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

A. Pettorossi, M. Proietti, V. Senni

CONSTRAINT-BASED CORRECTNESS PROOFS FOR LOGIC PROGRAM TRANSFORMATIONS

R. 24, 2011

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

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

Valerio Senni – Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, I-00133 Roma, Italy.
Email : senni@info.uniroma2.it. URL : http://www.disp.uniroma2.it/users/senni.

ISSN: 1128–3378

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

viale Manzoni 30, 00185 ROMA, Italy

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

Abstract

Many approaches proposed in the literature for proving the correctness of unfold/fold transformations of logic programs make use of measures associated with program clauses. When from a program P_1 we derive a program P_2 by a applying a sequence of transformations, suitable conditions on the measures of the clauses in P_2 guarantee that the transformation of P_1 into P_2 is correct, that is, P_1 and P_2 have the same least Herbrand model. In the approaches 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 clause measures which, instead, takes into account the particular program transformation at hand. During the application of a sequence of transformations we construct a system of linear equalities and inequalities over nonnegative integers 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 more powerful and practical than other methods proposed in the literature. In particular, we are 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 in advance sophisticated clause measures.

Key words: Constraints; Logic programming; Program correctness; Program transformation; Transformation rules.

1. Introduction

Rule-based program transformation is a program development methodology by which one derives, starting from an initial program, a final program by applying a sequence of transformation rules [BuD77, TaS84]. The initial program can be regarded as a formal specification of a software module, while the final program can be regarded as an implementation of that specification. The fact that the rules preserve the intended semantics guarantees that the final program is correct by construction. In logic programming [Apt90, Llo87] program transformation is a deductive process. Indeed, programs are logical formulas and the transformation rules are rules for deducing new formulas from old ones. The logical soundness of the transformation rules implies that a transformation is *partially correct*, which means that an atomic formula 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*, which means that an atomic formula is true in the initial program if and only if it is true in the final program.

In particular, the transformations obtained by applying transformation rules such as *unfolding* and *folding*, which basically consist in applying equivalences that hold in the least Herbrand model of the initial program, are always partially correct. However, the final program derived by unfolding and folding may terminate (with respect to a suitable notion of termination) less often than the initial one. For instance, let us consider the program:

P: $p \leftarrow q$ $\qquad$ $r \leftarrow q$ $\qquad$ $q \leftarrow$

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: $p \leftarrow q$ $\qquad$ $r \leftarrow p$ $\qquad$ $q \leftarrow$

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: $p \leftarrow p$ $\qquad$ $r \leftarrow q$ $\qquad$ $q \leftarrow$

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, $p \notin M(R)$ because program R does not terminate for the goal p.

A sufficient condition for the total correctness of a transformation obtained by the unfolding and folding rules is that termination is preserved, that is, the final program terminates as often as the initial one. In particular, total correctness is guaranteed if the final program obtained by transformation always terminates. This method for proving total correctness is the one proposed in Burstall and Darlington's seminal paper [BuD77] and is sometimes referred to as *McCarthy's method* [McC63]. However, the termination condition may be, in practice, very hard to verify. For this reason some methods proposed in the context of functional programming are based on properties of the transformations that imply the preservation of termination, without actually having to verify the termination condition. For instance: (i) [Kot78] shows that under suitable assumptions, total correctness is guaranteed if the final program is derived by a sequence of transformations where the number of applications of the unfolding rule is not smaller than the number of applications of the folding rule, and (ii) [San96] identifies some applicability conditions for the unfolding and folding rules which ensure that the number of steps needed to evaluate a given expression is not increased and, therefore, program termination is preserved.

A lot of work has also been devoted to devise methods for proving the total correctness of transformations of logic programs (see, for instance, [BoC94, BCE92, CoG94, EtG96, GeK94, KaF86, LOP95, Mah87, Mah93, PeP08, RKR02, RKR04, Sek91, TaS84, TaS86]). The simplest

among these methods consists in considering *invertible* transformation rules, that is, rules which allow a program P_1 to be transformed into a program P_2 only if P_2 can be transformed back into P_1 [Mah87, Mah93]. The total correctness of a transformation obtained by an invertible rule immediately follows from the fact that, by partial correctness, both $M(P_1) \subseteq M(P_2)$ and $M(P_2) \subseteq M(P_1)$ holds. For instance, the transformation of program P into program Q shown above is invertible because Q can be transformed back into P by unfolding the clause $r \leftarrow p$. On the contrary, the transformation of P into R is not invertible. Unfortunately, this method of guaranteeing total correctness is of very limited use because many relevant transformations are not invertible (in particular, those transformations that derive recursive definitions from nonrecursive ones).

Other methods [CoG94, LOP95] propose sufficient conditions for total correctness which are explicitly based on the preservation of suitable termination properties (such as, the *universal* or the *existential* termination). However, as already mentioned, termination conditions may be, in practice, very hard to verify.

Some other methods, which we may call *history-based methods*, are based on conditions on the sequence of applications of the transformation rules that do not deal with termination explicitly, but nevertheless guarantee that termination is preserved. A notable example of these history-based methods is presented in [KaF86], 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, respectively) takes place. A sequence of transformations is totally correct if the counters of the clauses of the final program are all positive. This result can be viewed as an extension to logic programming of the approach presented in [Kot78].

Unfortunately, there are many simple transformations where the method based on counters is not able to prove 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 counter method, in the basic form we have described, does not allow us to show the total correctness of that transformation. In order to overcome the limitations of the basic counter method, some modifications and enhancements have been described in [KaF86, RKR02, RKR04, TaS86], where each clause is given a measure which is more complex than an integer counter. In particular, these complex clause measures may combine counters with suitable orderings on predicate symbols.

In this paper we present a different approach to the improvement of the basic counter method: instead of fixing *in advance* complex clause measures, we automatically generate for any given transformation a set of constraints on clause measures whose satisfiability guarantees the correctness of the transformation. For reasons of simplicity we assume that clause measures, which we call *weights*, are nonnegative integers, and constraints are linear equalities and inequalities. 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 briefly recall the *well-founded annotation* method proposed in [PeP08] on which the correctness of the method presented in this paper is based. In Section 3 we present the notion of a *transformation sequence*, that is, a sequence of programs constructed by applying the definition introduction, unfolding, folding, and clause deletion 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 equalities and inequalities which relate those weights. If the final set of constraints is satisfiable for a suitable assignment to the unknown weights, then the transformation

sequence is totally correct. This total correctness result is proved in Section 4. In Section 5 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 as well. In Section 6 we present a method for proving predicate properties which are needed for applying the goal replacement rule. Finally, in Section 7 we discuss related work in the field of program transformation and, in particular, we argue that our approach is more powerful than other history-based methods.

2. Well-Founded Annotations

In this section we briefly recall the theory of *well-founded annotations*, which is a general theory proposed in [PeP08] for proving the correctness of program transformations. This theory will be used to prove the correctness results presented in Sections 3–6 and also to motivate the technical definitions introduced in these sections.

In this paper we assume that a clause is of the form $A_0 \leftarrow A_1 \wedge \ldots \wedge A_n$, with $n \geq 0$, where $A_0, A_1, \ldots, A_n$ are atoms and the conjunction operator '$\wedge$' is associative and commutative with neutral element '*true*'. A program consists of a *multiset* of clauses, that is, a clause may have more than one occurrence in a program. This assumption will make it easier to deal with the fact that different occurrences of a clause which are derived by transformation may be associated with different annotations. For multisets we will use set-theoretic notations and it will be clear from the context when we refer to sets and when we refer to multisets. In particular, we will denote a multiset of clauses by $\{C_1, \ldots, C_n\}$, the predicate of multiset membership by $\in$, the empty multiset by $\emptyset$, and the union and the difference operations between multisets by $\cup$ and $-$, respectively.

The theory of well-founded annotations studies the correctness of a very general notion of program transformation, called *clause replacement*, consisting in replacing m clauses $C_1, \ldots, C_m$, with $m \geq 0$, occurring in a program by n clauses $D_1, \ldots, D_n$, with $n \geq 0$, such that $C_1, \ldots, C_m$ are equivalent to $D_1, \ldots, D_n$ with respect to a suitable equivalence relation. In particular, the transformations obtained by applying the unfolding, folding, and goal replacement rules often considered in the literature, can be regarded as clause replacement transformations. The well-founded annotation method consists in associating with each atom occurrence an *annotation*, that is, an extra argument that holds the measure of a proof of the atom. The theory of well-founded annotations provides some sufficient conditions for the total correctness of a clause replacement based on the preservation of a suitable well-founded ordering on the annotations.

Let us introduce the notions of implication and equivalence between multisets of clauses upon which a clause replacement depends.

Definition 2.1. Let I be an Herbrand interpretation and let Γ_1 and Γ_2 be two multisets 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 $I \models \Gamma_1 \Leftarrow \Gamma_2$.

Note that if (a) $I \models \Gamma_1 \Rightarrow \Gamma_2$ holds, then (b) in the interpretation I the conjunction of the clauses of Γ_1 implies (in the sense of classical first order logic) the conjunction of the clauses of Γ_2. However, in general it is not the case that if (b) holds, then (a) holds. For instance, the empty conjunction *true* implies the tautology $p \leftarrow p$, while $I \models \emptyset \Rightarrow \{p \leftarrow p\}$ does not hold. Our

definition of the $\Rightarrow$ relation between multisets of clauses plays a crucial role in Theorem 2.4 and Corollary 2.5 below, which do not hold if we replace $\Rightarrow$ with classical first order logic implication.

For all Herbrand interpretations I and multisets Γ_1, Γ_2, and Γ_3 of clauses 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 [Apt90, Llo87]. The T_P operator has a least and a greatest fixpoint, denoted by $lfp(T_P)$ and $gfp(T_P)$, respectively. Recall that, $M(P)$ denotes the least Herbrand model of P and we have that $M(P) = lfp(T_P)$.

Now let us consider the transformation of a program P into a program Q consisting in replacing a multiset Γ_1 of clauses in P by a new multiset Γ_2 of clauses. The following result expresses the *partial correctness* of the transformation of P into Q.

Theorem 2.2 (Partial Correctness) *Given two programs P and Q, such that: (i) for some multisets 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 2.3 (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 shows that the converse property of partial correctness holds when the program derived by a clause replacement is univocal.

Theorem 2.4 (Conservativity) *Given two programs P and Q, such that: (i) for some multisets 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 2.2 and 2.4 we get the following result.

Corollary 2.5 (Total Correctness Via Unique Fixpoint) *Given two programs P and Q such that: (i) for some multisets 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 $M(P) = M(Q)$.*

In Theorem 2.4 the hypothesis (iii) is crucial. Indeed, in the example we have given in the Introduction, we have that Q is univocal and the transformation from P to Q is totally correct, while R is not univocal and the transformation from P to R is not totally correct.

Corollary 2.5 has severe applicability limitations because: (i) to prove that a program is univocal may be very difficult in practice, as it may require to prove that the program is terminating for all ground goals [Bez89], and (ii) one may want to derive programs that are not univocal (and, thus, not terminating).

In order to overcome these limitations the method proposed in [PeP08] introduced the notion of *annotated program*. Here we will define a subclass of the annotated programs, called *weighted programs*, which will be sufficient for our purposes.

Definition 2.6 (Weighted Clause and Weighted Program) *Given a clause C of the form $p_0(t_0) \leftarrow p_1(t_1) \wedge \ldots \wedge p_m(t_m)$, where $t_0, t_1, \ldots, t_m$ are tuples of terms, a* weighted clause *associated with C is a clause of the form:*

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

where $N_0, N_1, \ldots, N_m$ are variables not occurring in C and w is a nonnegative integer called the weight *of $\overline{C}[w]$. Clause $\overline{C}[w]$ is denoted by $\overline{C}$ when we do not need refer to its weight w. Given a program $P = \{C_1, \ldots, C_r\}$, a* weighted program *associated with P is a program $\overline{P}$ of the form $\{\overline{C}_1, \ldots, \overline{C}_r\}$.*

Let us now explain in an informal way the meaning of a weighted program. Basically, $\overline{P}$ is constructed by associating a nonnegative weight with each clause of P. This association induces a notion of weight on proofs of atoms. Indeed, we may define the weight of a given proof of an atom A as the sum of the weights of all clause instances used in that proof of A. Thus, by the definition of $\overline{P}$, we may say that the atom $p(t, n)$ holds if $p(t)$ has a proof 'of weight less than or equal to n' in P.

The semantics of weighted programs is defined similarly to the semantics of *constraint logic programs* [JaM94]. Let $\mathcal{N}$ be the first order interpretation defined as follows: (i) the carrier of $\mathcal{N}$ is the set $\mathbb{N}$ of the *nonnegative* integers, (ii) each nonnegative integer number is interpreted as the corresponding element in $\mathbb{N}$, (iii) the function symbol $+$ is interpreted as the addition operation in $\mathbb{N}$, and (iv) the predicate symbol $\geq$ is interpreted as the greater-than-or-equal-to relation on $\mathbb{N}$. An *$\mathcal{N}$-interpretation* for a weighted program $\overline{P}$ is a subset of the following set, where t denotes a tuple of terms:

$B_{\mathbb{N}} = \{p(t, n) \mid p(t)$ is a ground instance of an atom from P and $n \in \mathbb{N}\}$

The notions of (i) *truth* of a formula (in particular, of a weighted program) in an $\mathcal{N}$-interpretation, (ii) the *$\mathcal{N}$-model* of a formula, and (iii) the *immediate consequence operator* associated to a weighted program, are defined as usual in constraint logic programming [JaM94]. Given a formula F and an $\mathcal{N}$-interpretation I, by $I \models F$ we denote that F is true in I. Given a weighted program $\overline{P}$, it can be shown that there exists a *least* $\mathcal{N}$-model, denoted by $M(\overline{P})$. The immediate consequence operator $T_{\overline{P}}$ associated with $\overline{P}$ has a least fixpoint, denoted by $lfp(T_{\overline{P}})$, and $M(\overline{P}) = lfp(T_{\overline{P}})$.

Definitions 2.1 and 2.3, Theorems 2.2 and 2.4, and Corollary 2.5, extend straightforwardly to weighted programs.

Now we present some results for weighted programs which are immediate consequences of similar results proved for annotated programs in [PeP08].

The following lemma establishes the relationship between the semantics of a program P and the semantics of any weighted program $\overline{P}$ associated with P.

Lemma 2.7. *Let P be a program. For every ground atom $p(t)$, $p(t) \in M(P)$ iff there exists $n \in \mathbb{N}$ such that $p(t, n) \in M(\overline{P})$.*

By erasing weights from clauses we preserve clause implications, in the sense stated by the following lemma.

Lemma 2.8. *Let P be a program, and Γ_1 and Γ_2 be any two multisets of clauses. If $M(\overline{P}) \models \overline{\Gamma}_1 \Rightarrow \overline{\Gamma}_2$ then $M(P) \models \Gamma_1 \Rightarrow \Gamma_2$.*

Definition 2.9 (Decreasing, Weighted Program) *A weighted program $\overline{P}$ is said to be* decreasing *if every clause in $\overline{P}$ has a positive weight.*

Lemma 2.10. *Every decreasing program is univocal.*

Finally, by Lemmata 2.7 and 2.10, and Theorems 2.2 and 2.4, we have the following result which, unlike Corollary 2.5, 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 2.11 (Total Correctness Via Weights) *Given two programs P and Q such that: (i) for some multisets Γ_1, Γ_2 of clauses, $Q = (P - \Gamma_1) \cup \Gamma_2$, (ii) $M(P) \models \Gamma_1 \Rightarrow \Gamma_2$, (iii) $M(\overline{P}) \models \overline{\Gamma}_1 \Leftarrow \overline{\Gamma}_2$, and (iv) $\overline{Q}$ is decreasing. Then $M(P) = M(Q)$.*

Note that, by Theorem 2.4, Conditions (*iii*) and (*iv*) of Theorem 2.11 imply that $M(\overline{P}) \subseteq M(\overline{Q})$ which, by taking into account the informal meaning of a weighted program, can be read as follows: for every ground atom A, if A has a proof of weight at most n in P, then A has a (not necessarily shorter) proof of weight at most n in Q.

As already mentioned, the properties $M(P) \models \Gamma_1 \Rightarrow \Gamma_2$ and $M(P) \models \Gamma_1 \Leftarrow \Gamma_2$ are guaranteed when Γ_2 is derived from Γ_1 by applying the usual unfolding, folding, and goal replacement rules. However, in order to use Theorem 2.11 and prove that the transformation of P into $Q = (P - \Gamma_1) \cup \Gamma_2$ is totally correct, one has to associate with the clauses of P (in particular, with the clauses of Γ_1) and with the clauses of Γ_2, suitable weights such that also the property $M(\overline{P}) \models \overline{\Gamma}_1 \Leftarrow \overline{\Gamma}_2$ holds and, moreover, $\overline{Q}$ is decreasing.

The problem of finding suitable weights in an automatic way in the cases where Γ_2 is derived by Γ_1 by applying the unfolding, folding, and goal replacement rules will be addressed in the following Sections 3, 4, 5, and 6.

3. Unfold/Fold Transformation Rules with Weight Constraints

In this section we consider sequences of programs obtained by applying the definition introduction, unfolding, folding, and deletion transformation rules, and we address the problem of finding suitable weight assignments so that, by Theorem 2.11, the transformation of the initial program of the sequence into the final program of the sequence is totally correct. With every clause of the initial program and with every new clause generated during the transformation, we associate an unknown, called *weight unknown*, ranging over nonnegative integers, called *weights*, and while constructing the sequence of programs, we construct a system of linear equalities and inequalities which should be satisfied by the weight unknowns. The total correctness of the transformation is guaranteed by the satisfiability of that system.

Before presenting the transformation rules, let us introduce some terminology concerning systems of linear equalities and inequalities with integer coefficients and variables ranging over nonnegative integers.

By $\mathcal{P}_{LIN}$ we denote the set of linear polynomials with integer coefficients. Variables occurring in polynomials are called *weight unknowns*, or *unknowns*, for short, to distinguish them from logical variables occurring in programs. By $\mathcal{U}$ we denote the set of unknowns in $\mathcal{P}_{LIN}$. By $\mathscr{C}_{LIN}$ we denote the set of linear equalities and inequalities with integer coefficients, that is, $\mathscr{C}_{LIN}$ 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 $\mathscr{C}_{LIN}$ is called a *constraint*. A *valuation* for a set $\{u_1, \ldots, u_r\}$ of unknowns is a mapping $\sigma\colon \{u_1, \ldots, u_r\} \to \mathbb{N}$, where $\mathbb{N}$ is the set of nonnegative integers. Let $\{u_1, \ldots, u_r\}$ be the set of unknowns occurring in the constraint $c \in \mathscr{C}_{LIN}$. Given a valuation σ whose domain is a superset of $\{u_1, \ldots, u_r\}$, we denote by $\sigma(c)$ the constraint obtained from c by

replacing the occurrences of $u_1, \ldots, u_r$ by the *weights* $\sigma(u_1), \ldots, \sigma(u_r)$, respectively. A valuation σ is a *solution* of the constraint c if σ is a valuation whose domain is a superset of the set of variables occurring in c and $\mathcal{N} \models \sigma(c)$. A valuation σ is a solution of a finite set $\mathscr{C}$ of constraints if, for every $c \in \mathscr{C}$, σ is a solution of c. We say that a constraint c is *satisfiable* if there exists a solution of c. Similarly, we say that a set $\mathscr{C}$ of constraints is *satisfiable* if there exists a solution of $\mathscr{C}$. A *weight function* for a multiset S of clauses is a function $\gamma : S \to \mathcal{U}$ such that for any two distinct occurrences C_1 and C_2 of clauses in S, $\gamma(C_1) \neq \gamma(C_2)$. In particular, for two distinct occurrences of the same clause C in S, γ returns two different unknowns. The value $\gamma(C)$ is called the *unknown associated with* C.

A *transformation sequence* is a sequence of programs, denoted by $P_0 \mapsto P_1 \mapsto \cdots \mapsto P_n$, 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: *definition introduction*, *unfolding*, *folding*, and *deletion*. These rules will be defined below. For $k = 0, \ldots, n$, we define: (i) a weight function $\gamma_k : P_k \to \mathcal{U}$, (ii) a finite set $\mathscr{C}_k$ of constraints, (iii) a multiset 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 $\delta_k : P_0 \cup \mathit{Defs}_k \to \mathcal{U}$. The function γ_0 can be taken to be any weight function. By definition, we have that: $\mathscr{C}_0 = \emptyset$, $\mathit{Defs}_0 = \emptyset$, and $\delta_0 = \gamma_0$.

Each application of a transformation rule consists in replacing a multiset Γ_1 of clauses in P_k by a multiset Γ_2. Thus, $P_{k+1} = (P_k - \Gamma_1) \cup \Gamma_2$. We make the following assumptions on the weight functions γ_k and γ_{k+1}:

(1) for every clause C occurring in Γ_2 the value of the weight function γ_{k+1} is a new unknown, that is, for $i = 0, \ldots, k$, for every clause D occurring in P_i, $\gamma_{k+1}(C) \neq \gamma_i(D)$, and

(2) for every clause of P_{k+1} inherited from P_k, the value of the weight function γ_{k+1} is equal to the value of γ_k, that is, for every clause C occurring in $P_k - \Gamma_1$, $\gamma_{k+1}(C) = \gamma_k(C)$.

For $k = 0, \ldots, n$, for every clause C occurring in $P_0 \cup \mathit{Defs}_k$, the value of $\delta_k(C)$ is the unknown which has been associated with C when C was first introduced during the transformation sequence $P_0 \mapsto P_1 \mapsto \cdots \mapsto P_k$.

We also assume that, for $k = 0, \ldots, n-1$, $\mathscr{C}_{k+1} = \mathscr{C}_k$ and $\mathit{Defs}_{k+1} = \mathit{Defs}_k$, unless otherwise specified in the definition of the transformation rule applied for deriving P_{k+1} from P_k.

In the sequel, we will feel free to rename variables of clauses, whenever needed (in particular, when applying the unfolding and folding rules), and for any goal (or set of goals) G, by $\mathit{vars}(G)$ we denote the set of variables occurring in G.

Rule 1 (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 $P_0 \cup \mathit{Defs}_k$. (Note that the head predicate of D_i may be equal to the head predicate of D_j, for some $i \neq j$ in $\{1, \ldots, m\}$.) By *definition introduction* from P_k we derive $P_{k+1} = P_k \cup \{D_1, \ldots, D_m\}$. We define:

$\mathit{Defs}_{k+1} = \mathit{Defs}_k \cup \{D_1, \ldots, D_m\}$.

As a consequence of our definitions, given a transformation sequence $P_0 \mapsto P_1 \mapsto \cdots \mapsto P_n$, we have that, for $k = 0, \ldots, n$, for any clause $C \in P_0 \cup \mathit{Defs}_k$, if $C \in P_0$, then $\delta_k(C) = \gamma_0(C)$ and, otherwise, if $C \in \mathit{Defs}_k$, then $\delta_k(C) = \gamma_i(C)$, where $i \in \{1, \ldots, k\}$, $C \notin \mathit{Defs}_{i-1}$, and $C \in \mathit{Defs}_i$.

Note that the definition introduction rule does not introduce any constraint on the weight unknowns and, thus, we are free to assign any weight to the clauses introduced by that rule.

Rule 2 (Unfolding) Let C: $H \leftarrow G_L \wedge A \wedge G_R$ be a clause in P_k and let C_1: $H_1 \leftarrow G_1, \ldots, C_m$: $H_m \leftarrow G_m$, with $m \geq 0$, be *all* clauses in $P_0 \cup \mathit{Defs}_k$ such that, for $i = 1, \ldots, m$, A is unifiable with H_i via a most general unifier ϑ_i.

By *unfolding C with respect to 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 define:

$\mathscr{C}_{k+1} = \mathscr{C}_k \cup \{\gamma_{k+1}(D_1) = \gamma_k(C) + \delta_k(C_1),\ \ldots,\ \gamma_{k+1}(D_m) = \gamma_k(C) + \delta_k(C_m)\}$.

Let us briefly explain the definition of $\mathscr{C}_{k+1}$ in Rule 2. As already mentioned, our aim is to prove the total correctness of a transformation by exploiting Theorem 2.11. We will show in Lemma 4.1 of Section 4 that, if we associate with the clauses $C, C_1, \ldots, C_m, D_1, \ldots$, and D_m, a set of weights satisfying the set $\mathscr{C}_{k+1}$ of constraints, then Point (iii) of Theorem 2.11 holds, that is, $M(\overline{P}_0 \cup \overline{Defs}_{k+1}) \models \{\overline{C}\} \Leftarrow \{\overline{D}_1, \ldots, \overline{D}_m\}$. (Actually, Lemma 4.1 proves a more general property which refers to any $n \geq k+1$.) In order to see why this property holds now we present a simple example.

Example 1. Let C: $p \leftarrow q$ be a clause in P_k and let C_1: $q \leftarrow r$ be the only clause defining q in $P_0 \cup Defs_k$. By unfolding C with respect to q using C_1 we get D_1: $p \leftarrow r$. Let w and w_1 be the weights associated with C and C_1, respectively. Thus, in order to satisfy $\mathscr{C}_{k+1}$, the weight of D_1 must be $w + w_1$. The weighted clauses associated with C, C_1, and D_1 are:

$\overline{C}$: $p(N) \leftarrow N \geq U + w \ \wedge \ q(U)$
$\overline{C}_1$: $q(N) \leftarrow N \geq V + w_1 \ \wedge \ r(V)$
$\overline{D}_1$: $p(N) \leftarrow N \geq V + w + w_1 \ \wedge \ r(V)$

By the definition of $\Leftarrow$ (see Definition 2.1), in order to prove that $M(\overline{P}_0 \cup \overline{Defs}_{k+1}) \models \{\overline{C}\} \Leftarrow \{\overline{D}_1\}$ we have to show that for every ground goal of the form $n \geq u + w \wedge q(u)$ that holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$, there exists a ground goal of the form $n \geq v + w + w_1 \wedge r(v)$ that holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$. This immediately follows from the fact that $\overline{C}_1$ is the only clause defining $q(N)$ in $\overline{P}_0 \cup \overline{Defs}_{k+1}$ and, thus, if $q(u)$ holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$ then there exists v such that $u \geq v + w_1 \wedge r(v)$ holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$.

Rule 3 (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 $P_0 \cup Defs_k$. 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 $(P_0 \cup Defs_k) - \{D_1, \ldots, D_m\}$ whose head is unifiable with $K\vartheta$, and (iii) for $i = 1, \ldots, m$ and for every variable U in $vars(B_i) - vars(K)$: (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 define:

$\mathscr{C}_{k+1} = \mathscr{C}_k \cup \{\gamma_{k+1}(E) \leq \gamma_k(C_1) - \delta_k(D_1),\ \ldots,\ \gamma_{k+1}(E) \leq \gamma_k(C_m) - \delta_k(D_m)\}$.

Similarly to the unfolding rule, the definition of $\mathscr{C}_{k+1}$ in Rule 3 allows us to prove that, when we derive clause E by folding clauses $C_1, \ldots, C_m$ using clauses $D_1, \ldots, D_m$, then Point (iii) of Theorem 2.11 holds. In Lemma 4.2 of Section 4 we will show that, if we associate with the clauses $C_1, \ldots, C_m, D_1, \ldots, D_m, E$ a set of weights satisfying the set $\mathscr{C}_{k+1}$ of constraints, then we have that $M(\overline{P}_0 \cup \overline{Defs}_{k+1}) \models \{\overline{C}_1, \ldots, \overline{C}_m\} \Leftarrow \{\overline{E}\}$. Now we show that this property holds in an example.

Example 2. Let C_1: $p \leftarrow q \wedge r_1$ and C_2: $p \leftarrow q \wedge r_2$ be clauses in P_k and let D_1: $r \leftarrow r_1$ and D_2: $r \leftarrow r_2$ be the only clauses defining r in $P_0 \cup Defs_k$. By folding C_1, C_2 using D_1, D_2 we get E:

$p \leftarrow q \wedge r$. Let w_1, w_2 and z_1, z_2 be the weights associated with C_1, C_2 and D_1, D_2, respectively. Thus, in order to satisfy $\mathscr{C}_{k+1}$, the weight of E is a nonnegative integer w such that $w \leq w_1 - z_1$ and $w \leq w_2 - z_2$. The weighted clauses associated with C_1, C_2, D_1, D_2, E are:

$\overline{C}_1: \ p(N) \leftarrow N \geq K+L+w_1 \ \wedge \ q(K) \ \wedge \ r_1(L)$
$\overline{C}_2: \ p(N) \leftarrow N \geq K+L+w_2 \ \wedge \ q(K) \ \wedge \ r_2(L)$
$\overline{D}_1: \ r(M) \leftarrow M \geq L+z_1 \ \wedge \ r_1(L)$
$\overline{D}_2: \ r(M) \leftarrow M \geq L+z_2 \ \wedge \ r_2(L)$
$\overline{E} \ : \ p(N) \leftarrow N \geq K+M+w \ \wedge \ q(K) \ \wedge \ r(M)$

According to the definition of $\Leftarrow$ (see Definition 2.1), we have that in order to prove that $M(\overline{P}_0 \cup \overline{Defs}_{k+1}) \models \{\overline{C}_1, \overline{C}_2\} \Leftarrow \{\overline{E}\}$ we have to show that: (1) for every ground goal of the form $n \geq j + l + w_1 \wedge q(j) \wedge r_1(l)$ that holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$, there exists a ground goal of the form $n \geq j + m + w \wedge q(j) \wedge r(m)$ that holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$, and (2) for every ground goal of the form $n \geq j + l + w_2 \wedge q(j) \wedge r_2(l)$ that holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$, there exists a ground goal of the form $n \geq j + m + w \wedge q(j) \wedge r(m)$ that holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$. Properties (1) and (2) immediately follow from the following facts: (i) $w_1 \geq z_1 + w$, (ii) $w_2 \geq z_2 + w$, and, by clauses D_1, D_2, (iii) for every m, $r(m)$ holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$ if either $m \geq l + z_1 \wedge r_1(l)$ or $m \geq l + z_2 \wedge r_2(l)$ holds in $M(\overline{P}_0 \cup \overline{Defs}_{k+1})$.

Rule 4 (Deletion of Subsumed Clauses) Let C: $H_1 \leftarrow G_1 \wedge R$ and D: $H_2 \leftarrow G_2$ be two clauses in P_k such that, for some substitution ϑ, we have $H_1 \leftarrow G_1 = (H_2 \leftarrow G_2)\vartheta$. C is said to be *subsumed* by D.
By *deletion* from P_k we derive $P_{k+1} = P_k - \{C\}$. We define:

$\mathscr{C}_{k+1} = \mathscr{C}_k \cup \{\gamma_k(D) \leq \gamma_k(C)\}$.

Note that by Rule 4 we can delete multiple occurrences of a clause from a program. Similarly to Rules 2 and 3, the definition of $\mathscr{C}_{k+1}$ in Rule 4 allows us to prove that when we apply this rule, Point (iii) of Theorem 2.11 holds. Indeed, in Lemma 4.3 of Section 4 we will show that, if we associate with clauses C and D a set of weights satisfying the set $\mathscr{C}_{k+1}$ of constraints, then $M(\overline{P}_0 \cup \overline{Defs}_{k+1}) \models \{\overline{C}, \overline{D}\} \Leftarrow \{\overline{D}\}$.

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

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

We say that a transformation sequence $P_0 \mapsto \cdots \mapsto P_n$ is *totally correct* if we have that $M(P_0 \cup Defs_n) = M(P_n)$.

The following result that is proved in Section 4, guarantees the total correctness of transformation sequences constructed by using Rules 1–4.

Theorem 3.1 (Total Correctness of Unfold/Fold Transformations) *Let $P_0 \mapsto \cdots \mapsto P_n$ be a transformation sequence constructed by using Rules 1–4, and let $\mathscr{C}_{final}$ be its associated correctness constraint system. If $\mathscr{C}_{final}$ is satisfiable then $M(P_0 \cup Defs_n) = M(P_n)$.*

Note that during the construction of a transformation sequence $P_0 \mapsto \cdots \mapsto P_n$ we may eliminate some unknowns from a set $\mathscr{C}_k$ of constraints, with $0 \leq k \leq n$, without affecting the satisfiability of $\mathscr{C}_{final}$. In particular, let us suppose that $\mathscr{C}_k$ is the set $\{c_1, \ldots, c_m\}$ and u is an unknown not belonging to the range of either γ_k or δ_k. Let us also suppose that, for some set $\{d_1, \ldots, d_r\}$ of constraints, $\mathcal{N} \models \forall\, ((\exists u\, (c_1 \wedge \ldots \wedge c_m)) \leftrightarrow (d_1 \wedge \ldots \wedge d_r))$. Then we can replace $\mathscr{C}_k$ by $\{d_1, \ldots, d_r\}$. Indeed, by the definitions of the transformation rules, u will not occur in any constraint added

during the construction of the transformation sequence $P_k \mapsto \cdots \mapsto P_n$. Thus, $\mathscr{C}_{final}$ is of the form $\{c_1, \ldots, c_m, c_{m+1}, \ldots, c_s\}$, where u does not occur in $c_{m+1}, \ldots, c_s$ and, therefore, $\mathscr{C}_{final}$ is satisfiable if and only if $\{d_1, \ldots, d_r, c_{m+1}, \ldots, c_s\}$ is satisfiable. During the presentation of our derivations we will often eliminate unknowns to simplify the sets of constraints.

Let us now present an example of application of the transformation rules. In this example and in other examples below, we will enumerate clauses and we will denote by u_i the value of the weight function γ for clause i. We will write the constraints on the unknown associated with the clauses on a column to the right of the clauses themselves.

Example 3. (*Continuation Passing Style Transformation*) Let us consider the initial program P_0 consisting of the following three clauses:

1. $p \leftarrow$
2. $p \leftarrow p \wedge q$
3. $q \leftarrow$

We want to derive a *continuation passing style* program [Wan80] defining a predicate p_{cont} equivalent to the predicate p defined by the program P_0. Intuitively, the continuation of a predicate call is a term encoding the computation to be performed after that call. In order to derive a continuation passing style program, we introduce, by Rule 1, the following clause:

4. $p_{cont} \leftarrow p \wedge cont(true)$

and also the following three clauses for the predicate *cont*:

5. $cont(true) \leftarrow$
6. $cont(p(X)) \leftarrow p \wedge cont(X)$
7. $cont(q(X)) \leftarrow q \wedge cont(X)$

where, by abuse of notation, we used p and q both as function symbols and predicate symbols. By folding clause 4 using clause 6 we get:

8. $p_{cont} \leftarrow cont(p(true))$ $\qquad u_8 \leq u_4 - u_6$

By unfolding clause 6 with respect to p using clauses 1 and 2, we get:

9. $cont(p(X)) \leftarrow cont(X)$ $\qquad u_9 = u_6 + u_1$
10. $cont(p(X)) \leftarrow p \wedge q \wedge cont(X)$ $\qquad u_{10} = u_6 + u_2$

Then by folding clause 10 using clause 7 we get:

11. $cont(p(X)) \leftarrow p \wedge cont(q(X))$ $\qquad u_{11} \leq u_{10} - u_7$

and by folding clause 11 using clause 6 we get:

12. $cont(p(X)) \leftarrow cont(p(q(X)))$ $\qquad u_{12} \leq u_{11} - u_6$

Finally, by unfolding clause 7 with respect to q we get:

13. $cont(q(X)) \leftarrow cont(X)$ $\qquad u_{13} = u_7 + u_3$

The final program is made out of clauses 1, 2, and 3, together with the following clauses:

8. $p_{cont} \leftarrow cont(p(true))$
5. $cont(true) \leftarrow$
9. $cont(p(X)) \leftarrow cont(X)$
12. $cont(p(X)) \leftarrow cont(p(q(X)))$
13. $cont(q(X)) \leftarrow cont(X)$

The correctness constraint system $\mathscr{C}_{final}$ is constructed as follows. For clauses 1, 2, 3, 5, 8, 9, 12, and 13, that are the clauses belonging to the final program, $\mathscr{C}_{final}$ contains the constraints:

$u_1 \geq 1, \quad u_2 \geq 1, \quad u_3 \geq 1, \quad u_5 \geq 1, \quad u_8 \geq 1, \quad u_9 \geq 1, \quad u_{12} \geq 1, \quad u_{13} \geq 1.$

For the unfolding and folding steps, $\mathscr{C}_{final}$ contains the constraints:

$u_8 \leq u_4 - u_6, \quad u_9 = u_6 + u_1, \quad u_{10} = u_6 + u_2, \quad u_{11} \leq u_{10} - u_7, \quad u_{12} \leq u_{11} - u_6, \quad u_{13} = u_7 + u_3.$

This system of constraints is satisfiable and, thus, the transformation from program P_0 to the final program is totally correct.

We end this section by presenting an example where the constraint system associated with a transformation sequence is unsatisfiable and, as expected, the transformation sequence is not totally correct.

Example 4. Let us consider the initial program P_0 consisting of the following three clauses:

1. $p \leftarrow p$
2. $p \leftarrow q$
3. $q \leftarrow$

By unfolding clause 1, we replace it by the following two clauses:

4. $p \leftarrow p$ $\qquad u_4 = u_1 + u_1$
5. $p \leftarrow q$ $\qquad u_5 = u_1 + u_2$

We derive the program P_1 which is made out of clauses 2, 3, 4, and 5, and the set of constraints $\mathscr{C}_1$ which is $\{u_4 = u_1 + u_1,\ u_5 = u_1 + u_2\}$. By clause deletion, from P_1 we remove clause 2, which is subsumed by (in fact, it is identical to) clause 5, and we derive P_2 made out of clauses 3, 4, and 5 together with the set of constraints $\mathscr{C}_2$ which is $\mathscr{C}_1 \cup \{u_5 \leq u_2\}$. Finally, we fold clauses 4 and 5 using clauses 1 and 2 (which belong to P_0) and we get:

6. $p \leftarrow p$ $\qquad u_6 \leq u_4 - u_1,\ u_6 \leq u_5 - u_2$

We derive the final program P_3 which is made out of clauses 3 and 6 together with the set of constraints $\mathscr{C}_3$ which is $\mathscr{C}_2 \cup \{u_6 \leq u_4 - u_1,\ u_6 \leq u_5 - u_2\}$. The correctness constraint system $\mathscr{C}_{final}$ associated with the transformation sequence $P_0 \mapsto \cdots \mapsto P_3$ is obtained by adding to $\mathscr{C}_3$ the constraints $u_3 \geq 1$ and $u_6 \geq 1$ corresponding to clauses 3 and 6, respectively. Thus, $\mathscr{C}_{final}$ consists of the following constraints:

$u_3 \geq 1, \quad u_6 \geq 1, \quad u_4 = u_1 + u_1, \quad u_5 = u_1 + u_2, \quad u_5 \leq u_2, \quad u_6 \leq u_4 - u_1, \quad u_6 \leq u_5 - u_2.$

The transformation sequence $P_0 \mapsto \cdots \mapsto P_3$ is not totally correct, because $M(P_0) = \{p, q\}$ and $M(P_3) = \{q\}$ and, indeed, $\mathscr{C}_{final}$ is unsatisfiable (because $u_5 \leq u_2$ implies $u_6 \leq 0$).

Note that, however, the unsatisfiability of the system of constraints associated with a transformation sequence does not imply that the sequence is not totally correct. Indeed, due to the undecidability of program equivalence, our method is incomplete and we can find examples of totally correct transformation sequences whose associated system of constraints is unsatisfiable.

4. Total Correctness of Unfold/Fold Transformations

In order to prove Theorem 3.1, that is, the total correctness of a given transformation sequence $P_0 \mapsto \cdots \mapsto P_n$ constructed by using Rules 1–4, we will use Theorem 2.11 of Section 2. We proceed as follows.

For $k = 0, \ldots, n$, we associate with program P_k a suitable weighted program $\overline{P}_k$, and we associate with $Defs_k$ a suitable weighted program $\overline{Defs}_k$. We assume that the correctness constraint system $\mathscr{C}_{final}$ of the given transformation sequence is satisfiable. Then, we prove the following properties:

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

(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.

Thus, the total correctness of the transformation sequence $P_0 \mapsto \cdots \mapsto P_n$ follows immediately from Properties (P1), (P2), and (P3), and from Theorem 2.11. 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 $\mathscr{C}_{\mathit{final}}$ associated with the given transformation sequence is satisfiable.

Let P be a program consisting of clauses $C_1, \ldots, C_r$ and let γ be a weight function for P. Given a valuation σ, for $i = 1, \ldots, r$, we denote by $\overline{C}_i[\sigma]$ the weighted clause $\overline{C}_i[\sigma(\gamma(C_i))]$. By $\overline{P}[\sigma]$ we denote the weighted program $\{\overline{C}_1[\sigma], \ldots, \overline{C}_r[\sigma]\}$. For example, given the clause C: $p \leftarrow q$ such that $\gamma(C) = u$ and $\sigma(u) = 2$, we have that $\overline{C}[\sigma]$ is the weighted clause $p(N_0) \leftarrow N_0 \geq N_1 + 2 \ \wedge \ q(N_1)$.

Now, let σ be a solution of $\mathscr{C}_{\mathit{final}}$. For $k = 0, \ldots, n$ and for every clause C occurring either in P_k or in Defs_k, we take $\overline{C} = \overline{C}[\sigma]$ (where $\overline{C}[\sigma]$ is constructed by using the weight function γ_k, if $C \in P_k$ and, otherwise, if $C \in \mathit{Defs}_k$, the weight function δ_k). Thus, $\overline{P}_k = \overline{P}_k[\sigma]$ and $\overline{\mathit{Defs}}_k = \overline{\mathit{Defs}}_k[\sigma]$.

In order to prove Properties (P1) and (P2) we need the following three lemmata, whose proofs are in Appendix A.

Lemma 4.1. *Let $P_0 \mapsto \cdots \mapsto P_n$ be a transformation sequence and let $1 \leq k < n$. Let C be a clause in P_k, let $C_1, \ldots, C_m$ be clauses in $P_0 \cup \mathit{Defs}_k$, and let $D_1, \ldots, D_m$ be the clauses in P_{k+1} derived by unfolding C with respect to an atom in its body using $C_1, \ldots, C_m$, 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 4.2. *Let $P_0 \mapsto \cdots \mapsto P_n$ be a transformation sequence and let $1 \leq k < n$. Let $C_1, \ldots, C_m$ be clauses in P_k, let $D_1, \ldots, D_m$ be clauses in $P_0 \cup \mathit{Defs}_k$, and let E be the clause in P_{k+1} 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) $M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models \{\overline{C}_1, \ldots, \overline{C}_m\} \Leftarrow \{\overline{E}\}$

Lemma 4.3. *Let $P_0 \mapsto \cdots \mapsto P_n$ be a transformation sequence and let $1 \leq k < n$. Let C and D be clauses in P_k such that C is subsumed by D. Then*:

(i) $M(P_0 \cup \mathit{Defs}_n) \models \{C, D\} \Rightarrow \{D\}$

(ii) $M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models \{\overline{C}, \overline{D}\} \Leftarrow \{\overline{D}\}$

We are now ready to prove Theorem 3.1.

Proof. [Proof of Theorem 3.1.] For a transformation sequence $P_0 \mapsto \cdots \mapsto P_n$, the following properties hold for $k = 0, \ldots, n-1$:

(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.

(*Case* 1) 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) trivially hold.

(*Case* 2) 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 $Defs_k = Defs_{k+1}$. Hence, Properties (R1) and (R2) follow from Lemma 2.8, Lemma 4.1 and from the monotonicity of $\Rightarrow$.
(*Case* 3) 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 $Defs_k = Defs_{k+1}$. Hence, Properties (R1) and (R2) follow from Points (i) and (ii), respectively, of Lemma 4.2 and from the monotonicity of $\Rightarrow$.
(*Case* 4) If P_{k+1} is derived from P_k by applying the deletion rule, then $P_{k+1} = P_k - \{C\}$ and $Defs_k = Defs_{k+1}$. Hence, Properties (R1) and (R2) follow from Points (i) and (ii), respectively, of Lemma 4.3 and from 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 of $\mathscr{C}_{final}$ and $\overline{P}_n = \overline{P}_n[\sigma]$, every clause in $\overline{P}_n$ has a positive weight and, hence, Property (P3) holds. Thus, by Theorem 2.11, $M(P_0 \cup Defs_n) = M(P_n)$. ∎

5. Goal Replacement

In this section we extend the notion of a transformation sequence $P_0 \mapsto P_1 \mapsto \cdots \mapsto P_n$ by assuming that P_{k+1} can be derived from P_k by applying, besides the definition introduction, unfolding, folding, and deletion rules, also the goal replacement rule as defined by Rule 5 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 $M(P_0 \cup Defs_k)$. The transformation sequences obtained by the goal replacement rule are guaranteed to be partially correct, and in order to guarantee their total correctness, we will require some extra conditions on the goals G_1 and G_2 to be satisfied in $M(\overline{P}_0 \cup \overline{Defs}_k)$. To define these conditions we will introduce the notion of a *replacement law* (see Definition 5.1).

For reasons of simplicity, we will define the replacement laws in the case where G_1 and G_2 are *atomic* goals of the form $p_1(X)$ and $p_2(X)$, respectively, where X is a tuple of variables. The general case where we want to replace non-atomic goals, can be treated by using, besides atomic goal replacement, also the definition introduction, folding, and unfolding rules. We will see this technique in action in Examples 5 and 6 below.

In the following Definition 5.1 we introduce the notion of an replacement law which will be used in the goal replacement rule (see Rule 5).

Definition 5.1 (Atomic Replacement Law) Let P be a program and $\mathscr{C}$ be a finite set of constraints. Let $p_1(X)$ and $p_2(X)$ be atoms such that: (i) p_1 and p_2 occur in P, (ii) X is a tuple of variables. We say that the *atomic replacement law* (or *replacement law*, for short) $p_1(X) \Rrightarrow p_2(X)$ holds in $\overline{P}[\mathscr{C}]$, and in this case we write $\overline{P}[\mathscr{C}] \models p_1(X) \Rrightarrow p_2(X)$, if the following conditions hold:

(i) $M(P) \models \forall X\, (p_1(X) \leftarrow p_2(X))$, and

(ii) for every solution σ of $\mathscr{C}$, $M(\overline{P}[\sigma]) \models \forall X \forall N\, (p_1(X, N) \rightarrow p_2(X, N))$.

By using Lemma 2.8 one can show that, if $\mathscr{C}$ is satisfiable, then Condition (ii) of Definition 5.1 implies $M(P) \models \forall X\, (p_1(X) \rightarrow p_2(X))$ and, therefore, if $\mathscr{C}$ is satisfiable and $\overline{P}[\mathscr{C}] \models p_1(X) \Rrightarrow p_2(X)$, we have that $M(P) \models \forall X\, (p_1(X) \leftrightarrow p_2(X))$, which is one of the usual conditions for the applicability of the goal replacement rule [TaS84]. Furthermore, a replacement law also establishes a relationship between the weights of the proofs of $p_1(X)$ and $p_2(X)$ (we have informally introduced the notion of the weight of a proof in Section 2 above). Thus, when

writing $\overline{P}[\mathscr{C}] \models p_1(X) \Rrightarrow p_2(X)$, we mean that given any weight assignment to the clauses of P which is a solution of the set $\mathscr{C}$ of constraints and, given any ground term t, if $p_1(t)$ has a proof of weight less than or equal to n, then $p_2(t)$ has a proof of weight less than or equal to n.

Now we give an example of a replacement law.

Example 5. (*Associativity of List Concatenation*) Let us consider the following program for list concatenation.

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

The associativity of list concatenation can be expressed as follows. Let us first introduce the following two clauses:

3. $lassoc(L_1, L_2, L_3, L) \leftarrow app(L_1, L_2, M) \wedge app(M, L_3, L)$
4. $rassoc(L_1, L_2, L_3, L) \leftarrow app(L_2, L_3, R) \wedge app(L_1, R, L)$

Then, associativity can be written as the following replacement law:

Law (α): $lassoc(L_1, L_2, L_3, L) \Rrightarrow rassoc(L_1, L_2, L_3, L)$

Let *Append* be the program consisting of clauses 1, 2, 3, and 4. In Example 7 below we will show that Law (α) holds with respect to $\overline{Append}[\mathscr{C}]$, where $\mathscr{C}$ is the set of constraints $\{u_1 \geq 1, u_2 \geq 1, u_3 \geq u_4\}$ and, for $i = 1, \ldots, 4$, u_i is the unknown associated with clause i.

In Section 6 we will present a method, called *weighted unfold/fold proof method*, whose objective is to generate, for any given pair of atoms $p_1(X)$ and $p_2(X)$, a suitable set $\mathscr{C}$ of constraints such that $\overline{P}[\mathscr{C}] \models p_1(X) \Rrightarrow p_2(X)$ holds.

Now we introduce the atomic goal replacement rule based on a replacement law. This rule is a variant of the usual goal replacement rule (see, for instance, [TaS84]).

Rule 5 (Atomic Goal Replacement) Let C: $H \leftarrow G_L \wedge p_1(t) \wedge G_R$ be a clause in program P_k and let $\mathscr{C}$ be a set of constraints such that the replacement law $\lambda : p_1(X) \Rrightarrow p_2(X)$ holds in $(\overline{P}_0 \cup \overline{Defs}_k)[\mathscr{C}]$.
By applying the replacement law λ, from C we derive $D : H \leftarrow G_L \wedge p_2(t) \wedge G_R$, and from P_k we derive by *atomic goal replacement* (or *goal replacement*, for short) $P_{k+1} = (P_k - \{C\}) \cup \{D\}$. We define:

$\mathscr{C}_{k+1} = \mathscr{C}_k \cup \mathscr{C} \cup \{\gamma_{k+1}(D) \leq \gamma_k(C)\}$.

The following Lemma 5.2, whose proof is given in Appendix A, is analogous to Lemmata 4.1, 4.2, and 4.3 proved for the unfolding, folding, and deletion rules, respectively. This lemma will be used to prove the total correctness of any transformation sequence constructed by using Rule 5, besides Rules 1–4.

Similarly to Section 4, given any transformation sequence $P_0 \mapsto \cdots \mapsto P_n$ constructed by using Rules 1–5, for $k = 0, \ldots, n$, we denote by $\overline{P}_k$ and $\overline{Defs}_k$ the weighted programs $\overline{P}_k[\sigma]$ and $\overline{Defs}_k[\sigma]$, respectively, where σ is any solution of $\mathscr{C}_{final}$.

Lemma 5.2. *Let $P_0 \mapsto \cdots \mapsto P_n$ be a transformation sequence and let $1 \leq k < n$. Let C be a clause in P_k and D be the clause in P_{k+1} derived by applying a replacement law λ that holds in $(\overline{P}_0 \cup \overline{Defs}_k)[\mathscr{C}]$, as described in Rule 5. Then*:

(i) $M(P_0 \cup Defs_n) \models \{C\} \Rightarrow \{D\}$, *and*

(ii) $M(\overline{P}_0 \cup \overline{Defs}_n) \models \{\overline{C}\} \Leftarrow \{\overline{D}\}$

Now we are ready to prove the total correctness of the transformation sequences constructed by using Rules 1–5.

Theorem 5.3 (Total Correctness of Unfold/Fold/Replacement Transformations) *Let $P_0 \mapsto \cdots \mapsto P_n$ be a transformation sequence constructed by using Rules 1–5, and let $\mathscr{C}_{final}$ be its associated correctness constraint system. If $\mathscr{C}_{final}$ is satisfiable then $M(P_0 \cup Defs_n) = M(P_n)$.*

Proof. The proof of this theorem is like the one of Theorem 3.1, except that, when we make the proofs of Properties (R1) and (R2) by cases on the transformation rule applied to derive P_{k+1} from P_k, we have to consider also the following alternative case.
(*Case* 5) If P_{k+1} is derived from P_k by applying the goal replacement rule, then $P_{k+1} = (P_k - \{C\}) \cup \{D\}$ and $Defs_k = Defs_{k+1}$. Hence, Properties (R1) and (R2) follow from Points (i) and (ii), respectively, of Lemma 5.2 and from the monotonicity of $\Rightarrow$. ∎

Example 6. (*List Reversal*) Let *Reverse* be a program for list reversal consisting of the clauses of *Append* (see clauses 1–4 in Example 5) together with the following two clauses:

5. $rev([\,],[\,]) \leftarrow$
6. $rev([H|T],L) \leftarrow rev(T,R) \wedge app(R,[H],L)$

We will transform the *Reverse* program into a program that uses an accumulator [BuD77]. In order to do so, by Rule 1 we introduce the following clause:

7. $g(L_1,L_2,A) \leftarrow rev(L_1,R) \wedge app(R,A,L_2)$

We apply the unfolding rule twice starting from clause 7 and we get (recall that we write to the right of a clause the constraints on the unknown associated with the clause):

8. $g([\,],L,L) \leftarrow$ $\qquad u_8 = u_7 + u_5 + u_1$
9. $g([H|T],L,A) \leftarrow rev(T,R) \wedge app(R,[H],S) \wedge app(S,A,L)$ $\qquad u_9 = u_7 + u_6$

Now, in order to apply the associativity of concatenation, we apply the folding, goal replacement, and unfolding rules as follows. By folding clause 9 using clause 3 we derive:

10. $g([H|T],L,A) \leftarrow rev(T,R) \wedge lassoc(R,[H],A,L)$ $\qquad u_{10} \leq u_9 - u_3$

Then, by applying the replacement law (α), from clause 10 we derive:

11. $g([H|T],L,A) \leftarrow rev(T,R) \wedge rassoc(R,[H],A,L)$ $\qquad u_{11} \leq u_{10}$

and we also add the constraints: $\{u_1 \geq 1, u_2 \geq 1, u_3 \geq u_4\}$ (see Example 5). Next, by unfolding clause 11 we derive:

12. $g([H|T],L,A) \leftarrow rev(T,R) \wedge app([H],A,S) \wedge app(R,S,L)$ $\qquad u_{12} = u_{11} + u_4$

Note that, the effect of the last three transformation step is the replacement of the non-atomic goal $app(R,[H],S) \wedge app(S,A,L)$ by the non-atomic goal $app([H],A,S) \wedge app(R,S,L)$. Now, by two applications of the unfolding rule, from clause 12 we get:

13. $g([H|T],L,A) \leftarrow rev(T,R) \wedge app(R,[H|A],L)$ $\qquad u_{13} = u_{12} + u_2 + u_1$

By folding clause 13 using clause 7 we get:

14. $g([H|T],L,A) \leftarrow g(T,L,[H|A])$ $\qquad u_{14} \leq u_{13} - u_7$

Finally, by folding clause 6 using clause 7 we get:

15. $rev([H|T],L) \leftarrow g(T,L,[H])$ $\qquad u_{15} \leq u_6 - u_7$

The final program consists of the following clauses:

5. $rev([\,],[\,]) \leftarrow$

15. $rev([H|T], L) \leftarrow g(T, L, [H])$
8. $g([\,], L, L) \leftarrow$
14. $g([H|T], L, A) \leftarrow g(T, L, [H|A])$

together with clauses 1 and 2 for *append* of Example 5.

The correctness constraint system associated with the transformation sequence is as follows.
For clauses 1, 2, 3, and 4: $u_1 \geq 1, \quad u_2 \geq 1, \quad u_3 \geq 1, \quad u_4 \geq 1.$
For clauses 5, 15, 8, and 14: $u_5 \geq 1, \quad u_{15} \geq 1, \quad u_8 \geq 1, \quad u_{14} \geq 1.$
For the unfolding steps: $u_8 = u_7 + u_5 + u_1, \quad u_9 = u_7 + u_6, \quad u_{12} = u_{11} + u_4, \quad u_{13} = u_{12} + u_2 + u_1.$
For the replacement: $u_1 \geq 1, \quad u_2 \geq 1, \quad u_3 \geq u_4, \quad u_{11} \leq u_{10}.$
For the folding steps: $u_{10} \leq u_9 - u_3, \quad u_{14} \leq u_{13} - u_7, \quad u_{15} \leq u_6 - u_7.$
This set of constraints is satisfiable and, therefore, the transformation sequence is totally correct.

6. The Weighted Unfold/Fold Proof Method

In this section we present a method for proving the replacement laws to be used in Rule 5. By following the approach of [Kot82, PeP99], this method is itself based on the application of the transformation rules of Sections 3 and 5 which use weights and, for this reason, it is called the *weighted unfold/fold proof method*.

Before introducing the weighted unfold/fold proof method, let us briefly recall how the unfold/fold proof method works in the case where weights are not present. Suppose that we want to prove that two atoms $p_1(X)$ and $p_2(X)$ are equivalent in the least Herbrand model $M(P)$ of a program P, that is, $M(P) \models \forall X \ (p_1(X) \leftrightarrow p_2(X))$.

Without loss of generality, we assume that P is of the form $T \cup \{D_1, D_2\}$, where D_1 and D_2 are clauses of the form:

D_1. $p_1(X) \leftarrow G_1$
D_2. $p_2(X) \leftarrow G_2$

and the predicate symbols p_1 and p_2 occur in P in the head of D_1 and D_2 only. (Indeed, if this is not the case, we can always introduce two new clauses $newp_1(X) \leftarrow p_1(X)$ and $newp_2(X) \leftarrow p_2(X)$, and then prove $M(P) \models \forall X \ (newp_1(X) \leftrightarrow newp_2(X))$.)
Then, by two *totally correct* transformation sequences, from $T \cup \{D_1\}$ and $T \cup \{D_2\}$ we derive two new programs Q_1 and Q_2 which are *syntactically equivalent*, that is, they are equal modulo predicate and variable renaming.

As it stands, the unfold/fold proof method is not able to prove that the replacement law $p_1(X) \Rrightarrow p_2(X)$ holds in $\overline{P}$, as we need to show condition (ii) of Definition 5.1 which is stronger than goal equivalence. Indeed, as remarked immediately after Definition 5.1, we also need to show suitable relationships between the weights of the proofs for (ground instances of) $p_1(X)$ and $p_2(X)$. The weighted unfold/fold proof method is an extension of the unfold/fold proof method in that it shows the equivalence of $p_1(X)$ and $p_2(X)$, and also it establishes the required relationships between their proofs, under suitable restrictions on the applications of the folding and goal replacement rules.

In order to present the weighted unfold/fold proof method, now we introduce the notions of: (i) *syntactic equivalence*, (ii) *symmetric folding*, and (iii) *symmetric goal replacement*.

A *predicate renaming* is a bijective mapping $\rho : Preds_1 \rightarrow Preds_2$, where $Preds_1$ and $Preds_2$ are two sets of predicate symbols. Given a program P, by $preds(P)$ we denote the set of predicate symbols occurring in P. Suppose that $preds(P) = Preds_1$, then by $\rho(P)$ we denote the program obtained from P by replacing every predicate symbol p by $\rho(p)$.

Definition 6.1 (Syntactic Equivalence) *Two programs Q and R are* syntactically equivalent *if there exists a predicate renaming $\rho : preds(Q) \to preds(R)$, such that $R = \rho(Q)$, modulo variable renaming.*

Syntactic equivalence implies semantic equivalence, as stated by the following lemma, whose proof is straightforward.

Lemma 6.2. *If a program Q is syntactically equivalent to a program R via a predicate renaming ρ, then, for every predicate p occurring in Q and tuple t of ground terms, $p(t) \in M(Q)$ iff $\rho(p)(t) \in M(R)$.*

Definition 6.3 (Symmetric Folding) *An application of the folding rule is said to be* symmetric *if*

$\mathscr{C}_{k+1} = \mathscr{C}_k \cup \{\gamma_{k+1}(E) = \gamma_k(C_1) - \delta_k(D_1), \ldots, \gamma_{k+1}(E) = \gamma_k(C_m) - \delta_k(D_m)\}$.

Definition 6.4 (Symmetric Replacement Law) *Given a program P and a set $\mathscr{C}$ of constraints, a replacement law $p_1(X) \Rrightarrow p_2(X)$ which holds in $\overline{P}[\mathscr{C}]$, is said to be* symmetric*, and we write $\overline{P}[\mathscr{C}] \models p_1(X) \Lleftrightarrow p_2(X)$, if the following condition holds:*
(ii) for every solution σ of $\mathscr{C}$, $M(\overline{P}[\sigma]) \models \forall X \forall N\, (p_1(X, N) \leftrightarrow p_2(X, N))$.*

Note that, by Lemma 2.8, Condition (ii*) implies Conditions (i) and (ii) of Definition 5.1.

Definition 6.5 (Symmetric Goal Replacement) *An application of the goal replacement rule is said to be* symmetric *if (i) it consists in applying a symmetric replacement law, and (ii) $\mathscr{C}_{k+1} = \mathscr{C}_k \cup \mathscr{C} \cup \{\gamma_{k+1}(D) = \gamma_k(C)\}$.*

A 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 (thus, no applications of the deletion rule occur in a symmetric transformation sequence).

If the correctness constraint system associated with a symmetric transformation sequence is satisfiable, then both total correctness is guaranteed (by Theorem 5.3) and also the following result holds (its proof is in Appendix A).

Theorem 6.6 (Strong Correctness of Symmetric Transformations) *Let $P_0 \mapsto \cdots \mapsto P_n$ be a symmetric transformation sequence and let $\mathscr{C}_{final}$ be its associated correctness constraint system. If $\mathscr{C}_{final}$ is satisfiable, then $M(\overline{P}_0 \cup \overline{Defs}_n) = M(\overline{P}_n)$.*

Now we are ready to present the weighted unfold/fold proof method.

(A) We construct two transformation sequences of the form:
$T \cup \{D_1\} \mapsto \cdots \mapsto Q$ and $T \cup \{D_2\} \mapsto \cdots \mapsto R$, such that the following three conditions hold:
(1) the correctness constraint systems $\mathscr{C}_Q$ and $\mathscr{C}_R$ associated with the transformation sequences $T \cup \{D_1\} \mapsto \cdots \mapsto Q$ and $T \cup \{D_2\} \mapsto \cdots \mapsto R$, respectively, are satisfiable;
(2) there exists a predicate renaming ρ such that $\rho(p_1) = p_2$ and $\rho(Q) = R$; and
(3) the transformation sequence $T \cup \{D_2\} \mapsto \cdots \mapsto R$ is symmetric.

(B.1) Then, we consider the weight functions γ_Q and γ_R for the programs Q and R, respectively, and we state, by definition, that the following relation holds:

$\overline{P}[\mathscr{C}] \vdash_{UF} p_1(X) \Rrightarrow p_2(X)$
where $\mathscr{C}$ is the set $\{\gamma_Q(C) \geq \gamma_R(\rho(C)) \mid C \in Q\} \cup \mathscr{C}_Q \cup \mathscr{C}_R$ of constraints.

(B.2) Moreover, if the transformation sequence $T \cup \{D_1\} \mapsto \cdots \mapsto Q$ is symmetric, we state, by definition, that the following relation holds:
$\overline{P}[\mathscr{C}] \vdash_{UF} p_1(X) \Lleftrightarrow p_2(X)$
where $\mathscr{C}$ is the set $\{\gamma_Q(C) = \gamma_R(\rho(C)) \mid C \in Q\} \cup \mathscr{C}_Q \cup \mathscr{C}_R$ of constraints.

The following result, whose proof is given in Appendix A, ensures the soundness of the weighted unfold/fold proof method.

Theorem 6.7 (Soundness of the Weighted Unfold/Fold Proof Method) *We have that:*
(i) if $\overline{P}[\mathscr{C}] \vdash_{UF} p_1(X) \Rrightarrow p_2(X)$ *then* $\overline{P}[\mathscr{C}] \models p_1(X) \Rrightarrow p_2(X)$, *and*
(ii) if $\overline{P}[\mathscr{C}] \vdash_{UF} p_1(X) \Lleftrightarrow p_2(X)$ *then* $\overline{P}[\mathscr{C}] \models p_1(X) \Lleftrightarrow p_2(X)$.

Example 7. (*Proving a Replacement Law Using the Weighted Unfold/Fold Method*) Let us consider again the program *Append* and the replacement law (α), expressing the associativity of list concatenation, presented in Example 5. By applying the weighted unfold/fold proof method we will generate a set $\mathscr{C}$ of constraints such that law (α) holds in $Append[\mathscr{C}]$.

Let T be the program consisting of clauses 1 and 2 defining the predicate *app* for list concatenation, D_1 be clause 3 defining the predicate *lassoc* and D_2 be clause 4 defining the predicate *rassoc*. Thus, $Append = T \cup \{D_1, D_2\}$. Let us denote by u_1 and u_2 the unknowns associated with clauses 1 and 2, respectively, and by u_3 and u_4 the unknowns associated with D_1 and D_2, respectively.

Step 1. First, let us construct a transformation sequence starting from $T \cup \{D_1\}$. In this transformation sequence we denote by e_i the unknown associated with the clause E_i, for $i > 0$. By two applications of the unfolding rule, from clause D_1 we derive:

E_1. $lassoc([\,], L_2, L_3, L) \leftarrow app(L_2, L_3, L)$ $\qquad e_1 = u_3 + u_1$
E_2. $lassoc([H|T], L_2, L_3, [H|R]) \leftarrow app(T, L_2, M) \wedge app(M, L_3, R)$ $\qquad e_2 = u_3 + 2u_2$

By folding clause E_2 using clause D_1 we derive:

E_3. $lassoc([H|T], L_2, L_3, [H|R]) \leftarrow lassoc(T, L_2, L_3, R)$ $\qquad e_3 \leq e_2 - u_3$

Now, let us construct a transformation sequence starting from $T \cup \{D_2\}$. In this transformation sequence we denote by f_i the unknown associated with clause F_i, for $i > 0$. By unfolding clause D_2 w.r.t. $a(L_1, R, L)$ in its body we get:

F_1. $rassoc([\,], L_2, L_3, L) \leftarrow app(L_2, L_3, L)$ $\qquad f_1 = u_4 + u_1$
F_2. $rassoc([H|T], L_2, L_3, [H|R]) \leftarrow app(L_2, L_3, M) \wedge app(T, M, R)$ $\qquad f_2 = u_4 + u_2$

By symmetric folding using clause D_2, from clause F_2 we get:

F_3. $rassoc([H|T], L_2, L_3, [H|R]) \leftarrow rassoc(T, L_2, L_3, R)$ $\qquad f_3 = f_2 - u_4$

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

Step 2. The correctness constraint system associated with the transformation sequence $T \cup \{D_1\} \mapsto \cdots \mapsto T \cup \{E_1, E_3\}$ is the following:

$\mathscr{C}_1$: $\{u_1 \geq 1,\ u_2 \geq 1,\ e_1 \geq 1,\ e_3 \geq 1,\ e_1 = u_3 + u_1,\ e_2 = u_3 + 2u_2,\ e_3 \leq e_2 - u_3\}$

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:

$\mathscr{C}_2$: $\{u_1 \geq 1,\ u_2 \geq 1,\ f_1 \geq 1,\ f_3 \geq 1,\ f_1 = u_4 + u_1,\ f_2 = u_4 + u_2,\ f_3 = f_2 - u_4\}$

Both $\mathscr{C}_1$ and $\mathscr{C}_2$ are satisfiable. Let $\mathscr{C}_{12}$ be the set $\{e_1 \geq f_1,\ e_3 \geq f_3\} \cup \mathscr{C}_1 \cup \mathscr{C}_2$. In $\mathscr{C}_{12}$ the constraints $e_1 \geq f_1$ and $e_3 \geq f_3$ are determined by the two pairs of syntactically equivalent clauses (E_1, F_1) and (E_3, F_3), respectively. By eliminating from $\mathscr{C}_{12}$ all unknowns different from u_1, u_2, u_3, and u_4, we obtain the set of constraints $\mathscr{C} = \{u_1 \geq 1, u_2 \geq 1, u_3 \geq u_4\}$, and we get:

$Append[\mathscr{C}] \vdash_{UF} lassoc(L_1, L_2, L_3, L) \Rrightarrow rassoc(L_2, L_3, L_1, L)$.

7. Related work and Conclusions

In this paper which is based upon the results presented in [PPS07, PeP08], 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 applying 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. We also construct a set of linear constraints that these weights must satisfy to guarantee the total correctness of that 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. It can be shown that our method is incomplete and indeed, in general, there is no algorithm 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 various methods presented in the literature for proving the correctness of program transformations by showing that suitable conditions on the transformation sequences hold. For the case of *definite* logic programs, that is, the case of clauses without negative literals in the premise, the reader may refer, for instance, to [BCE92, GeK94, KaF86, RKR04, TaS84, TaS86]. Among these methods, the one presented in [RKR04] is the most general one and it makes use of *clause measures* to express complex conditions on the transformation sequences. The main novelty of our method with respect to [RKR04] is that in [RKR04] clause measures are fixed in advance, independently of the specific transformation sequence which is performed, while the method proposed in this paper allows us to automatically generate specific clause measures for each transformation sequence to be proved correct.

Thus, our method is more flexible than the one presented in [RKR04]. For instance, we checked that the transformation sequence presented in our Example 3 cannot be proved correct by using the default clause measures provided by the SCOUT transformation system that implements the method presented in [RKR04]. (In Appendix B we also present a simpler example which cannot be worked out by SCOUT and, instead, can be easily dealt with by following our approach.)

In order to check the power of our constraint-based method, and compare it with other 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. 3.12.8) and for the satisfiability of sets of constraints over nonnegative integers it uses a procedure for integer programming provided by the *clpq* SICStus library.

By using our system we did the transformation examples presented in this paper (see Examples 3, 6, and 7) 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 [KaF86], (ii) the *Equal Frontiers* program, which checks whether or not the frontiers of two binary trees

are equal [BuD77, TaS86], (iii) a program for solving the N-queens problem [SaT85], (iv) the *In_Correct_Position* program taken from [GeK94], and (v) the program that encodes a liveness property of an n-bit shift register [RKR04]. Even in the most complex derivation we carried out, that is, the *Equal Frontiers* example, consisting of 97 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 weighted unfold/fold proof method described in Section 6.

Our approach can be extended in several ways. First of all, one may take into consideration *general* logic programs, that is, logic programs with negation. Some work has been recently done on the correctness of unfold/fold transformations of general logic programs with the *stable model* and the *perfect model* semantics [Sek09, Sek10]. Similarly to what we have done here for the case of definite logic programs, we can extend our constraint-based approach to general logic programs with various semantics of negation.

Another recent area of research is the extension of the unfold/fold transformation method to logic programs defined on the domain of finite and *infinite* structures. In this area we mention two papers: (i) the [PPS10] paper, which presents correctness results of some transformation rules for *locally stratified* general programs whose semantics is an extension of the perfect model, and (ii) the [Sek11] paper, which shows the correctness of a set of transformation rules for *coinductive* logic programs, that is, logic programs whose semantics is defined by means of greatest fixpoints, besides the usual least fixpoints [SMB06]. We leave it to future studies the extension of our constraint-based approach to logic programs on infinite structures. This extension is challenging, because the notion of termination we have considered in this paper, should be modified for dealing with the case of infinite computations.

Acknowledgements

This work has been partially supported by MIUR PRIN 2008 n.20089M932N (Project of the Italian Ministry of Education). We thank John Gallagher and Hirohisa Seki for stimulating discussions on the issues addressed in this paper. We also thank Fabio Fioravanti for his contribution in the implementation of the MAP transformation system. Many thanks to the editors Peter Höfner and Robert van Glabbeek, who invited us to write this paper in honor of Carroll Morgan.

References

[Apt90] K. R. Apt: Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–576. Elsevier, 1990.

[BoC94] A. Bossi and N. Cocco: Preserving universal termination through unfold/fold. In *Proceedings ALP '94*, Lecture Notes in Computer Science 850, pages 269–286, Berlin, Springer-Verlag, 1994.

[BCE92] 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.

[BuD77] R. M. Burstall and J. Darlington: A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.

[Bez89] 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.

[CoG94] J. Cook and J. P. Gallagher: A transformation system for definite programs based on termination analysis. In L. Fribourg and F. Turini, editors, *Proceedings of LOPSTR '94 and META '94, Pisa, Italy*, Lecture Notes in Computer Science 883, pages 51–68. Springer-Verlag, 1994.

[Dix95] J. Dix: A classification theory of semantics of normal logic programs: II weak properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.

[EtG96] S. Etalle and M. Gabbrielli: Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.

[GeK94] 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.

[JaM94] J. Jaffar and M. Maher: Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[KaF86] T. Kanamori and H. Fujita: Unfold/fold transformation of logic programs with counters. Technical Report 179, ICOT, Tokyo, Japan, 1986.

[Kot78] L. Kott: About transformation system: A theoretical study. In *3ème Colloque International sur la Programmation*, pages 232–247, Paris (France), Dunod, 1978.

[Kot82] L. Kott: The McCarthy's induction principle: 'oldy' but 'goody'. *Calcolo*, 19(1):59–69, 1982.

[Llo87] J. W. Lloyd: *Foundations of Logic Programming.* Second Edition. Springer-Verlag, Berlin, 1987.

[LOP95] 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 the 1995 International Logic Programming Symposium (ILPS '95)*, pages 480–494. MIT Press, 1995.

[Mah87] M. J. Maher: Correctness of a logic program transformation system. IBM Research Report RC 13496, T. J. Watson Research Center, 1987.

[Mah93] M. J. Maher: A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science*, 110:377–403, 1993.

[McC63] J. McCarthy: Towards a mathematical science of computation. In C.M. Popplewell, editor, *Information Processing: Proceedings of IFIP 1962*, pages 21–28, Amsterdam, North Holland, 1963.

[PeP99] A. Pettorossi and M. Proietti: Synthesis and transformation of logic programs using unfold/fold proofs. *Journal of Logic Programming*, 41(2&3):197–230, 1999.

[PeP08] A. Pettorossi and M. Proietti: Totally correct logic program transformations via well-founded annotations. *Higher-Order and Symbolic Computation*, 21:193–234, 2008.

[PPS07] A. Pettorossi, M. Proietti, and V. Senni: Automatic correctness proofs for logic program transformations. In V. Dahl and I. Niemelä, editors, *Proceedings of the 23rd International Conference on Logic Programming (ICLP '07)*, Lecture Notes in Computer Science 4670, pages 364–379, 2007.

[PPS10] A. Pettorossi, M. Proietti, and V. Senni: Transformations of logic programs on infinite lists. *Theory and Practice of Logic Programming, Special Issue on the 26th International Conference on Logic Programming (ICLP '10), Edinburgh, Scotland, UK*, 10(4–6):383–399, 2010.

[RKR02] A. Roychoudhury, K. Narayan Kumar, C. R. Ramakrishnan, and I. V. Ramakrishnan: Beyond Tamaki-Sato style unfold/fold transformations for normal logic programs. *International Journal on Foundations of Computer Science*, 13(3):387–403, 2002.

[RKR04] 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.

[San96] D. Sands: Total correctness by local improvement in the transformation of functional programs. *ACM Toplas*, 18(2):175–234, 1996.

[Sek91] H. Seki: Unfold/fold transformation of stratified programs. *Theoretical Computer Science*, 86:107–139, 1991.

[Sek09] H. Seki: On negative unfolding in the answer set semantics. In Michael Hanus, editor, *Logic-Based Program Synthesis and Transformation, 18th International Symposium, LOPSTR '08, Valencia, Spain, July 17-18, 2008, Revised Selected Papers*, Lecture Notes in Computer Science 5438, pages 168–184. Springer, 2009.

[Sek10] H. Seki: On inductive and coinductive proofs via unfold/fold transformations. In D. De Schreye, editor, *Proceedings of the 19th International Symposium on Logic Based Program Synthesis and Transformation (LOPSTR '09), Coimbra, Portugal, September 9-11, 2009*, Lecture Notes in Computer Science 6037, pages 82–96. Springer, 2010.

[Sek11] H. Seki: Proving properties of co-logic programs by unfold/fold transformations. In G. Vidal, editor, *Preliminary Proceedings of the 21th International Symposium on Logic-Based Synthesis and Transformation (LOPSTR '11), July 18-20, 2011, Odense, Denmark*, pages 112–126. University of Southern Denmark, 2011.

[SMB06] L. Simon, A. Mallya, A. Bansal, and G. Gupta: Coinductive logic programming. In S. Etalle and M. Truszczyński, editors, *22nd International Conference on Logic Programming, ICLP '06, Seattle, WA, USA, August 17-20, 2006*, Lecture Notes in Computer Science 4079, pages 330–345. Springer, 2006.

[SaT85] T. Sato and H. Tamaki: Examples of logic program transformation and synthesis. Case studies of transformation and synthesis of logic programs done up to Feb. 85, 1985.

[TaS84] 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 (ICLP '84)*, pages 127–138, Uppsala, Sweden, 1984. Uppsala University.

[TaS86] H. Tamaki and T. Sato: A generalized correctness proof of the unfold/fold logic program transformation. Technical Report 86-4, Ibaraki University, Japan, 1986.

[Wan80] M. Wand: Continuation-based program transformation strategies. *Journal of the ACM*, 27(1):164–180, 1980.

A. Proofs

Proofs for Section 4 (Total Correctness of Unfold/Fold)

Proof. [Proof of Lemma 4.1.] For reasons of simplicity we assume that: (i) C is of the form $p_0(t_0) \leftarrow p_1(t_1) \wedge p_2(t_2)$, (ii) C is unfolded w.r.t. $p_1(t_1)$, and (iii) for $i = 1, \ldots, m$, C_i is of the form $p_1(a_i) \leftarrow q_i(b_i)$. The extension to the general case where predicates have arbitrary arity, bodies of clauses have arbitrary numbers of atoms, and C is unfolded w.r.t. any atom in its body is straightforward. We have that: (iv) $\overline{C}$ is of the form

$$p_0(t_0, N_0) \leftarrow N_0 \geq N_1 + N_2 + w \wedge p_1(t_1, N_1) \wedge p_2(t_2, N_2)$$

where $w = \sigma(\gamma_k(C))$ (recall that σ is a solution of $\mathscr{C}_{\mathit{final}}$); (v) for $i = 1, \ldots, m$, $\overline{C}_i$ is of the form

$$p_1(a_i, M_i) \leftarrow M_i \geq Q_i + w_i \wedge q_i(b_i, Q_i)$$

where $w_i = \sigma(\delta_k(C_i))$; (vi) for $i = 1, \ldots, m$, D_i is of the form

$$p_0(t_0\vartheta_i) \leftarrow q_i(b_i\vartheta_i) \wedge p_2(t_2\vartheta_i)$$

where ϑ_i is a most general unifier of t_1 and a_i; and (vii) for $i = 1, \ldots, m$, $\overline{D}_i$ is of the form

$$p_0(t_0\vartheta_i, N_0) \leftarrow N_0 \geq Q_i + N_2 + z_i \wedge q_i(b_i\vartheta_i, Q_i) \wedge p_2(t_2\vartheta_i, N_2)$$

where $z_i = \sigma(\gamma_{k+1}(D_i))$. We also have that $z_i = w + w_i$. Indeed, by construction the equality $\gamma_{k+1}(D_i) = \gamma_k(C) + \delta_k(C_i)$ belongs to $\mathscr{C}_{k+1}$, hence it belongs to $\mathscr{C}_{\mathit{final}}$, and σ is a solution of $\mathscr{C}_{\mathit{final}}$.

Let us prove that $M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models \{\overline{C}\} \Rightarrow \{\overline{D}_1, \ldots, \overline{D}_m\}$. By Definition 2.1 we have to prove that, for $i = 1, \ldots, m$, for every ground instance $\overline{D}_i\psi_i$ of clause $\overline{D}_i$ such that $M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models \mathit{bd}(\overline{D}_i\psi_i)$, there exists a ground instance $\overline{C}\varphi_i$ of $\overline{C}$ such that $\mathit{hd}(\overline{C}\varphi_i) = \mathit{hd}(\overline{D}_i\psi_i)$ and $M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models \mathit{bd}(\overline{C}\varphi_i)$.

Let $\overline{D}_i\psi_i$ be a ground instance of $\overline{D}_i$ such that

$$M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models (N_0 \geq Q_i + N_2 + z_i \wedge q_i(b_i\vartheta_i, Q_i) \wedge p_2(t_2\vartheta_i, N_2))\psi_i$$

Since $z_i = w + w_i$, we have that there exists $m_i \in \mathbb{N}$ such that

$$M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models (m_i \geq Q_i + w_i \wedge q_i(b_i\vartheta_i, Q_i))\psi_i$$

and

$$M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models (N_0 \geq m_i + N_2 + w \wedge p_2(t_2\vartheta_i, N_2))\psi_i \qquad (\dagger 1)$$

Let $\overline{C}_i\vartheta_i\psi_i\tau_i$ be a ground instance of $\overline{C}_i$, where τ_i is a ground substitution for the variables in $\mathit{vars}(a_i) - \mathit{vars}(b_i)$. Since $(m_i \geq Q_i + w_i \wedge q_i(b_i\vartheta_i, Q_i))\psi_i$ is a ground goal, then it is equal to $(m_i \geq Q_i + w_i \wedge q_i(b_i\vartheta_i, Q_i))\psi_i\tau_i$ and, thus, we have that:

$$M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models (m_i \geq Q_i + w_i \wedge q_i(b_i\vartheta_i, Q_i))\psi_i\tau_i$$

The clause $\overline{C}_i$ belongs to $\overline{P}_0 \cup \overline{\mathit{Defs}}_k$ and, therefore, it belongs to $\overline{P}_0 \cup \overline{\mathit{Defs}}_n$. Thus, $\overline{C}_i\vartheta_i\psi_i\tau_i$ is true in $M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n)$ and we have that:

$$M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models p_1(a_i, m_i)\vartheta_i\psi_i\tau_i$$

Since ϑ_i is a unifier of t_1 and a_i, we have that $p_1(t_1, m_i)\vartheta_i\psi_i\tau_i = p_1(a_i, m_i)\vartheta_i\psi_i\tau_i$ and, thus,

$$M(\overline{P}_0 \cup \overline{\mathit{Defs}}_n) \models p_1(t_1, m_i)\vartheta_i\psi_i\tau_i \qquad (\dagger 2)$$

Let φ_i be the substitution $\vartheta_i\psi_i\tau_i$ and let us consider the clause $\overline{C}\varphi_i$. The atom $p_0(t_0, N_0)\vartheta_i\psi_i$ is ground and, therefore, we have:

$$\mathit{hd}(\overline{C}\varphi_i) = p_0(t_0, N_0)\vartheta_i\psi_i\tau_i = p(t_0, N_0)\vartheta_i\psi_i = \mathit{hd}(\overline{D}_i\psi_i)$$

Since $(N_0 \geq N_1+N_2+w \wedge p_2(t_2,N_2))\vartheta_i\psi_i$ is a ground goal and it is equal to $(N_0 \geq N_1+N_2+w)\psi_i \wedge p_2(t_2\vartheta_i,N_2)\psi_i$, we have: $bd(\overline{C}\varphi_i) = (N_0 \geq N_1+N_2+w \wedge p_1(t_1,N_1) \wedge p_2(t_2,N_2))\vartheta_i\psi_i\tau_i = (N_0 \geq N_1+N_2+w)\psi_i \wedge p_1(t_1,N_1)\vartheta_i\psi_i\tau_i \wedge p_2(t_2\vartheta_i,N_2)\psi_i$ and, by (†1) and (†2), we get:

$M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{C}\varphi_i)$.

Now we prove that $M(\overline{P}_0 \cup \overline{Defs}_n) \models \{\overline{C}\} \Leftarrow \{\overline{D}_1,\ldots,\overline{D}_m\}$. By Definition 2.1 we have to prove that, for every ground instance $\overline{C}\varphi$ of $\overline{C}$ such that $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{C}\varphi)$, for some $i \in \{1,\ldots,m\}$ there exists a ground instance $\overline{D}_i\psi_i$ of $\overline{D}_i$ such that $hd(\overline{D}_i\psi_i) = hd(\overline{C}\varphi)$ and $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{D}_i\psi_i)$.

Let $\overline{C}\varphi$ be a ground instance of $\overline{C}$ such that

$M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq N_1+N_2+w \wedge p_1(t_1,N_1) \wedge p_2(t_2,N_2))\varphi$ (†3)

Thus, we have that $M(\overline{P}_0 \cup \overline{Defs}_n) \models p_1(t_1,N_1)\varphi$ and, since $M(\overline{P}_0 \cup \overline{Defs}_n)$ is a fixpoint of $T_{\overline{P}_0 \cup \overline{Defs}_n}$, there exists a ground instance $\overline{E}$ of a clause in $\overline{P}_0 \cup \overline{Defs}_n$ such that $p_1(t_1,N_1)\varphi = hd(\overline{E})$ and $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{E})$. By the definition of Rule 1, no clause with head predicate p_1 occurs in $Defs_n - Defs_k$ and, thus, $\overline{E}$ is $\overline{C}_i\varphi_i$ for some clause $\overline{C}_i \in \overline{P}_0 \cup \overline{Defs}_k$ and some substitution φ_i, such that $p_1(t_1,N_1)\varphi = hd(\overline{C}_i\varphi_i)$ and $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{C}_i\varphi_i)$. Since $vars(\overline{C}) \cap vars(\overline{C}_i) = \emptyset$, we may assume that $\overline{C}_i\varphi = \overline{C}_i$. Hence, $p_1(t_1,N_1)\varphi\varphi_i = p_1(t_1,N_1)\varphi$ (because $p_1(t_1,N_1)\varphi$ is a ground atom) $= hd(\overline{C}_i\varphi_i) = hd(\overline{C}_i\varphi\varphi_i)$ (because $\overline{C}_i\varphi = \overline{C}_i$) $= p_1(a_i,M_i)\varphi\varphi_i$, that is, $\varphi\varphi_i$ is a ground unifier of $p_1(t_1,N_1)$ and $p_1(a_i,M_i)$. Since ϑ_i is a most general unifier of t_1 and a_i, it follows that $\varphi\varphi_i = \vartheta_i\psi_i$ for some ground substitution ψ_i. Moreover, $N_1\vartheta_i\psi_i = M_i\vartheta_i\psi_i$ and $\overline{D}_i\psi_i$ is a ground clause.

We complete the proof by showing that: (i) $hd(\overline{D}_i\psi_i) = hd(\overline{C}\varphi)$ and (ii) $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{D}_i\psi_i)$. Indeed, Point (i) holds because $hd(\overline{D}_i\psi_i) = p_0(t_0,N_0)\vartheta_i\psi_i = p_0(t_0,N_0)\varphi\varphi_i$ (because $\vartheta_i\psi_i = \varphi\varphi_i$) $= p_0(t_0,N_0)\varphi$ (because $p_0(t_0,N_0)\varphi$ is a ground atom) $= hd(\overline{C}\varphi)$. Point (ii) holds because the following properties (ii.a)–(ii.d) hold:

(ii.a) $bd(\overline{D}_i\psi_i) = (N_0 \geq Q_i+N_2+z_i \wedge q_i(b_i,Q_i) \wedge p_2(t_2,N_2))\vartheta_i\psi_i$;

(ii.b) $M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq N_1+N_2+w \wedge p_2(t_2,N_2))\vartheta_i\psi_i$,

indeed, from (†3) it follows that $M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq N_1+N_2+w \wedge p_2(t_2,N_2))\varphi$, and $(N_0 \geq N_1+N_2+w \wedge p_2(t_2,N_2))\varphi = (N_0 \geq N_1+N_2+w \wedge p_2(t_2,N_2))\varphi\varphi_i$ (because the goal $(N_0 \geq N_1+N_2+w \wedge p_2(t_2,N_2))\varphi$ is ground) $= (N_0 \geq N_1+N_2+w \wedge p_2(t_2,N_2))\vartheta_i\psi_i$ (because $\varphi\varphi_i = \vartheta_i\psi_i$);

(ii.c) $M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_1 \geq Q_i+w_i \wedge q_i(b_i,Q_i))\vartheta_i\psi_i$,

indeed, recalling that $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{C}_i\varphi_i)$ and that $bd(\overline{C}_i\varphi_i) = (M_i \geq Q_i+w_i \wedge q_i(b_i,Q_i))\varphi_i$, we have that

$M(\overline{P}_0 \cup \overline{Defs}_n) \models (M_i \geq Q_i+w_i \wedge q_i(b_i,Q_i))\varphi_i$ and

$(M_i \geq Q_i+w_i \wedge q_i(b_i,Q_i))\varphi_i$ $(M_i \geq Q_i+w_i \wedge q_i(b_i,Q_i))\varphi\varphi_i$ (because $\overline{C}_i\varphi = \overline{C}_i$)

$= (M_i \geq Q_i+w_i \wedge q_i(b_i,Q_i))\vartheta_i\psi_i$ (because $\varphi\varphi_i = \vartheta_i\psi_i$)

$= (N_1 \geq Q_i+w_i \wedge q_i(b_i,Q_i))\vartheta_i\psi_i$ (because $\vartheta_i\psi_i$ is a unifier of $p_1(t_1,N_1)$ and $p_1(a_i,M_i)$); and

(ii.d) $z_i = w + w_i$. ∎

Proof. [Proof of Lemma 4.2.] For reasons of simplicity we assume that: (1) for $i = 1,\ldots,m$, C_i is of the form $p_0(t_0) \leftarrow p_1(u_i) \wedge p_2(t_2)$, (2) for $i = 1,\ldots,m$, D_i is of the form $q(a) \leftarrow p_1(b_i)$, and (3) there exists a substitution ϑ satisfying the following conditions (corresponding to Conditions (i)–(iii) of Rule 3): (i) for $i = 1,\ldots,m$, $u_i = b_i\vartheta$, (ii) there exists no clause in $(P_0 \cup Defs_k) - \{D_1,\ldots,D_m\}$ whose head is unifiable with $q(a\vartheta)$ and, hence, by the definition of Rule 1, there exists no clause in $(P_0 \cup Defs_n) - \{D_1,\ldots,D_m\}$ whose head is unifiable with

$q(a\vartheta)$, and (iii) for $i = 1, \ldots, m$ and for every variable U in $vars(b_i) - vars(a)$: (iii.1) $U\vartheta$ is a variable not occurring in $\{t_0, t_2\}$, 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. Thus, the clause E derived by folding $C_1, \ldots, C_m$ using $D_1, \ldots, D_m$ is of the form $p_0(t_0) \leftarrow q(a\vartheta) \wedge p_2(t_2)$.

The extension to the general case where predicates have arbitrary arities and bodies of clauses have arbitrary numbers of atoms is straightforward.

Point (i). In order to prove $M(P_0 \cup Defs_n) \models \{C_1, \ldots, C_m\} \Rightarrow \{E\}$, by Definition 2.1 we have to show that, for every ground instance $E\eta$ of E such that $M(P_0 \cup Defs_n) \models bd(E\eta)$, there exists a ground instance $C_i\varphi_i$ of C_i, for some $i \in \{1, \ldots, m\}$, such that $hd(C_i\varphi_i) = hd(E\eta)$ and $M(P_0 \cup Defs_n) \models bd(C_i\varphi_i)$.
Thus, let us assume that

$$M(P_0 \cup Defs_n) \models (q(a\vartheta) \wedge p_2(t_2))\eta \qquad (\dagger 4)$$

Let us consider the following substitutions: $\alpha = \{U/u \in \vartheta \mid U \in vars(a)\}$ and, for $i = 1, \ldots, m$, $\beta_i = \{U/u \in \vartheta \mid U \in vars(b_i) - vars(a)\}$. By Condition (iii) above (corresponding to Condition (iii) of Rule 3), β_i is of the form: $\{U_1/W_1, \ldots, U_{n_i}/W_{n_i}\}$, where $W_1, \ldots, W_{n_i}$ are distinct variables not occurring in E. Let, for $i = 1, \ldots, m$, ρ_i be the substitution of the form $\{W_1/U_1, \ldots, W_{n_i}/U_{n_i}\}$. The following two properties hold: (P1) $q(a\vartheta) = q(a\alpha)$ and (P2) $p_1(b_i\alpha) = p_1(b_i\vartheta\rho_i)$. From (†4) it follows that $M(P_0 \cup Defs_n) \models q(a\vartheta\eta)$ and thus, by Property (P1), $M(P_0 \cup Defs_n) \models q(a\alpha\eta)$. Since $M(P_0 \cup Defs_n)$ is a fixpoint of $T_{P_0 \cup Defs_n}$ and, by Condition (ii) above (corresponding to Condition (ii) of Rule 3), all clauses of $P_0 \cup Defs_n$ whose head is unifiable with $q(a\vartheta)$ are in $\{D_1, \ldots, D_m\}$, there exists a clause D_i: $q(a) \leftarrow p_1(b_i)$ in $\{D_1, \ldots, D_m\}$ and a ground substitution ν_i such that $M(P_0 \cup Defs_n) \models p_1(b_i\alpha\eta\nu_i)$. By Property (P2) we have that $M(P_0 \cup Defs_n) \models p_1(b_i\vartheta\rho_i\eta\nu_i)$ and, since no variable is common to ρ_i and η, $M(P_0 \cup Defs_n) \models p_1(b_i\vartheta\eta\rho_i\nu_i)$. Hence, by Condition (i) above (corresponding to Condition (i) of Rule 3), $M(P_0 \cup Defs_n) \models p_1(u_i\eta\rho_i\nu_i)$. From (†4) and from the fact that $p_2(t_2\eta)$ is a ground atom it follows that $M(P_0 \cup Defs_n) \models p_2(t_2\eta\rho_i\nu_i)$ and, thus, $M(P_0 \cup Defs_n) \models (p_1(u_i) \wedge p_2(t_2))\eta\rho_i\nu_i$. Let us now consider the substitution $\varphi_i = \eta\rho_i\nu_i$. We have that: $hd(C_i\varphi_i) = p_0(t_0\eta\rho_i\nu_i) = p_0(t_0\eta)$ (because $p_0(t_0\eta)$ is a ground atom) $= hd(E\eta)$ and $M(P_0 \cup Defs_n) \models bd(C_i\varphi_i)$.

Point (ii). Now we prove that $M(\overline{P}_0 \cup \overline{Defs}_n) \models \{\overline{C}_1, \ldots, \overline{C}_m\} \Leftarrow \{\overline{E}\}$. We have that: (4) for $i = 1, \ldots, m$, $\overline{C}_i$ is of the form

$$p_0(t_0, N_0) \leftarrow N_0 \geq U_i + N_2 + w_i \wedge p_1(u_i, U_i) \wedge p_2(t_2, N_2)$$

where $w_i = \sigma(\gamma_k(C_i))$ (recall that σ is a solution of $\mathscr{C}_{final}$), (5) for $i = 1, \ldots, m$, $\overline{D}_i$ is of the form

$$q(a, Q) \leftarrow Q \geq M_i + z_i \wedge p_1(b_i, M_i)$$

where $z_i = \sigma(\delta_k(D_i))$, and (6) $\overline{E}$ is of the form

$$p_0(t_0, N_0) \leftarrow N_0 \geq Q + N_2 + w \wedge q(a\vartheta, Q) \wedge p_2(t_2, N_2)$$

where $w = \sigma(\gamma_{k+1}(E))$.
By Definition 2.1, we have to prove that, for $i = 1, \ldots, m$, for every ground instance $\overline{C}_i\varphi_i$ of $\overline{C}_i$ such that $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{C}_i\varphi_i)$, there exists a ground instance $\overline{E}\eta$ of $\overline{E}$ such that $hd(\overline{E}\eta) = hd(\overline{C}_i\varphi_i)$ and $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{E}\eta)$.
Let $\overline{C}_i\varphi_i$ be a ground instance of $\overline{C}_i$ such that

$$M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq U_i + N_2 + w_i \wedge p_1(u_i, U_i) \wedge p_2(t_2, N_2))\varphi_i$$

We have that $w \leq w_i - z_i$. Indeed, by construction the inequality $\gamma_{k+1}(E) \leq \gamma_k(C_i) - \delta_k(D_i)$ belongs to $\mathscr{C}_{k+1}$, hence it belongs to $\mathscr{C}_{final}$, and σ is a solution of $\mathscr{C}_{final}$. Thus, there exists $r \in \mathbb{N}$ such that

$M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq r + N_2 + w_i \wedge r \geq U_i + z_i \wedge p_1(u_i, U_i) \wedge p_2(t_2, N_2))\varphi_i$ (†5)

Let us consider a ground instance of $\overline{D}_i$ of the form

$(q(a, Q) \leftarrow Q \geq M_i + z_i \wedge p_1(b_i, M_i))\vartheta\varphi_i\tau_i$

where τ_i is a ground substitution such that $Q\tau_i = r$ and $M_i\tau_i = U_i\varphi_i$. We may assume that $vars(\overline{C}_i) \cap vars(\overline{D}_i) = \emptyset$ and $\{Q, M_i\} \cap vars(D_i) = \emptyset$, and therefore, $Q\vartheta\varphi_i\tau_i = Q\tau_i = r$ and $M_i\vartheta\varphi_i\tau_i = M_i\tau_i = U_i\varphi_i$. Thus, by (†5), we have that

$M(\overline{P}_0 \cup \overline{Defs}_n) \models (Q \geq M_i + z_i \wedge p_1(b_i, M_i))\vartheta\varphi_i\tau_i$

and, since $M(\overline{P}_0 \cup \overline{Defs}_n)$ is a fixpoint of $T_{\overline{P}_0 \cup \overline{Defs}_n}$, we have that

$M(\overline{P}_0 \cup \overline{Defs}_n) \models q(a, Q)\vartheta\varphi_i\tau_i$

Thus, by using (†5), the fact that $(N_0 \geq r + N_2 + w_i \wedge p_2(t_2, N_2))\varphi_i$ is a ground goal, and the identity $r = Q\tau_i = Q\varphi_i\tau_i$, we have that

$M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq Q + N_2 + w_i \wedge q(a\vartheta, Q) \wedge p_2(t_2, N_2))\varphi_i\tau_i$

Now, let us consider the ground clause $\overline{E}\eta$ where η is the substitution $\varphi_i\tau_i$. We have just proved that $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{E}\eta)$. Moreover, $hd(\overline{E}\eta) = \overline{E}\varphi_i$ (because $\overline{E}\varphi_i$ is a ground clause) $= p_0(t_0, N_0)\varphi_i = hd(\overline{C}_i\varphi_i)$. ∎

Proof. [Proof of Lemma 4.3.]
Point (i). $M(P_0 \cup Defs_n) \models \{C, D\} \Rightarrow \{D\}$ follows directly from Definition 2.1.

Point (ii). For reasons of simplicity we assume that C is of the form $p_0(t_0\vartheta) \leftarrow p_1(t_1\vartheta) \wedge p_2(t_2)$ and D is of the form $p_0(t_0) \leftarrow p_1(t_1)$. Therefore, $\overline{C}$ is of the form $p_0(t_0\vartheta, N_0) \leftarrow N_0 \geq N_1 + N_2 + w \wedge p_1(t_1\vartheta, N_1) \wedge p_2(t_2, N_2)$ and $\overline{D}$ is of the form $p_0(t_0, N_0) \leftarrow N_0 \geq N_1 + z \wedge p_1(t_1, N_1)$, where $w = \sigma(\gamma_k(C))$ and $z = \sigma(\gamma_k(D))$ (recall that σ is a solution of $\mathscr{C}_{final}$).

In order to prove that $M(\overline{P}_0 \cup \overline{Defs}_n) \models \{\overline{C}, \overline{D}\} \Leftarrow \{\overline{D}\}$, by Definition 2.1 it is enough to show that for every ground instance $\overline{C}\varphi$ of $\overline{C}$ such that $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{C}\varphi)$, there exists a ground instance $\overline{D}\psi$ of $\overline{D}$ such that $hd(\overline{D}\psi) = hd(\overline{C}\varphi)$ and $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{D}\psi)$.

Let $\overline{C}\varphi$ be a ground clause such that $M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq N_1 + N_2 + w \wedge p_1(t_1\vartheta, N_1) \wedge p_2(t_2, N_2))\varphi$. We have that $z \leq w$. Indeed, by construction the inequality $\gamma_k(D) \leq \gamma_k(C)$ belongs to $\mathscr{C}_{k+1}$, hence it belongs to $\mathscr{C}_{final}$, and σ is a solution of $\mathscr{C}_{final}$. Since $N_2 \geq 0$ and ϑ binds neither N_0 nor N_1, we have that $M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq N_1 + z \wedge p_1(t_1, N_1))\vartheta\varphi$. Hence, $hd(\overline{D}\vartheta\varphi) = hd(\overline{C}\vartheta\varphi)$ and $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{D}\vartheta\varphi)$. ∎

Proofs for Section 5 (Goal Replacement)

In the proof of Lemma 5.2 we will use the following property.

Lemma A.1. *Let $P_0 \mapsto \cdots \mapsto P_n$ be a transformation sequence and let $1 \leq k \leq n$. For every closed formula φ such that all predicate symbols occurring in φ also occur in $P_0 \cup Defs_k$, we have that $M(P_0 \cup Defs_k) \models \varphi$ iff $M(P_0 \cup Defs_n) \models \varphi$.*

Proof. It is a straightforward consequence of the following two facts: (1) by the definition of Rule 1 the predicate symbols occurring in $P_0 \cup Defs_k$ do not depend on the head predicates of the clauses in $Defs_n - Defs_k$, and (2) the least Herbrand model semantics satisfies the *relevance* property [Dix95], that is, for all ground atoms A, $A \in M(P)$ iff $A \in M(rel(P, A))$, where $rel(P, A)$ is the set of clauses in P on whose head predicates the predicate of A depends. ∎

Lemma A.1 also holds for weighted programs.

Proof. [Proof of Lemma 5.2.] For reasons of simplicity we assume that: (1) C is of the form $p_0(t_0) \leftarrow p_1(t) \wedge q(u)$, and (2) the replacement law λ is of the form $p_1(X) \Rrightarrow p_2(X)$. Thus, the clause D derived from C by applying λ is of the form $p_0(t_0) \leftarrow p_2(t) \wedge q(u)$. The extension to the general case where predicates have an arbitrary arity and bodies of clauses have an arbitrary number of atoms is straightforward.

Point (i). In order to prove $M(P_0 \cup Defs_n) \models \{C\} \Rightarrow \{D\}$, by Definition 2.1 we will show that, for every ground instance $D\psi$ of D such that $M(P_0 \cup Defs_n) \models bd(D\psi)$, the clause $C\psi$ is a ground instance of C such that $hd(C\psi) = hd(D\psi)$ and $M(P_0 \cup Defs_n) \models bd(C\psi)$.

Let $D\psi$ be a ground instance of D such that $M(P_0 \cup Defs_n) \models (p_2(t) \wedge q(u))\psi$. Hence, $M(P_0 \cup Defs_n) \models p_2(t\psi)$, where $t\psi$ is a ground term. Since the replacement law $p_1(X) \Rrightarrow p_2(X)$ holds in $\overline{P}_0 \cup \overline{Defs}_k[\mathscr{C}]$, by Condition (i) of Definition 5.1 and by Lemma A.1, we have that: $M(P_0 \cup Defs_n) \models p_1(t\psi) \leftarrow p_2(t\psi)$. Therefore, $M(P_0 \cup Defs_n) \models p_1(t\psi)$ and, hence, $M(P_0 \cup Defs_n) \models (p_1(t) \wedge q(u))\psi$.

Thus, $C\psi$ is a ground instance of C such that: $hd(C\psi) = p_0(t_0\psi) = hd(D\psi)$ and, as shown above, $M(P_0 \cup Defs_n) \models bd(C\psi)$.

Point (ii). Recall that, for any solution σ of $\mathscr{C}_{final}$, we denote the weighted program $(\overline{P}_0 \cup \overline{Defs}_n)[\sigma]$ by $\overline{P}_0 \cup \overline{Defs}_n$, and the weighted clauses $\overline{C}[\sigma]$ and $\overline{D}[\sigma]$ by $\overline{C}$ and $\overline{D}$, respectively. We have to show that $M(\overline{P}_0 \cup \overline{Defs}_n) \models \{\overline{C}\} \Leftarrow \{\overline{D}\}$, where: (1) $\overline{C}$ is of the form $p_0(t_0, N_0) \leftarrow N_0 \geq N_1 + N_2 + w \wedge p_1(t, N_1) \wedge q(u, N_2)$, (2) $w = \sigma(\gamma_k(C))$, (3) $\overline{D}$ is of the form $p_0(t_0, N_0) \leftarrow N_0 \geq N_1 + N_2 + z \wedge p_2(t, N_1) \wedge q(u, N_2)$, and (4) $z = \sigma(\gamma_{k+1}(D))$. By Definition 2.1, it will suffice to prove that, for every ground instance $\overline{C}\varphi$ of $\overline{C}$ such that $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{C}\varphi)$, the clause $\overline{D}\varphi$ is a ground instance of $\overline{D}$ such that $hd(\overline{D}\varphi) = hd(\overline{C}\varphi)$ and $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{D}\varphi)$.

Let $\overline{C}\varphi$ be a ground instance of $\overline{C}$ such that

$M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq N_1 + N_2 + w \wedge p_1(t, N_1) \wedge q(u, N_2))\varphi$.

We have that $z \leq w$. Indeed, by the definition of the goal replacement rule, the inequality $\gamma_{k+1}(D) \leq \gamma_k(C)$ belongs to $\mathscr{C}_{k+1}$, hence it belongs to $\mathscr{C}_{final}$, and σ is a solution of $\mathscr{C}_{final}$. Since the replacement law λ holds in $(\overline{P}_0 \cup \overline{Defs}_k)[\mathscr{C}]$ and $\mathscr{C} \subseteq \mathscr{C}_{final}$, by Condition (ii) of Definition 5.1, we have that $M(\overline{P}_0 \cup \overline{Defs}_k) \models p_1(t, N_1)\varphi \rightarrow p_2(t, N_1)\varphi$. By Lemma A.1, we also have that $M(\overline{P}_0 \cup \overline{Defs}_n) \models p_1(t, N_1)\varphi \rightarrow p_2(t, N_1)\varphi$. Therefore, $M(\overline{P}_0 \cup \overline{Defs}_n) \models (N_0 \geq N_1 + N_2 + w \wedge p_2(t, N_1) \wedge q(u, N_2))\varphi$.

Thus, $\overline{D}\varphi$ is a ground instance of D such that: $hd(\overline{D}\varphi) = p_0(t_0, N_0)\varphi = hd(\overline{C}\varphi)$ and, as shown above, $M(\overline{P}_0 \cup \overline{Defs}_n) \models bd(\overline{D}\varphi)$. ∎

Proofs for Section 6 (The Weighted Unfold/Fold Proof Method)

For the proof of Theorem 6.6 we need the following Lemmata A.2 and A.3, which are the converses of Lemmata 4.2 (ii) and 5.2 (ii), respectively.

Lemma A.2. *Let $P_0 \mapsto \cdots \mapsto P_n$ be a transformation sequence and let $1 \leq k < n$. Let $C_1, \ldots, C_m$ be clauses in P_k, let $D_1, \ldots, D_m$ be clauses in $P_0 \cup Defs_k$, and let E be the clause in P_{k+1} derived by folding $C_1, \ldots, C_m$ using $D_1, \ldots, D_m$, as described in Rule 3. Suppose also that the application of the folding rule is symmetric. Then*:

(i) $M(\overline{P}_0 \cup \overline{Defs}_n) \models \{\overline{C}_1, \ldots, \overline{C}_m\} \Rightarrow \{\overline{E}\}$

Proof. The proof is similar to the one of Lemma 4.2, by using also the fact that folding is symmetric and, thus, for $i = 1, \ldots, m$, $\gamma_{k+1}(E) = \gamma_k(C_i) - \delta_k(D_i)$. ∎

Lemma A.3. *Let $P_0 \mapsto \cdots \mapsto P_n$ be a transformation sequence and let $1 \leq k < n$. Let C be a clause in P_k and D be the clause in P_{k+1} derived by applying a replacement law λ that holds in $(\overline{P}_0 \cup \overline{Defs}_k)[\mathscr{C}]$, as described in Rule 5. Suppose also that the application of the goal replacement rule is symmetric. Then*:

(i) $M(\overline{P}_0 \cup \overline{Defs}_n) \models \{\overline{C}\} \Rightarrow \{\overline{D}\}$.

Proof. The proof is similar to the one of Lemma 5.2, by using also the fact that goal replacement is symmetric and, thus, $\gamma_{k+1}(D) = \gamma_k(C)$. ∎

Now, we are ready to prove Theorem 6.6.

Proof. [Proof of Theorem 6.6.] Let $P_0 \mapsto \cdots \mapsto P_n$ be a symmetric transformation sequence and let $\mathscr{C}_{final}$ be its associated correctness constraint system, which is satisfiable by hypothesis. We have shown in Sections 4 and 5 that the following properties hold for any transformation sequence constructed by using the definition introduction, unfolding, folding, and goal replacement rules:

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

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

Thus, (P2) and (P3) also hold for symmetric transformation sequences. Now, we show that also the following property holds:

(P4) $M(\overline{P}_0 \cup \overline{Defs}_n) \models \overline{P}_k \cup (\overline{Defs}_n - \overline{Defs}_k) \Rightarrow \overline{P}_{k+1} \cup (\overline{Defs}_n - \overline{Defs}_{k+1})$.

We reason by cases on the transformation rule applied to derive P_{k+1} from P_k, as follows.
(*Case* 1) If P_{k+1} is derived from P_k by applying the definition introduction rule then $P_k \cup (Defs_n - Defs_k) = P_{k+1} \cup (Defs_n - Defs_{k+1})$ and, therefore, Property (P4) trivially holds.
(*Case* 2) 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 $Defs_k = Defs_{k+1}$. Hence, Property (P4) follows from Lemma 4.1 and monotonicity of $\Rightarrow$.
(*Case* 3) If P_{k+1} is derived from P_k by a symmetric application of the folding rule, then $P_{k+1} = (P_k - \{C_1, \ldots, C_m\}) \cup \{E\}$ and $Defs_k = Defs_{k+1}$. Hence, Property (P4) follows from Lemma A.2 and monotonicity of $\Rightarrow$.
(*Case* 4) If P_{k+1} is derived from P_k by a symmetric application of the goal replacement rule, then $P_{k+1} = P_k - \{C\}$ and $Defs_k = Defs_{k+1}$. Hence, Property (P4) follows from Lemma A.3 and monotonicity of $\Rightarrow$.

By the transitivity of $\Rightarrow$ and by Property (P4), we get

(P5) $M(\overline{P}_0 \cup \overline{Defs}_n) \models \overline{P}_0 \cup \overline{Defs}_n \Rightarrow \overline{P}_n$

Moreover, since $\overline{P}_n = \overline{P}_n[\sigma]$, where σ is a solution of $\mathscr{C}_{final}$, we have that $\overline{P}_n$ is decreasing and, by Lemma 2.10, it is univocal. Thus, by Corollary 2.5, $M(\overline{P}_0 \cup \overline{Defs}_n) = M(\overline{P}_n)$. ∎

In order to prove the soundness of the weighted unfold/fold proof method, that is, Theorem 6.7, we need the following lemma.

Lemma A.4. *Let P be a program, γ_1, γ_2 be weight functions for P, and σ_1, σ_2 be valuations such that, for every clause $C \in P$, $\sigma_1(\gamma_1(C)) \geq \sigma_2(\gamma_2(C))$. Then $M(\overline{P}[\sigma_1]) \subseteq M(\overline{P}[\sigma_2])$.*

Proof. [Proof of Lemma A.4.] For $i = 1, 2$, let T_i denote the immediate consequence operator $T_{\overline{P}[\sigma_i]}$. We have that, for $i = 1, 2$, $M(\overline{P}[\sigma_i]) = T_i \uparrow \omega$. We will prove that $T_1 \uparrow \omega \subseteq T_2 \uparrow \omega$, by showing that, for all $\alpha < \omega$, $T_1 \uparrow \alpha \subseteq T_2 \uparrow \alpha$. We proceed by induction on α.

Basis. We have that $T_1 \uparrow 0 = T_2 \uparrow 0 = \emptyset$.

Inductive Step. We assume that $T_1 \uparrow \alpha \subseteq T_2 \uparrow \alpha$ and $p(t,n) \in T_1 \uparrow (\alpha+1)$, where t is any (tuple of) ground term(s) and n is any nonnegative integer. Then there exists a clause $C \in P$ such that: (i) $C[\sigma_1]$ has a ground instance of the form $p(t,n) \leftarrow n \geq n_1 + \cdots + n_k + w_1 \wedge p_1(t_1,n_1) \wedge \ldots \wedge p_k(t_k,n_k)$, where $w_1 = \sigma_1(\gamma_1(C))$, (ii) $n \geq n_1 + \cdots + n_k + w_1$ holds and, (iii) for $i = 1,\ldots,k$, $p_i(t_i,n_i) \in T_1 \uparrow \alpha$. Hence, by hypothesis, $n \geq n_1 + \cdots + n_k + w_2$, where $w_2 = \sigma_2(\gamma_2(C))$ and, by the inductive hypothesis, for $i = 1,\ldots,k$, $p_i(t_i,n_i) \in T_2 \uparrow \alpha$. Thus, $p(t,n) \in T_2 \uparrow (\alpha+1)$. ∎

Now we can prove the soundness of the weighted unfold/fold proof method.

Proof. [Proof of Theorem 6.7.] Point (i). Assume $\overline{P}[\mathscr{C}] \vdash_{UF} p_1(X) \Rrightarrow p_2(X)$. By Definition 5.1 we have to show the following two properties:
(i.1) $M(P) \models \forall X(p_1(X) \leftarrow p_2(X))$, and
(i.2) for every solution σ of $\mathscr{C}$, $M(\overline{P}[\sigma]) \models \forall X \forall N\ (p_1(X,N) \rightarrow p_2(X,N))$.

Point (i.1). Let t be a tuple of ground terms. We assume that $M(P) \models p_2(t))$. We have to show that $M(P) \models p_1(t)$. Since P is of the form $T \cup \{D_1, D_2\}$ and the predicate p_2 does not depend on the head of D_1, we have that $M(T \cup \{D_2\}) \models p_2(t)$. By Condition (1) of the weighted unfold/fold proof method and by Theorem 5.3, we have that $M(R) \models p_2(t)$. By Condition (2) of the weighted unfold/fold proof method and by Lemma 6.2, we have that $M(Q) \models p_1(t)$. By Condition (1) of the weighted unfold/fold proof method and by Theorem 5.3, we have that $M(T \cup \{D_1\}) \models p_1(t)$. Since D_1 is the unique clause with head predicate p_1 in $T \cup \{D_1\}$, we have that $M(T) \models p_1(t)$ and, thus, $M(P) \models p_1(t)$.

Point (i.2). Let t be a tuple of ground terms, n be a nonnegative integer, and σ be a solution of $\mathscr{C}$ (we assume that the domain of σ contains all unknowns associated with clauses appearing in $T \cup \{D_1\} \mapsto \cdots \mapsto Q$ or $T \cup \{D_2\} \mapsto \cdots \mapsto R$). We assume that $M(\overline{P}[\sigma]) \models p_1(t,n)$. We have to show that $M(\overline{P}[\sigma]) \models p_2(t,n)$.

Since $P = T \cup \{D_1, D_2\}$ and p_1 does not depend on the head predicate of D_2 (that is, p_2), we have that $M((\overline{T} \cup \{\overline{D}_1\})[\sigma]) \models p_1(t,n)$. Let $DefsQ$ be the set of clauses introduced by Rule 1 during the transformation sequence $T \cup \{D_1\} \mapsto \cdots \mapsto Q$. The set of constraints associated with $T \cup \{D_1\} \cup DefsQ$ is empty and every valuation is a solution of the empty set of constraints. Thus, $M((\overline{T} \cup \{\overline{D}_1\} \cup \overline{DefsQ})[\sigma]) \models p_1(t,n)$. Since by hypothesis $\mathscr{C}_Q$ (which is a subset of $\mathscr{C}$) is satisfiable and σ is a solution of $\mathscr{C}_Q$, the weighted program $\overline{Q}[\sigma]$ is decreasing. By Lemmata 4.1–5.2, we have that $M((\overline{T} \cup \{\overline{D}_1\} \cup \overline{DefsQ})[\sigma]) \models (\overline{T} \cup \{\overline{D}_1\} \cup \overline{DefsQ})[\sigma] \Leftarrow \overline{Q}[\sigma]$ and, therefore, by Theorem 2.4, $M((\overline{T} \cup \{\overline{D}_1\} \cup \overline{DefsQ})[\sigma]) \subseteq M(\overline{Q}[\sigma])$. Then $M(\overline{Q}[\sigma]) \models p_1(t,n)$. By hypothesis Q is syntactically equivalent to R and since σ is a solution of $\mathscr{C}$, by Step (B.1) of the weighted unfold/fold proof method we have that, for every clause $C \in Q$, $\sigma(\gamma_Q(C)) \geq \sigma(\gamma_R(\rho(C)))$. Let us consider the weight function γ'_Q such that, for every $C \in Q$, $\gamma'_Q = \gamma_R(\rho(C))$, and let σ_Q and σ_R be the restrictions of σ to the set of unknowns occurring in the range of γ_Q and γ_R, respectively. We have that $\sigma_Q(\gamma_Q(C)) \geq \sigma_R(\gamma'_Q(C))$ and hence, by Lemma A.4, $M(\overline{Q}[\sigma_Q]) \subseteq M(\overline{Q}[\sigma_R])$. Since $\rho(\overline{Q}[\sigma_R]) = \overline{R}[\sigma_R]$ and $\rho(p_1) = p_2$, by Lemma 6.2 we have that $M(\overline{Q}[\sigma_R]) \models p_1(t,n)$ iff $M(\overline{R}[\sigma_R]) \models p_2(t,n)$. Thus, since $\overline{Q}[\sigma_Q] = \overline{Q}[\sigma]$ and $\overline{R}[\sigma_R] = \overline{R}[\sigma]$, we get that $M(\overline{R}[\sigma]) \models p_2(t,n)$. By Condition (3) of the weighted unfold/fold method and by Theorem 6.6, $M((\overline{T} \cup \{\overline{D}_2\} \cup \overline{DefsR})[\sigma]) \models p_2(t,n)$ and, since p_2 does not depend on the head predicates of the clauses in $DefsR$, we have that $M((\overline{T} \cup \{\overline{D}_2\})[\sigma]) \models p_2(t,n)$. Finally, since p_2 does not depend on the head predicate of D_1 (that is, p_1), we have that $M(\overline{P})[\sigma] \models p_2(t,n)$.

Point (ii). By Point (i) and by Definition 6.4 it is enough to show that if $\overline{P}[\mathscr{C}] \vdash_{UF} p_1(X) \Leftrightarrows p_2(X)$ then $\overline{P}[\mathscr{C}] \vdash_{UF} p_2(X) \Rrightarrow p_1(X)$, that is, it is enough to show that if Steps (A) and (B.2) of the weighted unfold/fold proof method can be performed for two suitable programs Q and R, then

Steps (A) and (B.1) can be performed by interchanging Q and R. This can be shown by using the following properties:
(A1) any predicate renaming ρ from Q to R whose existence is stipulated in Condition (2), has an inverse ρ^{-1} from R to Q,
(A2) by the assumption made at Step (B.2), $T \cup \{D_1\} \mapsto \cdots \mapsto Q$ is symmetric, and
(A3) for every clause $C \in R$ we have that $\{\gamma_R(C) \geq \gamma_Q(\rho^{-1}(C)) \mid C \in Q\} \cup \mathscr{C}_Q \cup \mathscr{C}_R$, because by Step (B.2) we have that $\{\gamma_Q(C) = \gamma_R(\rho(C)) \mid C \in Q\} \cup \mathscr{C}_Q \cup \mathscr{C}_R$. ∎

B. Comparing MAP and SCOUT through an Example

The SCOUT transformation system implements the transformation method presented in [RKR04] by associating with each clause a *measure* consisting of a pair of tuples of integers. The use of tuples is related to a *stratification* of the program to be transformed, that is, the i-th component of the tuple is a measure referring to the i-th stratum of the program. The use of pairs is relevant when a set of $n\,(\geq 2)$ clauses is folded by another set of n clauses, thereby deriving a single clause. More details can be found in [RKR04]. Here we only present an example where SCOUT is not able to prove the correctness of a folding step, while the MAP transformation system, based on the approach presented in this paper, easily proves the correctness of that step.

In our example we have one stratum only and, hence, clause measures consist of pairs of integers (that is, pairs of 1-tuples). Moreover, we want to fold one clause only and, thus, the use of *pairs* of integers is actually not needed. However, we will use those pairs for compliance with SCOUT.

Let us consider the following program, where every clause is annotated by the pair $\langle 1, 1\rangle$, as prescribed by the SCOUT system for the initial program of any transformation sequence.

1. $p(a) \leftarrow p(b)$ $\quad\quad\quad\quad$ $\langle 1, 1\rangle$
2. $p(c) \leftarrow p(b)$ $\quad\quad\quad\quad$ $\langle 1, 1\rangle$
3. $p(b) \leftarrow$ $\quad\quad\quad\quad$ $\langle 1, 1\rangle$

Suppose that we want to fold clause 1 using clause 2 and derive the clause:

4. $p(a) \leftarrow p(c)$

The transformation sequence constructed by the above folding step is totally correct. However, SCOUT disallows that step because it does not satisfy the applicability condition for the folding rule. Indeed, in this example folding is allowed only if the second component of the measure of clause 2 is strictly less than the first component of the measure of clause 1.

On the other hand, the MAP system proves that the transformation sequence constructed by folding clause 1 using clause 2 is totally correct. Indeed, the following set of constraints, where u_i is the unknown associated with clause i,

$$\{u_2 \geq 1, \quad u_3 \geq 1, \quad u_4 \geq 1, \quad u_4 \leq u_1 - u_2\}$$

is satisfiable.