Automatic Correctness Proofs for Logic Program Transformations*

Alberto Pettorossi[1], Maurizio Proietti[2], and Valerio Senni[1]

(1) DISP, University of Roma Tor Vergata, Via del Politecnico 1, I-00133 Roma, Italy
{pettorossi,senni}@disp.uniroma2.it
(2) IASI-CNR, Viale Manzoni 30, I-00185 Roma, Italy
proietti@iasi.rm.cnr.it

Abstract. The many approaches which have been proposed in the literature for proving the correctness of unfold/fold program transformations, consist in associating suitable well-founded orderings with the proof trees of the atoms belonging to the least Herbrand models of the programs. In practice, these orderings are given by 'clause measures', that is, measures associated with the clauses of the programs to be transformed. In the unfold/fold transformation systems proposed so far, clause measures are fixed in advance, independently of the transformations to be proved correct. In this paper we propose a method for the automatic generation of the clause measures which, instead, takes into account the particular program transformation at hand. During the transformation process we construct a system of linear equations and inequations whose unknowns are the clause measures to be found, and the correctness of the transformation is guaranteed by the satisfiability of that system. Through some examples we show that our method is able to establish in a fully automatic way the correctness of program transformations which, by using other methods, are proved correct at the expense of fixing sophisticated clause measures.

1 Introduction

Rule-based program transformation is a program development methodology which consists in deriving from an initial program a final program, via the application of semantics preserving transformation rules [5]. In the field of logic (or functional) programming, program transformation can be regarded as a deductive process. Indeed, programs are logical (or equational, resp.) theories and the transformation rules can be viewed as rules for deducing new formulas from old ones. The logical soundness of the transformation rules easily implies that a transformation is *partially correct*, which means that an atom (or an equation, resp.) is true in the final program only if it is true in the initial program. However, it is usually much harder to prove that a transformation is *totally correct*,

* Revised version of the paper published in: Proceedings of the 23rd International Conference on Logic Programming (ICLP'07), Porto, Portugal, 8-13 September, 2007.

which means that an atom (or an equation, resp.) is true in the initial program if and only if it is true in the final program.

In the context of functional programming, it has been pointed out in the seminal paper by Burstall and Darlington [5] that, if the transformation rules rewrite the equations of the program at hand by using equations which belong to the same program (like the *folding* and *unfolding* rules), the transformations are always partially correct, but the final program may terminate (w.r.t. a suitable notion of termination) less often than the initial one. Thus, a sufficient condition for total correctness is that the final program obtained by transformation always terminates. This method of proving total correctness is sometimes referred to as *McCarthy's method* [13]. However, the termination condition may be, in practice, very hard to check.

The situation is similar in the case of definite logic programs, where the folding and unfolding rules basically consist in applying equivalences that hold in the least Herbrand model of the initial program. For instance, let us consider the program:

$P\colon \quad p \leftarrow q \qquad\qquad r \leftarrow q \qquad\qquad q \leftarrow$

The least Herbrand model of P is $M(P) = \{p, q, r\}$ and $M(P) \models p \leftrightarrow q$. If we replace q by p in $r \leftarrow q$ (that is, we fold $r \leftarrow q$ using $p \leftarrow q$), then we get:

$Q\colon \quad p \leftarrow q \qquad\qquad r \leftarrow p \qquad\qquad q \leftarrow$

The transformation of P into Q is totally correct, because $M(P) = M(Q)$. However, if we replace q by p in $p \leftarrow q$ (that is, we fold $p \leftarrow q$ using $p \leftarrow q$ itself), then we get:

$R\colon \quad p \leftarrow p \qquad\qquad r \leftarrow q \qquad\qquad q \leftarrow$

and the transformation of P into R is partially correct, because $M(P) \supseteq M(R)$, but it is *not* totally correct, because $M(P) \neq M(R)$. Indeed, program R does not terminate for the goal p.

A lot of work has been devoted to devise methods for proving the total correctness of transformations based on various sets of rules, including the folding and the unfolding rules. These methods have been proposed both in the context of functional programming (see, for instance, [5, 10, 17]) and in the context of logic programming (see, for instance, [3, 4, 6–9, 11, 14–16, 19–21]).

Some of these methods (such as, [3, 5, 6, 11]) propose sufficient conditions for total correctness which are explicitly based on the preservation of suitable termination properties (such as, termination of call-by-name reduction for functional programs, and universal or existential termination for logic programs).

Other methods, which we may call *implicit methods*, are based on conditions on the sequence of applications of the transformation rules that guarantee that termination is preserved. A notable example of these implicit methods is presented in [9], where integer counters are associated with program clauses. The counters of the initial program are set to 1 and are incremented (or decremented) when an unfolding (or folding, resp.) is applied. A sequence of transformations is totally correct if the counters of the clauses of the final program are all positive.

The method based on counters allows us to prove the total correctness of many transformations. Unfortunately, there are also many simple derivations where the method fails to guarantee the total correctness. For instance, in the transformation from P to Q described above, we would get a value of 0 for the counter of the clause $r \leftarrow p$ in the final program Q, because it has been derived by applying the folding rule from clause $r \leftarrow p$. Thus, the method does not yield the total correctness of the transformation. In order to overcome the limitations of the basic counter method, some modifications and enhancements have been described in [9, 15, 16, 21], where each clause is given a *measure* which is more complex than an integer counter.

In this paper we present a different approach to the improvement of the basic counter method: instead of fixing *in advance* complex clause measures, for any given transformation we automatically generate, if at all possible, the clause measures that prove its correctness. For reasons of simplicity we assume that clause measures are non-negative integers, also called *weights*, and given a transformation starting from a program P, we look for a weight assignment to the clauses of P that proves that the transformation is totally correct.

Our paper is structured as follows. In Section 2 we present the notion of a *weighted transformation sequence*, that is, a sequence of programs constructed by applying suitable variants of the definition introduction, unfolding, and folding rules. We associate the clauses of the initial program of the sequence with some unknown weights, and during the construction of the sequence, we generate a set of constraints consisting of linear equations and inequations which relate those weights. If the final set of constraints is satisfiable for some assignment to the unknown weights, then the transformation sequence is totally correct. In Section 3 we prove our total correctness result which is based on the *well-founded annotations* method proposed in [14]. In Section 4 we consider transformation sequences constructed by using also the goal replacement rule and we present a method for proving the total correctness of those transformation sequences. Finally, in Section 5 we present a method for proving predicate properties which are needed for applying the goal replacement rule.

2 Weighted Unfold/Fold Transformation Rules

Let us begin by introducing some terminology concerning systems of linear equations and inequations with integer coefficients and non-negative integer solutions.

By $\mathcal{P}_{LIN}$ we denote the set of linear polynomials with integer coefficients. Variables occurring in polynomials are called *unknowns* to distinguish them from logical variables occurring in programs. By $\mathcal{C}_{LIN}$ we denote the set of linear equations and inequations with integer coefficients, that is, $\mathcal{C}_{LIN}$ is the set $\{p_1 = p_2,\ p_1 < p_2,\ p_1 \leq p_2 \mid p_1, p_2 \in \mathcal{P}_{LIN}\}$. By $p_1 \geq p_2$ we mean $p_2 \leq p_1$, and by $p_1 > p_2$ we mean $p_2 < p_1$. An element of $\mathcal{C}_{LIN}$ is called a *constraint*. A *valuation* for a set $\{u_1, \ldots, u_r\}$ of unknowns is a mapping $\sigma \colon \{u_1, \ldots, u_r\} \rightarrow \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. Let $\{u_1, \ldots, u_r\}$ be the set of unknowns occurring in $p \in \mathcal{P}_{LIN}$. Given a valuation σ for (a superset of) $\{u_1, \ldots, u_r\}$, $\sigma(p)$ is the integer obtained by replacing the occurrences of $u_1, \ldots, u_r$ in p by $\sigma(u_1), \ldots, \sigma(u_r)$, respectively, and then computing the value of the resulting arithmetic expression. A valuation

σ is a *solution* for the constraint $p_1 = p_2$ if σ is a valuation for a superset of the variables occurring in $p_1 = p_2$ and $\sigma(p_1) = \sigma(p_2)$ holds. Similarly, we define a solution for $p_1 < p_2$ and for $p_1 \leq p_2$. σ is a solution for a finite set $\mathcal{C}$ of constraints if, for every $c \in \mathcal{C}$, σ is a solution for c. We say that a constraint c is *satisfiable* if there exists a solution for c. Similarly, we say that a set $\mathcal{C}$ of constraints is *satisfiable* if there exists a solution for $\mathcal{C}$. A *weight function* for a set S of clauses is a function $\gamma : S \rightarrow \mathcal{P}_{LIN}$. A value of γ is also called a *weight polynomial*.

A *weighted unfold/fold transformation sequence* is a sequence of programs, denoted $P_0 \mapsto P_1 \mapsto \cdots \mapsto P_n$, such that $n \geq 0$ and, for $k = 0, \ldots, n-1$, P_{k+1} is derived from P_k by applying one of the following transformation rules: *weighted definition introduction*, *weighted unfolding*, and *weighted folding*. These rules, which will be defined below, are variants of the familiar rules without weights. For reasons of simplicity, when referring to the transformation rules, we will often omit the qualification 'weighted'. For $k = 0, \ldots, n$, we will define: (i) a weight function $\gamma_k : P_k \rightarrow \mathcal{P}_{LIN}$, (ii) a finite set $\mathcal{C}_k$ of constraints, (iii) a set Defs_k of clauses defining the new predicates introduced by the definition introduction rule during the construction of the sequence $P_0 \mapsto P_1 \mapsto \cdots \mapsto P_k$, and (iv) a weight function $\delta_k : P_0 \cup \mathit{Defs}_k \rightarrow \mathcal{P}_{LIN}$. The weight function γ_0 for the initial program P_0 is defined as follows: for every clause $C \in P_0$, $\gamma_0(C) = u$, where u is an unknown and, for each pair C and D of distinct clauses in P_0, we have that $\gamma_0(C) \neq \gamma_0(D)$. The initial sets $\mathcal{C}_0$ and Defs_0 are, by definition, equal to the empty set and $\delta_0 = \gamma_0$.

For every $k > 0$, we assume that P_0 and P_k have no variables in common. This assumption is not restrictive because we can always rename the variables occurring in a program without affecting its least Herbrand model. Indeed, in the sequel we will feel free to rename variables, whenever needed.

Rule 1 (Weighted Definition Introduction) Let $D_1, \ldots, D_m$, with $m > 0$, be clauses such that, for $i = 1, \ldots, m$, the predicate of the head of D_i does not occur in $P_0 \cup \mathit{Defs}_k$. By *definition introduction* from P_k we derive $P_{k+1} = P_k \cup \{D_1, \ldots, D_m\}$.
We set the following: (1.1) for all C in P_k, $\gamma_{k+1}(C) = \gamma_k(C)$, (1.2) for $i = 1, \ldots, m$, $\gamma_{k+1}(D_i) = u_i$, where u_i is a fresh new unknown, (2) $\mathcal{C}_{k+1} = \mathcal{C}_k$, (3) $\mathit{Defs}_{k+1} = \mathit{Defs}_k \cup \{D_1, \ldots, D_m\}$, (4.1) for all D in $P_0 \cup \mathit{Defs}_k$, $\delta_{k+1}(D) = \delta_k(D)$, and (4.2) for $i = 1, \ldots, m$, $\delta_{k+1}(D_i) = u_i$.

Rule 2 (Weighted Unfolding) Let C: $H \leftarrow G_L \wedge A \wedge G_R$ be a clause in P_k and let C_1: $H_1 \leftarrow G_1$, ..., C_m: $H_m \leftarrow G_m$, with $m \geq 0$, be *all* clauses in P_0 such that, for $i = 1, \ldots, m$, A is unifiable with H_i via a most general unifier ϑ_i. By *unfolding* C w.r.t. A *using* $C_1, \ldots, C_m$, we derive the clauses D_1: $(H \leftarrow G_L \wedge G_1 \wedge G_R)\vartheta_1$, ..., D_m: $(H \leftarrow G_L \wedge G_m \wedge G_R)\vartheta_m$, and from P_k we derive $P_{k+1} = (P_k - \{C\}) \cup \{D_1, \ldots, D_m\}$.
We set the following: (1.1) for all D in $P_k - \{C\}$, $\gamma_{k+1}(D) = \gamma_k(D)$, (1.2) for $i = 1, \ldots, m$, $\gamma_{k+1}(D_i) = \gamma_k(C) + \gamma_0(C_i)$, (2) $\mathcal{C}_{k+1} = \mathcal{C}_k$, (3) $\mathit{Defs}_{k+1} = \mathit{Defs}_k$, and (4) $\delta_{k+1} = \delta_k$.

For a goal (or set of goals) G, by $\mathit{vars}(G)$ we denote the set of variables occurring in G.

Rule 3 (Weighted Folding) Let C_1: $H \leftarrow G_L \wedge G_1 \wedge G_R$, ..., C_m: $H \leftarrow G_L \wedge G_m \wedge G_R$ be clauses in P_k and let D_1: $K \leftarrow B_1$, ..., D_m: $K \leftarrow B_m$ be clauses in $P_0 \cup \mathit{Defs}_k$. Suppose that there exists a substitution ϑ such that the following conditions hold: (i) for $i = 1, \ldots, m$, $G_i = B_i\vartheta$, (ii) there exists no clause in $(P_0 \cup \mathit{Defs}_k) - \{D_1, \ldots, D_m\}$ whose head is unifiable with $K\vartheta$, and (iii) for $i = 1, \ldots, m$ and for every variable U in $\mathit{vars}(B_i) - \mathit{vars}(K)$: (iii.1) $U\vartheta$ is a variable not occurring in $\{H, G_L, G_R\}$, and (iii.2) $U\vartheta$ does not occur in the term $V\vartheta$, for any variable V occurring in B_i and different from U.
By *folding* $C_1, \ldots, C_m$ *using* $D_1, \ldots, D_m$, we derive E: $H \leftarrow G_L \wedge K\vartheta \wedge G_R$, and from P_k we derive $P_{k+1} = (P_k - \{C_1, \ldots, C_m\}) \cup \{E\}$.
We set the following: (1.1) for all C in $P_k - \{C_1, \ldots, C_m\}$, $\gamma_{k+1}(C) = \gamma_k(C)$, (1.2) $\gamma_{k+1}(E) = u$, where u is a new unknown, (2) $\mathcal{C}_{k+1} = \mathcal{C}_k \cup \{u \leq \gamma_k(C_1) - \delta_k(D_1), \ldots, u \leq \gamma_k(C_m) - \delta_k(D_m)\}$, (3) $\mathit{Defs}_{k+1} = \mathit{Defs}_k$, and (4) $\delta_{k+1} = \delta_k$.

The *correctness constraint system* associated with a weighted unfold/fold transformation sequence $P_0 \mapsto \cdots \mapsto P_n$ is the set $\mathcal{C}_{\mathit{final}}$ of constraints defined as follows:

$$\mathcal{C}_{\mathit{final}} = \mathcal{C}_n \cup \{\gamma_n(C) \geq 1 \mid C \in P_n\}.$$

The following result, which will be proved in Section 3, guarantees the total correctness of weighted unfold/fold transformations. By $M(P)$ we denote the least Herbrand model of program P.

Theorem 1 (Total Correctness of Weighted Unfold/Fold Transformations). *Let $P_0 \mapsto \cdots \mapsto P_n$ be a weighted unfold/fold transformation sequence constructed by using Rules 1–3, and let $\mathcal{C}_{\mathit{final}}$ be its associated correctness constraint system. If $\mathcal{C}_{\mathit{final}}$ is satisfiable then $M(P_0 \cup \mathit{Defs}_n) = M(P_n)$.*

Example 1. (*Continuation Passing Style Transformation*) Let us consider the initial program P_0 consisting of the following three clauses whose weight polynomials are the unknowns u_1, u_2, and u_3, respectively (we write weight polynomials on a second column to the right of the corresponding clause):

1.	$p \leftarrow$	u_1
2.	$p \leftarrow p \wedge q$	u_2
3.	$q \leftarrow$	u_3

We want to derive a continuation-passing-style program defining a predicate p_{cont} equivalent to the predicate p defined by the program P_0. In order to do so, we introduce by Rule 1 the following clause 4 with its unknown u_4:

4.	$p_{cont} \leftarrow p$	u_4

and also the following three clauses for the unary continuation predicate *cont* with unknowns u_5, u_6, and u_7, respectively:

(∗)5.	$\mathit{cont}(f_{true}) \leftarrow$	u_5
6.	$\mathit{cont}(f_p(X)) \leftarrow p \wedge \mathit{cont}(X)$	u_6
7.	$\mathit{cont}(f_q(X)) \leftarrow q \wedge \mathit{cont}(X)$	u_7

where f_{true}, f_p, and f_q are three function symbols corresponding to the three predicates *true*, p, and q, respectively. By folding clause 4 using clause 5 we get the following clause with the unknown u_8 which should satisfy the constraint $u_8 \leq u_4 - u_5$ (we write constraints on a third column to the right of the corresponding clause):

8. $p_{cont} \leftarrow p \wedge cont(f_{true})$	u_8	$u_8 \leq u_4 - u_5$

By folding clause 8 using clause 6 we get the following clause 9 with unknown u_9 such that $u_9 \leq u_8 - u_6$:

$(*)$ 9. $p_{cont} \leftarrow cont(f_p(f_{true}))$	u_9	$u_9 \leq u_8 - u_6$

By unfolding clause 6 w.r.t. p using clauses 1 and 2, we get:

$(*)$ 10. $cont(f_p(X)) \leftarrow cont(X)$	$u_6 + u_1$
11. $cont(f_p(X)) \leftarrow p \wedge q \wedge cont(X)$	$u_6 + u_2$

Then by folding clause 11 using clause 7 we get:

12. $cont(f_p(X)) \leftarrow p \wedge cont(f_q(X))$	u_{12}	$u_{12} \leq u_6 + u_2 - u_7$

and by folding clause 12 using clause 6 we get:

$(*)$ 13. $cont(f_p(X)) \leftarrow cont(f_p(f_q(X)))$	u_{13}	$u_{13} \leq u_{12} - u_6$

Finally, by unfolding clause 7 w.r.t. q we get:

$(*)$ 14. $cont(f_q(X)) \leftarrow cont(X)$	$u_7 + u_3$

The final program is made out of clauses 5, 9, 10, 13, and 14, marked with $(*)$, and clauses 1, 2, and 3. The correctness constraint system $\mathcal{C}_{final}$ is made out of the following 11 constraints.

For clauses 5, 9, 10, 13, and 14: $u_5 \geq 1$, $u_9 \geq 1$, $u_6 + u_1 \geq 1$, $u_{13} \geq 1$, $u_7 + u_3 \geq 1$.
For clauses 1, 2, and 3: $u_1 \geq 1$, $u_2 \geq 1$, $u_3 \geq 1$.
For the four folding steps: $u_8 \leq u_4 - u_5$, $u_9 \leq u_8 - u_6$, $u_{12} \leq u_6 + u_2 - u_7$, $u_{13} \leq u_{12} - u_6$.
This system $\mathcal{C}_{final}$ of constraints is satisfiable and thus, the transformation from program P_0 to the final program is totally correct.

3 Proving Correctness Via Weighted Programs

In order to prove that a weighted unfold/fold transformation sequence $P_0 \mapsto \cdots \mapsto P_n$ is *totally correct* (see Theorem 1), we specialize the method based on *well-founded annotations* proposed in [14]. In particular, with each program P_k in the transformation sequence, we associate a *weighted program* $\overline{P}_k$ by adding an integer argument $n (\geq 0)$, called a *weight*, to each atom $p(\boldsymbol{t})$ occurring in P_k. Here and in the sequel, $\boldsymbol{t}$ denotes a generic m-tuple of terms $t_1, \ldots, t_m$, for some $m \geq 0$. Informally, $p(\boldsymbol{t}, n)$ holds in $\overline{P}_k$ if $p(\boldsymbol{t})$ 'has a proof of weight at least n' in P_k. We will show that if the correctness constraint system $\mathcal{C}_{final}$ is satisfiable, then it is possible to derive from $\overline{P}_0$ a weighted program $\overline{P}_n$ where the weight arguments determine, for every clause $\overline{C}$ in $\overline{P}_n$, a well-founded ordering between the head of $\overline{C}$ and every atom in the body $\overline{C}$. Hence $\overline{P}_n$ terminates for all ground goals (even if P_n need not) and the immediate consequence operator $T_{\overline{P}}$ has a unique fixpoint [2]. Thus, as proved in [14], the total correctness of the transformation sequence follows from the *unique fixpoint principle* (see Corollary 1).

Our transformation rules can be regarded as rules for replacing a set of clauses by an equivalent one. Let us introduce the notions of implication and equivalence between sets of clauses according to [14].

Definition 1. Let I be an Herbrand interpretation and let Γ_1 and Γ_2 be two sets of clauses. We write $I \models \Gamma_1 \Rightarrow \Gamma_2$ if 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 $I \models \Gamma_1 \Leftarrow \Gamma_2$ if $I \models \Gamma_2 \Rightarrow \Gamma_1$, and we write $I \models \Gamma_1 \Leftrightarrow \Gamma_2$ if ($I \models \Gamma_1 \Rightarrow \Gamma_2$ and $I \models \Gamma_1 \Leftarrow \Gamma_2$).

For all Herbrand interpretations 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 $I \models \Gamma_1 \cup \Gamma_3 \Rightarrow \Gamma_2 \cup \Gamma_3$.

Given a program P, we denote its associated *immediate consequence operator* by T_P [1, 12]. We denote the least and greatest fixpoint of T_P by $lfp(T_P)$ and $gfp(T_P)$, respectively. Recall that $M(P) = lfp(T_P)$.

Now let us consider the transformation of a program P into a program Q consisting in the replacement of a set Γ_1 of clauses in P by a new set Γ_2 of clauses. The following result, proved in [14], expresses the *partial correctness* of the transformation of P into Q.

Theorem 2 (Partial Correctness). *Given two programs P and Q, such that: (i) for some sets Γ_1 and Γ_2 of clauses, $Q = (P - \Gamma_1) \cup \Gamma_2$, and (ii) $M(P) \models \Gamma_1 \Rightarrow \Gamma_2$. Then $M(P) \supseteq M(Q)$.*

In order to establish a sufficient condition for the total correctness of the transformation of P into Q, that is, $M(P) = M(Q)$, we consider programs whose associated immediate consequence operators have unique fixpoints.

Definition 2 (Univocal Program). A program P is said to be *univocal* if T_P has a unique fixpoint, that is, $lfp(T_P) = gfp(T_P)$.

The following theorem is proved in [14].

Theorem 3 (Conservativity). *Given two programs P and Q, such that: (i) for some sets Γ_1 and Γ_2 of clauses, $Q = (P - \Gamma_1) \cup \Gamma_2$, (ii) $M(P) \models \Gamma_1 \Leftarrow \Gamma_2$, and (iii) Q is univocal. Then $M(P) \subseteq M(Q)$.*

As a straightforward consequence of Theorems 2 and 3 we get the following.

Corollary 1 (Total Correctness Via Unique Fixpoint). *Given two programs P and Q such that: (i) for some sets Γ_1, Γ_2 of clauses, $Q = (P - \Gamma_1) \cup \Gamma_2$, (ii) $M(P) \models \Gamma_1 \Leftrightarrow \Gamma_2$, and (iii) Q is univocal. Then $M(P) = M(Q)$.*

Corollary 1 cannot be directly applied to prove the total correctness of a transformation sequence generated by applying the unfolding and folding rules, because the programs derived by these rules need not be univocal. To overcome this difficulty we introduce the notion of weighted program.

Given a clause C of the form $p_0(\boldsymbol{t_0}) \leftarrow p_1(\boldsymbol{t_1}) \wedge \ldots \wedge p_m(\boldsymbol{t_m})$, where $\boldsymbol{t_0}, \boldsymbol{t_1}, \ldots, \boldsymbol{t_m}$ are tuples of terms, a *weighted clause*, denoted $\overline{C}(w)$, associated with C is a clause of the form:

$\overline{C}(w)$: $p_0(\boldsymbol{t_0}, N_0) \leftarrow N_0 \geq N_1 + \cdots + N_m + w \wedge p_1(\boldsymbol{t_1}, N_1) \wedge \ldots \wedge p_m(\boldsymbol{t_m}, N_m)$

where w is a natural number called the *weight* of $\overline{C}(w)$. Clause $\overline{C}(w)$ is denoted by $\overline{C}$ when we do not need refer to the weight w. A *weighted program* is a set of weighted clauses. Given a program $P = \{C_1, \ldots, C_r\}$, by $\overline{P}$ we denote a weighted program of the form $\{\overline{C}_1, \ldots, \overline{C}_r\}$. Given a weight function γ and a valuation σ, by $\overline{C}(\gamma, \sigma)$ we denote the weighted clause $\overline{C}(\sigma(\gamma(C)))$ and by $\overline{P}(\gamma, \sigma)$ we denote the weighted program $\{\overline{C}_1(\gamma, \sigma), \ldots, \overline{C}_r(\gamma, \sigma)\}$.

For reasons of conciseness, we do not formally define here when a formula of the form $N_0 \geq N_1 + \cdots + N_m + w$ (see clause $\overline{C}(w)$ above) holds in an interpretation, and we simply say that for every Herbrand interpretation I and ground terms $n_0, n_1, \ldots, n_m, w$, we have that $I \models n_0 \geq n_1 + \cdots + n_m + w$ holds iff $n_0, n_1, \ldots, n_m, w$ are (terms representing) natural numbers such that n_0 is greater than or equal to $n_1 + \cdots + n_m + w$.

The following lemma (proved in [14]) establishes the relationship between the semantics of a program P and the semantics of any weighted program $\overline{P}$ associated with P.

Lemma 1. *Let P be a program. For every ground atom $p(\boldsymbol{t})$, $p(\boldsymbol{t}) \in M(P)$ iff there exists $n \in \mathbb{N}$ such that $p(\boldsymbol{t}, n) \in M(\overline{P})$.*

By erasing weights from clauses we preserve clause implications, in the sense stated by the following lemma (proved in [14]).

Lemma 2. *Let P be a program, and Γ_1 and Γ_2 be any two sets of clauses. If $M(\overline{P}) \models \overline{\Gamma}_1 \Rightarrow \overline{\Gamma}_2$ then $M(P) \models \Gamma_1 \Rightarrow \Gamma_2$.*

A weighted program $\overline{P}$ is said to be *decreasing* if every clause in $\overline{P}$ has a positive weight.

Lemma 3. *Every decreasing program is univocal.*

Now, we have the following result, which is a consequence of Lemmata 1, 3, and Theorems 2 and 3. Unlike Corollary 1, this result can be used to prove the total correctness of the transformation of program P into program Q also in the case where Q is not univocal.

Theorem 4 (Total Correctness Via Weights). *Let P and Q be programs such that: (i) $M(P) \models P \Rightarrow Q$, (ii) $M(\overline{P}) \models \overline{P} \Leftarrow \overline{Q}$, and (iii) $\overline{Q}$ is decreasing. Then $M(P) = M(Q)$.*

By Theorem 4, in order to prove Theorem 1, that is, the total correctness of weighted unfold/fold transformations, it is enough to show that, given a weighted unfold/fold transformation sequence $P_0 \mapsto \cdots \mapsto P_n$, we have that:

(P1) $M(P_0 \cup \mathit{Defs}_n) \models P_0 \cup \mathit{Defs}_n \Rightarrow P_n$

and there exist suitable weighted programs $\overline{P}_0 \cup \overline{\mathit{Defs}}_n$ and $\overline{P}_n$, associated with $P_0 \cup \mathit{Defs}_n$ and P_n, respectively, such that:

(P2) $M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models \overline{P}_0 \cup \overline{\mathit{Defs}}_n \Leftarrow \overline{P}_n$, and

(P3) $\overline{P}_n$ is decreasing.

The suitable weighted programs $\overline{P}_0 \cup \overline{\mathit{Defs}}_n$ and $\overline{P}_n$ are constructed as we now indicate by using the hypothesis that the correctness constraint system $\mathcal{C}_{\mathit{final}}$ associated with the transformation sequence, is satisfiable. Let σ be a solution for $\mathcal{C}_{\mathit{final}}$. For every $k = 0, \ldots, n$ and for every clause $C \in P_k$, we take $\overline{C} = \overline{C}(\gamma_k, \sigma)$, where γ_k is the weight function associated with P_k. For $C \in \mathit{Defs}_k$ we take $\overline{C} = \overline{C}(\delta_k, \sigma)$. Thus, $\overline{P}_k = \overline{P}_k(\gamma_k, \sigma)$ and $\overline{\mathit{Defs}}_k = \overline{\mathit{Defs}}_k(\delta_k, \sigma)$.

In order to prove Theorem 1 we need the following two lemmata.

Lemma 4. *Let $P_0 \mapsto \cdots \mapsto P_k$ be a weighted unfold/fold transformation sequence. Let C be a clause in P_k, and let $D_1, \ldots, D_m$ be the clauses derived by* unfolding *C w.r.t. an atom in its body, as described in Rule* 2. *Then*:

$M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models \{\overline{C}\} \Leftrightarrow \{\overline{D}_1, \ldots, \overline{D}_m\}$

Lemma 5. *Let $P_0 \mapsto \cdots \mapsto P_k$ be a weighted unfold/fold transformation sequence. Let $C_1, \ldots, C_m$ be clauses in P_k, $D_1, \ldots, D_m$ be clauses in $P_0 \cup \mathit{Defs}_k$, and E be the clause derived by* folding *$C_1, \ldots, C_m$ using $D_1, \ldots, D_m$, as described in Rule* 3. *Then*:

(*i*) $M(P_0 \cup \mathit{Defs}_n) \models \{C_1, \ldots, C_m\} \Rightarrow \{E\}$

(*ii*) $M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models \{\overline{C}_1, \ldots, \overline{C}_m\} \Leftarrow \{\overline{E}\}$

We are now able to prove Theorem 1. For a weighted unfold/fold transformation sequence $P_0 \mapsto \cdots \mapsto P_n$, the following properties hold:

(R1) $M(P_0 \cup \mathit{Defs}_n) \models P_k \cup (\mathit{Defs}_n - \mathit{Defs}_k) \Rightarrow P_{k+1} \cup (\mathit{Defs}_n - \mathit{Defs}_{k+1})$, and

(R2) $M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models \overline{P}_k \cup (\overline{\mathit{Defs}}_n - \overline{\mathit{Defs}}_k) \Leftarrow \overline{P}_{k+1} \cup (\overline{\mathit{Defs}}_n - \overline{\mathit{Defs}}_{k+1})$.

Indeed, Properties (R1) and (R2) can be proved by reasoning by cases on the transformation rule applied to derive P_{k+1} from P_k, as follows. If P_{k+1} is derived from P_k by applying the definition introduction rule then $P_k \cup (\mathit{Defs}_n - \mathit{Defs}_k) = P_{k+1} \cup (\mathit{Defs}_n - \mathit{Defs}_{k+1})$ and, therefore, Properties (R1) and (R2) are trivially true. If P_{k+1} is derived from P_k by applying the unfolding rule, then $P_{k+1} = (P_k - \{C\}) \cup \{D_1, \ldots, D_m\}$ and $\mathit{Defs}_k = \mathit{Defs}_{k+1}$. Hence, Properties (R1) and (R2) follow from Lemma 2, Lemma 4 and from the monotonicity of $\Rightarrow$. If P_{k+1} is derived from P_k by applying the folding rule, then $P_{k+1} = (P_k - \{C_1, \ldots, C_m\}) \cup \{E\}$ and $\mathit{Defs}_k = \mathit{Defs}_{k+1}$. Hence, Properties (R1) and (R2) follow from Points (i) and (ii) of Lemma 5 and the monotonicity of $\Rightarrow$.

By the transitivity of $\Rightarrow$ and by Properties (R1) and (R2), we get Properties (P1) and (P2). Moreover, since σ is a solution for $\mathcal{C}_{\mathit{final}}$ and $\overline{P}_n = \overline{P}_n(\gamma_n, \sigma)$, Property (P3) holds. Thus, by Theorem 4, $M(P_0 \cup \mathit{Defs}_n) = M(P_n)$.

4 Weighted Goal Replacement

In this section we extend the notion of a weighted unfold/fold transformation sequence $P_0 \mapsto P_1 \mapsto \cdots \mapsto P_n$ by assuming that P_{k+1} is derived from P_k by applying, besides the definition introduction, unfolding, and folding rules, also the goal replacement rule defined as Rule 4 below. The goal replacement rule consists in replacing a goal G_1 occurring in the body of a clause of P_k, by a new goal G_2 such that G_1 and G_2 are equivalent in $M(P_0 \cup \mathit{Defs}_k)$. Some conditions are also needed in order to update the value of the weight function and the associated constraints. To define these conditions we introduce the notion of *weighted replacement law* (see Definition 3), which in turn is based on the notion of weighted program introduced in Section 3.

In Definition 3 below we will use the following notation. Given a goal G : $p_1(\boldsymbol{t_1})\wedge\ldots\wedge p_m(\boldsymbol{t_m})$, a variable N, and a natural number w, by $\overline{G}[N, w]$ we denote the formula $\exists N_1 \ldots \exists N_m (N \geq N_1 + \cdots + N_m + w \wedge p_1(\boldsymbol{t_1}, N_1) \wedge \ldots \wedge p_m(\boldsymbol{t_m}, N_m))$. Given a set $X = \{X_1, \ldots, X_m\}$ of variables, we will use '$\exists X$' as a shorthand for '$\exists X_1 \ldots \exists X_m$' and '$\forall X$' as a shorthand for '$\forall X_1 \ldots \forall X_m$'.

Definition 3 (Weighted Replacement Law). Let P be a program, γ be a weight function for P, and $\mathcal{C}$ be a finite set of constraints. Let G_1 and G_2 be goals, u_1 and u_2 be unknowns, and $X \subseteq \mathit{vars}(G_1) \cup \mathit{vars}(G_2)$ be a set of variables. We say that the *weighted replacement law* $(G_1, u_1) \Rightarrow_X (G_2, u_2)$ holds with respect to the triple $\langle P, \gamma, \mathcal{C}\rangle$, and we write $\langle P, \gamma, \mathcal{C}\rangle \models (G_1, u_1) \Rightarrow_X (G_2, u_2)$, if the following conditions hold:

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

(ii) for every solution σ for $\mathcal{C}$,

$M(\overline{P}) \models \forall X \forall U\, (\exists Y\, (\overline{G}_1[U, \sigma(u_1)]) \rightarrow \exists Z\, (\overline{G}_2[U, \sigma(u_2)]))$

where: (1) $\overline{P}$ is the weighted program $\overline{P}(\gamma, \sigma)$, (2) U is a variable, (3) $Y = \mathit{vars}(G_1) - X$, and (4) $Z = \mathit{vars}(G_2) - X$.

By using Lemma 2 it can be shown that, if $\mathcal{C}$ is satisfiable, then Condition (ii) of Definition 3 implies $M(P) \models \forall X\, (\exists Y\, G_1 \rightarrow \exists Z\, G_2)$ and, therefore, if $\langle P, \gamma, \mathcal{C}\rangle \models (G_1, u_1) \Rightarrow_X (G_2, u_2)$ and $\mathcal{C}$ is satisfiable, we have that $M(P) \models \forall X\, (\exists Y\, G_1 \leftrightarrow \exists Z\, G_2)$.

Example 2. (*Associativity of List Concatenation*) Let us consider the following program *Append* for list concatenation. To the right of each clause we indicate the corresponding unknown.

1. $a([\,], L, L)$	u_1
2. $a([H|T], L, [H|R]) \leftarrow a(T, L, R)$	u_2

The following replacement law expresses the associativity of list concatenation:

Law (α): $(a(L_1, L_2, M) \wedge a(M, L_3, L),\ w_1)$

$$\Rightarrow_{\{L_1, L_2, L_3, L\}} \ (a(L_2, L_3, R) \wedge a(L_1, R, L),\ w_2)$$

where w_1 and w_2 are new unknowns. In Example 4 below we will show that Law (α) holds w.r.t. $\langle \mathit{Append}, \gamma, \mathcal{C}\rangle$, where $\mathcal{C}$ is the set of constraints $\{u_1 \geq 1, u_2 \geq 1, w_1 \geq w_2, w_2 + u_1 \geq 1\}$.

In Section 5 we will present a method, called *weighted unfold/fold proof method*, for generating a suitable set $\mathcal{C}$ of constraints such that $\langle P,\gamma,\mathcal{C}\rangle \models (G_1,u_1) \Rightarrow_X (G_2,u_2)$ holds.

Now we introduce the Weighted Goal Replacement Rule, which is a variant of the rule without weights (see, for instance, [20]).

Rule 4 (Weighted Goal Replacement) Let C: $H \leftarrow G_L \wedge G_1 \wedge G_R$ be a clause in program P_k and let $\mathcal{C}$ be a set of constraints such that the weighted replacement law $\lambda:\ (G_1,u_1) \Rightarrow_X (G_2,u_2)$ holds w.r.t. $\langle P_0 \cup \mathit{Defs}_k, \delta_k, \mathcal{C}\rangle$, where $X = \mathit{vars}(\{H, G_L, G_R\}) \cap \mathit{vars}(\{G_1, G_2\})$.
By applying the replacement law λ, from C we derive $D: H \leftarrow G_L \wedge G_2 \wedge G_R$, and from P_k we derive $P_{k+1} = (P_k - \{C\}) \cup \{D\}$. We set the following: (1.1) for all E in $P_k - \{C\}$, $\gamma_{k+1}(E) = \gamma_k(E)$, (1.2) $\gamma_{k+1}(D) = \gamma_k(C) - u_1 + u_2$, (2) $\mathcal{C}_{k+1} = \mathcal{C}_k \cup \mathcal{C}$, (3) $\mathit{Defs}_{k+1} = \mathit{Defs}_k$, and (4) $\delta_{k+1} = \delta_k$.

The proof of the following result is similar to the one of Theorem 1.

Theorem 5 (Total Correctness of Weighted Unfold/Fold/Replacement Transformations). *Let $P_0 \mapsto \cdots \mapsto P_n$ be a weighted unfold/fold transformation sequence constructed by using Rules 1–4, and let $\mathcal{C}_{\mathit{final}}$ be its associated correctness constraint system. If $\mathcal{C}_{\mathit{final}}$ is satisfiable then $M(P_0 \cup \mathit{Defs}_n) = M(P_n)$.*

Example 3. (*List Reversal*) Let *Reverse* be a program for list reversal consisting of the clauses of *Append* (see Example 2) together with the following two clauses (to the right of the clauses we write the corresponding weight polynomials):

3.	$r([\,],[\,]) \leftarrow$	u_3
4.	$r([H\|T],L) \leftarrow r(T,R) \wedge a(R,[H],L)$	u_4

We will transform the *Reverse* program into a program that uses an accumulator [5]. In order to do so, we introduce by Rule 1 the following clause:

5.	$g(L_1,L_2,A) \leftarrow r(L_1,R) \wedge a(R,A,L_2)$	u_5

We apply the unfolding rule twice starting from clause 5 and we get:

6.	$g([\,],L,L) \leftarrow$	$u_5 + u_3 + u_1$
7.	$g([H\|T],L,A) \leftarrow r(T,R) \wedge a(R,[H],S) \wedge a(S,A,L)$	$u_5 + u_4$

By applying the replacement law (α), from clause 7 we derive:

8.	$g([H\|T],L,A) \leftarrow r(T,R) \wedge a([H],A,S) \wedge a(R,S,L)$	$u_5 + u_4 - w_1 + w_2$

together with the constraints (see Example 2): $u_1 \geq 1$, $u_2 \geq 1$, $w_1 \geq w_2$, $w_2 + u_1 \geq 1$.
By two applications of the unfolding rule, from clause 8 we get:

9.	$g([H\|T],L,A) \leftarrow r(T,R) \wedge a(R,[H\|A],L)$	$u_5+u_4-w_1+w_2+u_2+u_1$

By folding clause 9 using clause 5 we get:

10.	$g([H\|T],L,A) \leftarrow g(T,L,[H\|A])$	u_6

together with the constraint $u_6 \leq u_4 - w_1 + w_2 + u_2 + u_1$.
Finally, by folding clause 4 using clause 5 we get:

11.	$r([H\|T],L) \leftarrow g(T,L,[H])$	u_7

together with the constraint $u_7 \leq u_4 - u_5$.

The final program consists of clauses 1, 2, 3, 11, 6, and 10. The correctness constraint system associated with the transformation sequence is as follows.

For clauses 1, 2, and 3:	$u_1 \geq 1,\ u_2 \geq 1,\ u_3 \geq 1.$
For clauses 11, 6, and 10:	$u_7 \geq 1,\ u_5 + u_3 + u_1 \geq 1,\ u_6 \geq 1.$
For the goal replacement:	$u_1 \geq 1,\ u_2 \geq 1,\ w_1 \geq w_2,\ w_2 + u_1 \geq 1.$
For the two folding steps:	$u_6 \leq u_4 - w_1 + w_2 + u_2 + u_1,\ u_7 \leq u_4 - u_5.$

This set of constraints is satisfiable and, therefore, the transformation sequence is totally correct.

5 The Weighted Unfold/Fold Proof Method

In this section we present the unfold/fold method for proving the replacement laws to be used in Rule 4. In order to do so, we introduce the notions of: (i) *syntactic equivalence*, (ii) *symmetric folding*, and (iii) *symmetric goal replacement.*

A *predicate renaming* is a bijective mapping $\rho : \mathit{Preds}_1 \rightarrow \mathit{Preds}_2$, where Preds_1 and Preds_2 are two sets of predicate symbols. Given a formula (or a set of formulas) F, by $\mathit{preds}(F)$ we denote the set of predicate symbols occurring in F. Suppose that $\mathit{preds}(F) \subseteq \mathit{Preds}_1$, then by $\rho(F)$ we denote the formula obtained from F by replacing every predicate symbol p by $\rho(p)$. Two programs Q and R are *syntactically equivalent* if there exists a predicate renaming $\rho : \mathit{preds}(Q) \rightarrow \mathit{preds}(R)$, such that $R = \rho(Q)$, modulo variable renaming.

An application of the folding rule by which from program P_k we derive program P_{k+1}, is said to be *symmetric* if $\mathcal{C}_{k+1}$ is set to $\mathcal{C}_k \cup \{u = \gamma_k(C_1) - \delta_k(D_1), \ldots, u = \gamma_k(C_m) - \delta_k(D_m)\}$ (see Point 2 of Rule 3).
Given a program P, a weight function γ, and a set $\mathcal{C}$ of constraints, we say that the replacement law $(G_1, u_1) \Rightarrow_X (G_2, u_2)$ holds *symmetrically* w.r.t. $\langle P, \gamma, \mathcal{C} \rangle$, and we write $\langle P, \gamma, \mathcal{C} \rangle \models (G_1, u_1) \Leftrightarrow_X (G_2, u_2)$, if the following condition holds:
(ii*) for every solution σ for $\mathcal{C}$,

$$M(\overline{P}) \models \forall X \forall U\, (\exists Y\, (\overline{G}_1[U, \sigma(u_1)]) \leftrightarrow \exists Z\, (\overline{G}_2[U, \sigma(u_2)]))$$

where $\overline{P}$, U, Y, and Z are defined as in Definition 3. Note that, by Lemma 2, Condition (ii*) implies Condition (i) of Definition 3. An application of the *goal replacement rule* is *symmetric* if it consists in applying a replacement law that holds symmetrically w.r.t. $\langle P_0 \cup \mathit{Defs}_k,\ \delta_k,\ \mathcal{C} \rangle$. A weighted unfold/fold transformation sequence is said to be *symmetric* if it is constructed by applications of the definition and unfolding rules and by symmetric applications of the folding and goal replacement rules.

Now we are ready to present the weighted unfold/fold proof method, which is itself based on weighted unfold/fold transformations.

The Weighted Unfold/Fold Proof Method. Let us consider a program P, a weight function γ for P, and a replacement law $(G_1, u_1) \Rightarrow_X (G_2, u_2)$. Suppose that X is the set of variables $\{X_1, \ldots, X_m\}$ and let $\boldsymbol{X}$ denote the sequence $X_1, \ldots, X_m$.
Step 1. First we introduce two new predicates $\mathit{new}1$ and $\mathit{new}2$ defined by the following two clauses: D_1: $\mathit{new}1(\boldsymbol{X}) \leftarrow G_1$ and D_2: $\mathit{new}2(\boldsymbol{X}) \leftarrow G_2$, associated with the unknowns u_1 and u_2, respectively.

Step 2. Then we construct two weighted unfold/fold transformation sequences of the forms: $P \cup \{D_1\} \mapsto \cdots \mapsto Q$ and $P \cup \{D_2\} \mapsto \cdots \mapsto R$, such that the following three conditions hold:

(1) For $i = 1, 2$, the weight function associated with the initial program $P \cup \{D_i\}$ is γ_0^i defined as: $\gamma_0^i(C) = \gamma(C)$ if $C \in P$, and $\gamma_0^i(D_i) = u_i$;

(2) The final programs Q and R are syntactically equivalent; and

(3) The transformation sequence $P \cup \{D_2\} \mapsto \cdots \mapsto R$ is symmetric.

Step 3. Finally, we construct a set $\mathcal{C}$ of constraints as follows. Let γ_Q and γ_R be the weight functions associated with Q and R, respectively. Let $\mathcal{C}_Q$ and $\mathcal{C}_R$ be the correctness constraint systems associated with the transformation sequences $P \cup \{D_1\} \mapsto \cdots \mapsto Q$ and $P \cup \{D_2\} \mapsto \cdots \mapsto R$, respectively, and let ρ be the predicate renaming such that $\rho(Q) = R$. Suppose that both $\mathcal{C}_Q$ and $\mathcal{C}_R$ are satisfiable.

(3.1) Let the set $\mathcal{C}$ be $\{\gamma_Q(C) \geq \gamma_R(\rho(C)) \mid C \in Q\} \cup \mathcal{C}_Q \cup \mathcal{C}_R$. Then we infer:

$\langle P, \gamma, \mathcal{C} \rangle \vdash_{UF} (G_1, u_1) \Rightarrow_X (G_2, u_2)$

(3.2) Suppose that the transformation sequence $P \cup \{D_1\} \mapsto \cdots \mapsto Q$ is symmetric and let the set $\mathcal{C}$ be $\{\gamma_Q(C) = \gamma_R(\rho(C)) \mid C \in Q\} \cup \mathcal{C}_Q \cup \mathcal{C}_R$. Then we infer:

$\langle P, \gamma, \mathcal{C} \rangle \vdash_{UF} (G_1, u_1) \Leftrightarrow_X (G_2, u_2)$

It can be shown that the unfold/fold proof method is sound.

Theorem 6 (Soundness of the Unfold/Fold Proof Method).
If $\langle P, \gamma, \mathcal{C} \rangle \vdash_{UF} (G_1, u_1) \Rightarrow_X (G_2, u_2)$ *then* $\langle P, \gamma, \mathcal{C} \rangle \models (G_1, u_1) \Rightarrow_X (G_2, u_2)$.
If $\langle P, \gamma, \mathcal{C} \rangle \vdash_{UF} (G_1, u_1) \Leftrightarrow_X (G_2, u_2)$ *then* $\langle P, \gamma, \mathcal{C} \rangle \models (G_1, u_1) \Leftrightarrow_X (G_2, u_2)$.

Example 4. (*An Unfold/Fold Proof*) Let us consider again the program *Append* and the replacement law (α), expressing the associativity of list concatenation, presented in Example 2. By applying the unfold/fold proof method we will generate a set $\mathcal{C}$ of constraints such that law (α) holds w.r.t. $\langle Append, \gamma, \mathcal{C} \rangle$.

Step 1. We start off by introducing the following two clauses:

$D_1.\ \ new1(L_1, L_2, L_3, L) \leftarrow a(L_1, L_2, M) \wedge a(M, L_3, L) \qquad w_1$
$D_2.\ \ new2(L_1, L_2, L_3, L) \leftarrow a(L_2, L_3, R) \wedge a(L_1, R, L) \qquad w_2$

Step 2. First, let us construct a transformation sequence starting from $Append \cup \{D_1\}$. By two applications of the unfolding rule, from clause D_1 we derive:

$E_1.\ \ new1([\,], L_2, L_3, L) \leftarrow a(L_2, L_3, L) \qquad w_1 + u_1$
$E_2.\ \ new1([H|T], L_2, L_3, [H|R]) \leftarrow a(T, L_2, M) \wedge a(M, L_3, R) \qquad w_1 + 2u_2$

By folding clause E_2 using clause D_1 we derive:

$E_3.\ \ new1([H|T], L_2, L_3, [H|R]) \leftarrow new1(T, L_2, L_3, R) \qquad u_8$

together with the constraint $u_8 \leq 2u_2$.
Now, let us construct a transformation sequence starting from $Append \cup \{D_2\}$. By unfolding clause D_2 w.r.t. $a(L_1, R, L)$ in its body we get:

$F_1.\ \ new2([\,], L_2, L_3, L) \leftarrow a(L_2, L_3, L) \qquad w_2 + u_1$

F_2. $new2([H|T], L_2, L_3, [H|R]) \leftarrow a(L_2, L_3, M) \wedge a(T, M, R) \qquad w_2 + u_2$

By a symmetric application of the folding rule using clause D_2, from clause F_2 we get:

F_3. $new2([H|T], L_2, L_3, [H|R]) \leftarrow new2(T, L_2, L_3, R) \qquad u_9$

together with the constraint $u_9 = u_2$.

The final programs $Append \cup \{E_1, E_3\}$ and $Append \cup \{F_1, F_3\}$ are syntactically equivalent via the predicate renaming ρ such that $\rho(new1) = new2$. The transformation sequence $Append \cup \{D_2\} \mapsto \cdots \mapsto Append \cup \{F_1, F_3\}$ is symmetric.

Step 3. The correctness constraint system associated with the transformation sequence $Append \cup \{D_1\} \mapsto \cdots \mapsto Append \cup \{E_1, E_3\}$ is the following:

$\mathcal{C}_1$: $\{u_1 \geq 1,\ u_2 \geq 1,\ w_1 + u_1 \geq 1,\ u_8 \geq 1,\ u_8 \leq 2u_2\}$

The correctness constraint system associated with the transformation sequence $Append \cup \{D_2\} \mapsto \cdots \mapsto Append \cup \{F_1, F_3\}$ is the following:

$\mathcal{C}_2$: $\{u_1 \geq 1,\ u_2 \geq 1,\ w_2 + u_1 \geq 1,\ u_9 \geq 1,\ u_9 = u_2\}$

Both $\mathcal{C}_1$ and $\mathcal{C}_2$ are satisfiable and thus, we infer:

$\langle Append, \gamma, \mathcal{C}_{12}\rangle \vdash_{UF}$

$(a(L_1, L_2, M) \wedge a(M, L_3, L),\ w_1) \Rightarrow_{\{L_1,L_2,L_3,L\}} (a(L_2, L_3, R) \wedge a(L_1, R, L),\ w_2)$

where $\mathcal{C}_{12}$ is the set $\{w_1 + u_1 \geq w_2 + u_1,\ u_8 \geq u_9\} \cup \mathcal{C}_1 \cup \mathcal{C}_2$.

Notice that the constraints $w_1 + u_1 \geq w_2 + u_1$ and $u_8 \geq u_9$ are determined by the two pairs of syntactically equivalent clauses (E_1, F_1) and (E_3, F_3), respectively. By eliminating the unknowns u_8 and u_9, which occur in the proof of law (α) only, and by performing some simple simplifications we get, as anticipated, the following set $\mathcal{C}$ of constraints: $\{u_1 \geq 1,\ u_2 \geq 1,\ w_1 \geq w_2,\ w_2 + u_1 \geq 1\}$.

6 Conclusions

We have presented a method for proving the correctness of rule-based logic program transformations in an automatic way. Given a transformation sequence, constructed by using the unfold, fold, and goal replacement transformation rules, we associate some unknown natural numbers, called weights, with the clauses of the programs in the transformation sequence and we also construct a set of linear constraints that these weights must satisfy to guarantee the total correctness of the transformation sequence. Thus, the correctness of the transformation sequence can be proven in an automatic way by checking that the corresponding set of constraints is satisfiable over the natural numbers. However, it can be shown that our method is incomplete and, in general, it can be shown that there exists no algorithmic method for checking whether or not any given unfold/fold transformation sequence is totally correct.

As already mentioned in the Introduction, our method is related to the many methods given in the literature for proving the correctness of program transformation by showing that suitable conditions on the transformation sequence hold (see, for instance, [4, 8, 9, 16, 20, 21], for the case of definite logic programs).

Among these methods, the one presented in [16] is the most general and it makes use of *clause measures* to express complex conditions on the transformation sequence. The main novelty of our method with respect to [16] is that in [16] clause measures are fixed in advance, independently of the specific transformation sequence under consideration, while by the method proposed in this paper we automatically generate specific clause measures for each transformation sequence to be proved correct.

Thus, in principle, our method is more powerful than the one presented in [16]. For a more accurate comparison between the two methods, we did some practical experiments. We implemented our method in the MAP transformation system (`http://www.iasi.cnr.it/~proietti/system.html`) and we worked out some transformation examples taken from the literature. Our system runs on SICStus Prolog (v. 3.12.5) and for the satisfiability of the sets of constraints over the natural numbers it uses the *clpq* SICStus library.

By using our system we did the transformation examples presented in this paper (see Examples 1, 3, and 4) and the following examples taken from the literature: (i) the *Adjacent* program which checks whether or not two elements have adjacent occurrences in a list [9], (ii) the *Equal Frontier* program which checks whether or not the frontiers of two binary trees are equal [5, 21], (iii) a program for solving the N-queens problem [18], (iv) the *In_Correct_Position* program taken from [8], and (v) the program that encodes a liveness property of an n-bit shift register [16]. Even in the most complex derivation we carried out, that is, the *Equal Frontier* example taken from [21], consisting of 86 transformation steps, the system checked the total correctness of the transformation within milliseconds. For making that derivation we also had to apply several replacement laws which were proved correct by using the unfold/fold proof method described in Section 5.

Acknowledgements

We thank the anonymous referees for constructive comments.

References

1. K. R. Apt. Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pp. 493–576. Elsevier, 1990.
2. M. Bezem. Characterizing termination of logic programs with level mappings. In *Proc. of NACLP, Cleveland, Ohio (USA)*, pp. 69–80. MIT Press, 1989.
3. A. Bossi and N. Cocco. Preserving universal termination through unfold/fold. In *Proceedings ALP '94*, LNCS 850, pp. 269–286, Berlin, 1994. Springer-Verlag.
4. A. Bossi, N. Cocco, and S. Etalle. On safe folding. In *Proceedings of PLILP '92, Leuven, Belgium*, LNCS 631, pp. 172–186. Springer-Verlag, 1992.
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. 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*, LNCS 883, pp. 51–68. Springer-Verlag, 1994.
7. S. Etalle and M. Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.
8. M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. *Proceedings PLILP '94*, LNCS 844, pp. 340–354. Springer-Verlag, 1994.
9. T. Kanamori and H. Fujita. Unfold/fold transformation of logic programs with counters. Technical Report 179, ICOT, Tokyo, Japan, 1986.
10. L. Kott. About transformation system: A theoretical study. In *3ème Colloque International sur la Programmation*, pp. 232–247, Paris (France), 1978. Dunod.
11. K.-K. Lau, M. Ornaghi, A. Pettorossi, and M. Proietti. Correctness of logic program transformation based on existential termination. In J. W. Lloyd, editor, *Proceedings of ILPS '95*, pp. 480–494. MIT Press, 1995.
12. J. W. Lloyd. *Foundations of Logic Programming*. Springer, 1987. 2nd Edition.
13. J. McCarthy. Towards a mathematical science of computation. *Proceedings of IFIP 1962*, pp. 21–28, Amsterdam, 1963. North Holland.
14. A. Pettorossi and M. Proietti. A theory of totally correct logic program transformations. *Proceedings of PEPM '04*, pp. 159–168. ACM Press, 2004.
15. A. Roychoudhury, K. Narayan Kumar, C. R. Ramakrishnan, and I. V. Ramakrishnan. Beyond Tamaki-Sato style unfold/fold transformations for normal logic programs. *Int. Journal on Foundations of Computer Science*, 13(3):387–403, 2002.
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. Total correctness by local improvement in the transformation of functional programs. *ACM Toplas*, 18(2):175–234, 1996.
18. T. Sato and H. Tamaki. Examples of logic program transformation and synthesis. Unpublished manuscript, 1985.
19. H. Seki. Unfold/fold transformation of stratified programs. *Theoretical Computer Science*, 86:107–139, 1991.
20. H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. *Proceedings of ICLP '84*, pp. 127–138, Uppsala, Sweden, 1984. Uppsala University.
21. H. Tamaki and T. Sato. A generalized correctness proof of the unfold/fold logic program transformation. Technical Report 86-4, Ibaraki University, Japan, 1986.