Conditions for Efficiency Improvement by Tree Transducer Composition*

Janis Voigtländer**

Department of Computer Science, Dresden University of Technology
01062 Dresden, Germany. E-mail: voigt@tcs.inf.tu-dresden.de

Abstract. We study the question of efficiency improvement or deterioration for a semantic-preserving program transformation technique based on macro tree transducer composition. By annotating functional programs to reflect the internal property "computation time" explicitly in the computed output, and by manipulating such annotations, we formally prove syntactic conditions under which the composed program is guaranteed to be more efficient than the original program, with respect to call-by-need reduction to normal form. The developed criteria can be checked automatically, and thus are suitable for integration into an optimizing functional compiler.

1 Introduction

Lazy functional languages are well suited for a modular programming style, where a task is solved by combining solutions of subproblems. Unfortunately, modular programs often lack efficiency compared to other — often less understandable — programs that solve the same tasks. These inefficiencies are caused, e.g., by the production and consumption of structured intermediate results such as lists or trees. As an example, consider the following definitions in **Haskell**:

$$\textbf{data}\ \mathit{Nat} = S\ \mathit{Nat} \mid Z$$

$$\begin{array}{lll} \mathit{exp} :: \mathit{Nat} \to \mathit{Nat} \to \mathit{Nat} & \mathit{div} :: \mathit{Nat} \to \mathit{Nat} & \mathit{div}' :: \mathit{Nat} \to \mathit{Nat} \\ \mathit{exp}\ (S\ x)\ y = \mathit{exp}\ x\ (\mathit{exp}\ x\ y) & \mathit{div}\ (S\ x) = \mathit{div}'\ x & \mathit{div}'\ (S\ x) = S\ (\mathit{div}\ x) \\ \mathit{exp}\quad Z\quad y = S\ y & \mathit{div}\quad Z\quad = Z & \mathit{div}'\quad Z\quad = Z \end{array}$$

The function *exp* — computing $\mathit{exp}\ (S^n\ Z)\ (S^m\ Z) = S^{2^n+m}\ Z$ — is defined using an accumulating parameter y, which will also be called context parameter henceforth. The functions *div* and div' — computing $\mathit{div}\ (S^n\ Z) = S^{n\ \mathit{div}\ 2}\ Z$ — are defined by mutual recursion. If we sequentially compose *exp* and *div* — by computing for some value t of type *Nat* the expression $e = (\mathit{div}\ (\mathit{exp}\ t\ Z))$ — an intermediate data structure is created by *exp* and consumed by the *div*- and div'-functions. A standard technique for eliminating intermediate results is *classical deforestation* [13], an algorithmic instance of the *unfold/fold*-technique [1].

* In *RTA 2002, Proceedings*, volume 2378 of LNCS, pages 222–236.
** Research supported by the DFG under grant KU 1290/2-1.

However, classical deforestation does not succeed in optimizing e, due to its well-known problem of not reaching accumulating parameters [2]. Also, classical deforestation was only proved to be non-deteriorating for linear programs or for call-by-name evaluation without sharing [11].

Kühnemann [7,8] tackled the problem of eliminating intermediate data structures in accumulating parameters by using composition results from the theory of tree transducers [5]. The functional programs considered are extended schemes of primitive recursion — allowing mutual recursion and nesting of terms in context parameter positions — so called *macro tree transducers* (for short *mtts*). Already Engelfriet and Vogler [4] showed that the sequential composition of two mtts can be realized by a single mtt, if one of the original mtts is defined without using context parameters. For the above example, e would thus be transformed to $e' = (f\ t\ (div\ Z)\ (div'\ Z))$, with new functions f and g constructed as[1]:

$$\begin{aligned} f\ (S\ x)\ y_1\ y_2 &= f\ x\ (f\ x\ y_1\ y_2)\ (g\ x\ y_1\ y_2) \\ f\quad Z\quad y_1\ y_2 &= y_2 \\ g\ (S\ x)\ y_1\ y_2 &= g\ x\ (f\ x\ y_1\ y_2)\ (g\ x\ y_1\ y_2) \\ g\quad Z\quad y_1\ y_2 &= S\ y_1 \end{aligned}$$

The transformed program avoids the creation of an intermediate result and its eventual consumption, with obvious benefits for the efficiency. In particular, e' needs fewer lazy evaluation steps for reduction to normal form than e.

In general, the number of call-by-need reduction steps performed by the transformed program might also *increase* compared to the original one. If, for example, we consider the expression $(exp\ (div\ t)\ Z)$, tree transducer composition can again eliminate the intermediate result. However, in this case the transformed program will — except for very small inputs — perform more call-by-need reduction steps than the original program.

Clearly, in order to use tree transducer composition as a program transformation technique in an optimizing compiler, we should be able to discriminate programs for which the transformation degrades efficiency from those for which the transformation is indeed beneficial. In this paper we develop such criteria that can be checked automatically by a compiler. These criteria are sufficient to classify the various examples of the composition techniques from [4] given in [9], where the performance improvement achieved by tree transducer composition for particular programs was assured by ad hoc reasoning or by experiments. Our results improve on formal efficiency considerations by Kühnemann [8] for linear programs and by Höff [6] also for nonlinear ones. For example, our criteria can detect that the replacement of e from the introductory example by e' is safe with respect to efficiency, which was not captured by previous results. Since in the case that the first involved mtt has no context parameters the tree transducer composition technique is equivalent to classical deforestation (cf. [8]), our results also establish call-by-need performance improvements through classical deforestation for some nonlinear programs.

[1] The basic idea is to construct definitions for f and g such that for every $t = (S^n\ Z)$ and $t_1 = (S^m\ Z)$: $f\ t\ (div\ t_1)\ (div'\ t_1) = div\ (exp\ t\ t_1)$
$g\ t\ (div\ t_1)\ (div'\ t_1) = div'\ (exp\ t\ t_1)$.

The remainder of this paper is organized as follows. In Sect. 2 we define basic notations and concepts of macro tree transducer units. Section 3 recalls program transformation by tree transducer composition. In Sect. 4 we develop our formal efficiency analysis and give the main theorems, with application to tree transducer composition and classical deforestation. Finally, Sect. 5 contains future research topics.

2 Preliminaries

We denote by $\mathbb{N}$ the set of natural numbers including 0. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots, n\} \subseteq \mathbb{N}$, and by X_n the finite set $\{x_1, \ldots, x_n\}$ of variables; analogously for Y_n and Z_n.

We denote simultaneous substitution of $v_1, \ldots, v_n$ for $u_1, \ldots, u_n$ in v by $v[u_1, \ldots, u_n \leftarrow v_1, \ldots, v_n]$, but will also use an alternative notation similar to set comprehensions, e.g., $v[u_i \leftarrow v_i \mid i \in [n]]$. We write substitutions left-associative.

A *ranked alphabet* is a pair $(\Sigma, rank_\Sigma)$, where Σ is a finite, nonempty set of symbols and $rank_\Sigma$ assigns to every $\sigma \in \Sigma$ a natural number k, which will also be given by writing $\sigma^{(k)}$. For every $k \in \mathbb{N}$, we define $\Sigma^{(k)} = \{\sigma \in \Sigma \mid rank_\Sigma(\sigma) = k\}$. For every set A disjoint from Σ, we define the set $T_\Sigma(A)$ of *trees over Σ indexed by A* as the smallest set T such that *(i)* $A \subseteq T$ and *(ii)* if $\sigma \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T$, then also $(\sigma\ t_1 \cdots t_k) \in T$. If readability allows, outer brackets of trees will be omitted. We denote $T_\Sigma(\emptyset)$ by T_Σ. For every set $C \subseteq \Sigma \cup A$, we denote the *number of occurrences of symbols from C in a tree t* by $|t|_C$. For a singleton set $C = \{c\}$, we denote $|t|_{\{c\}}$ by $|t|_c$.

We consider *left-linear rewrite systems* [3] with *rewrite rules* of the form $lhs = rhs$ — with lhs and rhs being trees over ranked alphabets indexed by a set of variables, which will usually not be mentioned explicitly — where the right-hand side rhs contains only variables that also occur in lhs. A rewrite system R induces a binary nondeterministic *reduction relation* $\Rightarrow_R$ describing left-to-right application of a rule from R in some context. We use the well-known concepts of *confluence*, *termination* and *normal form*. For a confluent and terminating reduction relation $\Rightarrow_R$, we denote the unique normal form of a tree t with respect to $\Rightarrow_R$ by $nf(\Rightarrow_R, t)$. While $\Rightarrow_R$ does not fix a reduction strategy, our efficiency analysis is concerned with lazy functional programming languages. Hence, we consider *call-by-need reduction steps* [14] (leftmost-outermost reduction with sharing) and denote by $cbn(R, t)$ the number of such steps required to reach the normal form $nf(\Rightarrow_R, t)$.

We model functional programs by macro tree transducer units (for short *mtt units*). Firstly, we describe the possible shapes of right-hand sides of their rules.

Definition 1. *Let Q and Δ be ranked alphabets and $k, r \in \mathbb{N}$.*
The set $RHS(Q, \Delta, k, r)$ of right-hand sides over Q and Δ, with k recursion variables and r context variables, *is the smallest set $RHS \subseteq T_{\Delta \cup Q}(X_k \cup Y_r)$ such that (i) $Y_r \subseteq RHS$, (ii) for every $\delta \in \Delta^{(n)}$ and $\phi_1, \ldots, \phi_n \in RHS$: $(\delta\ \phi_1 \cdots \phi_n) \in RHS$, and (iii) for every $q \in Q^{(n+1)}$ with $n \in \mathbb{N}$, $x_i \in X_k$ and $\phi_1, \ldots, \phi_n \in RHS$: $(q\ x_i\ \phi_1 \cdots \phi_n) \in RHS$.*

Definition 2. *An* mtt unit M *is a tuple* (Q, Σ, Δ, R) *with a ranked alphabet* Q *of* states*, where* $Q^{(0)} = \emptyset$*, a ranked alphabet* Σ *of* input symbols*, a ranked alphabet* Δ *of* output symbols*, where* $Q \cap (\Sigma \cup \Delta) = \emptyset$*, and a set* R *of* rules*, such that* R *contains for every* $k, r \in \mathbb{N}$*,* $\sigma \in \Sigma^{(k)}$ *and* $q \in Q^{(r+1)}$*, exactly one rule of the form:* $q\ (\sigma\ x_1 \cdots x_k)\ y_1 \cdots y_r = rhs_{M,q,\sigma}$*, with* $rhs_{M,q,\sigma} \in RHS(Q, \Delta, k, r)$*.*

Of course, the actual variable names used in rules R of an mtt unit are not fixed to come from X_k and Y_r for some $k, r \in \mathbb{N}$; consistent renaming is allowed. The semantics of an mtt unit is given by the reduction relation induced by its rules.

We give a rather artificial example of three mtt units that will be used to illustrate important phenomena in Sect. 4. For more practical examples see [9], where it was demonstrated that also typical functions on polymorphic data types and some higher-order functions like *map* can be viewed as mtt units by choosing appropriate function and constructor symbols.

Example 1. Let $\Sigma = \Omega = \{S^{(1)}, Z^{(0)}\}$ and $\Delta = \{\delta^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$. Then $M_1 = (\{q^{(2)}\}, \Sigma, \Delta, R_1)$, $M_1' = (\{q'^{(3)}\}, \Sigma, \Delta, R_1')$ and $M_2 = (\{p_1^{(1)}, p_2^{(1)}\}, \Delta, \Omega, R_2)$ are mtt units, where R_1, R_1' and R_2 contain rules as follows:

$$
\begin{array}{ll}
R_1: & \\
\quad q\ (S\ x_1)\ y_1 = \gamma\ (q\ x_1\ (\delta\ (q\ x_1\ y_1)\ y_1)) & \\
\quad q\quad Z\quad y_1 = y_1 & \\
R_1': & \\
\quad q'\ (S\ x_1)\ y_1\ y_2 = \delta\ (q'\ x_1\ y_2\ (\gamma\ y_1))\ (\gamma\ (q'\ x_1\ y_1\ y_2)) & \\
\quad q'\quad Z\quad y_1\ y_2 = y_1 &
\end{array}
\qquad
\begin{array}{ll}
R_2: & p_1\ (\delta\ x_1\ x_2) = p_2\ x_1 \\
 & p_1\quad (\gamma\ x_1)\ = S\ (p_2\ x_1) \\
 & p_1\quad \alpha\quad\ = Z \\
 & p_2\ (\delta\ x_1\ x_2) = p_1\ x_2 \\
 & p_2\quad (\gamma\ x_1)\ = p_1\ x_1 \\
 & p_2\quad \alpha\quad\ = Z\ .
\end{array}
$$

Definition 3. *An mtt unit* $M = (Q, \Sigma, \Delta, R)$ *is:*

- *a* top-down tree transducer unit *(for short* tdtt unit*), if* $Q = Q^{(1)}$
- recursion-linear*, if for every* $q \in Q$*,* $\sigma \in \Sigma^{(k)}$*,* $i \in [k]$*:* $|rhs_{M,q,\sigma}|_{x_i} \leqq 1$
- context-linear*, if for every* $q \in Q^{(r+1)}$*,* $\sigma \in \Sigma$*,* $h \in [r]$*:* $|rhs_{M,q,\sigma}|_{y_h} \leqq 1$
- linear*, if it is recursion-linear and context-linear*
- recursion-nondeleting*, if for every* $q \in Q$*,* $\sigma \in \Sigma^{(k)}$ *and* $i \in [k]$*:* $|rhs_{M,q,\sigma}|_{x_i} \geqq 1$
- context-nondeleting*, if for every* $q \in Q^{(r+1)}$*,* $\sigma \in \Sigma$ *and* $h \in [r]$*:* $|rhs_{M,q,\sigma}|_{y_h} \geqq 1$
- basic*, if the right-hand sides of its rules do not contain nested calls, i.e., subtrees of the form* $(q\ x_i \cdots (q'\ x_{i'} \cdots) \cdots)$
- atmost*, if it is recursion-linear, and it is context-linear or basic*
- atleast*, if it is recursion-nondeleting, and it is context-nondeleting or basic.*

3 Tree Transducer Composition

In this section we recall the semantic-preserving composition of two mtt units into a single mtt unit (the meaning of "semantic-preserving" will be made precise in Lemma 1 below). Constructions for performing such a composition in the cases that the first or the second mtt unit is a tdtt unit were given already in [4]. Here, we present a single transformation that captures both cases.

Construction 1. Let $M_1 = (Q, \Sigma, \Delta, R_1)$ and $M_2 = (P, \Delta, \Omega, R_2)$ be mtt units, such that one of the two is a tdtt unit, and $Q \cap P = \emptyset$. Let $m \in \mathbb{N}$ be the number of elements of P, and fix some ordering on P, such that $P = \{p_1, \ldots, p_m\}$. The composed mtt unit will have the set of states $F = \{\overline{qp}^{(r*m+s+1)} \mid q \in Q^{(r+1)}, p \in P^{(s+1)}\}$. We use two rewrite systems:

Pre, which contains for every $h \in [r]$ with $Q^{(r+1)} \neq \emptyset$ and every $p \in P^{(1)}$, the rewrite rule: $p\ y_h = y_{h,p}$. Here the y_h and $y_{h,p}$ are treated as ordinary symbols, rather than as variables of the rewrite system.

Comp, which contains for every $q \in Q^{(r+1)}$ and $p \in P^{(s+1)}$, the rewrite rule:

$$p\ (q\ x\ y_1 \cdots y_r)\ z_1 \cdots z_s = \overline{qp}\ x\ (p_1\ y_1) \cdots (p_m\ y_1) \cdots (p_1\ y_r) \cdots (p_m\ y_r)\ z_1 \cdots z_s\ .$$

Here x, $y_1, \ldots, y_r$ and $z_1, \ldots, z_s$ are considered as variables of the rewrite system. Since M_1 or M_2 is a tdtt unit, never both the ys and zs will be present. We abbreviate the right-hand side of the above rewrite rule as $\zeta_{q,p}$.

Since the ranked alphabets $P, \Delta, \{y_1^{(0)}, y_2^{(0)}, \ldots\}$ and Q are pairwise disjoint, there are no *critical pairs* [3] in $R_2 \cup \mathit{Pre} \cup \mathit{Comp}$. Hence, the reduction relation $\Rightarrow_{R_2 \cup \mathit{Pre} \cup \mathit{Comp}}$ is confluent. It is also terminating, because for every rule the first arguments of calls to states of P in the right-hand side are proper subtrees of the first argument of the call on the left-hand side.

Now, we can construct the mtt unit $\overline{M_1 M_2} = (F, \Sigma, \Omega, \overline{R_1 R_2})$ with rules as follows. For every $q \in Q^{(r+1)}$, $\sigma \in \Sigma^{(k)}$, rule $q\ (\sigma\ x_1 \cdots x_k)\ y_1 \cdots y_r = \mathit{rhs}_{M_1,q,\sigma}$ in R_1, and $p \in P^{(s+1)}$, $\overline{R_1 R_2}$ contains the rule:

$$\overline{qp}\ (\sigma\ x_1 \cdots x_k)\ y_{1,p_1} \cdots y_{r,p_m}\ z_1 \cdots z_s = \mathit{nf}(\Rightarrow_{R_2 \cup \mathit{Pre} \cup \mathit{Comp}}, p\ \mathit{rhs}_{M_1,q,\sigma}\ z_1 \cdots z_s)$$

Then, $\overline{M_1 M_2}$ implements the sequential composition of M_1 and M_2, in the sense of the following lemma.

Lemma 1. *For every rewrite rule* $(p\ (q\ x\ y_1 \cdots y_r)\ z_1 \cdots z_s = \zeta_{q,p}) \in \mathit{Comp}$, $t \in T_\Sigma$, $t_1, \ldots, t_r \in T_\Delta$ *and* $t'_1, \ldots, t'_s \in T_\Omega$:

1. $\mathit{nf}(\Rightarrow_{R_1 \cup R_2}, p\ (q\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s) = \mathit{nf}(\Rightarrow_{\overline{R_1 R_2}}, \zeta_{q,p}[x \leftarrow t])$
2. $\mathit{nf}(\Rightarrow_{R_1 \cup R_2}, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)$
   $= \mathit{nf}(\Rightarrow_{\overline{R_1 R_2} \cup R_2}, \zeta_{q,p}[x \leftarrow t][y_1, \ldots, y_r, z_1, \ldots, z_s \leftarrow t_1, \ldots, t_r, t'_1, \ldots, t'_s])$.

Proof. Both assertions follow from statement (II)(a)(i) of Lemma A.12 in [12] for $\phi = (q\ x_1\ y_1 \cdots y_r)$, respectively for $\phi = (q\ x_1\ t_1 \cdots t_r)$. □

Example 2. We compose the mtt units M_1 and M_2 from Example 1, yielding the mtt unit $\overline{M_1 M_2} = (\{\overline{qp_1}^{(3)}, \overline{qp_2}^{(3)}\}, \Sigma, \Omega, \overline{R_1 R_2})$ with the following set of rules:

$$\begin{aligned}
&\overline{qp_1}\ (S\ x_1)\ y_{1,p_1}\ y_{1,p_2} = S\ (\overline{qp_2}\ x_1\ (\overline{qp_2}\ x_1\ y_{1,p_1}\ y_{1,p_2})\ y_{1,p_1})\\
&\overline{qp_1}\quad Z\quad y_{1,p_1}\ y_{1,p_2} = y_{1,p_1}\\
&\overline{qp_2}\ (S\ x_1)\ y_{1,p_1}\ y_{1,p_2} = \overline{qp_1}\ x_1\ (\overline{qp_2}\ x_1\ y_{1,p_1}\ y_{1,p_2})\ y_{1,p_1}\\
&\overline{qp_2}\quad Z\quad y_{1,p_1}\ y_{1,p_2} = y_{1,p_2}\ .
\end{aligned}$$

Note that here we have $\mathit{Comp} = \{p_1\ (q\ x\ y_1) = \zeta_{q,p_1}\ ,\ p_2\ (q\ x\ y_1) = \zeta_{q,p_2}\}$, where $\zeta_{q,p_1} = \overline{qp_1}\ x\ (p_1\ y_1)\ (p_2\ y_1)$ and $\zeta_{q,p_2} = \overline{qp_2}\ x\ (p_1\ y_1)\ (p_2\ y_1)$.

Another example of the application of Construction 1 can be found in the introduction, where $f = \overline{expdiv}$ and $g = \overline{expdiv'}$.

An optimizing compiler can take advantage of the composition construction by detecting appropriate places in the program where the rewrite rules from *Comp* can be applied. While the soundness of these rewritings is guaranteed by Lemma 1, it remains to be shown under which conditions Construction 1 actually leads to performance improvements over the original program (using mtt units M_1 and M_2) by the composed program (additionally containing the mtt unit $\overline{M_1 M_2}$, and with rewritings from *Comp* having been applied). From the form of rewrite rules in *Comp* it is obvious that intermediate data structures produced by M_1 in the original program have been removed in the composed program. Thus, no memory cells for this intermediate result have to be allocated in the heap and later be deallocated by the garbage collector. This beneficial effect, however, might be rendered useless, if the composed program needs to perform more reduction steps than the original one. Hence, we would like to establish conditions under which we have for every $q \in Q^{(r+1)}$, $p \in P^{(s+1)}$, $t \in T_\Sigma$, $t_1, \ldots, t_r \in T_\Delta$ and $t'_1, \ldots, t'_s \in T_\Omega$:

$$\begin{aligned} & cbn(\overline{R_1 R_2} \cup R_2, \zeta_{q,p}[x \leftarrow t][y_1, \ldots, y_r, z_1, \ldots, z_s \leftarrow t_1, \ldots, t_r, t'_1, \ldots, t'_s]) \\ \leqq\ & cbn(R_1 \cup R_2, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)\ . \end{aligned}$$

4 Efficiency Analysis

We want to formally relate the efficiency of a composed program obtained by Construction 1 to the efficiency of the original program. As computation time is an intensional property — i.e., it cannot be extracted from the result of a computation — such a study cannot directly use the algebraic methods developed to reason about extensional properties, e.g., to establish the correctness of the composition construction in Lemma 1. A well-known strategy in this situation is to externalize the internal property, e.g., by transforming or annotating programs.

Rosendahl [10] produces for every first-order functional program a step-counting version that — when called with the same arguments — returns the number of call-by-value reduction steps performed by the original program. Sands [11] developed a call-by-name improvement theory and used it to prove the correctness of unfold/fold-transformations [1] by annotating functional programs with a special identity function that represents a single "tick" of computation time, and by stating a set of laws that can be used to derive statements about the relative efficiency of program expressions.

This use of a special symbol to indicate performed reduction steps is similar to what we will be doing in Lemmata 2 and 3 in Sect. 4.2. Note, however, that our transformation technique goes beyond unfold/fold-steps if M_1 is not a tdtt unit, hence Sands' results can neither be used to prove Lemma 1, nor to establish criteria under which Construction 1 improves the efficiency of programs.

4.1 Annotating Programs

In the following, let $M_1 = (Q, \Sigma, \Delta, R_1)$ and $M_2 = (P, \Delta, \Omega, R_2)$ be two fixed mtt units, one of which is a tdtt unit, with $\diamond, \circ, \bullet, \star \notin \Sigma \cup \Delta \cup \Omega \cup Q \cup P$. We use the notions and naming conventions from Construction 1. Based on M_1 and M_2, we define — by annotating and adding rules — several new mtt units, the relevance of which will only become clear later.

Definition 4. *An mtt unit* $M_1^{\text{left}\to\text{right}}$ *has components* $(Q, \Sigma', \Delta', R_1^{\text{left}\to\text{right}})$ *with* Σ' *and* Δ' *obtained by adding to* Σ *and* Δ *symbols from* $\{\diamond^{(1)}, \circ^{(1)}, \bullet^{(1)}, \star^{(1)}\}$ *that are mentioned in* left *and in* right*, respectively, and with* $R_1^{\text{left}\to\text{right}}$ *containing rules as given in the table below.*
Several mtt units $M_2^{\text{left}\to\text{right}} = (P, \Delta', \Omega', R_2^{\text{left}\to\text{right}})$ *are introduced analogously.*

$R_1^{\to\diamond}$	$: q\ (\sigma\ x_1 \cdots x_k)\ y_1 \cdots y_r = \diamond\ (rhs_{M_1,q,\sigma})$	$\forall q \in Q^{(r+1)}, \sigma \in \Sigma^{(k)}$
$R_2^{\diamond\to\bullet}$	$: p\ (\delta\ x_1 \cdots x_k)\ z_1 \cdots z_s = \bullet\ (rhs_{M_2,p,\delta})$	$\forall p \in P^{(s+1)}, \delta \in \Delta^{(k)}$
	$p\quad(\diamond\ x_1)\quad z_1 \cdots z_s = \bullet\ (p\ x_1\ z_1 \cdots z_s)$	$\forall p \in P^{(s+1)}$
$R_2^{\to\bullet}$	$: p\ (\delta\ x_1 \cdots x_k)\ z_1 \cdots z_s = \bullet\ (rhs_{M_2,p,\delta})$	$\forall p \in P^{(s+1)}, \delta \in \Delta^{(k)}$
$R_2^{\diamond\to\circ}$	$: p\ (\delta\ x_1 \cdots x_k)\ z_1 \cdots z_s = rhs_{M_2,p,\delta}$	$\forall p \in P^{(s+1)}, \delta \in \Delta^{(k)}$
	$p\quad(\diamond\ x_1)\quad z_1 \cdots z_s = \circ\ (p\ x_1\ z_1 \cdots z_s)$	$\forall p \in P^{(s+1)}$
$R_2^{\to\star}$	$: p\ (\delta\ x_1 \cdots x_k)\ z_1 \cdots z_s = \star\ (rhs_{M_2,p,\delta})$	$\forall p \in P^{(s+1)}, \delta \in \Delta^{(k)}$
$R_2^{\circ\bullet\to\circ\bullet}$	$: p\ (\delta\ x_1 \cdots x_k)\ z_1 \cdots z_s = rhs_{M_2,p,\delta}$	$\forall p \in P^{(s+1)}, \delta \in \Delta^{(k)}$
	$p\quad(\circ\ x_1)\quad z_1 \cdots z_s = \circ\ (p\ x_1\ z_1 \cdots z_s)$	$\forall p \in P^{(s+1)}$
	$p\quad(\bullet\ x_1)\quad z_1 \cdots z_s = \bullet\ (p\ x_1\ z_1 \cdots z_s)$	$\forall p \in P^{(s+1)}$
$R_1^{\to\circ\bullet,0}$	$: q\ (\sigma\ x_1 \cdots x_k)\ y_1 \cdots y_r =$ $\circ\ (rhs_{M_1,q,\sigma}[(\delta \cdots) \leftarrow \bullet\ (\delta \cdots) \mid \delta \in \Delta])$	$\forall q \in Q^{(r+1)}, \sigma \in \Sigma^{(k)}$
$R_1^{\to\circ\bullet,1}$	$: q\ (\sigma\ x_1 \cdots x_k)\ y_1 \cdots y_r =$ $rhs_{M_1,q,\sigma}\,[(\delta \cdots) \leftarrow \bullet\ (\delta \cdots) \mid \delta \in \Delta]$ $[(q' \cdots) \leftarrow \circ\ (q' \cdots) \mid q' \in Q]$	$\forall q \in Q^{(r+1)}, \sigma \in \Sigma^{(k)}$
$R_2^{\diamond\to\circ\bullet}$	$: p\ (\delta\ x_1 \cdots x_k)\ z_1 \cdots z_s = \bullet\ (rhs_{M_2,p,\delta})$	$\forall p \in P^{(s+1)}, \delta \in \Delta^{(k)}$
	$p\quad(\diamond\ x_1)\quad z_1 \cdots z_s = \circ\ (p\ x_1\ z_1 \cdots z_s)$	$\forall p \in P^{(s+1)}$

Note that if M_1 fulfills one of the restrictions introduced in Definition 3, then also all the introduced $M_1^{\text{left}\to\text{right}}$ fulfill this restriction; analogously for M_2.

4.2 Ticking of Original Program

Note that $M_1^{\to\diamond}$ from Definition 4 outputs a $\diamond$-symbol in every rule application. If none of these symbols is afterwards duplicated, then the number of $\diamond$-symbols in the output produced by $M_1^{\to\diamond}$ is exactly the number of performed call-by-need reduction steps. This fact is used in the following lemma to determine the efficiency of the sequential composition of M_1 and M_2. For the rest of the paper, we fix some $q \in Q^{(r+1)}$, $p \in P^{(s+1)}$, $t \in T_\Sigma$, $t_1, \ldots, t_r \in T_\Delta$ and $t'_1, \ldots, t'_s \in T_\Omega$, and the substitution $\kappa = [y_1, \ldots, y_r, z_1, \ldots, z_s \leftarrow t_1, \ldots, t_r, t'_1, \ldots, t'_s]$.

Lemma 2. *If* M_1 *is context-linear or basic, and* M_2 *is atmost, then:*
$cbn(R_1 \cup R_2, p\,(q\,t\,t_1 \cdots t_r)\,t'_1 \cdots t'_s) = |nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\bullet}}, p\,(q\,t\,t_1 \cdots t_r)\,t'_1 \cdots t'_s)|_\bullet$

Proof. In every application of a rule from $R_1^{\to\diamond}$ (corresponding to an R_1-step), one $\diamond$-symbol is produced in the intermediate result. There is no other way how $\diamond$-symbols can be introduced, and since $M_1^{\to\diamond}$ is *(i)* context-linear or *(ii)* basic — i.e, no $\diamond$-symbols will appear in context parameters of states from Q in case *(ii)*, or those parameters cannot be copied in case *(i)* — none of these $\diamond$-symbols will be duplicated. Only those $\diamond$-symbols are reached and reproduced as $\bullet$-symbols by $R_2^{\diamond\to\bullet}$ that correspond to an R_1-step being forced by call-by-need reduction of $(p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)$. Since $M_2^{\diamond\to\bullet}$ is recursion-linear, every $\diamond$-symbol will be reproduced as $\bullet$-symbol at most once. Since $M_2^{\diamond\to\bullet}$ is context-linear or basic, those $\bullet$-symbols and also the $\bullet$-symbols produced by $M_2^{\diamond\to\bullet}$ at Δ-symbols — corresponding to the R_2-steps during the call-by-need reduction — will not be duplicated. Hence, the resulting number of $\bullet$-symbols is equal to the number of steps of R_1 and R_2 in the original program. □

Example 3. Recall the mtt units M'_1 and M_2 from Example 1, and $M_1'^{\to\diamond}$ and $M_2^{\diamond\to\bullet}$ as obtained from them according to Definition 4. Note that M'_1 is basic and M_2 is atmost, hence the preconditions of Lemma 2 are satisfied. With $t = S\ Z$ and $t_1 = t_2 = \alpha$, we have, e.g., the following reduction:

$$
\begin{array}{ll}
 & p_2\ (q'\ t\ t_1\ t_2) \\
\Rightarrow_{R_1'^{\to\diamond}} & p_2\ (\diamond\ (\delta\ (q'\ Z\ t_2\ (\gamma\ t_1))\ (\gamma\ (q'\ Z\ t_1\ t_2)))) \qquad \text{(mark step of } R'_1\text{)} \\
\Rightarrow_{R_2^{\diamond\to\bullet}} & \bullet\ (p_2\ (\delta\ (q'\ Z\ t_2\ (\gamma\ t_1))\ (\gamma\ (q'\ Z\ t_1\ t_2)))) \qquad \text{(count marked step)} \\
\Rightarrow_{R_1'^{\to\diamond}} & \bullet\ (p_2\ (\delta\ (\diamond\ t_2)\ (\gamma\ (q'\ Z\ t_1\ t_2)))) \qquad \text{(will } \textit{not} \text{ be counted)} \\
\Rightarrow_{R_2^{\diamond\to\bullet}} & \bullet\ (\bullet\ (p_1\ (\gamma\ (q'\ Z\ t_1\ t_2)))) \qquad \text{(count step of } R_2\text{)} \\
\Rightarrow_{R_2^{\diamond\to\bullet}} & \bullet\ (\bullet\ (\bullet\ (S\ (p_2\ (q'\ Z\ t_1\ t_2))))) \qquad \text{(count step of } R_2\text{)} \\
\Rightarrow_{R_1'^{\to\diamond}} & \bullet\ (\bullet\ (\bullet\ (S\ (p_2\ (\diamond\ t_1))))) \qquad \text{(mark step of } R'_1\text{)} \\
\Rightarrow_{R_2^{\diamond\to\bullet}} & \bullet\ (\bullet\ (\bullet\ (S\ (\bullet\ (p_2\ t_1))))) \qquad \text{(count marked step)} \\
\Rightarrow_{R_2^{\diamond\to\bullet}} & \bullet\ (\bullet\ (\bullet\ (S\ (\bullet\ (\bullet\ Z))))) \qquad \text{(count step of } R_2\text{)}
\end{array}
$$

Indeed, we have $cbn(R'_1 \cup R_2, p_2\ (q'\ t\ t_1\ t_2)) = |\bullet\ (\bullet\ (\bullet\ (S\ (\bullet\ (\bullet\ Z))))))|_\bullet = 5$. Note that a step of R'_1 that is not forced by call-by-need evaluation is not counted in the final output.

If we refrain from counting the steps of M_1 and settle for the approximation of only counting the steps of M_2, we can drop part of the restrictions in Lemma 2:

Lemma 3. *If M_2 is context-linear or basic, then:*
$cbn(R_1 \cup R_2, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s) > |nf(\Rightarrow_{R_1 \cup R_2^{\to\bullet}}, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)|_\bullet$

Proof. Every step of $R_2^{\to\bullet}$ (corresponding to an R_2-step) produces one $\bullet$-symbol. Since $M_2^{\to\bullet}$ is context-linear or basic, no such symbol is duplicated. Hence, the number of $\bullet$-symbols in $nf(\Rightarrow_{R_1 \cup R_2^{\to\bullet}}, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)$ is equal to the number of R_2-steps during call-by-need reduction of $(p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)$. Since during this reduction at least one step must be performed by R_1, the inequality follows. □

4.3 Ticking of Composed Program

Assume that Construction 1 composes M_1 and M_2 to $\overline{M_1 M_2} = (F, \Sigma, \Omega, \overline{R_1 R_2})$. Since $M_1^{\to\diamond}$ or $M_2^{\diamond\to\circ}$ is a tdtt unit, Construction 1 is also applicable to these mtt units, yielding $\overline{M_1^{\to\diamond} M_2^{\diamond\to\circ}} = (F, \Sigma, \Omega \cup \{\circ^{(1)}\}, \overline{R_1^{\to\diamond} R_2^{\diamond\to\circ}})$.

Example 4. For $M_1^{\to\diamond}$ and $M_2^{\diamond\to\circ}$ as obtained from the mtt units in Example 1, Construction 1 yields the mtt unit $\overline{M_1^{\to\diamond} M_2^{\diamond\to\circ}}$ with set of rules:

$$\begin{aligned}
&\overline{qp_1}\ (S\ x_1)\ y_{1,p_1}\ y_{1,p_2} = \circ\ (S\ (\overline{qp_2}\ x_1\ (\overline{qp_2}\ x_1\ y_{1,p_1}\ y_{1,p_2})\ y_{1,p_1}))\\
&\overline{qp_1}\quad Z\quad y_{1,p_1}\ y_{1,p_2} = \circ\ y_{1,p_1}\\
&\overline{qp_2}\ (S\ x_1)\ y_{1,p_1}\ y_{1,p_2} = \circ\ (\overline{qp_1}\ x_1\ (\overline{qp_2}\ x_1\ y_{1,p_1}\ y_{1,p_2})\ y_{1,p_1})\\
&\overline{qp_2}\quad Z\quad y_{1,p_1}\ y_{1,p_2} = \circ\ y_{1,p_2}\ .
\end{aligned}$$

In the previous example, the rules obtained by the composition construction for $M_1^{\to\diamond}$ and $M_2^{\diamond\to\circ}$ correspond to the rules obtained in Example 2 by composing M_1 and M_2, except that every right-hand side has an additional $\circ$-symbol on top. This is due to the $\diamond$-symbols on top of all right-hand sides of $R_1^{\to\diamond}$ and due to the added rules $p_1\ (\diamond\ x_1) = \circ\ (p_1\ x_1)$ and $p_2\ (\diamond\ x_1) = \circ\ (p_2\ x_1)$ in $R_2^{\diamond\to\circ}$. The following lemma establishes that this observation holds in general.

Lemma 4. *For every $f \in F$ and $\sigma \in \Sigma$: $rhs_{\overline{M_1^{\to\diamond} M_2^{\diamond\to\circ}},f,\sigma} = \circ\ (rhs_{\overline{M_1 M_2},f,\sigma})$.*

Proof. Straightforward by noting that the definitions of *Pre* and *Comp* only depend on Q and P, and that for every $q \in Q$, $p \in P^{(s+1)}$ and $\sigma \in \Sigma$:

$$\begin{aligned}
&nf(\Rightarrow_{R_2^{\diamond\to\circ} \cup Pre \cup Comp}, p\ (\diamond\ (rhs_{M_1,q,\sigma}))\ z_1 \cdots z_s)\\
&= nf(\Rightarrow_{R_2^{\diamond\to\circ} \cup Pre \cup Comp}, \circ\ (p\ rhs_{M_1,q,\sigma}\ z_1 \cdots z_s))\\
&= \circ\ (nf(\Rightarrow_{R_2 \cup Pre \cup Comp}, p\ rhs_{M_1,q,\sigma}\ z_1 \cdots z_s))
\end{aligned}$$ □

Hence, $\overline{M_1^{\to\diamond} M_2^{\diamond\to\circ}}$ can be used to approximate the number of reduction steps of $\overline{M_1 M_2}$, as exploited in the proof of the following lemma, which estimates the efficiency of the composed program without actually performing Construction 1.

Lemma 5. $$\begin{aligned}
&cbn(\overline{R_1 R_2} \cup R_2, \zeta_{q,p}[x \leftarrow t]\kappa)\\
&\leqq |nf(\Rightarrow_{R_2^{\to\star}}, (nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\circ}}, p\ (q\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s))\,\kappa)|_{\{\circ,\star\}}
\end{aligned}$$

Proof. By Lemma 4, we know that for every $f \in F$ and $\sigma \in \Sigma$, the following holds: $rhs_{\overline{M_1^{\to\diamond} M_2^{\diamond\to\circ}},f,\sigma} = \circ\ (rhs_{\overline{M_1 M_2},f,\sigma})$. Since all rules simply have an additional symbol on top, we get the following equivalence:

$$cbn(\overline{R_1 R_2} \cup R_2, \zeta_{q,p}[x \leftarrow t]\kappa) = cbn(\overline{R_1^{\to\diamond} R_2^{\diamond\to\circ}} \cup R_2^{\to\star}, \zeta_{q,p}[x \leftarrow t]\kappa).$$

During call-by-need reduction of $\zeta_{q,p}[x \leftarrow t]\kappa$ with $\Rightarrow_{\overline{R_1^{\to\diamond} R_2^{\diamond\to\circ}} \cup R_2^{\to\star}}$, every step of $\overline{R_1^{\to\diamond} R_2^{\diamond\to\circ}}$, produces a $\circ$-symbol, while every step of $R_2^{\to\star}$ produces a $\star$-symbol. Hence, the overall number of steps is $\leqq |nf(\Rightarrow_{\overline{R_1^{\to\diamond} R_2^{\diamond\to\circ}} \cup R_2^{\to\star}}, \zeta_{q,p}[x \leftarrow t]\kappa)|_{\{\circ,\star\}}$. By confluence considerations, we can obtain the same normal form by first reducing $\zeta_{q,p}[x \leftarrow t]$ to its normal form with respect to $\Rightarrow_{\overline{R_1^{\to\diamond} R_2^{\diamond\to\circ}}}$, then performing the substitution κ and further reducing to normal form with $\Rightarrow_{R_2^{\to\star}}$. This gives us the equivalent expression $|nf(\Rightarrow_{R_2^{\to\star}}, (nf(\Rightarrow_{\overline{R_1^{\to\diamond} R_2^{\diamond\to\circ}}}, \zeta_{q,p}[x \leftarrow t]))\,\kappa)|_{\{\circ,\star\}}$, which by Lemma 1 from Sect. 3 is equal to:

$$|nf(\Rightarrow_{R_2^{\to\star}}, (nf(\Rightarrow_{R_1^{\to\diamond} \cup R_2^{\diamond\to\circ}}, p\ (q\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s))\,\kappa)|_{\{\circ,\star\}}\,.$$ □

4.4 Combining Approximations

With Lemmata 2 and 3 we have two different ways to estimate the efficiency of the original program, while Lemma 5 approximates the efficiency of the composed program. In each case, this is done by annotating the rules of M_1 and M_2 appropriately and counting certain symbols in the produced output. In the following, we will combine these results to determine the relative efficiency of the composed vs. the original program. Using Lemmata 2 and 5, we obtain the first of our main theorems:

Theorem 1. *If M_1 is context-linear or basic, and M_2 is atmost, then:*
$cbn(\overline{R_1R_2} \cup R_2, \zeta_{q,p}[x \leftarrow t]\kappa) - cbn(R_1 \cup R_2, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s) \leqq 0$

Proof. By Lemmata 5 and 2, we have:

$$\begin{aligned} &cbn(\overline{R_1R_2} \cup R_2, \zeta_{q,p}[x \leftarrow t]\kappa) - cbn(R_1 \cup R_2, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s) \\ &\leqq |\chi|_{\{\circ,\star\}} - |nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)|_\bullet\ , \end{aligned}$$

where $\chi = nf(\Rightarrow_{R_2^{\rightarrow\star}}, (nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\circ}}, p\ (q\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s))\,\kappa)$.
It is clear that:

$$\begin{aligned} &|nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)|_\bullet \\ &= |nf(\Rightarrow_{R_2^{\diamond\rightarrow\bullet}}, (nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, p\ (q\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s))\,\kappa)|_\bullet. \end{aligned}$$

Since we have $t_1, \ldots, t_r \in T_\Delta$, the outer reduction with $\Rightarrow_{R_2^{\diamond\rightarrow\bullet}}$ will only apply rules at symbols from Δ. Hence, the previous expression is equivalent to $|\chi'|_{\{\bullet,\star\}}$, where $\chi' = nf(\Rightarrow_{R_2^{\rightarrow\star}}, (nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\bullet}}, p\ (q\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s))\,\kappa)$.
By comparing the definitions of $R_2^{\diamond\rightarrow\bullet}$ and $R_2^{\diamond\rightarrow\circ}$, it should be obvious that $|\chi|_{\{\circ,\star\}} - |\chi'|_{\{\bullet,\star\}} = |\chi|_\circ - |\chi'|_\bullet \leqq 0$. □

Note that by further considering the value of $|\chi|_\circ - |\chi'|_\bullet$ in the previous proof, we could obtain the more precise statement that under the preconditions of Theorem 1 the composed program saves at least as many reduction steps as were performed in the original program by states of M_2 on the part of the intermediate result produced by rules of M_1.

Also, note that Theorem 1 generalizes the efficiency statements about the composed program compared to the original program in Corollary 21 and Theorem 23 of [8], where M_1 and M_2 were required to be linear, and where in the case that M_1 is not a tdtt unit M_2 was restricted to have only one state.

In order to relax the a priori restrictions imposed by Theorem 1 on the involved mtt units, we can use Lemma 3 (instead of Lemma 2) as starting point:

Lemma 6. *If M_2 is context-linear or basic, then for every $\lambda \in \{0, 1\}$:*

$$\begin{aligned} &cbn(\overline{R_1R_2} \cup R_2, \zeta_{q,p}[x \leftarrow t]\kappa) - cbn(R_1 \cup R_2, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s) \\ &< \lambda + |\chi|_\circ - |\chi|_\bullet\ , \end{aligned}$$

where $\chi = nf(\Rightarrow_{R_2^{\bullet\rightarrow\circ\bullet}}, p\ (nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet,\lambda}}, q\ t\ t_1 \cdots t_r))\ t'_1 \cdots t'_s)$.

Proof. By Lemmata 5 and 3, we have:

$$\begin{aligned} &cbn(\overline{R_1R_2} \cup R_2, \zeta_{q,p}[x \leftarrow t]\kappa) - cbn(R_1 \cup R_2, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s) \\ &< |\chi_1|_{\{\circ,\star\}} - |nf(\Rightarrow_{R_1 \cup R_2^{\rightarrow\bullet}}, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)|_\bullet\ , \end{aligned}$$

where $\chi_1 = nf(\Rightarrow_{R_2^{\rightarrow\star}}, (nf(\Rightarrow_{R_1^{\rightarrow\diamond} \cup R_2^{\diamond\rightarrow\circ}}, p\ (q\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s))\,\kappa)$.
Replacing $R_2^{\diamond\rightarrow\circ}$ by $R_2^{\diamond\rightarrow\circ\bullet}$ in the expression defining χ_1 does not change the

number of occurrences of $\circ$ and $\star$, because only additional $\bullet$-symbols will appear in the output, hence $|\chi_1|_{\{\circ,\star\}} = |\chi_2|_{\{\circ,\star\}}$, where

$$\chi_2 = nf(\Rightarrow_{R_2^{\rightarrow\star}}, (nf(\Rightarrow_{R_1^{\rightarrow\diamond}\cup R_2^{\diamond\rightarrow\circ\bullet}}, p\ (q\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s))\,\kappa)\,.$$

Also, it is clear that:

$$\begin{aligned} &|nf(\Rightarrow_{R_1\cup R_2^{\rightarrow\bullet}}, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)|_\bullet \\ &= |nf(\Rightarrow_{R_1^{\rightarrow\diamond}\cup R_2^{\diamond\rightarrow\circ\bullet}}, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s)|_\bullet \\ &= |nf(\Rightarrow_{R_2^{\diamond\rightarrow\circ\bullet}}, (nf(\Rightarrow_{R_1^{\rightarrow\diamond}\cup R_2^{\diamond\rightarrow\circ\bullet}}, p\ (q\ t\ y_1 \cdots y_r)\ z_1 \cdots z_s))\,\kappa)|_\bullet\ . \end{aligned}$$

Since we have $t_1, \ldots, t_r \in T_\Delta$, the outer reduction with $\Rightarrow_{R_2^{\diamond\rightarrow\circ\bullet}}$ will only apply rules at symbols from Δ, producing $\bullet$-symbols. Hence, the previous expression is equivalent to $|\chi_2|_{\{\bullet,\star\}}$, and thus the right-hand side of the above inequality is equal to $|\chi_2|_{\{\circ,\star\}} - |\chi_2|_{\{\bullet,\star\}}$, respectively to $|\chi_3|_\circ - |\chi_3|_\bullet$, where

$$\chi_3 = nf(\Rightarrow_{R_2}, (nf(\Rightarrow_{R_2^{\diamond\rightarrow\circ\bullet}}, p\ (nf(\Rightarrow_{R_1^{\rightarrow\diamond}}, q\ t\ y_1 \cdots y_r))\ z_1 \cdots z_s))\,\kappa)\,.$$

The rules from $R_2^{\diamond\rightarrow\circ\bullet}$ replace every $\diamond$-symbol by one $\circ$-symbol, and produce a $\bullet$-symbol for every consumed symbol from the intermediate ranked alphabet Δ. The same effect can be achieved by firstly directly producing $\circ$-symbols instead of $\diamond$-symbols, and secondly marking every produced intermediate symbol from Δ with a $\bullet$-symbol and then just reproducing those. Hence, χ_3 is equal to:

$$nf(\Rightarrow_{R_2}, (nf(\Rightarrow_{R_2^{\circ\bullet\rightarrow\circ\bullet}}, p\ (nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet,0}}, q\ t\ y_1 \cdots y_r))\ z_1 \cdots z_s))\,\kappa)\,.$$

Since, furthermore, the $t_1, \ldots, t_r \in T_\Delta$ do not contain $\circ$- or $\bullet$-symbols, and the rules of R_2 are contained in $R_2^{\circ\bullet\rightarrow\circ\bullet}$, the previous expression is by confluence considerations equal to:

$$\chi_4 = nf(\Rightarrow_{R_2^{\circ\bullet\rightarrow\circ\bullet}}, p\ (nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet,0}}, q\ t\ t_1 \cdots t_r))\ t'_1 \cdots t'_s).$$

In $nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet,0}}, q\ t\ t_1 \cdots t_r)$, the rules from $R_1^{\rightarrow\circ\bullet,0}$ produce one $\circ$-symbol whenever a state is applied at some input symbol. Alternatively, we could produce one $\circ$-symbol atop every state call. Hence, χ_4 is also equal to:

$$\chi_5 = nf(\Rightarrow_{R_2^{\circ\bullet\rightarrow\circ\bullet}}, p\ (nf(\Rightarrow_{R_1^{\rightarrow\circ\bullet,1}}, \circ\ (q\ t\ t_1 \cdots t_r)))\ t'_1 \cdots t'_s).$$

In the case $\lambda = 0$, the lemma now follows from $|\chi_4|_\circ - |\chi_4|_\bullet = 0 + |\chi|_\circ - |\chi|_\bullet$, while in the case $\lambda = 1$, the lemma follows from $|\chi_5|_\circ - |\chi_5|_\bullet = 1 + |\chi|_\circ - |\chi|_\bullet$ (by normal form and definition of $|\cdot|_\circ$). □

Lemma 6 is promising, because it estimates the efficiency improvement or deterioration of the composed program over the original one, without the need of actually performing the composition construction. But, in contrast to the above Theorem 1, the approximation is still input-dependent, while we would like to obtain a statement about *all* runs of the original and the composed program.

Since we are interested in the *difference* in the number of occurrences of $\circ$- and $\bullet$-symbols, we can manipulate the right-hand sides of $R_1^{\rightarrow\circ\bullet,\lambda}$, as long as we do not decrease the value of $|\chi|_\circ - |\chi|_\bullet$ in Lemma 6. A trivial way to do this is by removing $\bullet\ (\circ \cdots)$- and $\circ\ (\bullet \cdots)$-contexts, because the rules in $R_2^{\circ\bullet\rightarrow\circ\bullet}$ would just reproduce those. If $\circ$- and $\bullet$-symbols do *not* occur together, certain conditions have to be fulfilled to allow "bringing them closer" without decreasing the overall value of $|\chi|_\circ - |\chi|_\bullet$. Such conditions are established in the following definition and lemma, and are then used to prove our second main theorem.

Definition 5. *The rewrite system Elim contains the following rewrite rules (with variables $u, u_1, u_2, \ldots$):*

1. $\bullet\ (\circ\ u) = u$
2. $\circ\ (\bullet\ u) = u$
3. *if M_2 is atmost, then for every $\delta \in \Delta^{(n)}$ and $i \in [n]$:* $\bullet\ (\delta\ u_1 \cdots u_n) = (\delta\ u_1 \cdots (\bullet\ u_i) \cdots u_n)$
4. *if M_2 is atleast, then for every $\delta \in \Delta^{(n)}$ and $i \in [n]$:* $(\delta\ u_1 \cdots (\bullet\ u_i) \cdots u_n) = \bullet\ (\delta\ u_1 \cdots u_n)$
5. *if M_1 is context-nondeleting and M_2 is atleast, then for every $q' \in Q^{(n+1)}$ and $i \in [n]$:* $(q'\ u\ u_1 \cdots (\bullet\ u_i) \cdots u_n) = \bullet\ (q'\ u\ u_1 \cdots u_n)$

Lemma 7. *Let $M = (Q, \Sigma, \Delta \cup \{\circ^{(1)}, \bullet^{(1)}\}, R)$ be an mtt unit, where Q, Σ and Δ are the ranked alphabets of M_1 (and R will typically contain annotated versions of the rules from M_1, which however is not a technical precondition of this lemma). Assume that M is context-nondeleting, if M_1 is context-nondeleting.*
If we rewrite one right-hand side in R with one rewrite step of Elim, yielding an mtt unit M' with set of rules R', then $|\chi|_\circ - |\chi|_\bullet \leqq |\chi'|_\circ - |\chi'|_\bullet$, where:

$$\chi = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, p\ (nf(\Rightarrow_R, q\ t\ t_1 \cdots t_r))\ t'_1 \cdots t'_s)$$
$$\chi' = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, p\ (nf(\Rightarrow_{R'}, q\ t\ t_1 \cdots t_r))\ t'_1 \cdots t'_s)\ .$$

Proof sketch. The lemma can be proved by structural induction on $t \in T_\Sigma$, using a nested induction on the structures of right-hand sides from R and $R_2^{\circ\bullet\to\circ\bullet}$, respectively, and using — for every $p' \in P^{(s'+1)}$, $n \in \mathbb{N}$, $i \in [n]$, $\tau, \tau_1, \ldots, \tau_n \in T_{\Delta \cup \{\circ^{(1)}, \bullet^{(1)}\}}$, $t' \in T_\Sigma$, and $\delta \in \Delta^{(n)}$ or $q' \in Q^{(n+1)}$ — one of the following properties (depending on the rule from *Elim* that was applied):

1. $|\chi_1|_\circ - |\chi_1|_\bullet \leqq |\chi'_1|_\circ - |\chi'_1|_\bullet$
2. $|\chi_2|_\circ - |\chi_2|_\bullet \leqq |\chi'_2|_\circ - |\chi'_2|_\bullet$
3. $|\chi_3|_\circ - |\chi_3|_\bullet \leqq |\chi'_3|_\circ - |\chi'_3|_\bullet$, if M_2 is atmost
4. $|\chi_4|_\circ - |\chi_4|_\bullet \leqq |\chi'_4|_\circ - |\chi'_4|_\bullet$, if M_2 is atleast
5. $|\chi_5|_\circ - |\chi_5|_\bullet \leqq |\chi'_5|_\circ - |\chi'_5|_\bullet$, if M_1 is context-nondeleting and M_2 is atleast,

where:
$$\chi_1 = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, p'\ (\bullet\ (\circ\ \tau))\ z_1 \cdots z_{s'})$$
$$\chi_2 = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, p'\ (\circ\ (\bullet\ \tau))\ z_1 \cdots z_{s'})$$
$$\chi'_1 = \chi'_2 = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, p'\ \tau\ z_1 \cdots z_{s'})$$
$$\chi_3 = \chi'_4 = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, p'\ (\bullet\ (\delta\ \tau_1 \cdots \tau_n))\ z_1 \cdots z_{s'})$$
$$\chi'_3 = \chi_4 = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, p'\ (\delta\ \tau_1 \cdots (\bullet\ \tau_i) \cdots \tau_n)\ z_1 \cdots z_{s'})$$
$$\chi_5 = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, p'\ (nf(\Rightarrow_R, q'\ t'\ \tau_1 \cdots (\bullet\ \tau_i) \cdots \tau_n))\ z_1 \cdots z_{s'})$$
$$\chi'_5 = nf(\Rightarrow_{R_2^{\circ\bullet\to\circ\bullet}}, p'\ (nf(\Rightarrow_R, \bullet\ (q'\ t'\ \tau_1 \cdots \tau_n)))\ z_1 \cdots z_{s'})\ .$$

Properties 1 and 2 are immediate from $\chi_1 = \bullet\ (\circ\ \chi'_1)$ and $\chi_2 = \circ\ (\bullet\ \chi'_2)$. For $j \in \{3, 4, 5\}$ it can easily be seen that $|\chi_j|_\circ = |\chi'_j|_\circ$. Hence, to validate properties 3–5, it remains to prove that under the appropriate restrictions: $|\chi'_3|_\bullet \leqq |\chi_3|_\bullet$, $|\chi'_4|_\bullet \leqq |\chi_4|_\bullet$, respectively, $|\chi'_5|_\bullet \leqq |\chi_5|_\bullet$. These inequations can be established by using the rule $p'\ (\bullet\ x_1)\ z_1 \cdots z_{s'} = \bullet\ (p'\ x_1\ z_1 \cdots z_{s'})$ in $R_2^{\circ\bullet\to\circ\bullet}$, and the following two facts[2]:

[2] For property 5 we additionally use the observation that for context-nondeleting M, $nf(\Rightarrow_R, q'\ t'\ \tau_1 \cdots \tau_n)$ is obtained from $nf(\Rightarrow_R, q'\ t'\ \tau_1 \cdots (\bullet\ \tau_i) \cdots \tau_n)$ by removing at least one $\bullet$-symbol.

Fact 1: if M_2 is atmost, $\tau' \in T_{\Delta \cup \{\circ^{(1)}, \bullet^{(1)}\}}$, and τ'' is obtained by inserting at most one $\bullet$-symbol into τ', then:
$|nf(\Rightarrow_{R_2^{\circ}\bullet\to\circ\bullet}, p'\ \tau''\ z_1 \cdots z_{s'})|_\bullet \leqq 1 + |nf(\Rightarrow_{R_2^{\circ}\bullet\to\circ\bullet}, p'\ \tau'\ z_1 \cdots z_{s'})|_\bullet$.

Fact 2: if M_2 is atleast, $\tau' \in T_{\Delta \cup \{\circ^{(1)}, \bullet^{(1)}\}}$, and τ'' is obtained by removing at least one $\bullet$-symbol from τ', then:
$1 + |nf(\Rightarrow_{R_2^{\circ}\bullet\to\circ\bullet}, p'\ \tau''\ z_1 \cdots z_{s'})|_\bullet \leqq |nf(\Rightarrow_{R_2^{\circ}\bullet\to\circ\bullet}, p'\ \tau'\ z_1 \cdots z_{s'})|_\bullet$. □

Theorem 2. *If M_2 is context-linear or basic, and there exists $\lambda \in \{0, 1\}$, such that the rules of $R_1^{\to\circ\bullet,\lambda}$ can be rewritten with (finitely many applications of) Elim until no $\circ$-symbols remain, then:*

$$cbn(\overline{R_1 R_2} \cup R_2, \zeta_{q,p}[x \leftarrow t]\kappa) - cbn(R_1 \cup R_2, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s) < \lambda$$

Proof. By the preconditions we know that $R_1^{\to\circ\bullet,\lambda}$ can be rewritten with *Elim* to some R^* containing no $\circ$-symbols. By Lemma 6 and repeated applications of Lemma 7, we then have:

$$cbn(\overline{R_1 R_2} \cup R_2, \zeta_{q,p}[x \leftarrow t]\kappa) - cbn(R_1 \cup R_2, p\ (q\ t\ t_1 \cdots t_r)\ t'_1 \cdots t'_s) < \lambda + |\chi'|_\circ - |\chi'|_\bullet\ ,$$

where $\chi' = nf(\Rightarrow_{R_2^{\circ}\bullet\to\circ\bullet}, p\ (nf(\Rightarrow_{R^*}, q\ t\ t_1 \cdots t_r))\ t'_1 \cdots t'_s)$. Since R^* produces no $\circ$-symbols, no such symbols can occur in χ', i.e. $|\chi'|_\circ = 0$. Hence, the theorem follows from $|\chi'|_\bullet \in \mathbb{N}$. Actually, we could obtain a more precise statement by further considering the value of $|\chi'|_\bullet$. □

Note that since Theorem 2 is based on the approximation from Lemma 3, which disregarded the steps performed by M_1 in the original program, we actually obtain that the composed program performs at most as many reduction steps as were performed in the original program by states of M_2, plus λ.

4.5 Application

In Theorems 1 and 2 we have given criteria under which the composed program obtained from Construction 1 is at least as efficient as the original program. Note that these sufficient conditions for non-deterioration can be checked on the *original* program, hence an optimizing compiler will perform the composition construction only if it has ensured beforehand that this is indeed beneficial.

As an example, consider $e = (div\ (exp\ t\ Z))$ from the introduction. Since the mtt unit defining *exp* is context-linear and the tdtt unit defining *div* and *div*$'$ is atmost, Theorem 1 is sufficient to automatically decide that a replacement of e by e' is safe with respect to efficiency (for every possible input t).

Note that also Theorem 2 is constructive, in the sense that we can *algorithmically* decide, whether there exists an $\lambda \in \{0, 1\}$ such that the rules of a given $R_1^{\to\circ\bullet,\lambda}$ can be rewritten with *Elim* until no $\circ$-symbols remain, because the right-hand sides of mtt rules are finite trees and none of the rewrite rules in *Elim* introduces new symbols.

Example 5. Recall the mtt units M_1 and M_2 from Example 1 (composed in Example 2). Since M_2 is context-linear, Theorem 2 is applicable as follows. Consider

the rules in ${R_1}^{\rightarrow\circ\bullet,1}$ (i.e., $\lambda = 1$):

$$q\ (S\ x_1)\ y_1 = \bullet\ (\gamma\ (\circ\ (q\ x_1\ (\bullet\ (\delta\ (\circ\ (q\ x_1\ y_1))\ y_1)))))$$
$$q\quad Z\quad y_1 = y_1$$

Since M_2 is atmost, *Elim* contains the rewrite rules of points 1–3 in Definition 5 from the previous subsection. Hence, we can rewrite as follows:

$rhs_{{M_1}^{\rightarrow\circ\bullet,1},q,S} \Rightarrow_{Elim(3)} \gamma\ (\bullet\ (\circ\ (q\ x_1\ (\bullet\ (\delta\ (\circ\ (q\ x_1\ y_1))\ y_1)))))$
$\Rightarrow_{Elim(1)} \gamma\ (q\ x_1\ (\bullet\ (\delta\ (\circ\ (q\ x_1\ y_1))\ y_1))) \Rightarrow_{Elim(3)} \gamma\ (q\ x_1\ (\delta\ (\bullet\ (\circ\ (q\ x_1\ y_1)))\ y_1))$
$\Rightarrow_{Elim(1)} \gamma\ (q\ x_1\ (\delta\ (q\ x_1\ y_1)\ y_1))$.

From Theorem 2 it now follows that for every $t \in T_\Sigma$ and $t_1 \in T_\Delta$:

$$cbn(\overline{R_1 R_2} \cup R_2, \overline{qp_1}\ t\ (p_1\ t_1)\ (p_2\ t_1)) - cbn(R_1 \cup R_2, p_1\ (q\ t\ t_1)) < 1$$
$$cbn(\overline{R_1 R_2} \cup R_2, \overline{qp_2}\ t\ (p_1\ t_1)\ (p_2\ t_1)) - cbn(R_1 \cup R_2, p_2\ (q\ t\ t_1)) < 1\ .$$

4.6 Results about Classical Deforestation

Kühnemann [8] compares tree transducer composition with classical deforestation [13]. From his Lemma 20 follows that if M_1 is a tdtt unit, then our composition construction essentially yields the same result as classical deforestation with implicit let-expressions. Hence, Theorems 1 and 2 can be used to establish conditions under which classical deforestation for lazy languages is guaranteed to improve efficiency even for nonlinear programs:

Corollary 1. *In the following cases, classical deforestation leads to a program at least as efficient as the original program*[3]*:*

1. *M_1 is a tdtt unit and M_2 is an atmost mtt unit.*
2. *M_1 is a tdtt unit, every rule of which has a right-hand side of the form $(\delta \cdots)$ for some $\delta \in \Delta$, and M_2 is a context-linear or basic mtt unit.*

Proof. The proposition for case 1 follows from Theorem 1. In case 2 the proposition follows from Theorem 2 with $\lambda = 0$, applying the *Elim*-rule 2 from Definition 5. □

5 Future Work

The presented efficiency analysis gives sufficient conditions for when call-by-need reduction of the composed program to normal form does not need more steps than for the original expression, independent from the input. In lazy functional programs, however, such expressions might also occur in a context where their reduction to normal form is not necessary. Hence, the analysis should be made context-independent. We conjecture that the efficiency statement from Theorem 1 remains valid also for partial call-by-need reductions, as does the statement of Theorem 2, if the *Elim*-rules under points 4 and 5 of Definition 5 are abandoned.

[3] By the remarks below the proofs of Theorems 1 and 2, we can even show that the resulting program performs strictly fewer reduction steps than the original program.

Voigtländer and Kühnemann [12] presented a new composition technique that generalizes the transformation considered here, by handling also cases where both involved mtt units use accumulating parameters. Our formal efficiency analysis method is also applicable to the extended transformation and then successfully classifies all examples from [7,8,9,12], but due to space constraints we could not elaborate on this more general setting in the present paper.

Acknowledgment

I would like to thank Armin Kühnemann and the anonymous referees for helpful comments and suggestions.

References

1. R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *J. ACM*, 24:44–67, 1977.
2. W.N. Chin. Safe fusion of functional expressions II: Further improvements. *J. Funct. Prog.*, 4:515–555, 1994.
3. N. Dershowitz and J.P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 243–320. Elsevier Science Publishers B.V., 1990.
4. J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31:71–145, 1985.
5. Z. Fülöp and H. Vogler. *Syntax-Directed Semantics—Formal Models Based on Tree Transducers*. Monographs in Theoretical Computer Science. Springer-Verlag, 1998.
6. M. Höff. Vergleich von Verfahren zur Elimination von Zwischenergebnissen bei funktionalen Programmen. Master thesis, Dresden University of Technology, 1999.
7. A. Kühnemann. Benefits of tree transducers for optimizing functional programs. In *Foundations of Software Technology & Theoretical Computer Science, Chennai, India, Proceedings*, volume 1530 of *LNCS*, pages 146–157. Springer-Verlag, 1998.
8. A. Kühnemann. Comparison of deforestation techniques for functional programs and for tree transducers. In *Functional and Logic Programming, Tsukuba, Japan, Proceedings*, volume 1722 of *LNCS*, pages 114–130. Springer-Verlag, 1999.
9. A. Kühnemann and J. Voigtländer. Tree transducer composition as deforestation method for functional programs. Technical Report TUD-FI01-07, Dresden University of Technology, 2001.
10. M. Rosendahl. Automatic complexity analysis. In *Functional Programming Languages and Computer Architecture, London, England, Proceedings*, pages 144–156. ACM Press, 1989.
11. D. Sands. Total correctness by local improvement in the transformation of functional programs. *ACM Trans. on Prog. Lang. and Systems*, 18:175–234, 1996.
12. J. Voigtländer and A. Kühnemann. Composition of functions with accumulating parameters. Technical Report TUD-FI01-08, Dresden University of Technology, 2001. *http://wwwtcs.inf.tu-dresden.de/~voigt/TUD-FI01-08.ps.gz*.
13. P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoret. Comput. Sci.*, 73:231–248, 1990.
14. C.P. Wadsworth. *Semantics and Pragmatics of the Lambda Calculus*. PhD thesis, Oxford University, 1971.