A Theory of Totally Correct Logic Program Transformations

Alberto Pettorossi
DISP, University of Roma Tor Vergata
Via del Politecnico 1
I-00133 Roma, Italy
adp@iasi.rm.cnr.it

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

ABSTRACT

We address the problem of proving total correctness of transformation rules for 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 clause replacement is 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 constructing from the given program P an annotated program $\alpha(P)$, (ii) applying clause replacements and transforming the annotated program $\alpha(P)$ into a *terminating* annotated program $\beta(Q)$, and (iii) erasing the annotations from $\beta(Q)$, thereby getting Q.

Our method does not require that either P or Q terminates and it is parametric w.r.t. the annotations. By providing different definitions for these annotations, we can easily prove the total correctness of many versions of the unfolding, folding, and goal replacement rules proposed in the literature.

Categories and Subject Descriptors

F.3.1 [**Logics and Meaning of Programs**]: Specifying and Verifying and Reasoning about Programs, Semantics of Programming Languages; D.1.2 [**Programming Techniques**]: Automatic Programming—*Program Transformation*; D.3.2 [**Programming Languages**]: Language Classification—*Constraint and Logic Languages.*

General Terms

Languages, Theory, Verification.

Keywords

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

1. INTRODUCTION

Rules for program transformation can be viewed as conditional rewritings of programs. Indeed the transformation of a program P into a program Q can be realized by using rules, each of which rewrites a statement s_1 of the program P into a new statement s_2, provided that s_1 and s_2 are equivalent w.r.t. a suitable semantics (see, for instance, [13]).

In this paper we consider definite logic programs together with a general transformation rule, called *clause replacement*, which is a conditional rewriting of programs 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 logical equivalence between Γ_1 and Γ_2 holds in the least Herbrand model $M(P)$ of the program P. Most transformation rules proposed in the literature, including the popular unfolding and folding rules [5, 18], can be viewed as particular cases of the clause replacement rule.

We will give the formal definition of the clause replacement rule in Section 2. In this Introduction, we will use the following particular instance of the clause replacement rule, called *goal replacement*: 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 goals G_1 and G_2 are equivalent in the least Herbrand model of P, that is, $M(P) \models \forall\,(G_1 \leftrightarrow G_2)$, where $\forall(\varphi)$ denotes the universal closure of formula φ.

Much work has been devoted to the study of the correctness of program transformation rules for definite programs (see, for instance, [3, 8, 10, 11, 16, 18, 19]). Two notions of correctness properties of the rules have been considered: *partial correctness* and *total correctness.* A rule which transforms 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 $M(P) = M(Q)$.

We will show in Section 2 that the partial correctness of the clause replacement rule is a straightforward consequence of the fact that this rule is based on a logical equivalence. However, the application of logical equivalences does not ensure the opposite inclusion, i.e., $M(P) \subseteq M(Q)$. Indeed, in general, clause replacement is *not* totally correct. We show this point in the particular case of goal replacement, by means of the following example.

Example 1. Let us consider the transformation of program P into program Q, where P and Q are as follows:

$$P\!: \quad \begin{array}{l} p \leftarrow q \\ q \leftarrow \end{array} \qquad\qquad Q\!: \quad \begin{array}{l} p \leftarrow p \\ q \leftarrow \end{array}$$

The transformation of P into Q is an application of the goal replacement rule defined above, which is justified because

$M(P) \models p \leftrightarrow q$ holds. This goal replacement is not totally correct, because we have that $M(P) = \{p, q\} \supset \{q\} = M(Q)$. $\Box$

Since the pioneering work by Tamaki and Sato [18], various authors have proposed suitable extra conditions which ensure the total correctness of the unfolding, folding, and goal replacement rules [8, 10, 16, 18, 19]. However, the verification of these conditions requires the proof of some invariants of the transformation of program P into program Q, and these invariants refer to complex measures of the proofs of the atoms which are in $M(P)$.

For instance, Tamaki and Sato proved that the replacement of goal G_1 by goal G_2 is totally correct if, in addition to the condition $M(P) \models \forall (G_1 \leftrightarrow G_2)$, we have that for all ground instances $G_1\vartheta$ and $G_2\vartheta$ of G_1 and G_2, respectively, if $G_1\vartheta$ has a proof in P then also $G_2\vartheta$ has a proof in P and this proof of $G_2\vartheta$ has a measure which is not larger than the one of $G_1\vartheta$. The measure of a proof is defined in terms of the number of nodes in the proof tree [18]. More sophisticated measures are defined in [16, 19]. However, no general methodology is given in [16, 18, 19] for comparing proof measures and checking the conditions which ensure the total correctness of goal replacements.

The main contribution of this paper is a method for proving the total correctness of the clause replacement rule and its particular cases consisting of the unfolding, folding, and goal replacement rules. By our method we can express the conditions which ensure total correctness of clause replacement as first order formulas that can be checked by standard deductive techniques.

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

In order to show the total correctness of the replacement of G_1 by G_2, we may use the *unique fixpoint principle*, which can be formulated as follows: *if* $M(P) \models \forall (G_1 \rightarrow G_2)$ and the immediate consequence operator T_Q associated with Q has a unique fixpoint, *then* $M(P) \subseteq M(Q)$ [6, 15]. A sufficient (but not necessary) condition ensuring that T_Q has a unique fixpoint is that, for all ground goals, Q *terminates* with success or failure [2]. However, this condition based on the existence of a unique fixpoint is too restrictive in practice, because for many useful non-terminating logic programs the immediate consequence operator T_Q does not have a unique fixpoint.

We overcome this limitation by introducing *program annotations* as we now indicate. For reasons of simplicity, let us assume that every clause of P is of the form $H \leftarrow A_1$, where A_1 is an atom. When A_1 is *true* the clause $H \leftarrow A_1$ will also be written as $H \leftarrow$. The general case where the bodies are conjunctions of n atoms, with $n \geq 0$, is analogous and we will consider it in Sections 3 and 4. An annotation for program P is a function α that associates with every clause γ: $H \leftarrow A_1$ of P an annotated clause $\alpha(\gamma)$ of the form:

$$H\langle X\rangle \leftarrow c_1(X, Y) \wedge A_1\langle Y\rangle$$

where: (i) X and Y are distinct *annotation variables* ranging over a given set W, and (ii) $c_1(X, Y)$ is an *annotation formula*, denoting a binary relation on $W \times W$. The annotation variables should be considered as extra arguments of the atoms in the clause γ. For instance, the annotated atom $p(a)\langle Y\rangle$ should be considered identical to the atom $p(a, Y)$. Thus, by considering the annotation formulas as *constraints*, the annotated program $\alpha(P)$ is a constraint logic program for which we can define a least model, denoted by $M(\alpha(P))$ [9].

Now, let us suppose that we replace A_1 in the body of clause γ by a new atom A_2, thereby deriving a new program Q. Our method for proving $M(P) \subseteq M(Q)$ consists in showing the following three properties.

(1) The program annotation α is *enhancing*, that is, for every ground atom $A \in M(P)$ there exists $w \in W$ such that $A\langle w\rangle \in M(\alpha(P))$.

(2) For some annotation formula $c_2(X, Y)$, we have that:

$$M(\alpha(P)) \models \forall X\, (\exists Y (c_1(X, Y) \wedge A_1\langle Y\rangle) \rightarrow \exists Y (c_2(X, Y) \wedge A_2\langle Y\rangle)).$$

(3) Let $\beta(Q)$ be the annotated program obtained by replacing $c_1(X, Y) \wedge A_1\langle Y\rangle$ by $c_2(X, Y) \wedge A_2\langle Y\rangle$. We have that β is a *well-founded* annotation for Q, that is, for every annotated clause $K\langle X\rangle \leftarrow d(X, Y) \wedge B\langle Y\rangle$ in $\beta(Q)$, the implication $\forall X\, \forall Y\, (d(X, Y) \rightarrow X \succ Y)$ holds, where $\succ$ is a well-founded ordering on W (that is, no infinite descending sequence $w_1 \succ \ldots \succ w_n \succ \ldots$ exists in W).

Let us briefly explain the reasons why Properties (1)–(3) ensure that $M(P) \subseteq M(Q)$. Let us take $A \in M(P)$. By Property (1) there exists $w \in W$ such that $A\langle w\rangle \in M(\alpha(P))$. In Section 3 we will prove that if α is a well-founded annotation for a program P, then $T_{\alpha(P)}$ has a unique fixpoint. Thus, by Properties (2) and (3), and by the unique fixpoint principle, we have that $M(\alpha(P)) \subseteq M(\beta(Q))$ and, hence, $A\langle w\rangle \in M(\beta(Q))$. (To see that Property (2) is an instance of $M(\alpha(P)) \models \forall (G_1 \rightarrow G_2)$, we observe that the variables occurring in the body of a clause and not occurring in the head, can be considered as variables which are existentially quantified in front of the body.) Finally, since for every annotated atom $A\langle w\rangle$, if $A\langle w\rangle \in M(\beta(Q))$ then $A \in M(Q)$, we conclude that $A \in M(Q)$.

In practice, in order to prove Property (3), that is, the well-foundedness of β, we usually start from an annotation α for P which is well-founded, and then we prove that the goal replacement preserves well-foundedness. Then, to prove that goal replacement preserves well-foundedness it is enough to show that the following implication holds:

(3*) $\forall X\, \forall Y\, (c_2(X, Y) \rightarrow X \succ Y)$

Thus, starting from an annotated program $\alpha(P)$ where α is enhancing and well-founded, at every transformation step we only need to prove that Properties (2) and (3*) hold, and these proofs, as already mentioned, can be made by using standard deductive techniques. In particular, many powerful well-founded orderings can be axiomatized as first order theories, and their properties can be proved by using techniques developed in the field of term rewriting systems [7].

Notice also that the property that α is an enhancing, well-founded annotation for P is, in fact, independent of the program P and can be proved in advance for many program annotations. We will give examples of these annotations in Section 3.

Finally, we would like to stress the fact that, when applying our method for proving the total correctness of the transformation of program P into program Q, neither P nor Q is required to terminate.

Now we present a simple example which shows that our method can be applied also for proving the total correctness of transformations realized by applications of the unfolding and folding rules.

Example 2. Let us consider the following program P:

1. $q(a) \leftarrow$
2. $q(A) \leftarrow q(A)$
3. $p \leftarrow q(f(B))$

In order to construct a totally correct transformation, we first consider the following annotated program $\alpha(P)$:

1a. $q(a)\langle X\rangle \leftarrow$
2a. $q(A)\langle X\rangle \leftarrow X > Y \wedge q(A)\langle Y\rangle$
3a. $p\langle X\rangle \leftarrow X > Y \wedge q(f(B))\langle Y\rangle$

where X and Y range over natural numbers and $>$ is the usual 'greater than' ordering on natural numbers. Then we apply the unfolding and folding transformation rules to the annotated program $\alpha(P)$. By unfolding clause 3a w.r.t. $q(f(B))\langle Y\rangle$ we derive:

4a. $p\langle X\rangle \leftarrow X > Y \wedge Y > Z \wedge q(f(B))\langle Z\rangle$

By folding clause 4a w.r.t. $Y > Z \wedge q(f(B))\langle Z\rangle$ we derive:

5a. $p\langle X\rangle \leftarrow X > Y \wedge p\langle Y\rangle$

The annotated program derived by the above applications of the unfolding and folding rules is $\beta(Q) = \{1a, 2a, 5a\}$. Let us consider the program obtained by erasing the annotations from $\beta(Q)$, that is, $Q = \{1, 2, 5\}$, where clause 5 is:

5. $p \leftarrow p$

The partial correctness of the transformation of P into Q follows from the partial correctness of the usual unfolding and folding transformations for non-annotated programs. Indeed, P can be transformed into Q by applying the usual unfolding and folding rules, in the same way as $\alpha(P)$ has been transformed into $\beta(Q)$. Thus, $M(P) \supseteq M(Q)$.

The total correctness of the transformation of P into Q can be proved by our method based on well-founded annotations similarly to the case of goal replacement. In particular, α is enhancing (and well-founded) and β is well-founded, because in every clause of $\beta(Q)$ the annotation of the head is greater than the annotation of each atom in the body (for instance, in clause 5a, we have $X > Y$). Thus, $M(P) \subseteq M(Q)$ and the transformation of the given program P into the final program Q is totally correct, that is, $M(P) = M(Q)$. □

Since our method is sound (see Section 3), it cannot be used to prove the total correctness of the transformation of Example 1 which, indeed, is not totally correct. Below we show why this proof fails.

Example 3. Let us consider the following program $\alpha(P)$, which is an annotated version of the program P presented in Example 1:

1a. $p\langle X\rangle \leftarrow X > Y \wedge q\langle Y\rangle$
2a. $q\langle X\rangle \leftarrow$

The annotation α is enhancing and well-founded.
By clause 1a we have that:

$$M(\alpha(P)) \models \forall X\,(\exists Y\,(X > Y \wedge q\langle Y\rangle) \rightarrow (\exists Y\, X = Y \wedge p\langle Y\rangle))$$

Thus, Property (2) of our method for the total correctness of goal replacement is satisfied. However, if we replace $X > Y \wedge q\langle Y\rangle$ by $X = Y \wedge p\langle Y\rangle$ in clause 1a, we get a new annotated program whose annotation is not well-founded, because $X = Y$ does not imply that $X \succ Y$ for any well-founded ordering $\succ$, and thus, Property (3) of our method is not satisfied. □

The paper is structured as follows. In Section 2 we introduce the clause replacement transformation rule, which generalizes the unfolding, folding, and goal replacement transformations. We prove the partial correctness of clause replacement, as well as other relevant properties. In Section 3 we introduce program annotations and, in particular, well-founded annotations. Then we prove a sufficient condition for ensuring the total correctness of clause replacement based on well-founded annotations. 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 these rules are totally correct. In Section 5 we present an extended example of application of the unfolding, folding, and goal replacement rules. Finally, in Section 6 we compare our method to related techniques for proving total correctness of program transformations. In particular, we argue that by our method we can easily prove total correctness of the various versions of the unfolding/folding rules proposed in the literature (see, for instance, [8, 10, 16, 18, 19]).

2. CLAUSE REPLACEMENT

In this section we introduce the *clause replacement* transformation rule. All usual program transformation rules, such as unfolding, folding, and goal replacement, are instances of this clause replacement rule. Indeed, we prove that clause replacement is the most general program transformation rule, in the sense that, any totally correct program transformation can be obtained by applying this rule (see Theorem 1). Then we study the partial and total correctness of clause replacement (see Theorems 2 and 3). In particular, we give a sufficient condition for the total correctness of clause replacement, which is based on the uniqueness of the fixpoint of the immediate consequence operator of the annotated transformed program (see Corollary 1).

In order to define the clause replacement rule we introduce the following implications and equivalences between sets of clauses[1].

Definition 1. Let I be a Herbrand interpretation and let Γ_1, Γ_2 be two sets of clauses. We write $I \models \Gamma_1 \Rightarrow \Gamma_2$ iff for every ground instance $H \leftarrow G_2$ of a clause in Γ_2 such that $I \models G_2$ there exists a ground instance $H \leftarrow G_1$ of a clause in Γ_1 such that $I \models G_1$. We write $I \models \Gamma_1 \Leftarrow \Gamma_2$ iff $I \models \Gamma_2 \Rightarrow \Gamma_1$ and we write $I \models \Gamma_1 \Leftrightarrow \Gamma_2$ iff $I \models \Gamma_1 \Rightarrow \Gamma_2$ and $I \models \Gamma_1 \Leftarrow \Gamma_2$.

For every Herbrand interpretation I and sets of clauses $\Gamma, \Gamma_1, \Gamma_2, \Gamma_3$ the following properties hold:

Reflexivity: $I \models \Gamma \Rightarrow \Gamma$
Transitivity: if $I \models \Gamma_1 \Rightarrow \Gamma_2$ and $I \models \Gamma_2 \Rightarrow \Gamma_3$ then $I \models \Gamma_1 \Rightarrow \Gamma_3$

[1]The notions we introduce in the following Definitions 1 and 2 are symmetric w.r.t. those introduced in [15]. The reason is that here we deal with implications between sets of *clauses*, while in [15] we deal with implications between sets of *bodies of clauses*.

Monotonicity: if $I \models \Gamma_1 \Rightarrow \Gamma_2$ then $I \models \Gamma_1 \cup \Gamma \Rightarrow \Gamma_2 \cup \Gamma$.

Definition 2. A *clause replacement* is a pair (P, Q) of programs, denoted $P \mapsto Q$, such that, for some sets Γ_1 and Γ_2 of clauses with $\Gamma_1 \subseteq P$, we have: $Q = (P - \Gamma_1) \cup \Gamma_2$. Program Q is also denoted $P[\Gamma_1/\Gamma_2]$. A clause replacement $P \mapsto P[\Gamma_1/\Gamma_2]$ is said to be: (i) *implication-based* iff $M(P) \models \Gamma_1 \Rightarrow \Gamma_2$, (ii) *reverse-implication-based* iff $M(P) \models \Gamma_1 \Leftarrow \Gamma_2$, and (iii) *equivalence-based* iff $M(P) \models \Gamma_1 \Leftrightarrow \Gamma_2$. A clause replacement $P \mapsto Q$ is said to be: (iv) *partially correct* iff $M(P) \supseteq M(Q)$, (v) *increasing* iff $M(P) \subseteq M(Q)$, and (vi) *totally correct* iff $M(P) = M(Q)$.

By monotonicity, we have the following property which will be useful in the proofs of (partial and total) correctness of clause replacement.

LEMMA 1. *Let $P \mapsto Q$ be a clause replacement.*
(i) $P \mapsto Q$ is implication-based iff $M(P) \models P \Rightarrow Q$.
(ii) $P \mapsto Q$ is reverse-implication-based iff $M(P) \models P \Leftarrow Q$.

The following theorem, whose proof is left to the reader, shows that equivalence-based clause replacement is a *complete* transformation rule for the derivation of equivalent programs, in the sense that, for any two programs P and Q such that $M(P) = M(Q)$, there exists an equivalence-based clause replacement $P \mapsto Q$.

THEOREM 1. (Completeness of Equivalence-Based Clause Replacement) *Given two programs P and Q, if $M(P) = M(Q)$ then $M(P) \models P \Leftrightarrow Q$.*

Now we present some sufficient conditions for ensuring that a clause replacement is partially correct and increasing. Given a program P, we denote its associated *immediate consequence operator* by T_P [1]. We denote the least and greatest fixpoint of T_P by $lfp(T_P)$ and $gfp(T_P)$, respectively. Recall that $M(P) = lfp(T_P)$.

First we show that every implication-based clause replacement is partially correct.

THEOREM 2. (Partial Correctness) *Given two programs P and Q, if $P \mapsto Q$ is an implication-based clause replacement then $M(P) \supseteq M(Q)$.*

PROOF. We first show that $M(P)$ is a prefixpoint of T_Q, that is, $T_Q(M(P)) \subseteq M(P)$. Let A be a ground atom in $T_Q(M(P))$. By definition of T_Q there exists a ground instance $A \leftarrow G_2$ of a clause in Q such that $M(P) \models G_2$. Since $P \mapsto Q$ is an implication-based clause replacement, by Lemma 1 $M(P) \models P \Rightarrow Q$. Thus, by Definition 1, there exists a ground instance $A \leftarrow G_1$ of a clause in P such that $M(P) \models G_1$. Hence, by definition of T_P, $A \in T_P(M(P))$. Since $M(P)$ is a prefixpoint of T_P, we have that $T_P(M(P)) \subseteq M(P)$ and, therefore, $A \in M(P)$. Thus, we have proved that $M(P)$ is a prefixpoint of T_Q. Since $M(Q) = lfp(T_Q)$ and $lfp(T_Q)$ is the least prefixpoint of T_Q (see, for instance, [1]), we have that $M(P) \supseteq M(Q)$. □

Since every equivalence-based clause replacement is an implication-based clause replacement, Theorem 2 proves also that equivalence-based clause replacements are partially correct.

In order to prove that a clause replacement is increasing we propose a method based on the *unique fixpoint principle* [6, 15].

A program P is said to be *univocal* iff T_P has a unique fixpoint, that is, $lfp(T_P) = gfp(T_P)$. A sufficient condition for a program to be univocal is that it is *terminating* [2].

THEOREM 3. (Increasingness) *Given two programs P and Q, if $P \mapsto Q$ is a reverse-implication-based clause replacement and Q is univocal then $M(P) \subseteq M(Q)$.*

PROOF. We first show that $M(P)$ is a postfixpoint of T_Q, that is, $T_Q(M(P)) \supseteq M(P)$. Let A be a ground atom in $M(P)$. Since $M(P)$ is a fixpoint of T_P, that is, $T_P(M(P)) = M(P)$, we have that there exists a ground instance $A \leftarrow G_1$ of a clause in P such that $M(P) \models G_1$. By Lemma 1, $M(P) \models P \Leftarrow Q$ and, therefore, by Definition 1, there exists a ground instance $A \leftarrow G_2$ of a clause in Q such that $M(P) \models G_2$. By definition of T_Q, we have that $A \in T_Q(M(P))$. Thus, we have proved that $M(P)$ is a postfixpoint of T_Q. Since $gfp(T_Q)$ is the greatest postfixpoint of T_Q (see, for instance, [1]), we have that $M(P) \subseteq gfp(T_Q)$. Finally, by the hypothesis that Q is univocal, we get that $M(P) \subseteq M(Q)$. □

As a consequence of Theorems 2 and 3 we have the following result.

COROLLARY 1. (Total Correctness via Unique Fixpoint) *Given two programs P and Q, if $P \mapsto Q$ is an equivalence-based clause replacement and Q is univocal then $M(P) = M(Q)$.*

Corollary 1 gives us a useful method for proving the total correctness of clause replacements. However, from a practical point of view, this method has the following two limitations: (1) the property that a program is univocal is, in general, undecidable, and (2) the method cannot be applied when the program derived by clause replacement, is *not* univocal. In the next section we will present a method that partly overcomes these limitations.

3. WELL-FOUNDED ANNOTATIONS

In this section we present our method based on the notion of *program annotation* for proving the total correctness of clause replacements. In particular, we will introduce the so called *well-founded* annotations, that is, annotations which generate terminating (and, thus, univocal) annotated programs. First, we present the syntax and the semantics of annotated programs and then we introduce the notion of clause replacement for the class of annotated programs. Finally, we prove the main result of this paper, that is, a sufficient condition for ensuring the total correctness of clause replacements (see Theorem 6 below).

The syntax of annotated programs is defined as follows. We consider a first order language L_A for writing annotations. We assume that L_A is disjoint from the first order language L_P for writing programs. We also assume that in the set of predicate symbols of L_A there is the symbol $\succ$, which is interpreted as a well-founded ordering relation on a suitable domain. Variables, terms, and formulas of L_A are called *annotation variables*, *annotation terms*, and *annotation formulas*, respectively. For reasons of simplicity, we assume that annotation formulas do not have bound variables. 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. An *annotated clause* is of the form:

$H\langle w\rangle \leftarrow c \wedge A_1\langle w_1\rangle \wedge \ldots \wedge A_n\langle w_n\rangle$

where (i) $w, w_1, \ldots, w_n$ are annotation terms, (ii) c is an annotation formula, and (iii) $H\langle w\rangle$, $A_1\langle w_1\rangle, \ldots, A_n\langle w_n\rangle$ are annotated atoms. An *annotated program* is a set of annotated clauses. Annotated atoms, conjunctions of annotated atoms, and annotated programs are denoted by overlined metavariables, such as $\overline{A}$, $\overline{G}$, and $\overline{P}$.

The definition of the semantics of annotated programs is similar to the one of *constraint logic programs* [9] and it is given as follows. We fix an interpretation $\mathcal{W}$ for L_A, whose carrier is a set W. We assume that the predicate symbol $\succ$ is interpreted as a well-founded ordering relation on W which, by abuse of language, we will also denote by $\succ$. For any annotation formula c, the satisfaction relation $\mathcal{W} \models c$ is defined as usual in first order logic. The interpretation $\mathcal{W}$ will also be referred to as the *well-founded ordering* $(W, \succ)$. A $\mathcal{W}$*-interpretation* is a subset of

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

Given a $\mathcal{W}$-interpretation I, a ground annotation formula c, and a ground conjunction $\overline{A}_1 \wedge \ldots \wedge \overline{A}_n$ of annotated atoms, the satisfaction relation $I \models c \wedge \overline{A}_1 \wedge \ldots \wedge \overline{A}_n$ holds iff $\mathcal{W} \models c$ and, for $i = 1, \ldots, n$, $\overline{A}_i \in I$. A ground annotated clause $\overline{H} \leftarrow c \wedge \overline{G}$, where $\overline{G}$ is a conjunction of annotated atoms, is true in a $\mathcal{W}$-interpretation I iff $I \models c \wedge \overline{G}$ implies $\overline{H} \in I$. An annotated clause is true in a $\mathcal{W}$-interpretation I iff all its ground instances are true in I. A $\mathcal{W}$*-model* of an annotated program $\overline{P}$ is a $\mathcal{W}$-interpretation such that all annotated clauses in $\overline{P}$ are true. It can be shown that every annotated program $\overline{P}$ has a least $\mathcal{W}$-model which is denoted by $M(\overline{P})$ (this is the same notation which is used for the least Herbrand model of definite logic programs).

Similarly to the case of definite logic programs, the least $\mathcal{W}$-model of an annotated program can be computed as the least fixpoint of a particular operator over $\mathcal{W}$-interpretations. With every annotated program $\overline{P}$ we associate an *immediate consequence operator* $T_{\overline{P}}$ from $\mathcal{P}(B_{\mathcal{W}})$ to $\mathcal{P}(B_{\mathcal{W}})$, 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 logic programs (see, for instance, [1]), we have the following result.

Theorem 4. *For every annotated program $\overline{P}$, $T_{\overline{P}}$ is a continuous function from $\mathcal{P}(B_{\mathcal{W}})$ to $\mathcal{P}(B_{\mathcal{W}})$ and $lfp(T_{\overline{P}}) = M(\overline{P})$.*

In the next definition we introduce the main notion of this paper. In this definition we consider the following four sets: *Clauses*, *AClauses*, *Programs*, and *APrograms*, which consist of all clauses, annotated clauses, programs, and annotated programs, respectively. We will also 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. An *annotation over* $\mathcal{W}$ is a function α : *Clauses* $\rightarrow$ *AClauses* such that, for every clause γ of the form $H \leftarrow A_1 \wedge \ldots \wedge A_n$, the annotated clause $\alpha(\gamma)$ is of the form $H\langle X\rangle \leftarrow c \wedge A_1\langle X_1\rangle \wedge \ldots \wedge A_n\langle X_n\rangle$, where $X, X_1, \ldots, X_n$ are annotation variables. An annotation α : *Clauses* $\rightarrow$ *AClauses* can be extended to a function, also denoted by α, from *Programs* to *APrograms*, by stipulating that, for every program $\{\gamma_1, \ldots, \gamma_n\}$, $\alpha(\{\gamma_1, \ldots, \gamma_n\})$ is $\{\alpha(\gamma_1), \ldots, \alpha(\gamma_n)\}$. Let P be a program in *Programs*.

(i) An annotation α is said to be *well-founded* (for P) iff for every annotated clause $\alpha(\gamma)$: $\overline{H} \leftarrow c \wedge \overline{A}_1 \wedge \ldots \wedge \overline{A}_n$ of $\alpha(P)$, we have that:

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

(ii) An annotation α is said to be *enhancing* (for P) iff for every $A \in M(P)$ there exists a ground annotation term w such that $A\langle w\rangle \in M(\alpha(P))$.

Now we give some examples of annotations.

Example 4. Let $\mathcal{N}$ be the well-founded ordering $(Nat, >)$, where *Nat* is the set of the natural numbers and $>$ is the usual 'greater than' ordering on *Nat*.

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

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

(ii) The annotation α_2 over $\mathcal{N}$ is defined as follows: for every clause γ: $H \leftarrow A_1 \wedge \ldots \wedge A_n$, the annotated clause $\alpha_2(\gamma)$ 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. The annotations α_1 and α_2 are enhancing and well-founded. In particular, we have that:

$$\mathcal{N} \models \forall X\, \forall X_1 \ldots \forall X_n\, (X > X_1 + \ldots + X_n \rightarrow X > X_1 \wedge \ldots \wedge X > X_n). \qquad \Box$$

In the following example we present a well-founded annotation which is *not* enhancing.

Example 5. Let us consider the following program P:

$p \leftarrow q$
$q \leftarrow$

Let us also consider the annotation α_3 over $\mathcal{N}$ that associates with P the following annotated program:

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

We have that $M(P) = \{p, q\}$, while $M(\alpha_3(P)) = \{q\langle 1\rangle\}$ and, thus, α_3 is not enhancing. $\Box$

In order to erase annotations from annotated clauses we use the projection function π : *AClauses* $\rightarrow$ *Clauses* such that, for every clause γ and annotation α, we have $\pi(\alpha(\gamma)) = \gamma$. The function π : *AClauses* $\rightarrow$ *Clauses* can be extended to a function, also denoted by π, from *APrograms* to *Programs*, by stipulating that, for every program P and annotation α, we have $\pi(\alpha(P)) = P$.

We have the following straightforward property of the projection function.

Proposition 1. *For every annotated program $\overline{P}$ and ground annotated atom $A\langle w\rangle$, if $A\langle w\rangle \in M(\overline{P})$ then $A \in M(\pi(\overline{P}))$.*

Similarly to programs which are not annotated, we have the following definition of the $\Rightarrow$ relation.

Definition 4. Let I be a $\mathcal{W}$-interpretation and let Γ_1, Γ_2 be sets of annotated clauses. We write $I \models \Gamma_1 \Rightarrow \Gamma_2$ iff for every ground instance $\overline{H} \leftarrow c_2 \wedge \overline{G}_2$ of an annotated clause in Γ_2 such that $I \models c_2 \wedge \overline{G}_2$ there exists a ground instance $\overline{H} \leftarrow c_1 \wedge \overline{G}_1$ of an annotated clause in Γ_1 such that $I \models c_1 \wedge \overline{G}_1$. We write $I \models \Gamma_1 \Leftarrow \Gamma_2$ iff $I \models \Gamma_2 \Rightarrow \Gamma_1$ and we also write $I \models \Gamma_1 \Leftrightarrow \Gamma_2$ iff $I \models \Gamma_1 \Rightarrow \Gamma_2$ and $I \models \Gamma_1 \Leftarrow \Gamma_2$.

The notion of clause replacement for annotated programs can be introduced by a definition similar to Definition 2 of Section 2, where one considers 'annotated programs', instead of 'programs'. Analogously, one can also introduce the notions of: (i) *implication-based*, (ii) *reverse-implication-based*, (iii) *equivalence-based*, (iv) *partially correct*, (v) *increasing*, and (vi) *totally correct* clause replacement for annotated programs. We have that the properties stated by Theorems 1, 2, 3, and Corollary 1 of Section 2, hold for annotated programs as well.

THEOREM 5. *Let α be a well-founded annotation over $\mathcal{W}$. Then, for every program P, the annotated program $\alpha(P)$ is univocal, that is, $lfp(T_{\alpha(P)}) = gfp(T_{\alpha(P)})$, and $M(\alpha(P))$ is the unique fixpoint of $T_{\alpha(P)}$.*

PROOF. Let $\mathcal{W}$ be the well-founded ordering $(W, \succ)$. Assume that I and J are fixpoints of $T_{\alpha(P)}$. By well-founded induction on $\succ$ we prove that: for every ground annotated atom $\overline{A}$, $\overline{A} \in I$ iff $\overline{A} \in J$. The inductive hypothesis is: for every ground annotated atom $\overline{B}$, if $\mathcal{W} \models \overline{A} \succ \overline{B}$ then $\overline{A} \in I$ iff $\overline{A} \in J$. Assume that $\overline{A} \in I$. Since $I = T_{\alpha(P)}(I)$, we have that there exists a clause, say γ, of the form $\overline{A} \leftarrow c \wedge \overline{A}_1 \wedge \ldots \wedge \overline{A}_n$ in $\alpha(P)$ such that $\mathcal{W} \models c$ and, for $i = 1, \ldots, n$, $\overline{A}_i \in I$. Since α is well-founded, we have that, 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 $T_{\alpha(P)}$ 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 $\overline{A} \in I$. $\Box$

We are now able to show the main result of this paper.

THEOREM 6. (Total Correctness via Well-Founded Annotations) *Let P, Q be programs which are not annotated. Let $P \mapsto Q$ be a clause replacement, and α, β be annotations such that:*

(i) *$P \mapsto Q$ is implication-based, that is, $M(P) \models P \Rightarrow Q$,*

(ii) *$\alpha(P) \mapsto \beta(Q)$ is a reverse-implication-based clause replacement, that is, $M(\alpha(P)) \models \alpha(P) \Leftarrow \beta(Q)$,*

(iii) *α is enhancing, and*

(iv) *β is well-founded.*

Then $P \mapsto Q$ is totally correct, that is, $M(P) = M(Q)$.

PROOF. By Hypothesis (i) and Theorem 2, $P \mapsto Q$ is partially correct, that is, $M(P) \supseteq M(Q)$. Let us now prove that $P \mapsto Q$ is increasing, that is, $M(P) \subseteq M(Q)$. Let A be a ground atom in $M(P)$. Since α is enhancing (see Hypothesis (iii)), there exists a ground annotation term w such that $A\langle w\rangle$ belongs to $M(\alpha(P))$. Since β is well-founded (see Hypothesis (iv)), by Theorem 5 $\beta(Q)$ is univocal and, since $\alpha(P) \mapsto \beta(Q)$ is a reverse-implication-based clause replacement (see Hypothesis (ii)), by Theorem 3, $M(\alpha(P)) \subseteq M(\beta(Q))$. Thus, $A\langle w\rangle$ belongs to $M(\alpha(Q))$ and, by Proposition 1, A belongs to $M(Q)$. $\Box$

$$\begin{tikzcd}
P \arrow[r, mapsto] \arrow[d, mapsto, "\alpha"'] & Q \\
\alpha(P) \arrow[r, mapsto] & \beta(Q) \arrow[u, mapsto, "\pi"']
\end{tikzcd}$$

Figure 1: Program Transformation via Well-Founded Annotations.

Notice that the annotation α is required to be enhancing, but not well-founded, while β is required to be well-founded, but not enhancing. In practice, however, it is often useful to start from an enhancing, well-founded annotation α and apply clause replacements that preserve the well-foundedness of annotations, so that the annotation β is well-founded by construction.

Thus, Theorem 6 supports a methodology for program transformation which consists of the following steps (see also Figure 1). Given an initial program P, in order to construct a totally correct transformation starting from P:

(1) first, we choose an annotation α which is enhancing and well-founded;

(2) then, we apply to $\alpha(P)$ a clause replacement $\alpha(P) \mapsto \beta(Q)$ such that:

(i) $M(P) \models P \Rightarrow Q$,

(ii) $M(\alpha(P)) \models \alpha(P) \Leftarrow \beta(Q)$, and

(iii) β is well-founded; and

(3) finally, we apply the projection π and we erase the annotations from $\beta(Q)$.

Notice that neither P nor Q is required to be univocal (and in particular, they are not required to be terminating).

4. UNFOLD/FOLD TRANSFORMATION RULES WITH ANNOTATIONS

Step 2 of the methodology presented at the end of the previous section, whereby we derive the annotated program $\beta(Q)$ from the annotated program $\alpha(P)$, may be realized by a sequence of applications of transformation rules. These rules, which we will present below, transform annotated programs rather than programs, and are variants of the usual unfolding, folding, and goal replacement rules for definite logic programs. Given the program P and the annotation α, by n applications of these rules we construct a sequence $\alpha_0(P_0), \ldots, \alpha_n(P_n)$ of annotated programs such that $\alpha_0 = \alpha$, $P_0 = P$, $\alpha_n = \beta$, and $P_n = Q$. We assume that α_0 is an enhancing, well-founded annotation. Moreover, we will show that the applicability conditions of the transformation rules ensure that if α_0 is a well-founded annotation, then α_n is a well-founded annotation. The total correctness of the transformation of the program P_0 into program P_n follows from Theorem 6 because, for $k = 0, \ldots, n-1$, we have that: (i) $M(P_0) \models P_k \Rightarrow P_{k+1}$ and (ii) $M(\alpha_0(P_0)) \models \alpha_k(P_k) \Leftarrow \alpha_{k+1}(P_{k+1})$. Thus, by transitivity of $\Rightarrow$ and $\Leftarrow$, the sequence of applications of the unfolding, folding, and goal replacement rules can be viewed as a single clause replacement $\alpha_0(P_0) \mapsto \alpha_n(P_n)$ such that: (i) $M(P_0) \models P_0 \Rightarrow P_n$, and (ii) $M(\alpha_0(P_0)) \models \alpha_0(P_0) \Leftarrow \alpha_n(P_n)$.

The rules presented in this section are parametric w.r.t. the annotation α_0 chosen for the initial program P_0.

An *annotated transformation sequence* $\overline{P}_0, \ldots, \overline{P}_n$ is a sequence of annotated programs constructed as follows. Suppose that we have constructed the transformation sequence $\overline{P}_0, \ldots, \overline{P}_k$. Then, for $0 \leq k \leq n-1$, program $\overline{P}_{k+1}$ is derived from program $\overline{P}_k$ by the application of one of the three transformation rules R1, R2, and R3 defined below. Notice that among the transformation rules here we do not include the *definition introduction* rule [18]. This rule is useful in practice, but its absence is not a limitation when we study the correctness of program transformations. Indeed, we may assume that the definitions of the predicates which are needed during a transformation sequence, are introduced at the beginning of its construction and, thus, they can be considered to be already present in the initial program $\overline{P}_0$. This simplifying assumption is also made in [8, 16, 19].

R1. Unfolding. Let $\gamma : \overline{H} \leftarrow c \wedge \overline{G}_L \wedge \overline{A} \wedge \overline{G}_R$ be a clause in the annotated program $\overline{P}_k$ and let $\overline{P}'_0$ be a variant of $\overline{P}_0$ without common variables with $\overline{P}_k$. Let

$$\begin{array}{ll} \gamma_1: & \overline{H}_1 \leftarrow c_1 \wedge \overline{G}_1 \\ & \ldots \\ \gamma_m: & \overline{H}_m \leftarrow c_m \wedge \overline{G}_m \end{array}$$

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* γ *w.r.t. the atom* $\overline{A}$ we derive the clauses

$$\begin{array}{ll} \eta_1: & (\overline{H} \leftarrow c \wedge c_1 \wedge \overline{G}_L \wedge \overline{G}_1 \wedge \overline{G}_R)\vartheta_1 \\ & \ldots \\ \eta_m: & (\overline{H} \leftarrow c \wedge c_m \wedge \overline{G}_L \wedge \overline{G}_m \wedge \overline{G}_R)\vartheta_m \end{array}$$

and from program $\overline{P}_k$ we derive the program $\overline{P}_{k+1} = (\overline{P}_k - \{\gamma\}) \cup \{\eta_1, \ldots, \eta_m\}$.

Basically, the unfolding rule for annotated programs is like the usual unfolding rule for definite logic programs. Notice, however, that we cannot unfold an annotated program w.r.t. an annotation formula, but only w.r.t. an annotated atom. In the following rules the set of variables occurring in an expression e is denoted by $vars(e)$.

R2. Folding. Let $\overline{P}'_0$ be a variant of $\overline{P}_0$ without common variables with $\overline{P}_k$. Let

$$\begin{array}{ll} \delta_1: & \overline{K} \leftarrow d_1 \wedge \overline{G}_1 \\ & \ldots \\ \delta_m: & \overline{K} \leftarrow d_m \wedge \overline{G}_m \end{array}$$

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

$$\begin{array}{ll} \gamma_1: & \overline{H} \leftarrow c_1 \wedge \overline{G}_L \wedge \overline{G}_1\vartheta \wedge \overline{G}_R \\ & \ldots \\ \gamma_m: & \overline{H} \leftarrow c_m \wedge \overline{G}_L \wedge \overline{G}_m\vartheta \wedge \overline{G}_R \end{array}$$

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

1. no clause in $\overline{P}'_0 - \{\delta_1, \ldots, \delta_m\}$ has its head unifiable with $\overline{K}\vartheta$;
2. there exists an annotation formula c such that, for $i = 1, \ldots, m$, we have: $\mathcal{W} \models \forall X\, ((\exists Y\, c_i) \rightarrow \exists Z\, (c \wedge d_i\vartheta))$, where $X = vars(\{\overline{H}, \overline{G}_L, \overline{G}_i\vartheta, \overline{G}_R\})$, $Y = vars(c_i) - X$, and $Z = vars(c \wedge d_i\vartheta) - X$;
3. for $i = 1, \ldots, m$ and for every variable U in the set $vars(d_i \wedge \overline{G}_i) - vars(\overline{K})$: (i) $U\vartheta$ is a variable not occurring in $\{\overline{H}, c, \overline{G}_L, \overline{G}_R\}$, and (ii) $U\vartheta$ does not occur in the term $V\vartheta$, for any variable V occurring in $d_i \wedge \overline{G}_i$ and different from U; and
4. $\mathcal{W} \models \forall\, (c \rightarrow (\overline{H} \succ \overline{G}_L \wedge \overline{K}\vartheta \wedge \overline{G}_R))$.

By *folding clauses* $\gamma_1, \ldots, \gamma_m$ *using clauses* $\delta_1, \ldots, \delta_m$ we derive the clause η: $\overline{H} \leftarrow c \wedge \overline{G}_L \wedge \overline{K}\vartheta \wedge \overline{G}_R$ and from program $\overline{P}_k$ we derive the program $\overline{P}_{k+1} = (\overline{P}_k - \{\gamma_1, \ldots, \gamma_m\}) \cup \{\eta\}$.

The difference between the folding rule for annotated programs and the usual folding rule for definite logic programs consists in the extra Conditions 2 and 4. However, as already mentioned, the usual folding rule ensures partial correctness only. The following example illustrates an application of the folding rule R2.

Example 6. Let us consider the following annotated program $\overline{P}_0$, where we use the well-founded annotation α_1 over $\mathcal{N}$ of Example 4:

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. $p(A)\langle X_1\rangle$ and by renaming variables, we get:

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

By two applications of the unfolding rule and by renaming variables, from clause 7 we derive:

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

Now, Conditions 1–4 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) c to be the annotation formula $Y > Z \wedge Z > Y_2 \wedge Y > Y_1$. By folding clauses 9 and 11 using clauses 6 and 7 we derive the clause:

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

In order to introduce the goal replacement rule, we need the following definition of replacement law.

Definition 5. (Replacement Law) Let P be a program, α be a program annotation, $\overline{G}_1$ and $\overline{G}_2$ be annotated conjunctions, $X \subseteq vars(\{\overline{G}_1, \overline{G}_2\})$ be a set of variables, and d be an annotation formula. We say that the *replacement law* $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ holds in $\alpha(P)$ iff the following conditions hold:

(i) $M(\alpha(P)) \models \forall X\, (\exists Y\, (c_1 \wedge \overline{G}_1) \rightarrow \exists Z\, (c_2 \wedge \overline{G}_2))$ and

(ii) $M(P) \models \forall X\, (\exists Y\, G_1 \leftarrow \exists Z\, G_2)$

where (1) $Y = vars(c_1 \wedge \overline{G}_1) - X$, (2) $Z = vars(c_2 \wedge \overline{G}_2) - X$, and (3) G_1 and G_2 are the conjunctions obtained by erasing the annotation terms from $\overline{G}_1$ and $\overline{G}_2$, respectively.

R3. Goal Replacement. Let $\gamma : \overline{H} \leftarrow c \wedge c_1 \wedge \overline{G}_L \wedge \overline{G}_1 \wedge \overline{G}_R$ be a clause of the annotated program $\overline{P}_k$ and let $\overline{G}_2$ be an annotated conjunction such that the following replacement law $c_1 \wedge \overline{G}_1 \Rightarrow_X c_2 \wedge \overline{G}_2$ holds in P_0, where $X = \mathit{vars}(\{\overline{H}, c, \overline{G}_L, \overline{G}_R\}) \cap \mathit{vars}(\{c_1, \overline{G}_1, c_2, \overline{G}_2\})$. Suppose also that:

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

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

The goal replacement rule R3 for annotated programs differs from the usual (partially correct) goal replacement rule for definite logic programs because of Condition (†). We will see an example of application of the goal replacement rule in the next section.

By using the results of Section 3 we can prove the total correctness of the transformation rules.

THEOREM 7. (Total Correctness of the Transformation Rules) *Let $\alpha_0(P_0), \ldots, \alpha_n(P_n)$ be an annotated transformation sequence such that α_0 is an enhancing, well-founded annotation. Then $M(P_0) = M(P_n)$.*

PROOF. (Sketch) For $k = 0, \ldots, n-1$, let $\alpha_{k+1}(P_{k+1})$ be the annotated program derived from $\alpha_k(P_k)$ by the application of a transformation rule in R1–R3. Then, for some sets of annotated clauses $\alpha_k(\Gamma_k)$ and $\alpha_{k+1}(\Gamma_{k+1})$, we have that $\alpha_{k+1}(P_{k+1}) = (\alpha_k(P_k) - \alpha_k(\Gamma_k)) \cup \alpha_{k+1}(\Gamma_{k+1})$. We have the following properties: (P1) $M(P_0) \models \Gamma_k \Rightarrow \Gamma_{k+1}$, (P2) $M(\alpha_0(P_0)) \models \alpha_k(\Gamma_k) \Leftarrow \alpha_{k+1}(\Gamma_{k+1})$, and (P3) if α_k is well-founded then α_{k+1} is well-founded. The proofs of properties P1–P3 are straightforward applications of the definitions, and they are left to the reader. Notice that in the proof of P3 we use Condition 4 of the folding rule and Condition (†) of the goal replacement rule.
Now let us consider the clause replacement $P_0 \mapsto P_n$. The following properties hold.

(i) $P_0 \mapsto P_n$ is an implication-based clause replacement. Indeed, by Property P1, Lemma 1, and transitivity of $\Rightarrow$ we have: $M(P_0) \models P_0 \Rightarrow P_n$.

(ii) $\alpha_0(P_0) \mapsto \alpha_n(P_n)$ is a reverse-implication-based clause replacement. Indeed, by Property P2, Lemma 1, and transitivity of $\Leftarrow$ we have: $M(\alpha_0(P_0)) \models \alpha_0(P_0) \Leftarrow \alpha_n(P_n)$.

(iii) α_0 is enhancing (by hypothesis).

(iv) α_n is well-founded (by the hypothesis that α_0 is well-founded and Property P3).

Thus, by Theorem 6 $P_0 \mapsto P_n$ is totally correct, that is, $M(P_0) = M(P_n)$. □

5. AN EXTENDED EXAMPLE

In this section we revisit an example of program transformation taken from [16]. In that paper the proof of total correctness of the transformation rules is rather intricate. On the contrary we show that the total correctness of this transformation can be established by our well-founded annotation method in a very easy way. Let us consider the following program P:

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

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

The formula $\forall X\,(\mathit{thm}(X) \leftrightarrow \mathit{gen}(X))$ is true in the least Herbrand model of program P. As indicated in [16], the truth of this formula can be established by constructing a totally correct transformation of P into a program Q where the predicates thm and gen are defined by two sets of clauses which are identical up to a predicate renaming. Let us see how we construct this transformation by applying our rules of Section 4.

Let $\mathcal{N}$ be the well-founded ordering $(\mathit{Nat}, >)$, where Nat is the set of the natural numbers and $>$ is the usual 'greater than' ordering on Nat. Let us consider the well-founded annotation α that associates with every clause $H \leftarrow A_1 \wedge \ldots \wedge A_k$ the annotated clause:

$$H\langle N\rangle \leftarrow N > N_1 + \ldots + N_k \wedge A_1\langle N_1\rangle \wedge \ldots \wedge A_k\langle N_k\rangle$$

where the annotation variables $N, N_1, \ldots, N_k$ range over natural numbers. Thus, the annotated program $\alpha(P)$ is the following one:

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

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

The reader may verify that the replacement law

$$\mathit{test}([1|Y])\langle N_3\rangle \Rightarrow_{\{Y,N3\}} (N_3 \geq N_4 \wedge \mathit{test}(Y)\langle N_4\rangle)$$

holds in $\alpha(P)$. Indeed, we have that:

(i) $M(\alpha(P)) \models \forall Y \forall N_3\,(\mathit{test}([1|Y])\langle N_3\rangle \rightarrow \exists N_4 (N_3 \geq N_4 \wedge \mathit{test}(Y)\langle N_4\rangle))$, and

(ii) $M(P) \models \forall Y\,(\mathit{test}([1|Y]) \leftarrow \mathit{test}(Y))$.

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. $thm([0|X])\langle N\rangle \leftarrow N > N_6 + 3 \wedge thm(X)\langle N_6\rangle$

The final annotated program is $(\alpha(P) - \{1a\}) \cup \{10a, 15a\}$. 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 7 of Section 4 the transformation of P into Q is totally correct. In Q the predicates *thm* and *gen* are defined by sets of clauses which are equal up to predicate renaming (namely, clauses 10, 15 and clauses 2, 3, respectively) 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.

6. RELATED WORK AND CONCLUSIONS

We have studied the correctness of a general transformation rule, called clause replacement, which is an adaptation to the case of definite logic programs of the *rule replacement* transformation for *inductive definitions* introduced in [15]. Clause replacement generalizes the familiar unfolding, folding, and goal replacement transformations of definite logic programs. The clause replacement rule generalizes also the *simultaneous replacement operation* (when restricted to definite programs), which simultaneously replaces $n(>0)$ conjunctions of literals each of which occurs in the body of a clause [4]. Moreover, clause replacement strictly generalizes simultaneous replacement, because the unfolding rule is not an instance of simultaneous replacement. We have shown that, in fact, clause replacement is the most general transformation rule, in the sense that every correct transformation can be expressed as an equivalence-based clause replacement (see Theorem 1 of Section 2).

The main contribution of this paper is 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 introduce well-founded annotations, which ensure that the annotated program is terminating and, thus, it has a unique fixpoint [2]. Annotated logic programs are a generalization of the *instrumented* SOS rules introduced in [17], because SOS rules [14] can be considered as particular logic programs.

Our proof method uses well-founded annotations and the unique fixpoint method [6, 15] to prove the total correctness of clause replacement. 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, in practice the unique fixpoint method requires the proof of the termination of Q, while according to Theorem 6, we need only to construct a well-founded annotation β for Q so that $\beta(Q)$ is terminating, but Q itself need not be terminating.

Our proof method is also more general than the *improvement induction* method [17] in the sense that our method allows us to prove the total correctness of clause replacements which are *not* improvements, as we now see in the particular case where clause replacement is realized by the goal replacement rule R3 (see Section 4). By adapting the definitions of [17] to our context, here we say that, given a program P, an annotated atom $A_1\langle X_1\rangle$ is *improved* by an annotated atom $A_2\langle X_2\rangle$ iff for every ground instance $a_1\langle w_1\rangle$ of $A_1\langle X_1\rangle$ belonging to $M(P)$, there exists a ground instance $a_2\langle w_2\rangle$ of $A_2\langle X_2\rangle$ in $M(P)$ such that $w_1 \geq w_2$. The reader may verify that the goal replacement rule R3 allows us to replace an annotated atom $A_1\langle X_1\rangle$ by a new annotated atom $A_2\langle X_2\rangle$ even if $A_1\langle X_1\rangle$ is not improved by $A_2\langle X_2\rangle$.

We would like to stress that our unfolding, folding, and goal replacement rules presented in Section 4 are parametric w.r.t. the choice of suitable program annotations. Indeed, these program annotations are specified only by the properties they should fulfill. By suitable choices of the annotations we obtain transformation rules which are equivalent to the different variants of the unfolding, folding, and goal replacement rules proposed in the literature [8, 10, 16, 18, 19]. For instance, the reader may verify that the rules presented in [10] are a particular case of our rules where we choose the annotation α_2 of Example 4. Here we do not show in detail how other existing transformation rules can be viewed as instances of our rules of Section 4.

It should also be noticed that the use of our general proof method based on well-founded annotations (see Theorem 6) greatly simplifies the proofs of total correctness of the transformation rules w.r.t. those presented in [8, 10, 16, 18, 19].

Finally, we would like to notice that the notion of total correctness considered in this paper is different from the one for imperative programs, where a program is said to be *totally correct w.r.t. a given specification* iff its input-output relation satisfies the specification and, moreover, the program terminates (see, for instance, [12]). In fact, as already mentioned, a clause replacement $P \mapsto Q$ can be totally correct even if Q is not terminating. However, in order to prove that $P \mapsto Q$ is totally correct we have to transform an annotated program $\alpha(P)$ into a *terminating* annotated program $\beta(Q)$. In this sense we may say that program $\beta(Q)$ is totally correct w.r.t. the specification given by program $\alpha(P)$. Similarly to the proofs of total correctness for imperative programs based on the axiomatic approach [12], also the derivation of the terminating program $\beta(Q)$ is performed by applying first order logical inferences and proving suitable well-founded ordering relations.

Acknowledgments

We thank the anonymous referees for useful comments and suggestions.

7. REFERENCES

[1] K. R. Apt. Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–576. Elsevier, 1990.

[2] M. Bezem. Characterizing termination of logic programs with level mappings. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming, Cleveland, Ohio (USA)*, pages 69–80. MIT Press, 1989.

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

[4] A. Bossi, N. Cocco, and S. Etalle. Simultaneous replacement in normal programs. *Journal of Logic and Computation*, 6(1):79–120, 1996.

[5] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.

[6] B. Courcelle. Infinite trees in normal form and recursive equations having a unique solution. *Mathematical Systems Theory*, 13:131–180, 1979.

[7] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1-2):69–116, 1987.

[8] M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. In M. Hermenegildo and J. Penjam, editors, *Proceedings Sixth International Symposium on Programming Language Implementation and Logic Programming (PLILP '94)*, Lecture Notes in Computer Science 844, pages 340–354. Springer-Verlag, 1994.

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

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

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

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

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

[14] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, Aarhus, Denmark, 1981.

[15] M. Proietti and A. Pettorossi. Transforming inductive definitions. In D. De Schreye, editor, *Proceedings of the 1999 International Conference on Logic Programming*, pages 486–499. MIT Press, 1999.

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

[17] D. Sands. From SOS rules to proof principles: An operational metatheory for functional languages. In *Proceedings of POPL '97*, pages 428–441. ACM Press, 1997.

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

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