ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
"Antonio Ruberti"
CONSIGLIO NAZIONALE DELLE RICERCHE

A. Pettorossi, M. Proietti

TOTALLY CORRECT LOGIC PROGRAM TRANSFORMATIONS VIA WELL-FOUNDED ANNOTATIONS

R. 639, Febbraio 2006

Alberto Pettorossi – Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, I-00133 Roma, Italy, and Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy.
Email : pettorossi@info.uniroma2.it. URL : http://www.iasi.cnr.it/~adp.

Maurizio Proietti – Istituto di Analisi dei Sistemi ed Informatica del CNR, Viale Manzoni 30, I-00185 Roma, Italy. Email : proietti@iasi.cnr.it.
URL : http://www.iasi.cnr.it/~proietti.

This paper is a revised, extended version of: A. Pettorossi and M. Proietti. A Theory of Totally Correct Logic Program Transformations. In: *Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, August 24–25, 2004, Verona, Italy*, ACM Press, 2004, pp. 159–168.

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.cnr.it
URL: http://www.iasi.cnr.it

Abstract

We address the problem of proving the total correctness of transformations of definite logic programs. We consider a general transformation rule, called clause replacement, which consists in transforming a program P into a new program Q by replacing a set Γ_1 of clauses occurring in P by a new set Γ_2 of clauses, provided that Γ_1 and Γ_2 are equivalent in the least Herbrand model $M(P)$ of the program P.

We propose a general method for proving that transformations based on clause replacement are totally correct, that is, $M(P) = M(Q)$. Our method consists in showing that the transformation of P into Q can be performed by: (i) adding extra arguments to predicates, thereby deriving from the given program P an annotated program $\overline{P}$, (ii) applying a variant of the clause replacement rule and transforming the annotated program $\overline{P}$ into a terminating annotated program $\overline{Q}$, and (iii) erasing the annotations from $\overline{Q}$, thereby getting Q.

Our method does not require that either P or Q are terminating and it is parametric with respect to the annotations. By providing different annotations we can easily prove the total correctness of program transformations based on various versions of the popular unfolding, folding, and goal replacement rules, which can all be viewed as particular cases of our clause replacement rule.

Key words: Program transformation rules, logic programming, partial and total correctness, well-founded orderings.

1. Introduction

Program transformation rules can be viewed as conditional rewritings of programs. Indeed, the transformation of a given program P into a new program Q can be performed by using various rules, each of which rewrites a fragment of the program P into a new fragment, provided that these two fragments are equivalent with respect to a given semantics (see, for instance, [19]). In this paper we consider definite logic programs and, in order to transform them, we introduce a general transformation rule, called *clause replacement*, which is a conditional rewriting of the following form: a set Γ_1 of clauses of a program P is rewritten into a new set Γ_2 of clauses, provided that a suitable equivalence between Γ_1 and Γ_2 holds in the least Herbrand model of the program P. This model is denoted by $M(P)$. Most transformation rules proposed in the literature, including the popular unfolding, folding, and goal replacement rules [7, 28], can be viewed as particular cases of this clause replacement rule.

Much work has been devoted to the study of the correctness of program transformations of definite programs (see, for instance, [4, 8, 11, 13, 16, 24, 28, 29]). Two correctness properties have been considered: *partial correctness* and *total correctness*. A transformation of a program P into a program Q is said to be *partially correct* iff $M(P) \supseteq M(Q)$, and it is said to be *totally correct* iff $M(P) = M(Q)$.

In Section 2 we will show that any transformation from an initial program P to a final program Q performed by applying the clause replacement rule is partially correct. However, this transformation may *not* be totally correct, that is, $M(P) \subseteq M(Q)$ may not hold. We will illustrate this fact in Example 1 below, where we consider the goal replacement rule which, as already mentioned, is a particular instance of the clause replacement rule. (Other, more realistic applications of the goal replacement rule will be presented in Example 9 and Section 6.) The goal replacement rule can be defined as follows. A clause $H \leftarrow G_L \wedge G_1 \wedge G_R$ of a program P is replaced by a new clause $H \leftarrow G_L \wedge G_2 \wedge G_R$, provided that the following equivalence holds in the least Herbrand model of P: $M(P) \models \forall X\, (\exists Y_1\, G_1 \leftrightarrow \exists Y_2\, G_2)$, where X is the set of variables occurring in $\{H, G_L, G_R\}$ and, for $i = 1, 2$, Y_i is the set of variables occurring in the goal G_i and not in X. A totally correct variant of the goal replacement rule will be introduced in Section 4.

Example 1. (i) Let us consider the transformation of P into Q, where P and Q are programs defined as follows:

P:	$p(f(X)) \leftarrow q(X)$	Q:	$p(f(X)) \leftarrow q(X)$
	$q(a) \leftarrow$		$q(a) \leftarrow$
	$q(X) \leftarrow p(f(X))$		$q(X) \leftarrow q(X)$

This transformation is a legal application of the goal replacement rule defined above, because $M(P) \models \forall X\, (p(f(X)) \leftrightarrow q(X))$. The transformation of P into Q is totally correct, because $M(P) = \{p(f(a)), q(a)\} = M(Q)$.

(ii) Since equivalence is symmetric, also the replacement of $q(X)$ by $p(f(X))$ in the body of the first clause of P is a legal application of the goal replacement rule. However, this goal replacement determines a transformation which is *not* totally correct. Indeed, if we replace $q(X)$ by $p(f(X))$ in the first clause of P, we get the following program:

R:	$p(f(X)) \leftarrow p(f(X))$
	$q(a) \leftarrow$
	$q(X) \leftarrow p(f(X))$

and we have that $M(P) = \{p(f(a)), q(a)\} \supset \{q(a)\} = M(R)$. □

4.

Since the pioneering work by Tamaki and Sato [28], various authors have proposed suitable extra conditions which guarantee the total correctness of transformations determined by applications of the unfolding, folding, and goal replacement rules [4, 8, 11, 13, 16, 24, 28, 29]. The essential idea presented by Tamaki and Sato in [28] is that the replacement of G_1 by G_2 determines a totally correct transformation if, in addition to the condition $M(P) \models \forall X\,(\exists Y_1\,G_1 \leftrightarrow \exists Y_2\,G_2)$, we have that, for every ground substitution σ for the variables in X, if there exist a ground substitution ϑ_1 for the variables in Y_1 and a proof π_1 of $G_1\sigma\vartheta_1$ in P, then there exist a ground substitution ϑ_2 for the variables in Y_2 and a proof π_2 of $G_2\sigma\vartheta_2$ in P, such that the measure of π_2 is not larger than the measure of π_1. Here, by *measure of a proof* we mean the number of nodes of the proof when it is represented as an *and-tree* of atoms (the measure defined in [28] is slightly more general, but that generality is not required here).

For instance, let us consider the program P of Example 1 and the equivalence $M(P) \models \forall X\,(p(f(X)) \leftrightarrow q(X))$. The only provable ground instances of $p(f(X))$ and $q(X)$ are $p(f(a))$ and $q(a)$, respectively. We have that every proof of $p(f(a))$ in P has measure greater than or equal to 2, whereas there exists a proof of $q(a)$ in P which has measure 1. Thus, the replacement of $p(f(X))$ by $q(X)$ satisfies the condition by Tamaki and Sato (and, indeed, this transformation is totally correct), while the replacement of $q(X)$ by $p(f(X))$ does *not* satisfy that condition (and, indeed, this transformation is *not* totally correct). More sophisticated proof measures are defined in [24, 29]. However, in [24, 29] and also in [28] one cannot find any general methodology for comparing proof measures and checking the conditions which ensure the total correctness of the transformations based on goal replacements.

The main contribution of this paper is a method for proving the total correctness of the transformations based on the clause replacement rule and, as a consequence, the total correctness of the transformations based on the unfolding, folding, and goal replacement rules. By our method we can express the conditions which ensure total correctness by first order formulas that can be checked by using standard theorem proving techniques.

Let us briefly describe our method in the particular case of the goal replacement rule presented above. Suppose that program P is transformed into program Q by replacing goal G_1 in the clause $H \leftarrow G_L \wedge G_1 \wedge G_R$ of P by the new goal G_2. In order to show the partial correctness of this transformation, that is, in order to show that $M(P) \supseteq M(Q)$, it suffices to prove that $M(P) \models \forall X\,(\exists Y_1\,G_1 \leftarrow \exists Y_2\,G_2)$ (and, thus, $M(P) \models \forall X\,\forall Y_1\,(H \leftarrow G_L \wedge G_1 \wedge G_R) \rightarrow \forall X\,\forall Y_2\,(H \leftarrow G_L \wedge G_2 \wedge G_R)$).

In order to show also the reverse inclusion $M(P) \subseteq M(Q)$, and thus, the total correctness of the replacement of G_1 by G_2, we will use the *unique fixpoint principle* (see [9] for a short presentation in the case of recursive equation programs) and our new proof method based on *program annotations*. In our context, the unique fixpoint principle can be formulated as follows: *if* $M(P) \models \forall X\,(\exists Y_1\,G_1 \rightarrow \exists Y_2\,G_2)$ and the *immediate consequence operator* T_Q [1, 15] associated with the derived program Q has a unique fixpoint, *then* $M(P) \subseteq M(Q)$. Now, a sufficient (but not necessary) condition ensuring that T_Q has a unique fixpoint is that Q is *terminating*, that is, every SLD-derivation starting from a ground goal is finite [3].

However, the condition that the operator T_Q has a unique fixpoint, is too restrictive in practice, because it is often the case that for a logic program Q, T_Q does *not* have a unique fixpoint. For instance, the operator T_Q associated with the non-terminating program Q of Example 1(i) has infinitely many fixpoints, each of which is of the form:

$\{p(f(a)), \ldots, p(f^{n+1}(a)), q(a), \ldots, q(f^n(a))\}$ for some $n \geq 0$.

Thus, the unique fixpoint principle alone is not sufficient to prove the total correctness of the

transformation presented in Example 1(i). A more realistic example of non-terminating program whose immediate consequence operator has more than one fixpoint, is the reachability program of Example 9 below.

We will overcome these limitations by introducing *program annotations* as we now explain by looking at the transformation of program P into program Q presented in Example 1(i). Program Q can be derived from program P by performing the following three steps.

(*Step* 1) From program P we derive the following *annotated* program:

$\overline{P}$: 1. $p(f(X))\langle M\rangle \leftarrow M > N \wedge q(X)\langle N\rangle$
2. $q(a)\langle M\rangle \leftarrow$
3. $q(X)\langle M\rangle \leftarrow M > N \wedge p(f(X))\langle N\rangle$

where: (i) M and N are distinct *annotation variables* ranging over the set $\mathbb{N}$ of natural numbers, (ii) $>$ denotes the usual *greater-than* well-founded ordering on $\mathbb{N}$, and (iii) $M > N$ is an *annotation formula*. The annotation variables should be considered as extra arguments of the annotated atoms. For instance, the annotated atom $q(X)\langle N\rangle$ should be considered identical to the atom $q(X, N)$. Thus, by considering the annotation formulas as *constraints*, the annotated program $\overline{P}$ is a constraint logic program for which we can define a least $\mathbb{N}$-model, denoted by $M(\overline{P})$ (see [12]).

The following property holds for $M(P)$ and $M(\overline{P})$:

(Property 1) For every ground atom $A \in M(P)$ there exists $n \in \mathbb{N}$ such that $A\langle n\rangle \in M(\overline{P})$.

Since the least $\mathcal{D}$-model of a constraint logic program, for any constraint domain $\mathcal{D}$, is also a model of the *completion* of the program, which is obtained by replacing sets of clauses by if-and-only-if definitions [1, 12, 15], we have the following property which is derived from clause 1:

(Property 2) $M(\overline{P}) \models \forall N\, (p(f(X))\langle N\rangle \leftrightarrow \exists K\, (N > K \wedge q(X)\langle K\rangle))$

(*Step* 2) By replacing $p(f(X))\langle N\rangle$ by $N > K \wedge q(X)\langle K\rangle$ in the body of clause 3 in program $\overline{P}$, we derive the following clause:

4. $q(X)\langle M\rangle \leftarrow M > N \wedge N > K \wedge q(X)\langle K\rangle$

Let $\overline{Q}$ be the annotated program consisting of clauses 1, 2, and 4. We have the following property:

(Property 3) Every annotated clause of the form $H\langle M\rangle \leftarrow c(M, N) \wedge A\langle N\rangle$ in $\overline{Q}$ is *decreasing* (with respect to $>$), that is, the implication $\forall M\, \forall N\, (c(M, N) \rightarrow M > N)$ holds. (In particular, clause 4 is decreasing because $\forall M\, \forall N\, \forall K\, (M > N \wedge N > K \rightarrow M > K)$ holds.)

(*Step* 3) Finally, we get program Q by erasing all annotation variables and annotation formulas from program $\overline{Q}$. We have the following property:

(Property 4) For every ground atom A, if there exists $n \in \mathbb{N}$ such that $A\langle n\rangle \in M(\overline{Q})$, then $A \in M(Q)$.

Let us present a few remarks on Properties 1–4. Property 1 says that the annotations added to program P do not restrict its least Herbrand model. In Section 3 we will provide conditions on annotations which guarantee that this property holds in general (see Definition 3.3 and Proposition 3.4). Property 2 is of the form: $M(\overline{P}) \models \forall X\, (\exists Y_1\, G_1 \leftrightarrow \exists Y_2\, G_2)$, where G_1 and G_2 are goals containing annotation variables and annotation formulas. However, only the only-if part $M(\overline{P}) \models \forall X\, (\exists Y_1\, G_1 \rightarrow \exists Y_2\, G_2)$ is used to prove the reverse inclusion $M(P) \subseteq M(Q)$. Property 3 ensures that program $\overline{Q}$ terminates according to the following notion of termination,

which is weaker than the notion considered in [3]: every SLD-derivation starting from a ground goal and constructed using the left-to-right atom selection rule is finite. (Indeed, the left-to-right atom selection rule ensures that annotation formulas are selected first.) This weaker termination notion, which is also called *left-termination*, is sufficient to guarantee that $T_{\overline{Q}}$ has a unique fixpoint [2]. In the sequel we need not distinguish between termination in the sense of [3], and left-termination, because these notions will not be used in the technical results we will derive. In particular, in Section 3 we will make a direct proof that the fixpoint of $T_{\overline{Q}}$ is unique by using the hypothesis that $\overline{Q}$ is decreasing. Finally, Property 4 is the converse of Property 1, where P is replaced by Q, and holds in general for every program Q and annotated program $\overline{Q}$, as shown in Proposition 3.2 of Section 3.

Now, let us explain why the above Properties 1–4 ensure that $M(P) \subseteq M(Q)$. Let us take $A \in M(P)$. By Property 1 there exists $n \in \mathbb{N}$ such that $A\langle n \rangle \in M(\overline{P})$. By Properties 2 and 3, and by the unique fixpoint principle, we have that $M(\overline{P}) \subseteq M(\overline{Q})$ and, therefore, $A\langle n \rangle \in M(\overline{Q})$. Finally, by Property 4, we conclude that $A \in M(Q)$.

We would like to stress that we have applied our method for proving the total correctness of the transformation of program P into program Q, where neither P nor Q is terminating.

Our method based on program annotations can be used not only to prove that a given transformation is totally correct, but also to prevent incorrect transformations. Indeed, we can rule out incorrect transformations by requiring that the annotated clauses which are derived by program transformation, are decreasing with respect to a suitable well-founded ordering. Let us consider, for instance, the replacement of $M > N \wedge q(X)\langle N \rangle$ by $p(f(X))\langle M \rangle$ in the body of the first clause of $\overline{P}$. By Property 2 above, this is a legal goal replacement, but the corresponding program transformation is not totally correct. Now, by applying this goal replacement, we would get the following annotated clause:

$p(f(X))\langle M \rangle \leftarrow p(f(X))\langle M \rangle$

which is *not* decreasing and, thus, Property 3 does not hold. Thus, if we restrict ourselves to transformations that produce decreasing annotated clauses, then the incorrect replacement of $M > N \wedge q(X)\langle N \rangle$ by $p(f(X))\langle M \rangle$ is ruled out.

The paper is structured as follows. In Section 2 we introduce the clause replacement transformation rule, which generalizes the unfolding, folding, and goal replacement rules. We prove the partial correctness of the transformations based on the clause replacement rule and we also give a sufficient condition for their total correctness based on the unique fixpoint principle. In Section 3 we introduce program annotations and, in particular, *well-founded annotations*, that is, annotations which produce decreasing programs. Then we prove a sufficient condition based on well-founded annotations that ensures the total correctness of the transformations based on the clause replacement rule. This condition is the basis of the method we propose in this paper for proving the total correctness of program transformations. In Section 4 we present variants of the unfolding, folding, and goal replacement rules for annotated programs and we use the results of Section 3 for showing that the transformations based on these rules are totally correct. In Section 5 we present a technique for proving program properties, such as implications and equivalences between goals, by means of the unfolding, folding, and goal replacement rules for annotated programs. In Section 6 we present an extended example of application of our method for proving the total correctness of program transformations when they are based on the unfolding, folding, and goal replacement rules. Finally, in Section 7 we compare our method with other related techniques published in the literature.

2. Clause Replacement

In this section we will introduce the *clause replacement* transformation rule for definite logic programs. All usual program transformation rules, such as unfolding, folding, and goal replacement, are instances of this clause replacement rule. Indeed, we will prove that clause replacement is the most general program transformation rule, in the sense that every totally correct program transformation can be obtained by applying this rule (see Theorem 2.9). Then we will extend to the transformations based on clause replacements some correctness results which have been established for the transformations based on the unfolding, folding, and goal replacements (see, for instance, [4, 8, 11, 13, 16, 24, 28, 29]. In particular, (i) we will prove the partial correctness of the program transformations based on clause replacements (see Theorem 2.10), and (ii) we will give a sufficient condition for the total correctness of these transformations. This condition is based on the uniqueness of the fixpoint of the immediate consequence operator of the program derived by the transformation (see Corollary 2.14 below).

In what follows, unless otherwise stated, we will adopt the standard notions and terminology which are used in logic programming [1, 15]. However, unlike [1, 15], a clause is denoted by $A \leftarrow A_1 \wedge \ldots \wedge A_n$, with $n \geq 0$ (instead of $A \leftarrow A_1, \ldots, A_n$) and a *goal* is defined to be a conjunction of atoms (instead of the negation of a conjunction of atoms). The empty conjunction is identified with *true* and a clause of the form $A \leftarrow \mathit{true}$ is also written as $A \leftarrow$. The empty disjunction is identified with *false*. The set of variables occurring in a term t is denoted by $\mathit{vars}(t)$. A similar notation is also used for the set of variables occurring in a formula or in a set of formulas. Given a clause C of the form $A \leftarrow G$, the head A of C is denoted by $\mathit{hd}(C)$ and the body G of C is denoted by $\mathit{bd}(C)$. Given a predicate p and a clause C with predicate p in its head, C is said to be *a clause for* p.

Let us briefly summarize the fixpoint semantics of definite logic programs. Recall that an Herbrand interpretation I is a set of ground atoms. For a ground atom A, we write $I \models A$ iff $A \in I$. For a first order formula φ the satisfaction relation $I \models \varphi$ is defined as usual in first order logic by induction on the structure of φ. The *immediate consequence operator* associated with a program P is a function T_P from Herbrand interpretations to Herbrand interpretations, defined as follows:

$T_P(I) = \{A \mid$ there exists a ground instance $A \leftarrow G$ of a clause in P such that $I \models G\}$

T_P is a continuous function on the complete lattice of Herbrand interpretations ordered by set inclusion. Thus, T_P has a least fixpoint, denoted by $\mathit{lfp}(T_P)$, and a *greatest fixpoint*, denoted by $\mathit{gfp}(T_P)$. We have that $\mathit{lfp}(T_P)$ is also the least *prefixpoint* of T_P, that is, the least Herbrand interpretation I such that $T_P(I) \subseteq I$. Similarly, $\mathit{gfp}(T_P)$ is also the greatest *postfixpoint* of T_P, that is, the greatest Herbrand interpretation I such that $T_P(I) \supseteq I$. Since T_P is continuous, $\mathit{lfp}(T_P)$ is the least upper bound of the chain $\{T_P^n(\emptyset) \mid n \in \mathbb{N}\}$. Moreover, $\mathit{lfp}(T_P)$ is the least Herbrand model of P, that is, the least Herbrand interpretation I such that $I \models P$. $\mathit{lfp}(T_P)$ is also denoted by $M(P)$.

We assume that in every Herbrand interpretation I, the equality predicate, denoted $=$, is interpreted as the identity relation over ground terms, that is, for any two ground terms t_1 and t_2, we have that $I \models t_1 = t_2$ (or, equivalently, $t_1 = t_2 \in I$) iff t_1 is syntactically identical to t_2.

Before giving the formal definition of the clause replacement rule, let us introduce that rule by means of a simple example. Let us consider the following *EvenOdd* program:

8.

EvenOdd:

1. $p(X) \leftarrow even(X)$
2. $p(X) \leftarrow odd(X)$
3. $even(0) \leftarrow$
4. $even(s(s(X))) \leftarrow even(X)$
5. $odd(s(0)) \leftarrow$
6. $odd(s(s(X))) \leftarrow odd(X)$
7. $nat(0) \leftarrow$
8. $nat(s(X)) \leftarrow nat(X)$

We have that the conjunction of clauses 1 and 2 is logically equivalent to $p(X) \leftarrow even(X) \vee odd(X)$, and the following equivalence holds in the least Herbrand model of *EvenOdd*:

$M(EvenOdd) \models \forall X\,((even(X) \vee odd(X)) \leftrightarrow nat(X))$

The clause replacement rule allows us to derive a new program by replacing clauses 1 and 2 by the following clause:

9. $p(X) \leftarrow nat(X)$

In order to define the clause replacement rule in a formal way (see Definition 2.5 below) we need some auxiliary notions. First, in Definition 2.1 below we define the *if-form* of a set Γ of clauses for a predicate p, as a formula of the form $p(\ldots) \leftarrow \varphi$, where φ is a disjunction of existentially quantified conjunctions of atoms. For instance, in the *EvenOdd* example, the *if*-form of the set consisting of clauses 1 and 2 is $p(X) \leftarrow even(X) \vee odd(X)$. Then, in Definition 2.2 we define the notions of implication ($\Rightarrow$), reverse implication ($\Leftarrow$), and equivalence ($\Leftrightarrow$) between sets of clauses, based on implication, reverse implication, and equivalence between the premises of their *if*-forms. For instance, in the *EvenOdd* example, we have that $M(EvenOdd) \models \{\text{clause 1, clause 2}\} \Leftrightarrow \{\text{clause 9}\}$.

Definition 2.1 (*if*-form) Let p be a predicate symbol of arity j (≥ 0) and let Γ be a set of n (≥ 0) clauses for p. The *if-form* of Γ, denoted by $if(\Gamma)$, is a formula constructed as specified by the following three steps, where $X_1, \ldots, X_j$ are distinct variables not occurring in Γ.

(*Step* 1 : *Introduce equalities*) Transform each clause $p(t_1, \ldots, t_j) \leftarrow A_1 \wedge \ldots \wedge A_k$ of Γ into

$p(X_1, \ldots, X_j) \leftarrow X_1 = t_1 \wedge \ldots \wedge X_j = t_j \wedge A_1 \wedge \ldots \wedge A_k$

(*Step* 2 : *Introduce existential quantifiers*) Transform each formula $p(X_1, \ldots, X_j) \leftarrow F$ derived at the end of Step 1 into

$p(X_1, \ldots, X_j) \leftarrow \exists Y_1 \ldots \exists Y_m\, F$

where $\{Y_1, \ldots, Y_m\} = vars(F) - \{X_1, \ldots, X_j\}$.

(*Step* 3 : *Introduce disjunctions*) Let

$p(X_1, \ldots, X_j) \leftarrow R_1$

$\ldots$

$p(X_1, \ldots, X_j) \leftarrow R_n$

be the formulas obtained at the end of Step 2. Then $if(\Gamma)$ is the formula:

$p(X_1, \ldots, X_j) \leftarrow R_1 \vee \ldots \vee R_n$

Note that Steps 1–3 of Definition 2.1 are the first three steps in the construction of the completion of a logic program [1, page 536]. The following is an example of *if*-form of a set of clauses.

Example 2. Let Γ be the set consisting of the following two clauses:

$p(a) \leftarrow$

$p(f(Y_1)) \leftarrow q(Y_1, Y_2) \wedge r(Y_2)$

Then $if(\Gamma)$ is the formula: $p(X) \leftarrow X = a \vee \exists Y_1 \exists Y_2\,(X = f(Y_1) \wedge q(Y_1, Y_2) \wedge r(Y_2))$. □

Note that if Γ is the empty set of clauses, that is, in Definition 2.1 we have $n=0$, then $if(\Gamma)$ is the formula $p(X_1,\ldots,X_j) \leftarrow \mathit{false}$ (recall that *false* is the empty disjunction). Note also that any set Γ of clauses for a predicate p is logically equivalent to $if(\Gamma)$, in the sense that if $\Gamma = \{C_1,\ldots,C_n\}$, then $\models \forall\,(C_1 \wedge \ldots \wedge C_n) \leftrightarrow \forall\,(if(\Gamma))$.

Given a set Γ of clauses and a predicate symbol p, by $\Gamma \restriction p$ we denote the set of clauses for p in Γ.

Definition 2.2. (Implication, Reverse-Implication, and Equivalence between Sets of Clauses) Let I be an Herbrand interpretation. Let Γ_1 and Γ_2 be two sets of clauses for the same predicate p of arity j (≥ 0). Let $if(\Gamma_1)$ be $p(X_1,\ldots,X_j) \leftarrow \varphi_1$ and let $if(\Gamma_2)$ be (a variant of) $p(X_1,\ldots,X_j) \leftarrow \varphi_2$. We say that Γ_1 *implies* Γ_2 in the interpretation I, and we write $I \models \Gamma_1 \Rightarrow \Gamma_2$, iff

$I \models \forall X_1 \ldots \forall X_j\,(\varphi_2 \rightarrow \varphi_1)$

Let Γ_1 and Γ_2 be any two sets of clauses.
(*Implication*) We say that Γ_1 *implies* Γ_2 in the interpretation I, and we write $I \models \Gamma_1 \Rightarrow \Gamma_2$, iff for every predicate p occurring in $\Gamma_1 \cup \Gamma_2$, we have

$I \models (\Gamma_1 \restriction p) \Rightarrow (\Gamma_2 \restriction p)$

(*Reverse-Implication*) We say that Γ_1 *is implied by* Γ_2 in the interpretation I, and we write $I \models \Gamma_1 \Leftarrow \Gamma_2$, iff $I \models \Gamma_2 \Rightarrow \Gamma_1$.
(*Equivalence*) We say that Γ_1 *is equivalent to* Γ_2 in the interpretation I, and we write $I \models \Gamma_1 \Leftrightarrow \Gamma_2$, iff $I \models \Gamma_1 \Rightarrow \Gamma_2$ and $I \models \Gamma_1 \Leftarrow \Gamma_2$.

Note that the implication $\Rightarrow$ between sets of clauses Γ_1 and Γ_2 implies the logical implication $\rightarrow$ between Γ_1 and Γ_2. Indeed, for any Herbrand interpretation I, if $I \models \forall X_1 \ldots \forall X_j\,(\varphi_2 \rightarrow \varphi_1)$ holds, then $I \models \forall X_1 \ldots \forall X_j\,((p(X_1,\ldots,X_j) \leftarrow \varphi_1) \rightarrow (p(X_1,\ldots,X_j) \leftarrow \varphi_2))$ holds.

Let us now give an example of equivalence between sets of clauses.

Example 3. Let us consider the sets of clauses Γ_1 and Γ_2, consisting of the following clauses:

$$\begin{array}{lll} \Gamma_1: & p(a) \leftarrow & \Gamma_2: \quad p(a) \leftarrow \\ & p(b) \leftarrow & \qquad\quad p(b) \leftarrow \\ & q(f(a)) \leftarrow & \qquad\quad q(f(Y)) \leftarrow p(Y) \\ & q(f(b)) \leftarrow & \end{array}$$

Let $M(\Gamma_1)$ be the least Herbrand model of Γ_1. The following equivalence holds:

$M(\Gamma_1) \models \Gamma_1 \Leftrightarrow \Gamma_2$

Indeed, we have that:

$if(\Gamma_1 \restriction p) \;=\; if(\Gamma_2 \restriction p) \;=\; (p(X) \leftarrow X=a \vee X=b)$
$if(\Gamma_1 \restriction q) \;=\; (q(X) \leftarrow X=f(a) \vee X=f(b))$
$if(\Gamma_2 \restriction q) \;=\; (q(X) \leftarrow \exists Y(X=f(Y) \wedge p(Y))$

and we have that:

$M(\Gamma_1) \models \forall X\,((X=f(a) \vee X=f(b)) \leftrightarrow \exists Y(X=f(Y) \wedge p(Y)))$ □

The following lemma, whose proof is given in the Appendix, will be useful to prove our partial and total correctness results.

Lemma 2.3. *Let I be an Herbrand interpretation. Let Γ_1 and Γ_2 be two sets of clauses. We have that $I \models \Gamma_1 \Rightarrow \Gamma_2$ iff for every ground instance C_2 of a clause in Γ_2 such that $I \models bd(C_2)$ there exists a ground instance C_1 of a clause in Γ_1 such that $hd(C_1) = hd(C_2)$ and $I \models bd(C_1)$.*

For every Herbrand interpretation I and sets of clauses Γ_1, Γ_2, and Γ_3 the following properties hold:

Reflexivity: $I \models \Gamma_1 \Rightarrow \Gamma_1$
Transitivity: if $I \models \Gamma_1 \Rightarrow \Gamma_2$ and $I \models \Gamma_2 \Rightarrow \Gamma_3$ then $I \models \Gamma_1 \Rightarrow \Gamma_3$
Monotonicity: if $I \models \Gamma_1 \Rightarrow \Gamma_2$ then $I \models \Gamma_1 \cup \Gamma_3 \Rightarrow \Gamma_2 \cup \Gamma_3$.

Now we introduce our basic program transformation rule, called *clause replacement rule*, which allows us to construct a sequence of programs starting from a given initial program P_0. This sequence of programs will be called a *transformation sequence* and it is formally defined as follows.

Definition 2.4 (Transformation Sequence) A *transformation sequence* from an initial program P_0 to a final program P_n, with $n \geq 0$, is a sequence of programs, denoted $P_0 \mapsto \ldots \mapsto P_n$, such that, for $k = 0, \ldots, n-1$, program P_{k+1} is derived from program P_k by applying the following *clause replacement rule*.

Definition 2.5 (Clause Replacement Rule) Let us consider a transformation sequence $P_0 \mapsto \ldots \mapsto P_k$, for any $k \geq 0$. Let Γ_k be a set of clauses for a predicate p such that $\Gamma_k \subseteq P_k$, and let Δ_k be a set of clauses for p such that $M(P_0) \models \Gamma_k \Leftrightarrow \Delta_k$.
By applying the clause replacement rule we derive the new program $P_{k+1} = (P_k - \Gamma_k) \cup \Delta_k$ and we derive the transformation sequence $P_0 \mapsto \ldots \mapsto P_k \mapsto P_{k+1}$.

Note that the familiar unfolding, folding, and goal replacement rules [28] are instances of the clause replacement rule.

By abuse of notation, by $P_0 \mapsto \ldots \mapsto P_n$, for $n \geq 0$, we will also denote *any* sequence of $n+1$ programs from P_0 to P_n. The context will tell the reader whether or not $P_0 \mapsto \ldots \mapsto P_n$ is a transformation sequence, that is, it is derived by n applications of the clause replacement rule.

In order to prove the partial and total correctness properties of transformation sequences, we will find it useful to introduce the following two notions of program sequences, called *implication-based* and *reverse-implication-based* program sequences. These notions arise by separating the equivalence $\Gamma_k \Leftrightarrow \Delta_k$ which has to be satisfied when applying the clause replacement rule (see Definition 2.5), into the two conjuncts $\Gamma_k \Rightarrow \Delta_k$ and $\Gamma_k \Leftarrow \Delta_k$.

Definition 2.6. (Implication-based and Reverse-implication-based Program Sequence) A sequence of programs $P_0 \mapsto \ldots \mapsto P_n$, for $n \geq 0$, is said to be *implication-based* (or *reverse-implication-based*) iff for $k = 0, \ldots, n-1$, there exist two sets Γ_k and Δ_k of clauses for the same predicate such that:
(i) $\Gamma_k \subseteq P_k$,
(ii) $M(P_0) \models \Gamma_k \Rightarrow \Delta_k$ (or $M(P_0) \models \Gamma_k \Leftarrow \Delta_k$, respectively), and
(iii) $P_{k+1} = (P_k - \Gamma_k) \cup \Delta_k$.

A transformation sequence is both an implication-based and a reverse-implication-based program sequence, and it will also be called an *equivalence-based* program sequence.

Definition 2.7 (Correctness of Program Sequences) A sequence of programs $P_0 \mapsto \ldots \mapsto P_n$, for $n \geq 0$, is said to be:
(i) *partially correct* iff $M(P_0) \supseteq M(P_n)$,
(ii) *conservative* iff $M(P_0) \subseteq M(P_n)$, and
(iii) *totally correct* iff $M(P_0) = M(P_n)$.

Note that in a transformation sequence $P_0 \mapsto \ldots \mapsto P_n$, we require that for $k = 0, \ldots, n-1$, if P_{k+1} is $(P_k - \Gamma_k) \cup \Delta_k$ then the equivalence $\Gamma_k \Leftrightarrow \Delta_k$ should hold in the least Herbrand model $M(P_0)$ of the initial program P_0. Recall also that, by Definition 2.2, any two sets Γ_k and Δ_k of clauses are equivalent if and only if $\Gamma_k \restriction p$ and $\Delta_k \restriction p$ are equivalent for every predicate p in $\Gamma_k \cup \Delta_k$. As a consequence, by an application of the clause replacement rule we can introduce a clause of the form $newp(X) \leftarrow B$ for a new predicate *newp* only when $M(P_0) \models \forall (B \leftrightarrow \mathit{false})$ holds. Thus, the *definition introduction* rule [28] *cannot* be viewed as an instance of the clause replacement rule.

For reasons of simplicity, we have chosen *not* to include the definition introduction rule among our transformation rules. However, the total correctness results presented in this paper can easily be extended to the case where program sequences are constructed by using the definition introduction rule in addition to the clause replacement rule, because the total correctness of a program sequence constructed by using the clause replacement rule and the definition introduction rule, can be reduced, as we now explain, to the total correctness of a transformation sequence, that is, a sequence of programs constructed by using the clause replacement rule only.

Indeed, similarly to [28], we may stipulate that a program sequence $P_0 \mapsto \ldots \mapsto P_n$ constructed by using the clause replacement rule and the definition introduction rule is totally correct iff $M(P_0 \cup \mathit{Defs}) = M(P_n)$, where *Defs* is the set of clauses for the new predicates, that is, the predicates not occurring in P_0 which are added during the sequence by applying the definition introduction rule. Then, given a program sequence $P_0 \mapsto \ldots \mapsto P_n$ constructed by using the clause replacement rule and the definition introduction rule, we can construct a program sequence of the form $P_0 \cup \mathit{Defs} \mapsto \ldots \mapsto P_n$ by using the clause replacement rule only. In other words, we may think as if all new predicate definitions were added to the initial program P_0, while, in practice, we allow ourselves to add these new predicate definitions to any program of the sequence, when they are actually needed.

The following lemma is a straightforward consequence of the reflexivity, transitivity, and monotonicity properties of $\Rightarrow$.

Lemma 2.8. *Let $P_0 \mapsto \ldots \mapsto P_n$, for $n \geq 0$, be a sequence of programs.*
(i) *If $P_0 \mapsto \ldots \mapsto P_n$ is implication-based, then $M(P_0) \models P_0 \Rightarrow P_n$.*
(ii) *If $P_0 \mapsto \ldots \mapsto P_n$ is reverse-implication-based, then $M(P_0) \models P_0 \Leftarrow P_n$.*

The following theorem shows that the clause replacement rule is a *complete* transformation rule in the sense that, given any two programs having the same least Herbrand model, it is possible to derive one of them from the other by a transformation sequence, that is, a sequence of clause replacements. Note, however, that in general, the applicability condition for the clause replacement rule, that is, $M(P_0) \models \Gamma_k \Leftrightarrow \Delta_k$, is undecidable.

Theorem 2.9 (Completeness of the Clause Replacement Rule) *Given two programs P and Q, if $M(P) = M(Q)$ then there exists a transformation sequence $P_0 \mapsto \ldots \mapsto P_n$ with $P = P_0$ and $Q = P_n$.*

Proof. Let $p_1, \ldots, p_n$ be the predicates occurring in $P \cup Q$. For $k = 1, \ldots n$, let $\Gamma_k = P \restriction p_k$ and $\Delta_k = Q \restriction p_k$. Let us consider the sequence $P_0 \mapsto \ldots \mapsto P_n$ of programs, where $P_0 = P$, $P_n = Q$, and, for $k = 1, \ldots n$, $P_k = (P_{k-1} - \Gamma_k) \cup \Delta_k$. We will show that $P_0 \mapsto \ldots \mapsto P_n$ is a transformation sequence, that is, for $k = 1, \ldots n$, $M(P) \models \Gamma_k \Leftrightarrow \Delta_k$.

By Lemma 2.3 it is enough to show that, for $k = 1, \ldots n$, there exists a ground instance $H \leftarrow G_1$ of a clause in Γ_k such that $M(P) \models G_1$ iff there exists a ground instance $H \leftarrow G_2$ of a clause in Δ_k such that $M(P) \models G_2$.

Let $H \leftarrow G_1$ be a ground instance of a clause in Γ_k (thus, the predicate of H is p_k) such that $M(P) \models G_1$. By definition of an Herbrand model, we have that $H \in M(P)$. Since $M(Q)$ is a postfixpoint of T_Q and $M(P) = M(Q)$, also $M(P)$ is a postfixpoint of T_Q, that is, $M(P) \subseteq T_Q(M(P))$. Thus, $H \in T_Q(M(P))$ and, by definition of T_Q, there exists a ground instance $H \leftarrow G_2$ of a clause in Q such that $M(P) \models G_2$. Since the predicate of H is p_k, we have that $H \leftarrow G_2$ is a ground instance of a clause in Δ_k.

Similarly, we can prove that if there exists a ground instance $H \leftarrow G_2$ of a clause in Δ_k such that $M(P) \models G_2$, then there exists a ground instance $H \leftarrow G_1$ of a clause in Γ_k such that $M(P) \models G_1$. ∎

Now we present some sufficient conditions ensuring that a sequence of programs is partially correct and conservative, and thus, totally correct. First we show that every implication-based sequence of programs is partially correct.

Theorem 2.10 (Partial Correctness) *If $P_0 \mapsto \ldots \mapsto P_n$, for $n \geq 0$, is an implication-based sequence of programs, then $M(P_0) \supseteq M(P_n)$.*

Proof. First we show that $M(P_0)$ is a prefixpoint of T_{P_n}, that is, $T_{P_n}(M(P_0)) \subseteq M(P_0)$. Let A be a ground atom in $T_{P_n}(M(P_0))$. By definition of T_{P_n} there exists a ground instance $A \leftarrow G_2$ of a clause in P_n such that $M(P_0) \models G_2$. Since $P_0 \mapsto \ldots \mapsto P_n$ is an implication-based sequence of programs, by Lemma 2.8(i) we have that $M(P_0) \models P_0 \Rightarrow P_n$. Thus, by Lemma 2.3, there exists a ground instance $A \leftarrow G_1$ of a clause in P_0 such that $M(P_0) \models G_1$. Hence, by definition of T_{P_0}, $A \in T_{P_0}(M(P_0))$. Since $M(P_0)$ is a prefixpoint of T_{P_0}, we have that $T_{P_0}(M(P_0)) \subseteq M(P_0)$ and, therefore, $A \in M(P_0)$. Thus, we have proved that $M(P_0)$ is a prefixpoint of T_{P_n}. Since $M(P_n) = \mathit{lfp}(T_{P_n})$ and $\mathit{lfp}(T_{P_n})$ is the least prefixpoint of T_{P_n}, we have that $M(P_0) \supseteq M(P_n)$. ∎

Since every transformation sequence is an implication-based sequence of programs, Theorem 2.10 also tells us that every transformation sequence is partially correct.

Corollary 2.11 (Partial Correctness of Transformation Sequences) *If $P_0 \mapsto \ldots \mapsto P_n$, for $n \geq 0$, is a transformation sequence, then $M(P_0) \supseteq M(P_n)$.*

Now we describe a method based on the *unique fixpoint principle* [9, 22], which can be applied to prove that a sequence of programs is conservative. Let us first introduce the following definition.

Definition 2.12 (Univocal Program) A program P is said to be *univocal* iff T_P has a unique fixpoint, that is, $\mathit{lfp}(T_P) = \mathit{gfp}(T_P)$.

A sufficient condition for a program to be univocal is that it is *terminating* in the sense of [3]. Note, however, that this condition is *not* necessary. For instance, consider the program

P: $p \leftarrow$
$p \leftarrow p$

This program is not terminating and, nevertheless, the immediate consequence operator T_P has a unique fixpoint which is $\{p\}$.

Theorem 2.13 (Conservativity) *If $P_0 \mapsto \ldots \mapsto P_n$, for $n \geq 0$, is a reverse-implication-based sequence of programs and P_n is univocal, then $M(P_0) \subseteq M(P_n)$.*

Proof. We first show that $M(P_0)$ is a postfixpoint of T_{P_n}, that is, $T_{P_n}(M(P_0)) \supseteq M(P_0)$. Let A be a ground atom in $M(P_0)$. Since $M(P_0)$ is a postfixpoint of T_{P_0}, that is, $T_{P_0}(M(P_0)) \supseteq M(P_0)$, we have that there exists a ground instance $A \leftarrow G_1$ of a clause in P_0 such that $M(P_0) \models G_1$. Since $P_0 \mapsto \ldots \mapsto P_n$ is a reverse-implication-based sequence of programs, by Lemma 2.8(ii) we have that $M(P_0) \models P_0 \Leftarrow P_n$. Therefore, by Lemma 2.3, there exists a ground instance $A \leftarrow G_2$ of a clause in P_n such that $M(P_0) \models G_2$. By definition of T_{P_n}, we have that $A \in T_{P_n}(M(P_0))$. Thus, we have proved that $M(P_0)$ is a postfixpoint of T_{P_n}. Since $gfp(T_{P_n})$ is the greatest postfixpoint of T_{P_n}, we have that $M(P_0) \subseteq gfp(T_{P_n})$. Finally, by the hypothesis that P_n is univocal, that is, $gfp(T_{P_n}) = lfp(T_{P_n}) = M(P_n)$, we get that $M(P_0) \subseteq M(P_n)$. ∎

Since every transformation sequence is an equivalence-based sequence of programs, by Theorems 2.10 and 2.13 we have the following total correctness result.

Corollary 2.14 (Total Correctness via the Unique Fixpoint Principle) *If $P_0 \mapsto \ldots \mapsto P_n$, for $n \geq 0$, is a transformation sequence and P_n is univocal, then $M(P_0) = M(P_n)$.*

Note that, in order to apply the unique fixpoint principle, it is important that the notion of equivalence between sets of clauses and the notion of transformation sequence are defined as stated in Definitions 2.2 and 2.4, respectively. To illustrate this point, let us first consider the following alternative definition of equivalence between sets of clauses: two sets of clauses $\{C_1, \ldots, C_k\}$ and $\{D_1, \ldots, D_m\}$ are equivalent with respect to an interpretation I iff $I \models \forall (C_1 \wedge \ldots \wedge C_k) \leftrightarrow \forall (D_1 \wedge \ldots \wedge D_m)$. If we use this alternative definition of equivalence, then Corollary 2.14 does not hold, as shown by the following example.

Example 4. Let us consider the sequence $P_0 \mapsto P_1$ of programs, where:

P_0: $p \leftarrow \mathit{true}$ $\qquad$ P_1: $p \leftarrow \mathit{false}$

We have that $M(P_0) \models (p \leftarrow \mathit{true}) \leftrightarrow (p \leftarrow \mathit{false})$, because p (and, therefore, $p \leftarrow \mathit{true}$) is true in $M(P_0)$ and $p \leftarrow \mathit{false}$ is true in every interpretation. We also have that P_1 is univocal (the only fixpoint of T_{P_1} is the empty set). However, the sequence $P_0 \mapsto P_1$ of programs is not totally correct, because $\{p\} = M(P_0) \neq M(P_1) = \emptyset$.

Note that, on the contrary, if the equivalence relation $\Leftrightarrow$ is defined as in Definition 2.2, then we have that $M(P_0) \not\models \{p \leftarrow \mathit{true}\} \Leftrightarrow \{p \leftarrow \mathit{false}\}$, because $M(P_0) \not\models \mathit{true} \leftrightarrow \mathit{false}$. □

Let us now consider an alternative definition of a transformation sequence $P_0 \mapsto \ldots \mapsto P_n$ in which we assume that for $k = 0, \ldots, n-1$, when deriving P_{k+1} from P_k we have that: $M(P_k) \models \Gamma_k \Leftrightarrow \Delta_k$, instead of $M(P_0) \models \Gamma_k \Leftrightarrow \Delta_k$ (see Definition 2.5). If we use this alternative definition of a transformation sequence, Corollary 2.14 does not hold, as shown by the following example.

Example 5. Let us consider the sequence $P_0 \mapsto P_1 \mapsto P_2$ of programs, where:

P_0: $p \leftarrow \mathit{true}$ $\qquad$ P_1: $p \leftarrow p$ $\qquad$ P_2: $p \leftarrow \mathit{false}$

We have that $M(P_0) \models \{p \leftarrow \mathit{true}\} \Leftrightarrow \{p \leftarrow p\}$, because $M(P_0) \models \mathit{true} \leftrightarrow p$. We also have that $M(P_1) \models \{p \leftarrow p\} \Leftrightarrow \{p \leftarrow \mathit{false}\}$, because $M(P_1) \models p \leftrightarrow \mathit{false}$. Finally, P_2 is univocal, but the sequence of programs $P_0 \mapsto P_1 \mapsto P_2$ is not totally correct, because it is the case that $\{p\} = M(P_0) \neq M(P_2) = \emptyset$.

Note that $P_0 \mapsto P_1 \mapsto P_2$ is not a transformation sequence according to our Definition 2.4 above. Indeed, P_2 cannot be derived from P_1 by applying the clause replacement rule of Definition 2.5, because $M(P_0) \not\models p \leftrightarrow \mathit{false}$. □

The result of Corollary 2.14 gives us a useful method for proving the total correctness of a transformation sequence. However, this method cannot be applied when the program derived by clause replacement is *not* univocal. For instance, Corollary 2.14 cannot be applied to prove the total correctness of the transformation sequences presented in Example 1(i) of the Introduction and in the more realistic Example 9 of the following Section 3. Note that, in particular, for program Q of Example 1(i), T_Q has more than one fixpoint. In Section 3 we will present a method that overcomes this limitation and can be applied even if the programs derived by clause replacement are not univocal.

3. Well-Founded Annotations

In this section we present our method based on the notion of *program annotation* for proving that a transformation sequence is totally correct. In particular, we introduce *well-founded* annotations, that is, annotations which generate *decreasing* (thus, terminating and univocal) annotated programs. First, we present the syntax and the semantics of annotated clauses and annotated programs. Then, we present an extension of the clause replacement rule that can be used for transforming annotated programs. Finally, we present a sufficient condition based on well-founded annotations, which guarantees the total correctness of the transformation sequences constructed by using the clause replacement rule.

Let L_p be a first order language, called the *language of programs*, and let us consider the sets of definite clauses, definite logic programs, and well-formed formulas of the language L_p which we call *Clauses*, *Programs*, and *Formulas*, respectively.

Let us also consider a different first order language, called the *language of annotations*, denoted L_a, such that $L_a \cap L_p = \emptyset$. We introduce the *annotation formulas* of the language L_a as follows. We assume that in the set of predicate symbols of L_a there is a symbol $\succ$, which will be interpreted as a well-founded ordering relation on a given set W (thus, by definition, no infinite descending sequence $w_1 \succ \ldots \succ w_n \succ \ldots$ exists in W). A variable of the language L_a is called an *annotation variable*. Similarly, a term, an atom, and a well-formed formula of L_a is called an *annotation term*, an *annotation atom*, and an *annotation formula*, respectively. We will add the qualification *ordinary* to variables, terms, atoms, and formulas of L_p when we want to distinguish them from those of L_a.

Now, we introduce the *annotated formulas* of L_p and L_a as follows. An *annotated atom* is of the form $A\langle w\rangle$, where A is an atom of L_p and w is an annotation term of L_a. A formula φ is an *annotated formula* iff one of the following holds:

(i) φ is an annotation formula;
(ii) φ is an annotated atom;
(iii) $\varphi = \neg\varphi_1$, where φ_1 is an annotated formula;
(iv) $\varphi = \varphi_1 \wedge \varphi_2$, where φ_1 and φ_2 are annotated formulas;
(v) $\varphi = \forall X\, \varphi_1$, where X is a variable of $L_a \cup L_p$ and φ_1 is an annotated formula.

When constructing annotated formulas we will also use the connectives $\vee$, $\leftarrow$, $\rightarrow$, $\leftrightarrow$, and the quantifier $\exists$, which are defined as usual in terms of $\neg$, $\wedge$, and $\forall$. The set of annotated formulas is denoted by *AFormulas*. As usual in the first order predicate calculus, we say that an ordinary or annotated term (or formula) is *ground* iff it contains no occurrences of variables, and we say that an ordinary or annotated formula is *closed* iff it contains no occurrences of free variables.

An *annotated goal* is a conjunction of annotated atoms. An *annotated clause* is an annotated formula of the form:

$H\langle w\rangle \leftarrow c \wedge A_1\langle w_1\rangle \wedge \ldots \wedge A_n\langle w_n\rangle \qquad$ with $n \geq 0$,

where: (i) $H\langle w\rangle$, $A_1\langle w_1\rangle, \ldots, A_n\langle w_n\rangle$ are annotated atoms, and (ii) c is an annotation formula. For reasons of simplicity, we assume that in an annotated clause no quantifiers occur in the annotation formula c. The set of annotated clauses is denoted by *AClauses*. An *annotated program* is a set of annotated clauses. The set of annotated programs is denoted by *APrograms*. Annotated atoms, annotated goals, annotated clauses, and annotated programs are denoted by overlined symbols, such as $\overline{A}$, $\overline{G}$, $\overline{C}$, and $\overline{P}$, respectively. An example of an annotated program is program $\overline{P}$ given in the Introduction. More examples will be given in the sequel.

The semantics of annotated formulas (and, in particular, annotated programs) can be defined similarly to the semantics of *constraint logic programs* [12] as we now describe. We fix an interpretation $\mathcal{W}$ for L_a. Let W be the carrier of $\mathcal{W}$ which, without loss of generality, is assumed to be a set of ground annotation terms. We assume that the predicate symbol $\succ$ is interpreted as a well-founded ordering relation on W which, by abuse of language, will also be denoted by $\succ$. The interpretation $\mathcal{W}$ will also be called the *well-founded ordering* $(W, \succ)$. A $\mathcal{W}$-*interpretation* is a subset of the following set $B_{\mathcal{W}}$:

$B_{\mathcal{W}} = \{A\langle w\rangle \mid A \text{ is a ground atom and } w \in W\ \}$

Let us consider a $\mathcal{W}$-interpretation I. A closed annotation formula c is true in I iff $\mathcal{W} \models c$, where the satisfaction relation $\models$ is defined as usual in the first order predicate calculus. A closed annotated atom (that is, a ground annotated atom) $A\langle w\rangle$ is true in I iff $A\langle w_{\mathcal{W}}\rangle \in I$, where $w_{\mathcal{W}}$ is the interpretation of the term w in $\mathcal{W}$. A closed annotated formula of the form $\neg\varphi$ is true in I iff φ is not true in I. A closed annotated formula of the form $\varphi_1 \wedge \varphi_2$ is true in I iff both φ_1 and φ_2 are true in I. A closed annotated formula of the form $\forall X\, \varphi$ is true in I iff for every ground substitution $\{X/t\}$, $\varphi\{X/t\}$ is true in I. An annotated formula φ such that the variables $X_1, \ldots, X_n$ occur free in φ, is true in I iff for every ground substitution $\{X_1/t_1, \ldots, X_n/t_n\}$, $\varphi\{X_1/t_1, \ldots, X_n/t_n\}$ is true in I. A set Γ of annotated formulas is true in I iff every annotated formula of Γ is true in I. If an annotated formula φ is true in a $\mathcal{W}$-interpretation I, then we write $I \models \varphi$ and we say that I is a $\mathcal{W}$-*model* of φ. The same terminology will also be used for sets of annotated formulas (in particular, for annotated programs).

Similarly to the case of constraint logic programs, it can be shown that every annotated program $\overline{P}$ has a *least* $\mathcal{W}$-*model* which we will denote by $M(\overline{P})$ (for least $\mathcal{W}$-models we adopt the same notation used for least Herbrand models of definite logic programs). It can also be shown that for every annotated program $\overline{P}$ the least $\mathcal{W}$-model of $\overline{P}$ can be computed as the least fixpoint of a suitable continuous function $T_{\overline{P}}$ over $\mathcal{W}$-interpretations, called the *immediate consequence operator* of the program $\overline{P}$. $T_{\overline{P}}$ is defined as follows. For every annotated program $\overline{P}$ we define a function $T_{\overline{P}} : \mathcal{P}(B_{\mathcal{W}}) \rightarrow \mathcal{P}(B_{\mathcal{W}})$, where $\mathcal{P}(B_{\mathcal{W}})$ denotes the powerset of $B_{\mathcal{W}}$, such that for every $I \in \mathcal{P}(B_{\mathcal{W}})$,

$$T_{\overline{P}}(I) = \{\overline{H} \mid \text{there exists a ground instance } \overline{H} \leftarrow c \wedge \overline{G} \text{ of an annotated clause in } \overline{P} \text{ such that } I \models c \wedge \overline{G}\}$$

Similarly to the case of definite and constraint logic programs (see, for instance, [1, 12, 15]), we have the following result.

Theorem 3.1. *The set $\mathcal{P}(B_{\mathcal{W}})$ is a complete lattice with respect to set inclusion and, for every annotated program $\overline{P}$, the immediate consequence operator $T_{\overline{P}} : \mathcal{P}(B_{\mathcal{W}}) \rightarrow \mathcal{P}(B_{\mathcal{W}})$ is a continuous function. Thus, $T_{\overline{P}}$ has a least fixpoint $lfp(T_{\overline{P}})$ and a greatest fixpoint $gfp(T_{\overline{P}})$. Moreover, $lfp(T_{\overline{P}})$ is the least upper bound of the chain $\{T^n_{\overline{P}}(\emptyset) \mid n \in \mathbb{N}\}$ and $lfp(T_{\overline{P}})$ is the least $\mathcal{W}$-model $M(\overline{P})$ of $\overline{P}$.*

We can erase annotation formulas and annotation terms from annotated atoms, annotated goals, annotated clauses, and annotated programs by using the *projection* function π defined as follows:

- $\pi(A\langle w\rangle) = A$, for every annotated atom $A\langle w\rangle$,
- $\pi(\overline{A}_1 \wedge \ldots \wedge \overline{A}_n) = \pi(\overline{A}_1) \wedge \ldots \wedge \pi(\overline{A}_n)$, for every annotated goal $\overline{A}_1 \wedge \ldots \wedge \overline{A}_n$,
- $\pi(\overline{H} \leftarrow c \wedge \overline{G}) = \pi(\overline{H}) \leftarrow \pi(\overline{G})$, for every annotated clause $\overline{H} \leftarrow c \wedge \overline{G}$, and
- $\pi(\{\overline{C}_1, \ldots, \overline{C}_k\}) = \{\pi(\overline{C}_1), \ldots, \pi(\overline{C}_k)\}$, for every annotated program $\{\overline{C}_1, \ldots, \overline{C}_k\}$.

When we apply the projection function π to an annotated program $\overline{P}$, we obtain an ordinary program $\pi(\overline{P})$ such that the least $\mathcal{W}$-model of $\overline{P}$ is isomorphic to a subset of the least Herbrand model of $\pi(\overline{P})$. This property is formally stated by the following proposition whose proof is straightforward and is omitted.

Proposition 3.2. *Let $\overline{P}$ be an annotated program and let P be $\pi(\overline{P})$. For every ground atom A, if there exists a ground annotation term w such that $A\langle w\rangle \in M(\overline{P})$ then $A \in M(P)$.*

The converse of Proposition 3.2 does not hold in the sense that, for some ground annotated atom $A\langle w\rangle$, we have that: $A\langle w\rangle \notin M(\overline{P})$ and $A \in M(P)$, as shown by the following example.

Example 6. Let us also consider the following annotated program $\overline{P}$:

$p\langle 1\rangle \leftarrow q\langle 0\rangle$
$q\langle 1\rangle \leftarrow$

By applying the projection function π we get the following program P:

$p \leftarrow q$
$q \leftarrow$

We have that $M(\overline{P}) = \{q\langle 1\rangle\}$ and $M(P) = \{p, q\}$. □

For the theory of total correctness of logic program transformations developed in this paper it is important to construct a class of annotated programs for which the converse of Proposition 3.2 holds, so that $A \in M(P)$ *if and only if* there exists a ground annotation term w such that $A\langle w\rangle \in M(\overline{P})$. In order to construct such a class of annotated programs we introduce the following definition of *annotation function*.

Definition 3.3 (Annotation Function) Let $\mathcal{W}$ be a well-founded ordering $(W, \succ)$ providing an interpretation for the annotation language L_a. An *annotation over* $\mathcal{W}$ is a function α : *Clauses* $\rightarrow$ *AClauses* such that, for every clause C of the form $H \leftarrow A_1 \wedge \ldots \wedge A_n$, the annotated clause $\alpha(C)$ is of the form $H\langle X\rangle \leftarrow c \wedge A_1\langle X_1\rangle \wedge \ldots \wedge A_n\langle X_n\rangle$, such that:

(i) $X, X_1, \ldots, X_n$ are distinct annotation variables, and

(ii) $\mathcal{W} \models \forall X_1 \ldots \forall X_n \, \exists Y_1 \ldots \exists Y_m \, c$, where $\{Y_1, \ldots, Y_m\} = vars(c) - \{X_1, \ldots, X_n\}$.

The annotation α can be extended to a function, also denoted by α, from *Programs* to *APrograms*, by stipulating that, for every program $\{C_1, \ldots, C_n\}$, the annotated program $\alpha(\{C_1, \ldots, C_n\})$ is $\{\alpha(C_1), \ldots, \alpha(C_n)\}$.

For any program P, we have $P = \pi(\alpha(P))$. The following proposition states the converse of Proposition 3.2 under the hypothesis that annotated programs are constructed by using annotation functions.

Proposition 3.4. *Let P be a program, let α be an annotation function, and let $\overline{P}$ be the annotated program $\alpha(P)$. For every ground atom A, if $A \in M(P)$ then there exists a ground annotation term w such that $A\langle w\rangle \in M(\overline{P})$.*

Proof. Recall that $M(P)$ is the least upper bound of the chain $\{T_P^n(\emptyset) \mid n \in \mathbb{N}\}$ and $M(\overline{P})$ is the least upper bound of the chain $\{T_{\overline{P}}^n(\emptyset) \mid n \in \mathbb{N}\}$. Thus, it is enough to prove that, for every $n \in \mathbb{N}$, the following property holds:

$\varphi(n)$: if $A \in T_P^n(\emptyset)$ then there exists a ground annotation term w such that $A\langle w\rangle \in T_{\overline{P}}^n(\emptyset)$

We proceed by induction on n. (Basis) $\varphi(0)$ is trivially true. (Step) We assume $\varphi(k)$ and we prove $\varphi(k+1)$. Suppose that $A \in T_P^{k+1}(\emptyset)$. Then, by the definition of T_P, there exists a ground instance $(H \leftarrow A_1 \wedge \ldots \wedge A_m)\vartheta$, for some substitution ϑ, of a clause $H \leftarrow A_1 \wedge \ldots \wedge A_m$ in P such that $A = H\vartheta$ and $A_1\vartheta \in T_P^k(\emptyset), \ldots, A_m\vartheta \in T_P^k(\emptyset)$. Let $H\langle X\rangle \leftarrow c \wedge A_1\langle X_1\rangle \wedge \ldots \wedge A_m\langle X_m\rangle$ be the annotated clause $\alpha(H \leftarrow A_1 \wedge \ldots \wedge A_m)$ in $\overline{P}$. By the induction hypothesis $\varphi(k)$, there exist m ground annotation terms $w_1, \ldots, w_m$ such that $A_1\vartheta\langle w_1\rangle \in T_{\overline{P}}^k(\emptyset), \ldots, A_m\vartheta\langle w_m\rangle \in T_{\overline{P}}^k(\emptyset)$. Let σ be the substitution $\{X_1/w_1, \ldots, X_m/w_m\}$. By Conditions (i) and (ii) of Definition 3.3, there exists a ground substitution τ such that $c\,\sigma\tau$ is a ground annotation formula, $\mathcal{W} \models c\,\sigma\tau$, and $X\tau$ is a ground annotation term. Thus, by the definition of $T_{\overline{P}}$, $A\langle X\tau\rangle \in T_{\overline{P}}^{k+1}(\emptyset)$. ∎

In the following example we present two annotation functions.

Example 7. Let $\mathcal{N}$ be the well-founded ordering $(\mathbb{N}, >)$, where $\mathbb{N}$ is the set of natural numbers and $>$ is the usual 'greater than' ordering on $\mathbb{N}$.

(i) The annotation α_1 over $\mathcal{N}$ is defined as follows: for every clause C: $H \leftarrow A_1 \wedge \ldots \wedge A_n$, the annotated clause $\alpha_1(C)$ is

$H\langle X\rangle \leftarrow X > X_1 + \ldots + X_n \wedge A_1\langle X_1\rangle \wedge \ldots \wedge A_n\langle X_n\rangle$

where $+$ is interpreted in $\mathcal{N}$ as the addition of natural numbers.

(ii) The annotation α_2 over $\mathcal{N}$ is defined as follows: for every clause C: $H \leftarrow A_1 \wedge \ldots \wedge A_n$, the annotated clause $\alpha_2(C)$ is

$H\langle X\rangle \leftarrow X > X_1 \wedge \ldots \wedge X > X_n \wedge A_1\langle X_1\rangle \wedge \ldots \wedge A_n\langle X_n\rangle$

Both α_1 and α_2 are indeed annotation functions, because the following properties hold:

$\mathcal{N} \models \forall X_1 \ldots \forall X_n\, \exists X\, (X > X_1 + \ldots + X_n)$

$\mathcal{N} \models \forall X_1 \ldots \forall X_n\, \exists X\, (X > X_1 \wedge \ldots \wedge X > X_n)$ □

Now we give an example of annotated program which is *not* obtained by applying an annotation function.

Example 8. Let us consider the following annotated program $\overline{P}$:

$p\langle Y\rangle \leftarrow Y = 1 \wedge X = 0 \wedge q\langle X\rangle$

$q\langle Y\rangle \leftarrow Y = 1$

which is an equivalent way of writing the annotated program of Example 6. We have that $\mathcal{N} \not\models \forall X\, \exists Y\, (Y = 1 \wedge X = 0)$ and, thus, $\overline{P}$ is not obtained by applying an annotation function. □

In the rest of the paper, we assume that *every annotated program is constructed by applying an annotation function.* Moreover, for any program P, we denote by $\overline{P}$ the annotated program obtained from P by applying some annotation function to P. Thus, for any P, we have that: (i) there exists an annotation function α such that $\overline{P} = \alpha(P)$, and (ii) $\pi(\overline{P}) = P$. A similar

notation is also used for clauses and, thus, for any clause C, we denote by $\overline{C}$ the annotated clause obtained from C by applying some annotation function.

Let us now introduce the notion of *well-founded* annotation, which is an annotation function yielding annotated programs that are terminating and, thus, univocal. We will use the following notations. Given two annotated atoms $\overline{A}_1 = A_1\langle w_1\rangle$ and $\overline{A}_2 = A_2\langle w_2\rangle$, the formula $w_1 \succ w_2$ is also written as $\overline{A}_1 \succ \overline{A}_2$. Moreover, given an annotated atom $\overline{H}$ and an annotated goal $\overline{A}_1 \wedge \ldots \wedge \overline{A}_n$, the formula $\overline{H} \succ \overline{A}_1 \wedge \ldots \wedge \overline{H} \succ \overline{A}_n$ is also written as $\overline{H} \succ (\overline{A}_1 \wedge \ldots \wedge \overline{A}_n)$.

Definition 3.5 (Well-Founded Annotation) Let $\mathcal{W}$ be the well-founded ordering $(W, \succ)$. An annotated clause $\overline{H} \leftarrow c \wedge \overline{A}_1 \wedge \ldots \wedge \overline{A}_n$ is said to be *decreasing* w.r.t. $\succ$ iff

$\mathcal{W} \models \forall (c \rightarrow \overline{H} \succ (\overline{A}_1 \wedge \ldots \wedge \overline{A}_n))$

An annotated program $\overline{P}$ is said to be *decreasing* w.r.t. $\succ$ iff every clause in $\overline{P}$ is decreasing w.r.t. $\succ$. An annotation α is said to be *well-founded* w.r.t. $\succ$ iff for every program P, the annotated program $\alpha(P)$ is decreasing w.r.t. $\succ$.

The annotations α_1 and α_2 presented in Example 7 are both well-founded w.r.t. $>$. The next theorem gives us a sufficient condition for an annotated program to be univocal. This condition is based on the notion of decreasing, annotated program.

Theorem 3.6. *Suppose that an annotated program $\overline{P}$ is decreasing w.r.t. a given well-founded ordering. Then $\overline{P}$ is univocal and $M(\overline{P})$ is the unique fixpoint of $T_{\overline{P}}$.*

Proof. Let α be an annotation over the well-founded ordering $\mathcal{W} = (W, \succ)$ such that $\overline{P} = \alpha(P)$ and $\overline{P}$ is decreasing w.r.t. $\succ$. Assume that I and J are fixpoints of $T_{\overline{P}}$. By well-founded induction on $\succ$ we prove that: for every ground annotated atom $\overline{A}$, we have that $\overline{A} \in I$ iff $\overline{A} \in J$. The inductive hypothesis is the following: for every ground annotated atom $\overline{B}$, if $\mathcal{W} \models \overline{A} \succ \overline{B}$ then $\overline{B} \in I$ iff $\overline{B} \in J$. Assume that $\overline{A} \in I$. Since $I = T_{\overline{P}}(I)$, we have that there exists a clause of the form $\overline{A} \leftarrow c \wedge \overline{A}_1 \wedge \ldots \wedge \overline{A}_n$ in $\overline{P}$ such that $\mathcal{W} \models c$ and, for $i = 1, \ldots, n$, $\overline{A}_i \in I$. Since $\overline{P}$ is decreasing w.r.t. $\succ$, we have that, by definition, for $i = 1, \ldots, n$, $\mathcal{W} \models \overline{A} \succ \overline{A}_i$. Therefore, by the inductive hypothesis, for $i = 1, \ldots, n$, we have that $\overline{A}_i \in J$. Since J is a fixpoint of $T_{\overline{P}}$ and $\mathcal{W} \models c$, we get that $\overline{A} \in J$. Thus, we have proved that if $\overline{A} \in I$ then $\overline{A} \in J$. Similarly, we can prove that if $\overline{A} \in J$ then $\overline{A} \in I$. Thus, $T_{\overline{P}}$ has a unique fixpoint, which is equal to its least fixpoint $M(\overline{P})$. ∎

The notions introduced in Definition 2.1 (*if*-form of a set of clauses), Definition 2.2 (implication, reverse-implication, and equivalence between sets of clauses), and Definition 2.6 (implication-based and reverse-implication-based program sequence) can be extended to annotated clauses and annotated programs by simply considering annotated formulas, instead of formulas, and $\mathcal{W}$-interpretations, instead of Herbrand interpretations. For reasons of brevity, we will not present these definitions in the cases of annotated clauses and annotated programs and, instead, we will refer, also in these cases, to Definitions 2.1, 2.2, and 2.6 given above. The context will tell the reader whether these definitions are used in the annotated case or in the ordinary case. We have that the properties stated by Lemmata 2.3 and 2.8, and Theorems 2.10 and 2.13, hold for annotated programs as well.

Unlike the above mentioned notions, the clause replacement rule will be re-defined in the case of annotated programs (see Definition 3.8 below), and the results concerning the total correctness of the transformation sequences constructed by applying this rule will be given later on.

We are now able to prove a sufficient condition for a sequence $P_0 \mapsto \ldots \mapsto P_n$ of programs to be totally correct. This condition is based on the existence of a sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ of annotated programs such that $\overline{P}_n$ is decreasing with respect to a given well-founded ordering and, thus, by Theorem 3.6, $\overline{P}_n$ is univocal. This sufficient condition will be used to prove that any transformation sequence constructed by applying the clause replacement rule for annotated programs is totally correct.

Theorem 3.7 (Total Correctness via Well-Founded Annotations) *Let $\overline{P}_0, \ldots, \overline{P}_n$ be annotated programs over a well-founded ordering $(W, \succ)$. Suppose that:*
(i) *$P_0 \mapsto \ldots \mapsto P_n$ is an implication-based sequence of programs,*
(ii) *$\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ is a reverse-implication-based sequence of programs, and*
(iii) *$\overline{P}_n$ is decreasing w.r.t. $\succ$.*
Then:
(1) *$M(\overline{P}_0) \subseteq M(\overline{P}_n)$, and*
(2) *the sequence $P_0 \mapsto \ldots \mapsto P_n$ of programs is totally correct, that is, $M(P_0) = M(P_n)$.*

Proof. (Point 1) By Hypothesis (iii), $\overline{P}_n$ is decreasing w.r.t. $\succ$, and thus, it follows from Theorem 3.6 that $\overline{P}_n$ is univocal. Since, by Hypothesis (ii), $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ is a reverse-implication-based sequence of programs, by Theorem 2.13 we get that $M(\overline{P}_0) \subseteq M(\overline{P}_n)$.
(Point 2) By Hypothesis (i) and Theorem 2.10, $P_0 \mapsto \ldots \mapsto P_n$ is partially correct, that is, $M(P_0) \supseteq M(P_n)$. Now it remains to prove that $P_0 \mapsto \ldots \mapsto P_n$ is conservative, that is, $M(P_0) \subseteq M(P_n)$. Let A be a ground atom in $M(P_0)$. By Proposition 3.4 there exists a ground annotation term w such that $A\langle w \rangle$ belongs to $M(\overline{P}_0)$. By Point (1), we have that $M(\overline{P}_0) \subseteq M(\overline{P}_n)$. Thus, $A\langle w \rangle$ belongs to $M(\overline{P}_n)$ and, by Proposition 3.2, A belongs to $M(P_n)$. ∎

Note that the annotated programs $\overline{P}_0, \ldots, \overline{P}_{n-1}$ are *not* required to be decreasing, while $\overline{P}_n$ is required to be decreasing. In practice, however, it is often useful to start from an annotated program $\overline{P}_0$ which is decreasing w.r.t. a given well-founded ordering $\succ$, and to apply the following clause replacement rule which acts on annotated programs so that the decreasingness w.r.t. $\succ$ is preserved and, thus, the final annotated program $\overline{P}_n$ is decreasing by construction. In the definition below we assume that the notion of transformation sequence for annotated programs is the obvious extension of the notion given in Definition 2.4 for ordinary programs.

Definition 3.8 (Clause Replacement Rule for Annotated Programs) Let $\succ$ be a well-founded ordering. Let us consider a transformation sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_k$, for any $k \geq 0$, such that, for $i = 0, \ldots, k$, the annotated program $\overline{P}_i$ is decreasing w.r.t. $\succ$. Let $\overline{\Gamma}_k$ be a set of clauses for a predicate p such that $\overline{\Gamma}_k \subseteq \overline{P}_k$, and let $\overline{\Delta}_k$ be a set of annotated clauses for p such that:
(i) $M(P_0) \models \Gamma_k \Leftarrow \Delta_k$,
(ii) $M(\overline{P}_0) \models \overline{\Gamma}_k \Rightarrow \overline{\Delta}_k$, and
(iii) $\overline{\Delta}_k$ is decreasing w.r.t. $\succ$.
By applying the clause replacement rule we derive the program $\overline{P}_{k+1} = (\overline{P}_k - \overline{\Gamma}_k) \cup \overline{\Delta}_k$ and we derive the transformation sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_k \mapsto \overline{P}_{k+1}$.

We would like to note that, if $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ is a transformation sequence of annotated programs, then the sequence $P_0 \mapsto \ldots \mapsto P_n$ obtained by applying the projection function π to each program of the given sequence, is a transformation sequence of ordinary programs. Indeed, by the following Lemma 3.9 (whose proof is given in the Appendix), Point (ii) of Definition 3.8 implies $M(P_0) \models \Gamma_k \Rightarrow \Delta_k$ and, thus, by Point (i) of the same definition, we have that $P_0 \mapsto \ldots \mapsto P_n$ is an equivalence-based sequence of programs.

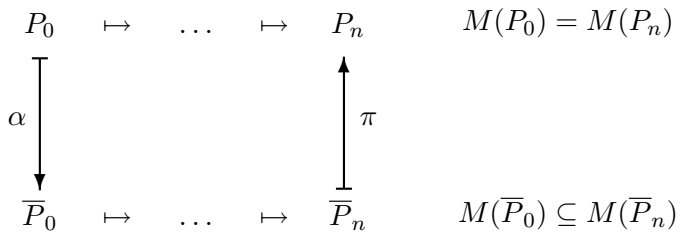

Figure 1: Program Transformation via Well-Founded Annotations. α is a well-founded annotation w.r.t. a given well-founded ordering. π is the projection function.

Lemma 3.9. *Let $\overline{P}$ be an annotated program and let $\overline{\Gamma}_1$ and $\overline{\Gamma}_2$ be sets of annotated clauses. If $M(\overline{P}) \models \overline{\Gamma}_1 \Rightarrow \overline{\Gamma}_2$ then $M(P) \models \Gamma_1 \Rightarrow \Gamma_2$.*

The transformation sequence $P_0 \mapsto \ldots \mapsto P_n$ constructed by applying the projection function π to the programs of a transformation sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ is totally correct. Indeed, by Point (i) of Definition 3.8, $P_0 \mapsto \ldots \mapsto P_n$ is an implication-based sequence of programs, by Point (ii) of Definition 3.8, $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ is a reverse-implication-based sequence of programs, and by Point (iii) of Definition 3.8, $\overline{P}_n$ is decreasing. Hence, by Theorem 3.7 we have that $M(\overline{P}_0) \subseteq M(\overline{P}_n)$ and $M(P_0) = M(P_n)$. Thus, we have proved the following total correctness result for transformation sequences constructed by using the clause replacement rule.

Corollary 3.10 (Total Correctness of Transformation Sequences) *Let $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ be a transformation sequence constructed by applying the clause replacement rule of Definition 3.8. Then:*
(1) *$M(\overline{P}_0) \subseteq M(\overline{P}_n)$, and*
(2) *$M(P_0) = M(P_n)$.*

Note that in the transformation sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ of Corollary 3.10, the programs $P_0, \ldots, P_n$ are not required to be univocal and, in particular, they are not required to be terminating.

Corollary 3.10 supports a methodology for program transformation which consists of the following steps (see also Figure 1). Given an initial program P_0, in order to derive a program P_n such that $M(P_0) = M(P_n)$,

(1) first, we choose a well-founded ordering $(W, \succ)$ and an annotation α which is well-founded w.r.t. $\succ$ (thus, the annotated program $\overline{P}_0 = \alpha(P_0)$ is decreasing w.r.t. $\succ$);

(2) then, we construct a transformation sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ where, for $k = 0, \ldots, n-1$, $\overline{P}_{k+1}$ is derived from $\overline{P}_k$ by applying the clause replacement rule of Definition 3.8 and replacing a set $\overline{\Gamma}_k$ of annotated clauses in $\overline{P}_k$ by a new set $\overline{\Delta}_k$, such that every clause in $\overline{\Delta}_k$ is decreasing w.r.t. $\succ$; and

(3) finally, we apply the projection π to $\overline{P}_n$, thereby erasing the annotation terms and the annotation formulas from $\overline{P}_n$.

We conclude this section by giving an example of application of Corollary 3.10. Note that the total correctness of the transformation sequence considered in this example cannot be shown by using in a straightforward way the results of Section 2.

Example 9. Let us consider the following program R_1:

C_1: $reach(X, X) \leftarrow$
C_2: $reach(X, Z) \leftarrow reach(X, Y) \wedge arc(Y, Z)$
C_3: $arc(a, a) \leftarrow$
C_4: $arc(b, b) \leftarrow$

The following equivalence holds:

$M(R_1) \models \forall X \forall Z \, (\exists Y \, (reach(X, Y) \wedge arc(Y, Z)) \leftrightarrow \exists Y \, (arc(X, Y) \wedge reach(Y, Z)))$

Hence, the following clause:

C_5: $reach(X, Z) \leftarrow arc(X, Y) \wedge reach(Y, Z)$

is equivalent to clause C_2 in $M(R_1)$, that is, $M(R_1) \models \{C_2\} \Leftrightarrow \{C_5\}$. Thus, by applying the clause replacement rule for ordinary programs (see Definition 2.5) we get the transformation sequence $R_1 \mapsto R_2$, where R_2 is the program derived from R_1 by replacing clause C_2 by clause C_5. Unfortunately, we cannot apply Corollary 2.14 of Section 2 to prove the total correctness of $R_1 \mapsto R_2$, because R_2 is not univocal. Indeed, $reach(a, b) \in gfp(T_{R_2})$ and $reach(a, b) \notin lfp(T_{R_2})$ and, therefore, $lfp(T_{R_2}) \neq gfp(T_{R_2})$.

Let us now use the method based on well-founded annotations to prove the total correctness of $R_1 \mapsto R_2$. We consider the following annotated program $\overline{R}_1$, obtained by applying the annotation α_1 considered in Example 7:

$\overline{C}_1$: $reach(X, X)\langle K \rangle \leftarrow$
$\overline{C}_2$: $reach(X, Z)\langle K \rangle \leftarrow K > M+N \wedge reach(X, Y)\langle M \rangle \wedge arc(Y, Z)\langle N \rangle$
$\overline{C}_3$: $arc(a, a)\langle K \rangle \leftarrow$
$\overline{C}_4$: $arc(b, b)\langle K \rangle \leftarrow$

$\overline{R}_1$ is decreasing w.r.t. $>$. We have that:

$$M(\overline{R}_1) \models \forall K \forall X \forall Z \, (\, \exists M \exists N \exists Y \, (K > M+N \wedge reach(X, Y)\langle M \rangle \wedge arc(Y, Z)\langle N \rangle) \leftrightarrow \exists M \exists N \exists Y \, (K > M+N \wedge arc(X, Y)\langle M \rangle \wedge reach(Y, Z)\langle N \rangle))$$

and, therefore, $M(\overline{R}_1) \models \{\overline{C}_2\} \Leftrightarrow \{\overline{C}_5\}$, where $\overline{C}_5$ is the following annotated clause:

$\overline{C}_5$: $reach(X, Z)\langle K \rangle \leftarrow K > M+N \wedge arc(X, Y)\langle M \rangle \wedge reach(Y, Z)\langle N \rangle$

Moreover, $\overline{C}_5$ is decreasing w.r.t. $>$ and, thus, by applying the clause replacement rule of Definition 3.8 we derive the transformation sequence $\overline{R}_1 \mapsto \overline{R}_2$, where $\overline{R}_2 = (\overline{R}_1 - \{\overline{C}_2\}) \cup \{\overline{C}_5\}$. By Corollary 3.10, $R_1 \mapsto R_2$ is totally correct. □

4. Unfold/Fold Transformation Rules for Annotated Programs

In this section we use the results presented in Section 3 to prove the total correctness of program sequences constructed by applying suitable variants of the usual unfolding, folding, and goal replacement rules. These three rules are collectively called *unfold/fold transformation rules* and the program sequences constructed by using the unfold/fold transformation rules are called *unfold/fold transformation sequences*. We will show that the unfold/fold transformation rules are instances of the clause replacement rule for annotated programs presented in Definition 3.8, and thus, the total correctness of unfold/fold transformation sequences is guaranteed by Corollary 3.10. In particular, a totally correct unfold/fold transformation sequence from program P_0 to program P_n can be constructed, by following the three-step methodology described at the end of Section 3, as follows:

(1) first, we choose a well-founded ordering $(W, \succ)$ and an annotation α over $(W, \succ)$, such that the annotated program $\overline{P}_0 = \alpha(P_0)$ is decreasing w.r.t. $\succ$;

(2) then, we construct an unfold/fold transformation sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ where, for $k = 0, \ldots, n-1$, program $\overline{P}_{k+1}$ is derived from program $\overline{P}_k$ by applying any one of the unfold/fold transformation rules for annotated programs (which, as shown below, preserve decreasingness); and
(3) finally, we apply the projection π to $\overline{P}_n$ thereby erasing the annotation terms and the annotation formulas from $\overline{P}_n$.

The unfold/fold transformation rules for annotated programs are very similar to the unfold/fold transformation rules for ordinary programs and, indeed, for any transformation sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ constructed by applying the unfold/fold transformation rules for annotated programs, the transformation sequence $P_0 \mapsto \ldots \mapsto P_n$ can be constructed by applying the usual unfold/fold transformation rules for ordinary programs. However, the unfold/fold transformation rules for annotated programs also ensure that decreasingness is preserved. Indeed, if $\overline{\Gamma}_k$ is a set of clauses which are decreasing w.r.t. $\succ$, then the clauses in the set $\overline{\Delta}_k$ derived from $\overline{\Gamma}_k$ by applying a transformation rule are all decreasing w.r.t. $\succ$, whenever suitable applicability conditions based on the annotation formulas hold.

Now we show how the unfold/fold transformation sequences are constructed by using unfold/fold transformation rules for annotated programs. Given an unfold/fold transformation sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_k$, for some $k \geq 0$, program $\overline{P}_{k+1}$ is derived from program $\overline{P}_k$ by applying one of the three transformation rules: R1 (unfolding), R2 (folding), and R3 (goal replacement), which are defined below. As already mentioned is Section 2, for reasons of simplicity, among the transformation rules here we do not include the *definition introduction* rule [28]. This simplifying assumption is made also in [11, 24, 29].

The presentation of the transformation rules is parametric with respect to an arbitrarily chosen well-founded ordering $\mathcal{W} = (W, \succ)$. We assume that the initial annotated program $\overline{P}_0$ of the unfold/fold transformation sequence $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_k$ is annotated over $\mathcal{W}$ and is decreasing with respect to $\succ$. We also assume that $\overline{P}_0$ and $\overline{P}_k$ have no variables in common. This assumption is not restrictive because we can always rename the variables of an annotated program without changing its least $\mathcal{W}$-model. In fact, we will feel free to rename variables whenever needed.

Rule R1 (Unfolding) Let $\overline{C}$: $\overline{H} \leftarrow c \wedge \overline{G}_L \wedge \overline{A} \wedge \overline{G}_R$ be a clause of the annotated program $\overline{P}_k$. Let

$$\begin{array}{ll} \overline{C}_1: & \overline{H}_1 \leftarrow c_1 \wedge \overline{G}_1 \\ & \ldots \\ \overline{C}_m: & \overline{H}_m \leftarrow c_m \wedge \overline{G}_m \end{array}$$

with $m \geq 0$, be all clauses of program $\overline{P}_0$ such that, for $i = 1, \ldots, m$, $\overline{A}$ is unifiable with $\overline{H}_i$ via a most general unifier ϑ_i. By *unfolding clause* $\overline{C}$ *w.r.t. atom* $\overline{A}$ we derive the clauses

$$\begin{array}{ll} \overline{D}_1: & (\overline{H} \leftarrow c \wedge c_1 \wedge \overline{G}_L \wedge \overline{G}_1 \wedge \overline{G}_R)\vartheta_1 \\ & \ldots \\ \overline{D}_m: & (\overline{H} \leftarrow c \wedge c_m \wedge \overline{G}_L \wedge \overline{G}_m \wedge \overline{G}_R)\vartheta_m \end{array}$$

and from program $\overline{P}_k$ we derive program $\overline{P}_{k+1} = (\overline{P}_k - \{\overline{C}\}) \cup \{\overline{D}_1, \ldots, \overline{D}_m\}$.

Basically, the unfolding rule for annotated programs is like the usual, totally correct unfolding rule for definite logic programs. Note, however, that we cannot unfold an annotated clause with respect to an annotation formula (such as c in clause $\overline{C}$ above), but only with respect to an annotated atom. The following lemma, whose proof is given in the Appendix, shows that the unfolding rule is a particular case of the clause replacement rule for annotated programs.

Lemma 4.1. *Let $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_k$ be an unfold/fold transformation sequence whose programs are annotated over the well-founded ordering $\mathcal{W} = (W, \succ)$. Let $\overline{C}$ be a clause in the annotated program $\overline{P}_k$, and let $\overline{D}_1, \ldots, \overline{D}_m$ be the clauses derived by unfolding $\overline{C}$ w.r.t. an annotated atom in its body, as described in Rule* R1. *Then:*
(1) $M(\overline{P}_0) \models \{\overline{C}\} \Leftrightarrow \{\overline{D}_1, \ldots, \overline{D}_m\}$, *and*
(2) $\overline{D}_1, \ldots, \overline{D}_m$ *are decreasing w.r.t.* $\succ$.

Note that, by Lemma 3.9, Point (1) of Lemma 4.1 implies $M(P_0) \models \{C\} \Rightarrow \{D_1, \ldots, D_m\}$, that is, Condition (i) of Definition 3.8.

Rule R2 (Folding) Let

$$\begin{array}{ll} \overline{C}_1: & \overline{H} \leftarrow c_1 \wedge \overline{G}_1 \\ & \ldots \\ \overline{C}_m: & \overline{H} \leftarrow c_m \wedge \overline{G}_m \end{array}$$

with $m \geq 1$, be clauses in $\overline{P}_0$ and, for a substitution ϑ, let

$$\begin{array}{ll} \overline{D}_1: & \overline{K} \leftarrow d \wedge c_1\vartheta \wedge \overline{G}_L \wedge \overline{G}_1\vartheta \wedge \overline{G}_R \\ & \ldots \\ \overline{D}_m: & \overline{K} \leftarrow d \wedge c_m\vartheta \wedge \overline{G}_L \wedge \overline{G}_m\vartheta \wedge \overline{G}_R \end{array}$$

be clauses in $\overline{P}_k$. Suppose that the following conditions hold:

1. there exists no clause in $\overline{P}_0 - \{\overline{C}_1, \ldots, \overline{C}_m\}$ whose head is unifiable with $\overline{H}\vartheta$;
2. for $i = 1, \ldots, m$ and for every variable U in the set $vars(c_i \wedge \overline{G}_i) - vars(\overline{H})$: (i) $U\vartheta$ is a variable not occurring in $\{\overline{K}, d, \overline{G}_L, \overline{G}_R\}$, and (ii) $U\vartheta$ does not occur in the term $V\vartheta$, for any variable V occurring in $c_i \wedge \overline{G}_i$ and different from U; and
3. $\mathcal{W} \models \forall\, (d \rightarrow \overline{K} \succ (\overline{G}_L \wedge \overline{H}\vartheta \wedge \overline{G}_R))$.

By *folding clauses* $\overline{D}_1, \ldots, \overline{D}_m$ *using clauses* $\overline{C}_1, \ldots, \overline{C}_m$ we derive the clause
$\overline{E}$: $\overline{K} \leftarrow d \wedge \overline{G}_L \wedge \overline{H}\vartheta \wedge \overline{G}_R$
and from program $\overline{P}_k$ we derive program $\overline{P}_{k+1} = (\overline{P}_k - \{\overline{D}_1, \ldots, \overline{D}_m\}) \cup \{\overline{E}\}$.

The only difference between the folding rule for annotated programs and the usual, partially correct folding rule for definite logic programs consists in the extra Condition 3. This extra condition ensures that the annotated clause $\overline{E}$ derived by folding is decreasing w.r.t. $\succ$.

The following Lemma 4.2 (together with Lemma 3.9) shows that, like the unfolding rule, also the folding rule is a particular case of the clause replacement rule for annotated programs. The proof is given in the Appendix.

Lemma 4.2. *Let $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_k$ be an unfold/fold transformation sequence whose programs are annotated over the well-founded ordering $\mathcal{W} = (W, \succ)$. Let $\overline{C}_1, \ldots, \overline{C}_m$ be clauses in $\overline{P}_0$, let $\overline{D}_1, \ldots, \overline{D}_m$ be clauses in $\overline{P}_k$, and let $\overline{E}$ be the clause derived by folding $\overline{D}_1, \ldots, \overline{D}_m$ using $\overline{C}_1, \ldots, \overline{C}_m$, as described in Rule* R2. *Then:*
(1) $M(\overline{P}_0) \models \{\overline{D}_1, \ldots, \overline{D}_m\} \Leftrightarrow \{\overline{E}\}$, *and*
(2) $\overline{E}$ *is decreasing w.r.t.* $\succ$.

The goal replacement rule for annotated programs consists in replacing a conjunction of the form $c_1 \wedge \overline{G}_1$ occurring in the body of a clause of $\overline{P}_k$, by a new conjunction of the form $c_2 \wedge \overline{G}_2$. As already mentioned, we want to present this goal replacement rule as a particular case of the clause replacement rule for annotated programs, so that the total correctness of the unfold/fold

transformation sequences will easily follow from Corollary 3.10. Thus, Conditions (i), (ii), and (iii) of Definition 3.8 should be satisfied when performing a goal replacement.

Condition (i) and (ii) of Definition 3.8 are ensured by suitably quantified implications which correspond to Conditions (i) and (ii) of the definition of *replacement law* (see Definition 4.3 below). Condition (iii) of Definition 3.8 is ensured by Condition (δ) of the goal replacement rule R3 below.

In Definition 4.3 we will use the following notation. Given a set $X = \{X_1, \ldots, X_m\}$ of variables, $\forall X$ is a shorthand for $\forall X_1 \ldots \forall X_m$ and analogously for $\exists X$.

Definition 4.3 (Replacement Law) Let c_1, c_2 be annotation formulas, let $\overline{G}_1, \overline{G}_2$ be annotated goals, and let $X \subseteq vars(\{c_1, c_2, \overline{G}_1, \overline{G}_2\})$ be a set of variables. We say that the *replacement law* $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ holds in $\overline{P}_0$ iff the following conditions hold:
(i) $M(P_0) \models \forall X' \, (\exists Y' \, G_1 \leftarrow \exists Z' \, G_2)$ and
(ii) $M(\overline{P}_0) \models \forall X \, (\exists Y \, (c_1 \wedge \overline{G}_1) \rightarrow \exists Z \, (c_2 \wedge \overline{G}_2))$
where: (1) G_1 and G_2 are the goals $\pi(\overline{G}_1)$ and $\pi(\overline{G}_2)$, respectively, (2) $X' = X \cap vars(\{G_1, G_2\})$, (3) $Y' = vars(G_1) - X'$, (4) $Z' = vars(G_2) - X'$, (5) $Y = vars(c_1 \wedge \overline{G}_1) - X$, and (6) $Z = vars(c_2 \wedge \overline{G}_2) - X$.

By using Lemma 3.9 and Condition (ii) of Definition 4.3, it can be shown that $M(P_0) \models \forall X' \, (\exists Y' \, G_1 \leftrightarrow \exists Z' \, G_2)$, which corresponds to one of the applicability conditions of the goal replacement rule for definite programs given in [28].

Let us consider again the program R_1 given in Example 9 and its annotated version $\overline{R}_1$. The following replacement law holds in the annotated program $\overline{R}_1$:

$$K > M + N \wedge reach(X, Y)\langle M \rangle \wedge arc(Y, Z)\langle N \rangle \Rightarrow_{\{K,X,Z\}} K > M + N \wedge arc(X, Y)\langle M \rangle \wedge reach(Y, Z)\langle N \rangle \quad \text{(Swap)}$$

Indeed, as already mentioned, we have that:

$$M(\overline{R}_1) \models \forall K \, \forall X \, \forall Z \, (\, \exists M \, \exists N \, \exists Y \, (K > M + N \wedge reach(X, Y)\langle M \rangle \wedge arc(Y, Z)\langle N \rangle) \rightarrow \exists M \, \exists N \, \exists Y \, (K > M + N \wedge arc(X, Y)\langle M \rangle \wedge reach(Y, Z)\langle N \rangle))$$

and

$$M(R_1) \models \forall X \, \forall Z \, (\exists Y \, (reach(X, Y) \wedge arc(Y, Z)) \leftarrow \exists Y \, (arc(X, Y) \wedge reach(Y, Z))).$$

In the next section we will present a method, called *unfold/fold proof method*, for proving that a replacement law holds in an annotated program.

Rule R3 (Goal Replacement) Let $\overline{C}$: $\overline{H} \leftarrow c \wedge c_1 \wedge \overline{G}_L \wedge \overline{G}_1 \wedge \overline{G}_R$ be a clause of the annotated program $\overline{P}_k$ and suppose that the replacement law λ: $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ holds in $\overline{P}_0$, where $X = vars(\{\overline{H}, c, \overline{G}_L, \overline{G}_R\}) \cap vars(\{c_1, \overline{G}_1, c_2, \overline{G}_2\})$. Suppose also that:

$$\mathcal{W} \models \forall \, ((c \wedge c_2) \rightarrow \overline{H} \succ (\overline{G}_L \wedge \overline{G}_2 \wedge \overline{G}_R)) \qquad (\delta)$$

By *goal replacement using law* λ, from clause $\overline{C}$ we derive the clause $\overline{D}$: $\overline{H} \leftarrow c \wedge c_2 \wedge \overline{G}_L \wedge \overline{G}_2 \wedge \overline{G}_R$ and from program $\overline{P}_k$ we derive program $\overline{P}_{k+1} = (\overline{P}_k - \{\overline{C}\}) \cup \{\overline{D}\}$.

The goal replacement rule R3 for annotated programs differs from the usual, partially correct goal replacement rule for definite logic programs because of Condition (δ). This condition ensures that the annotated clause $\overline{D}$ derived by goal replacement is decreasing w.r.t. $\succ$. (The notion of replacement law for definite logic programs can be obtained from the notion of replacement law for annotated programs by replacing: (i) c_1 and c_2 by *true*, and (ii) $\overline{G}_1$ and $\overline{G}_2$ by G_1 and G_2, respectively. Thus, $X' = X$, $Y' = Y$, and $Z' = Z$).

The following lemma, whose proof is given in the Appendix, shows that also the goal replacement rule is a particular case of the clause replacement rule for annotated programs.

Lemma 4.4. *Let $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_k$ be an unfold/fold transformation sequence whose programs are annotated over the well-founded ordering $\mathcal{W} = (W, \succ)$. Let $\overline{C}$ be a clause in $\overline{P}_k$ and let $\overline{D}$ be a clause derived from $\overline{C}$ by goal replacement, as described in Rule* R3*. Then:*
(1) $M(P_0) \models \{C\} \Rightarrow \{D\}$,
(2) $M(\overline{P}_0) \models \{\overline{C}\} \Leftarrow \{\overline{D}\}$, *and*
(3) $\overline{D}$ *is decreasing w.r.t.* $\succ$.

An application of the goal replacement rule is given in the above Example 9. Indeed, the derivation of clause $\overline{C}_5$ from clause $\overline{C}_2$ can be viewed as an application of Rule R3 based on the fact that the above replacement law (Swap) holds in $\overline{R}_1$.

Note that a particular application of the goal replacement rule allows us to replace in the body of a clause of the form $\overline{H} \leftarrow c \wedge c_1 \wedge \overline{G}$ the annotation formula c_1 by an annotation formula c_2 provided that the following two conditions are satisfied:

(A1) $\mathcal{W} \models \forall X\, (\exists Y\, c_1 \rightarrow \exists Z\, c_2)$

where: (i) $X = vars(\{\overline{H}, c, \overline{G}\}) \cap vars(\{c_1, c_2\})$, (ii) $Y = vars(c_1) - X$, and (iii) $Z = vars(c_2) - X$, and

(A2) $\mathcal{W} \models \forall\, ((c \wedge c_2) \rightarrow \overline{H} \succ \overline{G})$.

Indeed, Conditions (A1) and (A2) imply Conditions (i) and (ii) of Definition 4.3 and Condition (δ) of Rule R3, when both $\overline{G}_1$ and $\overline{G}_2$ are the empty conjunction *true* and $\overline{G}_L \wedge \overline{G}_R$ is $\overline{G}$. This particular replacement of annotation formulas will be called *annotation weakening*. For instance, by annotation weakening, the clause

$p(A)\langle X\rangle \leftarrow X > Y \wedge Y > Z \wedge q(A)\langle Z\rangle$

can be replaced by the clause

$p(A)\langle X\rangle \leftarrow X > Z \wedge q(A)\langle Z\rangle$

The following example illustrates the construction of an unfold/fold transformation sequence of annotated programs.

Example 10. Let us consider the following annotated program $\overline{P}_0$, where we use the well-founded annotation α_2 over $\mathcal{N}$ of Example 7:

1. $p(a)\langle X\rangle \leftarrow$
2. $p(A)\langle X\rangle \leftarrow X > X_1 \wedge X > X_2 \wedge t(A,B)\langle X_1\rangle \wedge p(B)\langle X_2\rangle$
3. $q(b)\langle X\rangle \leftarrow$
4. $q(A)\langle X\rangle \leftarrow X > X_1 \wedge r(A)\langle X_1\rangle$
5. $r(A)\langle X\rangle \leftarrow X > X_1 \wedge X > X_2 \wedge t(A,B)\langle X_1\rangle \wedge q(B)\langle X_2\rangle$
6. $s(A)\langle X\rangle \leftarrow X > X_1 \wedge p(A)\langle X_1\rangle$
7. $s(A)\langle X\rangle \leftarrow X > X_1 \wedge q(A)\langle X_1\rangle$

By unfolding clause 6 w.r.t. $p(A)\langle X_1\rangle$, we get:

8. $s(a)\langle Y\rangle \leftarrow Y > Y_1$
9. $s(C)\langle Y\rangle \leftarrow Y > Y_1 \wedge Y_1 > Y_2 \wedge Y_1 > Y_3 \wedge t(C,D)\langle Y_2\rangle \wedge p(D)\langle Y_3\rangle$

Thus, $\overline{P}_1$ is $(\overline{P}_0 - \{6\}) \cup \{8, 9\}$. By two applications of the unfolding rule, from clause 7 we derive:

10. $s(b)\langle Y\rangle \leftarrow Y > Y_1$
11. $s(C)\langle Y\rangle \leftarrow Y > Z \wedge Z > Y_1 \wedge Y_1 > Y_2 \wedge Y_1 > Y_3 \wedge t(C,D)\langle Y_2\rangle \wedge q(D)\langle Y_3\rangle$

Thus, $\overline{P}_3$ is $(\overline{P}_0 - \{6, 7\}) \cup \{8, 9, 10, 11\}$. Now, by annotation weakening from clauses 9 and 11 we derive:

12. $s(C)\langle Y\rangle \leftarrow Y > Y_2 \wedge Y > Y_1 \wedge Y_1 > Y_3 \wedge t(C,D)\langle Y_2\rangle \wedge p(D)\langle Y_3\rangle$
13. $s(C)\langle Y\rangle \leftarrow Y > Y_2 \wedge Y > Y_1 \wedge Y_1 > Y_3 \wedge t(C,D)\langle Y_2\rangle \wedge q(D)\langle Y_3\rangle$

By the above two transformation steps we get program $\overline{P}_5$ which is $(\overline{P}_0 - \{6,7\}) \cup \{8,12,10,13\}$. Now, Conditions 1–3 of the folding rule are verified by taking: (i) ϑ to be the substitution $\{X/Y_1,\ X_1/Y_3,\ A/D\}$, and (ii) d to be the annotation formula $Y > Y_2 \wedge Y > Y_1$. By folding clauses 12 and 13 using clauses 6 and 7 we derive the clause:

14. $s(C)\langle Y\rangle \leftarrow Y > Y_2 \wedge Y > Y_1 \wedge t(C,D)\langle Y_2\rangle \wedge s(D)\langle Y_1\rangle$.

which is decreasing w.r.t. $>$. The final program is $\overline{P}_6 = (\overline{P}_0 - \{6,7\}) \cup \{8,10,14\}$. By Theorem 4.5 we have that $M(P_0) = M(P_6)$, where $P_0 = \pi(\overline{P}_0)$ and $P_6 = \pi(\overline{P}_6)$. □

The total correctness of the transformation sequences constructed by applying Rules R1, R2, and R3 readily follows from the fact that these rules are particular cases of the clause replacement rule for annotated programs and Corollary 3.10.

Theorem 4.5 (Total Correctness of Unfold/Fold Transformation Sequences) *Let $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ be an unfold/fold transformation sequence. Then:*
(1) $M(\overline{P}_0) \subseteq M(\overline{P}_n)$, *and*
(2) $M(P_0) = M(P_n)$.

Proof. For $k = 0, \ldots, n-1$, the annotated program $\overline{P}_{k+1}$ is derived from $\overline{P}_k$ by the application of a transformation rule among R1, R2, and R3. Then, for some sets $\overline{\Gamma}_k$ and $\overline{\Delta}_k$ of annotated clauses, we have that $\overline{P}_{k+1} = (\overline{P}_k - \overline{\Gamma}_k) \cup \overline{\Delta}_k$. By Lemmata 4.1, 4.2, and 4.4, we have the following properties:
(1) $M(P_0) \models \Gamma_k \Rightarrow \Delta_k$,
(2) $M(\overline{P}_0) \models \overline{\Gamma}_k \Leftarrow \overline{\Delta}_k$, and
(3) $\overline{\Delta}_k$ is decreasing w.r.t. $\succ$.
Thus, $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ can be viewed as a transformation sequence constructed by using the clause replacement rule of Definition 3.8 and, by Corollary 3.10, we have that: (1) $M(\overline{P}_0) \subseteq M(\overline{P}_n)$, and (2) $M(P_0) = M(P_n)$. ∎

5. Unfold/Fold Proofs of Replacement Laws

In this section we describe a method which can be used to prove that a replacement law holds in an annotated program. (Recall that, in order to apply the goal replacement rule, we have to show that a suitable replacement law does hold.) Our method constructs the proof of the given replacement law by using the transformation rules presented in Section 4 and, thus, it is an extension to annotated programs of the unfold/fold proof method presented in [20] in the case of ordinary programs. We will use the term *unfold/fold proof method* also for the method for annotated programs presented in this section. The basic idea behind the unfold/fold proof method is that when we transform a program into a new one by using semantics preserving rules, we also prove an implication (or equivalence) between predicate definitions with respect to the given semantics.

Let us consider the replacement law λ: $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$. For reasons of simplicity, we assume that $X = \{V, N\}$ where V is an ordinary variable and N is an annotation variable. The generalization to the case where V and N are tuples of variables, instead of single variables, is straightforward. In order to prove that λ holds in the annotated program $\overline{P}$, the unfold/fold

proof method works as follows. First we introduce two new predicates *new*1 and *new*2 defined by the following two clauses:

$\overline{D}_1$: $new1(V)\langle N\rangle \leftarrow c_1 \wedge \overline{G}_1$
$\overline{D}_2$: $new2(V)\langle N\rangle \leftarrow c_2 \wedge \overline{G}_2$

Then we construct two unfold/fold transformation sequences of the forms:

$\overline{P} \cup \{\overline{D}_1\} \mapsto \ldots \mapsto \overline{Q}$
$\overline{P} \cup \{\overline{D}_2\} \mapsto \ldots \mapsto \overline{R}$

such that the following two conditions hold:

1. Program $\overline{R}$ can be obtained from program $\overline{Q}$ by renamings of predicates and variables;
2. The unfold/fold transformation sequence $\overline{P} \cup \{\overline{D}_2\} \mapsto \ldots \mapsto \overline{R}$ is equivalence-based (not only reverse-implication-based, as guaranteed by the use of the transformation rules of Section 4).

Now we introduce the notion of *syntactic equivalence* between programs, which formalizes the above Condition 1. Then we will give a simple condition which ensures that Condition 2 is indeed satisfied.

Given a set *Preds* of predicate symbols, a *predicate renaming* over *Preds* is a bijective mapping $\rho : Preds \rightarrow Preds$. Two annotated programs $\overline{Q}$ and $\overline{R}$ are *syntactically equivalent* if there exist: (i) a variant $\overline{Q}'$ of $\overline{Q}$, and (ii) a predicate renaming ρ over the set of predicate symbols occurring in $\overline{Q} \cup \overline{R}$, such that $\overline{R}$ is obtained from $\overline{Q}'$ by replacing every predicate symbol p occurring in $\overline{Q}$ by $\rho(p)$. Syntactic equivalence implies semantic equivalence, as stated by the following lemma, whose straightforward proof is omitted.

Lemma 5.1. *If program $\overline{Q}$ is syntactically equivalent to program $\overline{R}$ via a predicate renaming ρ then, for every predicate p occurring in $\overline{Q}$, ground ordinary term t, and ground annotation term w, $p(t)\langle w\rangle \in M(\overline{Q})$ iff $\rho(p)(t)\langle w\rangle \in M(\overline{R})$.*

Let us now introduce a restricted version of the goal replacement rule R3 of Section 4, called *symmetric goal replacement*, such that every unfold/fold transformation sequence constructed by applying the unfolding, folding, and symmetric goal replacement rules, is an equivalence-based transformation sequence. Given an annotated program $\overline{P}$, we say that the replacement law $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ *holds symmetrically* in $\overline{P}$ if in Definition 4.3 Conditions (i) and (ii) are replaced by the following stronger condition:

(ii′) $M(\overline{P}) \models \forall X\, (\exists Y\, (c_1 \wedge \overline{G}_1) \leftrightarrow \exists Z\, (c_2 \wedge \overline{G}_2))$

(Recall that, by Lemma 3.9, Condition (ii′) implies Condition (i) of Definition 4.3.) An application of the *symmetric goal replacement rule* consists in an application of the goal replacement rule R3 using a replacement law that holds symmetrically in the initial program $\overline{P}_0$ of the unfold/fold transformation sequence. An unfold/fold transformation sequence is said to be *symmetric* iff it is constructed by applications of the unfolding and folding rules and/or by symmetric applications of the goal replacement rule.

If $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ is a symmetric unfold/fold transformation sequence, then we not only have that $M(P_0) = M(P_n)$, like for any unfold/fold transformation sequence (see Theorem 4.5), but we also have that $M(\overline{P}_0) = M(\overline{P}_n)$, as shown by the following lemma.

Lemma 5.2. *Let $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ be a symmetric unfold/fold transformation sequence. Then $M(\overline{P}_0) = M(\overline{P}_n)$.*

Proof. For $k = 0, \ldots, n-1$, $\overline{P}_{k+1} = (\overline{P}_k - \overline{\Gamma}_k) \cup \overline{\Delta}_k$, where $\overline{\Gamma}_k$ and $\overline{\Delta}_k$ are sets of annotated clauses such that, by Lemmata 4.1, 4.2, 4.4, and Condition (ii′) the following properties hold:
(1) $M(\overline{P}_0) \models \overline{\Gamma}_k \Leftrightarrow \overline{\Delta}_k$, and
(2) $\overline{\Delta}_k$ is decreasing w.r.t. $\succ$.
Thus, $\overline{P}_0 \mapsto \ldots \mapsto \overline{P}_n$ is an equivalence-based sequence of programs and $\overline{P}_n$ is decreasing w.r.t. $\succ$. By Theorems 2.10, 2.13, and 3.6, we have that $M(\overline{P}_0) = M(\overline{P}_n)$. ∎

Now we are able to show the soundness of our unfold/fold proof method.

Theorem 5.3 (Soundness of the Unfold/Fold Proof Method) *Let $\overline{P}$ be an annotated program and let $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ be a replacement law. Let us consider the following two annotated clauses:*

$\overline{D}_1$: $new1(V)\langle N\rangle \leftarrow c_1 \wedge \overline{G}_1$
$\overline{D}_2$: $new2(V)\langle N\rangle \leftarrow c_2 \wedge \overline{G}_2$

where $X = \{V, N\}$, V is an ordinary variable, and N is an annotation variable. Suppose that there exist an unfold/fold transformation sequence of the form:

$\overline{P} \cup \{\overline{D}_1\} \mapsto \ldots \mapsto \overline{Q}$

and a symmetric unfold/fold transformation sequence of the form:

$\overline{P} \cup \{\overline{D}_2\} \mapsto \ldots \mapsto \overline{R}$

such that $\overline{Q}$ is syntactically equivalent to $\overline{R}$. Then the replacement law λ: $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ holds in $\overline{P}$.

Proof. From the definition of replacement law (see Definition 4.3) and from the fact that the predicates $new1$ and $new2$ are defined by clauses $\overline{D}_1$ and $\overline{D}_2$, respectively, it follows that the replacement law λ holds in $\overline{P}$ iff the following two properties hold for every ground ordinary term t and ground annotation term w:

(I1) $new1(t) \in M(P \cup \{D_1\})$ if $new2(t) \in M(P \cup \{D_2\})$
(I2) $new1(t)\langle w\rangle \in M(\overline{P} \cup \{\overline{D}_1\})$ only if $new2(t)\langle w\rangle \in M(\overline{P} \cup \{\overline{D}_2\})$

Now we show that indeed Properties (I1) and (I2) do hold. Since there exists an unfold/fold transformation sequence $\overline{P} \cup \{\overline{D}_1\} \mapsto \ldots \mapsto \overline{Q}$, by Theorem 4.5 we have that $M(P \cup \{D_1\}) = M(Q)$ and $M(\overline{P} \cup \{\overline{D}_1\}) \subseteq M(\overline{Q})$. Thus,

(J1) $new1(t) \in M(P \cup \{D_1\})$ iff $new1(t) \in M(Q)$
(J2) $new1(t)\langle w\rangle \in M(\overline{P} \cup \{\overline{D}_1\})$ only if $new1(t)\langle w\rangle \in M(\overline{Q})$

Since $\overline{Q}$ is syntactically equivalent to $\overline{R}$ via the predicate renaming ρ such that $\rho(new1) = new2$, by Lemma 5.1 we have that:

(K1) $new1(t) \in M(Q)$ iff $new2(t) \in M(R)$
(K2) $new1(t)\langle w\rangle \in M(\overline{Q})$ iff $new2(t)\langle w\rangle \in M(\overline{R})$

Now, since $\overline{P} \cup \{\overline{D}_2\} \mapsto \ldots \mapsto \overline{R}$ is a symmetric unfold/fold transformation sequence, by Lemma 5.2 we have that $M(P \cup \{D_2\}) = M(R)$ and $M(\overline{P} \cup \{\overline{D}_2\}) = M(\overline{R})$. Thus,

(L1) $new2(t) \in M(R)$ iff $new2(t) \in M(P \cup \{D_2\})$
(L2) $new2(t)\langle w\rangle \in M(\overline{R})$ iff $new2(t)\langle w\rangle \in M(\overline{P} \cup \{\overline{D}_2\})$

and we have proved Properties (I1) and (I2). Actually, we have also proved the converse of property (I1). ∎

Note that in the proof of Theorem 5.3 we have indeed used the hypothesis that $\overline{P} \cup \{\overline{D}_2\} \mapsto \ldots \mapsto \overline{R}$ is a symmetric unfold/fold transformation sequence, and thus, it is an equivalence-based transformation sequence. In particular, Property (L2) holds if $\overline{P} \cup \{\overline{D}_2\} \mapsto \ldots \mapsto \overline{R}$ is equivalence-based, but it may not hold for an arbitrary unfold/fold transformation sequence.

As an example of application of the unfold/fold proof method, we prove the replacement law (Swap) considered in Section 4.

Example 11. Let us consider again the annotated program $\overline{R}_1$ given in Example 9 and let us prove that the replacement law (Swap) holds in $\overline{R}_1$. By applying the unfold/fold proof method we introduce the following two clauses:

$\overline{D}_1 : \ new1(X,Z)\langle K\rangle \leftarrow K > M+N \wedge reach(X,Y)\langle M\rangle \wedge arc(Y,Z)\langle N\rangle$
$\overline{D}_2 : \ new2(X,Z)\langle K\rangle \leftarrow K > M+N \wedge arc(X,Y)\langle M\rangle \wedge reach(Y,Z)\langle N\rangle$

Now we construct two symmetric transformation sequences $\overline{R}_1 \cup \{\overline{D}_1\} \mapsto \ldots \mapsto \overline{Q}$ and $\overline{R}_1 \cup \{\overline{D}_2\} \mapsto \ldots \mapsto \overline{R}$, where $\overline{Q}$ and $\overline{R}$ are syntactically equivalent. Note that, actually, for the unfold/fold proof of the replacement law (Swap) it is not needed that $\overline{R}_1 \cup \{\overline{D}_1\} \mapsto \ldots \mapsto \overline{Q}$ be symmetric. Indeed, by constructing two symmetric transformation sequences we prove both (Swap) and the following inverse replacement law:

$$K > M+N \wedge arc(X,Y)\langle M\rangle \wedge reach(Y,Z)\langle N\rangle \Rightarrow_{\{K,X,Z\}} K > M+N \wedge reach(X,Y)\langle M\rangle \wedge arc(Y,Z)\langle N\rangle \qquad \text{(Inv-Swap)}$$

Let us now show how the first transformation sequence $\overline{R}_1 \cup \{\overline{D}_1\} \mapsto \ldots \mapsto \overline{Q}$ is constructed. By unfolding clause $\overline{D}_1$ w.r.t. $reach(X,Y)\langle M\rangle$, by replacing annotation formulas by equivalent ones, and by renaming variables, we derive the following two clauses:

$\overline{E}_1$: $new1(X_1,Z_1)\langle K_1\rangle \leftarrow K_1 > M_1 \wedge arc(X_1,Z_1)\langle M_1\rangle$
$\overline{E}_2$: $new1(X_1,Z_1)\langle K_1\rangle \leftarrow K_1 > M_1+N_1 \wedge M_1 > M_2+N_2 \wedge reach(X_1,Y_1)\langle M_2\rangle \wedge arc(Y_1,Y_2)\langle N_2\rangle \wedge arc(Y_2,Z_1)\langle N_1\rangle$

By folding clause $\overline{E}_2$ using clause $\overline{D}_1$ we derive:

$\overline{E}_3$: $new1(X_1,Z_1)\langle K_1\rangle \leftarrow K_1 > M_1+N_1 \wedge new1(X_1,Y_2)\langle M_1\rangle \wedge arc(Y_2,Z_1)\langle N_1\rangle$

The final program $\overline{Q}$ of the first transformation sequence is $\overline{R}_1 \cup \{\overline{E}_1, E_3\}$. Let us now construct the second transformation sequence $\overline{R}_1 \cup \{\overline{D}_2\} \mapsto \ldots \mapsto \overline{R}$. By unfolding clause $\overline{D}_2$ w.r.t. $reach(Y,Z)\langle N\rangle$, by replacing annotation formulas by equivalent ones, and by renaming variables, we derive the following clauses:

$\overline{F}_1$: $new2(X_1,Z_1)\langle K_1\rangle \leftarrow K_1 > M_1 \wedge arc(X_1,Z_1)\langle M_1\rangle$
$\overline{F}_2$: $new2(X_1,Z_1)\langle K_1\rangle \leftarrow K_1 > M_1+N_1 \wedge M_1 > M_2+N_2 \wedge arc(X_1,Y_1)\langle M_2\rangle \wedge reach(Y_1,Y_2)\langle N_2\rangle \wedge arc(Y_2,Z_1)\langle N_1\rangle$

By folding clause $\overline{F}_2$ using clause $\overline{D}_2$ we derive:

$\overline{F}_3$: $new2(X_1,Z_1)\langle K_1\rangle \leftarrow K_1 > M_1+N_1 \wedge new2(X_1,Y_2)\langle M_1\rangle \wedge arc(Y_2,Z_1)\langle N_1\rangle$

The final program $\overline{R}$ of the second transformation sequence is $\overline{R}_1 \cup \{\overline{F}_1, \overline{F}_3\}$. We have that $\overline{R}$ is syntactically equivalent to $\overline{Q}$ via the predicate renaming that maps $new1$ to $new2$, $new2$ to $new1$, and it is the identity on the other predicate symbols occurring in $\overline{Q} \cup \overline{R}$.

Let us observe that the applications of the goal replacement rule during the construction of the two transformation sequences shown above are symmetric. Indeed, they consist of replacements of annotation formulas by equivalent ones. □

6. An Extended Example

In this section we revisit an example of program transformation taken from [24]. The authors of [24] justify that transformation by a rather intricate proof of the total correctness of the transformation sequences. Now we show that the total correctness of the program transformation of that example can easily be established by our well-founded annotation method. Let us consider the following program P:

1. $thm(X) \leftarrow gen(X) \wedge test(X)$
2. $gen([\,]) \leftarrow$
3. $gen([0|X]) \leftarrow gen(X)$
4. $test(X) \leftarrow canon(X)$
5. $test(X) \leftarrow trans(X,Y) \wedge test(Y)$
6. $canon([\,]) \leftarrow$
7. $canon([1|X]) \leftarrow canon(X)$
8. $trans([0|X],[1|X]) \leftarrow$
9. $trans([1|X],[1|Y]) \leftarrow trans(X,Y)$

where we have that $thm(X)$ holds iff X is a list of 0's that can be transformed into a list of 1's by repeated applications of $trans(X,Y)$. Given the list X, the predicate $trans(X,Y)$ generates the list Y by replacing the leftmost 0 in X by 1.

We want to prove that the formula $\forall X\,(thm(X) \leftrightarrow gen(X))$ is true in the least Herbrand model of program P. As a special case of the unfold/fold proof method, the truth of this formula can be established by constructing a totally correct transformation sequence from program P into a program Q where the predicates *thm* and *gen* are defined by two syntactically equivalent sets of clauses. Let us see how we construct this transformation sequence by applying our rules of Section 4.

Let us consider the well-founded annotation α_1 introduced in Example 7. By applying α_1 we get the following annotated program $\overline{P}$:

1a. $thm(X)\langle N\rangle \leftarrow N > N_1+N_2 \wedge gen(X)\langle N_1\rangle \wedge\ test(X)\langle N_2\rangle$
2a. $gen([\,])\langle N\rangle \leftarrow$
3a. $gen([0|X])\langle N\rangle \leftarrow N > N_1 \wedge gen(X)\langle N_1\rangle$
4a. $test(X)\langle N\rangle \leftarrow N > N_1 \wedge canon(X)\langle N_1\rangle$
5a. $test(X)\langle N\rangle \leftarrow N > N_1+N_2 \wedge trans(X,Y)\langle N_1\rangle \wedge\ test(Y)\langle N_2\rangle$
6a. $canon([\,])\langle N\rangle \leftarrow$
7a. $canon([1|X])\langle N\rangle \leftarrow N > N_1 \wedge canon(X)\langle N_1\rangle$
8a. $trans([0|X],[1|X])\langle N\rangle \leftarrow$
9a. $trans([1|X],[1|Y])\langle N\rangle \leftarrow N > N_1 \wedge trans(X,Y)\langle N_1\rangle$

Now, let us construct a totally correct transformation sequence by using our rules of Section 4. By applying several times the unfolding rule, from clause 1a we derive:

10a. $thm([\,])\langle N\rangle \leftarrow N \geq 3$
11a. $thm([0|X])\langle N\rangle \leftarrow N > N_1+N_2+4 \wedge gen(X)\langle N_1\rangle \wedge\ canon(X)\langle N_2\rangle$
12a. $thm([0|X])\langle N\rangle \leftarrow N > N_1+N_2+N_3+4 \wedge\ gen(X)\langle N_1\rangle \wedge$
$\qquad\qquad trans(X,Y)\langle N_2\rangle \wedge\ test([1|Y])\langle N_3\rangle$

The replacement law

$$test([1|Y])\langle N_3\rangle \Rightarrow_{\{Y,N_3\}} (N_3 \geq N_4 \wedge test(Y)\langle N_4\rangle)$$

holds in $\overline{P}$ (see the unfold/fold proof of this law in the Appendix) and, moreover,

$\mathcal{N} \models \forall\, ((N > N_1 + N_2 + N_3 + 4 \wedge N_3 \geq N_4) \rightarrow (N > N_1 \wedge N > N_2 \wedge N > N_4))$.

Thus, we may apply the goal replacement rule and we replace clause 12a by the following clause:

13a. $thm([0|X])\langle N\rangle \leftarrow N > N_1 + N_2 + N_3 + 4 \wedge N_3 \geq N_4 \wedge\ gen(X)\langle N_1\rangle \wedge$
$trans(X, Y)\langle N_2\rangle \wedge\ test(Y)\langle N_4\rangle$

By folding clauses 11a and 13a using clauses 4a and 5a we get:

14a. $thm([0|X])\langle N\rangle \leftarrow N > N_1 + N_5 + 3 \wedge gen(X)\langle N_1\rangle \wedge\ test(X)\langle N_5\rangle$

Finally, by folding clause 14a using clause 1a, we derive:

15a. $thm([0|X])\langle N\rangle \leftarrow N > N_6 + 2 \wedge thm(X)\langle N_6\rangle$

The final annotated program is $(\overline{P} - \{1a\}) \cup \{10a, 15a\}$. By applying the projection π we erase the annotations from clauses 10a and 15a and we get:

10. $thm([\,]) \leftarrow$
15. $thm([0|X]) \leftarrow thm(X)$

Thus, the final program is $Q = (P - \{1\}) \cup \{10, 15\}$. By Theorem 4.5 of Section 4 the transformation of P into Q is totally correct. In Q the predicates *thm* and *gen* are defined by two sets of clauses (namely, clauses 10, 15 and clauses 2, 3, respectively) which are syntactically equivalent and, therefore, as mentioned above, we may conclude that $\forall X\, (thm(X) \leftrightarrow gen(X))$ is true in the least Herbrand model of P.

7. Related Work and Conclusions

We have proposed a general transformation rule, called clause replacement, which generalizes the familiar unfolding, folding, and goal replacement transformations of definite logic programs. Then we have introduced a method for proving the total correctness of the clause replacement rule. Our method is based on program annotations, which are functions that add suitable arguments to the predicates occurring in a given program. In particular, we have introduced well-founded annotations, which ensure that the annotated program is terminating and, thus, it has a unique fixpoint [3]. Note that annotated logic programs can be considered as a generalization of the *instrumented* SOS rules introduced in [26], because SOS rules [21] can be viewed as particular logic programs.

Our proof method uses the unique fixpoint principle, which has been first introduced for proving properties of recursive equation programs (see [9] for a brief presentation and more bibliographic references) and it has also been extended to *inductive definitions* [22]. The unique fixpoint principle generalizes McCarthy's *recursion induction* principle [18] by replacing the requirement that a set of equations (viewed as rewriting rules) terminate, by the requirement that this set of equations has a unique solution in a suitable semantic domain.

However, our proof method is more general than the unique fixpoint method. Indeed, in order to prove the total correctness of the transformation of program P into program Q, the unique fixpoint method requires that the immediate consequence operator T_Q has a unique fixpoint, while according to Theorem 3.7 of Section 3, we only need to derive from the annotated program $\overline{P}$ an annotated program $\overline{Q}$ such that $T_{\overline{Q}}$ has a unique fixpoint (and this is ensured by the fact that $\overline{Q}$ is decreasing and, thus, terminating). Note, however, that in order to apply the well-founded annotation method, T_Q need not have a unique fixpoint and, in particular, Q need not be terminating (see, for instance, Example 9 of Section 3).

We claim that our proof method is also more general than the *improvement* method [25, 26], in the sense that if the total correctness of a transformation can be proved by the improvement

method, then it can also be proved by our method, and not vice versa, as we now show in the particular case where the transformation is performed by using the goal replacement rule R3 (see Section 4). Suppose that, by applying Rule R3, an annotated clause of the form $\overline{C}$: $H\langle X\rangle \leftarrow c_1(X, X_1) \wedge A_1\langle X_1\rangle$ is replaced by an annotated clause of the form $\overline{D}$: $H\langle X\rangle \leftarrow c_2(X, X_2) \wedge A_2\langle X_2\rangle$. Suppose also that, as required by the hypotheses of Theorem 4.5, clause $\overline{C}$ is decreasing w.r.t. a suitable well-founded ordering $\succ$, that is, $\mathcal{W} \models \forall\, (c_1(X, X_1) \rightarrow X \succ X_1)$. Let $X \succeq Y$ be defined as $(X \succ Y \vee Y = X)$. By adapting the definitions of [25, 26] to our context, we have that the replacement of $\overline{C}$ by $\overline{D}$ is an improvement iff $c_2(X, X_2)$ is of the form $c_1(X, X_1) \wedge X_1 \succeq X_2$ and $A_1\langle X_1\rangle \Rightarrow_{\{X_1\}\cup vars(\{A_1,A_2\})} X_1 \succeq X_2 \wedge A_2\langle X_2\rangle$ holds in the initial program $\overline{P}_0$ of the transformation sequence, that is, for every ground instance $a_1\langle w_1\rangle$ of $A_1\langle X_1\rangle$ belonging to $M(\overline{P}_0)$, there exists a ground instance $a_2\langle w_2\rangle$ of $A_2\langle X_2\rangle$ in $M(\overline{P}_0)$ such that $w_1 \succeq w_2$. Since $\overline{C}$ is decreasing w.r.t. $\succ$, if the replacement of $\overline{C}$ by $\overline{D}$ is an improvement, then Condition (δ) of Rule R3 is fulfilled, that is, $\mathcal{W} \models \forall\, (c_2(X, X_2) \rightarrow X \succ X_2)$. Thus, the total correctness of this replacement is a consequence of our well-founded annotation method. However, the opposite implication is not true, that is, there exists a goal replacement which satisfies Condition (δ) of Rule R3, and it is not an improvement. Indeed, with reference to the formalization of the notion of improvement considered above, it is possible to find a replacement such that, for a suitable well-founded ordering $\succ$, we have that $X \succ X_1$ and $X \succ X_2$, and it is not the case that $X_1 \succeq X_2$.

As already mentioned, the clause replacement rule presented here is more general than the unfolding, folding, and goal replacement rules for definite programs presented in the landmark paper by Tamaki and Sato [28] and in subsequent papers (see, for instance, [4, 11, 13, 24, 29]).

Recall that, as already discussed in the Introduction, the total correctness of the unfolding, folding, and goal replacement rules presented in [11, 13, 24, 28, 29] is ensured if suitable (rather complex) proof measures do not increase when these rules are applied during the construction of a transformation sequence. The fact that a given proof measure does not increase can be viewed as an improvement in the sense of [25, 26]. (Actually, the basic idea underlying the improvement method has been strongly influenced by early work in the field of logic program transformation.) Thus, the general argument used above to claim that our proof method is more powerful than the improvement method, can also be used to claim that our unfolding, folding, and goal replacement rules of Section 4 are more powerful than the unfolding, folding, and goal replacement rules presented in [11, 13, 24, 28, 29], in the sense that there are transformations that can be performed by our rules and cannot be proved correct by showing that a given proof measure does not increase.

In practice, when we limit ourselves to the use of the unfolding and folding rules presented in [11, 13, 24, 28, 29] (that is, we do not use the goal replacement rule), we may avoid checking that the given proof measure does not increase, and we may, instead, analyze the transformation sequence and check that each application of the folding rule is preceded by suitable applications of the unfolding rule. However, if we also use the goal replacement rule, the verification that the given proof measure does not increase cannot be avoided and, unfortunately, no general method is proposed in [11, 13, 24, 28, 29] to do this verification.

Note also that, unlike [11, 13, 24, 28, 29], our conditions for the total correctness of a transformation sequence constructed by using the unfolding, folding, and goal replacement rules of Section 4, do not depend on the results of an analysis of the transformation sequence and can be checked by proving that suitable first order formulas hold in the least $\mathcal{W}$-model of the initial annotated program of the transformation sequence.

The unfolding and folding rules presented in [4] do not depend on proof-theoretic conditions, like the ones in [11, 13, 24, 28, 29]. Instead, the applicability condition of the folding rule is based on a property, called *semantic delay*, of the immediate consequence operator associated with the program to be transformed. Let us briefly recall the notion of semantic delay in the simple case where the semantics of the program is defined as its least Herbrand model and the application of the folding rule consists in replacing a ground atom A_1 in the body of a clause of a program P by a new ground atom A_2. The semantic delay of A_2 with respect to A_1 is the least integer number n such that, for every natural number m, if $A_1 \in T_P^m(\emptyset)$ then $A_2 \in T_P^{m+n}(\emptyset)$. A sufficient condition for the total correctness of folding is that the semantic delay of A_2 with respect to A_1 is not positive.

The notion of semantic delay can be extended to other semantics and more complex replacements. In particular, in [5] Bossi et al. consider general logic programs with Fitting's three-valued semantics and introduce the *simultaneous replacement* transformation rule, which simultaneously replaces n (>0) conjunctions of literals, each of which occurs in the body of a clause. Then in [5] it is shown that if each conjunction of literals is replaced by an equivalent (with respect to Fitting's semantics) new conjunction of literals and the semantic delay of each new conjunction with respect to the corresponding old conjunction is not positive, then the initial and the transformed program have the same Fitting three-valued model.

When restricted to definite programs the simultaneous replacement rule is less general than our clause replacement rule. Indeed, the unfolding rule is not an instance of the simultaneous replacement rule, while it is an instance of the clause replacement rule. Moreover, the equivalence with respect to the least Herbrand model does not imply the equivalence with respect to Fitting's semantics (while the opposite implication holds) and, thus, some clause replacements may not be performed by simultaneous replacements. Finally, similarly to [11, 13, 24, 28, 29], in [5] Bossi et al. do not provide any method to prove that the semantic delay is not positive, while by using our method based on well-founded annotations, the total correctness of a clause replacement can be shown by proving suitable first order formulas.

Various notions of termination have been considered in [6, 8, 14] to prove the total correctness of transformations of logic programs. Let us briefly describe how the results of these papers are related to the method presented here.

In [8] Cook and Gallagher present a result which ensures the total correctness of the goal replacement rule based on the termination of the programs derived by applying this rule. This result is generalized by our Corollary 2.14 in Section 2. Indeed, our clause replacement rule is more general than the goal replacement rule and the uniqueness of fixpoint is a more general property than termination. Thus, the result by Cook and Gallagher suffers from the same limitation as our Corollary 2.14, in that it cannot be used to prove the total correctness of a transformation when the derived program is not terminating. In particular, the total correctness of the transformation of the reachability program presented in Example 9 cannot be proved by using the results in [8]. To overcome this limitation is, indeed, one of the main motivations of our paper, and our method based on well-founded annotations allows us to prove the correctness of a transformation sequence even if the final program in the sequence is not terminating.

In [6] Bossi and Etalle prove that the unfolding and folding transformation rules for general logic programs presented in [27] preserve *acyclicity*, and this property implies the termination of each program of a transformation sequence. In the case of definite programs, the results presented here are strictly more general than the ones presented in [6] because: (i) the unfolding and folding rules considered in [6] are particular cases of our clause replacement rule, and

(ii) similarly to [8], the results in [6] cannot be used to prove the total correctness of a transformation that produces a non-terminating (thus, non-acyclic) program (consider, once again, our reachability Example 9).

In [14] the correctness of the unfold/fold program transformations is proved under the additional hypothesis that they preserve *existential termination*. A program P is said to be existentially terminating with respect to an atom A iff there exists a finite SLD-tree for $P \cup \{\neg A\}$ with either a success branch or all failure branches. However, no method is given in [14] to check whether or not existential termination is preserved by a program transformation, while the well-founded annotation method presented here provides first order formulas to be checked for proving the total correctness of transformation sequences.

From a practical point of view, the main advantage of using the transformation rules proposed in this paper is that, as already mentioned, the correctness of a transformation sequence is guaranteed by the validity of suitable first-order annotation formulas, instead of proof-theoretic or semantics-based conditions, and the validity of these formulas can be checked by using available theorem provers. For the sake of generality, we have assumed that annotation formulas are arbitrary first-order formulas. However, in practice, as shown by our examples, linear constraints over natural numbers are sufficient to deal with a large class of program transformations. Even though the validity problem for this class of constraints is NP-complete, some tools that work efficiently in most practical cases have been developed (see, for instance, [23]).

In order to make use of our transformation rules in practice, one has to choose a suitable well-founded ordering $\succ$ and a suitable annotation function for the initial program of a transformation sequence. In Section 4 the presentation of the rules is parametric with respect to this well-founded ordering $\succ$ and this annotation function, but in general a suitable choice is needed to be able to derive a final annotated program which is decreasing w.r.t. $\succ$. The well-founded orderings and the annotation functions given in the examples of this paper are quite powerful in practice. However, more sophisticated well-founded orderings may be needed, depending on the specific applications of the transformation rules. These sophisticated orderings can be constructed by using well-established techniques developed for proving termination of Term Rewrite Systems [10].

Due to its generality, we believe that our approach can easily be extended to other logic programming languages and, in particular, to normal logic programs. This extension can be based on the fact that the immediate consequence operator of an *acyclic* normal logic program has a unique fixpoint. Thus, we may construct a totally correct transformation from program P_0 to program P_n by the following three steps: (i) we construct from P_0 an annotated program $\overline{P}_0$, (ii) we transform the annotated program $\overline{P}_0$ into an acyclic annotated program $\overline{P}_n$, and (iii) we erase the annotations from $\overline{P}_n$, thereby getting P_n. Note that, if at Step (i) we construct an *acyclic* annotated program, then we can use the transformation rules that preserve acyclicity considered in [6] to perform Step (ii).

Finally, we would like to note that the notion of total correctness considered in this paper is different from the one used in the case of imperative programs, where a program is said to be *totally correct with respect to a given specification* iff its input-output relation satisfies the specification and, moreover, the program terminates (see, for instance, [17]). In fact, as already mentioned, the transformation of program P into program Q can be totally correct even if Q is not terminating. However, in order to prove that the transformation of P into Q is totally correct we transform an annotated program $\overline{P}$ into a *terminating* annotated program $\overline{Q}$. In this sense we may say that the program $\overline{Q}$ is totally correct with respect to the specification provided

by the program $\overline{P}$. Similarly to the proofs of total correctness for imperative programs based on the axiomatic approach [17], also the derivation of the terminating program $\overline{Q}$ is performed by proving first order implications and suitable well-founded ordering relations.

Acknowledgments

We would like to thank Sandro Etalle and John Gallagher for many stimulating discussions concerning various issues addressed in this paper. Our thanks also go to the anonymous referees for helpful comments and suggestions.

Appendix

In this Appendix we will use the following notation. Given a substitution $\vartheta = \{X_1/t_1, \ldots, X_n/t_n\}$, by $dom(\vartheta)$ we denote the set of variables $\{X_1, \ldots, X_n\}$. Given a set V of variables, by $\vartheta \restriction V$ we denote the substitution $\{X/t \mid X/t \in \vartheta \text{ and } X \in V\}$.

Proof of Lemma 2.3

For reasons of simplicity we assume that Γ_1 is of the form:

$\{p(t_1) \leftarrow B_1,\ \ldots,\ p(t_m) \leftarrow B_m\}$

and Γ_2 is of the form:

$\{p(u_1) \leftarrow D_1,\ \ldots,\ p(u_n) \leftarrow D_n\}$

The general case where the heads of the clauses have several predicate symbols of arbitrary arities is a straightforward extension.

It follows directly from Definitions 2.1 and 2.2 and from the definition of Herbrand interpretation that $I \models \Gamma_1 \Rightarrow \Gamma_2$ iff the following property, called *IMP*, holds.

Property IMP: for every ground term x, for every j, with $1 \leq j \leq n$, such that

$I \models \exists Z_1 \ldots \exists Z_k\, (x = u_j \wedge D_j)$

where $\{Z_1, \ldots, Z_k\} = vars(u_j) \cup vars(D_j)$, there exists i, with $1 \leq i \leq m$, such that

$I \models \exists Y_1 \ldots \exists Y_h\, (x = t_i \wedge B_i)$

where $\{Y_1, \ldots, Y_h\} = vars(t_i) \cup vars(B_i)$.

Let us prove the only-if part of the lemma. Assume that Property *IMP* holds. Now, let us take a ground instance $(p(u_j) \leftarrow D_j)\vartheta$ of a clause in Γ_2 such that $I \models D_j\vartheta$ and ϑ is a ground substitution of the form $\{Z_1/z_1, \ldots, Z_k/z_k\}$. Let x be the ground term $u_j\vartheta$. We have that $I \models \exists Z_1 \ldots \exists Z_k\, (x = u_j \wedge D_j)$ and, thus, by Property *IMP*, for some i, with $1 \leq i \leq m$, we have that $I \models \exists Y_1 \ldots \exists Y_h\, (x = t_i \wedge B_i)$. By the definition of Herbrand interpretation, there exists a ground substitution η of the form $\{Y_1/y_1, \ldots, Y_h/y_h\}$, such that $I \models (x = t_i \wedge B_i)\eta$. Thus, there exists a ground instance $(p(t_i) \leftarrow B_i)\eta$ of a clause in Γ_1 which has the same head as $(p(u_j) \leftarrow D_j)\vartheta$ (because $t_i\eta$ is identical to $u_j\vartheta$, which is x) and $I \models B_i\eta$.

Let us now prove the if part of the lemma. Assume that for every ground instance C_2 of a clause in Γ_2 such that $I \models bd(C_2)$ there exists a ground instance C_1 of a clause in Γ_1 such that $hd(C_1) = hd(C_2)$ and $I \models bd(C_1)$. Now, let us consider a ground term x and a clause $p(u_j) \leftarrow D_j$ in Γ_2 such that $I \models \exists Z_1 \ldots \exists Z_k\, (x = u_j \wedge D_j)$. By definition of Herbrand interpretation there exists a ground substitution $\vartheta = \{Z_1/z_1, \ldots, Z_k/z_k\}$ such that x is identical to $u_j\vartheta$ and $I \models D_j\vartheta$. Thus, there exists a ground instance $(p(u_j) \leftarrow D_j)\vartheta$ of a clause in Γ_2 such that $I \models D_j\vartheta$. By

hypothesis, there exists a ground instance $(p(t_i) \leftarrow B_i)\eta$ of a clause in Γ_1 such that: (i) η is a ground substitution of the form $\{Y_1/y_1, \ldots, Y_h/y_h\}$, (ii) $(p(t_i) \leftarrow B_i)\eta$ has the same head as $(p(u_j) \leftarrow D_j)\vartheta$, and (iii) $I \models B_i\eta$. By (ii), $t_i\eta$ is identical to $u_j\vartheta$, which is x. Thus, $I \models \exists Y_1 \ldots \exists Y_h\,(x{=}t_i \wedge B_i)$ and we have proved that Property *IMP* holds. □

Proof of Lemma 3.9

Suppose that $M(\overline{P}) \models \overline{\Gamma}_1 \Rightarrow \overline{\Gamma}_2$. Without loss of generality we assume that Γ_1 and Γ_2 are sets of clauses for a single predicate p. By Definition 2.2, $M(\overline{P}) \models \forall U\, \forall X\, (\overline{\varphi}_2 \rightarrow \overline{\varphi}_1)$, where:
(i) U is a set $\{U_1, \ldots, U_h\}$ of ordinary variables,
(ii) X is an annotation variable,
(iii) $if(\overline{\Gamma}_1)$ is of the form $p(U_1, \ldots, U_h)\langle X\rangle \leftarrow \overline{\varphi}_1$,
(iv) $\overline{\varphi}_1$ is of the form $\exists V_1 \exists Y_1\, (c_1 \wedge \overline{G}_1) \vee \ldots \vee \exists V_m \exists Y_m\, (c_m \wedge \overline{G}_m)$, where, for $i = 1, \ldots, m$, V_i is the set of ordinary variables occurring in G_i and not in U, and Y_i is the set of annotation variables occurring in $c_i \wedge \overline{G}_i$ and different from X,
(v) $if(\overline{\Gamma}_2)$ is of the form $p(U_1, \ldots, U_h)\langle X\rangle \leftarrow \overline{\varphi}_2$, and
(vi) $\overline{\varphi}_2$ is of the form $\exists W_1 \exists Z_1\, (d_1 \wedge \overline{Q}_1) \vee \ldots \vee \exists W_n \exists Z_n\, (d_n \wedge \overline{Q}_n)$, where, for $i = 1, \ldots, n$, W_i is the set of ordinary variables occurring in Q_i and not in U, and Z_i is the set of annotation variables occurring in $d_i \wedge \overline{Q}_i$ and different from X.
We want to show that $M(P) \models \Gamma_1 \Rightarrow \Gamma_2$, that is, by Definition 2.2, $M(P) \models \forall U\, (\varphi_2 \rightarrow \varphi_1)$, where:
(vii) φ_1 is of the form $\exists V_1\, G_1 \vee \ldots \vee \exists V_m\, G_m$, and
(viii) φ_2 is of the form $\exists W_1\, Q_1 \vee \ldots \vee \exists W_n\, Q_n$.
Let α be a ground substitution with $dom(\alpha) = U$, and suppose that $M(P) \models \varphi_2\alpha$. Hence there exist $i \in \{1, \ldots, n\}$ and a ground substitution β_i such that $dom(\beta_i) = W_i$ and $M(P) \models Q_i\alpha\beta_i$. Let Q_i be a conjunction of atoms of the form $A_1 \wedge \ldots \wedge A_k$. Thus, for $r = 1, \ldots, k$, $A_r\alpha\beta_i \in M(P)$. Let $\overline{Q}_i$ be a conjunction of annotated atoms of the form $A_1\langle N_1\rangle \wedge \ldots \wedge A_k\langle N_k\rangle$, where $N_1, \ldots, N_k$ are distinct annotation variables. Since $N_1, \ldots, N_k$ are distinct variables, by Proposition 3.4 there exists a ground substitution γ_i such that $dom(\gamma_i) = \{N_1, \ldots, N_k\}$ and, for $r = 1, \ldots, k$, $A_r\langle N_r\rangle\alpha\beta_i\gamma_i \in M(\overline{P})$. Thus, $M(\overline{P}) \models \overline{Q}_i\alpha\beta_i\gamma_i$. Moreover, since $dom(\gamma_i) = \{N_1, \ldots, N_k\}$, by Definition 3.3, there exists a ground substitution δ_i such that $dom(\delta_i) = vars(d_i) - \{N_1, \ldots, N_k\}$ and $\mathcal{W} \models d_i\gamma_i\delta_i$. Therefore, $M(\overline{P}) \models d_i\gamma_i\delta_i \wedge \overline{Q}_i\alpha\beta_i\gamma_i$. Since $d_i = d_i\alpha\beta_i$ (because $dom(\alpha\beta_i) \cap vars(d_i) = \emptyset$) and $\overline{Q}_i\alpha\beta_i\gamma_i = \overline{Q}_i\alpha\beta_i\gamma_i\delta_i$ (because $\overline{Q}_i\alpha\beta_i\gamma_i$ is a ground goal), we have that $M(\overline{P}) \models (d_i \wedge \overline{Q}_i)\alpha\beta_i\gamma_i\delta_i$. Hence, $M(\overline{P}) \models \exists W_i \exists Z_i\, (d_i \wedge \overline{Q}_i)\alpha\eta$, where $\eta = \delta_i \restriction \{X\}$, and thus, $M(\overline{P}) \models \overline{\varphi}_2\alpha\eta$. Since $M(\overline{P}) \models \forall U\, \forall X\, (\overline{\varphi}_2 \rightarrow \overline{\varphi}_1)$, we have that $M(\overline{P}) \models \overline{\varphi}_1\alpha\eta$. Thus, there exist $j \in \{1, \ldots, m\}$ and two ground substitutions ϑ_j, λ_j such that: (i) $dom(\vartheta_j) = V_j$, (ii) $dom(\lambda_j) = Y_j$, and (iii) $M(\overline{P}) \models (c_j \wedge \overline{G}_j)\alpha\eta\vartheta_j\lambda_j$. Hence, $M(\overline{P}) \models \overline{G}_j\alpha\eta\vartheta_j\lambda_j$ and, by Proposition 3.2, since $(dom(\eta) \cup dom(\lambda_j)) \cap vars(G_j) = \emptyset$, we have that $M(P) \models G_j\alpha\vartheta_j$. Therefore, $M(P) \models \exists V_1\, G_1\alpha \vee \ldots \vee \exists V_m\, G_m\alpha$, and thus, we have that $M(P) \models \varphi_1\alpha$. □

Proof of Lemma 4.1

Let $\overline{C}$ be a clause of the form $\overline{H} \leftarrow c \wedge \overline{G}_L \wedge \overline{A} \wedge \overline{G}_R$ and let $\overline{C}_1$: $\overline{H}_1 \leftarrow c_1 \wedge \overline{G}_1$, …, $\overline{C}_m$: $\overline{H}_m \leftarrow c_m \wedge \overline{G}_m$, with $m \geq 0$, be all the clauses of program $\overline{P}_0$ such that, for $i = 1, \ldots, m$, $\overline{A}$ is unifiable with $\overline{H}_i$ via a most general unifier ϑ_i. Then, for $i = 1, \ldots, m$, $\overline{D}_i$ is a clause of the form $(\overline{H} \leftarrow c \wedge c_i \wedge \overline{G}_L \wedge \overline{G}_i \wedge \overline{G}_R)\vartheta_i$.

Proof of (1). We will first give the proof of $M(\overline{P}_0) \models \{\overline{C}\} \Rightarrow \{\overline{D}_1, \ldots, \overline{D}_m\}$ and then the proof of $M(\overline{P}_0) \models \{\overline{C}\} \Leftarrow \{\overline{D}_1, \ldots, \overline{D}_m\}$.

In order to prove $M(\overline{P}_0) \models \{\overline{C}\} \Rightarrow \{\overline{D}_1, \ldots, \overline{D}_m\}$, by Lemma 2.3 it is enough to prove that, for $i = 1, \ldots, m$, for every ground instance $\overline{D}_i\sigma_i$ of clause $\overline{D}_i$ such that $M(\overline{P}_0) \models bd(\overline{D}_i\sigma_i)$, there exists a ground instance $\overline{C}\tau$ of $\overline{C}$ such that $hd(\overline{C}\tau) = hd(\overline{D}_i\sigma_i)$ and $M(\overline{P}_0) \models bd(\overline{C}\tau)$.

Let $\overline{D}_i\sigma_i$ be a clause of the form $(\overline{H} \leftarrow c \wedge c_i \wedge \overline{G}_L \wedge \overline{G}_i \wedge \overline{G}_R)\vartheta_i\sigma_i$ such that $M(\overline{P}_0) \models (c \wedge c_i \wedge \overline{G}_L \wedge \overline{G}_i \wedge \overline{G}_R)\vartheta_i\sigma_i$ and, therefore, $M(\overline{P}_0) \models (c \wedge \overline{G}_L \wedge \overline{G}_R)\vartheta_i\sigma_i$ and $M(\overline{P}_0) \models (c_i \wedge \overline{G}_i)\vartheta_i\sigma_i$. Let $(\overline{H}_i \leftarrow c_i \wedge \overline{G}_i)\vartheta_i\sigma_i\tau_i$ be a ground instance of $\overline{C}_i$, where τ_i is a ground substitution such that $dom(\tau_i) = vars(\overline{H}_i) - vars(c_i \wedge \overline{G}_i)$. Since $(c_i \wedge \overline{G}_i)\vartheta_i\sigma_i\tau_i = (c_i \wedge \overline{G}_i)\vartheta_i\sigma_i$ (because $(c_i \wedge \overline{G}_i)\vartheta_i\sigma_i$ is ground), $M(\overline{P}_0) \models (c_i \wedge \overline{G}_i)\vartheta_i\sigma_i$, and $\overline{H}_i \leftarrow c_i \wedge \overline{G}_i$ is true in $M(\overline{P}_0)$, we have that $M(\overline{P}_0) \models \overline{H}_i\vartheta_i\sigma_i\tau_i$. Since ϑ_i is a unifier of $\overline{A}$ and $\overline{H}_i$, we have that $\overline{A}\vartheta_i\sigma_i\tau_i = \overline{H}_i\vartheta_i\sigma_i\tau_i$ and $M(\overline{P}_0) \models \overline{A}\vartheta_i\sigma_i\tau_i$. Let us consider the clause $\overline{C}\vartheta_i\sigma_i\tau_i$, which is of the form $\overline{H}\vartheta_i\sigma_i \leftarrow c\vartheta_i\sigma_i \wedge \overline{G}_L\vartheta_i\sigma_i \wedge \overline{A}\vartheta_i\sigma_i\tau_i \wedge \overline{G}_R\vartheta_i\sigma_i$, because $\overline{H}\vartheta_i\sigma_i$, $c\vartheta_i\sigma_i$, $\overline{G}_L\vartheta_i\sigma_i$, and $\overline{G}_R\vartheta_i\sigma_i$ are ground and, thus, $\overline{H}\vartheta_i\sigma_i\tau_i = \overline{H}\vartheta_i\sigma_i$, $c\vartheta_i\sigma_i\tau_i = c\vartheta_i\sigma_i$, $\overline{G}_L\vartheta_i\sigma_i\tau_i = \overline{G}_L\vartheta_i\sigma_i$, and $\overline{G}_R\vartheta_i\sigma_i\tau_i = \overline{G}_R\vartheta_i\sigma_i$. We have that $\overline{C}\vartheta_i\sigma_i\tau_i$ is a ground instance of $\overline{C}$ such that: (i) $hd(\overline{C}\vartheta_i\sigma_i\tau_i) = hd(\overline{D}_i\sigma_i)$ and (ii) $M(\overline{P}_0) \models bd(\overline{C}\vartheta_i\sigma_i\tau_i)$.

Now we prove that $M(\overline{P}_0) \models \{\overline{C}\} \Leftarrow \{\overline{D}_1, \ldots, \overline{D}_m\}$. By Lemma 2.3 it is enough to prove that, for every ground instance $\overline{C}\sigma$ of $\overline{C}$ such that $dom(\sigma) = vars(\overline{C})$ and $M(\overline{P}_0) \models bd(\overline{C}\sigma)$, there exists $i \in \{1, \ldots, m\}$ and a ground instance $\overline{D}_i\tau_i$ of $\overline{D}_i$ such that $dom(\tau_i) = vars(\overline{D}_i)$, $hd(\overline{D}_i\tau_i) = hd(\overline{C}\sigma)$, and $M(\overline{P}_0) \models bd(\overline{D}_i\tau_i)$.

Let $\overline{C}\sigma$ be a clause of the form $(\overline{H} \leftarrow c \wedge \overline{G}_L \wedge \overline{A} \wedge \overline{G}_R)\sigma$ such that $M(\overline{P}_0) \models (c \wedge \overline{G}_L \wedge \overline{A} \wedge \overline{G}_R)\sigma$. We have that $M(\overline{P}_0) \models \overline{A}\sigma$ and, since $M(\overline{P}_0)$ is a fixpoint of $T_{\overline{P}_0}$, there exists a ground instance $(\overline{H}_i \leftarrow c_i \wedge \overline{G}_i)\sigma_i$ of a clause $\overline{C}_i \in \overline{P}_0$, whose head $\overline{H}_i$ is unifiable with $\overline{A}$, such that: (i) $\overline{A}\sigma = \overline{H}_i\sigma_i$ and (ii) $M(\overline{P}_0) \models (c_i \wedge \overline{G}_i)\sigma_i$. Since $vars(\overline{C}) \cap vars(\overline{C}_i) = \emptyset$, we may assume that $dom(\sigma) \cap vars(\overline{C}_i) = \emptyset$. Thus, we have that: $\overline{A}\sigma\sigma_i = \overline{A}\sigma$ (because $\overline{A}\sigma$ is a ground annotated atom) $= \overline{H}_i\sigma_i = \overline{H}_i\sigma\sigma_i$ (because $dom(\sigma) \cap vars(H_i) = \emptyset$), that is, $\sigma\sigma_i$ is a unifier of $\overline{A}$ and $\overline{H}_i$. Since ϑ_i is the most general unifier of $\overline{A}$ and $\overline{H}_i$, it follows that $\sigma\sigma_i = \vartheta_i\tau_i$ for some ground substitution τ_i. Let us now consider the clause $\overline{D}_i$ of the form $(\overline{H} \leftarrow c \wedge c_i \wedge \overline{G}_L \wedge \overline{G}_i \wedge \overline{G}_R)\vartheta_i$. We have that $\overline{D}_i\tau_i$ is a ground instance of $\overline{D}_i$ such that: (i) $hd(\overline{D}_i\tau_i) = hd(\overline{C}\sigma)$ and (ii) $M(\overline{P}_0) \models bd(\overline{D}_i\tau_i)$. Indeed, we have that: (i) $hd(\overline{D}_i\tau_i) = \overline{H}\vartheta_i\tau_i = \overline{H}\sigma\sigma_i$ (because $\vartheta_i\tau_i = \sigma\sigma_i$) $= \overline{H}\sigma$ (because $\overline{H}\sigma$ is a ground annotated atom) $= hd(\overline{C}\sigma)$, and we have that: (ii.a) $bd(\overline{D}_i\tau_i)$ is $(c \wedge c_i \wedge \overline{G}_L \wedge \overline{G}_i \wedge \overline{G}_R)\vartheta_i\tau_i$, (ii.b) $M(\overline{P}_0) \models (c \wedge \overline{G}_L \wedge \overline{G}_R)\vartheta_i\tau_i$, because $M(\overline{P}_0) \models (c \wedge \overline{G}_L \wedge \overline{G}_R)\sigma$ and $(c \wedge \overline{G}_L \wedge \overline{G}_R)\sigma = (c \wedge \overline{G}_L \wedge \overline{G}_R)\sigma\sigma_i$ (because $(c \wedge \overline{G}_L \wedge \overline{G}_R)\sigma$ is a ground annotated formula) $= (c \wedge \overline{G}_L \wedge \overline{G}_R)\vartheta_i\tau_i$ (because $\sigma\sigma_i = \vartheta_i\tau_i$), and (ii.c) $M(\overline{P}_0) \models (c_i \wedge \overline{G}_i)\vartheta_i\tau_i$, because $M(\overline{P}_0) \models (c_i \wedge \overline{G}_i)\sigma_i$ and $(c_i \wedge \overline{G}_i)\sigma_i = (c_i \wedge \overline{G}_i)\sigma\sigma_i$ (because $dom(\sigma) \cap vars(c_i \wedge \overline{G}_i) = \emptyset$) $= (c_i \wedge \overline{G}_i)\vartheta_i\tau_i$ (because $\sigma\sigma_i = \vartheta_i\tau_i$).

Proof of (2). Let us consider a clause $\overline{D}_i \in \{\overline{D}_1, \ldots, \overline{D}_m\}$ of the form $(\overline{H} \leftarrow c \wedge c_i \wedge \overline{G}_L \wedge \overline{G}_i \wedge \overline{G}_R)\vartheta_i$. By hypothesis, the clause $\overline{C}$ of the form $\overline{H} \leftarrow c \wedge \overline{G}_L \wedge \overline{A} \wedge \overline{G}_R$ is decreasing w.r.t. $\succ$, that is, $\mathcal{W} \models \forall\, (c \rightarrow \overline{H} \succ (\overline{G}_L \wedge \overline{A} \wedge \overline{G}_R))$ and, therefore, $\mathcal{W} \models \forall\, ((c \rightarrow \overline{H} \succ (\overline{G}_L \wedge \overline{A} \wedge \overline{G}_R))\vartheta_i)$. Let us now consider the clause $\overline{C}_i \in \overline{P}_0$ of the form $\overline{H}_i \leftarrow c_i \wedge \overline{G}_i$. Since every clause of $\overline{P}_0$ is decreasing w.r.t. $\succ$, we have that $\mathcal{W} \models \forall\, (c_i \rightarrow \overline{H}_i \succ \overline{G}_i)$ and, therefore, $\mathcal{W} \models \forall\, ((c_i \rightarrow \overline{H}_i \succ \overline{G}_i)\vartheta_i)$. Since $\overline{A}\vartheta_i = \overline{H}_i\vartheta_i$, by transitivity of $\succ$, we conclude that $\mathcal{W} \models \forall\, ((c \wedge c_i \rightarrow \overline{H} \succ (\overline{G}_L \wedge \overline{G}_i \wedge \overline{G}_R))\vartheta_i)$, that is, $\overline{D}_i$ is decreasing w.r.t. $\succ$. $\square$

Proof of Lemma 4.2

For $i = 1, \ldots, m$, $\overline{C}_i$ is a clause in $\overline{P}_0$ of the form $\overline{H} \leftarrow c_i \wedge \overline{G}_i$ and $\overline{D}_i$ is a clause in $\overline{P}_k$ of the form $\overline{K} \leftarrow d \wedge c_i\vartheta \wedge \overline{G}_L \wedge \overline{G}_i\vartheta \wedge \overline{G}_R$, where ϑ is a substitution such that Conditions 1–3 of Rule R2 are satisfied. The clause $\overline{E}$ derived by folding $\overline{D}_1, \ldots, \overline{D}_m$ using clauses $\overline{C}_1, \ldots, \overline{C}_m$ is of the form $\overline{K} \leftarrow d \wedge \overline{G}_L \wedge \overline{H}\vartheta \wedge \overline{G}_R$.

Proof of (1). We will first give the proof of $M(\overline{P}_0) \models \{\overline{D}_1, \ldots, \overline{D}_m\} \Rightarrow \{\overline{E}\}$ and then the proof of $M(\overline{P}_0) \models \{\overline{D}_1, \ldots, \overline{D}_m\} \Leftarrow \{\overline{E}\}$.

In order to prove $M(\overline{P}_0) \models \{\overline{D}_1, \ldots, \overline{D}_m\} \Rightarrow \{\overline{E}\}$, by Lemma 2.3 it is enough to prove that, for every ground instance $\overline{E}\sigma$ of $\overline{E}$ such that $dom(\sigma) = vars(\overline{E})$ and $M(\overline{P}_0) \models bd(\overline{E}\sigma)$, there exist $i \in \{1, \ldots, m\}$ and a ground instance $\overline{D}_i\tau_i$ of $\overline{D}_i$ such that $dom(\tau_i) = vars(\overline{D}_i)$, $hd(\overline{D}_i\tau_i) = hd(\overline{E}\sigma)$, and $M(\overline{P}_0) \models bd(\overline{D}_i\tau_i)$.
Let $\overline{E}\sigma$ be a clause of the form $(\overline{K} \leftarrow d \wedge \overline{G}_L \wedge \overline{H}\vartheta \wedge \overline{G}_R)\sigma$ and suppose that $M(\overline{P}_0) \models (d \wedge \overline{G}_L \wedge \overline{H}\vartheta \wedge \overline{G}_R)\sigma$. Let us consider the following substitutions: $\eta = \vartheta \restriction vars(\overline{H})$ and, for $i = 1, \ldots, m$, $\gamma_i = \vartheta \restriction (vars(c_i \wedge \overline{G}_i) - vars(\overline{H}))$. By Condition 2 of Rule R2, γ_i is of the form: $\{U_1/W_1, \ldots, U_{n_i}/W_{n_i}\}$, where $W_1, \ldots, W_{n_i}$ are distinct variables not occurring in $\overline{E}$. Let ρ_i be the substitution $\{W_1/U_1, \ldots, W_{n_i}/U_{n_i}\}$. The following two properties hold: (C1) $\overline{H}\vartheta = \overline{H}\eta$ and (C2) $(c_i \wedge \overline{G}_i)\eta = (c_i \wedge \overline{G}_i)\vartheta\rho_i$. Since $M(\overline{P}_0) \models (d \wedge \overline{G}_L \wedge \overline{H}\vartheta \wedge \overline{G}_R)\sigma$, we also have that $M(\overline{P}_0) \models \overline{H}\vartheta\sigma$ and, by Property C1, $M(\overline{P}_0) \models \overline{H}\eta\sigma$. Since $M(\overline{P}_0)$ is a fixpoint of $T_{\overline{P}_0}$ and, by Condition 1 of Rule R2, all clauses of $\overline{P}_0$ whose head is unifiable with $\overline{H}\vartheta$ are in $\{\overline{C}_1, \ldots, \overline{C}_m\}$, there exist a clause $\overline{C}_i$: $\overline{H} \leftarrow c_i \wedge \overline{G}_i$ in $\{\overline{C}_1, \ldots, \overline{C}_m\}$ and a ground substitution ν such that $M(\overline{P}_0) \models (c_i \wedge \overline{G}_i)\eta\sigma\nu$ where $dom(\nu) = vars(c_i \wedge \overline{G}_i) - vars(\overline{H})$. We have that: $(c_i \wedge \overline{G}_i)\eta\sigma\nu = (c_i \wedge \overline{G}_i)\vartheta\rho_i\sigma\nu$ (by Property C2) $= (c_i \wedge \overline{G}_i)\vartheta\sigma\rho_i\nu$ (because no variable occurs simultaneously in ρ_i and σ) and, therefore, $M(\overline{P}_0) \models (c_i \wedge \overline{G}_i)\vartheta\sigma\rho_i\nu$. Since $(d \wedge \overline{G}_L \wedge \overline{G}_R)\sigma$ is a ground goal and $M(\overline{P}_0) \models (d \wedge \overline{G}_L \wedge \overline{G}_R)\sigma$, we also have that $M(\overline{P}_0) \models (d \wedge \overline{G}_L \wedge \overline{G}_R)\sigma\rho_i\nu$ and, therefore, $M(\overline{P}_0) \models (c_i\vartheta \wedge d \wedge \overline{G}_L \wedge \overline{G}_i\vartheta \wedge \overline{G}_R)\sigma\rho_i\nu$. Let us now consider the substitution $\tau_i = \sigma\rho_i\nu$. We have that: $hd(\overline{D}_i\tau_i) = \overline{K}\sigma\rho_i\nu = \overline{K}\sigma$ (because $\overline{K}\sigma$ is a ground atom) $= hd(\overline{E}\sigma)$ and $M(\overline{P}_0) \models bd(\overline{D}_i\tau_i)$.

Now we prove that $M(\overline{P}_0) \models \{\overline{D}_1, \ldots, \overline{D}_m\} \Leftarrow \{\overline{E}\}$. By Lemma 2.3, it is enough to prove that, for $i = 1, \ldots, m$, for every ground instance $\overline{D}_i\sigma_i$ of $\overline{D}_i$ such that $M(\overline{P}_0) \models bd(\overline{D}_i\sigma_i)$, there exists a ground instance $\overline{E}\tau$ of $\overline{E}$ such that $hd(\overline{E}\tau) = hd(\overline{D}_i\sigma_i)$ and $M(\overline{P}_0) \models bd(\overline{E}\tau)$.
Let $\overline{D}_i\sigma_i$ be a clause of the form $(\overline{K} \leftarrow d \wedge c_i\vartheta \wedge \overline{G}_L \wedge \overline{G}_i\vartheta \wedge \overline{G}_R)\sigma_i$ and suppose that $M(\overline{P}_0) \models (d \wedge c_i\vartheta \wedge \overline{G}_L \wedge \overline{G}_i\vartheta \wedge \overline{G}_R)\sigma_i$. Let $(\overline{H} \leftarrow c_i \wedge \overline{G}_i)\vartheta\sigma_i\tau_i$ be a ground instance of $\overline{C}_i$, where $dom(\tau_i) = vars(\overline{H}) - vars(c_i \wedge \overline{G}_i)$. Since $M(\overline{P}_0) \models \overline{H} \leftarrow c_i \wedge \overline{G}_i$, we have that $M(\overline{P}_0) \models (d \wedge \overline{G}_L \wedge \overline{H}\vartheta \wedge \overline{G}_R)\sigma_i\tau_i$. Now, let us consider the clause $\overline{E}\sigma_i\tau_i$ of the form $(\overline{K} \leftarrow d \wedge \overline{G}_L \wedge \overline{H}\vartheta \wedge \overline{G}_R)\sigma_i\tau_i$. We have that: (i) $hd(\overline{E}\sigma_i\tau_i) = \overline{K}\sigma_i\tau_i = \overline{K}\sigma_i$ (because $\overline{K}\sigma_i$ is a ground annotated atom) $= hd(\overline{D}_i\sigma_i)$ and (ii) $M(\overline{P}_0) \models bd(\overline{E}\sigma_i\tau_i)$.

Proof of (2). Straightforward from Condition 3 of Rule R2. □

Proof of Lemma 4.4

Let $\overline{C}$ be an annotated clause of the form $\overline{H} \leftarrow c \wedge c_1 \wedge \overline{G}_L \wedge \overline{G}_1 \wedge \overline{G}_R$. Suppose that the replacement law $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ holds in $\overline{P}_0$, where $X = vars(\{\overline{H}, c, \overline{G}_L, \overline{G}_R\}) \cap vars(\{c_1, \overline{G}_1, c_2, \overline{G}_2\})$. Then, the clause $\overline{D}$ derived by goal replacement is of the form $\overline{H} \leftarrow c \wedge c_2 \wedge \overline{G}_L \wedge \overline{G}_2 \wedge \overline{G}_R$.

Proof of (1). C is a clause of the form $H \leftarrow G_L \wedge G_1 \wedge G_R$ and D is a clause of the form $H \leftarrow G_L \wedge G_2 \wedge G_R$. We will prove that for every ground instance $D\sigma$ of D such that $M(P_0) \models bd(D\sigma)$,

there exists a ground instance $C\tau$ of C such that $hd(C\tau) = hd(D\sigma)$ and $M(P_0) \models bd(C\tau)$. Then, by Lemma 2.3, $M(P_0) \models \{C\} \Rightarrow \{D\}$.
Let $D\sigma$ be a clause of the form $(H \leftarrow G_L \wedge G_2 \wedge G_R)\sigma$ and suppose that $M(P_0) \models (G_L \wedge G_2 \wedge G_R)\sigma$. Then $M(P_0) \models G_2\sigma$ and, since the replacement law $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ holds in $\overline{P}_0$, by Condition (i) of Definition 4.3, we have that: $M(P_0) \models (\exists Y' G_1\sigma) \leftarrow G_2\sigma$, where $Y' = vars(G_1\sigma) - vars((H, G_L, G_R)\sigma)$. Thus, there exists a ground substitution η such that $dom(\eta) = Y'$ and $M(P_0) \models G_1\sigma\eta$. Therefore, $M(P_0) \models (G_L \wedge G_1 \wedge G_R)\sigma\eta$. Let us now consider the ground instance $C\sigma\eta$: $(H \leftarrow G_L \wedge G_1 \wedge G_R)\sigma\eta$ of C. We have that: $hd(C\sigma\eta) = H\sigma\eta = H\sigma$ (because $H\sigma$ is a ground atom) $= hd(D\sigma)$ and $M(P_0) \models bd(C\sigma\eta)$.

Proof of (2). We show that, for every ground instance $\overline{C}\sigma$ of $\overline{C}$ such that $M(\overline{P}_0) \models bd(\overline{C}\sigma)$, there exists a ground instance $\overline{D}\tau$ of $\overline{D}$ such that $hd(\overline{D}\tau) = hd(\overline{C}\sigma)$ and $M(\overline{P}_0) \models bd(\overline{D}\tau)$. Then, by Lemma 2.3, $M(\overline{P}_0) \models \{\overline{C}\} \Leftarrow \{\overline{D}\}$.
Let $\overline{C}\sigma$ be a clause of the form $(\overline{H} \leftarrow c \wedge c_1 \wedge \overline{G}_L \wedge \overline{G}_1 \wedge \overline{G}_R)\sigma$ and suppose that $M(\overline{P}_0) \models (c \wedge c_1 \wedge \overline{G}_L \wedge \overline{G}_1 \wedge \overline{G}_R)\sigma$. Then $M(\overline{P}_0) \models (c_1 \wedge \overline{G}_1)\sigma$ and, since the replacement law $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ holds in $\overline{P}_0$, by Condition (ii) of Definition 4.3, we have that: $M(\overline{P}_0) \models (c_1 \wedge \overline{G}_1)\sigma \rightarrow \exists Z\, (c_2 \wedge \overline{G}_2)\sigma$, where $Z = vars(c_2 \wedge \overline{G}_2) - vars((\overline{H}, c, \overline{G}_L, \overline{G}_R)\sigma)$. Thus, there exists a ground substitution η such that $dom(\eta) = Z$ and $M(\overline{P}_0) \models (c_2 \wedge \overline{G}_2)\sigma\eta$. Therefore, $M(\overline{P}_0) \models (c \wedge c_2 \wedge \overline{G}_L \wedge \overline{G}_2 \wedge \overline{G}_R)\sigma\eta$. Let us now consider the ground instance $\overline{D}\sigma\eta$: $(\overline{H} \leftarrow c \wedge c_2 \wedge \overline{G}_L \wedge \overline{G}_2 \wedge \overline{G}_R)\sigma\eta$ of $\overline{D}$. We have that: $hd(\overline{D})\sigma\eta = \overline{H}\sigma\eta = \overline{H}\sigma$ (because $\overline{H}\sigma$ is a ground annotated atom) $= hd(\overline{C}\sigma)$ and $M(\overline{P}_0) \models bd(\overline{D}\sigma\eta)$.

Proof of (3). Straightforward from Condition (δ) of Rule R3. □

Proof of the Replacement Law of Section 6

We want to prove that the replacement law

$\tau:\ test([1|X])\langle N\rangle \Rightarrow_{\{X,N\}} N \geq N_1 \wedge test(X)\langle N_1\rangle$

holds in the annotated program $\overline{P}$ considered in Section 6. (Note that we have renamed the annotation variables.) We start off by introducing the following two clauses:

$\overline{D}_1:\ new1(X)\langle N\rangle \leftarrow test([1|X])\langle N\rangle$
$\overline{D}_2:\ new2(X)\langle N\rangle \leftarrow N \geq N_1 \wedge test(X)\langle N_1\rangle$

Let us construct a transformation sequence from $\overline{P} \cup \{\overline{D}_1\}$. By unfolding, from clause $\overline{D}_1$ we get:

$\overline{E}_1:\ new1(X)\langle N\rangle \leftarrow N > N_1 \wedge N_1 > N_2 \wedge canon(X)\langle N_2\rangle$
$\overline{E}_2:\ new1(X)\langle N\rangle \leftarrow N > N_1 + N_2 \wedge N_1 > N_3 \wedge trans(X,Y)\langle N_3\rangle \wedge test([1|Y])\langle N_2\rangle$

By annotation weakening and variable renaming we get:

$\overline{E}_3:\ new1(X)\langle N\rangle \leftarrow N > N_1 \wedge canon(X)\langle N_1\rangle$
$\overline{E}_4:\ new1(X)\langle N\rangle \leftarrow N > N_1 + N_2 \wedge trans(X,Y)\langle N_1\rangle \wedge test([1|Y])\langle N_2\rangle$

By folding clause $\overline{E}_4$ using clause $\overline{D}_1$ we derive:

$\overline{E}_5:\ new1(X)\langle N\rangle \leftarrow N > N_1 + N_2 \wedge trans(X,Y)\langle N_1\rangle \wedge new1(Y)\langle N_2\rangle$

The final program of the transformation sequence starting from $\overline{P} \cup \{\overline{D}_1\}$ is $\overline{P} \cup \{\overline{E}_3, \overline{E}_5\}$. Now we construct a symmetric transformation sequence starting from $\overline{P} \cup \{\overline{D}_2\}$. By unfolding clause $\overline{D}_2$ we derive:

$\overline{F}_1:\ new2(X)\langle N\rangle \leftarrow N \geq N_1 \wedge N_1 > N_2 \wedge canon(X)\langle N_2\rangle$
$\overline{F}_2:\ new2(X)\langle N\rangle \leftarrow N \geq N_1 \wedge N_1 > N_2 + N_3 \wedge trans(X,Y)\langle N_2\rangle \wedge test(Y)\langle N_3\rangle$

By symmetric applications of the goal replacement rule, consisting in the replacement of annotation formulas by equivalent ones (in particular, here we use the equivalence $\mathcal{N} \models \forall\,(N_1 > N_2+N_3 \leftrightarrow \exists\, N_4(N_1 > N_2+N_4 \wedge N_4 \geq N_3))$), and by variable renaming, we get:

$\overline{F}_3:\ \ new2(X)\langle N\rangle \leftarrow N > N_1 \ \wedge\ canon(X)\langle N_1\rangle$

$\overline{F}_4:\ \ new2(X)\langle N\rangle \leftarrow N > N_1+N_2 \ \wedge\ N_2 \geq N_3 \ \wedge\ trans(X,Y)\langle N_1\rangle \ \wedge\ test(Y)\langle N_3\rangle$

By folding clause F_4 using clause D_2 and by variable renaming, we derive:

$\overline{F}_5:\ \ new2(X)\langle N\rangle \leftarrow N > N_1+N_2 \ \wedge\ trans(X,Y)\langle N_1\rangle \ \wedge\ new2(Y)\langle N_2\rangle$

The final program of the transformation sequence starting from $\overline{P} \cup \{\overline{D}_2\}$ is $\overline{P} \cup \{\overline{F}_3, \overline{F}_5\}$. Since $\overline{P} \cup \{\overline{D}_1\}$ is syntactically equivalent to $\overline{P} \cup \{\overline{D}_2\}$, we have proved that the replacement law τ holds in $\overline{P}$.

References

[1] K. R. Apt, "Introduction to logic programming," in *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), pp. 493–576, Elsevier, 1990.

[2] K. R. Apt and D. Pedreschi, "Reasoning about termination of pure logic programs," *Information and Computation*, vol. 106, pp. 109–157, 1993.

[3] M. Bezem, "Strong termination of logic programs," *Journal of Logic Programming*, vol. 15, pp. 79–97, 1993.

[4] A. Bossi, N. Cocco, and S. Etalle, "On safe folding," in *Proceedings PLILP '92, Leuven, Belgium*, Lecture Notes in Computer Science 631, pp. 172–186, Springer-Verlag, 1992.

[5] A. Bossi, N. Cocco, and S. Etalle, "Simultaneous replacement in normal programs," *Journal of Logic and Computation*, vol. 6, no. 1, pp. 79–120, 1996.

[6] A. Bossi and S. Etalle, "Transforming acyclic programs," *ACM Transactions on Programming Languages and Systems*, vol. 16, pp. 1081–1096, July 1994.

[7] R. M. Burstall and J. Darlington, "A transformation system for developing recursive programs," *Journal of the ACM*, vol. 24, pp. 44–67, January 1977.

[8] J. Cook and J. P. Gallagher, "A transformation system for definite programs based on termination analysis," in *Proceedings of LoPSTr'94 and META'94, Pisa, Italy* (L. Fribourg and F. Turini, eds.), Lecture Notes in Computer Science 883, pp. 51–68, Springer-Verlag, 1994.

[9] B. Courcelle, "Recursive applicative program schemes," in *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), vol. B, pp. 459–492, Elsevier, 1990.

[10] N. Dershowitz, "Termination of rewriting," *Journal of Symbolic Computation*, vol. 3, no. 1-2, pp. 69–116, 1987.

[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)* (M. Hermenegildo and J. Penjam, eds.), Lecture Notes in Computer Science 844, pp. 340–354, Springer-Verlag, 1994.

[12] J. Jaffar, M. Maher, K. Marriott, and P. Stuckey, "The semantics of constraint logic programming," *Journal of Logic Programming*, vol. 37, pp. 1–46, 1998.

[13] T. Kanamori and H. Fujita, "Unfold/fold transformation of logic programs with counters," Technical Report 179, ICOT, Tokyo, Japan, 1986.

[14] K.-K. Lau, M. Ornaghi, A. Pettorossi, and M. Proietti, "Correctness of logic program transformation based on existential termination," in *Proceedings of the 1995 International Logic Programming Symposium (ILPS '95)* (J. W. Lloyd, ed.), pp. 480–494, MIT Press, 1995.

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

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

[17] Z. Manna and A. Pnueli, "Axiomatic approach to total correctness of programs," *Acta Informatica*, vol. 3, pp. 243–263, 1974.

[18] J. McCarthy, "Towards a mathematical science of computation," in *Information Processing: Proceedings of IFIP 1962* (C. Popplewell, ed.), (Amsterdam), pp. 21–28, North Holland, 1963.

[19] H. A. Partsch, *Specification and Transformation of Programs*. Springer-Verlag, 1990.

[20] A. Pettorossi and M. Proietti, "Synthesis and transformation of logic programs using unfold/fold proofs," *Journal of Logic Programming*, vol. 41, no. 2&3, pp. 197–230, 1999.

[21] G. D. Plotkin, "A structural approach to operational semantics," Tech. Rep. DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, 1981.

[22] M. Proietti and A. Pettorossi, "Transforming inductive definitions," in *Proceedings of the 1999 International Conference on Logic Programming* (D. De Schreye, ed.), pp. 486–499, MIT Press, 1999.

[23] W. Pugh, "A practical algorithm for exact array dependence analysis," *Communications of the ACM*, vol. 35, no. 8, pp. 102–114, 1992.

[24] A. Roychoudhury, K. N. Kumar, C. R. Ramakrishnan, and I. V. Ramakrishnan, "An unfold/fold transformation framework for definite logic programs," *ACM Transactions on Programming Languages and Systems*, vol. 26, pp. 264–509, 2004.

[25] D. Sands, "Total correctness by local improvement in the transformation of functional programs," *ACM Toplas*, vol. 18, no. 2, pp. 175–234, 1996.

[26] D. Sands, "From SOS rules to proof principles: An operational metatheory for functional languages," in *Proceedings of the 24th ACM Symposium on Principles of Programming Languages (POPL '97)*, pp. 428–441, ACM Press, 1997.

[27] H. Seki, "Unfold/fold transformation of stratified programs," *Theoretical Computer Science*, vol. 86, pp. 107–139, 1991.

[28] H. Tamaki and T. Sato, "Unfold/fold transformation of logic programs," in *Proceedings of the Second International Conference on Logic Programming* (S.-Å. Tärnlund, ed.), (Uppsala, Sweden), pp. 127–138, Uppsala University, 1984.

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