Correctness of Logic Program Transformations Based on Existential Termination*

Kung-Kiu Lau
Department of Computer Science, University of Manchester,
Oxford Road, Manchester M13 9PL, United Kingdom. kung-kiu@cs.man.ac.uk

Mario Ornaghi
Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano,
Via Comelico 39/41, Milano, Italy. ornaghi@hermes.mc.dsi.unimi.it

Alberto Pettorossi
Department of Electronic Engineering, University of Roma Tor Vergata,
Via della Ricerca Scientifica, 00133 Roma, Italy. adp@iasi.rm.cnr.it

Maurizio Proietti
IASI-CNR, Viale Manzoni 30, 00185 Roma, Italy. proietti@iasi.rm.cnr.it

Abstract

We study the relationships between the correctness of logic program transformation and program termination. We consider definite programs and we identify some 'invariants' of the program transformation process. The validity of these invariants ensures the preservation of the success set semantics, provided that the existential termination of the initial program implies the existential termination of the final program. We also identify invariants for the preservation of the finite failure set semantics.

We consider four very general transformation rules: definition introduction, definition elimination, iff-replacement, and finite failure. Many versions of the transformation rules proposed in the literature, including unfolding, folding, and goal replacement, are instances of the iff-replacement rule.

By using our proposed invariants which are based on Clark completion, we prove, for our transformation rules, various results concerning the preservation of both the success set and finite failure set semantics. By exploiting some powerful properties of the Clark completion, we are able to derive simple proofs of these preservation results. These proofs are much simpler than those done by induction on the construction of the SLD-trees, like the ones proposed in the literature for related results.

*A shorter version of this paper appears in: J. W. Lloyd (ed.) Proceedings of the International Logic Programming Symposium, Portland, Oregon, USA, December 4–7, 1995, pp. 480–494, MIT Press.

1 Introduction

When we derive a new program from an initial program by transformation, we need to keep the semantics unchanged.

A very simple method to achieve this objective, in the case of recursive equation programs with the least fixpoint semantics, was proposed in Burstall and Darlington's pioneering paper [6]. This method for recursive equation programs can be described as follows. A program P_1 is transformed into a program P_2 by using transformation rules which preserve *partial correctness*, in the sense that the partial function denoted by P_2 is contained in the partial function denoted by P_1 (when viewing partial functions as sets of pairs). If, moreover, P_2 is *defined* for all inputs for which P_1 is defined, then the transformation is *totally correct*, in the sense that the functions denoted by P_1 and P_2 are equal. Thus, in order to prove the total correctness of the transformation from P_1 to P_2, it is enough to show that P_2 terminates for every input for which P_1 terminates.

Unfortunately, such a method for showing total correctness does not work in a straightforward way in the case of definite logic programs (which we will simply call 'logic programs' or 'programs') with the least Herbrand model semantics. Indeed, let us consider two logic programs P_1 and P_2 such that P_1 is transformed into P_2 using transformation rules which preserve partial correctness, in the sense that the least Herbrand model of P_2 is contained in the least Herbrand model of P_1. Let us also consider the notion of *universal termination* [9], which says that a program P universally terminates wrt a given goal g iff all SLD-derivations of $P \cup \{\neg g\}$ are finite. Now, even if program P_2 universally terminates wrt any goal g whenever P_1 does, we cannot conclude that P_1 and P_2 have the same least Herbrand model semantics. To see this, take, for instance, $P_1 = \{p(a) \leftarrow,\ \ p(b) \leftarrow\}$ and $P_2 = \{p(a) \leftarrow\}$.

This difficulty may explain why in early papers on logic program transformation, such as those by Hogger [12] and Tamaki and Sato [20], the problem of verifying total correctness is not based on termination properties. In particular, in [12] total correctness is proved by first considering transformation rules which preserve partial correctness and then proving the inclusion of the least Herbrand model of P_1 in the least Herbrand model of P_2.

In order to ensure total correctness, Tamaki and Sato [20] adapt Burstall and Darlington's unfolding and folding transformation rules to logic programming by imposing some restrictions on the use of the rules, in particular, on the use of the folding rule. However, these restrictions limit, in practice, the use of the transformation methodology, especially when the unfolding and folding rules are combined with more complex transformations, such as the goal replacement rule.

Along the lines of Tamaki and Sato's work, many other researchers have proposed restrictions on the use of the transformation rules for making sure that they preserve various semantics of logic programs, such as the computed answer substitutions and the finite failure set. In some cases those restrictions are explicitly based on termination properties [4, 5, 8]. A similar approach has also been extended to general logic programs under various semantics (see [18] for a detailed discussion).

In contrast, in this paper we propose a more flexible approach, which has been inspired by some of the ideas presented in [13] in the related field of logic program synthesis. In our approach, during the process of deriving new programs from old programs by transformation, one is allowed to apply transformation rules which guarantee that the following invariant is maintained: 'the least Herbrand model of the initial program together with the definitions introduced so far, is a model of the completion of the current program'. The validity of this invariant implies

that, if from program P_1 one derives program P_2, then an element of the the least Herbrand model of P_1 belongs to either the least Herbrand model of P_2 or the infinite failure set of P_2. Then, in order to show that the transformation is totally correct w.r.t. the least Herbrand model semantics one is only required to prove that the *existential termination* (see Definition 2 below) of the initial program implies the existential termination of the derived program. Notice that universal termination implies existential termination, but not vice versa.

A different approach to the adaptation of Burstall and Darlington's technique for preserving total correctness when transforming logic programs, is to assume that the semantics is a partial function from the Herbrand base to $\{true, false\}$. However, this is equivalent to the introduction of 3-valued models, while in this paper we want to stick to 2-valued models. For results dealing with the preservation of 3-valued models during transformation, the reader may see, for instance, [19].

The structure of the paper is as follows. In Section 2 we consider three basic transformation rules: definition introduction, definition elimination, and clause replacement. Then, in Section 3 we show that, in general, these rules preserve partial correctness wrt the least Herbrand model semantics. In Section 4 we consider the iff-replacement and finite failure rules, which are instances of the clause replacement rule, and we prove a total correctness result based on the preservation of the invariant we have mentioned above, and the notion of existential termination. In Section 5 we examine the relationship between various existential termination properties and the preservation of the finite failure set during program transformation. To this aim we also consider rules which ensure the validity of a stronger invariant, that is, the fact that the completion of the initial program together with the definitions introduced so far, implies the completion of the current program. In Section 6 we show that the iff-replacement rule is a generalization of the unfolding, folding, and goal replacement rules introduced in [20], thus, the correctness results proved for the iff-replacement rule can be applied to the unfolding, folding, and goal replacement rules. Finally, in Section 7 we compare the results presented in this paper with related ones in the literature.

2 The Transformation Process

In this section we introduce three basic transformation rules, together with the terminology we will use. Our three rules are called *definition introduction*, *definition elimination*, and *clause replacement*. The rules introduced in [20] can all be viewed as instances of our rules. Also the iff-replacement and finite failure rules considered below, are instances of the clause replacement rule.

Throughout this paper we shall consider definite logic programs, and we shall assume that all programs are constructed using a subset of a fixed first-order language $\mathcal{L}$ with finitely many predicate and function symbols.

By $\mathcal{M}(P)$ we denote the least Herbrand interpretation of $\mathcal{L}$ which is a model of the program P (that is, the least Herbrand model of P). Thus, we make it explicit that all atoms with predicates not occurring in the head of any clause of P are false in $\mathcal{M}(P)$.

$Comp(P)$ is the completion of P constructed as in [14]. *iff* (P) is the formula constructed from program P exactly as $Comp(P)$, but: (i) without adding $\forall X.\neg p(X)$ for every predicate p occurring in the body of a clause of P and not occurring in the head of any clause of P, and (ii) without adding the formulas of Clark Equality Theory.

By $hd(C)$ and $bd(C)$ we denote the head and body of clause C, respectively. A *goal* is a conjunction of atoms.

The *definition* of a predicate p in a program P is the set of clauses of P in whose head p occurs. A predicate is said to be *defined* in a program P iff it occurs in the head of at least one clause of P.

The program transformation process starting from a given initial program P_0 is formalized as a sequence of programs $P_0, \ldots, P_k$, called *transformation sequence*, such that program P_{j+1}, with $0 \leq j \leq k-1$, is obtained from program P_j by the application of one of the transformation rules.

We assume that, when transforming an initial program P_0, we are interested in preserving the semantics of P_0 only wrt a given set $\mathcal{G}$ of ground goals, called *pertinent goals*, such that all predicate symbols occurring in $\mathcal{G}$ are defined in P_0. (This notion of pertinent goals will be used in Section 3 when defining the correctness of a transformation sequence.)

In what follows we need the following concepts.

We say that a predicate p *immediately depends* on a predicate q in a program P iff there exists in P a clause of the form $p(\ldots) \leftarrow B$ such that q occurs in the goal B. We say that p *depends* on q in P iff either p immediately depends on q in P or p immediately depends on a predicate r in P and r depends on q in P. We say that a formula φ *requires* a predicate q in P iff either q occurs in φ or there exists a predicate occurring in φ which depends on q in P.

The *relevant part* $Rel(P, \varphi)$ of a program P for a closed formula φ is the set of definitions of all predicates in P which are required by φ.

$vars(t)$ denotes the set of variables in the term (or atom, or goal, or clause) t.

Now we define our three basic transformation rules.

Definition Introduction Rule. Let $P_0, \ldots, P_j$ be a transformation sequence. We may get program P_{j+1} from program P_j by adding n definite clauses of the form $newp_i(t_1, \ldots, t_m) \leftarrow B_i$, for $i = 1, \ldots, n$, such that the predicate symbol $newp_i$ does not occur in any program of the sequence $P_0, \ldots, P_j$ and every predicate in B_i different from $newp_1 \ldots, newp_n$ is defined in P_j.

This rule is similar to the definition rule given in [15] and it allows us to simultaneously introduce one or more new predicate definitions which possibly consist of mutually recursive clauses. Notice that each new predicate may be defined by more than one clause, that is, $newp_h$ may be equal to $newp_i$ for some h and i in $\{1, \ldots, n\}$ such that $h \neq i$.

Given the transformation sequence $P_0, \ldots, P_j$, its associated *extended initial program* is the set of clauses $P_0 \cup Def_j$, where Def_j is the set of all clauses introduced by the definition introduction rule during the construction of the *entire* transformation sequence $P_0, \ldots, P_j$.

Definition Elimination Rule. Let $P_0, \ldots, P_j$ be a transformation sequence and $\mathcal{G}$ be the set of pertinent goals. We may get program P_{j+1} from program P_j by deleting the definitions of predicates $q_1, \ldots, q_n$ such that, for every $k = 1, \ldots, n$: (i) no goal in $\mathcal{G}$ requires q_k in P_j, and (ii) every predicate in P_j which depends on q_k, is in the set $\{q_1, \ldots, q_n\}$.

A rule similar to definition elimination is presented in [15], where it is called *deletion*, and also in [3], where it is called *restricting operation*.

Clause Replacement Rule. Let $P_0, \ldots, P_j$ be a transformation sequence and $P_0 \cup Def_j$ be the extended initial program associated with $P_0, \ldots, P_j$. Let $C_1, \ldots, C_m$,

with $m \geq 0$, be the definition of a predicate p in P_j, and $D_1, \ldots, D_n$, with $n \geq 0$, be clauses such that p occurs in all their heads and

$$\mathcal{M}(P_0 \cup Def_j) \models (C_1, \ldots, C_m) \rightarrow (D_1, \ldots, D_n) \tag{1}$$

where, as usual, every clause is assumed to be a universal formula.

We get the new program P_{j+1} from P_j by replacing clauses $C_1, \ldots, C_m$ by clauses $D_1, \ldots, D_n$.

Notice that the model-theoretic Condition (1) in the clause replacement rule is implied by a proof-theoretic condition of the form

$$\mathcal{F} \vdash (C_1, \ldots, C_m) \rightarrow (D_1, \ldots, D_n) \tag{2}$$

where $\mathcal{F}$ is a set of closed formulas which are true in $\mathcal{M}(P_0 \cup Def_j)$.

Condition (2), since it is proof-theoretical, may be preferable to Condition (1) for the construction of an automatic transformation system. The formula $\mathcal{F}$ may include $Comp(P_0 \cup Def_j)$, or Clark Equality Theory [7], or the Domain Closure Axiom [16]. $\mathcal{F}$ may also include any formula taken from the recursively enumerable set of formulas which express the first-order induction principle over the Herbrand universe of $\mathcal{L}$.

3 Partial Correctness

We now introduce the notions of *partial correctness* and *total correctness* of a transformation sequence. These notions are based on the preservation of the least Herbrand model of the initial program. In Section 5 we will discuss a notion of correctness which is based on the preservation of the finite failure set.

Definition 1 (Correctness of a Transformation Sequence) Given a transformation sequence $P_0, \ldots, P_k$ and a set $\mathcal{G}$ of pertinent goals, we say that $P_0, \ldots, P_k$ is *partially correct* wrt $\mathcal{G}$ iff for every goal $g \in \mathcal{G}$

$$\mathcal{M}(P_k) \models g \quad \textit{implies} \quad \mathcal{M}(P_0) \models g \tag{3}$$

We say that the transformation sequence $P_0, \ldots, P_k$ is *totally correct* wrt $\mathcal{G}$ iff for every goal $g \in \mathcal{G}$

$$\mathcal{M}(P_k) \models g \quad \textit{if and only if} \quad \mathcal{M}(P_0) \models g \tag{4}$$

In order to prove the partial correctness of a transformation sequence constructed by using our rules, we need the following two lemmas.

Lemma 1 (Relevance of the Least Herbrand Model Semantics) *Given any closed formula φ and definite program P,*

$$\mathcal{M}(P) \models \varphi \quad \textit{if and only if} \quad \mathcal{M}(Rel(P, \varphi)) \models \varphi$$

Proof. By induction on the structure of φ. □

Lemma 2 *Let $P_0, \ldots, P_k$ be a transformation sequence and $\mathcal{G}$ be a set of pertinent goals. If $\mathcal{M}(P_0 \cup Def_k) \models P_k$, then the sequence $P_0, \ldots, P_k$ is partially correct wrt $\mathcal{G}$.*

Proof. Let g be a goal in $\mathcal{G}$ such that $\mathcal{M}(P_k) \models g$. By van Emden and Kowalski's result (see Theorem 6.2 in [14]), we have that $P_k \models g$ and therefore, by hypothesis, we get $\mathcal{M}(P_0 \cup Def_k) \models g$. Now, since g does not require predicates defined in Def_k, by Lemma 1 we have that $\mathcal{M}(P_0) \models g$. □

Theorem 1 (Partial Correctness) *Let $P_0, \ldots, P_k$ be a transformation sequence constructed using the definition introduction, definition elimination, and clause replacement rules, and let $\mathcal{G}$ be the set of pertinent goals. Then $P_0, \ldots, P_k$ is partially correct wrt $\mathcal{G}$.*

Proof. By Lemma 2, it is enough to show that $\mathcal{M}(P_0 \cup Def_j) \models P_j$, for $j = 0, \ldots, k$. The proof is by induction on j. The base case ($j = 0$) is obvious, because Def_0 is empty. For the step case we assume that $\mathcal{M}(P_0 \cup Def_j) \models P_j$ for an arbitrary $j \geq 0$, and we prove $\mathcal{M}(P_0 \cup Def_{j+1}) \models P_{j+1}$. We have the following three cases: α, β, and γ.
(α) *Definition Introduction Rule.* Let P_{j+1} be $P_j \cup Newp$, where *Newp* is the set of clauses defining the new predicates introduced at step $j+1$. Thus, $Def_{j+1} = Def_j \cup Newp$. Since P_j does not require any predicate defined in *Newp*, by the relevance of the least Herbrand model semantics (see Lemma 1) and the inductive hypothesis, we have that $\mathcal{M}(P_0 \cup Def_{j+1}) \models P_j$. Moreover, $\mathcal{M}(P_0 \cup Def_{j+1}) \models Newp$, because $Newp \subseteq Def_{j+1}$, and therefore we have that $\mathcal{M}(P_0 \cup Def_{j+1}) \models P_j \cup Newp$.
(β) *Definition Elimination Rule.* The thesis follows from the inductive hypothesis and the fact that a model of a set of formulas is also a model of each of its subsets.
(γ) *Clause Replacement Rule.* The thesis follows from the inductive hypothesis and Condition (1) of the clause replacement rule. □

However, as the following example shows, by using the clause replacement rule, we may construct transformation sequences which are not totally correct.

Example 1 Let us consider the program $P_0 = \{p \leftarrow q, \quad p \leftarrow r, \quad q \leftarrow \}$. Since $\mathcal{M}(P_0) \models (p \leftarrow q,\ p \leftarrow r) \rightarrow (p \leftarrow r)$, by applying the clause replacement rule we may get the following program $P_1 = \{p \leftarrow r, \quad q \leftarrow \}$. We have that $\mathcal{M}(P_0) \models p$, but $\mathcal{M}(P_1) \not\models p$.

4 Total Correctness

In this section we establish a sufficient condition for total correctness, based on the completion of the final program and the notion of *existential termination* [9], which is defined as follows.

Definition 2 (Existential Termination) A definite program P *existentially terminates* wrt a goal g iff there exists an SLD-tree for $P \cup \{\neg g\}$ which *either* has at least one success branch *or* it is finitely failed.

Given a program P and a goal g, if there exists an SLD-tree with at least one success branch for $P \cup \{\neg g\}$, then we will also say that P *succeeds* wrt g, while in the case where there exists a finitely failed SLD-tree for $P \cup \{\neg g\}$ we will also say that P *finitely fails* wrt g.

Example 1 shows a transformation sequence constructed by using the clause replacement rule such that its final program existentially terminates wrt all goals, and yet the transformation sequence is not totally correct. To get total correctness we have to weaken the clause replacement rule. Thus, we introduce two instances

of that rule, called *iff-replacement* and *finite failure* transformation rules. In Theorem 2 below we will show that a transformation sequence constructed by using the definition introduction, definition elimination, iff-replacement, and finite failure rules, is totally correct if its final program existentially terminates wrt any pertinent goal g whenever the initial program existentially terminates wrt g.

Iff-Replacement Rule. Let $P_0, \ldots, P_j$ be a transformation sequence and $P_0 \cup Def_j$ be the extended initial program associated with $P_0, \ldots, P_j$. Let $C_1, \ldots, C_m$, with $m \geq 1$, be the definition of a predicate p in P_j, and $D_1, \ldots, D_n$, with $n \geq 1$, be clauses such that predicate p occurs in all their heads. Suppose that all predicates occurring in $D_1, \ldots, D_n$ occur in P_j and suppose also that the following condition holds:

$$\mathcal{M}(P_0 \cup Def_j) \models \mathit{iff}\,(C_1, \ldots, C_m) \rightarrow \mathit{iff}\,(D_1, \ldots, D_n) \tag{5}$$

Then we get the new program P_{j+1} from P_j by replacing clauses $C_1, \ldots, C_m$ by clauses $D_1, \ldots, D_n$.

As in the case of the clause replacement rule, the model-theoretic Condition (5) is implied by a proof-theoretic condition of the form

$$\mathcal{F} \vdash \mathit{iff}\,(C_1, \ldots, C_m) \rightarrow \mathit{iff}\,(D_1, \ldots, D_n) \tag{6}$$

where $\mathcal{F}$ is any set of closed formulas which are true in $\mathcal{M}(P_0 \cup Def_j)$.

Finite Failure Rule. Let $P_0, \ldots, P_j$ be a transformation sequence, $P_0 \cup Def_j$ be the extended initial program associated with $P_0, \ldots, P_j$, and C be a clause in P_j such that $P_0 \cup Def_j$ finitely fails wrt $bd(C)$. We get the new program P_{j+1} by deleting C from P_j.

Notice that the finite failure rule cannot be considered to be an instance of the iff-replacement rule, because by using the iff-replacement rule the set of predicates occurring in the program may not be changed.

In order to prove our total correctness result of Theorem 2 below, we need the following two lemmas.

Lemma 3 *Let $P_0, \ldots, P_k$ be a transformation sequence constructed by using the definition introduction, definition elimination, iff-replacement, and finite failure rules. Then the following property holds:*

$$\mathcal{M}(P_0 \cup Def_k) \models Comp(P_k) \tag{7}$$

Proof. It is by induction on the length k of the transformation sequence $P_0, \ldots, P_k$. The base case ($k = 0$) is obvious, because Def_0 is empty and the least Herbrand model of program P_0 is a model of the completion of P_0. For the step case, we assume that $\mathcal{M}(P_0 \cup Def_k) \models Comp(P_k)$ for $k \geq 0$, and we prove $\mathcal{M}(P_0 \cup Def_{k+1}) \models Comp(P_{k+1})$. We have the following four cases.

- *Definition Introduction Rule.* Let P_{k+1} be $P_k \cup Newp$, where $Newp$ is the set of clauses defining the new predicates introduced at step $k+1$. Thus, $Def_{k+1} = Def_k \cup Newp$.

  All predicates occurring in the body of a clause in $Newp$ are defined in P_k, and hence in P_{k+1}. Therefore, $Comp(P_{k+1}) = Comp(P_k) \cup \mathit{iff}\,(Newp)$, because no formula of the form $\forall X.\neg p(X)$ is added to $Comp(P_k)$ to get $Comp(P_{k+1})$.

Since $Comp(P_k)$ does not require any predicate symbol in *Newp*, by the relevance of the least Herbrand model semantics (see Lemma 1) and by inductive hypothesis, we have that $\mathcal{M}(P_0 \cup Def_{k+1}) \models Comp(P_k)$. Moreover, $\mathcal{M}(P_0 \cup Def_{k+1}) \models \mathit{iff}\,(Newp)$, because $Newp \subseteq Def_{k+1}$, and therefore $\mathcal{M}(P_0 \cup Def_{k+1}) \models Comp(P_{k+1})$.

- *Definition Elimination Rule.* Our thesis follows from the inductive hypothesis and the fact that $Comp(P_{k+1}) \subseteq Comp(P_k)$.

- *Iff-Replacement Rule.* All predicates occurring in the body of a clause in $D_1, \ldots, D_n$ occur in P_k. Therefore, $Comp(P_{k+1}) = (Comp(P_k) - \mathit{iff}\,(C_1, \ldots, C_m)) \cup \mathit{iff}\,(D_1, \ldots, D_n)$, because no formula of the form $\forall X.\neg p(X)$ is added to $Comp(P_k)$ to get $Comp(P_{k+1})$. Then the thesis follows from the inductive hypothesis and Condition (5) of the iff-replacement rule.

- *Finite Failure Rule.* We have that $\mathcal{M}(P_0 \cup Def_k) \models \forall X.\neg bd(C)$, where X is the tuple of variables occurring in $bd(C)$. If $hd(C)$ is of the form $p(t)$ and in P_k there is at least one more clause whose head predicate is p, then the finite failure rule is an instance of the iff-replacement rule. Otherwise, by inductive hypothesis, $\mathcal{M}(P_0 \cup Def_k) \models \forall Y.\neg p(Y)$, because $bd(C)$ finitely fails. Now there are two cases. (i) If p occurs in the body of at least one clause in P_{k+1} then $Comp(P_{k+1}) = (Comp(P_k) - \mathit{iff}\,(C)) \cup \{\forall Y.\neg p(Y)\}$. (ii) If p does not occur in P_{k+1} then $Comp(P_{k+1}) = Comp(P_k) - \mathit{iff}\,(C)$.

  In both cases (i) and (ii), we get the thesis, because $\mathcal{M}(P_0 \cup Def_{k+1}) = \mathcal{M}(P_0 \cup Def_k)$.

□

Lemma 4 *Let $P_0, \ldots, P_k$ be a transformation sequence constructed by using the definition introduction, definition elimination, iff-replacement, and finite failure rules. Let $\mathcal{G}$ be the set of pertinent goals. Then, for every $g \in \mathcal{G}$, either $\mathcal{M}(P_0 \cup Def_k) \models \neg g$ or all predicates occurring in g are defined in P_k.*

Proof. It is by induction on the length k of the transformation sequence $P_0, \ldots, P_k$. The base case ($k = 0$) is obvious because, by definition of pertinent goals, all predicates occurring in $g \in \mathcal{G}$ are defined in P_0. For the step case, we assume that, for an arbitrary $k \geq 0$ and a goal $g \in \mathcal{G}$, either (Case α) $\mathcal{M}(P_0 \cup Def_k) \models \neg g$ or (Case β) all predicates occurring in g are defined in P_k. We will show that either $\mathcal{M}(P_0 \cup Def_{k+1}) \models \neg g$ or all predicates occurring in g are defined in P_{k+1}. We have the following four cases.

- *Definition Introduction Rule.* Let P_{k+1} be $P_k \cup Newp$, where *Newp* is the set of clauses defining the new predicates introduced at step $k+1$. Thus, $Def_{k+1} = Def_k \cup Newp$.

  In Case (α) we have that $\mathcal{M}(P_0 \cup Def_{k+1}) \models \neg g$, because g does not require any predicate defined in *Newp* and the least Herbrand model semantics is relevant (see Lemma 1).

  In Case (β) all predicates occurring in g are defined in P_{k+1}, which properly contains P_k.

- *Definition Elimination Rule.* In Case (α) we have that $\mathcal{M}(P_0 \cup Def_{k+1}) \models \neg g$, because $Def_k) = Def_{k+1}$).

Consider now Case (β) and let p be a predicate occurring in g. Then p is defined in P_k. p is also defined in P_{k+1}, because clauses defining predicates which are required by pertinent goals cannot be discarded by definition elimination.

- *Iff-Replacement Rule.* We have the thesis, because by an application of the iff-replacement rule, the set of predicates defined in P_k is equal to the set of predicates defined in P_{k+1} and $Def_k = Def_{k+1}$.

- *Finite Failure Rule.* In Case (α) we have that $\mathcal{M}(P_0 \cup Def_{k+1}) \models \neg g$, because $Def_k) = Def_{k+1}$).

  Let us now consider Case (β). Take a predicate p occurring in g. If p does not occur in $hd(C)$, then p is defined in P_{k+1}. If $hd(C)$ is of the form $p(t)$ and in P_k there is at least one more clause whose head predicate is p, then p is defined in P_{k+1}.

  Otherwise, the definition of p in P_k is clause C only. We also have that $\mathcal{M}(P_0 \cup Def_k) \models \forall X.\neg bd(C)$, where X is the tuple of variables occurring in $bd(C)$. Since by Lemma 3 we have that $\mathcal{M}(P_0 \cup Def_k) \models Comp(P_k)$ and thus, $\mathcal{M}(P_0 \cup Def_k) \models \forall Y.\neg p(Y)$. Therefore, $\mathcal{M}(P_0 \cup Def_k) \models \neg g$, and we get the thesis as in Case (α).

□

Theorem 2 (Total Correctness) *Let $P_0, \ldots, P_k$ be a transformation sequence constructed by using the definition introduction, definition elimination, iff-replacement, and finite failure rules. Let $\mathcal{G}$ be the set of pertinent goals and suppose that, for every $g \in \mathcal{G}$, if P_0 existentially terminates wrt g, then P_k existentially terminates wrt g. Then $P_0, \ldots, P_k$ is totally correct wrt $\mathcal{G}$.*

Proof. Since the iff-replacement and finite failure rules are instances of the clause replacement rule, by Theorem 1 the sequence $P_0, \ldots, P_k$ is partially correct wrt $\mathcal{G}$. Thus, to prove the theorem, it is enough to show that, for every $g \in \mathcal{G}$, if $\mathcal{M}(P_0) \models g$ then $\mathcal{M}(P_k) \models g$.

Let $g \in \mathcal{G}$ be a goal such that $\mathcal{M}(P_0) \models g$. Then, by the completeness of SLD-resolution, P_0 existentially terminates wrt g and therefore, by hypothesis, so does P_k. By Lemma 1 and the fact that pertinent goals do not require any predicate defined in Def_k, we have that $\mathcal{M}(P_0 \cup Def_k) \models g$. Thus, $\mathcal{M}(P_0 \cup Def_k) \not\models \neg g$ and, by Lemma 3, $Comp(P_k) \not\vdash \neg g$. Since $\mathcal{M}(P_0 \cup Def_k) \models g$, by Lemma 4 all predicates occurring in g are defined in P_k and, therefore, by Clark's characterization of $Comp$ [7], $Comp(P_k) \not\vdash \neg g$ is equivalent to saying that P_k does *not* finitely fail wrt g. Since P_k existentially terminates wrt g, $P_k \cup \{\neg g\}$ has an SLD-refutation. By the soundness of SLD-resolution we conclude that $\mathcal{M}(P_k) \models g$. □

5 Preservation of Finite Failure

In this section we establish some conditions based on existential termination which ensure the preservation of the finite failure set when transforming programs. We start off by showing that, in the hypotheses of Theorem 2, finitely failed computations of the initial program wrt pertinent goals are not transformed into non-terminating computations of the derived program.

Theorem 3 *Let $P_0, \ldots, P_k$ be a transformation sequence constructed by using the definition introduction, definition elimination, iff-replacement, and finite failure rules. Let $\mathcal{G}$ be the set of pertinent goals and suppose that, for every $g \in \mathcal{G}$, if P_0 existentially terminates wrt g then P_k existentially terminates wrt g. Then for every $g \in \mathcal{G}$, if P_0 finitely fails wrt g then P_k finitely fails wrt g.*

Proof. Suppose that P_0 finitely fails wrt a goal $g \in \mathcal{G}$. Therefore, by the completeness of SLD-resolution, $\mathcal{M}(P_0) \not\models g$ and hence, by Theorem 2, $\mathcal{M}(P_k) \not\models g$. Thus, by the soundness of SLD-resolution we have that P_k does not succeed wrt g. Since P_0 finitely fails wrt g, we have that P_0 existentially terminates wrt g and, by our assumptions, P_k existentially terminates wrt g. Now, since P_k does not succeed wrt g, we derive that P_k finitely fails wrt g. □

The following example shows that, by using the iff-replacement rule, one may get a program which finitely fails wrt a goal while the initial program does not existentially terminate wrt that goal.

Example 2 Let us consider the initial program $P_0 = \{p \leftarrow p, \quad q \leftarrow r\}$. Since $\mathcal{M}(P_0) \models \neg p$ and $\mathcal{M}(P_0) \models \neg r$, we have that $\mathcal{M}(P_0) \models (p \leftrightarrow p) \rightarrow (p \leftrightarrow r)$. Thus, by applying the iff-replacement rule we get the program $P_1 = \{p \leftarrow r, \quad q \leftarrow r\}$. We have that P_1 finitely fails wrt p, while P_0 does not existentially terminate wrt p.

In order to have that the initial program of a transformation sequence finitely fails wrt a pertinent goal if the final one does, we strengthen the termination condition as indicated by the following theorem.

Theorem 4 *Let $P_0, \ldots, P_k$ be a transformation sequence constructed by using the definition introduction, definition elimination, iff-replacement, and finite failure rules. Let $\mathcal{G}$ be the set of pertinent goals and suppose that, for every $g \in \mathcal{G}$, if P_k existentially terminates wrt g, then P_0 existentially terminates wrt g. Then for every $g \in \mathcal{G}$, if P_k finitely fails wrt g then P_0 finitely fails wrt g.*

Proof. Suppose that P_k finitely fails wrt a goal $g \in \mathcal{G}$. By Theorem 2 and the soundness and completeness of SLD-resolution, P_0 does not succeed wrt g. Since P_k finitely fails wrt g, by definition of existential termination and our hypotheses, P_0 existentially terminates wrt g and, since P_0 does not succeed wrt g, P_0 finitely fails wrt g. □

The hypothesis of Theorem 4 saying that if P_k existentially terminates wrt g then also P_0 existentially terminates wrt g, can be dropped, provided that we restrict the application of the iff-replacement rule by replacing Condition (5) by the following stronger condition:

$$Comp(P_0 \cup Def_j) \vdash \mathit{iff}\,(C_1, \ldots, C_m) \rightarrow \mathit{iff}\,(D_1, \ldots, D_n) \tag{8}$$

Indeed, we have the following result.

Theorem 5 *Let $P_0, \ldots, P_k$ be a transformation sequence constructed by using the definition introduction, definition elimination, iff-replacement, and finite failure rules. Suppose that for each application of the iff-replacement rule Condition (8) holds. Let $\mathcal{G}$ be the set of pertinent goals and suppose that for every $g \in \mathcal{G}$, if P_0 existentially terminates wrt g, then P_k existentially terminates wrt g. Then, for every $g \in \mathcal{G}$, P_0 finitely fails wrt g iff P_k does.*

Proof. By Theorem 3 we only need to show that, for every $g \in \mathcal{G}$, if P_k finitely fails wrt g then P_0 finitely fails wrt g. Similarly to Lemma 4, we have that for every $g \in \mathcal{G}$, either $Comp(P_0 \cup Def_k) \vdash \neg g$ or all predicates occurring in g are defined in P_k (The proof is by induction on k.) Now there are two cases: α and β.
(α). Assume that $Comp(P_0 \cup Def_k) \vdash \neg g$. Then $P_0 \cup Def_k$ finitely fails wrt g and, therefore, also P_0 finitely fails wrt g.
(β). Assume that all predicates occurring in g are defined in P_k. We also have that P_k finitely fails wrt g. This implies that $Comp(P_k) \vdash \neg g$. Similarly to Lemma 3 we have that $Comp(P_0 \cup Def_k) \vdash Comp(P_k)$. (The proof is by induction on k.) Therefore $Comp(P_0 \cup Def_k) \vdash \neg g$. Thus, as in case ($\alpha$), we get the thesis, and the proof is completed. □

Notice that the derivation of program P_1 from program P_0 in Example 2 cannot be performed by using the iff-replacement rule with Condition (8), because $Comp(P_0) \not\vdash (p \leftrightarrow p) \rightarrow (p \leftrightarrow r)$.

6 Correctness Properties of the Unfold/Fold Rules

In this section we show that many transformation rules presented in the literature (see, for instance, [10, 11, 15, 20]), including various forms of unfolding, folding, and goal replacement, can be viewed as instances of our iff-replacement rule. As a consequence, we can apply our correctness results to these rules, as we now show.

In this section we will consider a transformation sequence $P_0, \ldots, P_j$ and the extended initial program $P_0 \cup Def_j$ associated with this sequence. We also assume that the variables of the clauses which are involved in the application of each transformation rule, are suitably renamed so that no two clauses have variables in common.

Unfolding Rule. Let C be a clause in P_j of the form $H \leftarrow F, A, G$, where A is an atom and F and G are (possibly empty) goals. Suppose that

1. $E_1, \ldots, E_n$, with $n \geq 1$, is the set of all clauses of program P_j such that A is unifiable with $hd(E_1), \ldots, hd(E_n)$, with most general unifiers $\theta_1, \ldots, \theta_n$, respectively, and
2. D_i is the clause $(H \leftarrow F, bd(E_i), G)\theta_i$, for $i = 1, \ldots, n$.

If we *unfold* C *wrt* A, then we derive the clauses $D_1, \ldots, D_n$ and we obtain the new program P_{j+1} which is $(P_j - \{C\}) \cup \{D_1, \ldots, D_n\}$.

An unfolding step corresponds to an application of SLD-resolution to clause C with the selection of the atom A and the input clauses $E_1, \ldots, E_n$.

Lemma 5 *Let $C_1, C_2, \ldots, C_m$ be the definition of a predicate p in P_j. Suppose that by unfolding clause C_1 wrt atom A we derive clauses $D_1, \ldots, D_n$. Then we have*

$$Comp(P_j) \vdash \mathit{iff}\,(C_1, C_2, \ldots, C_m) \leftrightarrow \mathit{iff}\,(D_1, \ldots, D_n, C_2, \ldots, C_m)$$

Proof. Since $Comp(P_j) \vdash \mathit{iff}\,(C_1, C_2, \ldots, C_m)$, to prove the lemma it is enough to show that $Comp(P_j) \vdash \mathit{iff}\,(D_1, \ldots, D_n, C_2, \ldots, C_m)$. This is a consequence of Theorem 4.1(a) of [10], where it is shown that if P_{j+1} is obtained from P_j by unfolding, then $Comp(P_j) \vdash Comp(P_{j+1})$. □

Folding Rule. Let $C_1, \ldots, C_n$, with $n \geq 1$, be clauses in P_j and $E_1, \ldots, E_n$ be a set of clauses in the extended initial program $P_0 \cup Def_j$. Suppose that there exists

an atom A and two goals F and G such that for each $i = 1, \ldots, n$, there exists a substitution θ_i which satisfies the following conditions:

1. C_i is a variant of the clause $H \leftarrow F, bd(E_i)\theta_i, G$
2. $A = hd(E_i)\theta_i$
3. the predicate of A is defined in P_j
4. for any clause E in $(P_0 \cup Def_j) - \{E_1, \ldots, E_n\}$, $hd(E)$ is not unifiable with A
5. for every variable X in $vars(E_i) - vars(hd(E_i))$, $X\theta_i$ is a variable which occurs
   - neither in $\{H, F, G\}$
   - nor in the term $Y\theta_i$, for each variable Y occurring in $bd(E_i)$ and different from X.

If we *fold* $C_1, \ldots, C_n$ *using* $E_1, \ldots, E_n$, we derive the clause $D = H \leftarrow F, A, G$, and we obtain the new program P_{j+1} which is $(P_j - \{C_1, \ldots, C_n\}) \cup \{D\}$.

Our version of the folding rule is similar to the one in [11], where simultaneous folding of n (≥ 1) clauses is allowed. However, in [11] some conditions on the transformation sequence are imposed for ensuring total correctness. We have achieved a similar total correctness result by requiring, instead of those conditions, the preservation of existential termination (see Section 4).

Tamaki and Sato's folding rule [20] is much simpler than our folding rule and it basically corresponds to our rule for $n = 1$. They also impose some extra conditions on the transformation sequence for ensuring total correctness.

The folding rules considered by Maher in [15, 17] differ from our folding rule because in Maher's rules the clauses $E_1, \ldots, E_n$ used for folding are all taken from program P_j, instead of $P_0 \cup Def_j$. Maher's assumption makes the correctness proofs very simple, but it turns out to be a very severe restriction in practice, when one uses program transformation to improve efficiency (see [18] for a more detailed discussion of this issue).

By generalizing analogous results in [10, 15, 17], we may prove the following fact.

Lemma 6 *Suppose that we derive clause D_1 by folding clauses $C_1, \ldots, C_n$ using $E_1, \ldots, E_n$ in $P_0 \cup Def_j$. Let p be the predicate symbol occurring in the heads of clauses $C_1, \ldots, C_n$, and let $C_1, \ldots, C_n, C_{n+1}, \ldots, C_r$ be the definition of p in P_j. Then we have*

$$Comp(P_0 \cup Def_j) \vdash \mathit{iff}\,(C_1, \ldots, C_n, C_{n+1}, \ldots, C_r) \leftrightarrow \mathit{iff}\,(D_1, C_{n+1}, \ldots, C_r).$$

Goal Replacement Rule. Let $P_0, \ldots, P_j$ be a transformation sequence, $P_0 \cup Def_j$ be the extended initial program associated with $P_0, \ldots, P_j$, C be a clause in P_j, and G_1 be a goal occurring in the body of C. Suppose that for some goal G_2 containing only predicates which are defined in P_j, the following holds:

$$\mathcal{M}(P_0 \cup Def_j) \models \forall X_1, \ldots, X_k.\, (\exists Y_1, \ldots, Y_m.\, G_1 \leftrightarrow \exists Z_1, \ldots, Z_n.\, G_2) \qquad (9)$$

where: (i) $X_1, \ldots, X_k$ are all variables of G_1 also occurring outside G_1 in C, (ii) $\{Y_1, \ldots, Y_m\} = vars(G_1) - \{X_1, \ldots, X_k\}$, (iii) $\{Z_1, \ldots, Z_n\} = vars(G_2) - \{X_1, \ldots, X_k\}$, and (iv) $\{Z_1, \ldots, Z_n\} \cap vars(C) = \{\}$. Then, we get program P_{j+1} from P_j by replacing goal G_1 by goal G_2 in the body of clause C.

As already mentioned, one may prove Condition (9) by proving

$$\mathcal{F} \vdash \forall X_1, \ldots, X_k.\, (\exists Y_1, \ldots, Y_m.\, G_1 \leftrightarrow \exists Z_1, \ldots, Z_n.\, G_2) \qquad (10)$$

where $\mathcal{F}$ is any formula which is true in $\mathcal{M}(P_0 \cup Def_j)$. In particular, we may consider a goal replacement rule where Condition (9) has been replaced by the following condition:

$$Comp(P_0 \cup Def_j) \vdash \forall X_1, \ldots, X_k. \, (\exists Y_1, \ldots, Y_m. \, G_1 \leftrightarrow \exists Z_1, \ldots, Z_n. \, G_2). \qquad (11)$$

Our formulation of the goal replacement rule with Condition (9) is similar to that in [20] (which, incidentally, was incorrectly stated as pointed out in [10] and later corrected in [21]). If we replace Condition (9) by Condition (11), then we essentially get the version of the goal replacement rule presented in [15].

Lemma 7 *Suppose that by goal replacement from clause C_1 we derive clause D_1. Let p be the predicate symbol occurring in $hd(C_1)$, and clauses $C_1, C_2, \ldots, C_r$ be the definition of p in P_j. Then we have*

(a) $\mathcal{M}(P_0 \cup Def_j) \models$ *iff* $(C_1, C_2, \ldots, C_r) \leftrightarrow$ *iff* $(D_1, C_2, \ldots, C_r)$.

(b) If Condition (11) *holds then*

$$Comp(P_0 \cup Def_j) \vdash \textit{iff}\,(C_1, C_2, \ldots, C_r) \leftrightarrow \textit{iff}\,(D_1, C_2, \ldots, C_r).$$

Proof. (a) Let C_1 be the clause $p(t) \leftarrow H, G_1, K$ where t is a tuple of terms. Let us assume that: (i) Condition (9) holds for clause C_1 and program P_j, and (ii) we replace goal G_1 by goal G_2. The formula *iff* $(C_1, C_2, \ldots, C_r)$ can be written as $\forall U. \, p(U) \leftrightarrow ((\exists V. \, U = t \wedge H \wedge (\exists Y. G_1) \wedge K) \ \vee \ \Gamma)$, where U is a tuple of fresh new variables, $Y = Y_1, \ldots, Y_m$ is the m-tuple of variables occurring in G_1 and not in the rest of clause C_1, and V is the tuple of variables occurring in $vars(C_1) - \{Y_1, \ldots, Y_m\}$. Γ is a suitable formula relative to clauses $C_2, \ldots, C_r$. Then (a) follows from Condition (9).
The proof of (b) is similar to the proof of (a). □

Theorem 6 *Let $P_0, \ldots, P_k$ be a transformation sequence constructed by using the definition introduction, definition elimination, unfolding, folding, goal replacement (with Condition 9), and finite failure rules, and let $\mathcal{G}$ be a set of pertinent goals. Suppose that for every $g \in \mathcal{G}$, if P_0 existentially terminates wrt g then P_k existentially terminates wrt g. Then the following properties hold:*

(a) $P_0, \ldots, P_k$ is totally correct wrt $\mathcal{G}$, and for every $g \in \mathcal{G}$, if P_0 finitely fails wrt g then P_k finitely fails wrt g.

(b) Suppose that for every $g \in \mathcal{G}$, if P_k existentially terminates wrt g then P_0 existentially terminates wrt g. Then for every $g \in \mathcal{G}$, P_0 finitely fails wrt g iff P_k finitely fails wrt g.

(c) Suppose that for each application of the goal replacement rule Condition (11) *holds. Then for every $g \in \mathcal{G}$, P_0 finitely fails wrt g iff P_k finitely fails wrt g.*

Proof. We first prove that the unfolding, folding, and goal replacement rules (with Condition 9) are instances of the iff-replacement rule (with Condition 5). If P_{j+1} is obtained from P_j by using the unfolding, folding, and goal replacement rules, the definition $C_1, \ldots, C_m$, with $m \geq 1$, of a predicate p in P_j, is replaced by a set of clauses $D_1, \ldots, D_n$, with $n \geq 1$, such that all predicates occurring in $D_1, \ldots, D_n$ also occur in P_j.

We have to show that Condition (5) also holds. We have the following three cases: α, β, and γ.
(α). P_{j+1} is obtained from P_j by unfolding. In this case Condition (5) follows from Lemma 5 and the fact that, by Lemma 3, $\mathcal{M}(P_0 \cup Def_j) \models Comp(P_j)$.

(β). P_{j+1} is obtained from P_j by folding. In this case Condition (5) follows from Lemma 6 and the fact that $\mathcal{M}(P_0 \cup Def_j) \models Comp(P_0 \cup Def_j)$.
(γ). P_{j+1} is obtained from P_j by goal replacement (with Condition 9). In this case Condition (5) follows from Lemma 7(a).

Thus, Point (a) follows from Theorems 2 and 3, and Point (b) follows from Theorems 3 and 4. Point (c) is proved as follows. By the fact that $Comp(P_0 \cup Def_j) \vdash Comp(P_j)$ (the proof is by induction on j) and by Lemma 5, we have that for each application of the unfolding rule Condition (8) holds. We also have that Condition (8) holds for each application of the folding rule and the goal replacement rule with Condition (11), because of Lemmas 6 and 7(b), respectively. Thus, Point (c) follows from Theorem 5. □

In the following Example 3 we show the use of the goal replacement rule to transform a program P_0 into a program P_1 which existentially terminates w.r.t. all ground goals. The total correctness of the transformation is then a consequence of Theorem 6. In contrast, the methods proposed in [4, 8] which require the existence of finite SLD-trees instead of existential termination, cannot be used to show the total correctness of this transformation, because there exists a goal g such that $P_1 \cup \{\neg g\}$ has an infinite SLD-tree independently of the choice of the selection rule. Further comparisons with [4, 8] are made in the next section.

Example 3 Let us consider the following program P_0 for computing the transitive closure of a relation p:

$t(X, Y) \leftarrow p(X, Y)$
$t(X, Z) \leftarrow t(X, Y), p(Y, Z)$
$p(a, a) \leftarrow$
$p(a, b) \leftarrow$
$p(b, c) \leftarrow$

We have that $\mathcal{M}(P_0) \models \forall X, Z.(\exists Y.t(X, Y), p(Y, Z)) \leftrightarrow (\exists V.p(X, V), t(V, Z))$. Thus, by the goal replacement rule we derive a new program P_1 by replacing the second clause of P_0 by the following one:

$t(X, Z) \leftarrow p(X, V), t(V, Z)$

Program P_1 existentially terminates w.r.t. all ground goals. However, the SLD-tree for $P_1 \cup \{\neg t(a, a)\}$ is infinite, independently of the choice of the selection rule.

7 Related Work

Many authors have investigated the correctness of logic program transformation rules following the approach proposed by Tamaki and Sato [20], which consists in imposing restrictions on the use of the rules so as to ensure total correctness. One problem with this approach is that these restrictions are either very constraining or they require the verification of complex, sometimes undecidable, properties [18].

Recently, advances in the study of logic program termination (see [9] for a survey) have stimulated the investigation of the relationship between program transformation and program termination. For instance, Amtoft [1] attempts to give a general semantic framework where one can reason about the preservation of termination when transforming definite logic programs by unfolding and folding. However, more general transformation rules, such as our iff-replacement rule, are not taken into consideration.

Bossi and Cocco [4] focus their study on definite logic programs which are evaluated using the Prolog leftmost computation rule. They consider the following notion, called *left-termination* in [2]: a program P left-terminates wrt a goal g iff $P \cup \{\neg g\}$ has a finite SLD-tree constructed using the leftmost computation rule. They prove that restricted versions of Tamaki and Sato's unfolding and folding rules preserve both the set of computed answer substitutions and left-termination.

Bossi and Etalle [5] consider general logic programs and show that Tamaki and Sato's rules preserve *acyclicity* of programs [9]. This property is used to derive various correctness results for the unfolding and folding transformations.

Our approach is quite different from those in [4, 5]. We do not deal with the preservation of termination *during* transformation, as done in those papers, but we study the preservation of properties which, together with the existential termination of the final programs, ensure the total correctness of the entire transformation. Our approach allows us to formulate the transformation rules in a very general way, and to get, as straightforward corollaries, the results on the total correctness of the unfolding, folding, and goal replacement rules. Moreover, the notion of existential termination used in this paper is weaker than that of left-termination in [4], and that of acyclicity in [5]. Notice also that the property that all answer substitutions computed by a given initial program are also computed by any derived program, which is used in [4] to show the correctness results, turns out to be redundant for our purposes here.

The paper by Cook and Gallagher [8] is closely related to ours. They provide some termination conditions that ensure the preservation of both the success set and the finite failure set of definite logic programs. They consider two transformation rules only: unfolding and goal replacement. Our rules are more general than theirs, and in particular, our folding rule cannot be viewed as an instance of Cook and Gallagher's goal replacement. Moreover, our goal replacement rule is different from the one they propose, because, in order to replace a goal G_1 in the body of a clause, say $H \leftarrow G_1, A$, by a goal G_2, Cook and Gallagher's rule requires that G_1 and G_2 have the same set of computed answer substitutions wrt the current program in the transformation sequence, whereas we require that G_1 and G_2 be equivalent wrt the least Herbrand model (or wrt the completion) of the extended initial program.

Cook and Gallagher's goal replacement rule also requires that, after the replacement step all ground instances of H have a *finite* SLD-tree. In contrast, we only need to check existential termination (which does not imply the finiteness of the SLD-tree) at the end of the transformation process. One more very relevant difference between Cook and Gallagher's approach and ours, is the already mentioned use of invariants which greatly simplifies the correctness proofs.

Conclusions

We have investigated in the case of logic programs the relationship between the correctness of the transformation rules and the existential termination of programs, and we have derived a method for ensuring totally correct transformations, which is similar to that originally proposed in [6].

We would like to make a general remark on the role of the termination analysis for program transformation. Indeed, since termination is an undecidable property, one may wonder about the usefulness of our approach wrt the one proposed in [20].

As noted in [8], a lot of work has recently been done in the area of automatic termination analysis, and one may take advantage of the proposed techniques for

the verification of the total correctness of program transformations. Moreover, as shown in [4, 5, 8], one may also find various syntactic conditions to ensure that the termination properties needed for the proof of correctness are preserved during the transformation process.

We think that our study has contributed to the understanding of the problem of the total correctness of program transformations by separating the concern of proving partial correctness, which basically involves model-theoretic reasoning, from the concern of proving termination properties, which basically involves proof-theoretic reasoning.

Acknowledgments

We would like to thank A. Bossi, P. Dell'Acqua, J. Gallagher, and our colleagues of the Esprit Compulog II Project for stimulating discussions about the issues of logic program transformation and termination. Thanks also to the anonymous referees for their suggestions. The first author was partially supported by the European Union HCM Project on Logic Program Synthesis and Transformation (contract 93/414), the CNR (Italy), and the British Council (Italy). The last two authors were partially supported by the Esprit Compulog II Project and the CNR Project Anatra.

References

[1] T. Amtoft. Unfold/fold transformations preserving termination properties. In *Proc. PLILP'92, Leuven, Belgium*, Lecture Notes in Computer Science 631, pages 187–201. Springer-Verlag, 1992.

[2] K. R. Apt and D. Pedreschi. Studies in Pure Prolog: Termination. In J. W. Lloyd, editor, *Proceedings of the Symposium on Computational Logic, Brussels, Belgium*, pages 150–176. Springer-Verlag, 1990.

[3] A. Bossi and N. Cocco. Basic transformation operations which preserve computed answer substitutions of logic programs. *Journal of Logic Programming*, 16(1&2):47–87, 1993.

[4] A. Bossi and N. Cocco. Preserving universal termination through unfold/fold. In *Proceedings ALP'94*, LNCS 850, pages 269–286. Springer-Verlag, 1994.

[5] A. Bossi and S. Etalle. Transforming acyclic programs. *ACM Transactions on Programming Languages and Systems*, 16(4):1081–796, July 1994.

[6] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–77, January 1977.

[7] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[8] J. Cook and J. P. Gallagher. A transformation system for definite programs based on termination analysis. In L. Fribourg and F. Turini, editors, *Proceedings of LOPSTR'94 and META'94, Pisa, Italy*, LNCS 883, pages 51–78. Springer-Verlag, 1994.

[9] D. De Schreye and S. Decorte. Termination of logic programs: The never-ending story. *Journal of Logic Programming*, 19, 20:199–260, 1994.

[10] P. A. Gardner and J. C. Shepherdson. Unfold/fold transformations of logic programs. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic, Essays in Honor of Alan Robinson*, pages 565–583. MIT, 1991.

[11] M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. In *Proceedings Sixth International Symposium on Programming Language Implementation and Logic Programming* (*PLILP'94*), LNCS 844, pages 340–354. Springer-Verlag, 1994.

[12] C. J. Hogger. Derivation of logic programs. *Journal of the ACM*, 28(2):372–392, 1981.

[13] K.-K. Lau, M. Ornaghi, and S.-Å.Tärnlund. The halting problem for deductive synthesis of logic programs. In P. van Hentenryck, editor, *Proc.* 11^{th} *Int. Conf. on Logic Programming*, pages 665–783. MIT Press, 1994.

[14] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, Second Edition, 1987.

[15] M. J. Maher. Correctness of a logic program transformation system. IBM Research Report RC 13496, T. J. Watson Research Center, 1987.

[16] M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proc. of the 3rd Annual Symposium on Logic in Computer Science*, pages 348–357, Edinburgh, 1988. IEEE Computer Society Press.

[17] M. J. Maher. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science*, 110:377–403, 1993.

[18] A. Pettorossi and M. Proietti. Transformation of logic programs: Foundations and techniques. *Journal of Logic Programming*, 19,20:261–320, 1994.

[19] T. Sato. An Equivalence Preserving First Order Unfold/Fold Transformation System. *Theoretical Computer Science*, 105:57–84, 1992.

[20] H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In S.-Å. Tärnlund, editor, *Proceedings Second International Conference on Logic Programming, Uppsala, Sweden*, pages 127–138. Uppsala University, 1984.

[21] H. Tamaki and T. Sato. A generalized correctness proof of the unfold/fold logic program transformation. Technical Report 86-4, Ibaraki University, Japan, 1986.