$ iff

- there is $t' \in SF(F, s_{in}, paths(\tilde{e}), \Delta \cup \{u\})$ with exactly one occurrence of the 0-ary symbol $u \notin (F_+ \cup \Delta \cup \mathbb{N}^*)$,
- there is $n \in \mathbb{N}$, a function $f \in F_+^{(n+1)}$, and a path $p \in paths(\tilde{e})$,
- there are $t'_1, \ldots, t'_n \in SF(F, s_{in}, paths(\tilde{e}), \Delta)$ with
- $t_1 = t'[u/f(p, t'_1, \ldots, t'_n)]$, and

1. - either $f \in S_+$,
   - there is $k \geq 0$ and $\sigma \in \Sigma_+^{(k)}$ with $label(\tilde{e}, p) = \sigma$, and
   - there is a rule $(f(z, y_1, \ldots, y_n) \to \varrho) \in R_\sigma$, such that
   - $t_2 = t'[u/\varrho[z/p][y_l/t'_l\,;\, l \in [n]]]$,

2. – or $f \in I$,
   – there is $p' \in paths(\tilde{e})$, $k \geq 1$, $\sigma \in \Sigma_+^{(k)}$, and $j \in [k]$ with $label(\tilde{e}, p') = \sigma$ and $p = p'j$, and
   – there is a rule $(f(zj, y_1, \ldots, y_n) \rightarrow \varrho) \in R_\sigma$, such that
   – $t_2 = t'[u/\varrho[z/p'][y_l/t'_l\,;\, l \in [n]]]$. □

If M or $\tilde{e}$ are known, then we drop the corresponding indices from $\Rightarrow$.

Example 3.7 Let $\tilde{e} = root(A(A(B(E))))$. Then we can derive as follows:

- $s_{in}(\varepsilon) \Rightarrow_{M_{rv}} rv(1, E) \Rightarrow_{M_{rv}} rv(11, A(E)) \Rightarrow_{M_{rv}} rv(111, A(A(E))) \Rightarrow_{M_{rv}} rv(1111, B(A(A(E)))) \Rightarrow_{M_{rv}} B(A(A(E)))$
- $s_{in}(\varepsilon) \Rightarrow_{M^*_{rv}} s(1) \Rightarrow_{M^*_{rv}} s(11) \Rightarrow_{M^*_{rv}} s(111) \Rightarrow_{M^*_{rv}} s(1111) \Rightarrow_{M^*_{rv}} i(1111) \Rightarrow_{M^*_{rv}} B(i(111)) \Rightarrow_{M^*_{rv}} B(A(i(11))) \Rightarrow_{M^*_{rv}} B(A(A(i(1)))) \Rightarrow_{M^*_{rv}} B(A(A(E)))$ □

Matts can be *circular* as attribute grammars, i.e. a function instance may depend on itself. Only *noncircular* matts induce noetherian and confluent derivation relations and thus every sentential form has a unique normal form (cf. [14]).

Definition 3.8 Let $M = (S, I, \Sigma, \Delta, s_{in}, root, R)$ be a noncircular matt. The *tree transformation* $\tau(M) : T\langle\Sigma\rangle \longrightarrow T\langle\Delta\rangle$ *computed by* M, is defined as follows. For every $e \in T\langle\Sigma\rangle$, $\tau(M)(e) = nf(\Rightarrow_{M,\tilde{e}}, s_{in}(\varepsilon))$. □

In the following we always mean noncircular matts when we talk about matts.

Definition 3.9 An matt $M = (S, I, \Sigma, \Delta, s_{in}, root, R)$ is called

- *noncopying*, if for every $(f(\eta, y_1, \ldots, y_n) \rightarrow \varrho) \in R$ and for every $j \in [n]$, the variable y_j occurs at most once in ϱ,
- *weakly single-use*, if for every $k \geq 0$, $\sigma \in \Sigma_+^{(k)}$, and $f(\eta) \in out_M(\sigma)$, $f(\eta)$ occurs in a right-hand side of at most one rule in R_σ and there exactly once,
- and *single-use*, if M is a weakly single-use att. □

Example 3.10 The mtt M_{rv} is noncopying and weakly single-use. The att M^*_{rv} is (trivially) noncopying and weakly single-use. Thus M^*_{rv} is also single-use. □

The *classes of tree transformations*, which are computed by tdtts, atts, mtts, and matts are denoted by T, AT, MT, and MAT, respectively. We add an index *wsu* and an index *nc* to these classes, if we deal with weakly single-use tts and noncopying tts, respectively, where tt is an abbreviation for tree transducer. Instead of AT_{wsu} and T_{wsu} we use AT_{su} and T_{su}, respectively.

4 Decomposition

Engelfriet and Vogler have shown that an mtt can be simulated by the composition of a tdtt with a yield function (cf. [4, 6]). We have generalized this result in [14] to matts, proving that an matt can be simulated by the composition of an att with a yield function. Here we informally recall the construction for the decomposition of an matt M into an att M' and a function $yield_h$.

The key idea is to simulate the task of an $(n+1)$-ary function f of M by a unary function f' of M'. Since f' is unary, it does not know the current values for the Y-parameters to which f is applied. Instead of it, f' introduces a new 0-ary symbol Π_j, wherever f uses its j-th parameter in a calculation. For this purpose, every variable y_j in the right-hand sides of the rules of M are substituted by Π_j.

The function $yield_h$ has to substitute every occurrence of Π_j in a sentential form by the j-th current Y-parameter, thus we define $h(\Pi_j) = y_j$. The current Y-parameters themselves are integrated into the calculation of M' by the substitution of a right-hand side of the form $f(\eta, t_1, \ldots, t_n)$ in M by $SUB_n(f'(\eta), t'_1, \ldots, t'_n)$, where SUB_n is a new $(n+1)$-ary output symbol and $t'_1, \ldots, t'_n$ result from $t_1, \ldots, t_n$ by recursive substitutions. Roughly speaking, $yield_h$ interprets an occurrence of SUB_n as substitution, which replaces every occurrence of Π_j in the subtree, which is generated by $f'(\eta)$, by t'_j.

To avoid the distinction of SUB-symbols from output symbols with a rank greater than 0 in the yield function, also for every r-ary output symbol δ of M a 0-ary output symbol δ' in M' is used. Thus, right-hand sides of the form $\delta(t_1, \ldots, t_r)$ in M are substituted by $SUB_r(\delta', t'_1, \ldots, t'_r)$ and $h(\delta') = \delta(y_1, \ldots, y_r)$ is defined to undo this substitution later. This should be improved in implementations.

This decomposition result and the corresponding composition result (cf. [6, 14]), which is not relevant for our aim, have the following consequences.

Theorem 4.1 $MAT = AT \circ YIELD$ ([14]) and $MT = T \circ YIELD$ ([4, 6]) $\square$

Example 4.2 For M_{rv} we construct a tdtt M'_{rv} and a yield base function h_{rv} such that $\tau(M'_{rv}) \circ yield_{h_{rv}} = \tau(M_{rv})$. $M'_{rv} = (S'_{rv}, \emptyset, \Sigma_{rv}, \Omega_{rv}, s'_{in}, root, R'_{rv})$ with $S'_{rv} = \{rv'^{(1)}\}$, $\Omega_{rv} = \{A'^{(0)}, B'^{(0)}, E'^{(0)}, \Pi_1^{(0)}, SUB_0^{(1)}, SUB_1^{(2)}\}$, and

$$\begin{aligned}
(R'_{rv})_{root} &= \{s'_{in}(z) \rightarrow SUB_1(rv'(z1), SUB_0(E'))\} \\
(R'_{rv})_A &= \{rv'(z) \rightarrow SUB_1(rv'(z1), SUB_1(A', \Pi_1))\} \\
(R'_{rv})_B &= \{rv'(z) \rightarrow SUB_1(rv'(z1), SUB_1(B', \Pi_1))\} \\
(R'_{rv})_E &= \{rv'(z) \rightarrow \Pi_1\}
\end{aligned}$$

Note that M'_{rv} is single-use. The function $h_{rv} : \Omega_{rv}^{(0)} \longrightarrow T\langle\Sigma_{rv}\rangle(Y_1)$ is defined by $h_{rv}(A') = A(y_1)$, $h_{rv}(B') = B(y_1)$, $h_{rv}(E') = E$, and $h_{rv}(\Pi_1) = y_1$. $\square$

The decomposition and the composition, respectively, carries over the weakly single-use restriction from the matt to the att and vice versa, respectively:

Theorem 4.3 $MAT_{wsu} = AT_{su} \circ YIELD$ and $MT_{wsu} = T_{su} \circ YIELD$ ([13])
Proof. Let M be a weakly single-use matt (a weakly single-use mtt, respectively). The construction for the decomposition result delivers an att (a tdtt, respectively) which is single-use, because every outside function occurrence in a right-hand side of a rule of M is translated into exactly one outside attribute occurrence in a right-hand side of a rule of M'. The proof for the composition result is not relevant for the aim of this paper (cf. [13]). $\square$

5 Realization of Yield Functions

Engelfriet has demonstrated, how a yield function can be calculated by an attribute grammar with only one synthesized attribute (cf. [5]). In our framework we can construct an att as follows.

Let $yield_h : T\langle\Omega\rangle \longrightarrow T\langle\Delta\rangle(Y_{mx})$ be a yield function with $mx \in I\!N$ and $h : \Omega^{(0)} \longrightarrow T\langle\Delta\rangle(Y_{mx})$. The instances of the unique synthesized attribute s represent the results of applying $yield_h$ to the subtrees of a tree in $T\langle\Omega\rangle$. The parameters $y_1, \ldots, y_{mx}$ are represented by inherited attributes $i_1, \ldots, i_{mx}$.

Since for every $\omega \in \Omega^{(0)}$ (thus for Π_j- and δ'-symbols from Section 4), $yield_h(\omega) = h(\omega)$, the value of s at an occurrence of ω is obtained from $h(\omega) \in T\langle\Delta\rangle(Y_{mx})$, where for every occurrence of y_j in $h(\omega)$ the value of i_j at ω is used.

For every occurrence of $\omega \in \Omega^{(n+1)}$ (thus for SUB_n-symbols from Section 4) the substitution principle is imitated as follows. The value of s at the first successor of ω is passed to s at ω itself. For every $j \in [mx]$, the value of i_j at ω is passed to i_j at every successor of ω, but the first. If $n \leq mx$, then i_j at the first successor of ω gets for every $j \in [n]$ the value from s at the $(j+1)$-st successor, and for every $j \in [mx] - [n]$ the value from i_j at ω. If $n > mx$ (which cannot appear in yield functions constructed in our decompositions), then for every $j \in [mx]$, i_j at the first successor gets the value from s at the $(j+1)$-st successor. In Figure 1 the situation for $mx = 3$ and for a symbol SUB_2 is shown.

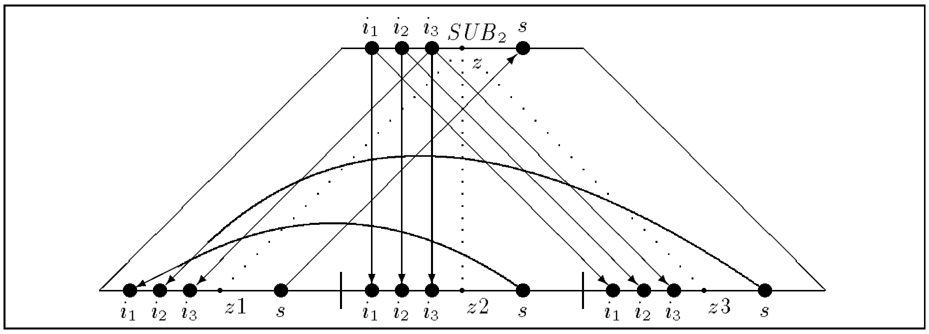

Fig. 1. Substitution principle for $mx = 3$ and SUB_2.

Lemma 5.1 *YIELD* $\subseteq$ *AT* ([5]) $\square$

Example 5.2 We give for h_{rv} the att M''_{rv} such that $\tau(M''_{rv}) = yield_{h_{rv}}$. $M''_{rv} = (S''_{rv}, I''_{rv}, \Omega_{rv}, \Sigma_{rv} \cup \{y_1\}, s_{in}, root, R''_{rv})$ with $S''_{rv} = \{s^{(1)}\}$, $I''_{rv} = \{i_1^{(1)}\}$ and

$$
\begin{array}{llll}
(R''_{rv})_{root} & = \{s_{in}(z) \rightarrow s(z1), & (R''_{rv})_{SUB_0} & = \{s(z) \rightarrow s(z1),\\
& \quad\; i_1(z1) \rightarrow y_1\} & & \quad\; i_1(z1) \rightarrow i_1(z)\}\\
(R''_{rv})_{\Pi_1} & = \{s(z) \rightarrow i_1(z)\} & (R''_{rv})_{SUB_1} & = \{s(z) \rightarrow s(z1),\\
(R''_{rv})_{E'} & = \{s(z) \rightarrow E\} & & \quad\; i_1(z1) \rightarrow s(z2),\\
(R''_{rv})_{A'} & = \{s(z) \rightarrow A(i_1(z))\} & & \quad\; i_1(z2) \rightarrow i_1(z)\}\\
(R''_{rv})_{B'} & = \{s(z) \rightarrow B(i_1(z))\} & &
\end{array}
$$

Note that M''_{rv} is single-use. Further note that y_1 in the second rule of $(R''_{rv})_{root}$ is an output symbol of M''_{rv}, which does not occur in any output tree. $\square$

In [13] we have shown $YIELD \not\subseteq AT_{su}$. Thus, the observation from our example cannot be generalized. The reason is, that the value of an attribute i_j at $\omega \in \Omega^{(n+1)}$ may be used for the value of i_j at more than one successor (cf. Figure 1). In the following we show, under which conditions a yield function resulting from a decomposition of an matt can be realized by a single-use att.

If the yield base function is constructed according to Section 4, then in the case $n \leq mx$ we can cancel[1] unconditionally for every $\omega \in \Omega^{(n+1)}$ and $j \in [mx]-[n]$ the rule for i_j at the first successor of ω (in Figure 1 for i_3 at $z1$). This holds, because in the decomposition an occurrence of SUB_n in the right-hand side of a rule of the att M' is used, iff an $(n+1)$-ary function or an n-ary output symbol occurs in a right-hand side of a rule of the matt M. Thus the calculation inside the first subtree t_0 of a subtree $SUB_n(t_0, t_1, \ldots, t_n)$ generated by M' only depends on $t_1, \ldots, t_n$, which are passed into t_0 via $i_1, \ldots, i_n$.

A rule for i_j with $j \in [mx]$ at another successor of ω may only be cancelled, if the j-th parameter is not used there. Unfortunately this property cannot be deduced from a yield function alone. One has to consider the structure of possible input trees for the yield function. The structure of these trees is known, if they are output trees of an att M' resulting from a decomposition of an matt M.

In particular, if M is noncopying, then every subtree of every right-hand side of M' has at most one occurrence of a symbol Π_j with $j \in [mx]$. This holds in particular for subtrees, which are labeled by a symbol SUB_n at their roots. Thus, the j-th parameter has to be passed into at most one subtree of an occurrence of SUB_n in the intermediate result calculated by M'. Hence rules can be cancelled, leaving for every $j \in [mx]$ at most one rule, in which i_j at SUB_n is used.

Since generally at different occurrences of SUB_n the value of i_j has to be passed into subtrees with different number, instead of SUB_n we need a set $\{SUB_{(m_1,\ldots,m_n)} \mid m_j \subseteq \{\Pi_1, \ldots, \Pi_{mx}\}$ for every $j \in [n]$ and $m_j \cap m_{j'} = \emptyset$ for $j \neq j'\}$ of $(n+1)$-ary SUB-symbols. In the decomposition of M we use a symbol $SUB_{(m_1,\ldots,m_n)}$ in the right-hand side of a rule of M', iff for every $l \in [n]$ and $\Pi_j \in m_l$, the symbol Π_j occurs in the $(l+1)$-st subtree of $SUB_{(m_1,\ldots,m_n)}$. This information can be computed from the parameter structure of M. We show in Figure 2 the substitution principle for $mx = 3$ and for $SUB_{(\{\Pi_2\},\{\Pi_1,\Pi_3\})}$.

Lemma 5.3 $MAT_{nc} \subseteq AT \circ AT_{su}$ and $MAT_{wsu,nc} \subseteq AT_{su} \circ AT_{su}$ □

There is a possibility to simulate general yield functions by weakly single-use tts. In [6] it was shown, how an "(input-) linear mtt" can calculate a yield function. In our framework this means, that for every input symbol $\sigma^{(k)}$ with $k \geq 0$, for every $j \in [k]$ and for every synthesized function s, the path zj occurs at most once in the right-hand side ϱ of the rule $s(z, \ldots) \rightarrow \varrho$ in R_σ. Since this mtt has only one function, this restriction induces the weakly single-use property.

Lemma 5.4 $YIELD \subseteq MT_{wsu}$ ([6, 13]) □

---

[1] Actually, to cancel a rule means to define its right-hand side as arbitrary 0-ary symbol, in order to fulfill the completeness condition for rules of an matt.

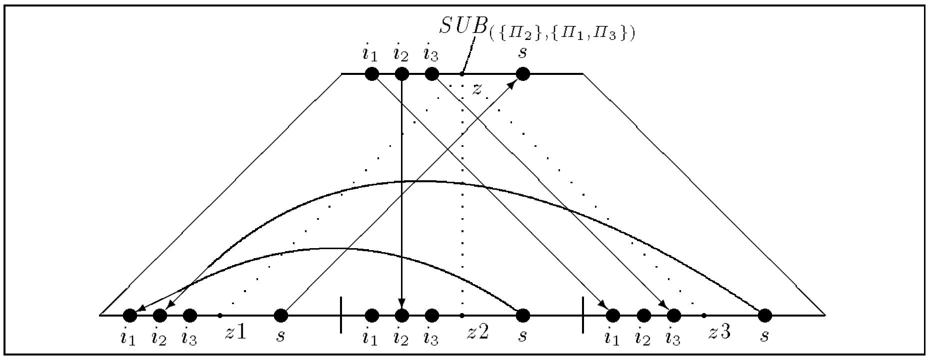

Fig. 2. Substitution principle for $mx = 3$ and $SUB_{(\{\Pi_2\},\{\Pi_1,\Pi_3\})}$.

6 Composition

Fülöp has shown that generally two atts cannot be composed to one att (cf. [7]). Two restrictions are known, under which two atts M_1 and M_2 can be composed to an att M: The construction in [7] requires that M_2 is a tdtt. It takes the cartesian product of the attribute sets of M_1 and M_2 as attribute set of M and computes the right-hand sides of rules of M by "translating" the right-hand sides of rules of M_1 with the help of M_2. The other construction is based on composition results for language morphisms (cf. [8]) and for attribute coupled grammars (cf. [9, 10]) and restricts M_1 to be single-use. In the framework of tts it was presented in [13] by generalizing the translation technique of [7]:

Let $M_1 = (S_1, I_1, \Sigma, \Omega, s_{in,1}, root, R_1)$ and $M_2 = (S_2, I_2, \Omega, \Delta, s_{in,2}, root, R_2)$ be atts. For the att $M = (S, I, \Sigma, \Delta, s_{in}, root, R)$ we define $S = (S_1 \times S_2) \cup (I_1 \times I_2)$ and $I = (S_1 \times I_2) \cup (I_1 \times S_2)$. For every $\sigma \in \Sigma_+^{(k)}$ the rules in R_σ are constructed as follows. For every $s_1(z) \to \varrho$ (and $i_1(zj) \to \varrho$, respectively) in $(R_1)_\sigma$ with $s_1 \in S_1$ ($i_1 \in I_1$ and $j \in [k]$, respectively), ϱ serves as input tree for M_2:

- If a derivation of M_2 is started with $s_2 \in S_2$ at the root of ϱ, then its result is the right-hand side of the rule in R_σ with left-hand side $(s_1, s_2)(z)$ (and $(i_1, s_2)(zj)$, respectively).
- If a derivation of M_2 is started with $i_2 \in I_2$ at a leaf of ϱ, which is labeled by an $a_1(zj') \in out_{M_1}(\sigma)$ with $a_1 \in S_1 \cup I_1$ and $j' \in [k] \cup \{\varepsilon\}$, then its result is the right-hand side of the rule in R_σ with left-hand side $(a_1, i_2)(zj')$.

If during a derivation of M_2 an $i_2' \in I_2$ hits the root of ϱ, then it is replaced by $(s_1, i_2')(z)$ (by $(i_1, i_2')(zj)$, respectively). If an $s_2' \in S_2$ hits an $\hat{a}_1(z\hat{j}) \in out_{M_1}(\sigma)$ with $\hat{a}_1 \in S_1 \cup I_1$ and $\hat{j} \in [k] \cup \{\varepsilon\}$, then it is replaced by $(\hat{a}_1, s_2')(z\hat{j})$.

In general, for every $i_2 \in I_2$ and $a_1(zj') \in out_{M_1}(\sigma)$, the rule with left-hand side $(a_1, i_2)(zj')$ is not unique, because more than one leaf of the right-hand sides of rules in $(R_1)_\sigma$ may be labeled by $a_1(zj')$. This can be avoided, if one of our two restrictions holds, i.e. if M_1 is single-use or M_2 has no inherited attributes.

Since the construction carries over the single-use restriction from M_2 to M (cf. [13]) and since also the corresponding decomposition results hold, we get:

Theorem 6.1 $AT \circ T = AT$ ([7]), $T \circ T = T$ ([15]), $AT_{su} \circ AT = AT$ ([8, 10, 13]), $AT_{su} \circ AT_{su} = AT_{su}$ ([8, 10, 13]), and $T_{su} \circ T_{su} = T_{su}$ ([13]) $\Box$

Example 6.2 The composition construction produces for the single-use atts M'_{rv} and M''_{rv} a single-use att, which is isomorphic to the att M^*_{rv}.

We show the construction of the right-hand side of the rule with left-hand side $(rv', i_1)(z1)$ in the rules for A: Since $(rv'(z) \to \varrho) \in (R'_{rv})_A$ with $\varrho = SUB_1(rv'(z1), SUB_1(A', \Pi_1))$ and since $rv'(z1) \in out_{M_1}(A)$, the control tree $\tilde{\varrho} = root(SUB_1(rv'(z1), SUB_1(A', \Pi_1)))$ allows to derive $i_1(11) \Rightarrow s(12) \Rightarrow s(121) \Rightarrow A(i_1(121)) \Rightarrow A(s(122)) \Rightarrow A(i_1(122)) \Rightarrow A(i_1(12)) \Rightarrow A(i_1(1))$ with M''_{rv}.

Since 1 is the path to the root of ϱ in $\tilde{\varrho}$, $i_1(1)$ is replaced by $(rv', i_1)(z)$, yielding the rule $(rv', i_1)(z1) \to A((rv', i_1)(z))$, which is isomorphic to the rule $i(z1) \to A(i(z))$ in $(R^*_{rv})_A$. $\Box$

It is interesting that M_{rv}, which corresponds to the most natural way to describe the reversal by a functional program, is transformed into M^*_{rv}, which corresponds to the most natural way to describe the reversal by an attribute grammar. Since M^*_{rv} is single-use, it can also be composed with itself (cf. [10]). Note that the construction for the result $AT \subseteq MT$ (cf. [2]) can transform the composition result again into a functional program, which does not use an intermediate result!

7 Theoretical Consequences

The first result shows the limited power of weakly single-use matts. Note that we have proved $MAT = AT \circ AT$ in [14] for the unrestricted class of matts.

Theorem 7.1 $MAT_{wsu} \subseteq AT$
Proof. $MAT_{wsu} = AT_{su} \circ YIELD \subseteq AT_{su} \circ AT = AT$ by Theorem 4.3, Lemma 5.1, and Theorem 6.1. $\Box$

We do not know yet, whether MAT_{wsu} is strictly included in AT or whether these classes are equal. Adding the noncopying restriction, we get a characterization, which reflects the theoretical background of our running example:

Theorem 7.2 $MAT_{wsu,nc} = AT_{su}$
Proof. $MAT_{wsu,nc} \subseteq AT_{su} \circ AT_{su} = AT_{su}$ by Lemma 5.3 and Theorem 6.1 and $AT_{su} \subseteq MAT_{wsu,nc}$, since atts have no parameters from Y. $\Box$

If we use Lemma 5.4 to calculate yield functions, then we get a nice characterization for the class MAT_{wsu} and the result, that MT_{wsu} is closed under left-composition with T_{su}, in analogy to the result $MT = T \circ MT$ from [6].

Theorem 7.3 $MAT_{wsu} = AT_{su} \circ MT_{wsu}$ and $MT_{wsu} = T_{su} \circ MT_{wsu}$
Proof.

MAT_{wsu}	MT_{wsu}	
$= AT_{su} \circ YIELD$	$= T_{su} \circ YIELD$	(Theorem 4.3)
$\subseteq AT_{su} \circ MT_{wsu}$	$\subseteq T_{su} \circ MT_{wsu}$	(Lemma 5.4)
$= AT_{su} \circ T_{su} \circ YIELD$	$= T_{su} \circ T_{su} \circ YIELD$	(Theorem 4.3)
$\subseteq AT_{su} \circ YIELD$	$= T_{su} \circ YIELD$	(Theorem 6.1)
$= MAT_{wsu}$	$= MT_{wsu}$	(Theorem 4.3) $\Box$

8 Further Research Topics

The strategies deforestation and composition are incomparable, since both can solve problems, which the other cannot solve. Is it possible, to integrate both strategies? At least a formal characterization of problems would be useful, for which deforestation has advantages compared with composition, and vice versa.

We think that matts are not only suitable to present composition and decomposition results for tts in a uniform way. Since overlapping research in functional programming and attribute grammars is often hampered by their different formalisms as stated e.g. in [3], we also consider matts to be an interesting possibility to compare or even to integrate the composition results for both concepts.

References

1. L. Correnson, E. Duris, D. Parigot, and G. Roussel. Symbolic composition. Technical Report 3348, INRIA, France, 1998.
2. B. Courcelle and P. Franchi-Zannettacci. Attribute grammars and recursive program schemes. *Theoret. Comput. Sci.*, 17:163–191, 235–257, 1982.
3. E. Duris, D. Parigot, G. Roussel, and M. Jourdan. Attribute grammars and folds: generic control operators. Technical Report 2957, INRIA, France, 1996.
4. J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book, editor, *Formal language theory; perspectives and open problems*, pages 241–286. New York, Academic Press, 1980.
5. J. Engelfriet. Tree transducers and syntax directed semantics. Technical Report Memorandum 363, Technische Hogeschool Twente, 1981.
6. J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comp. Syst. Sci.*, 31:71–145, 1985.
7. Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.
8. H. Ganzinger. Increasing modularity and language-independency in automatically generated compilers. *Science of Computer Programming*, 3:223–278, 1983.
9. H. Ganzinger and R. Giegerich. Attribute coupled grammars. In *Symposium on Compiler Construction, SIGPLAN Notices Vol.19, No.6*, pages 157–170, 1984.
10. R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica*, 25:355–423, 1988.
11. T. Johnsson. Attribute grammars as a functional programming paradigm. In *FPCA'87, Portland*, volume 274 of *LNCS*, pages 154–173. Springer-Verlag, 1987.
12. D.E. Knuth. Semantics of context–free languages. *Math. Syst. Th.*, 2:127–145, 1968. Corrections in *Math. Syst. Th.*, 5:95-96, 1971.
13. A. Kühnemann. *Berechnungsstärken von Teilklassen primitiv-rekursiver Programmschemata*. PhD thesis, Technical Univ. of Dresden, 1997. Shaker Verlag.
14. A. Kühnemann and H. Vogler. Synthesized and inherited functions - a new computational model for syntax-directed semantics. *Acta Informatica*, 31:431-477, 1994.
15. W.C. Rounds. Mappings and grammars on trees. *Math. Syst.Th.*, 4:257–287, 1970.
16. H. Seidl and M.H. Sørensen. Constraints to stop higher-order deforestation. In *POPL'97, Paris*, pages 400–413. ACM Press, 1997.
17. J.W. Thatcher. Generalized^2 sequential machine maps. *J.Comp.Syst.Sci.*, 4:339–367, 1970.
18. P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theor. Comp. Sci.*, 73:231–248, 1990.

This article was processed using the LaTeX macro package with LLNCS style