Termination Issues in Automated Syntheses

F. Bellegarde*
Pacific Software Research Center
Oregon Graduate Institute of Science & Technology

Abstract

Synthesis by completion is used for a fully automated system for program transformation. In the paper, we present a set of transition rules for a full automation of a sound and terminating synthesis process which preserves orthogonality and termination of a constructor-based rewrite system.

Motivation

The transformational approach to the development of programs is attractive for writing small components of large software systems. In this approach, developing a software component consists simply of writing an initial, possibly inefficient, but correct program P_0 and then, in transforming P_0 into a sequence of programs $P_1, P_2, \ldots, P_n$ to get a new, semantically equivalent program P_n which is more efficient. The transformation phase, to be effective, must be fully automated so that it is not necessary to be an expert in transformation strategies to use the transformational approach for software design. In our work, synthesis is used as a mechanism for an automated system for transformation of functional programs: Astre. A prototype of this system is a component of the tool suite that support a Method for Software Design for Reliability and Reuse developed in the Pacific Software Research Center [2]. The underlying methodology is folding-unfolding [9]: Synthesis of a new program version is done by a completion procedure [13, 14, 3]. The programs in the sequence $P_0, P_1, \ldots, P_n$ are presented by a terminating, constructor-based and orthogonal first-order rewrite system. The transformation strategies that are automated are: *fusion or deforestation* (elimination of useless intermediate data structures), and *two-loops fusion or tupling* (consolidation of similar control structures) [8, 9, 22].

Number of issues occur to automate the synthesis process. Mechanisms are needed to control or enhance the completion procedure. For example, a mechanism to control the production of critical pairs during the completion process is required to maintain a reasonable performance. Generation of useless critical pairs is the major drawback for using completion in its application to synthesis [21]. In our case, a careful control prevents divergence of the completion. In this

*The author was supported by a contract with Air Force Materiel Command (F19628-93-C-0069).

paper, we consider the termination issues that arise by using rewriting techniques for this fully automatic transformational approach: Termination of the completion process that performs a synthesis is one of them. Another issue is the preservation of the termination of the rewrite system during the synthesis. Usually, termination of a rewrite system during its completion is ensured by using a reduction ordering to prove termination of the rewrite system and to orient the critical pairs. As noticed by Dershowitz in [14], it is the production of critical pairs, conveniently oriented into constructor-based rules for defining a synthesized symbol which permits a completion process to perform a synthesis. A reduction ordering may not always be appropriate to orient the critical pairs in the way required by the synthesis. Our fully automatic process does not use a reduction ordering for the synthesis. It controls the orientation of the critical pairs into rules as required by the transformation strategy. So doing, there is a potential danger to loose the termination of the rewrite system since preservation of the termination of the source program is not guaranteed in general by a fold-unfold method [19]. However we prove that termination is indeed preserved by our process for a constructor-based orthogonal rewrite system. Furthermore, given a tactical for transformation, it is necessary to guarantee the termination of the sequence of syntheses from the source rewrite system P_0 into the succession of synthesized rewrite systems $P_1, P_2, \ldots$. This problem is not trivial. What we call in the paper a *chain of syntheses* (sequence of syntheses originated from one synthesis) can be related to overlap closures (backward and forward closures) [16, 17], or to a surreduction process [15].

The control mechanism for the production and the orientation of the critical pairs is developped in Section 1. Termination of the synthesis process as well as the preservation of the termination of the rewrite system is proved in Section 2. In Section 3, we give a sufficient condition to prevent a tactical to ask for an infinite number of syntheses.

Preliminaries

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 operator . denotes the concatenation of two positions. The notation $t = C[s]$ emphasizes that the term t contains s as subterm in the context u. A term t is said to be *linear* if no variable occur more than once in t. Variable-free terms are called ground terms. We only consider well-typed terms.

A rewrite rule is an ordered pair of terms, written as $l \to r$, where $V(r) \subseteq V(l)$. A rule $l \to r$ is *left-linear* if l is linear, it is *right-linear* if r is linear. If $V(l) = V(r)$, the rule is variable preserving. A rewrite system is a set of rewrite rules.

The rewriting relation is denoted as $\to$. The relations on terms $\to^{-1}$, or $\leftarrow$ denote the *converse* of the relation $\to$ between two terms. We write $\to_{R_1} . \to_{R_2}$ for the *composition* of the two relations $\to_{R_2}$ and $\to_{R_1}$. The transitive closure of the relation $\to$ is denoted as $\to^+$ and its reflexive and transitive closure denoted as $\to^*$.

The rewrite system R *is terminating* if and only if there is no infinite sequence of terms

$t_1, t_2, \ldots$, such that $t_1 \rightarrow_R t_2 \rightarrow_R \ldots$. Termination of rewrite systems is ensured when R is contained into a well-founded ordering closed under context and substitutions called a reduction ordering. The ordering that are used in practice are simplification orderings that contains the subterm ordering and the embedding we denote by $\trianglelefteq$. Path orderings are simplification orderings built from a well-founded ordering on the function symbols called a precedence.

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 (the two rules are renamed so that they have disjoint set of variables). 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 *othogonal* rewrite system is a left-linear and non-overlapping rewrite system. A rewrite system is *constructor-based*[1] 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. A *constructor term* is a member of $T(C, V)$. 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. R_f is *complete* if every $f(t_1, t_2, \ldots, t_n)$ where t_i are ground constructor terms is R-reducible.

The *inductive positions* of a defined symbol f are the positions of constructors in the left-hand sides of R_f. This way the function *append* defined by:

$$append([], x) \rightarrow x \qquad append(x :: xs, y) \rightarrow x :: append(xs, y)$$

has inductive position 1 and the function fib defined by:

$$\begin{array}{l} fib(0) \rightarrow 0 \\ fib(S(0)) \rightarrow S(0) \end{array} \qquad fib(S(S(x))) \rightarrow fib(x) + fib(S(x))$$

has inductive position 1 and 1.1. Let $t = C[f(t_1, t_2, \ldots, t_n)]$. A variable x located in a constructor term t_i at an inductive position of f is called a *surreductive variable* of t relative to f. For example x is a surreductive variable (relative to *fib*) in the term $fib(s(x))$. A *surreductive term* is a term $f(t_1, t_2, \ldots, t_n)$ with constructor terms at inductive positions, and with at least one surreductive variable. When R_f is complete, there exists a well-typed instance by constructor terms of any surreductive term $f(t_1, t_2, \ldots, t_n)$ which is R_f-reducible. If p is a position of a surreductive variable x relative to f in a term t, and $p.u$ is an inductive position greater or equal than p, then u is a *surreductive position* associated to x. For example 1 is a surreductive position of x in the term $fib(s(x))$.

We suppose the reader familiar with the fold-unfold method [9]. It consists of 6 rules, namely *Definition, Instantiation, Unfolding, Folding, Abstraction*, and *Law*. These rules introduce new identities that are equational consequences of existing identities. Dershowitz [14] 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 a *synthesis*

[1] A constructor-based system of equalities is similar to set of definition equalities with pattern-matching arguments in functional programming.

rule by the user, *Abstraction* is used for a tupling tactic. We implement the tupling strategy by using three additional functional symbols (reserved symbols), *fst, snd*, and *pair* for the theory *Pair*: $fst(pair(x,y)) = x,\ snd(pair(x,y)) = y$.

1 Study of the synthesis

We assume that the first-order functional program is presented by a *constructor-based orthogonal and terminating* rewrite system R. In this section, we describe the synthesis process. *Synthesis rules* introduce a new function whose functional symbol (synthesized symbol) is a fresh functional symbol (a symbol that does not occur in R). We denote them by $h, h_0, h_1, \ldots$.

1.1 Synthesis rules

In the fold-unfold methodology synthesis rules are called *definition rules* or *eurekas rules* because they are introduced through the insight of a clever user. Mechanisms to generate automatically synthesis rules and to foresee inductive positions of the corresponding synthesized symbols for deforestation and two-loops fusion strategies are presently implemented in Astre. These mechanisms are described in [5]. In this paper, we do not address this problem. We simply consider that a set of synthesis rules SR is input to the synthesis process and we assume that the inductive positions of the synthesized symbols are known. Synthesis rules are rewrite rules used for *folding*, therefore, we better have $R \cup SR$ terminating.

Let us guess what a synthesis rule can be. It is a rewrite rule $s \to C[h(t_1, t_2, \ldots, t_n)]$. The synthesis symbol h does not occur in the left-hand side s since the rule introduces h. The context C is here to accommodate the tupling strategy where two synthesis rules $s_1 \to fst(h(t_1, t_2, \ldots, t_n)$ and $s_2 \to snd(h(t_1, t_2, \ldots, t_n)$ are necessary to introduce a synthesized symbol of co-arity 2. The context C is either *fst* or *snd*. For example:

$$sum(x) \to fst(h_1(x)) \tag{1}$$
$$length(x) \to snd(h_1(x)) \tag{2}$$

introduce the synthesis of a function h_1 which computes the pair of the sum and the length of a list. For other strategies, one rule $s \to h(x_1, x_2 \ldots, x_n)$ is enough. For example:

$$length(append(x,y)) \to h_2(x,y). \tag{3}$$

A synthesis rule is always right-linear since we want the synthesis to return left-linear rules for definition of h. The synthesis rule must be variable preserving since we want to orient a rule of the definition of h in the opposite direction. Essentially, the synthesis process computes critical pairs between a rule in R and a synthesis rule, and, later, between a rule of R and such a *synthesis critical pair*. For example a critical pair of the rule 3 with a rule $append([], x) \to x$ gives the critical pair $(length([]), h([]))$ which is turned in a rule $h([]) \to length([])$ of R_h. This computation put constructors at, *and only at*, surreductive positions of the left-hand side of the synthesis critical-pair $C[h(\sigma(x_1), \sigma(x_2), \ldots, \sigma(x_n))]$. This is the key to the termination and soundness of the synthesis process. For example, suppose a synthesis critical pair is

$(append(append(x,y),length(z)),h(x,y,z))$, where 1 is the unique inductive position of h. The critical pair with a rule $length([]) \rightarrow 0$ is not considered for the synthesis.

A more sophisticated synthesis rule is $s \rightarrow h(t_1,t_2,\ldots,t_n)$ where some t_i are terms instead of variables. For example:

$$length(append(x,y)) \rightarrow h_2(x,length(y)) \tag{4}$$

It is introduced to force a top symbol *length* to go down in a term so that it *consumes* a non inductive variable y. These synthesis rules are introduced automatically to force a folding. For example, the rule 4 is introduced to reduce the right-hand side of the rule $h(x :: xs) \rightarrow length(append(x,flatten(xs)))$ so that folding by rule 3 is possible. The result is the rule: $h(x :: xs) \rightarrow h_2(x,h(xs))$. This is automatized very easily and is a good substitute for the inductive law $length(append(x,y)) \rightarrow append(length(x),length(y))$. Notice that the term t_i which is not a constructor term (here $length(x)$) is embedded in s. As we will see later, this embedding ensures *SR-termination*.

A synthesis rule is said to be *collapsing* if it reduces a left-hand side of R. For example the two synthesis rules 1 and 2 are collapsing. For the synthesis process to begin, it is necessary than at least one synthesis critical pair (or a collapse) with R exists. Consider the following example. Assume that 1 and 3 are not inductive position of h. There is no synthesis critical pairs between R and the synthesis critical pair $(append(append(x,y),length(z)),h(x,y,z))$ since the surreductive variables x and z in the left-hand side are not surreductive variables of $h(x,y,z)$. Since a synthesis rule must generate synthesis critical pairs, its left-hand side s must contain a surreductive subterm whose surreductive variables are also surreductive variables of its right-hand side $C[h(t_1,t_2,\ldots,t_n)]$.

There is a difficulty when we have two synthesis rules $s_1 \rightarrow C[h(t_1,t_2,\ldots t_n)]$ and $s_1 \rightarrow C[h(t_1,t_2,\ldots t_n)]$. For clarity, it is always possible to rename the right-hand sides so that they are identicals. Tupling synthesis critical pairs must come by pairs. In other words, when we get one critical pair with one right-hand side, we want to be to get the twin critical pair with the same right-hand side (modulo renaming). For that, it is sufficient that the same substitutions apply for the overlaps between R and the two tupling rules that give synthesis critical-pairs. In other words, it is sufficient that the surreductive positions in s_1 and s_2 of the surreductive variables in $C[h(t_1,t_2,\ldots,t_n)]$ are the same. For example in the rules 1 and 2 the left-hand sides $sum(x)$ and $length(x)$ have the same surreductive positions associated to x.

Also, tupling synthesis rules are often collapsing. In this case they rather not match the same rules of R since, in this case, the process we describe later will not find twin collapsing pairs. For example $f(x,y,z) \rightarrow fst(h(x))$, and $f(x,z,y) \rightarrow snd(h(x))$ where 2 and 3 are not inductive positions of f reduce both the left-hand sides of R_f.

Let us now define a synthesis rule:

Definition 1 *A* **right linear**, **variable preserving** *rule* $(s,C[h(t_1,t_2,\ldots,t_n)])$ *where* h *does not occur in* s *is a synthesis rule for* h *if:*

1. **Orthogonality:** *Let* x *be a variable of* $C[h(t_1,t_2,\ldots,t_n)]$. *All the occurrences of* x *as a surreductive variable of* s *have the same surreductive positions.*

2. **Existence:** *There exists one surreductive subterm in s whose surreductive variables are also surreductive variables of $C[h(t_1, t_2, \ldots, t_n)]$.*

3. **Tupling:** *C is either empty or $C = fst$ (or snd). In the later case, both rules $s_1 \rightarrow fst(h(t_1, t_2, \ldots, t_n))$, and $s_2 \rightarrow snd(h(t_1, t_2, \ldots, t_n)))$ are elements of SR (modulo a renaming). Moreover surreductive positions in s_1 and s_2 of the surreductive variables of $C[(h(t_1, t_2, \ldots, t_n)]$ are the same. Also, s_1 and s_2 must not match the same rules in R.*

4. **SR-termination:** *Every t_i which is not a constructor term is embedded into s,*

5. **Generalization:** *For a non-constructor term t_i, i is a non-inductive position of h.*

These conditions are sufficient to prove the correctness of the synthesis. *Orthogonality* ensures that the synthesis process does not creates overlapping rules. *Generalization* allows us to use a simple generalization to obtain a constructor-based system from critical pairs with R and the pairs where some $t_i, 1 \leq i \leq n$ are not constructor terms. The trick here is that, in this case, the synthesis process does not change the term t_i which is not a variable. In other words, the synthesis can be done without utilizing this term. For example, suppose that the strategy gives 1 as inductive position of h_5, we would not allow a synthesis rule like $reverse(append(x, y)) \rightarrow h_5(reverse(x), y)$. For us, h_5 is not synthesizable since critical pairs with R_{append} substitute x with a constructor term, and then, $reverse(\sigma(x))$ is reduced into a term which is not a constructor term. Then the synthesis process is insufficient, as it is now, to derive a constructor-based definition of h_5.

The synthesis process is driven by the computation the *synthesis critical pairs*. The synthesis process begins to computes critical-pairs between rules of R and non-collapsing synthesis rules $s \rightarrow C[h(t_1, t_2, \ldots, t_n)]$. The goal is to derive constructor-based rules for h from the synthesis critical pairs. As we have said before, synthesis critical pairs are critical pairs between R and a pair $(s, C[h(t_1, t_2, \ldots, t_n)])$ which put constructors at, and only at, surreductive positions of the right-hand side $C[h(t_1, t_2, \ldots, t_n)]$. Another way to tell the same thing is that the non-surreductive variables are not instantiated.

Definition 2 *Let $l \rightarrow r$ be a rule of R and (s, d) a synthesis rule, or, recursively, a synthesis critical pair. A critical pair $(\sigma(s[r]), \sigma(d))$ is a synthesis critical pair if and only if for every non-surreductive variable $x \in V(d)$, $\sigma(x)$ is simply a renaming.*

$SCP(R, E)$ denotes the set of synthesized critical pairs between the rewrite system R and a set E of pairs $(s, C[h(t_1, t_2, \ldots, t_n)])$. For example, let $SR = \{length(append(x, y)) \rightarrow h_2(x, y)\}$ and let R_{append} and R_{length} be the following:

$$\begin{array}{lcllcl} length([]) & \rightarrow & 0 & append([], y) & \rightarrow & y \\ length(x :: xs) & \rightarrow & S(length(xs)) & append(x :: xs, y) & \rightarrow & x :: append(xs, y) \end{array}$$

where S is the successor function. $SCP(R, SR)$ contains two synthesis critical pairs:

$$(length([]), h_2([], y)) \qquad (length(x :: append(xs, y)), h_2(x :: xs, y))$$

Moreover a set of rewrite rules LR (inductive theorems of R) can be input to the synthesis process. These are the *laws* that usually facilitate the process in the fold-unfold method. Diverses mechanisms to introduce automatically laws in the deforestation strategy has been explored by Chin and Bellegarde [10, 4]. The technique of *rippling* found in recent works [7] for automated theorem prover provide an interesting general mechanism to generate laws and synthesis rules. However, we have noticed [5] that most of the situations that requires laws in a deforestation can be simply handled by introducing synthesis rules $s \to C[h(t_1, t_2, \ldots, t_n)]$ where some of the $t_i, 1 \leq i \leq n$ are terms and not variables like in the rule $length(append(x, y)) \to h(x, length(y))$. In this paper, we do not address the problem of automatic generation of the laws. We only consider the impact of laws on termination issues. We can now describe the synthesis process.

1.2 Transition rules for a synthesis

We assume that $R \cup SR \cup LR$ is terminating. In Section 2, we will see that left-linear synthesis rules guarantee that $R \cup SR \cup LR$ is terminating when $R \cup LR$ is terminating.

The synthesis process is expressed by the set of transition rules presented in Figure 1. A transition rule transforms a pair (R, E), where R is an orthogonal rewrite system and where E is a set of synthesis equalities. It uses a set of synthesized rules SR which is preserved during a synthesis. The synthesis procedure is a strategy to apply the transition rules repeatedly to $(R, SCP(R, SR))$ until none is applicable. Consider the role of each transition rules.

- **Deduce** adds to E synthesized critical pairs between R and E.
- **Collapse**-R simplifies the left-hand side of a rule in R and turns it into a pair in E. This pair has the same status than a synthesis critical pair. **Collapse**-R applies in tupling strategy when the left-hand sides of a synthesized pair is of the form $f(x_1, x_2, \ldots, x_n)$ where the x_i at inductive position i is a variable.
- **Unfold-Fold**-E uses synthesis rules and R to simplify a left member of an equality in E.
- **Laws** uses inductive laws to simplify a left-member of an equality in E.
- **Delete** remove any trivial equality.
- **Fold**-R simplifies right-hand sides of rules in R by a synthesis rule.
- **Orient** turns an equality in E into a rule of R when no synthesis critical pair can be computed.
- **Pair** applies when h is synthesized from two pairs $(s, C[h(t_1, t_2, \ldots, t_n)])$ where C is *fst* for one pair and *snd* for the second.

For example, let $SR = \{length(append(x, y)) \to h_2(x, y)\}$. The synthesis process begins with:

$$E = SCP(R, SR) = \{(length([]), h_2([], y)), \quad (length(x :: append(xs, y)), h_2(x :: xs, y))\}$$

$$
\begin{array}{lll}
\textit{Deduce}: & (R, E) & \vdash \quad (R, E \cup \{(p,q)\}) \\
 & & \quad \textit{if } (p,q) \in SCP(R,E) \\
\textit{Collapse-R}: & (R \cup \{(l,r)\}, E) & \vdash \quad (R, E \cup \{(r,l')\}) \\
 & & \quad \textit{if } l \rightarrow_{SR} l' \\
\textit{Unfold/Fold-E:} & ((R, E \cup \{(p,q)\})) & \vdash \quad (R, E \cup \{(p',q)\}) \\
 & & \quad \textit{if } p \rightarrow_{SR \cup R} p' \\
\textit{Laws}: & (R, E \cup \{(p,q)\}) & \vdash \quad (R, E \cup \{(p',q)\}) \\
 & & \quad \textit{if } p \rightarrow_{LR} p' \\
\textit{Delete}: & (R, E \cup \{(p,p)\}) & \vdash \quad (R, E) \\
\textit{Fold-R}: & (R \cup \{(l,r)\}, E) & \vdash \quad (R \cup \{(l,r')\}, E) \\
 & & \quad \textit{if } r \rightarrow_{SR} r' \\
\textit{Pair} & (R \cup \{C_1[h(t_1, t_2, \ldots, t_n)] \rightarrow s_1\}, & \\
 & E \cup \{s_2 \rightarrow C_2[h(t'_1, t'_2, \ldots, t'_n)]\}) & \vdash \quad (R \cup \{h(\sigma(t_1), \sigma(t_2), \ldots, \sigma(t_n)) \\
 & & \quad \rightarrow pair(\sigma(s_1), \sigma(s_2))\}, E)
\end{array}
$$

if $SCP(R, \{(s2, C_2[h(t'_1, t'_2, \ldots, t'_n)]) = \emptyset$, s_2 *is* $(R \cup SR)$*-irreducible, where* $C_1 = fst$ *and* $C_2 =$ *snd or the converse, and* σ *is the more general unifier of the* $t_i, 1 \leq i \leq n$.

$$
\textit{Orient}: \qquad (R, E \cup \{(p,q)\}) \qquad \vdash \quad (R \cup \{(q',p')\}, E)
$$

if $SCP(R, \{(p,q)\}) = \emptyset$, p is $R \cup SR$*-irreducible, and where* $(q', p') = (\mu(p), \mu(q))$, q' *is the least generalization which substitutes every* $t_i \notin T(C,V), 1 \leq i \leq n$ *in* $q = C[h(t_1, t_2, \ldots, t_n)]$ *by fresh distincts variables*

Figure 1: Synthesis transition rules

By **Unfold/Fold**-E three times, we get:

$$
E = \{(0, h_2([], y)), \quad (S(h_2(xs, y)), h_2(x :: xs, y))\}
$$

By **Orient** two times we get:

$$
R_{h_2} = \{h_2([], y) \rightarrow 0, \quad h_2(x :: xs, y) \rightarrow S(h_2(xs, y))\}
$$

Another example with two collapsing rules:

$$
SR: \begin{array}{lcl} sum(x) & \rightarrow & fst(h(x)) \\ prod(x) & \rightarrow & snd(h(x)) \end{array}
$$

$$
R_{sum}: \begin{array}{lcl} sum([]) & \rightarrow & 0 \\ sum(x :: xs) & \rightarrow & x + sum(xs) \end{array} \qquad R_{prod}: \begin{array}{lcl} prod([]) & \rightarrow & 0 \\ prod(x :: xs) & \rightarrow & x * prod(xs) \end{array}
$$

$$
foo(x) \rightarrow sum(x) + prod(x)
$$

Apply *Collapse-R* four times:

$$E: \begin{array}{lcl} 0 & \to & fst(h([])) \\ x + sum(xs) & \to & fst(h(x :: xs)) \\ 0 & \to & snd(h([])) \\ x * prod(xs) & \to & snd(h(x :: xs)) \end{array} \qquad R : foo(x) \to sum(x) + prod(x)$$

Apply *Fold-R* two times, and *Fold-E* two times:

$$E: \begin{array}{lcl} 0 & \to & fst(h([])) \\ x + fst(h(xs)) & \to & fst(h(x :: xs)) \\ 0 & \to & snd(h([])) \\ x * snd(h(xs)) & \to & snd(h(x :: xs)) \end{array} \qquad R : foo(x) \to fst(h(x)) + snd(h(x))$$

Apply *Orient* two times and *Pair* two times:

$$\begin{array}{l} h([]) \to pair(0, 0) \\ h(x :: xs) \to pair(x + fst(h(xs)), x * snd(h(xs))) \\ foo(x) \to fst(h(x)) + snd(h(x)) \end{array}$$

When no more transition rules applies, the system R is a constructor-based orthogonal rewrite system. Conditions we impose to the definition of the synthesis rules are sufficient to get this result. Moreover, the transition rules are sound with the following meaning: Assuming that $f(t_1, t_2, \ldots, t_n) \to_R^* s$ where $f \in D$, $t_i, 1 \leq i \leq n$ and s are ground constructor terms, then $f(t_1, t_2, \ldots, t_n) \to_{R' \cup CSR}^* s$ where R' is the rewrite system result of the synthesis and CSR is the set of the collapsing synthesis rules. Sketch of a proof can be found in appendix. The result of interest for the present paper is the termination of the synthesis process.

2 Termination of the synthesis

During a synthesis process, the rewrite system $R \cup SR \cup LR$ must be terminating so that reductions terminate. We do not use reduction orderings to prove termination of rewrite systems during the synthesis. We always assume termination of $R \cup LR$ but we can get for free termination of $R \cup LR \cup SR$ when the synthesis rules are left-linear.

2.1 Study of the termination of $R \cup LR \cup SR$

Consider the termination of SR. It contains synthesis rules as defined in Section 1. To prove its termination, it is sufficient to take a recursive path ordering where all the functional symbols that occurs left-hand sides s of rules in SR have a greater precedence than the synthesized symbols h. Let us prove now that SR quasi-commute over $R \cup LR$ when SR is left-linear.

Theorem 1 *Let $R \cup LR$ be terminating and SR be left-linear, $R \cup LR \cup SR$ is terminating.*

Proof: Let $s \to C[h(t_1, t_2, \ldots, t_n)]$ be a rule in SR and $l \to r$ be a rule in $R \cup LR$. Assume a rewriting by $l \to r$ follows a rewriting by $s \to C[h(t_1, t_2, \ldots, t_n)]$, we have

$$t = u[\sigma(s)] \to_{SR} u[C[h(\sigma(t_1), \sigma(t_2), \ldots, \sigma(t_n))]] \to_R t'$$

The rewriting by $l \to r$ cannot occur at the position of h in d. Therefore it occurs either in the context u or it rewrites one of the subterms $\sigma(t_i)$. In the first case, t' can obviously be obtained by rewriting first by $l \to r$ and then by $s \to C[h(t_1, t_2, \ldots, t_n)]$. In the second case, the same can be done because t_i occurs only once in the linear term s of a variable preserving synthesis rule. Since $R \cup LR$ and SR are terminating and SR quasi-commutes over $R \cup LR$, $R \cup LR \cup SR$ is terminating [1, 18]. □

Hence, when SR is left-linear and $R \cup LR$ is terminating, we are certain, without using any reduction ordering that all the reductions terminates during the synthesis process. The synthesis rules are left-linear for deforestation, tupling, and recursion removal strategies. However some syntheses cannot be introduced by left-linear synthesis rules. For example, one might want a synthesis rule like $x + x \to h(x)$. In this case, we do not get for free the termination of $R \cup LR \cup SR$ from the termination of $R \cup LR$. It needs to be proved otherwise. Now, we assume that $R \cup SR \cup LR$ terminates and we prove that the synthesis process terminates.

2.2 Termination of the synthesis process

To prove that repeated applications of the transition rules always terminate, consider the triplet (I, E, R) where I is the finite set of all possible substitutions by constructor terms at inductive positions of the synthesized symbols. The triplets are ordered by lexicographic extension of well-founded orderings on the sets I, E, and R. I is ordered by its size. The sets E and R are ordered by a multiset extension ordering on a well-founded ordering on their ordered pairs of terms. We order the pairs as follows: $(p, q) > (p', q')$ if $p > p'$ or $p = p'$ and $q > q'$ where the term order is the rewrite order defined by the terminating rewrite system $R \cup SR \cup LR$. We show that each transition rule decreases lexicographically the triplet (I, E, R).

- **Deduce** decreases the size of I by pushing one constructor term in an inductive position.
- **Collapse**-R decreases the size of I by pushing one constructor term in an inductive position.
- **Unfold/Fold**-E and **Laws** decreases E by rewriting a left-hand side of a pair.
- **Delete** and **Orient** decreases E by removing one of its pairs.
- **Fold-R** decreases R by rewriting a right-hand side of a pair.
- **Pair** and **Orient** decreases E.

It remains to prove that the synthesis process preserves termination of R.

2.3 Preservation of the termination of R

To prove the preservation of the termination of R, we need more notations on relations.

Notations on relations: 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 [6].

Lemma 1 (See [6]) *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 $l \ (\rightarrow_S /(\rightarrow_T \cup \leftarrow_T))^+ \ r$ 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. The proof uses also the following lemma.

Lemma 2 *Let SR be synthesis rules for a left-linear, constructor-based, and terminating rewrite system R at the begining of the synthesis and let $T = SR^{-1}$ be the* converse of synthesis rules. *Assume that R is terminating, then $R \cup T$ is terminating.*

> **Proof:** We prove that a $(R \cup T)$-reducible term t cannot be the beginning of an infinite derivation. Let us use a mathematical induction on the number of occurrences of h in t. Assume t has no occurrences of h, we can only have $t \rightarrow_R t'$ and t' has no occurrence of h since the left-hand sides of rules in R does not contain h. Therefore every derivation from t is a R-derivation and hence terminating. Consider now a term t which contains n occurrence of h. Consider the first occurrence of a rewriting by T in the derivation, $t \rightarrow_R^* . \rightarrow_T t'$. the term t' contains $n-1$ occurrences of h since the left-hand sides of the synthesis rules does not contain h. No infinite derivation can come from t' and hence from t. □

Let us prove now that the synthesis process preserves the termination of R.

> **Proof:** Let us call R_{fold} the rewrite system R after applications of the transition rule **Fold**-R. The result of the synthesis is $R_{fold} \cup R_h \cup CSR$ where CSR are the collapsing synthesis rules. Let us prove that $R_{fold} \cup R_h$ is terminating.
>
> - 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 since synthesis rules are variable preserving and right-linear. Therefore R locally cooperates with T.
> - $R \cup T$ is terminating by Lemma 2.

Consequently $(\rightarrow_R /(\rightarrow_T \cup \leftarrow_T))^+$ 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

$$l \ (\rightarrow_R /(\rightarrow_T \cup \leftarrow_T))^+ \ r \quad , \textit{or}$$

2. $l \rightarrow r \in R_h$. Then it comes from a pair (r, l) in E. A pair in E comes from a synthesis critical pair between R and $SR = T^{-1}$, or from a collapse of R by a synthesis rule. In both cases we have $l \rightarrow_T \rightarrow_R r$. Subsequent pairs in E are obtained by rewriting of r by SR so $l \rightarrow_T \rightarrow_R \leftarrow_T^* r$, or by rewriting by R_{fold} so $l \rightarrow_T \rightarrow_R .(\leftarrow_T \cup (\rightarrow_R \leftarrow_T^*))^* r$, or by critical pair with R_{fold} which does not change the relation. Consequently we always have: $l \ \rightarrow_R /(\rightarrow_T \cup \leftarrow_T))^+ \ r$

This achieves the proof that $R_{fold} \cup R_h$ is terminating. We now prove that $R_{fold} \cup R_h \cup CSR$ is terminating. To prove termination, we use a result of Bachmair and Dershowitz [1]. They have proven that if R_1/R_2 and $R2$ terminates, then $R1 \cup R2$ terminates. In the first part of the proof we show that $(R_{fold} \cup R_h) \subseteq R/(T \cup T^{-1})$ gives the termination of $R_{fold} \cup R_h$ but it gives also termination of $(R_{fold} \cup R_h)/(T \cup T^{-1})$. Therefore $R_{fold} \cup R_h \cup CSR$ terminates since $CSR \subseteq T^{-1} \subseteq (T \cup T^{-1})$, and CSR terminates. □

Unfortunately termination of a sole synthesis is not enough. For a given tactical, the system automatically generates synthesis rules and run the synthesis process since no more synthesis rules can be generated. Let us study the termination of the tactical process.

3 Tactical termination

The tactical process generates a sequence of set of synthesis rules $SR_0, SR_1, \ldots$. To simplify, we consider that, each time, only one synthesis symbol is synthesized. Let us then consider the sequence of the left-hand sides (or of the pair of the twin left-hand sides for a tupling) $s_0, s_1, \ldots, s_i, \ldots$ of the synthesis rules for the synthesis of symbols $h_0, h_1, \ldots, h_i, \ldots$. Since there are only a finite number of synthesis rules that can be generated from a finite set of rules, the tactical termination problem is the same that the termination of the sequences $s_0, \ldots, s_i, s_{i+1} \ldots$ where s_{i+1} is generated from R_{h_i}. We call such a sequence a *chain of syntheses*. The tactical process generates a new s_{i+1} from one of the right-hand sides r_i of R_{h_i}. We call S the relation between s_i and r_i and G the relation between r_i and s_{i+1}. More precisely, $s_i \rightarrow_S r_i$ if and only if there exists a substitution σ of surreductive variables of h_i by constructor terms such that $\sigma(s_i) \rightarrow^+_{R \cup LR \cup SR} r_i$. It is a special case of narrowing derivation [15]. The relation G is defined by the tactical but we can assume that it is always contained into the inverse of the encompassment ordering. In other words, if $r_i \rightarrow_G s_{i+1}$, there exists a substitution η, and a position p such that $\eta(s_{i+1}) = r_i|_p$. A chain of syntheses C is simply a sequence $s_0 \rightarrow_S . \rightarrow_G s_1 \rightarrow_S \rightarrow_G s_2 \ldots$. The tactical termination is thus $\rightarrow_S . \rightarrow_G$ termination.

In the following, we assume that we rename all the variables into a unique dummy constant □. We also assume that a synthesis symbol $h_j, 0 \leq j \leq i$ does not occur in a term s_{i+1} in

the chain of syntheses. Therefore the terms s_i are ground terms constructed on a finite set of symbols. In this case, every infinite sequence of ground terms is self-embedding by Kruskall's tree theorem.

Suppose that we have a relation $\preceq$ on terms such that if $s_i \preceq r$ where r is one of the right-hand sides of the rules in R_{h_i}. Assume that $s_i \prec r$ implies that s_i cannot be embedded into a term that comes afterwards in C. In other words, $s_i \preceq r$ foresee a future self-embedding in the sequence C. Then the tactical process can follow a backtracking strategy to guarantee its termination by application of Kruskall's tree theorem. The definition of $\preceq$ uses the relation Φ on terms defined by $f \Phi g$ if and only if g occurs in one of the left-hand sides of R_f.

Definition 3 *We have $s = f(s_1, \ldots, s_m) \preceq g(t_1, t_2, \ldots, t_n) = t$ if and only if either of the following cases hold:*

1. *$g \Phi f$ and $s_i \preceq t, 1 \leq i \leq m$; or*
2. *$f = g$ and $s_i \preceq t_i, 1 \leq i \leq m$; or*
3. *$f(s_1, \ldots, s_m) \preceq t_i$ for some $i, 1 \leq i \leq n$.*

We now propose the following:

Proposition 1 *Let $s_i \rightarrow_S r \rightarrow_G .(\rightarrow_S . \rightarrow_G)^*)s_j$. If $s_i \trianglelefteq s_j$ then $s_i \preceq r$.*

> **Sketch of proof:** We prove that if $s = f(u_1, \ldots, u_n) \trianglelefteq t$ and $r \rightarrow_G .(\rightarrow_S . \rightarrow_G)^* t$ then $s \preceq r$ by induction on terms structure. Take the smallest subterm $t' = f(v_1, \ldots, v_n)$ of t such that $s \trianglelefteq t$. Take the smallest subterm $r' = g(w_1, \ldots, w_m)$ of r such that $r' \rightarrow_G .(\rightarrow_S . \rightarrow_G)^* t'$. Then, either $f = g$ or $g \Phi f$. Proof of $s \preceq r'$ is sufficient for proof of $s \preceq r$.
>
> - $g = f$: We have $u_i \trianglelefteq v_i, 1 \leq i \leq n$ by definition of $\trianglelefteq$. Since $r' \rightarrow_G .(\rightarrow_S . \rightarrow_G)^* t'$ and $r' = f(w_1, \ldots, w_n)$, $w_i \rightarrow_G .(\rightarrow_S . \rightarrow_G)^* v_i, 1 \leq i \leq n$. Then we have $u_i \preceq v_i, 1 \leq i \leq n$ by induction, and $s \preceq r'$ by definition of $\preceq$.
> - $g \Phi f$: We have $u_i \trianglelefteq v_i, 1 \leq i \leq n$ by definition of $\trianglelefteq$. Since $r' \rightarrow_G (\rightarrow_S . \rightarrow_G)^* t'$, we also have $r' \rightarrow_G (\rightarrow_S . \rightarrow_G)^* v_i, 1 \leq i \leq n$ by changing the last application of $\rightarrow_G$. Then we have $u_i \preceq r'$ by induction, and $s \preceq r'$ by definition of $\preceq$.
>
> □

4 Related works and Conclusion

Fully automatized algorithms for deforestation based on a fold-unfold method are those of Wadler [22] and more recently of Chin [10, 11]. Though this paper does not consider a particular tactical or the way we generate synthesis rules, let us observe the deforestation tactical. Its goal is to eliminate useless data-structures. In the left-hand side of the synthesis rule, there are functional symbols *consumers* of a data structure that is produced by a functional symbol *producer* of the data-structure that could be eliminated. Chin and Wadler impose restrictive

conditions on the synthesis rules since they do not have the powerful synthesis rules with non-constructor terms arguments of the synthesis symbol to achieve fusion with a functional symbol f that does not *produce* a constructor as top symbol of the right-hand sides of the rules in R_h. Also, their algorithms do not integrate easily the use of laws. Only to illustrate the difficulty of the termination problem, notice that the termination proof of Chin's deforestation algorithm takes four pages in [11]. Chin's and Wadler's algorithms are specialized towards the deforestation tactical so that it does not seem to be flexible enough to include other tacticals. The Focus system [20] is a general system for synthesis based on rewriting techniques. However, it is not fully automatic. We do not know a fully automatic process which supports other tacticals than deforestation. Chin and Darlington have done a remarkable work to take account of higher-order functional programs in [12]. An implementation of the Chin-Darlington's specialization algorithm is used as a preprocessor of ASTRE for input of functional programs. The actual limitation of our technology is the termination property of the rewrite system.

In the paper, we propose a set of transition rules that describe a fully automatic synthesis procedure for the transformation of functional programs that are presented by terminating orthogonal constructor-based rewrite systems. We proved that the synthesis process is sound and that it preserves the property of the rewite system and particularly its termination. We proved that the synthesis processis terminating and we know how to ensure termination of a chain of syntheses. The synthesis rule conditions are satisfied for automatization of deforestation and two-loops tupling. The set of transition rules can be completed for handling other tacticals. In particular adding a transition rule that computes critical pairs between the synthesis rules and a set of laws does not change the results presented in this paper.

References

[1] L. Bachmair and N. Dershowitz. Commutation, Transformation, and Termination. In *Proceedings of the eight conference on Automated Deduction*, volume 230 of *LNCS*, pages 5–20. Springer-Verlag, 1986.

[2] J. Bell et al. Software Design for Reliability and Reuse: A proof-of-concept demonstration. Technical report, Department of Computer Science and Engineering, Oregon Graduate Institute, 1994. To be presented at the Tri-Ada conference in November.

[3] F. Bellegarde. Program Transformation and Rewriting. In *Proceedings of the fourth conference on Rewriting Techniques and Applications*, volume 488 of *LNCS*, pages 226–239. Springer-Verlag, 1991.

[4] F. Bellegarde. A transformation system combining partial evaluation with term rewriting. In *Higher Order Algebra, Logic and Term Rewriting (HOA '93)*, volume 816 of *LNCS*, pages 40–58. Springer-Verlag, September 1993.

[5] F. Bellegarde. Automatic Synthesis by Completion. Technical Report 94023, Department of Computer Science and Engineering, Oregon Graduate Institute, 1994.

[6] F. Bellegarde and P. Lescanne. Termination by Completion. *Journal of Applied Algebra in Engineering, Communication and Computing*, 1:79–96, 1990.

[7] A. Bundy et al. Rippling: A heuristic for guiding inductive proofs. *Journal of Artificial Intelligence*, 62:185–253, 1993.

[8] W.H. Burge. *Recursive Programming Techniques.* Addisson-Wesley, 1975.

[9] R. M. Burstall and J. Darlington. A Transformation System for Developing Recursive Programs. *Journal of the ACM*, 24:44–67, 1977.

[10] W. Chin. Safe Fusion of Functional Expressions. *Proceedings of the ACM Symposium on Lisp and Functional Programming, San Francisco, Ca.*, pages 11–20, June 1992.

[11] W. Chin. Safe Fusion of Functional Expressions II: Further Improvements. *Journal of Functional Programming*, 11:1–40, 1994.

[12] W. Chin and J. Darlington. Higher-Order Removal: A modular approach. Unpublished work, 1993.

[13] N. Dershowitz. Synthesis by Completion. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 208–214, Los Angeles, 1985.

[14] N. Dershowitz. Completion and its Applications. In *Resolution of Equations in Algebraic Structures.* Academic Press, New York, 1988.

[15] M. Fay. First-order unification in an equational theory. In *Proceedings of the fourth Workshop on Automated Deduction*, pages 161–167, 1979.

[16] J. V. Guttag, D. Kapur, and D. R. Musser. On proving uniform termination and restricted termination of rewrite systems. *SIAM Journal on Computing*, 12(1):189–214, February 1983.

[17] M. Hermann. *Divergence des systèmes de réécriture et schématisation des ensembles infinis de termes.* PhD thesis, Université de Nancy I, March 1994.

[18] M. Munoz J.P. Jouannaud. Termination of a set of rules modulo a set of equations. In *Proceedings of the 7th Conference of Automated Deduction*, volume 170 of *LNCS*, pages 175–193. Springer-Verlag, 1984.

[19] L. Kott. About a transformation system: a theoretical study. In *Proceedings of the 3rd Symp. on Programming*, 1978.

[20] U. S. Reddy. Transformational derivation of programs using the Focus system. In *ACM Symposium on Practical Software Development Environments*, pages 163–172, December 1988.

[21] U. S. Reddy. Rewriting Techniques for Program Synthesis. In *Proceedings of the third Conference on Rewriting Techniques and Applications*, volume 355 of *LNCS*, pages 388–403. Springer-Verlag, 1989.

[22] P. Wadler. Deforestation: Transforming Programs to eliminate trees. In *Proceedings of the second European Symposium on Programming ESOP'88*, volume 300 of *LNCS*. Springer-Verlag, 1988.

A Soundness of the transition rules

We prove that the synthesis process, as described in Section 1, preserves successful computations of ground terms. Let (R', E') be the result of the synthesis process beginning with $(R, SCP(R, SR))$ and let CSR be the collapsing synthesis rules. Consider a derivation $f(t_1, t_2, \ldots, t_n) \rightarrow_R^* c$ where $f \in D$, $t_i, 1 \leq i \leq n$ and c are ground constructor terms. We prove that $f(t_1, t_2, \ldots, t_n) \rightarrow_{R' \cup CSR}^* c$.

Proof: Consider a derivation D by R towards a ground constructor term c. Let us reflect the changes made to this derivation during the synthesis process and let us show that they can be turned into a R' derivation. We call R the evolving rewrite system during the synthesis process. Changes to R comes either from folding a rule by a synthesis rule or from collapsing a rule by a synthesis rule. In the first case $u \rightarrow_R v$ is replaced by $u \rightarrow_R u' \leftarrow_{SR} v$ and in the second case $u \rightarrow_R v$ is replaced by $u \rightarrow_{CSR} u' \leftarrow_E v$. In both cases $u' = K[C[h(s_1, s_2, \ldots, s_n)]]$ where s_i are grounds terms. Since c is a ground constructor term, all the terms s_i at inductive positions of h can be reduced to ground constructor terms by a derivation which commutes over these derivation steps. So, consider a derivation step: $u' = K[C[h(\sigma(t_1), \sigma(t_1), \sigma(t_2), \ldots, \sigma(t_n))]] \leftarrow_{SR} K[C[\sigma(s)]] = v$ where $\sigma(t_i)$ at inductive position i of h is a ground constructor term. By *Existence* condition in Definition 1, there exist synthesis critical pairs issued from overlaps between the synthesis rule and some R_g. Since g disappear, one of these synthesis critical pairs in E : concerns our derivation step. It gives: $u' \rightarrow_R u'' = K[C[h(\sigma'(t'_1), \sigma'(t'_1), \sigma'(t'_2), \ldots, \sigma'(t'_n))]] \leftarrow_E K[C[\sigma'(s') = v]]$. In other words, every $\leftarrow_{SR}$ step becomes a $\leftarrow_E$ step.

Consider now an element $u'' \leftarrow_E v$. It can be folded or unfolded by a rewriting by SR: $u'' \leftarrow_E v' \leftarrow_{SR} v$ or by a rewriting by R: $u'' \leftarrow_E v' \leftarrow_R v$. It remains two cases: $\leftarrow_E$ is turned into $\rightarrow_R$ by *Orient* or a synthesis critical pair is derived changing $\leftarrow_E$ into $\leftarrow_R \leftarrow_E$. Since there are a finite number of such pairs, every step $\leftarrow_E$ is finally oriented into $\rightarrow_R$. *Peaks* $\leftarrow_R \rightarrow_R$ can all be changed into *valleys* $\rightarrow_R^* . \leftarrow_R^*$ by confluence of R yielding a derivation of R towards irreducible c. □