On Unfolding Completeness for Rewriting Logic Theories Technical Report

María Alpuente
DSIC, Universidad Politécnica de Valencia
Camino de Vera s/n, Apdo. 22012,
46071 Valencia, Spain.
Email: alpuente@dsic.upv.es

Michele Baggi, Moreno Falaschi
Dip. di Scienze Matematiche e Informatiche
Pian dei Mantellini 44,
53100 Siena, Italy.
Email: {baggi,moreno.falaschi}@unisi.it

Demis Ballis
Dip. Matematica e Informatica
Via delle Scienze 206,
33100 Udine, Italy.
Email: demis@dimi.uniud.it

***Abstract*—Many transformation systems for program optimization, program synthesis, and program specialization are based on fold/unfold transformations. In this paper, we investigate the semantic properties of a narrowing-based *unfolding* transformation that is useful to transform rewriting logic theories. We also present a transformation methodology that is able to determine whether an unfolding transformation step would cause incompleteness and avoid this problem by completing the transformed rewrite theory with suitable extra rules. More precisely, our methodology identifies the sources of incompleteness and derives a set of rules that are added to the transformed rewrite theory in order to preserve the semantics of the original theory.**

I. Introduction

Program transformation is a method for deriving correct and efficient programs. The folding and unfolding transformations, which were first introduced by Burstall and Darlington [10] for functional programs, are the most basic and powerful techniques for a program transformation framework. Unfolding was introduced in logic programming by Komorowski [22]. The combined effect of unification with rewriting by means of narrowing was first proposed in [2] and was also achieved in [13], [14], [29] by means of a superposition procedure for program synthesis. Unlike the case of pure logic or pure functional programs, where unfolding is correct w.r.t. practically all available semantics, unrestricted unfolding using narrowing does not preserve program meaning, even when we consider the normalization semantics (i.e., the set of normal forms) or the evaluation semantics (i.e., the set of values) of the program. In [2], some conditions were ascertained which guarantee that an equivalent program w.r.t. the semantics of computed answers is obtained for functional logic programs.

Unfolding is essentially the replacement of a call by its body, with appropriate substitutions. Folding is the inverse transformation, i.e., the replacement of some piece of code by an equivalent function call. For functional programs, folding and unfolding steps involve only pattern matching. The fold/unfold transformation approach was first adapted to logic programs by Tamaki and Sato [32] by replacing pattern matching with unification in the transformation rules. A lot of literature has been devoted to proving the correctness of fold/unfold systems w.r.t. the various semantics proposed for functional programs [10], [23], logic programs [21], [28], [31], [32], functional logic programs [3], and constraint logic programs [16].

Quite often, however, transformations may have to be carried out in contexts in which the function symbols satisfy certain *equational axioms*. For example, in rule-based languages such as ASF+SDF [7], Elan [8], OBJ [19], CafeOBJ [15], and Maude [12], some function symbols may be declared to obey given algebraic laws (the so-called *equational attributes* of OBJ, CafeOBJ and Maude), whose effect is to compute with equivalence classes modulo such axioms while avoiding the risk of non-termination. Similarly, theorem provers, both general first-order logic ones and inductive theorem provers, routinely support commonly occurring equational theories (e.g. associative-commutative theories) for function symbols. Moreover, several of the afore-mentioned languages and provers have an expressive *order-sorted* typed setting with sorts and subsorts (where subsort inclusions form a partial order and are interpreted semantically as set-theoretic inclusions of the corresponding data sets). The unfolding transformation has been scarcely studied so far for rewriting logic theories that may include sorts, rules, equational theories, and algebraic laws (such as commutativity and associativity). Apart from our preliminary work [1], where we developed a narrowing-based, fold/unfold transformation framework for optimizing rewriting logic theories, we are not aware of any other fold/unfold transformation technique which can deal with such advanced rewriting logic features.

Our contribution. In this paper, we formalize a powerful narrowing-based unfolding transformation for rewriting logic theories that preserves the rewriting logic semantics of the original theory. Our technique relies on the fact that rewriting logic also supports the narrowing mechanism [11] that successfully combines term rewriting and unification [17] and is efficiently implemented in the functional programming language Maude [12]. Roughly speaking, unfolding is defined by applying narrowing steps to the right-hand sides of both rules and equations of the rewrite theory under examination in order to obtain the unfolded theory. Narrowing allows us to empower the unfold operation by implicitly embedding the instantiation rule (the operation of the Burstall and Darlington framework [10] that introduces an instance of an existing rule) into unfolding by means of unification.

This work greatly improves the unfolding operator described in [1], by relaxing several strong syntactic restrictions which we had to enforce to guarantee the completeness of the unfolding transformation. However, we do not consider in this paper other transformation rules like folding or definition introduction/elimination, which we did study in [1].

A related but different unfolding technique for transforming (canonical) conditional term-rewriting systems (TRS), is proposed in [2], where the main goal is to preserve the semantics of (narrowing) computed answers. Here, a completeness result is proved for left-linear and L-closed programs, where the closedness notion compares all calls in the right-hand side of the program rules w.r.t. the left-hand side of the rules similarly to the closedness notion used in Partial Evaluation [4], [5]. Then, a generalized notion of unfolding is provided, which, in the case of unconditional programs, keeps the original rule in the transformed program. With this generalized unfolding operation, completeness holds under less demanding conditions. In this work, we consider possibly non-confluent and non-terminating rewriting logic theories, and we study the unfolding operation w.r.t. the standard rewriting logic semantics of ground normal forms. Thus, in our setting, no notion similar to the L-closedness is needed.

However, there are pathological situations where unfolding may cause incompleteness. Hence, we develop a transformation methodology that is able to determine whether an unfolding operation would cause incompleteness and overcome this problem by deriving a set of new rules that are added to the transformed program in order to preserve the semantics of the original program.

The paper is organized as follows. In Section II, we recall some essential notions about rewriting logic, and, in Section III, we recall the notion of narrowing for rewriting logic theories. Section IV formalizes the unfolding operation for order-sorted rewrite theories and identifies the causes of incompleteness. Then, a methodology is proposed that is able to recover the semantics of the original program. In Section V, we demonstrate the correctness and completeness of the unfolding operation w.r.t. the considered semantics, and we conclude in Section VI. Proofs of all results are given in Appendix II.

II. Preliminaries

We consider an *order-sorted signature* Σ, with a finite poset of sorts $(S, \leq)$. We assume an S-sorted family $\mathcal{X} = \{\mathcal{X}_s\}_{s \in S}$ of disjoint variable sets. $\mathcal{T}_\Sigma(\mathcal{X})_s$ and $\mathcal{T}_{\Sigma s}$ are the sets of terms and ground terms of sort s, respectively. We write $\mathcal{T}_\Sigma(\mathcal{X})$ and $\mathcal{T}_\Sigma$ for the corresponding term algebras. The set of variables that occur in a term t is denoted by $\mathcal{V}ar(t)$. We write $\overline{o_n}$ for the *list* of syntactic objects $o_1, \ldots, o_n$.

A *position* p in a term t is represented by a sequence of natural numbers. Λ denotes the empty sequence, and by $root(t)$ we denote the symbol of t that is rooted at position Λ. Positions are ordered by the *prefix* ordering: $p \leq q$, if $\exists w$ such that $p.w = q$. Two positions q and p are not comparable if $q \not\leq p$ and $p \not\leq q$. Given a term t, let $\mathcal{P}os(t)$ and $\mathcal{NVP}os(t)$, respectively, denote the set of positions and the set of non-variable positions of t (i.e., positions where a variable does not occur). $t|_p$ denotes the *subterm* of t at position p, and $t[s]_p$ denotes the result of *replacing the subterm* $t|_p$ in t by the term s.

A substitution σ is a mapping from variables to terms $\{x_1/t_1, \ldots, x_n/t_n\}$ such that $x_i\sigma = t_i$ for $i = 1, \ldots, n$ (with $x_i \neq x_j$ if $i \neq j$), and $x\sigma = x$ for all other variables x. The identity substitution is denoted by id. A substitution σ is called *ground* if for each $x/t \in \sigma$, t is a ground term.

An *(order-sorted) equational theory* is a pair $E = (\Sigma, \Delta \cup B)$, where Σ is an order-sorted signature, Δ is a collection of equations ($l = r$, with $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$), and B is a collection of equational axioms that express associativity (A) and/or commutativity (C) for some defined symbols of Σ. We assume Σ is a partition $\Sigma = \mathcal{C} \uplus \mathcal{D}$ of symbols $c \in \mathcal{C}$, called *constructors*, and symbols $f \in \mathcal{D}$, called *defined symbols*, each of which has a fixed arity, where $\mathcal{D} = \{f \mid f(\bar{t}) = r \in \Delta\}$ and $\mathcal{C} = \Sigma - \mathcal{D}$.

The equations in an equational theory E are considered as simplification rules by using them only in the left to right direction; therefore, for any term t, by repeatedly applying the equations as simplification rules, we eventually reach a term to which no further equations apply. The result is called the *canonical form* of t w.r.t. E. This is guaranteed by the fact that E is required to be terminating and Church-Rosser [9]. The set of equations in Δ together with the equational axioms of B in an equational theory E induce a congruence relation on the set of terms $\mathcal{T}_\Sigma(\mathcal{X})$, which is usually denoted by $=_E$. E is a presentation or axiomatization of $=_E$. In abuse of notation, we speak of the equational theory E to denote the theory axiomatized by E. Given an equational theory E, we say that a substitution σ is an *E-unifier* of two terms t and t' if $t\sigma$ and $t'\sigma$ are both reduced to the same canonical form modulo the equational theory (in symbols $t\sigma =_E t'\sigma$). For substitutions σ, ρ and a set of variables V, we define $\sigma =_E \rho$ if $x\sigma =_E x\rho$ for all $x \in V$, and we define $\sigma \ll_E \rho$ if there is a substitution η such that $\rho =_E (\eta \circ \sigma)$. Given two terms $t, t' \in \mathcal{T}_\Sigma(\mathcal{X})$, a set of substitutions $CSU_E(t, t')$ is said to be a *complete* set of unifiers w.r.t. t and t' if (i) each $\sigma \in CSU_E(t, t')$ is an E-unifier of t and t', and (ii) for any E-unifier ρ of t and t', there is a $\sigma \in CSU_E(t, t')$ such that $\sigma \ll_E \rho$. For AC theories, a finite complete set of unifiers does exist [6].

A *(order-sorted) rewrite theory* is a triple $\mathcal{R} = (\Sigma, \Delta \cup B, R)$, where R is a set of rewrite rules of the form $l \to r$, with $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$, Σ is the pairwise disjoint union $\mathcal{D}_1 \uplus \mathcal{D}_2 \uplus \mathcal{C}$ such that $(\mathcal{D}_1 \uplus \mathcal{C}, \Delta \cup B)$ is an order-sorted equational theory, and $\mathcal{D}_2 = \{f \mid f(\bar{t}) \to r \in R\}$ is the set of symbols defined by the rules of R. We omit Σ when no confusion can arise. Throughout the paper, a rewrite theory is also called a program.

Given a rule $(l \to r)$ or an equation $(l = r)$, terms l and r are called the *left-hand side* (or *lhs*) and the *right-hand side* (or *rhs*) of the rule (*resp.* equation). An equation of the form $t = t'$ or a rule of the form $t \to t'$ is said to be *left (*resp. *right) linear*, if t (*resp.* t') is *linear*, i.e., no variable occurs in the term more than once. The equation $t = t'$ (resp. the rule $t \to t'$) is called *linear* if both t and t' are linear. A set of equations/rules is said to be (left or right) linear, if each equation/rule in it is (left or right) linear.

We define the *one-step rewrite relation* on $\mathcal{T}_\Sigma(\mathcal{X})$ as follows: $t \to_R t'$ if there is a position $p \in \mathcal{NVP}os(t)$, a rule $l \to r$ in R, and a substitution σ such that $t|_p = l\sigma$ and $t' = t[r\sigma]_p$. The relation $\to_{R/E}$ for rewriting modulo E is defined as $=_E \circ \to_R \circ =_E$. Let $\to \subseteq A \times A$ be a binary relation on a set A. We denote the transitive closure by $\to^+$, the reflexive and transitive closure by $\to^*$, and rewriting up to normal forms by $\to^!$.

Considering the rewrite relation $\to_{R/E}$, since E-congruence classes can be infinite, $\to_{R/E}$-reducibility is undecidable in general. One way to overcome this problem is to implement R/E-rewriting by a combination of rewriting using oriented equations (oriented from left to right) and rules [33]. We define the relation $\to_{\Delta,B}$ on $\mathcal{T}_\Sigma(\mathcal{X})$ as follows: $t \to_{\Delta,B} t'$ if there is a position $p \in \mathcal{NVP}os(t)$, $l = r$ in Δ, and a substitution σ such that $t|_p =_B l\sigma$ and $t' = t[r\sigma]_p$. The relation $\to_{R,B}$ is similarly defined, and we define $\to_{R\cup\Delta,B}$ as $\to_{R,B} \cup \to_{\Delta,B}$. The idea is to implement $\to_{R/E}$ using $\to_{R\cup\Delta,B}$.

The computability of $\to_{R\cup\Delta,B}$ as well as its equivalence w.r.t. $\to_{R/E}$ are assured by enforcing some conditions on the considered rewrite theories [24], [33]. More specifically, we ask for *coherence* between the rules and the equations as well as the assumption of *Church-Rosser* and *termination* properties of Δ modulo the equational axioms B. A formal description of these requirements can be found in Appendix I.

Example II.1 *Consider the following rewrite theory* $(\Sigma, \Delta \cup B, R)$ *such that* $\mathcal{C} = \{b, c, e\}$, $\mathcal{D}_1 = \{a, d\}$, $\mathcal{D}_2 = \{f\}$, $\Delta = \{a = b, d = e\}$, *and* $R = \{f(b, c) \to d\}$ *where* B *contains the commutativity axiom for* f. *Then we can* R/E*-rewrite term* $f(c, a)$ *to* e *by means of the following* $\to_{R\cup\Delta,B}$ *rewrite sequence* $f(c, a) \to_\Delta f(c, b) =_B f(b, c) \to_R d =_\Delta e$.

A term t is called a *redex*, if there exist a rule $l \to r$, or equation $l = r$, and a substitution σ such that $t =_B l\sigma$. A term t without redexes is called a *normal form*. A rewrite theory $\mathcal{R}$ is *weakly normalizing* if every term t has a normal form in $\mathcal{R}$, though infinite rewrite sequences starting from t may exist. A rewrite theory is *sufficiently complete* [20] if enough rules/equations have been specified so that functions of the theory are fully defined on all relevant data (that is, defined symbols do not appear in any ground term in normal form).

III. Narrowing in Rewriting Logic

Narrowing [17] generalizes term rewriting by allowing free variables in terms (as in logic programming) and by performing unification (at non-variable positions) instead of matching in order to (non-deterministically) reduce a term. The narrowing relation for rewriting logic theories is defined as follows [27].

Definition III.1 **($R \cup \Delta, B$-Narrowing)** *Let* $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ *be an order-sorted rewrite theory. The* $R \cup \Delta, B$-narrowing *relation on* $\mathcal{T}_\Sigma(\mathcal{X})$ *is defined as* $t \leadsto_{\sigma,p,R\cup\Delta,B} t'$ *if there exist* $p \in \mathcal{NVP}os(t)$, *a rule* $l \to r$ *or equation* $l = r$ *in* $R \cup \Delta$, *and* $\sigma \in CSU_B(t|_p, l)$ *such that* $t' = (t[r]_p)\sigma$. $t \leadsto_{\sigma,p,R\cup\Delta,B} t'$ *is also called a* $R \cup \Delta, B$-narrowing step.

Example III.1 *Consider the rewrite theory of Example II.1 where we substitute the rule in* R *with the following rule* $f(x, f(y, b)) \to d$. *Then we can perform the narrowing step* $f(f(w, z), c) \leadsto_{\sigma,\Lambda,R\cup\Delta,B} d$, *with* $\sigma = \{x/c, z/b, w/y\}$, *since by the commutativity of* f *we have that* $f(f(w, z), c)\{z/b, w/y\} =_B f(x, f(y, b))\{x/c\}$.

When it is clear from the context, we omit $(R \cup \Delta, B)$ from the narrowing relation. Narrowing derivations are denoted by $t_0 \leadsto^*_\sigma t_n$, which is shorthand for the sequence of narrowing steps $t_0 \leadsto_{\sigma_1,p_1} \ldots \leadsto_{\sigma_n,p_n} t_n$ with $\sigma = \sigma_n \circ \ldots \circ \sigma_1$ (if $n = 0$ then $\sigma = id$). Completeness of narrowing for several meaningful classes of rewriting logic theories (e.g. topmost theories, linear theories, *etc.*) has been studied in [27].

In rewriting logic implementation such as Maude, defined symbols can be given the commutativity axiom or both commutativity and associativity, but not the associativity alone since unification modulo associativity is infinitary, i.e., infinitely many unifiers may exist modulo associativity [6].

In what follows, we always consider weakly normalizing and sufficiently complete rewrite theories. These conditions are essential in order to prove the correctness and completeness of the unfolding operation w.r.t. the considered semantics (i.e., Theorem V.1).

IV. Incompleteness due to Unfolding

In [1], we proposed a fold/unfold-based transformation framework for optimizing rewriting logic theories which is based on narrowing. Starting from an initial, maybe inefficient, program we can transform it by using some elementary transformation rules. The essential rules are folding and unfolding, i.e., contraction and expansion of subexpressions of a program using the definitions of the program itself (or of a preceding one). We employ narrowing in order to empower the unfolding operation by calculating the instance of an existing rule to embed the unfolding rule automatically via unification. Other rules that have been considered are, instantiation, definition introduction/elimination and abstraction. In this paper, we focus on the unfolding operation, which allows us to expand a redex in the rhs of an equation or rule as follows.

Definition IV.1 (Unfolding) *Let $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ be a program and F be an equation (*resp. *rule) of the form $l = r$ (*resp. *$l \to r$) in $\mathcal{R}$. We obtain a new program from $\mathcal{R}$ by replacing F with the set of equations (*resp. *rules)*

$$\{l\sigma = r' \mid r \rightsquigarrow_{\sigma,\Delta,B} r' \text{ is a } \Delta, B \text{ narrowing step}\}$$

$$\{l\sigma \to r' \mid r \rightsquigarrow_{\sigma,R\cup\Delta,B} r' \text{ is a } R \cup \Delta, B \text{ narrowing step}\}$$

The following example suggests that right linearity must be required for completeness. For the sake of simplicity we omit sort declarations when specifying rewriting logic theories.

Example IV.1 *Consider the following rewrite theory $\mathcal{R} = (\Sigma_{\mathcal{R}}, \emptyset, R)$, where $\Sigma_{\mathcal{R}}$ is the signature containing all the symbols of R and*

	$R:$			$R':$		
1.	$f(d,d)$	$\to$	a	$f(d,d)$	$\to$	a
2.	$f(b,c)$	$\to$	b	$f(b,c)$	$\to$	b
3.	a	$\to$	b	a	$\to$	b
4.	a	$\to$	c	a	$\to$	c
5.	$g(x)$	$\to$	$f(x,x)$			
6.				$\mathbf{g(d)}$	$\to$	$\mathbf{a}$

We obtain program $\mathcal{R}' = (\Sigma_{\mathcal{R}}, \emptyset, R')$ from $\mathcal{R}$ by applying an unfolding step over the rule 5 in R, through the narrowing step $f(x,x) \rightsquigarrow_{x/d} a$. Let us consider term $g(a)$. In the original program, $g(a)$ can rewrite to the normal form b by the rewrite sequence: (i) $g(a) \to_5 f(a,a) \to_3 f(b,a) \to_4 f(b,c) \to_2 b$. In the transformed program, such a rewrite sequence is no longer possible from term $g(a)$, and, hence, the normal form b is lost.

We consider the standard semantics of rewrite theories given by the following definition.

Definition IV.2 (Program Semantics) *Given a rewrite theory $\mathcal{R} = (\Sigma, \Delta \cup B, R)$, the semantics of $\mathcal{R}$ is the set $\mathcal{S}(\mathcal{R}) = \{(s,t) \mid s \in \mathcal{T}_\Sigma, s \to^*_{R\cup\Delta,B} t\}$.*

Since we consider rewrite theories where defined symbols are allowed to be arbitrarily nested in left-hand sides of rules, rule unfolding may cause a loss of completeness for the transformed program w.r.t. the semantics of the original one. Let us illustrate this problem by means of some examples. Since the equational axioms for associativity and commutativity do not affect the incompleteness problem that we want to describe, for the sake of simplicity, in the following examples, we consider defined symbols without any equational axiom. A discussion on equational axioms and incompleteness is postponed until Section IV-C.

Example IV.2 *Consider the following rewrite theory $\mathcal{R} = (\Sigma_{\mathcal{R}}, \emptyset, R)$, where $\Sigma_{\mathcal{R}}$ is the signature containing all the symbols of R and*

	$R:$			$R':$		
1.	$g_1(x)$	$\to$	x	$g_1(x)$	$\to$	x
2.	$h(x)$	$\to$	0	$h(x)$	$\to$	0
3.	$h(g_1(x))$	$\to$	1	$h(g_1(x))$	$\to$	1
4.	$f(x)$	$\to$	$g_1(x)$			
5.				$\mathbf{f(x)}$	$\to$	$\mathbf{x}$

We get program $\mathcal{R}' = (\Sigma_{\mathcal{R}}, \emptyset, R')$ from $\mathcal{R}$ by applying an unfolding step over rule 4 in R, through the narrowing step $g_1(x) \rightsquigarrow_{\varepsilon} x$. Term $h(f(0))$ can be rewritten in $\mathcal{R}$ to the normal forms 0 or 1 by means of the rewrite sequences $h(f(0)) \to_4 h(g_1(0)) \to_1 h(0) \to_2 0$, and $h(f(0)) \to_4 h(g_1(0)) \to_3 1$, respectively. The only possible rewrite sequences from $h(f(0))$ in $\mathcal{R}'$ are $h(f(0)) \to_5 h(0) \to_2 0$, and $h(f(0)) \to_2 0$, thus we miss normal form 1. In fact, symbol g_1 is needed for rule 3 to be applied, and function f provides that occurrence of g_1 needed to reach the normal form 1. However, the unfolding of rule 4 forces the occurrence of symbol g_1 to be evaluated, and, hence, that rewrite sequence is no longer available in $\mathcal{R}'$.

A naïve attempt to identify the rules that are involved in the loss of completeness might be to look for those rules whose left-hand sides contain an instance of the right-hand side of the rule that we want to unfold. In Example IV.2, the right-hand side of rule 4 is embedded in the left-hand side of rule 3. Hence, in order to avoid incompleteness, we could forbid the unfolding operation whenever one such a rule existed in the program. In [1] we imposed such a syntactic restriction on the unfolding operation. Unfortunately, as shown by Example IV.3, in general, incompleteness can be caused by the interference among several rules, which cannot be identified by using the naïve criterion proposed in [1].

Example IV.3 *Consider the following rewrite theory $\mathcal{R} = (\Sigma_{\mathcal{R}}, \emptyset, R)$, where $\Sigma_{\mathcal{R}}$ is the signature that contains all the symbols of R and*

	$R:$			$R':$		
1.	$g_1(x,0)$	$\to$	x	$g_1(x,0)$	$\to$	x
2.	$g_1(x,1)$	$\to$	x	$g_1(x,1)$	$\to$	x
3.	$g_1(0,g_1(x,y))$	$\to$	0	$g_1(0,g_1(x,y))$	$\to$	0
4.	$g_2(x)$	$\to$	x	$g_2(x)$	$\to$	x
5.	$h(x,y)$	$\to$	x	$h(x,y)$	$\to$	x
6.	$h(g_2(x),y)$	$\to$	$p(x,y)$	$h(g_2(x),y)$	$\to$	$p(x,y)$
7.	$p(x,y)$	$\to$	x	$p(x,y)$	$\to$	x
8.	$p(g_1(x,y),z)$	$\to$	1	$p(g_1(x,y),z)$	$\to$	1
9.	$k(x)$	$\to$	x	$k(x)$	$\to$	x
10.	$k(g_2(x))$	$\to$	1	$k(g_2(x))$	$\to$	1
11.	$f(x,y)$	$\to$	$g_2(g_1(x,y))$			
12.				$\mathbf{f(x,0)}$	$\to$	$\mathbf{g_2(x)}$
13.				$\mathbf{f(x,1)}$	$\to$	$\mathbf{g_2(x)}$
14.				$\mathbf{f(0,g_1(x,y))}$	$\to$	$\mathbf{g_2(0)}$
15.				$\mathbf{f(x,y)}$	$\to$	$\mathbf{g_1(x,y)}$

We obtain program $\mathcal{R}' = (\Sigma_{\mathcal{R}}, \emptyset, R')$ from $\mathcal{R}$ by applying an unfolding step over rule 11 in R, through the following narrowing steps: (i) $g_2(g_1(x,y)) \rightsquigarrow_{\varepsilon} g_1(x,y)$, (ii) $g_2(g_1(x,y)) \rightsquigarrow_{y/0} g_2(x)$, (iii) $g_2(g_1(x,y)) \rightsquigarrow_{y/1} g_2(x)$, and (iv) $g_2(g_1(x,y)) \rightsquigarrow_{x/0,y/g_1(x',y')} g_2(0)$. The following *rewrite sequence can be proved in $\mathcal{R}$: $h(f(0,1),0) \rightarrow_{11} h(g_2(g_1(0,1)),0) \rightarrow_6 p(g_1(0,1),0) \rightarrow_8 1$. In $\mathcal{R}'$ we cannot reach the normal form 1 starting from term $h(f(0,1),0)$ because rules 6 or 8 cannot be applied. This is due to the fact that the occurrences of both symbols g_2 and g_1 is essential for rules 6 and 8 to be applied in order to obtain the normal form 1, while the unfolding step forces these occurrences to be evaluated. Therefore, in the transformed program, the rewrite sequence leading to normal form 1 is no longer viable. In this example, rules 6 and 8 are both involved in the loss of completeness.*

The naïve idea outlined above to solve the case in Example IV.2 does not apply to Example IV.3 because the right-hand side of rule 11 does not appear in the left-hand side of any rule; however, it is distributed between the left-hand sides of rules 6 and 8.

In the following, we develop a methodology that is able to identify whether an unfolding operation causes incompleteness, and we overcome this problem by conveniently extending the transformed program. More precisely, according to the identified incompleteness sources, the methodology derives a set of new rules that are added to the transformed program in order to recover the ground semantics of the original program.

A. Analyzing potential incompleteness

Let $\mathcal{R} = (\Sigma, E, R)$ be a program, let $R^u : lhs_u \rightarrow rhs_u \in R$ be the rule that we want to unfold and let $\mathcal{R}'$ be the program obtained from $\mathcal{R}$ by performing the unfolding operation.

Step 1) *Looking for rules that may be involved in incompleteness.*
At the beginning, we look for rules in R whose left-hand side contains a proper subterm rooted by the root symbol of rhs_u. Let $\{R_1, \ldots, R_n\}$ be such a set of rules, and for each $lhs_i, i \in \{1, \ldots, n\}$, let $p_1, \ldots, p_{k_i}$ be the positions in lhs_i where an occurrence of the root symbol of rhs_u has been found. Then we construct the following set of terms $L = \{lhs_i[rhs_u]_{p_j} \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, k_i\}\}$, where we replace the subterm rooted at position p_j in each lhs_i, with the right-hand side rhs_u. In order to avoid interference among the variables of rhs_u and the context $lhs_i[\ ]_{p_j}$, we consider a variable renaming of rhs_u with fresh variables.

Finally, for each term $lhs_i[rhs_u]_{p_j}$, we try to perform just one narrowing step at the root position using the corresponding rule R_i. In symbols, we try to perform the following narrowing step: $lhs_i[rhs_u]_{p_j} \rightsquigarrow_{\sigma_j, \Lambda, (R_i \cup \Delta, B)} r'_j$. We collect the derived terms in a set T of triples of the form $(lhs_i[lhs_u]_{p_j}, \sigma_j, r'_j)$, where the first component is the lhs_i where we replaced the subterm rooted at position p_j with the left-hand side lhs_u. We consistently apply to lhs_u the same variable renaming applied to rhs_u.

Roughly speaking, if the considered narrowing step cannot be done, it follows that no rewrite step can be performed with rule R_i from any instance of term $lhs_i[rhs_u]_{p_j}$, and, hence, there is no incompleteness. Otherwise, the methodology proceeds to verify whether term r'_j can be reached in $\mathcal{R}'$ from term $lhs_i[lhs_u]_{p_j}\sigma_j$.

Example IV.4 *Let us again consider the rules of Example IV.3. Recall that the rule for unfolding is $f(x,y) \rightarrow g_2(g_1(x,y))$. We first look for rules whose left-hand sides contain a proper subterm rooted with symbol g_2, and we find rules 6 and 10. We then construct the set L that contains terms $h(g_2(g_1(w,z)),y)$ and $k(g_2(g_1(w,z)))$, and we try to perform a narrowing step at the root position from each of these terms by using rules 6 and 10, respectively.*

We can perform the following narrowing steps:

$$h(g_2(g_1(w,z)),y) \rightsquigarrow_{\sigma,\Lambda,R_6} p(g_1(w,z),y)$$

where the computed unifier is $\sigma = \{x/g_1(w,z)\}$, and

$$k(g_2(g_1(w,z))) \rightsquigarrow_{\rho,\Lambda,R_{10}} 1$$

where the computed unifier is $\rho = \{x/g_1(w,z)\}$.

Finally, we construct the set T which contains the triples

$$(h(f(w,z),y), \{x/g_1(w,z)\}, p(g_1(w,z),y))$$
$$(k(f(w,z)), \{x/g_1(w,z)\}, 1)$$

Step 2) *Restoring Completeness.*
For each triple $(t_1, \sigma, t_2) \in T$, we add rule $t_1\sigma \rightarrow t_2$ to $\mathcal{R}'$. This guarantees that the ground semantics of $\mathcal{R}$ is preserved in the new program $\mathcal{R}'$, as stated by Theorem V.1. In our example, we add rules $h(f(w,z),y) \rightarrow p(g_1(w,z),y)$ and $k(f(w,z)) \rightarrow 1$ to $\mathcal{R}'$.

Figure 1 shows the backbone of the procedure that implements the methodology above. The *restoreCompleteness* procedure takes the initial program $\mathcal{R}$, the transformed program $\mathcal{R}'$, and the right-hand side of the unfolded rule as arguments, and it returns $\mathcal{R}'$ extended with some new rules computed as explained above. The *getInvolvedRules* call detects the rules in $\mathcal{R}$ that contain a proper term whose root symbol is $root(rhs_u)$ in their lhs. $subst\_rhs_u$ replaces the subterms rooted with the function symbol $root(rhs_u)$ in the lhs of the rules by term rhs_u, and *narrowingOneStep* tries to perform a narrowing step from the obtained terms by using the corresponding suspicious rules, obtaining the set of triples $\{(t_1, \sigma, t_2)\}$. Finally, for each one of these triples, the *prodRules* call returns a new rule of the form $t_1\sigma \rightarrow t_2$ to be added to the program $\mathcal{R}'$.

Example IV.5 *Consider again the Example IV.3. The call $restoreCompleteness((\Sigma_{\mathcal{R}}, \emptyset, R), (\Sigma_{\mathcal{R}}, \emptyset, R'), g_2(g_1(x,y)))$ yields $(\Sigma_{\mathcal{R}}, \emptyset, R' \cup \{h(f(w,z),y) \rightarrow p(g_1(w,z),y), k(f(w,z)) \rightarrow 1\})$.*

B. Methodology optimization

In the methodology above, in order to prevent a possible incompleteness problem, we add a rule of the form $t_1\sigma \rightarrow t_2$ to the transformed program $\mathcal{R}'$ for each triple (t_1, σ, t_2) found at Step 1, even if the transformed program is actually complete. Consider again the rules of Example IV.3, and term $k(f(x,y))$.

By applying to $k(f(x, y))$ the substitution $\{y/0\}$ computed by the unfolding operation, we can rewrite the obtained term in the transformed program to the normal form 1, by means of the following rewrite sequence: $k(f(x, 0)) \rightarrow_{12} k(g_2(x)) \rightarrow_{10} 1$. Hence, the rule $k(f(w, z)) \rightarrow 1$ added to the transformed program by the methodology is redundant because rule 10 does not provoke incompleteness.

To refine the methodology, we can add an intermediate step that checks whether it is really necessary to add a new rule to the program. Let Σ_u denote the set of substitutions computed by narrowing during the unfolding operation extended with the empty substitution. Then, for each triple $(t_1, \sigma, t_2) \in T$, we want to check whether there exists $\sigma_u \in \Sigma_u$ such that t_2 is reachable from $(t_1\sigma)\sigma_u$ in $\mathcal{R}'$ by rewriting. If that is the case, there is no reason to add a rule that would be redundant; otherwise, this is a symptom of incompleteness, and we can proceed as in Step 2.

Example IV.6 *Consider again the Example IV.3, and the triples* $(h(f(w, z), y), \{x/g_1(w, z), y/y\}, p(g_1(w, z), y))$ *and* $(k(f(w, z)), \{x/g_1(w, z)\}, 1)$ *computed at Step 1. The set* Σ_u *contains the substitutions* $\{\varepsilon\}, \{y/0\}, \{y/1\}$, *and* $\{x/0, y/g_1(x', y')\}$. *Then, we check whether there exists* $\sigma_u \in \Sigma_u$ *such that* $h(f(w, z), y)\sigma_u \rightarrow^*_{\mathcal{R}'} p(g_1(w, z), y))$, *and whether there exists* $\sigma_u \in \Sigma_u$ *such that* $k(f(w, z))\sigma_u \rightarrow^*_{\mathcal{R}'} 1$. *The first reachability goal is unsatisfiable, while the second is satisfied by substitutions* $\{y/0\}, \{y/1\}$, *and* $\{x/0, y/g_1(x', y')\}$.

Hence, the optimized solution is to add only the rule $h(f(w, z), y) \rightarrow (g_1(w, z), y))$ *to the transformed program.*

The reachability problem for rewriting is undecidable in general, but it has been proved to be decidable for particular classes of rewrite theories [18], [25]. For example, in [18] reachability is proved to be decidable for right-linear and right-shallow TRSs. The right-shallow property asks for variables that appear in the right-hand side of the rules to occur at depth 0 or 1. Hence, the proposed refinement has to pay the cost of the additional syntactic restrictions of right-linearity and right-shallowness to be effective. An alternative method to make reachability decidable is presented in [26], where the original rewrite theory is extended by adding a terminating and (ground) Church-Rosser set of extra equations powerful enough to collapse infinite sets of reachable terms into finite sets. Also in this case, several strong conditions are required on the extended rewrite theory in order to make such an analysis effective.

$$
\begin{aligned}
&restoreCompleteness((\Sigma, E, R), (\Sigma, E, R'), rhs_u) = \\
&\quad (\Sigma, E, R' \cup \{t_1\sigma \rightarrow t_2\}) \\
&\quad where \\
&\quad \{R_1, \ldots, R_n\} \leftarrow getInvolvedRules(R, rhs_u|_\Lambda) \\
&\quad L \leftarrow subst\_rhs_u(\{lhs_1, \ldots, lhs_n\}, rhs_u) \\
&\quad \{(t_1, \sigma, t_2)\} \leftarrow narrowingOneStep(L, \{R_1, \ldots, R_n\}) \\
&\quad \{t_1\sigma \rightarrow t_2\} \leftarrow prodRules(\{(t_1, \sigma, t_2)\})
\end{aligned}
$$

Fig. 1. Procedure to check and restore completeness of unfolding

C. Incompleteness and Equational Axioms

Up to now, we have explained the incompleteness problem that may arise due to the unfolding operation, without considering equational axioms which can be associated with defined symbols. Nevertheless, the unfolding operation uses the $R \cup \Delta, B$ narrowing relation, which takes into account the equational axioms for associativity and commutativity. However, the axioms are not an extra source of incompleteness, as discussed below.

Let us modify the rewrite theory of Example IV.3 by declaring the symbols h, p and g_1 to obey associativity and commutativity. The transformed program will have a higher number of unfolded rules due to the increased number of unifiers computed by narrowing modulo the considered axioms, but exactly the same incompleteness problem arises. The new rules computed by unfolding are:

$$
\begin{array}{lrcl}
12. & f(x, 0) & \rightarrow & g_2(x) \\
13. & f(0, x) & \rightarrow & g_2(x) \\
14. & f(x, 1) & \rightarrow & g_2(x) \\
15. & f(1, x) & \rightarrow & g_2(x) \\
16. & f(0, g_1(x, y)) & \rightarrow & g_2(0) \\
17. & f(g_1(x, y), 0) & \rightarrow & g_2(0) \\
18. & f(g_1(0, x), y) & \rightarrow & g_2(0) \\
19. & f(y, g_1(0, x)) & \rightarrow & g_2(0) \\
20. & f(x, y) & \rightarrow & g_1(x, y)
\end{array}
$$

and allow us to bring back the original semantics for the transformed program. Note that rules 13, 15, 17, 18, and 19 are needed because f is not associative neither commutative.

V. Completeness of the Transformation

The main result of this paper is Theorem V.1, which states that the unfolding transformation followed by the *restore-Completeness* procedure preserves the ground semantics of a program. Moreover, the equational unfolding preserves the canonical forms as stated in Proposition V.1.

Proposition V.1 *Let* $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ *be a program, and let* $\mathcal{R}' = (\Sigma, \Delta' \cup B, R)$ *be the program obtained from* $\mathcal{R}$ *by unfolding an equation* $E^u \in \Delta$. *Then, for each* $t \in \mathcal{T}_\Sigma$, *if* s *is its canonical form w.r.t* Δ *and* s' *is its canonical form w.r.t.* Δ', *then* $s =_B s'$.

Theorem V.1 *Let* $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ *be a program, and let* $\mathcal{R}' = (\Sigma, \Delta \cup B, R')$ *be the program obtained from* $\mathcal{R}$ *by the unfolding of a rule* $R^u \in R$ *and the* restoreCompleteness *procedure. Then, for each term* $t \in \mathcal{T}_\Sigma$, *we have that:*

- $t \rightarrow^*_{\mathcal{R}'} s' \Rightarrow t \rightarrow^*_{\mathcal{R}} s$ *and* $s =_{\Delta,B} s'$;
- $t \rightarrow^*_{\mathcal{R}} s \Rightarrow t \rightarrow^*_{\mathcal{R}'} s'$ *and* $\exists s''$ *s.t.* $s \rightarrow^*_{\mathcal{R}} s''$, $s' =_{\Delta,B} s''$.

Basically Theorem V.1 states that (i) the ground reducts of the transformed program are exactly the same as in the original one, and (ii) for each ground reduct s of the original program, there exists s' in the transformed one such that s can still be reduced to a term that is equivalent to s'. This asymmetry in the result is due to the nature of unfolding. In fact, the unfolding of a rule in the initial program forces some symbols that appear in its right-hand side to be reduced by narrowing, and, hence, a reduct s obtained by an application of that rule may contain those symbols. Therefore, we need to consider the possibility of some further reduction steps from s in the initial program in order to reduce those symbols and thereby obtain a term that is equivalent to the one reachable in the transformed program. A detailed proof can be found in Appendix II. Basically, in order to prove Theorem V.1 we first prove that the Unfolding operation preserves the semantics of ground normal forms. The proof uses an induction on the rewrite sequence and an *ad hoc* reordering function on the sequence itself. Then, the result is straightforwardly extended to infinite sequences.

VI. Conclusions

In this paper, we have considered the completeness of the unfolding transformation w.r.t. the standard semantics of rewriting logic theories. We have taken on the systematic study of program transformations for unrestricted narrowing because it brings to light some common problems caused by the basic mechanism that are not tied to the intricacies of any particular strategy. We have ascertained and exemplified general conditions that guarantee that the meaning of the program is not modified by the transformation. These conditions, which are quite natural in practical rewriting logic specifications, cover many common cases and are easy to check since they are mostly syntactical and do not depend on the final program, but only on the initial one. Actually, they can be checked by using the Maude Church-Rosser, Termination, Sufficient Completeness, and Coherence tools [12]. The Unfolding operation discussed in this paper can be employed together with other program transformation operators (e.g. folding and definition introduction/elimination) to synthesize new optimized, and semantically equivalent rewrite theories from naïve specifications. In particular, the synthesis of rewrite theories based on Fold/Unfold transformations has been used in [1] to implement Code Carrying Theory (CCT) — a methodology for securing delivery of code from a producer to a consumer where only a certificate (usually in the form of assertions and proofs) is transmitted from a producer to a consumer who can check its validity and then synthesize executable code from it.

As future work it would be interesting to investigate whether the presented results remain valid if we consider an arbitrary equational theory with finitary unification instead of C- and AC-theories only.

References

[1] M. Alpuente, M. Baggi, D. Ballis, and M. Falaschi. A Fold/Unfold Transformation Framework for Rewrite Theories extended to CCT. In ACM, editor, *In Proc. of ACM SIGPLAN 2010 Workshop on Partial Evaluation and Program Manipulation (PEPM'10)*, pages 43–52, New York, NY, USA, 2010.

[2] M. Alpuente, M. Falaschi, G. Moreno, and G. Vidal. Safe folding/unfolding with conditional narrowing. In *6th International Joint Conference on Algebraic and Logic Programming*, pages 1–15. Springer-Verlag, 1997.

[3] M. Alpuente, M. Falaschi, G. Moreno, and G. Vidal. Rules + strategies for transforming lazy functional logic programs. *Theoretical Computer Science*, 311(1-3):479–525, 2004.

[4] M. Alpuente, M. Falaschi, and G. Vidal. Partial Evaluation of Functional Logic Programs. *ACM Transactions on Programming Languages and Systems (TOPLAS '98)*, 20(4):768–844, 1998.

[5] M. Alpuente, M. Falaschi, and G. Vidal. A Unifying View of Functional and Logic Program Specialization. *ACM Computing Surveys*, 30(3es):9–es, September 1998.

[6] F. Baader and W. Snyder. *Handbook of Automated Reasoning*, chapter Unification Theory. Elsevier and MIT Press, 2001.

[7] J.A. Bergstra, J Heering, and P. Klint. *Algebraic Specification*. ACM Press, 1989.

[8] P. Borovanský, C. Kirchner, H. Kirchner, and P. E. Moreau. ELAN from a rewriting logic point of view. *Theoretical Computer Science*, 285:155–185, 2002.

[9] A. Bouhoula, J.P. Jouannaud, and J. Meseguer. Specification and Proof in Membership Equational Logic. *Theoretical Computer Science*, 236(1-2):35–132, 2000.

[10] R.M. Burstall and J. Darlington. A Transformation System for Developing Recursive Programs. *Journal of ACM*, 24(1):44–67, 1977.

[11] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. Unification and Narrowing in Maude 2.4. In Ralf Treinen, editor, *Procs. of 20th International Conference on Rewriting Techniques and Applications, (RTA '09), Brasília, Brazil*, volume 5595 of *Lecture Notes in Computer Science*, pages 380–390. Springer-Verlag, 2009.

[12] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude - A High-Performance Logical Framework*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[13] N. Dershowitz. Computing with Rewrite Systems. *Information and Control*, 64(2-3):122–157, 1985.

[14] N. Dershowitz and U. Reddy. Deductive and Inductive Synthesis of Equational Programs. *Journal of Symbolic Computation*, 15:467–494, 1993.

[15] R. Diaconescu and K. Futatsugi. *CafeOBJ Report*, volume 6 of *AMAST Series in Computing*. World Scientific, AMAST Series, 1998.

[16] S. Etalle and M. Gabbrielli. Modular Transformations of CLP Programs. In L. Sterling, editor, *26th International Conference on Logic Programming*. MIT Press, 1995.

[17] M. Fay. First Order Unification in an Equational Theory. In *Procs. of 4th International Conference on Automated Deduction*, pages 161–167, 1979.

[18] G. Feuillade, T. Genet, and V. Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. Research Report RR-4970, INRIA, 2003.

[19] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.P. Jouannaud. Introducing OBJ. In *Software Engineering with OBJ: Algebraic Specification in Action*, pages 3–167. Kluwer, 2000.

[20] J. Hendrix, J. Meseguer, and H. Ohsaki. A sufficient completeness checker for linear order-sorted specifications modulo axioms. In U. Furbach and N. Shankar, editors, *Third International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 151–155. Springer, 2006.

[21] T. Kawamura and T. Kanamori. Preservation of Stronger Equivalence in Unfold/Fold Logic Programming Transformation. In *Future Generation Computer Systems*, pages 413–422. ICOT, 1988.

[22] H.J. Komorowski. Partial Evaluation as a Means for Inferencing Data Structures in an Applicative Language: A Theory and Implementation in the Case of Prolog. In *Proc. of 9th ACM Symposium on Principles of Programming Languages*, pages 255–267, 1982.

[23] L. Kott. Unfold/fold program transformation. In M. Nivat and J.C. Reynolds, editors, *Algebraic methods in semantics*, chapter 12, pages 411–434. Cambridge University Press, 1985.

[24] N. Martí-Oliet and J. Meseguer. Rewriting Logic: Roadmap and Bibliography. *Theoretical Computer Science*, 285(2):121–154, 2002.

[25] R. Mayr and M. Rusinowitch. Reachability is Decidible for Ground AC Rewrite Systems. In *Proc. of the 3rd INFINITY Workshop*, pages 53–64, 1998.
[26] J. Meseguer, M. Palomino, and N. Martí-Oliet. Equational Abstractions. *Theoretical Computer Science*, 403(2-3):239–264, 2008.
[27] J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher Order Symbolic Computation*, 20(1-2):123–160, 2007.
[28] A. Pettorossi and M. Proietti. Transformation of Logic Programs: Foundations and Techniques. *Journal of Logic Programming*, 19,20:261–320, 1994.
[29] U. S. Reddy. Rewriting Techniques for Program Synthesis. In *Proc. of Rewriting Techniques and Applications, (RTA'89)*, volume 355 of *Lecture Notes in Computer Science*, pages 388–403. Springer, 1989.
[30] P. Réty. Improving Basic Narrowing Techniques and Commutation Properties. In *Rewriting Techniques and Applications*, volume 256 of *LNCS*, pages 228–241, 1987.
[31] H. Seki. Unfold/fold Transformation of General Logic Programs for the Well-Founded Semantics. *Journal of Logic Programming*, 16(1&2):5–23, 1993.
[32] H. Tamaki and T. Sato. Unfold/Fold Transformations of Logic Programs. In *Procs. of the 2nd International Conference on Logic Programming, (ICLP '84)*, pages 127–139, 1984.
[33] P. Viry. Rewriting: An Effective Model of Concurrency. In *Procs. of the 6th International Conference on Parallel Architectures and Languages Europe, (PARLE '94)*, pages 648–660, London, UK, 1994. Springer-Verlag.

VII. Appendix I

Let $\mathcal{R} = (\Sigma, E, R)$ with $E = \Delta \cup B$ be an order-sorted rewrite theory. An equation of the form $t = t'$ or a a rule of the form $t \to t'$ are said to be:

(1) *Non-erasing*, if $\mathcal{V}ar(t) = \mathcal{V}ar(t')$.
(2) *Sort preserving*, if for each substitution σ, we have $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_s$ if and only if $t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_s$.
(3) *Sort decreasing*, if for each substitution σ, $t'\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_s$ implies $t\sigma \in \mathcal{T}_\Sigma(\mathcal{X})_s$.

A set of equations/rules is said to be non-erasing, or sort decreasing, or sort preserving, if each equation/rule in it is.

When implementing the R/E-rewriting by means of the relation $R \cup \Delta, B$, we want the following equivalence to be satisfied: $t_1 \to_{R/E} t_2$ if and only if $t_1 \to^*_{\Delta,B}\to_{R,B} t_3$ for some $t_3 =_E t_2$.
This implies that $t_1 \to^*_{R/E} t_2$ if and only if $t_1 \to^*_{R\cup\Delta,B} t_3$ for some $t_3 =_E t_2$. In order to assure this equivalence, we enforce the following properties on E.

(i) B is *non-erasing* and *sort preserving*.
(ii) B has a finitary and complete unification algorithm, which implies that B-matching is decidable; and $\Delta \cup B$ has a complete (but not necessarily finite) unification algorithm.
(iii) Δ is *sort decreasing*, *Church-Rosser*, and *terminating modulo B*.
(iv) $\to_{\Delta,B}$ is *coherent with B*, i.e., $\forall t_1, t_2, t_3$, we have that $t_1 \to^+_{\Delta,B} t_2$ and that $t_1 =_B t_3$ implies $\exists t_4, t_5$ such that $t_2 \to^*_{\Delta,B} t_4$, $t_3 \to^+_{\Delta,B} t_5$, and $t_4 =_E t_5$.
(v) $\to_{R,B}$ is *E-consistent with B*, i.e., $\forall t_1, t_2, t_3$, we have that $t_1 \to_{R,B} t_2$ and that $t_1 =_B t_3$ implies $\exists t_4$ such that $t_3 \to_{R,B} t_4$, and $t_2 =_E t_4$.
(vi) $\to_{R,B}$ is *E-consistent with* $\to_{\Delta,B}$, i.e., $\forall t_1, t_2, t_3$, we have that $t_1 \to_{R,B} t_2$ and that $t_1 \to^*_{\Delta,B} t_3$ implies $\exists t_4, t_5$ such that $t_3 \to^*_{\Delta,B} t_4$, $t_4 \to_{R,B} t_5$, and $t_5 =_E t_2$[1].

VIII. Appendix II

The main result of this paper is Theorem V.1, which states that the unfolding transformation followed by the *restoreCompleteness* procedure preserves the ground semantics of a program. This result is obtained as a corollary of Theorem VIII.1, which states that the ground normal forms are preserved. The following definitions, propositions and lemmas are auxiliary.

Definition VIII.1 (**B-Matching**) *Let $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ be a rewrite theory. Given two terms t and s (not just variables), we say that t B-matches s at position $p \in \mathcal{NVP}os(s)$, if there exists a substitution σ such that $t\sigma =_B s|_p$.*

Definition VIII.2 *The restriction of a substitution σ to a set of variables V is defined as*

$$\sigma|_V(x) = \begin{cases} \sigma(x) & \textit{if } x \in V \\ x & \textit{otherwise} \end{cases}$$

Proposition VIII.1 *Let $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ be a rewrite theory, let t_1, t_2 be two terms such that $\mathcal{V}ar(t_1) \cap \mathcal{V}ar(t_2) = \emptyset$, and let $CSU_B(t_1, t_2)$ be the complete set of B-unifiers of t_1 and t_2. Let also θ be a ground substitution such that t_2 B-matches $t_1\theta$ at position Λ. Then, there exists a substitution $\sigma \in CSU_B(t_1, t_2)$, such that the restriction of σ to the variables of t_1 is more general than θ.*

Proof: From the hypothesis it follows that there exists a substitution ρ such that $t_2\rho =_B t_1\theta$. Since t_1 and t_2 do not have shared variables, we can define a substitution η as the union of θ and ρ, such that $\eta|_{\mathcal{V}ar(t_1)} = \theta$ and $\eta|_{\mathcal{V}ar(t_2)} = \rho$. Therefore, η is a B-unifier of t_1 and t_2, that is, $t_1\eta =_B t_2\eta$. From the definition of CSU_B, we know that there exists a substitution $\sigma \in CSU_B(t_1, t_2)$ such that $\sigma \ll_B \eta$. Hence, $\sigma|_{\mathcal{V}ar(t_1)} \ll_B \eta|_{\mathcal{V}ar(t_1)} = \theta$. ■

Proposition VIII.2 *Let $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ be a program, and let $\mathcal{R}' = (\Sigma, \Delta \cup B, R')$ be the program obtained from $\mathcal{R}$ by the unfolding of a rule $R^u : lhs_u \to rhs_u \in R$ and the* restore-Completeness *procedure. Then for each term $lhs_i[rhs_u]_{p_j}$ in L, we have that if $lhs_i[rhs_u]_{p_j} \leadsto_{\sigma,\Lambda,(R_i\cup\Delta,B)} t$, then $lhs_i[lhs_u]_{p_j}\sigma \to t$ in $\mathcal{R}'$.*

Proof: The proof follows immediately from the described methodology; indeed, if $lhs_i[rhs_u]_{p_j} \leadsto_{\sigma,\Lambda,(R_i\cup\Delta,B)} t$, then the triple $(lhs_i[lhs_u]_{p_j}, \sigma, t)$ belongs to set T. Thus, at Step 2 we add the rule $lhs_i[lhs_u]_{p_j}\sigma \to t$ to program $\mathcal{R}'$, which implies the thesis. ■

Lemma VIII.1 *Let $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ be a program, and let $\mathcal{R}' = (\Sigma, \Delta \cup B, R')$ be the program obtained from $\mathcal{R}$*

[1] Properties (iv) and (v) can be achieved by a simple preprocessing of rewrite rules, while property (vi) is guaranteed by a discipline that prevents the defined symbols of Δ to appear within the lhs's of the rules in R. For more details see [1].

by the unfolding of a rule $R^u : lhs_u \to rhs_u \in R$ and the restoreCompleteness *procedure. Let $R_i : lhs_i \to rhs_i$, $i \in \{1, \ldots, n\}$ be the rules returned by the $getInvolvedRules$ call of the* restoreCompleteness *procedure. Let t be a ground term such that $t \to_{R^u,\Delta\cup B,p} t' \to_{R_i,\Delta\cup B,p'} t''$ such that $p' < p$, and the occurrence of $root(rhs_u)$ at some position p_j in lhs_i matches the occurrence of $root(rhs_u)$ at position p in t'. Then, $t \to^* t''' =_{\Delta\cup B} t''$ in $\mathcal{R}'$.*

Proof: From the hypothesis, it follows that $t|_p =_B lhs_u\theta$ for some ground substitution θ, and $t|_{p'} =_B lhs_i[lhs_u]_{p_j}\theta$. It follows that t' must be of the form $t[lhs_i[rhs_u]_{p_j}]_{p'}\theta$. Moreover, $lhs_i[rhs_u]_{p_j}\theta$ B-matches lhs_i. From the methodology described in Section IV-A, we know that narrowing computes the complete set of B-unifiers $CSU_B(lhs_i[rhs_u]_{p_j}, lhs_i)$. By Proposition VIII.1, it follows that there exists $\sigma \in CSU_B(lhs_i[rhs_u]_{p_j}, lhs_i)$ such that σ is more general than θ (by renaming $lhs_i[rhs_u]_{p_j}$ with t_1 and lhs_i with t_2). Then, there exists a substitution ρ such that $\theta =_B \sigma\rho$. It follows that $t[lhs_i[rhs_u]_{p_j}]_{p'} \rightsquigarrow_{\sigma,p',(R_i\cup\Delta,B)} t^*$ and $t^*\rho =_B t''$. By Proposition VIII.2 it follows that $t[lhs_i[lhs_u]_{p_j}]_{p'}\sigma \to^* t^*$ in $\mathcal{R}'$. Hence, $t = t[lhs_i[lhs_u]_{p_j}]_{p'}\theta =_B (t[lhs_i[lhs_u]_{p_j}]_{p'}\sigma)\rho \to^*_{R',\Delta\cup B} t^*\rho =_B t''$. ■

Since we ask for Δ to be Church-Rosser and terminating modulo B, the equational unfolding preserves the canonical forms, as stated in the Proposition V.1.

Proof of Proposition V.1: Let E^u be an equation of the form $(lhs_u = rhs_u)$, and let $E_0, \ldots, E_k$ be the set of equations used to unfold E^u, each one of the form $(l_i = r_i)$ for $i = 0, \ldots, k$. Let $f_1, \ldots, f_n$ with $n \leq k$ be the set of symbols defined by equations $E_1, \ldots, E_k$. Also let $rhs_u \rightsquigarrow_{\sigma_j,\Delta,B} r'_j$ $(j \in \{1, \ldots, n\})$ be the Δ, B-narrowing step such that the result of unfolding E^u using E_j is the equation $E^u_j : (lhs_u\sigma_j = r'_j)$. From the definition and the correctness of narrowing, we recall that:

(1) $\forall j \,.\, rhs_u\sigma_j \to^{E_j} r'_j$
(2) $\forall j$ there exists position $p_j \in \mathcal{NVP}os(rhs_u)$ such that $rhs_u|_{p_j}\sigma_j =_B l_j\sigma_j$
(3) $\forall j \,.\, r'_j = (rhs_u[r_j]_{p_j})\sigma_j$

($\Rightarrow$) We want to prove that, given any ground term t, if $t \to^!_{\Delta,B} s$, then $t \to^!_{\Delta',B} s'$ and $s =_B s'$. From $t \to^!_{\Delta,B} s$, the Church-Rosser property, and the termination of $\Delta\cup B$, there exists a rewrite sequence from t to s where the left-most inner-most redex is reduced at each step. We will prove the result by induction on the length of this rewrite sequence.
(n = 0.) This case is immediate since $t =_B s$.
(n > 0.) Let us decompose the rewriting sequence from t to s as follows: $t \to t_1 \to^! s$. On the rewriting sequence from t_1 to s, we can apply the induction hypothesis, and we now concentrate on the first rewriting step. If t rewrites to t_1 without using equation E^u, the same step can be performed in Δ' and the claim holds. Otherwise, there exists a position $p \in \mathcal{NVP}os(t)$ and a substitution θ such that (i) $lhs_u\theta =_B t|_p$, (ii) $t|_p$ is the left-most inner-most redex, and (iii) $t_1 = t[rhs_u\theta]_p$. Note that from (ii) and the sufficient completeness of Δ, it follows that (iv) θ is a constructor substitution, that is, for each $x/t \in \theta$, t is a constructor term. From (ii) and (iv), it follows that if $rhs_u\theta$ contains a redex, it is the left-most inner-most redex in t_1 and its position p' belongs to $\mathcal{NVP}os(rhs_u)$. Since rhs_u contains at least one occurrence of the symbols $f_1, \ldots, f_n$, and Δ is sufficiently complete, $rhs_u\theta$ contains at least one redex. Let p' be the position of the left-most inner-most redex inside t_1. Now, consider the following rewrite step $rhs_u\theta \to_{p',E_j} rhs_u[r_j]_{p'}\theta$, $j \in \{1, \ldots, k\}$, which rewrites the redex in position p'. The obtained term t_2 is $t[rhs_u[r_j]_{p'}\theta]_p$. Since during the unfold operation, we perform narrowing at each possible position in rhs_u, the narrowing step $rhs_u \rightsquigarrow_{p',\sigma_j,E_j\cup B} r'_j$ can be proven in $\Delta\cup B$. By (iv) and the completeness of narrowing, the substitution computed by narrowing is more general then θ, which amounts to saying that there exists a substitution ρ such that (v) $\theta =_B \sigma_j\rho$. By the definition of unfolding, the equation $lhs_u\sigma_j = r'_j$ is one E^u_j belonging to Δ'. Finally, from (i) and (v), we can apply the equation E^u_j to term t, thus obtaining $t[r'_j\rho]_p = t[((rhs_u[r_j]_{p'})\sigma_j)\rho]_p =_B t[rhs_u[r_j]_{p'}\theta]_p = t_2$, and the claim follows by applying the inductive hypothesis to the rewrite sequence from t_2 to s.

($\Leftarrow$) We want to prove that, given any ground term t, if $t \to^!_{\Delta',B} s'$, then $t \to^!_{\Delta,B} s$ and $s =_B s'$. We will prove it by induction on the length of the rewriting sequence in Δ'.
(n = 0.) This case is immediate since $t =_B s'$.
(n > 0.) Let us decompose the rewriting sequence from t to s' as follows: $t \to t_1 \to^! s'$. On the rewriting sequence from t_1 to s, we can apply the induction hypothesis, and we now concentrate on the first rewriting step. If t rewrites to t_1 without using one of the equations E^u_j, the same step can be performed in Δ and the claim holds. Otherwise, if one of the equations E^u_j is used for the last rewriting step, there exists a substitution θ such that $(lhs_u\sigma_j)\theta =_B t|_p$, and $t_1 = t[r'_j\theta]_p$. By $rhs_u\sigma_j \to_{E_j} r'_j$ and the stability of rewriting, we have that $(rhs_u\sigma_j)\theta \to_{E_j} r'_j\theta$. Therefore, $t =_B t[lhs_u(\sigma_j\theta)]_p \to_{E^u} t[rhs_u(\sigma_j\theta)]_p = t[(rhs_u\sigma_j)\theta]_p \to_{E_j} t[r'_j\theta]_p = t_1$, which is a rewrite sequence leading to t_1 in Δ.

■

Before stating and proving Theorem VIII.1 let us recall the necessary definition of the antecedent of a position in a term.

Definition VIII.3 *Let $R : l \to r$ be a rule in a given rewrite theory and let $t \to_R t'$ be a rewrite step that reduces a redex at position $p \in \mathcal{P}os(t)$. According to [30], we say that a position $p' \in \mathcal{P}os(t)$ is an* antecedent *of a position $q \in \mathcal{P}os(t')$ iff*
(i) *q is not comparable with p and $q = p'$, or*
(ii) *there exists a position o of a variable x in r such that $q = p.o.w$ and $p' = p.u.w$ where u is a position of x in l.*

Now we are ready to establish that the unfolding transformation followed by the *restoreCompleteness* procedure preserve the semantics of ground normal forms.

Theorem VIII.1 *Let $\mathcal{R} = (\Sigma, \Delta \cup B, R)$ be a program, and let $\mathcal{R}' = (\Sigma, \Delta \cup B, R')$ be the program obtained from $\mathcal{R}$ by unfolding a rule $R^u \in R$ and the* restoreCompleteness *procedure. Then for each term $t \in \mathcal{T}_\Sigma$, $t \to^!_{\mathcal{R}} s$ iff $t \to^!_{\mathcal{R}'} s'$ and $s =_{\Delta \cup B} s'$.*

Proof: Let R^u be a rule of the form $(lhs_u \to rhs_u)$, and $R_1, \ldots, R_k$ be the set of rules used to unfold rule R^u, each one of the form $(l_i \to r_i)$ for $i = 1, \ldots, k$. Let $f_1, \ldots, f_n$ with $n \leq k$ the set of symbols defined by rules $R_1, \ldots, R_k$. Also let $rhs_u \leadsto_{\sigma_j, R\cup\Delta, B} r'_j$, $j \in \{1, \ldots, n\}$, be the $R \cup \Delta, B$-narrowing step such that the result of unfolding R^u using R_j is the rule $R^u_j : (lhs_u\sigma_j \to r'_j)$. From the definition and the correctness of narrowing, we recall that:

(1) $\forall\, j \,.\, rhs_u\sigma_j \to^{R_j} r'_j$

(2) $\forall\, j$ there exists position $p_j \in \mathcal{NVP}os(rhs_u)$ such that $rhs_u|_{p_j}\sigma_j =_B l_j\sigma_j$

(3) $\forall\, j \,.\, r'_j = (rhs_u[r_j]_{p_j})\sigma_j$

($\Rightarrow$) We want to prove that, given any ground term t, if $t \to^!_{\mathcal{R}} s$, then $t \to^!_{\mathcal{R}'} s'$ and $s =_{\Delta \cup B} s'$. We will prove it by induction on the length of the rewrite sequence in $\mathcal{R}$.

(n = 0.) This case is immediate since $t = s$.

(n $>$ 0.) Let us decompose the rewrite sequence from t to s as follows: $t \to t_1 \to^! s$. On the rewrite sequence from t_1 to s, we can apply the induction hypothesis, and we now concentrate on the first rewrite step. If t rewrites to t_1 without using rule R^u, the same step can be performed in $\mathcal{R}'$ and the claim holds. Otherwise, we want to describe a procedure to reorder an initial fragment of the rewrite sequence from t to s in such a way it is then trivial to simulate it in $\mathcal{R}'$ and then use the induction hypothesis on the rest of the sequence.

Consider a ground term w and a subsequent application of rules R^u and R_j in $\mathcal{R}$ as follows. If $w|_p =_B lhs_u\theta$, by applying $(\{R^u\}, \Delta \cup B)$, we obtain a $\Delta \cup B$-equivalent term to $w[rhs_u\theta]_p$, which embeds a ground instance of rhs_u. Therefore, this term contains some occurrences of the symbols $f_1, \ldots, f_n$. Then, if we can apply $(\{R_j\}, \Delta \cup B)$ (for some $j \in \{1, \ldots, k\}$) to reduce the redex having one such symbol as its root, we obtain a $\Delta \cup B$-equivalent term to $w[rhs_u[r_j]_{p_j}\theta]_p$. The key point is to note that this subsequent application of rules R^u and R_j in $\mathcal{R}$ can be simulated in $\mathcal{R}'$ by an application of rule R^u_j. In fact, since the rewrite step using R_j occurs at position $p_j \in \mathcal{NVP}os(rhs_u)$, it follows that the left hand side l_j of rule R_j unifies with the subterm $rhs_u|_{p_j}$ by substitution σ_j, which subsumes θ by Proposition VIII.1, taking $rhs_u|_{p_j}$ as t_1 and l_j as t_2. Therefore, the narrowing step $rhs_u \leadsto_{\sigma_j, p_j, (R_j \cup \Delta, B)} r'_j$ can be proved in $R, \Delta \cup B$. By the definition of unfolding, the rule $lhs_u\sigma_j \to r'_j$ is one R^u_j belonging to $\mathcal{R}'$. Finally, by applying (R^u_j, Δ) to term w we obtain a $\Delta \cup B$-equivalent term to $w[r'_j\rho]_p = w[((rhs_u[r_j]_{p_j})\sigma_j)\rho]_p = w[rhs_u[r_j]_{p_j}\theta]_p$.

The basic aim of the sequence reordering procedure $reorderSeq$, whose pseudo-code is shown in Figure 2, is to change the rule application order, thus obtaining an equivalent sequence (in the sense that the same normal form s is reached) where the application of rule R^u is immediately followed by an application of a rule R_j. In the procedure, a rewrite sequence is represented as a list of rewrite steps (R, p) where R is the applied rule and p the position of the reduced redex. Each rewrite step is intended to be followed by a Δ, B normalization. The procedure takes the rewrite sequence starting from the rewrite step using rule R^u as input and returns the reordered rewrite sequence. List s_1 contains the reordered portion of the sequence, which can be easily simulated in $\mathcal{R}'$, while s_2 contains the rest of the sequence (if any). The auxiliary procedure $reorder$ uses two auxiliary lists ns and vs. The former contains the sequence of steps that are moved before (R^u, p), while the latter contains the skipped steps during the reordering that will keep the same position in the final rewrite sequence. The final sequence is made up of the ns list, the consecutive steps (R^u, p), (R_j, p_j), the skipped steps in vs, and the rest of the sequence in ts. There is only one particular case in which the reordering procedure deletes some rewrite steps including the one using rule R^u, which will be discussed later. Let us explain the eight different cases of the ordering procedure in the $reorder$ function.

Case (1) is the easiest one because the applied rule is one R_j, which is used to reduce a redex in $rhs_u\theta$ having one symbol f_i at its root. In this case, the procedure terminates, returning the reordered sequence $ns, (R^u, p), (R_j, p_j), vs, ts$. In case (3), a rule that is different from R^u is used to reduce a redex in the substitution θ. Since the redex belongs to the substitution, this rewrite step is possible before the application of rule R^u at a position q', which is the antecedent of q. Hence, the rewrite step (R, q') is moved at the end of the ns list and the procedure follows with the rest of the sequence. Case (8) is analogous because a rule that is different from R^u is used to reduce a redex that contains the subterm $rhs_u\theta$ in the substitution without erasing it. This rewrite step can also be moved before the application of rule R^u, and, hence, it is put at the end of the ns list. Note that in this case, the antecedent of q is q itself because $q < p$. Case (4) considers a rule that reduces a redex whose root is not in $rhs_u\theta$ nor in a path from p to the term root. This is the case of a skippable rewrite step that is moved at the end of the vs list. Case (6) considers a rewrite step where the reduced redex contains term $rhs_u\theta$ in the substitution but erases it from the term (i.e., the variable that matches the subterm containing $rhs_u\theta$ does not occur in the rhs of the rule). This rule application makes all the rewrite steps stored in ns and the one using R^u useless, so they can be deleted from the sequence

$reorderSeq((R^u, p) : seq) =$
let $(s_1, s_2) = reorder((R^u, p), [\,], [\,], seq)$ **in** $merge(s_1, s_2)$
$reorder((R^u, p), ns, vs, (R, q) : ts) =$
case :
$q = p_j \in \mathcal{NVP}os(rhs_u)$ and $R = R_j \Rightarrow$
return $([ns, (R^u, p), (R_j, p_j)], [vs, ts])$ (1)
$q \notin \mathcal{NVP}os(rhs_u) > p$ and $R = R^u \Rightarrow$
let $(ns_1, ts_1) = reorder((R^u, q), [\,], [\,], ts)$
in $reorder((R^u, p), [ns, ns_1], vs, ts_1)$ (2)
$q \notin \mathcal{NVP}os(rhs_u) > p \Rightarrow$
$reorder((R^u, p), [ns, (R, q')], vs, ts)$ (3)
$q \not\leq p$ and $q \not> p \Rightarrow reorder((R^u, p), ns, [vs, (R, q)], ts)$ (4)
$q < p$ and $rhs_u|_\Lambda$ occurs in the lhs of R at non root position $\Rightarrow$
return $([ns, (R^u, p), (R, q)], [vs, ts])$ (5)
$q < p$ and $f_1, \ldots, f_n$ do not appear in the resulting term $\Rightarrow$
return $([(R, q)], [vs, ts])$ (6)
$q < p$ and $R = R^u \Rightarrow$
let $(ns_1, ts_1) = reorder((R^u, p), ns, vs, ts)$
in $reorder((R^u, q), [ns, ns_1], vs, ts_1)$ (7)
$q < p \Rightarrow reorder((R^u, p), [ns, (R, q)], vs, ts)$ (8)
where $q' =$ antecedent of q

Fig. 2. Rewrite sequence reordering procedure.

and the procedure terminates returning the step (R, q), the skipped steps, and the rest of the sequence. Case (5) considers a rewrite step where the left-hand side of the applied rule R matches the root symbol of rhs_u at some position p. This means that $rhs_u\theta$ is not contained in the matching substitution. We are then in the hypothesis of Lemma VIII.1, which states that the two subsequent rewrite steps (R^u, p) and (R, q) can be simulated in $\mathcal{R}'$. Hence, the procedure terminates returning the reordered sequence $ns, (R^u, p), (R, q), vs, ts$. Cases (2) and (7) consider a rewrite step where the same rule R^u is used to reduce a redex that is inside θ or that contains the subterm $rhs_u\theta$, respectively. The basic idea is that when another application of R^u is found, we first terminate the reordering w.r.t. the deeper application of R^u and then we recursively call the $reorder$ function to reorder the sequence w.r.t. the R^u application that is not as deep. In fact, in case (2), we suspend the reordering procedure w.r.t. the considered application of rule R^u, and we recursively call the function to reorder a fragment of the rewrite sequence w.r.t. the deeper R^u application. When the recursive call terminates, we resume the previous call putting the computed list ns_1 at the end of the ns list and following with the computed rest of the sequence ts_1. Case (7) does the reverse, by terminating the current reordering and then recursively calling the function w.r.t. the R^u application that is not as deep.

Termination. Since we consider programs to be sufficiently complete and the considered rewrite sequence ends with the normal form s, the occurrences of symbols $f_1, \ldots, f_n$ have to be reduced before reaching s by using either a rule R_j as considered in case (1), or a rule that makes them disappear as considered in case (6). In both cases the $reorder$ procedure terminates.

Correctness. We want to show that all the rewrite steps contained in list s_1 (which is then merged with the rest of the sequence s_2 in function $reorderSeq$) can be trivially simulated in $\mathcal{R}'$. List s_1 is the first component of the pair of lists returned by the $reorder$ function. Considering the termination cases, the first component can contain either the step (R, q) (case (6)) where R is different from R^u, or the list $ns, (R^u, p), (R_j, p_j)$ (case (1)), or the list $ns, (R^u, p), (R, q)$ (case (5)). When we apply a rule that is different from R^u it can be trivially simulated in $\mathcal{R}'$ by applying the same rule. Moreover, recall that a subsequent application of rules R^u and R_j can be simulated in $\mathcal{R}'$ by an application of rule R^u_j. Finally, the subsequent steps $(R^u, p), (R, q)$ considered by case (5) can be simulated in $\mathcal{R}'$ by Lemma VIII.1. Hence, the correctness holds.

Reduction of the sequence. It is easy to see that s_1 is never empty and the rest of the sequence s_2 is strictly shorter that the sequence from t_1 to s. Hence, we can use the inductive hypothesis on s_2.

($\Leftarrow$) We want to prove that, given any ground term t, if $t \to^!_{\mathcal{R}'} s'$, then $t \to^!_{\mathcal{R}} s$, and $s =_{\Delta \cup B} s'$. We will prove it by induction on the length of the rewriting sequence in $\mathcal{R}'$.

(n = 0.) This case is immediate since $t = s'$.

(n > 0.) Let us decompose the rewriting sequence form t to s' as follows: $t \to t_1 \to^! s'$. On the rewriting sequence from t_1 to s, we can apply the induction hypothesis, and we now concentrate on the first rewriting step. If t rewrites to t_1 without using one of the rules R^u_j, the same step can be performed in $\mathcal{R}$ and the claim holds. Otherwise, if one of the rules R^u_j is used for the last rewriting step, there exists a substitution θ such that $(lhs_u\sigma_j)\theta =_B t|_p$, and $t_1 = t[r'_j\theta]_p$. By $rhs_u\sigma_j \to^{R_j} r'_j$ and the stability of rewriting, we have that $(rhs_u\sigma_j)\theta \to^{R_j} r'_j\theta$. Therefore, $t =_B t[lhs_u(\sigma_j\theta)]_p \to^R t[rhs_u(\sigma_j\theta)]_p = t_1[(rhs_u\sigma_j)\theta]_p \to^{R_j} t[r'_j\theta]_p = t_1$, which is a rewriting sequence leading to t_1 in $\mathcal{R}$.

■

Finally, the main result of the paper immediately follows from the previous Theorem.

Proof of Theorem V.1: The (Corr.) part of the proof is perfectly equivalent to the ($\Leftarrow$) part of the proof of the Theorem VIII.1. For the (Comp.) part, note that since the program is weakly normalizing, if $t \to^*_{\mathcal{R}} s$, there exists at least a normal form s'' such that $s \to^*_{\mathcal{R}} s''$, and for Theorem VIII.1 $t \to^*_{\mathcal{R}'} s'$ with $s' =_{\Delta \cup B} s''$.

■

Remark. In order to prove Theorem V.1, stating that the unfolding operation and the *restoreCompleteness* procedure preserve the semantics of ground reducts of the original program, we had to prove that they preserve the semantics of ground normal forms (Theorem VIII.1). The reader may think that Theorem V.1 is just a trivial extension to the semantics of ground reducts, which is mainly based on Theorem VIII.1 and that we actually preserve only the semantics of ground normal forms. The fact is that there are cases of rewrite sequences starting from a ground term t where an unfolded symbol is not reduced until a normal form is reached, and since in the transformed program that symbol has been evaluated in advance in the unfolded rules, a rewrite sequence in the transformed program starting from t cannot reach a reduct equivalent to one in the rewrite sequence in the original program until the normal form. However, this is not the general case, as shown in the following example.

Example VIII.1 *Let us consider the rules of Example IV.3, and let us recall that the* restoreCompleteness *procedure has extended the set of rules* R' *with rules* $16.\ \ h(f(w,z),y) \rightarrow p(g_1(w,z),y)$ *and* $17.\ \ k(f(w,z)) \rightarrow 1$. *Consider the following rewrite sequence in* $\mathcal{R}$*:* $h(f(g_2(0),g_1(1,0)),1) \rightarrow_{11} h(g_2(g_1(g_2(0),g_1(1,0))),1) \rightarrow_6 p(g_1(g_2(0),g_1(1,0)),1)$. *The same ground reduct can be reached in* $\mathcal{R}'$ *by a rewrite step using rule* $16.$*:* $h(f(g_2(0),g_1(1,0)),1) \rightarrow_{16} p(g_1(g_2(0),g_1(1,0)),1)$. *Consider also the following rewrite sequence in* $\mathcal{R}$*:* $f(g_2(0),g_1(1,0)) \rightarrow_{11} g_2(g_1(g_2(0),g_1(1,0))) \rightarrow_4 g_1(g_2(0),g_1(1,0))$. *The same ground reduct can be reached in the transformed program by a rewrite step using the unfolded rule* $15.$*:* $f(g_2(0),g_1(1,0)) \rightarrow_{15} g_1(g_2(0),g_1(1,0))$.

In other words, we do not lose generality by considering rewriting up to normal form in our proof.