Automatic Transformations by Rewriting Techniques *.

Françoise Bellegarde

Oregon Graduate Institute of Science & Technology
PO Box 91000 Portland, Oregon
bellegar@cse.ogi.edu

Abstract. The paper shows how term rewriting techniques can be used to automatically transform first-order functional programs by both *deforestation* (eliminating useless intermediate data structures) and *tupling* (eliminating parallel traversals of identical data structures). Its novelty is that it includes these strategies for program improvement in a transformation system which uses completion procedures to automatically control a unfold/fold methodology. This means that eurekas for these strategies are automatically discovered and that they are processed by a completion procedure. The completion procedure is automatically constrained for orienting pairs into rules and for producing critical pairs. An interesting result is that the process preserves termination of the original set of rules, which is not guaranteed in general by a unfold/fold method.

Introduction

As it has often been said, functional programs are constructed using only functions as pieces. Data structures such as lists and trees are the glue to hold them together. Although this compositional style of programming is attractive, it comes at the expense of efficiency. Compositions produce many intermediate data structures when computed in an eager (call-by-value) evaluation. One way to circumvent this problem is to perform *deforestation* on programs as advocated by Wadler [17]. Several approaches for eliminating useless intermediate data structures have been proposed. First came the algorithm proposed by Wadler [17] which performs automatic deforestation on a restricted class of terms called *treeless* terms. Later, Chin's remarkable work on fusion [7] applies to a wider class of *e-treeless* terms and to higher-order programs in general. More recently, promotion theorems have been utilized to *normalize* programs [16]. This technique is applicable to a class of *potentially normalizable terms*. Also an automatic way to implement deforestation inside the Haskell's compiler has been shown in [11].

Deforestation algorithms do not recognize that an expression contains two or more functions that consume the same data structure. Such functions create a

* The work reported here is supported by the contract with Air Force Material Command (F1928-R-0032)

"parallel" traversal of a data structure. These functions can be put together in a tuple as a single function that traverses the data structure only once. This is another way of transforming programs according to the tupling lemma [10].

General purpose program transformation systems are based on a unfold/fold method [6]. Deforestation and tupling are particular instances of this strategy. In the Focus system [14], folding and unfolding are seen as rewritings. It has been pointed out in [9] that an unfold/fold strategy can be controlled by a completion procedure. Following this idea, the transformation system Astre [3, 4] is based on completion procedures. Astre takes into account of inductive laws provided by the user during the completion process. All these systems are interactive.

The paper shows how deforestation and tupling are automatable using completion. These strategies are implemented inside the system Astre. The programs are presented as a set of first-order equations. The system limits the number of critical pairs that are produced by the completion procedure. But because its purpose is to be general, it still generates excessive critical pairs for deforestation and tupling strategies. The paper presents a way to restrict the overlaps between left-hand sides of rules so that the completion procedure computes exactly the pairs that are needed. For automating the process, not only the production of critical pairs needs to be limited but the orientation of the critical pairs into rules has to be automated. Moreover, automatization demands that the combinations of function candidates for deforestation and the tuples for parallel traversal removals are also discovered algorithmically and used to build what is called a "eureka rule" in the unfold/fold method. Automating the process means that the system itself produces the eureka rules. The discovery of a eureka rule represents one transformation step, accomplished by a completion procedure, which is guaranteed to never fail and to always terminate. We give sufficient conditions that guarantee the correctness of the transformation step, e.g. that a transformation step does not transform a terminating program into a non-terminating one. The system processes one eureka after the other, combining transformation steps until no more deforestation and parallel traversal removals can be done. Sufficient conditions guarantee the termination of the process.

1 Application of Completion to Unfold/Fold Strategies

Basic Notations

Let F be a set of function symbols and V be a set of variables, $T(F, V)$ is the set of terms with symbols in F and variables in V. $V(t)$ is the set of all the variables occurring in t. A position or occurrence within a term t is represented as a finite sequence ω of positive integers describing the path from the root of t to the root of the subterm at that position, denoted by $t|_\omega$. The position of the root of a term t is ϵ. The notation $t = u[s]$ emphasizes that the term t contains s as subterm in the context u. G(t) is the set of the positions of all the function symbols in t. A term s is less than t for the subsumption ordering if and only if t is an instance of s. We write $s \sqsubseteq t$ if a **subterm** of t is an instance of s. A term t is said to be linear if no variable occurs more than once in t.

A rewrite rule is an ordered pair of terms, written as $l \rightarrow r$, where $V(r) \subseteq V(l)$. A rule $l \rightarrow r$ is *left-linear* if l is linear, it is *right-linear* if r is linear and, it is *variable preserving* if $V(l) = V(r)$. A rewrite system is a set of rewrite rules. The rewriting relation is denoted as $\rightarrow_R$ with its transitive closure denoted as $\rightarrow_R^+$ and its reflexive and transitive closure denoted as $\rightarrow_R^*$. The rewrite system R *is terminating* if and only if there is no infinite sequence of terms $t_1, t_2, \ldots$, such that $t_1 \rightarrow_R t_2 \rightarrow_R \ldots$ The *R-normal form of a term* t is a term $t \downarrow_R$ such that $t \rightarrow_R^* t \downarrow_R$ and there is no u such that $t \downarrow_R \rightarrow_R u$. Some particular[1] well-founded orderings $<$ allow to prove termination of a rewrite system R by proving only that $l > r$ for each rule $l \rightarrow r$ in R. One of them is the recursive path ordering [8] which is based on a precedence (well-founded quasi-ordering) of symbols.

A rewrite system is *overlapping* if there exists an *overlap* between left-hand sides of two rules $g \rightarrow d$ and $l \rightarrow r$, i.e. if there exists a position ω in $G(l)$ such that $l|_\omega$ and g are unifiable with the most general unifier σ. A *critical pair* is the identity $\sigma(l[\omega \leftarrow \sigma(d)]) = \sigma(r)$ where $t[\omega \leftarrow u]$ denotes the replacement in t of the subterm at position ω by u.

An *orthogonal* system is a left linear and non-overlapping rewrite system. A system is *constructor-based* if all proper subterms of its left-hand sides have only free constructor symbols and variables. The roots of left-hand sides are *defined symbols*. C and D denote respectively the set of constructors and the set of defined symbols. R_f is the set of all the rules $l \rightarrow r$ of a constructor-based rewrite system R where the root of l is f. A rewrite system is *confluent* if and only if the relation $\rightarrow^*$ verifies the diamond property. Confluence ensures the unicity of the normal form while termination ensures its existence. A non-overlapping and terminating rewrite system is confluent. A completion procedure aims at discovering critical pairs in a terminating rewrite system R to check whether the two sides of the pair rewrite to the same term. Otherwise, it adds the critical pairs to R, orienting them in such a way as to preserve the termination property. If the procedure does not fail and terminates, it returns a confluent and terminating system equivalent to R.

Completion Procedure and Unfold/Fold Method

The unfold/fold method [6] consists of 6 rules, namely *Definition, Instantiation, Unfolding, Folding, Abstraction*, and *Law*, that allow new identities to be introduced that are equational consequences of existing identities. Dershowitz [9] has shown how the combination of *Instantiation* and *Folding* is enabled by critical pair generation. *Unfolding* and *Law* are simplifications by rewriting. *Definition* is the introduction of an *eureka* rule. *Abstraction* is used for a tupling tactic.

Deforestation

Consider a naive example of a single deforestation of one term: $length(x@y)$ where

$$R_{length} : \begin{array}{ll} length([]) & \rightarrow 0 \\ length(x :: xs) & \rightarrow S(length(xs)) \end{array} \qquad R_{@} : \begin{array}{ll} []@y & \rightarrow y \\ (x :: xs)@y & \rightarrow x :: (xs@y) \end{array}$$

[1] These well-founded orderings are fully invariant reduction orderings (see [8]).

S is the successor function. The list x is traversed once to append it to y and once more to count the length of the result. A eureka rule $length(x@y) \rightarrow h(x,y)$ is introduced. It overlaps with rules of $R_{@}$ yielding two critical pairs:

$$\begin{aligned} length(y) &= h([],y) \\ length(x :: (xs@y)) &= h(x :: xs, y) \end{aligned}$$

The last pair simplifies by the second rule in R_{length} into $S(length(xs@y)) = h(x :: xs, y)$. Now h is defined by:

$$\begin{aligned} h([],y) &= length(y) \\ h(x :: xs, y) &= S(length(xs@y)) = S(h(xs,y)) \end{aligned}$$

which makes only one traversal of x to compute the result. For this very simple example, no law is necessary. But suppose the eureka rule is $length(rev(x)) \rightarrow h(x)$ where one rule of R_{rev} is: $rev(x :: xs) \rightarrow rev(xs)@[x]$, we need the rule $(Law)\ length(x@y) \rightarrow length(x) + length(y)$ to simplify the left-hand side of the pair $length(rev(xs@[x])) = h(x :: xs)$ according to the following derivation:

$$\begin{aligned} &length(rev(xs)@[x]) \rightarrow length(rev(xs)) \ + \ length([x]) \rightarrow h(xs) \ + \ length([x]) \\ &\rightarrow^{*} S(h(xs)) \quad yielding\ a\ R_h\ rule \quad h(x :: xs) \rightarrow S(h(xs)) \end{aligned}$$

Tupling Tactic

Consider another naive example, $Ave(x) \rightarrow sum(x)/length(x)$ where

$$R_{sum} : \begin{array}{l} sum([]) \rightarrow 0 \\ sum(x :: xs) \rightarrow x + sum(xs) \end{array} \quad R_{length} : \begin{array}{ll} length([]) & \rightarrow 0 \\ length(x :: xs) & \rightarrow S(length(xs)) \end{array}$$

The list x is traversed twice "in parallel" to compute the average. In this case, we introduce the rules:

$$Eureka : \begin{array}{l} sum(x) \rightarrow fst(h(x)) \\ length(x) \rightarrow snd(h(x)) \\ pair(fst(h(x)), snd(h(x))) \rightarrow h(x) \end{array} \qquad Comp : \begin{array}{ll} fst(pair(x,y)) & \rightarrow x \\ snd(pair(x,y)) & \rightarrow y \end{array}$$

By rewriting the left-hand side with the two first eureka rules, we get: $Ave(x) \rightarrow fst(h(x))/snd(h(x))$ which can be computed with a single traversal of x by sharing the common computation of $h(x)$. The two first eurekas rules overlap respectively with rules of R_{sum} and R_{length} yielding the pairs:

$$\begin{array}{lll} 0 & = fst(h([])) & \quad 0 = snd(h([])) \\ x + sum(xs) & = fst(h(x :: xs)) & \quad S(length(xs)) = snd(h(x :: xs)) \end{array}$$

which can be turned into rules from right to left:

$$\begin{array}{lll} fst(h([])) & \rightarrow 0\ (1) & \quad snd(h([])) \rightarrow 0 \\ fst(h(x :: xs)) & \rightarrow x + sum(xs)\ (2) & \quad snd(h(x :: xs)) \rightarrow S(length(xs)) \end{array}$$

Afterwards, these rules overlap with the third eureka yielding R_h rules:

$$h([]) \rightarrow pair(0,0),\ h(x :: xs) \rightarrow pair(x + fst(h(xs)), S(snd(h(xs))))$$

These last two rules reduce the left-hand sides of Rules 1 and 2. The *Comp* rules further reduce these left-hand sides so that they become identical to the right-hand sides. Then the rules can be deleted. It is worthwhile to notice that this tactic can be applied to transform a function that computes the n^{th} fibonacci number k in time proportional to k itself into a function that computes the same number in only n steps. This example has been used in [15] showing how a completion procedure produces useless explosion of critical pairs when controlling an unfold/fold transformation. Our way of implementing the tupling tactic always generates exactly the needed critical pairs. The completion, used for each transformation step, always terminates. Moreover, an algorithm for automatic deforestation and tupling must recognize terms candidates for such tactics.

2 Eureka Rules Discovery

A term is *normalized* for the above two transformation strategies if its computation traverses each data structure exactly once. It is useful to define the *traversal argument positions* of a defined symbol f:

Definition 1 (traversal argument positions). A symbol f traverses a data structure at **traversal argument positions** $i, i \in (1, \ldots, n)$, if there exists a rule $f(tc_1, tc_2, \ldots, tc_n) \rightarrow r$ in R_f such that $tc_i \notin V$

For example, the symbol @ defined by

$$[] \ @ \ x \ = \ x \qquad (x \ :: \ xs) \ @ \ y \ = \ x \ :: \ (xs \ @ \ y)$$

traverses a data structure *List* with constructors $[], ::$ at argument position 1. The *spine positions* indicate where traversals of data structures are located in a term.

Definition 2 (spine positions). A position ω in a term t is a **spine position** if

- either ω is ϵ, or
- $\omega = u.i$ where u is a spine position, the root f of $t|_u \in D$ and i is a traversal argument position of f.

The spine positions of $(x \ @ \ y) \ @ \ z$ are $\epsilon, 1, 1.1$ but the spine positions of $x \ @ \ (y \ @ \ z)$ are $\epsilon, 1$.

Definition 3. The **deforestation depth** of a term t is the length of the largest spine position in t.

The above terms have deforestation depths respectively equal to 2 and 1. Spine positions allow to control deforestation. For example a term $f(x)@y$ is a *deforestation candidate* because $f(x)$ occurs at a spine position. But the term $x@f(y)$ is not a *deforestation candidate* because $f(y)$ does not occur at a spine position. In particular, @ does not traverse the result of $f(y)$ in the latest term. For

$$revonto([], u) \ = \ u \qquad revonto(x :: xs, u) \ = \ revonto(xs, x :: u)$$

the term $revonto(f(x), y)$ is a deforestation candidate but $revonto(x, f(y))$ is not a deforestation candidate. This motivates us to define a candidate as:

Definition 4 (deforestation candidate). A term s, of deforestation depth greater than 1, is a **deforestation candidate** for a term t in R-normal form for a constructor-based, non-overlapping, and terminating rewrite system R if and only if s is linear, and $s \sqsubseteq t$.

The additional constraint that s does not contain a given symbol f is required for building a eureka rule. For example, in the recursive rule

$$concall(x :: xs) = all(x) \ @ \ concall(xs)$$

the deforestation candidate $all(x)$ @ u does not contain the recursive symbol *concall*. Given a depth n, we prefer a deforestation candidate which is a minimal element for the subsumption ordering. For example the term

$$t = flatten(x)@map\_sqr(rev(y))$$

is a deforestation candidate of depth 2 as well as $s_1 = flatten(x)@u \sqsubseteq t$ and $s_2 = map\_sqr(rev(y)) \sqsubseteq t$. But t is an instance of s_1. s_1 and s_2 are *the best deforestation candidates* of depth 2.

Definition 5 (best deforestation candidate). For a given a depth n, a **best deforestation candidate** for the term t is a minimal element for the subsumption ordering among deforestation candidates of depth n for t.

A best deforestation candidate of greatest depth is used to build a *deforestation eureka*:

Definition 6 (deforestation eureka). Suppose s is a best deforestation candidate for the left-hand side r of a rule $l \rightarrow r \in R_f$. Assume s is constrained to contain no occurrence of f and to be of maximal deforestation depth. Given $l \rightarrow r$, a **deforestation eureka** of R is a rule $s \rightarrow h(x_1, x_2, \ldots, x_n)$ where $\{x_1, x_2, \ldots, x_n\} = V(s)$. The **eureca symbol** h is a new symbol, i.e. h does not occur in R.

Given the rule $f(x, y) \rightarrow flatten(x)@map\_sqr(rev(y))$, $flatten(x)@y \rightarrow h(x, y)$ and $map\_sqr(rev(y)) \rightarrow h'(y)$ are deforestation eurekas. Notice how we can determine that 1 is the unique traversal argument position of h. Two subterms that traverse the same data structure are used to construct *tupling eurekas*:

Definition 7 (tupling eureka). Assume that one of the right-hand sides of the rules of R_f has a subterm $k(u_1, u_2, \ldots, u_n)$. Suppose there exists two distinct, non-variable terms, both $\notin T(C, V)$ u_i and u_j which have the same variables at the spine positions and which are both linear for these variables, then the **tupling eureka** of R_f is constituted by the rules:

$$u_i' \rightarrow fst(h(x_1, x_2, \ldots, x_n)) \ (1) \qquad u_j' \rightarrow snd(h(x_1, x_2, \ldots, x_n)) \ (2)$$
$$pair(fst(h(x_1, x_2, \ldots, x_n)), snd(h(x_1, x_2, \ldots, x_n))) \rightarrow h(x_1, x_2, \ldots, x_n) \ (3)$$

where h is a *eureka symbol*, and u'_i and $u'j$ are renamings of u_i and u_j, respectively. These renamings preserve the variables at the spine positions and make distinct all the variables located at the non-spine positions. Also *fst*, *snd*, and *pair* are reserved symbols (e.g. $\notin D \cup C$), with rewrite rules:

$$fst(pair(x, y)) \rightarrow x \qquad snd(pair(x, y)) \rightarrow y,$$

and $\{x_1, x_2, \ldots, x_n\} = V(u'_i) \cup V(u'_j)$.

It is required that u'_i and u'_j not be variables because a rule such as $x \rightarrow g(x)$ cannot belong to a rewrite system. Otherwise $x \rightarrow g(x) \rightarrow g(g(x)) \rightarrow \ldots$

Automatic discovery of eurekas is based on the above definitions.

3 Critical Pairs and Orientation

For clarity, we consider only the case of deforestation. Nevertheless, all that follows can be extended to tupling. The following theorem justifies the orientation of the deforestation eureka.

Theorem 1. *Let R be a non-overlapping, constructor-based, and terminating rewrite system and $g \rightarrow d$ a deforestation eureka of R_f. The system $R \cup \{g \rightarrow d\}$ is terminating.*

> **Proof:** First, the system $E = \{g \rightarrow d\}$ is terminating. For proof, it is sufficient to take a recursive path ordering where all the symbols occurring in g have a greater precedence than the eureka symbol h. Second, the system E quasi-commutes over R. Indeed, if a rewriting by R follows a rewriting by E, we have
>
> $$t = u[\sigma(g)] \rightarrow_E u[\sigma(d)] \rightarrow_R t'$$
>
> The rewriting by R cannot occur at the occurrence of the eureka symbol h in d. Therefore it occurs either in the context u or it rewrites one of the subterms $\sigma(x_i)$. In the first case, t' is obviously obtained by rewriting first by R and then by E. In the second case, the same can be done because x_i occurs only once in the linear term g. Since R and E are terminating and E quasi-commutes over R, $R \cup E$ is terminating [1, 12]. □

As a consequence, right-hand sides of the rules of R can be normalized by E, yielding a **terminating rewrite system we call $\mathbf{R_{fold}}$**. Overlaps between a deforestation eureka $g \rightarrow d$ and R at the spine positions of g produce critical pairs that substitute terms of $T(C, V)$ into the variables that are located under the eureka symbol. Two kinds of critical pairs are produced. Consider an example:

$$f(x :: xs) \rightarrow (x + x) :: f(xs) \tag{1}$$

$$g(x :: xs) \rightarrow (x * x) :: g(xs) \tag{2}$$

$$zip(x :: xs, y :: ys) \rightarrow (x, y) :: zip(xs, ys)$$

with the deforestation eureka: $zip(f(x), g(y)) \rightarrow h(x, y)$. From rule 1 we get the critical pair:

$$P : zip((x + x) :: f(xs), g(y)) = h(x :: xs, y)$$

If we orient P from left to right, then it overlaps with rule 2 yielding a pair:

$$Q : zip((x + x) :: f(xs), (y * y) :: g(ys)) = h(x :: xs, y :: ys)$$

The pair Q normalizes into a rule of R_h. The pair P is said to be **uncovered** and is oriented from left to right to enable overlap with rule 2. The pair Q is covered and oriented from right to left as part of R_h.

Definition2 (eureka critical pairs). Let R be a constructor-based system. Consider a rule $e : g \rightarrow h(tc_1, tc_2, \ldots, tc_n)$, where h is a eureka symbol and $tc_i \in T(C, V)$, $i \in (1, \ldots, n)$[2]. A **eureka critical pair** is one between e and a rule $l \rightarrow r$ of R built from an overlap at a spine position ω of the term g. More precisely, there exists a unifier σ with range T(C,V), such that $\sigma(g|_\omega) = \sigma(l)$ yielding the critical pair:

$$\sigma(g[\omega \leftarrow \sigma(r)]) = h(\sigma(tc_1), \sigma(tc_2), \ldots, \sigma(tc_n))$$

Definition3. A eureka critical pair $Q{:}g = h(tc_1, tc_2, \ldots, tc_n)$, in R-normal form is **covered** if and only if either

1. for every traversal argument position i of h, $tc_i \notin V$, or
2. there exists no overlap between R and Q.

Condition (1) alone is insufficient. For example the pair $[] = h([], y)$ is covered even if 2 is a traversal argument position for h. Indeed, $h([], y) \rightarrow []$ belongs to R_h. *Uncovered eureka critical pairs, or UCP pairs, are directed pairs from left to right (towards the eureka symbol).* Directed pairs in UCP are not used for rewriting. However, only their left-hand sides are overlapped with R. Left-hand sides of UCP pairs are normalized by R_{fold}. *Covered eureka critical pairs, or RCP rules, are directed from right to left (from the eureka symbol).* Right-hand sides of RCP rules are normalized by $R_{fold} \cup E$ where E is the deforestation eureka. Afterwards they becomes rules of R_h. RCP rules are combined with R_{fold}. This process is described by the set of transition rules in Section 4.

At this point, we must ensure that combining RCP rules with R_{fold} preserves the termination of the rewrite system. Still, the termination of R is not enough to ensure the termination of $R_{fold} \cup RCP$. It is well-known that the unfold/fold method preserves only partial correctness [13]. However, if the system R is left-linear the following theorem ensures termination of the system $R_{fold} \cup R_h$. Linear patterns is a usual requirement in functional programming, therefore this is not such a strong requirement.

Theorem 4. *Assume R is an orthogonal, constructor-based, and terminating rewrite system. Let $T = E^{-1}$ be the* **converse of the deforestation eureka** *of eureka symbol h. Consider also the rewrite system R_{fold}, i.e. R normalized by E. The rewrite system $R_{fold} \cup R_h$ is terminating.*

The reader can find the proof of Theorem 4 in appendix.

[2] e is either a deforestation eureka, or an uncovered critical pair.

4 Transition rules for automatic partial completion

From now on, we simply use **eureka** as a shorthand for deforestation eureka or tupling eureka. The deforestation and tupling process is described here by a set of transition rules. A transition rule, transforms a tuple *(R,E,UCP,RCP,L)* where R is a set of constructor-based rules, E is a set of rules containing a single eureka. UCP is a set of uncovered critical pairs, RCP is the set of covered critical pairs whose right-hand sides have not been normalized yet, L is an optional set of laws that can be used to simplify RCP. ECP denotes the set of eureka critical pairs between $E \cup UCP$ and R.

Eureka
$R, \emptyset, \emptyset, \emptyset, L \Longrightarrow R, E, \emptyset, \emptyset, L \quad$ *if* E *is a eureka for* R
Critical Pair
$R, E, UCP, RCP, L \Longrightarrow R, E, UCP \cup ECP, RCP, L \quad$ *if* $ECP \neq \emptyset$
UCP-Unfolding
$R, E, UCP \cup \{g = d\}, RCP, L \Longrightarrow R, E, UCP \cup \{g' = d\}, RCP, L$
if $g \rightarrow_R g'$
R-Folding
$R \cup \{l \rightarrow r\}, E, UCP, RCP, L \Longrightarrow R \cup \{l \rightarrow r'\}, E, UCP, RCP, L$
if $r \rightarrow_E r'$
Covered Pair
$R, E, UCP \cup \{g = d\}, RCP, L \Longrightarrow R, E, UCP, RCP \cup \{d \rightarrow g\}, L$
if $g = d$ *is covered*
RCP-Unfold/Fold
$R, E, UCP, RCP \cup \{p \rightarrow q\}, L \Longrightarrow R, E, UCP, RCP \cup \{p \rightarrow q'\}, L$
if $q \rightarrow_{R \cup E(\cup L)} q'$
Rule_for_h
$R, E, UCP, RCP \cup \{l \rightarrow r\}, L \Longrightarrow R \cup \{l \rightarrow r\}, E, UCP, RCP, L$
if r *is in* $R \cup E(\cup L)$*-normal form*
Flush
$R, E, UCP, RCP, L \Longrightarrow R, \emptyset, UCP, RCP, L \quad$ *if* $ECP = \emptyset$

For processing a tupling eureka, we must add the simplification of a left-hand side of a rule of R and the deletion of a pair of identical terms borrowed from transition rules of a standard completion procedure. In this case, we must also compute the overlaps between the left-hand sides of rules of a tupling eureka. The above transition rules, **except Eureka**, can be implemented by using a completion procedure. This completion is guaranteed to terminate. However, the entire process is not guaranteed to terminate because it could generate infinitely many eurekas. Assuming supplementary conditions on R, discussed later in the paper, we can derive an algorithm which always terminates and moreover achieves deforestation and parallel traversal removals.

Proposition 1. *Starting with an orthogonal, terminating, and constructor-based rewrite system* R*, and using the above transition rules repeatly until none is applicable results either in a constructor-based and terminating system* R' *equivalent to* R*, or else it generates infinitely new eurekas.*

Note that the proposition does not claim that the final result is free of useless data structures. This is achieved only when we can guarantee that R_h is *fused* for every deforestation eureka where h is the eureka symbol.

4.1 Fusability

Consider a best deforestation candidate $a(b(x))$ yielding the deforestation eureka $E : a(b(x)) \rightarrow h(x)$. According to a nomenclature invented by Chin, let us call the bottom symbol b a *producer*, and a a *consumer*. By overlapping E with rules $B : b(tc) \rightarrow r$ of R_b, we get UCP pairs of the form $a(r) = h(tc)$. If b occurs in r at a position ω i.e. if B is a **recursive rule**, all the symbols located above ω in r are *symbols produced* by g. The **fusion** can be achieved if during the normalization of $a(r)$ by $R(\cup L)$, all the *produced* symbols are *consumed* by a, creating a subterm $a(b(r|_\omega))$ which rewrites into $h(r|_\omega)$.

Definition2. A symbol $k \in F$ **is produced** by f if and only if k occurs at position $u < \omega$ in a recursive rule $l \rightarrow r \in R_f$ where ω is a position of f in r.

Definition3. In a best deforestation candidate, every symbol $b \in D$ at position different from ϵ is a **producer**. Every symbol a which has a producer b as argument is a **consumer**. A producer which is not also a consumer is a **bottom producer**.

Definition4. Let s be a best deforestation candidate of depth 2, left-hand side of the deforestation eureka E. Let b be a bottom producer in s, and B be a recursive rule of R_b. Let a be a consumer of b. The R_h rule $H : l \rightarrow r$, built from an overlap of B and E, is **fused for a** if and only if, either a does not occur in r, or else a occurs at a position ω in r; In this case, if b occurs also in r at a position $\alpha > \omega$, a symbol k occurring at position β, $\alpha > \beta > \omega$, must not be a symbol produced by b.

This means simply that the consumer a has "passed through" the symbols produced by the producer b to form redexes for the deforestation eureka. The above definition can be extended to best deforestation candidates of depth greater than 2. A *directly fusable consumer* a is guaranteed to "pass through" all the symbols produced by a *directly fusable producer* b argument of a.

Definition5. A symbol b is a **directly fusable producer** if and only if

1. every symbol k produced by $b \in C$, and
2. every symbol $f \in D$ occurring in R_b as argument of a produced symbol k (i.e. f can become later a producer symbol for a best candidate in R_h) is a directly fusable producer.

Definition6. A symbol a is a **directly fusable consumer at traversal argument position i** if and only if for every rule $l \rightarrow r$ in R_a

1. $l|_i \in C$ or $l|_i = c(x_1, x_2, \ldots, x_n)$ where $c \in C$, $x_i \in V, i = 1, \ldots, n$, and

2. every occurrence of a in r has only variables as arguments at traversal argument positions and,
3. every symbol $f \in D$ occurring in a right-hand side of R_a which has a variable at a traversal argument position (i.e. f can become a consumer for a best candidate in R_h) is a directly fusable consumer.

Definition 7 (directly fusable best deforestation candidate). A best deforestation candidate t is **directly fusable** if and only if its producers are directly fusable producers and its consumers are directly fusable consumers.

For example, $length(x@y)$ is directly fusable but $length(rev(x))$ is not.

Definition 8. An orthogonal, constructor-based, and terminating **rewrite system R** is **directly fusable** if every best deforestation candidate in R is directly fusable.

Lemma 9 (direct fusability lemma). *If a left-hand side of a deforestation eureka with eureka symbol h is a directly fusable best deforestation candidate, then rules in R_h are fused. Moreover R_h is directly fusable.*

The proof is straightforward by application of the above definitions. Every directly fusable consumer passes through the constructors produced by a directly fusable producer argument to form redexes of the deforestation eureka. The best deforestation candidates of R_h are directly fusable by condition (2) of Definition 5 and by condition (3) of Definition 6.

Consequence 10. *Let us start with an orthogonal, constructor-based, terminating, and directly fusable system R, and with $L = \emptyset$. Using the transition rules repeatly results in an orthogonal, constructor-based, terminating and directly fusable rewrite system R' which contains no deforestation candidate.*

Chin's *e-treeless terms* are similar to our directly fusable terms. If a system R is not directly fusable, laws in the set L can be used to force the fusion. Unfortunately, a fused system R' is not always more efficient than R.

4.2 Improvement

Consider for example the system R:

$$f(x) \to tails(downto(x))$$

$$\begin{array}{ll} tails(x :: xs) \to (x :: xs) :: tails(xs) \quad (1) & downto(0) \to [] \\ tails([]) \to [[]] & downto(s(x)) \to s(x) :: downto(x) \end{array}$$

R is terminating, left-linear, and directly fusable. A procedure based on the transition rules returns the system R':

$$f(x) \to h(x)$$

$$h(0) \to [[]] \qquad h(s(n)) \to (s(n) :: downto(n)) :: h(xs)$$

The list $downto(n) = n :: \ldots :: 3 :: 2 :: 1 :: []$ is computed for each $n = 1, \ldots, x$ in R'. This happens because the right-hand side of Rule (1) is not linear for the *accumulative variable* xs.

Definition 11. Let $F : l \rightarrow r$ be a recursive rule of R_f. A variable x is an **accumulative variable** of F if and only if x occurs in a non-variable proper subterm of l, and x occurs also in r at a position $u > \omega$ where ω is a position of an occurrence of f in r.

Definition 12. A symbol $f \in D$ is **safe** if every rule in R_f is right linear for its accumulative variables.

Lemma 13 (improvement lemma). *Let R be a fusable, terminating, orthogonal, and constructor-based rewrite system. Processing a deforestation eureka $s \rightarrow d$ by the transition rules returns a result R' at least as efficient as R if every symbol in s is safe.*

Let h be the eureka symbol. The right linearity plus fusability guarantee that a producer b in s does not occur in R_h. The data structure computed by b is therefore definitively eliminated.

The safety property of the system R is a too strong requirement. It can be useful to process deforestation eurekas with unsafe left-hand sides. For example with the deforestation eureka

$$map\_sum(tails(downto(x)) \rightarrow h(x),$$

where $map\_sum(x :: xs) = sum(x) + map\_sum(xs)$, $downto(x)$ is consumed by sum. Therefore it is eliminated from the result. Work need to be done to extend the notion of safety. Notice that tupling strategy, when it applies, always succeeds to improve a system R.

4.3 Combining tupling eurekas with deforestation eurekas

The order in which we treat deforestation eurekas with respect to tupling eurekas is worth considering. Ambiguities between both strategies are typified by the following example. Consider the rule

$$F : f(x) \rightarrow k(g_1(x), g_2(x))$$

Suppose 1 is the only traversal argument position of k. There exists a deforestation eureka E_1: $k(g_1(x), y) \rightarrow h_1(x)$, and an tupling eureka E_2:

$$g_1(x) \rightarrow fst(h_2(x)) \quad g_2(x) \rightarrow snd(h_2(x)) \quad pair(fst(h_2(x)), snd(h_2(x))) \rightarrow h_2(x)$$

Starting with E_1 results in $F_1 = \{f(x) \rightarrow h_1(x, g_2(x))\} \cup R_{h_1}$. There is no more tupling eureka for F_1.

Starting with E_2 results in $F_2 = \{f(x) \rightarrow k(fst(h_2(x)), snd(h_2(x)))\} \cup R_{h_2}$ There is no more deforestation eureka because $fst \notin D$ and $snd \notin D$.

Consider again the rule F, but suppose k has two traversal positions 1 and 2. There exists a different deforestation eureka E_1': $k(g_1(x), g_2(y)) \rightarrow h_1'(x, y)$ and the same tupling eureka E_2. Starting with E_1' results in $F_3 = \{f(x) \rightarrow h_1'(x, x)\} \cup R_{h_1'}$.

If all the deforestation eurekas are processed before tupling eurekas, the terms that remain candidates for tupling are only the terms $k(u_1, u_2, \ldots, u_n)$ where k is the reserved symbol *pair*, or k is a primitive symbol, i.e. a symbol that occurs only in right-hand sides of R, or k is not a fusable symbol. When there is an ambiguity between the two strategies, deforestation candidates become unavailable after tupling because *fst*, *snd* and *pair* are non-defined symbols. Moreover, no new deforestation candidate can appear after tupling for the same reason. We choose to process all the deforestation eurekas before the tupling eurekas.

5 Termination of the procedure

As we said earlier, processing a deforestation eureka with eureka symbol h, is likely to generate rules of R_h that contains new deforestation terms so that it can never end.

Lemma 1. *Let G be the set of symbols occurring in a left-hand side g of a deforestation eureka $g \rightarrow d$. Let $S = \bigcup_{f \in G} R_f$. Assume that S is directly fusable, and that there is no deforestation candidate in S. Then, a program which implements the transition rules to transform the system $S \cup g \rightarrow d$ always terminates.*

The assumptions about g ensure that best deforestation candidates in the R_h rules have depths no higher than 2. Proofs of similar results can be found in [17] or in [7]. Suppose we treat only deforestation eureka which obeys the assumptions of the above lemma, then more deforestation terms obeying the assumptions can be available, and so on. This "bottom-up" process must terminate. Therefore if R contains no mutually recursive function, termination is guaranteed as consequence of Lemma 1. The result remains valid even if we do not follow a "bottom-up" order in processing the deforestation eurekas. However requiring that R is not mutually recursive is too strong. When a deforestation eureka contains a symbol f that "calls" g and g "calls" f, it is enough to ensure that no deforestation candidate containing a call of f or g can appear later in the process. This is guaranteed by the **mutual safety condition**.

Definition 2. Let $K : l \rightarrow r$ be a rule. A spine position ω in r is a **dpos** position of the rule K if and only if there exists an accumulative variable $x \in V(r|_\omega)$, at position v in r, such that each position u, $\omega < u < v$ is a spine position.

For example, ϵ is a dpos position of $P : p(x :: xs, y) \rightarrow k(xs, g(x), f(d(y))$, but 2, 3, and 3.1 are not dpos positions of P.

Definition 3. Let $f \in D$ and let $g \in D$ occurring at a spine position in R_f. A symbol k is on **dpath from g to f** if and only if either k occurs in R_g at a dpos position, or k is on the dpath from j to f where j is a symbol that occurs in R_g at a dpos position.

Consider mutually recursive rules $P, M : m(y, x :: xs) = p(xs, y) :: m(y, xs)$, and $F : f(x) \rightarrow a(m(x, x))$, m, p are on the dpath from m to f but not f. Suppose

that g occurs in a best deforestation candidate of a rule of R_f. Symbols in a dpath from g to f, and no others, are likely to occur in a best deforestation candidate later.

Definition 4 (mutual safety condition). A system R is **mutually safe** if and only if for every symbol f occurring in a best deforestation candidate of R, there exists no symbol g occurring in best deforestation candidate of R_f such that f is in the dpath from g to f.

Theorem 5. *Given an orthogonal, constructor-based, terminating, mutually safe and fusable rewrite system R, a procedure using the transition rules terminates returning a terminating system R' equivalent to R which contains no deforestation candidate.*

6 Conclusion

Inside the system Astre, deforestation and tupling strategies are implemented as automatic transformations for orthogonal, fusable, terminating, and mutually safe constructor-based rewrite systems. Termination of the input rewrite system is an obvious requirement for a transformation system based on rewriting. We show in the paper that left-linearity guarantees the correctness of the transformation. The mutually safe property ensures that the process terminates. Fusability guarantees that every deforestation term can be fused. Directly fusable terms are fusable terms that corresponds to the *e-treeless* terms of Chin [7]. At the present time, fusability of other terms relies on a set of laws provided by the user. We are currently exploring ways to use the completion process to synthesize rules that enlarge the class of directly fusable terms. The enlarged class corresponds to the *potentially normalizable* terms of Sheard [16].

A completion procedure is used for controlling the unfold/fold process in each transformation step. It provides a great flexibility for testing a strategy on examples and validating the solutions before implementing them. Moreover, it provides an ideal framework for integrating new tactics and combining diverse strategies. We plan to integrate next the generalization tactic which allows for instance automatic recursion removals.

The superiority of Chin's work is that it is not restricted to first-order programs. Because we are using first-order term rewriting, it seems more difficult to integrate a 'defunctionalization' transformation. We have explored a way to combine partial evaluation with completion in [5].

Acknowledgement We have enjoyed discussions with L. Fegaras. Many thanks to J. Bell and D. Spencer for reading the current draft of the paper.

References

1. L. Bachmair, N. Dershowitz. Commutation, transformation, and termination. *Proc. 8th Int. Conf. on Automated Deduction*, LNCS 230, pages 5-20, 1986.

2. F. Bellegarde, P. Lescanne. Termination by Completion. *Journal of Applied Algebra in Engineering, Communication and Computing*, 1, pages 79-96, 1990.
3. F. Bellegarde. Program Transformation and Rewriting. *Proc. 4th Conf. on Rewriting Techniques and Applications.* LNCS 488, pages 226-239, 1991.
4. F. Bellegarde. Astre, a Transformation System using Completion. Technical Report, Oregon Graduate Institute, 1991.
5. F. Bellegarde. A transformation System Combining Partial Evaluation with Term Rewriting, Presented to HOA'93: An international Workshop on Higher Order Algebra, Logic and Term Rewriting, *Participant proc.*, Amsterdam, Sept. 93.
6. R. M. Burstall and J. Darlington. A Transformation System For Developing Recursive Programs. *Journal of the Association for Computing Machinery*, 24, pages 44-67, 1977.
7. W. N. Chin. *Safe Fusion of Functional Expressions. Proc. of the Conference on Lisp and Functional Programming*, San Francisco, 1992.
8. N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1&2), pages 69-116, 1987.
9. N. Dershowitz. Completion and its Applications. *Resolution of Equations in Algebraic Structures,*, 2, pages 31-86, Academic Press, 1988.
10. M. Fokkinga. Tupling and Mutamorphisms. *The Squiggolist*, vol. 1,4, 1989.
11. A. Gill, J. Launchbury and S.L. Peyton Jones. A short cut to Deforestation. *Proc. of the 6th Conf. on Functional Programming Languages and Computer Architecture*, Copenhagen, pages 223-232, June 1993.
12. J.P. Jouannaud, M. Munoz, Termination of a set of rules modulo a set of equations, *Proc. of the 7th Int. Conference of Automated Deduction.* LNCS 170, pages 175-193, 1984.
13. L. Kott. About a transformation system: a theoretical study. *Proc. of the 3rd Symp. on Programming*, Paris, 1978.
14. U. S. Reddy. Transformational derivation of programs using the Focus system. *Symp. Practical Software Development Environments*, pages 163-172, ACM, December 1988.
15. U. S. Reddy. Rewriting Techniques for Program Synthesis. *Proc. of the 3rd Conf. on Rewriting Techniques and Applications.* LNCS 355, pages 388-403, 1989.
16. T. Sheard and L. Fegaras. A fold for All Seasons. *6th Conf. on Functional Programming Languages and Computer Architecture*, pages 233-242, 1993.
17. P. Wadler, Deforestation: Transforming programs to eliminate trees. *ESOP'88.* LNCS 300, 1988.

7 Appendix

The proof of Theorem 4 Section 3 requires some preliminary lemmas.

Notations on relations The relations on terms $\rightarrow^{-1}$, or $\leftarrow$ denote the *converse* of the relation $\rightarrow$ between two terms. We write $\rightarrow_{R_1} . \rightarrow_{R_2}$ for the *composition* of the two relations $\rightarrow_{R_2}$ and $\rightarrow_{R_1}$. Given two relations $\rightarrow_R$ and $\rightarrow_S$, $\rightarrow_R / \rightarrow_S$ is called R *modulo* S and stands for the relation $\rightarrow_S^* . \rightarrow_R . \rightarrow_S^*$. Note that $\rightarrow_R / \rightarrow_S$ and $\rightarrow_R / \rightarrow_S^*$ are the same. In the proof, we use as lemma the following result from [2].

Lemma 1. *Let S and T be rewrite systems. Suppose S locally cooperates with T, $S \cup T$ is terminating and T is confluent. The relation $(\rightarrow_S /(\rightarrow_T \cup \leftarrow_T))^+$ can be used to prove termination, i.e. a rewrite system that satisfies*

$$l \ \ (\underset{S}{\rightarrow} /(\underset{T}{\rightarrow} \cup \underset{T}{\leftarrow}))^+ \ \ r$$

for all rules $l \rightarrow r$ is terminating.

The local cooperation of a system S with a system T is a kind of local confluence between rules of S and T that can be tested by a criteria on critical pairs between S and T when the system T is variable preserving and left-linear. Therefore, *if there is no overlap between S and T, and T is left-linear and variable preserving, then S* locally cooperates with T.

Lemma 2. *Let E be a deforestation eureka for a constructor-based, and terminating rewrite system R and $T = E^{-1}$ be the* converse of the eureka rule. *If R is left linear, then $R \cup T$ is terminating.*

> **Proof:** First, the system T is terminating. For proof, it is sufficient to take a recursive path ordering where all symbols occurring in the deforestation term (or in the terms for tupling) are less than the eureka symbol h. Second, R quasi-commutes over T. Consider a rewriting by R followed by a rewriting by T.
>
> $$u[\sigma(l)] \rightarrow_R u[\sigma(r)] \rightarrow_T t'$$
>
> Because the right-hand side r does not contain the symbol h, the rewriting by T can only occur either in the context u, then the two rewritings commute, or under a variable in r. This variable occurs in l once because R is left linear, therefore the two rewritings commute again. □

Proof of Theorem 4 Section 3

- There is no overlap between rules of T and T is terminating, therefore T is confluent.
- There is no overlap between R and T and T is variable preserving and left-linear by definition of a deforestation eureka. Therefore R locally cooperates with T.
- $R \cup T$ is terminating by Lemma 2.

then $(\rightarrow_R /(\rightarrow_T \cup \leftarrow_T))^+$ can be used to prove the termination of $R_{fold} \cup R_h$ by Lemma 1. There are two cases to consider:

1. either $l \rightarrow r \in R_{fold}$ then $l \rightarrow_R \leftarrow_T^* r$ by definition of R_{fold}, therefore
$$l \ \ (\underset{R}{\rightarrow} /(\underset{T}{\rightarrow} \cup \underset{T}{\leftarrow}))^+ \ \ r \quad , \textit{or}$$
2. $l \rightarrow r \in R_h$, then
$$l \leftarrow_{T^{-1}} \rightarrow_R \rightarrow_{(R \cup T^{-1})^*} \ r$$
because a rule in R_h is a RCP pair normalized by $R \cup T^{-1}$. Therefore
$$h(tc_1, tc_2, \ldots, tc_n) \ \ (\underset{R}{\rightarrow} /(\underset{T}{\rightarrow} \cup \underset{T}{\leftarrow}))^+ \ \ r$$

□