International Journal of Foundations of Computer Science
© World Scientific Publishing Company

Beyond Tamaki-Sato Style Unfold/Fold Transformations for Normal Logic Programs

Abhik Roychoudhury
Department of Computer Science, School of Computing, National University of Singapore, S16 Level 5, 3 Science Drive 2, Singapore 117543. e-mail: abhik@comp.nus.edu.sg

K. Narayan Kumar
Chennai Mathematical Institute, 92 G.N. Chetty Road, Chennai, India. e-mail: kumar@smi.ernet.in

C.R. Ramakrishnan
Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794, USA. e-mail: cram@cs.sunysb.edu

and

I.V. Ramakrishnan
Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794, USA. e-mail: ram@cs.sunysb.edu

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Unfold/fold transformation systems for logic programs have been extensively investigated. Existing unfold/fold transformation systems for normal logic programs typically fold using using a single, non-recursive clause *i.e.* the folding transformation is very restricted. In this paper we present a transformation system that permits folding in the presence of recursion, disjunction, as well as negation. We show that the transformations are correct with respect to various model theoretic semantics of normal logic programs including the well-founded model and stable model semantics.

Keywords: Logic Programming, Program Transformations, Unfold/Fold Transformations, Normal Logic Programs.

1. Introduction

Unfold/fold transformation systems for logic programs have been used for automated deduction [10, 17, 18], and program specialization and optimization [2, 4, 12]. Normal logic programs consist of definitions of the form A:– ϕ where A is an atom and ϕ is a boolean formula (not necessarily positive) over atoms. Unfolding replaces an occurrence of A in a program with ϕ while folding replaces an occurrence of ϕ with A. Folding is called *reversible* if its effects can be undone by an unfolding, and *irreversible* otherwise. Thus, in a folding step which replaces a boolean formula ϕ with A in program P

- if the definition A :– ϕ appears in P, then the folding is reversible
- otherwise, the folding step is irreversible

Given a logic program P, an unfold/fold transformation system generates a sequence of programs $P = P_0, P_1, \ldots, P_n$, such that for all $0 \leq i < n$, P_{i+1} is obtained from P_i by applying one of the two transformations. Unfold/fold transformation systems are proved correct by showing that all programs in any transformation sequence $P_0, P_1, \ldots, P_n$ are equivalent under a suitable semantics, such as the well-founded model semantics for normal logic programs. A comprehensive survey of research on logic program transformations appears in [16].

As an illustration of unfolding/folding, consider the sequence of normal logic programs in figure 1. In the figure, P_1 is derived from P_0 by unfolding the occurrence of `q(X)` in the first clause of P_0. Program P_2 is derived from P_1 by folding the literal `q(Y)` in the body of the second clause of `p/1` into `p(Y)` by using `p(X) :- q(X)` in P_0. This clause from a previous program which is used in a folding step (the clause `p(X) :- q(X)` of P_0 in this case) is called the *folder*. This is an example of an irreversible folding step since the clause `p(X) :- q(X)` does not appear in P_1.

1.1. Related Work

An unfold/fold transformation system for definite logic programs was first described in a seminal paper by Tamaki and Sato [24]. It allows folding using a single clause only (*conjunctive* folding) from the initial program. This folder clause is required to be non-recursive, but need not be present in the current program P_i. Maher [14] proposed a transformation system using only reversible folding in which the folder clause is always drawn from the current program. However, reversibility is a restrictive condition that limits the power of unfold/fold systems by disallowing

```
p(X):- q(X).
q([]).
q([X|Y]):- ¬r(X),q(Y).
```

Program P_0

```
p([]).
p([X|Y]):- ¬r(X),q(Y).
q([]).
q([X|Y]):- ¬r(X),q(Y).
```

Program P_1

```
p([]).
p([X|Y]):- ¬r(X),p(Y).
q([]).
q([X|Y]):- ¬r(X),q(Y).
```

Program P_2

Figure 1: Example of an unfold/fold transformation sequence

many correct transformations, such as the one used to derive P_2 from P_1 in Figure 1. Hence, there was considerable interest in developing irreversible unfold/fold transformation systems, for both definite and normal logic programs.

Existing unfold/fold transformation systems for normal logic programs can be broadly classified based on the semantics preserved by the transformations. Gardner and Shepherdson [6] presented unfold/fold transformations which preserve SLDNF derivations and Clark's completion semantics. On the other hand, a host of other authors have investigated the correctness of unfold/fold transformations w.r.t. model theoretic semantics which are constructively defined *e.g.* perfect model and well-founded model semantics. These works include [15, 17, 22] which study transformations for stratified programs preserving perfect model semantics, and [1, 23] which present transformations for general logic programs preserving well-founded model and other semantics. However, these works are either extensions of Maher's reversible transformation system or the original Tamaki-Sato system [24]. In particular, [15] extends Maher's reversible transformation system to stratified logic programs, and [1, 17, 22, 23] present extensions of the Tamaki-Sato style irreversible transformation system to normal logic programs. All these transformation systems have a more restrictive folding rule than ours. Our folding rule allows (a) multiple folder clauses, (b) recursive folder clauses and (c) folder clauses drawn from a previous program in the transformation sequence. However, some of these works (such as [17]) present a more powerful unfolding rule since they allow unfolding of negative literals.

To the best of our knowledge, the only work which studies correctness of unfold/fold transformations in presence of recursion, disjunction and negation is [21]. This work consider unfold/fold transformations of first-order programs, but the semantics considered is three-valued semantics. Even for definite logic programs, irreversible transformations of programs, until recently, did not allow folding using multiple recursive clauses *i.e.* folding in presence of disjunction and recursion. In [19] these restrictions were relaxed; the technique used there is conceptually related to Kanamori's work on logic programs in [11] and Sands' work on functional programs in [20]. In this paper, we substantially extend this work to allow folding in presence of disjunction, recursion and negation. The correctness of these transformations hold w.r.t. various model-theoretic semantics of normal logic programs such as well-founded model, stable model etc.

1.2. Overview of Results

The main result of this paper is a unfold/fold transformation system that performs folding in the presence of recursion, disjunction as well as negation (see Section 2). In [19] we proposed a transformation framework for definite logic programs which permits folding using multiple recursive clauses. These transformations associate *counters* with program clauses (*a la* Kanamori and Fujita [11]) to determine the applicability of fold and unfold transformations. In this paper, we extend this scheme to accommodate negative literals. We show that this extension is sufficient to guarantee that the resulting transformation system preserves a variety of seman-

tics for normal logic programs, such as the well-founded model, stable model, partial stable model, and stable theory semantics. Central to this proof is the result due to Dung and Kanchanasut [5] that preserving the *semantic kernel* of a program is sufficient to guarantee the preservation of the different semantics for negation listed above. The notion of semantic kernel is a powerful one, since it captures the commonality of various semantics of normal logic programs. An equivalent notion, called argumentation theoretic semantics, has also been proposed in [25]. In [1], Aravindan and Dung use this notion to develop an unfold/fold transformation system which preserves the semantic kernel, thereby preserving the various semantics of negation.

However, in contrast to [1] where this idea was used to prove the correctness of Tamaki-Sato style transformations, we present a two-step proof which explicitly uses the operational counterpart of semantic kernels (see Section 3). In the first step of our proof, we show that the transformations preserve positive ground derivations, which are derivations of the form $A \rightsquigarrow \neg B_1, \neg B_2, \ldots, \neg B_n$ such that there is a proof tree rooted at A with leaves labeled $\neg B_1$ through $\neg B_n$ (apart from true). We then show that preserving positive ground derivations is equivalent to preserving the semantic kernel of the program. Thus positive ground derivations are the operational analogues of semantic kernels.

Our correctness proof suggests that we can treat the negative literals in a program as atoms of new predicates defined in a different (external) module. The correctness of the transformation system is assured as long as the transformations respect module boundaries (see Section 4). This indicates how a transformation system originally designed for definite logic programs (such as the one we proposed in [19]) can be adapted for normal logic programs.

2. The Transformation System

In this section, we present our *unfold* and *fold* transformations for normal logic programs. We assume familiarity with the standard notions of terms, substitutions, unification, most general unifier (mgu), atoms, literals, clauses and stratification [13]. We will use the following symbols (possibly with primes and subscripts): P to denote a normal logic program; C and D for clauses; A, B to denote atoms ; L, K to denote literals ; $\mathcal{N}$ to denote sequence of literals and σ, θ for substitutions.

Recall that a transformation sequence is a sequence of programs $P_0, P_1, \ldots, P_n$ such that for all $0 \leq i < n$, P_{i+1} is obtained from P_i by applying a transformation (unfolding or folding). In any transformation sequence $P_0, P_1, \ldots, P_n$ we annotate each clause C in program P_i with a pair $(\gamma^i_{lo}(C), \gamma^i_{hi}(C))$ where $\gamma^i_{lo}(C), \gamma^i_{hi}(C) \in \mathbb{Z}$ and $\gamma^i_{lo}(C) \leq \gamma^i_{hi}(C)$. Thus, γ^i_{lo} and γ^i_{hi} are functions from the set of clauses in program P_i to the set of integers $\mathbb{Z}$. The transformation rules dictate the construction of γ^{i+1}_{lo} and γ^{i+1}_{hi} from γ^i_{lo} and γ^i_{hi}. We assume that for any clause C in the initial program P_0, $\gamma^0_{lo}(C) = \gamma^0_{hi}(C) = 1$. Intuitively, $\gamma^i_{lo}(C)$ and $\gamma^i_{hi}(C)$ for a clause C are analogous to the Kanamori-Fujita style counters [11]; the separation of *hi* and *lo* permits us to store estimates of the counter values which is crucial for allowing disjunctive folding.

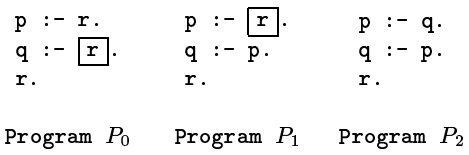

Figure 2: Indiscriminate folding is not semantics preserving

Rule 1 (Unfolding) Let C be a clause in P_i and A a positive literal in the body of C. Let $C_1, ..., C_m$ be the clauses in P_i whose heads are unifiable with A with most general unifiers $\sigma_1, ..., \sigma_m$. Let C'_j be the clause that is obtained by replacing $A\sigma_j$ by the body of $C_j\sigma_j$ in $C\sigma_j$ $(1 \leq j \leq m)$.

Then, assign $P_{i+1} := (P_i - \{C\}) \cup \{C'_1, ..., C'_m\}$. Set $\gamma^{i+1}_{lo}(C'_j) = \gamma^i_{lo}(C) + \gamma^i_{lo}(C_j)$ and $\gamma^{i+1}_{hi}(C'_j) = \gamma^i_{hi}(C) + \gamma^i_{hi}(C_j)$. The annotations of all other clauses in P_{i+1} are inherited from P_i. □

Rule 2 (Folding) Let $\{C_1, ..., C_m\}$ be clauses in P_i and $\{D_1, ..., D_m\}$ be clauses in P_j $(j \leq i)$ where C_l denotes the clause $A{:-}\ L_{l,1}, \ldots, L_{l,n_l}, L'_1, \ldots, L'_n$ and D_l denotes the clause $B_l{:-}\ K_{l,1}, \ldots, K_{l,n_l}$. Also, let

1. $\forall 1 \leq l \leq m\ \exists \sigma_l\ \forall 1 \leq k \leq n_l\ L_{l,k} = K_{l,k}\sigma_l$, where σ_l is a substitution.
2. $B_1\sigma_1 = B_2\sigma_2 = ... = B_m\sigma_m = B$
3. $D_1, ..., D_m$ are the only clauses in P_j whose heads are unifiable with B.
4. $\forall 1 \leq l \leq m\ \sigma_l$ substitutes the internal variables[a] of D_l to distinct variables which do not appear in $\{A, B, L'_1, ..., L'_n\}$.
5. $\forall 1 \leq l \leq m\ \gamma^j_{hi}(D_l) < \gamma^i_{lo}(C_l)$ + Number of positive literals in the sequence $L'_1, \ldots, L'_n$.

Then, assign $P_{i+1} := (P_i - \{C_1, ..., C_m\}) \cup \{C'\}$ where $C' \equiv A{:-}\ B, L'_1, ..., L'_n$. Set $\gamma^{i+1}_{lo}(C') = min_{1 \leq l \leq m}(\gamma^i_{lo}(C_l) - \gamma^j_{hi}(D_l))$ and $\gamma^{i+1}_{hi}(C') = max_{1 \leq l \leq m}(\gamma^i_{hi}(C_l) - \gamma^j_{lo}(D_l))$. The annotations of all other clauses in P_{i+1} are inherited from P_i. □

The first four conditions of the folding rule are essential for any correct application of a single folding step, including reversible folding. Finally, the fifth condition of the rule governs sound applications of folding in a transformation sequence. The necessity of the fifth condition arises from potential circularities imposed by indiscriminate folding as shown in Figure 2. We obtain program P_1 from P_0 by folding `q` using the definition of `p` as folder. Now, if we fold `p` using the definition of `q` in P_0 as folder, then `p`,`q` are removed from the least model.

To avoid the introduction of such circularities, our folding rule does not impose any restrictions on the syntax of the folder clauses. We avoid imposing restrictions on the syntax by maintaining annotations to each clause $C \in P_i$ for a transformation sequence $P_0, P_1, \ldots, P_n$. Restrictions are then imposed on the clause annotations to maintain correctness of folding. For definite programs, we define:

Definition 1 (Ground Proof of an Atom) *A ground proof of a ground atom in a definite logic program P is a tree T such that : (1) each internal node of T is*

[a] An internal variable of a clause C is a variable appearing in the body of C, but not in the head of C.

labeled with a ground atom (2) each leaf node of T is labeled with the special symbol `true` *(3) for any internal node A of T, A:– $B_1, \ldots, B_n$ must be a clause in program P where $B_1, \ldots, B_n$ are the children of A in T.*

Now let us consider a clause C in P_i. Informally, the annotations $\gamma^i_{lo}(C)$ and $\gamma^i_{hi}(C)$ denote the lower and upper bound on savings incurred in the size of any ground proof of a ground atom B in P_i (relative to the smallest proof of B in the initial program P_0) by applying the clause C. The folding rule is applicable provided the minimum savings accrued in the folded clause is more than the maximum savings in the folder clause. Note that this notion is similar to Sands' notion of "improvement" for functional programs [20]. In particular, this ensures that there is positive savings in the clause generated by folding.

For normal logic programs, we can augment the above notion of a ground proof of an atom to define :

Definition 2 (Positive ground derivation) *A positive ground derivation of a literal in a normal logic program P is a tree T such that : (1) each internal node of T is labeled with a ground atom (2) each leaf node of T is labeled with a negative ground literal or the special symbol* `true`*, and (3) for any internal node A of T, A:– $L_1, \ldots, L_n$ must be a clause in program P where $L_1, \ldots, L_n$ are the children of A in T.*

Now let T be a positive ground derivation in program P whose root is labeled with the ground literal L. Let $\mathcal{N}$ be the *sequence of negative literals*[b] derived in T *i.e.* $\mathcal{N}$ is formed by appending the negative literals appearing in the leaf nodes of T from left to right. Then we say that "L derives $\mathcal{N}$" and denote this as $L \rightsquigarrow_P \mathcal{N}$ or $L \rightsquigarrow \mathcal{N}$ if the program P is obvious from the context. Thus if $L \rightsquigarrow_{P_i} \mathcal{N}$, then starting from L we can repeatedly resolve positive literals (using clauses of program P_i) to obtain a sequence of negative literals $\mathcal{N}$. Note that if L is a ground negative literal, there is only one positive ground derivation for L in any program, namely the *empty* derivation $L \rightsquigarrow L$.

Let $P_0, P_1, \ldots, P_n$ be a transformation sequence of normal logic programs and consider a clause C in P_i. Then, roughly speaking, $\gamma^i_{lo}(C)$ and $\gamma^i_{hi}(C)$ denote the lower and upper bound on savings incurred in the size of any positive ground derivation $B \rightsquigarrow_{P_i} \mathcal{N}$ (relative to the size of the smallest derivation $B \rightsquigarrow \mathcal{N}$ in P_0) by applying the clause C. Folding is applicable if the minimum savings in the folded clause exceeds the maximum savings in the folder clause.

An Example: The following example (derived from [9]) illustrates the use of our unfold/fold transformation system. The `odd/1` and `even/1` predicates are encoded in the usual way and are not shown. The two integers to the right of each clause denote the counter values.

[b] For simplicity of exposition, we consider $\mathcal{N}$ to be a sequence. However, our results hold even if $\mathcal{N}$ is regarded as a multi-set.

C_1 : `in_position(X,L) :- in_odd(X,L), ¬ even(X).` (1,1)
C_2 : `in_position(X,L) :- in_even(X,L), ¬ odd(X).` (1,1)
C_3 : `in_odd(X,[X|L]).` (1,1)
C_4 : `in_odd(X,[Y,Z|L]) :- in_odd(X,L).` (1,1)
C_5 : `in_even(X,[Y,X|L]).` (1,1)
C_6 : `in_even(X,[Y,Z|L]) :- in_even(X,L).` (1,1)

In the above program, `in_odd(X,L)` (`in_even(X,L)`) is true if `X` appears in an odd (even) position in list `L`. Thus, `in_position(X,L)` is true if `X` is in an odd (even) position in list `L`, and `X` is not an even (odd) number. Unfolding `in_odd(X,L)` in C_1 we get the following:

C_7 : `in_position(X,[X|L]) :- ¬ even(X).` (2,2)
C_8 : `in_position(X,[Y,Z|L]) :- in_odd(X,L), ¬ even(X).` (2,2)
C_2 : `in_position(X,L) :- in_even(X,L), ¬ odd(X).` (1,1)

Unfolding `in_even(X,L)` in C_2 yields the following clauses:

C_7 : `in_position(X,[X|L]) :- ¬ even(X).` (2,2)
C_8 : `in_position(X,[Y,Z|L]) :- in_odd(X,L), ¬ even(X).` (2,2)
C_9 : `in_position(X,[Y,X|L]) :- ¬ odd(X).` (2,2)
C_{10} : `in_position(X,[Y,Z|L]) :- in_even(X,L), ¬ odd(X).` (2,2)

Finally, we fold clauses $\{C_8, C_{10}\}$ using the clauses $\{C_1, C_2\}$ from the initial program as the folder to obtain the following definition of `in_position/1`.

C_7 : `in_position(X,[X|L]) :- ¬ even(X).` (2,2)
C_9 : `in_position(X,[Y,X|L]) :- ¬ odd(X).` (2,2)
C_{11} : `in_position(X,[Y,Z|L]) :- in_position(X,L).` (1,1)

Note that the final step is an irreversible folding in presence of negation that uses multiple clauses as the folder. Such a folding step is neither allowed in Tamaki-Sato style transformation systems for normal logic programs nor in reversible transformation systems.

Remark: We can maintain more elaborate book-keeping information than integer counters, thereby deriving more expressive unfold/fold systems. For instance, as in the SCOUT system described in [19], we can make the counters range over use a tuple of integers, and obtain a system that is *strictly more powerful* (in terms of transformation sequences allowed) than the existing Tamaki-Sato-style systems. The construction parallels that of the SCOUT system in [19]; details are omitted.

3. Proof of Correctness

In this section, we show that our unfold/fold transformation system is correct with respect to various semantics of normal logic programs. This proof proceeds in three steps. First, we show that positive ground derivations (introduced in Definition 2) are preserved by the transformations. Secondly, we show that preserving positive ground derivations is equivalent to preserving semantic kernel [5]. Finally, following [1], preserving semantic kernel implies that the transformation system is correct with respect to various semantics for normal logic programs including well-founded model [7] and stable model semantics [8]. We begin with a review of semantic kernels.

Definition 3 (Quasi-Interpretation) [1, 5] *A quasi-interpretation of a normal logic program P is a set of ground clauses of the form $A{:-}\ \neg B_1, \ldots, \neg B_n$ $(n \geq 0)$ where $A, B_1, \ldots, B_n$ are ground atoms in the Herbrand Base of P.*

Quasi-interpretations form the universe over which semantic kernels are defined. For a given normal logic program P, the set of all quasi-interpretations of P (denoted $QI(P)$) forms a complete partial order with a least element (the empty set ϕ) with respect to the set inclusion relation $\subseteq$.

Definition 4 *Given a normal logic program P, let $Gnd(P)$ denote the set of all possible ground instantiations of all clauses of P. The function S_P on quasi-interpretations of P is defined as*

$$S_P : QI(P) \rightarrow QI(P)$$
$$S_P(I) = \{\mathcal{R}(C, D_1, \ldots, D_m) \mid C \in Gnd(P) \wedge D_i \in I,\ 1 \leq i \leq m\}$$

where, if $D_i (1 \leq i \leq m)$ are ground clauses

$$A_i{:-}\ \ \neg B_{i,1}, \ldots, \neg B_{i,n_i}\ (n_i \geq 0)$$

and $A_1, \ldots, A_m (m \geq 0)$ are the only positive literals appearing in the body of ground clause C, then $\mathcal{R}(C, D_1, \ldots, D_m)$ is the clause obtained by resolving the positive body literals $A_1, \ldots, A_m$ in C using clauses $D_1, \ldots, D_m$ respectively. □

If P is a definite program, then the function S_P is identical to the immediate logical consequence operator T_P [13]. The semantic kernel of the program P is defined in terms of S_P as:

Definition 5 (Semantic Kernel) [1, 5] *The semantic kernel of a normal logic program P, denoted by $SK(P)$, is the least fixed point of the function S_P, i.e.,*
$SK(P) = \bigcup_{n \in \omega} SK^n(P)$ *where* $SK^0(P) = \phi$ *and* $SK^{n+1}(P) = S_P(SK^n(P))$

Example : Consider the following normal logic program P:

```
p :- ¬ q, r.
r :- ¬ r.
```

The semantic kernel of P will be computed as follows.

$SK^0(P) = \{\}$.
$SK^1(P) = S_P(SK^0(P)) = \{$ (`r :- ¬ r`) $\}$
$SK^2(P) = S_P(SK^1(P)) = \{$ (`r :- ¬ r`), (`p :- ¬ q, ¬ r`) $\}$
$SK^3(P) = S_P(SK^2(P)) = SK^2(P)$
Therefore, $SK(P) = \{$ (`r :- ¬ r`), (`p :- ¬ q, ¬ r`) $\}$

The following theorem from [1] formally states the equivalence of P and $SK(P)$ with respect to various semantics of normal logic programs.

Theorem 1 [1] *Let P be a normal logic program and $SK(P)$ be its semantic kernel. Then :*
(1) If P is a definite logic program, then P and $SK(P)$ have the same least Herbrand Model.
(2) If P is a stratified program, then P and $SK(P)$ have the same perfect model semantics.
(3) P and $SK(P)$ have the same well-founded model.

(4) P and $SK(P)$ have the same stable model(s).
(5) P and $SK(P)$ have the same set of partial stable models.
(6) P and $SK(P)$ have the same stable theory semantics.

3.2. Preserving the Semantic Kernel

We now show that in any transformation sequence $P_0, P_1, \ldots, P_n$ where P_{i+1} is obtained from P_i by applying unfolding (rule 1) or folding (rule 2), the semantic kernel is preserved, i.e., $SK(P_0) = SK(P_1) = \ldots = SK(P_n)$. To do so, we will use the notion of a positive ground derivation introduced in the last section (refer Definition 2). We now define:

Definition 6 (Weight of a positive ground derivation) *Let $L \rightsquigarrow_P \mathcal{N}$ be a positive ground derivation. The number of internal nodes in this derivation (*i.e. *the number of nodes labeled with a ground positive literal) is called the weight of the derivation.*

Definition 7 (Weight of a pair) *Let $P_0, \ldots, P_n$ be a transformation sequence of normal logic programs. Let L be a ground literal, $\mathcal{N}$ be a (possibly empty) sequence of ground negative literals s.t. $L \rightsquigarrow_{P_0} \mathcal{N}$. Then, the weight of $(L, \mathcal{N})$, denoted by $w(L, \mathcal{N})$, is the minimum of the weights of positive ground derivations of the form $L \rightsquigarrow_{P_0} \mathcal{N}$.*

Note that for *any* program P_i in the transformation sequence, the weight of any pair $w(L, \mathcal{N})$ is defined as the weight of the smallest derivation $L \rightsquigarrow_{P_0} \mathcal{N}$.

Definition 8 *Let $P_0, P_1, ..., P_n$ be a transformation sequence of normal logic programs. A positive ground derivation $L \rightsquigarrow_{P_i} \mathcal{N}$ is said to be weakly weight-consistent if for every ground instance A:$-$ $L_1, ..., L_k$ of a clause C used in this derivation, we have $w(A, \mathcal{N}_A) \leq \gamma^i_{hi}(C) + \sum_{1 \leq l \leq k} w(L_l, \mathcal{N}_l)$ where $\mathcal{N}_A, \mathcal{N}_1, ..., \mathcal{N}_k$ are the negative literal sequences derived from $A, L_1, ..., L_k$ respectively in this derivation.*

Definition 9 *Let $P_0, P_1, ..., P_n$ be a transformation sequence of normal logic programs. A positive ground derivation $L \rightsquigarrow_{P_i} \mathcal{N}$ is said to be strongly weight-consistent if for every ground instance A:$-$ $L_1, ..., L_k$ of a clause C used in this derivation, we have*

- $w(A, \mathcal{N}_A) \geq \gamma^i_{lo}(C) + \sum_{1 \leq l \leq k} w(L_l, \mathcal{N}_l)$
- $\forall 1 \leq l \leq k \; w(A, \mathcal{N}_A) > w(L_l, \mathcal{N}_l)$

where $\mathcal{N}_A, \mathcal{N}_1, ..., \mathcal{N}_k$ are the negative literal sequences derived from $A, L_1, ..., L_k$ in this derivation.

Definition 10 (Weight consistent program) *Let $P_0, P_1, \ldots, P_n$ be a transformation sequence of normal logic programs. Then, program P_i is said to be weight consistent if*

- *for any pair $(L, \mathcal{N})$, whenever L derives $\mathcal{N}$ in P_i, there is a strongly weight consistent positive ground derivation $L \rightsquigarrow_{P_i} \mathcal{N}$.*
- *every positive ground derivation in P_i is weakly weight consistent.*

Using the above definitions, we now state certain invariants which always hold after the application of any unfold/fold transformation.

- $I1(P_i) \equiv \forall L \forall \mathcal{N}\ (L \rightsquigarrow_{P_0} \mathcal{N} \Leftrightarrow L \rightsquigarrow_{P_i} \mathcal{N})$.
- $I2(P_i) \equiv$ P_i is a weight consistent program

We now show that these invariants are maintained after every unfolding and folding step. This allows us to claim that the set of positive ground derivations of P_0 is identical to the set of positive ground derivations of program P_i.

Lemma 1 *If* $(\forall j \leq i\ I1(P_j))$ *holds, then* $\forall L \forall \mathcal{N}\ (\ L \rightsquigarrow_{P_{i+1}} \mathcal{N} \Rightarrow L \rightsquigarrow_{P_i} \mathcal{N}\ )$

Lemma 2 (Preserving Weak Weight Consistency) *Let* $P_0, ..., P_i, P_{i+1}$ *be an unfold/fold transformation sequence s.t.* $\forall 0 \leq j \leq i\ I1(P_j) \wedge I2(P_j)$. *Then, all positive ground derivations of* P_{i+1} *are weakly weight consistent.*

The proofs for both Lemma 1 and 2 follow by induction on the weight of positive ground derivations in P_{i+1}. The complete proofs appear in the appendix. We now establish the main theorem concerning the preservation of positive ground derivations in a transformation sequence.

Theorem 2 *Let* $P_0, P_1, \ldots$ *be a sequence of normal logic programs where* P_{i+1} *is obtained from* P_i *by applying unfolding (rule 1) or folding (rule 2). Then* $\forall i \geq 0\ I1(P_i) \wedge I2(P_i)$.

Proof : The proof proceeds by induction on i. For the base case, $I1(P_0)$ holds trivially, and $I2(P_0)$ holds because every positive ground derivation of P_0 is weakly weight consistent, and for any pair $(L, \mathcal{N})$ the smallest positive ground derivation $L \rightsquigarrow_{P_0} \mathcal{N}$ is strongly weight consistent.

For the induction step, we need to show $I1(P_{i+1}) \wedge I2(P_{i+1})$. By Lemma 1 we have $L \rightsquigarrow_{P_{i+1}} \mathcal{N} \Rightarrow L \rightsquigarrow_{P_i} \mathcal{N}$, and by Lemma 2 we know that all positive ground derivations of P_{i+1} are weakly weight consistent. We need to show that :

1. $L \rightsquigarrow_{P_i} \mathcal{N} \Rightarrow L \rightsquigarrow_{P_{i+1}} \mathcal{N}$, and
2. for any pair $(L, \mathcal{N})$ s.t. $L \rightsquigarrow_{P_{i+1}} \mathcal{N}$, there exists a strongly weight consistent derivation $L \rightsquigarrow \mathcal{N}$ in P_{i+1}.

Thus, it suffices to prove that for any pair $(L, \mathcal{N})$ s.t $L \rightsquigarrow_{P_i} \mathcal{N}$, there exists a strongly weight consistent derivation $L \rightsquigarrow_{P_{i+1}} \mathcal{N}$.

Consider a pair $(L, \mathcal{N})$ such that $L \rightsquigarrow_{P_i} \mathcal{N}$. Since P_i is weight consistent, therefore there exists a strongly weight consistent derivation $L \rightsquigarrow \mathcal{N}$ in P_i. Let this be called Dr. We now construct a strongly weight consistent derivation $Dr' \equiv L \rightsquigarrow_{P_{i+1}} \mathcal{N}$. Construction of Dr' proceeds by induction on the *weight of* $(L, \mathcal{N})$ *pairs*. The base case occurs when L is a negative literal, $\mathcal{N} = L$ and $w(L, \mathcal{N}) = 0$. We then trivially have the same derivation $L \rightsquigarrow \mathcal{N}$ in P_{i+1} as well. Otherwise if L is a positive literal, let C be the clause used at the root of Dr. Let $L{:-}\ L_1, \ldots, L_n$ be the ground instantiation of C used at the root of Dr. Since Dr is strongly weight consistent $w(L, \mathcal{N}) > w(L_l, \mathcal{N}_l)$ where $\mathcal{N}_l$ is the sequence of negative literals derived by L_l for all $1 \leq l \leq n$. Hence, we have strongly weight consistent derivations $L_l \rightsquigarrow_{P_{i+1}} \mathcal{N}_l$. We construct Dr' by considering the following cases :

Case 1: C is inherited from P_i to P_{i+1}

Dr' is constructed with the clause $L{:-}\ L_1, \ldots, L_n$ at the root and then appending the derivations $L_l \rightsquigarrow_{P_{i+1}} \mathcal{N}_l$ for all $1 \leq l \leq n$. This derivation Dr' is strongly weight consistent.

Case 2: C is unfolded.

Let the L_1 be the positive body literal of C that is unfolded. Let the clause used to resolve L_1 in the derivation Dr be C_1 and the ground instance of C_1 used be $L_1{:-}\ L_{1,1}, \ldots, L_{1,k}$. By definition of unfolding $L{:-}\ L_{1,1}, \ldots, L_{1,k}, L_2, \ldots, L_n$ is a ground instance of a clause C_1' in P_{i+1} with $\gamma_{lo}^{i+1}(C_1') = \gamma_{lo}^{i}(C) + \gamma_{lo}^{i}(C_1)$. Also, let $\mathcal{N}_{1,1}, \ldots, \mathcal{N}_{1,k}$ be the sequence of negative literals derived by $L_{1,1}, \ldots, L_{1,k}$ in Dr. Then, by strong weight consistency $w(L_{1,l}, \mathcal{N}_{1,l}) < w(L_1, \mathcal{N}_1) < w(L, \mathcal{N})$ for all $1 \leq l \leq k$. Thus we have strongly weight consistent derivations $L_{1,l} \rightsquigarrow_{P_{i+1}} \mathcal{N}_{1,l}$. We construct Dr' by applying $L{:-}\ L_{1,1}, \ldots, L_{1,k}, L_2, \ldots, L_n$ at the root and then appending the strongly weight consistent derivations $L_{1,l} \rightsquigarrow_{P_{i+1}} \mathcal{N}_{1,l}$ (for all $1 \leq l \leq k$) and $L_l \rightsquigarrow_{P_{i+1}} \mathcal{N}_l$ (for all $2 \leq l \leq n$). Since Dr is strongly weight consistent, therefore

$$
\begin{array}{ll}
 & w(L, \mathcal{N}) \geq \gamma_{lo}^{i}(C) + \sum_{1 \leq l \leq n} w(L_l, \mathcal{N}_l) \\
\text{and} & w(L_1, \mathcal{N}_1) \geq \gamma_{lo}^{i}(C_1) + \sum_{1 \leq l \leq k} w(L_{1,l}, \mathcal{N}_{1,l}) \\
\Rightarrow & w(L, \mathcal{N}) \geq \gamma_{lo}^{i+1}(C_1') + \sum_{1 \leq l \leq k} w(L_{1,l}, \mathcal{N}_{1,l}) + \sum_{2 \leq l \leq n} w(L_l, \mathcal{N}_l)
\end{array}
$$

This shows that Dr' is strongly weight consistent.

Case 3: C is folded

Let C (potentially with other clauses) be folded, using folder clause(s) from $P_j (j \leq i)$, to clause C' in P_{i+1}. Assume that $L_1, \ldots, L_k$ are the instances of the body literals of C which are folded. Then, C' must have a ground instance of the form $L : -B, L_{k+1}, \ldots, L_n$, where $B{:-}\ L_1, \ldots, L_k$ is a ground instance of a folder clause D in P_j. Since, we have derivations $L_l \rightsquigarrow_{P_i} \mathcal{N}_l$ for all $1 \leq l \leq k$, therefore by $I1(P_i) \wedge I1(P_j)$ there exist derivations $L_l \rightsquigarrow_{P_j} \mathcal{N}_l$. Then, there exists a derivation $B \rightsquigarrow_{P_j} \mathcal{N}_B$ where $\mathcal{N}_B$ is obtained by appending the sequences $\mathcal{N}_1, \ldots, \mathcal{N}_k$. Since P_j is a weight consistent program, this derivation must be weakly weight consistent, and therefore $w(B, \mathcal{N}_B) \leq \gamma_{hi}^{j}(D) + \sum_{1 \leq l \leq k} w(L_l, \mathcal{N}_l)$. By strong weight consistency of Dr, we have

$$
\begin{aligned}
w(L, \mathcal{N}) &\geq \gamma_{lo}^{i}(C) + \sum_{1 \leq l \leq k} w(L_l, \mathcal{N}_l) + \sum_{k+1 \leq l \leq n} w(L_l, \mathcal{N}_l) \\
&\geq \gamma_{lo}^{i}(C) + w(B, \mathcal{N}_B) - \gamma_{hi}^{j}(D) + \sum_{k+1 \leq l \leq n} w(L_l, \mathcal{N}_l) \qquad \cdots\cdots (*) \\
&> w(B, \mathcal{N}_B) \text{ (by condition (5) of folding)}
\end{aligned}
$$

Thus there exists a strongly weight consistent derivation $B \rightsquigarrow_{P_{i+1}} \mathcal{N}_B$. We construct Dr' with $L{:-}\ B, L_{k+1}, \ldots, L_n$ at the root and then appending the strongly weight consistent derivations $B \rightsquigarrow_{P_{i+1}} \mathcal{N}_B, L_{k+1} \rightsquigarrow_{P_{i+1}} \mathcal{N}_{k+1}, \ldots, L_n \rightsquigarrow_{P_{i+1}} \mathcal{N}_n$. To show that Dr' is strongly weight consistent, note that $\gamma_{lo}^{i+1}(C') \leq \gamma_{lo}^{i}(C) - \gamma_{hi}^{j}(D)$

since C and D are folded and folder clauses. Combining this with (*),

$$w(L, \mathcal{N}) \geq \gamma_{lo}^{i+1}(C') + w(B, \mathcal{N}_B) + \sum_{k+1 \leq l \leq n} w(L_l, \mathcal{N}_l)$$

This completes the proof. □

Thus we have shown that all positive ground derivations are preserved at every step of our transformation. Now we show how our notion of positive ground derivations directly corresponds to the notion of semantic kernel. Intuitively, this connection is clear, since a clause in the semantic kernel of program P is derived by repeatedly resolving the positive body literals of a ground instance of a clause in P until the body contains only negative literals. Formally, we prove that :

Theorem 3 *Let P be a normal logic program and $A, B_1, \ldots, B_n (n \geq 0)$ be ground atoms in the Herbrand base of P. Let $\mathcal{N}$ be the sequence $\neg B_1, \ldots, \neg B_n$. Then, A derives $\mathcal{N}$ in P iff $(A{:-}\ \mathcal{N}) \in SK(P)$*

Proof Sketch: We prove $A \rightsquigarrow_P \mathcal{N} \Rightarrow (A{:-}\ \mathcal{N}) \in SK(P)$ by well-founded induction on the weight (*i.e.* the number of internal nodes, refer definition 6) of the derivation $A \rightsquigarrow_P \mathcal{N}$. The proof for $(A{:-}\ \mathcal{N}) \in SK(P) \Rightarrow A \rightsquigarrow_P \mathcal{N}$ follows by fixed-point induction. □

We can now prove that the semantic kernel is preserved across any unfold/fold transformation sequence.

Corollary 3 (Preservation of Semantic Kernel) *Suppose $P_0, \ldots, P_n$ is a sequence of normal logic programs where P_{i+1} is obtained from P_i by unfolding (Rule 1) or folding (Rule 2). Then $\forall 0 < i \leq n\ SK(P_i) = SK(P_0)$.*

Proof: We prove that $SK(P_0) = SK(P_i)$ for any arbitrary i. By Theorem 2 we know that $A \rightsquigarrow_{P_0} \mathcal{N} \Leftrightarrow A \rightsquigarrow_{P_i} \mathcal{N}$ for any ground atom A and sequence of ground negative literals $\mathcal{N}$. Then, using Theorem 3 we get $(A{:-}\ \mathcal{N}) \in SK(P_0) \Leftrightarrow (A{:-}\ \mathcal{N}) \in SK(P_i)$. Thus, $SK(P_0) = SK(P_i)$. □

Following Theorem 1 and Corollary 3 we have:

Theorem 4 (Correctness of Transformations) *Let $P_0, \ldots, P_n$ be a sequence of normal logic programs where P_{i+1} is obtained from P_i by an application of unfolding (Rule 1) or folding (Rule 2). Then, for all $0 < i \leq n$ we have*

(1) If P_0 is a definite program, then P_0 and P_i have the same least Herbrand Model.

(2) If P_0 is a stratified program, then P_0 and P_i have the same perfect model.

(3) P_0 and P_i have the same well-founded model.

(4) P_0 and P_i have the same stable model(s).

(5) P_0 and P_i have the same set of partial stable models.

(6) P_0 and P_i have the same stable theory semantics.

Thus we have proved the correctness of any interleaved application of our unfold/fold transformation with respect to the different semantics for normal logic programs.

4. Discussions

Goal Replacement The transformation system presented in this paper can be extended to incorporate a *goal replacement* rule which allows the replacement of a conjunction of atoms in the body of a clause with another semantically equivalent conjunction of atoms provided certain conditions are satisfied (which ensure preservation of weight consistency). In future, it would be interesting to study how we can perform multiple replacements simultaneously without compromising correctness (as discussed in [3]).

Motivation : Verification of Parameterized Systems The motivation of the unfold/fold transformation system reported here is verification of parameterized concurrent systems. Parameterized systems are infinite families of finite state concurrent systems *e.g.* an n-process token ring for any n. Proving temporal properties of parameterized systems requires reasoning about each of the members of the infinite family, which can be accomplished by induction. Recently, we have used unfold/fold transformations of definite logic programs [19] to verify liveness and safety properties of parameterized systems [18]. However, temporal properties containing both greatest and least fixed point operators cannot be encoded as a definite logic program. A trivial example of such a property is : "it is the case that *always* if an input event occurs (infinitely often) then an output event *eventually* occurs (infinitely often)". This property contains the *always* operator (defined as a greatest fixed point) and the *eventually* operator (defined as a least fixed point). This property is commonly used in verification of hardware circuits. To use program transformations to construct induction proofs of such temporal properties we need unfold/fold transformation systems for normal logic programs. This indeed has been a motivation for the work reported in this paper.

Implications of the Correctness Proof Apart from the transformation system, the details of the underlying correctness proof reveal certain interesting issues. Note that we showed that positive ground derivations form the operational counterpart to semantic kernels. This result, which makes explicit an idea in the proof of Aravindan and Dung [1], enables the correctness proof to be completed by connecting the other two steps: an operational first step, where the measure consistency technique is used to show the preservation of positive ground derivations and the final model-theoretic step that applies the results of Dung and Kanchanasut [5] relating semantic kernels to various semantics for normal logic programs.

Interestingly, by its very nature, the notion of semantic kernel cannot be used in proving operational equivalences such as finite failure and computed answer sets. The important task then is to formulate a suitable operational notion that plays the role of semantic kernel in the correctness proofs with respect to these equivalences.

Acknowledgments

The authors would like to thank the anonymous referees for helpful suggestions. This work was partially supported by NSF grants CCR-9711386, CCR-9876242, CDA-9805735 and EIA-9705998.

References

1. C. Aravindan and P.M. Dung. On the correctness of unfold/fold transformations of normal and extended logic programs. *Journal of Logic Programming*, pages 295–322, 1995.
2. A. Bossi, N. Cocco, and S. Dulli. A method of specializing logic programs. *ACM TOPLAS*, pages 253–302, 1990.
3. A. Bossi, N. Cocco, and S. Etalle. Simultaneous replacement in normal programs. *Journal of Logic and Computation*, 6(1):79–120, February 1996.
4. D. Boulanger and M. Bruynooghe. Deriving unfold/fold transformations of logic programs using extended OLDT-based abstract interpretation. *Journal of Symbolic Computation*, pages 495–521, 1993.
5. P.M. Dung and K. Kanchanasut. A fixpoint approach to declarative semantics of logic programs. In *North American Conference on Logic Programming*, pages 604–625, 1989.
6. P. A. Gardner and J. C. Shepherdson. Unfold/fold transformations of logic programs. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 565–583. MIT Press, Cambridge, MA, 1991.
7. A. Van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
8. M. Gelfond and V. Lifshitz. The stable model semantics for logic programming. In *International Conference and Symposium on Logic Programming*, pages 1070–1080, 1988.
9. M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. In *PLILP, LNCS 844*, pages 340–354, 1994.
10. T. Kanamori and H. Fujita. Formulation of Induction Formulas in Verification of Prolog Programs. In *International Conference on Automated Deduction (CADE)*, pages 281–299, 1986.
11. T. Kanamori and H. Fujita. Unfold/fold transformation of logic programs with counters. In *USA-Japan Seminar on Logics of Programs*, 1987.
12. M. Leuschel, D. De Schreye, and A. De Waal. A conceptual embedding of folding into partial deduction : Towards a maximal integration. In *Joint International Conference and Symposium on Logic Programming*, pages 319–332, 1996.
13. J.W. Lloyd. *Foundations of Logic Programming, 2nd Ed.* Springer-Verlag, 1993.
14. M. J. Maher. Correctness of a logic program transformation system. Technical report, IBM T.J. Watson Research Center, 1987.
15. M. J. Maher. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science*, 110:377–403, 1993.
16. A. Pettorossi and M. Proietti. *Transformation of logic programs*, volume 5 of *Handbook of Logic in Artificial Intelligence*, pages 697–787. Oxford University Press, 1998.
17. A. Pettorossi and M. Proietti. Perfect model checking via unfold/fold transforma-

tions. In *Computational Logic, LNCS 1861*, pages 613–628, 2000.

18. A. Roychoudhury, K. Narayan Kumar, C. R. Ramakrishnan, I.V. Ramakrishnan, and S. A. Smolka. Verification of parameterized systems using logic program transformations. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS), LNCS 1785*, 2000.
19. A. Roychoudhury, K. Narayan Kumar, C.R. Ramakrishnan, and I.V. Ramakrishnan. A parameterized unfold/fold transformation framework for definite logic programs. In *Principles and Practice of Declarative Programming (PPDP), LNCS 1702*, pages 396–413, 1999.
20. D. Sands. Total correctness by local improvement in the transformation of functional programs. *ACM TOPLAS*, 18(2):175–234, 1996.
21. T. Sato. Equivalence-preserving first-order unfold/fold transformation systems. *Theoretical Computer Science*, 105:57–84, 1992.
22. H. Seki. Unfold/fold transformation of stratified programs. *Theoretical Computer Science*, pages 107–139, 1991.
23. H. Seki. Unfold/fold transformation of general logic programs for well-founded semantics. *Journal of Logic Programming*, pages 5–23, 1993.
24. H. Tamaki and T. Sato. Unfold/fold transformations of logic programs. In *Proceedings of International Conference on Logic Programming*, pages 127–138, 1984.
25. F. Toni and R. Kowalski. An argumentation-theoretic approach to logic program transformation. In M. Proietti, editor, *Logic Program Synthesis and Transformation, Proceedings LoPSTr '95, Utrecht, The Netherlands.*, Lecture Notes in Computer Science 1048, pages 61–75. Springer-Verlag, 1996.

Appendix

Lemma 1 $(\forall j \leq i\ I1(P_j)) \Rightarrow (L \rightsquigarrow_{P_{i+1}} \mathcal{N} \Rightarrow L \rightsquigarrow_{P_i} \mathcal{N})$

Proof : We consider a positive ground derivation $Dr \equiv L \rightsquigarrow \mathcal{N}$ in P_{i+1}, and construct a derivation $L \rightsquigarrow \mathcal{N}$ in P_i by induction on the weight (*i.e* the number of internal nodes) of Dr. If L is a negative literal, then the result is obvious since then $\mathcal{N} = L$, *i.e.* Dr is empty. Otherwise, let $L{:-}\ L_1, ..., L_n$ be the ground instance of a clause $C \in P_{i+1}$ used at the root of Dr. For all $1 \leq l \leq n$ let $\mathcal{N}_l$ be the sequence of negative ground literals derived from L_l in the derivation Dr. Then, by induction hypothesis, we have derivations $L_l \rightsquigarrow_{P_i} \mathcal{N}_l$ for all $1 \leq l \leq n$. We consider three possible cases.

Case 1 : C was inherited from P_i

$L \rightsquigarrow_{P_i} \mathcal{N}$ is constructed by applying $L{:-}\ L_1, ..., L_n$ at the root and then resolving $L_1, ..., L_n$ by using the derivations $L_l \rightsquigarrow_{P_i} \mathcal{N}_l$.

Case 2 : C was obtained by *unfolding*

Then, without loss of generality, P_i contains clauses with ground instantiations $L{:-}\ B, L_{k+1}, ..., L_n$ and $B{:-}\ L_1, ..., L_k$. We construct derivation $L \rightsquigarrow_{P_i} \mathcal{N}$ by first applying the clause $L{:-}\ B, L_{k+1}, ..., L_n$ then the clause $B{:-}\ L_1, ...L_k$ and then resolving $L_1, ..., L_n$ by using $L_l \rightsquigarrow_{P_i} \mathcal{N}_l$ for all $1 \leq l \leq n$.

Case 3: C was obtained by *folding*

Let L_1 be the atom introduced by folding, and let $P_j (j \leq i)$ be the program from which folder clauses were picked. By induction hypothesis, we have derivations

$L_l \rightsquigarrow_{P_i} \mathcal{N}_l$ for all $1 \leq l \leq n$. Then, $L_1 \rightsquigarrow_{P_i} \mathcal{N}_1$ and hence $L_1 \rightsquigarrow_{P_0} \mathcal{N}_1$ (by $I1(P_i)$) and hence $L_1 \rightsquigarrow_{P_j} \mathcal{N}_1$ (by $I1(P_j)$). Let a derivation $L_1 \rightsquigarrow \mathcal{N}_1$ in P_j be called Dr_j and let $L_1 \text{:-} L_{1,1}, ..., L_{1,k}$ be the ground clause used at the root of Dr_j. Then, by conditions (3) and (4) of the folding transformation, there must be a ground instantiation of a clause in P_i (one of the folded clauses) of the form $L \text{:-} L_{1,1}, ..., L_{1,k}, L_2, ..., L_n$. Also, by $I1(P_j)$, we must have derivations $L_{1,l} \rightsquigarrow_{P_0} \mathcal{N}_{1,l}$ for all $1 \leq l \leq k$, where $\mathcal{N}_{1,l}$ is the sequence of negative literals derived from $L_{1,l}$ in Dr_j. Then, by $I1(P_i)$, we must have derivations $L_{1,l} \rightsquigarrow_{P_i} \mathcal{N}_{1,l}$ for all $1 \leq l \leq k$. We can therefore construct $L \rightsquigarrow_{P_i} \mathcal{N}$ by applying the clause $L \text{:-} L_{1,1}, ..., L_{1,k}, L_2, ..., L_n$ at the root and then resolving $L_{1,1}, ..., L_{1,k}, L_2, ..., L_n$ by using $L_{1,l} \rightsquigarrow_{P_i} \mathcal{N}_{1,l}$ ($\forall\ 1 \leq l \leq k$) and $L_l \rightsquigarrow_{P_i} \mathcal{N}_l$ ($\forall\ 2 \leq l \leq n$). □

Lemma 2 Let $P_0, P_1, \ldots, P_i, P_{i+1}$ be an unfold/fold transformation sequence s.t. $\forall 0 \leq j \leq i\ I1(P_j) \wedge I2(P_j)$. Then, all positive ground derivations of P_{i+1} are weakly weight consistent.

Proof: The proof proceeds by induction on the weight (*i.e* number of internal nodes) of positive ground derivations in P_{i+1}. Let $Dr \equiv L \rightsquigarrow \mathcal{N}$ be a derivation in P_{i+1} and let $L \text{:-} L_1, \ldots L_n$ be the ground instance of a clause $C \in P_{i+1}$ that is used at the root of Dr. Then, the sub-derivations $L_l \rightsquigarrow_{P_{i+1}} \mathcal{N}_l$ (for all $1 \leq l \leq n$) of Dr are weakly weight consistent by induction hypothesis, where $\mathcal{N}_1, \ldots, \mathcal{N}_n$ are the sequence of negative literals derived from $L_1, \ldots, L_n$ in derivation Dr. Hence it suffices to show that $w(L, \mathcal{N}) \leq \gamma_{hi}^{i+1}(C) + \sum_{1 \leq l \leq n} w(L_l, \mathcal{N}_l)$. We consider three cases.

Case 1 : C was inherited from P_i.

Since $L_l \rightsquigarrow_{P_{i+1}} \mathcal{N}_l$, therefore by lemma 1, we have $L_l \rightsquigarrow_{P_i} \mathcal{N}_l$. Thus $L \text{:-} L_1, \ldots, L_n$ is used at the root of a positive ground derivation in P_i. Since P_i is a weight consistent program and $\gamma_{hi}^{i+1}(C) = \gamma_{hi}^{i}(C)$, the result follows.

Case 2: C was obtained by unfolding.

Let $L_1, \ldots, L_k$ be the body literals of C which were introduced through unfolding. Then, without loss of generality, P_i contains clauses C and C'' which have ground instantiations $L \text{:-} B, L_{k+1}, ..., L_n$ and $B \text{:-} L_1, ..., L_k$. Also, by lemma 1, we have $L_l \rightsquigarrow_{P_i} \mathcal{N}_l$ (for all $1 \leq l \leq n$). Then, the above mentioned two ground instances of C' and C'' are used in some positive ground derivation of P_i. Since P_i is weight consistent, we have :

$$w(L, \mathcal{N}) \ \leq\ \gamma_{hi}^{i}(C') + w(B, \mathcal{N}_B) + \sum_{k+1 \leq l \leq n} w(L_l, \mathcal{N}_l)$$

$$w(B, \mathcal{N}_B) \ \leq\ \gamma_{hi}^{i}(C'') + \sum_{1 \leq l \leq k} w(L_l, \mathcal{N}_l)$$

where $\mathcal{N}_B$ is the sequence of negative literals derived by B, *i.e.* $\mathcal{N}_B$ is obtained by appending $\mathcal{N}_1, \ldots, \mathcal{N}_k$. Then combining the above inequalities we have $w(L, \mathcal{N}) \leq \gamma_{hi}^{i+1}(C) + \sum_{1 \leq l \leq n} w(L_l, \mathcal{N}_l)$ since we know that $\gamma_{hi}^{i+1}(C) = \gamma_{hi}^{i}(C') + \gamma_{hi}^{i}(C'')$ by definition of unfolding.

Case 3: C was obtained by folding.

Let L_1 be the atom introduced by folding, and let $P_j (j \leq i)$ be the program from which folder clauses were picked. By lemma 1 we have $L_1 \rightsquigarrow_{P_i} \mathcal{N}_1$. Again since $I1(P_i) \wedge I1(P_j)$, therefore $L_1 \rightsquigarrow_{P_j} \mathcal{N}_1$. As P_j is a weight consistent program, therefore there exists a strongly weight consistent derivation $L_1 \rightsquigarrow \mathcal{N}_1$ in P_j. Let this derivation be called Dr_j. Let the clause used at the root of Dr_j be D' and let the ground instance of D' used at the root of Dr_j be $L_1 \text{:-} L_{1,1}, \ldots, L_{1,k}$. Then, by definition of strong weight consistency

$$w(L_1, \mathcal{N}_1) \geq \gamma^j_{lo}(D') + \sum_{1 \leq l \leq k} w(L_{1,l}, \mathcal{N}_{1,l})$$

where $\mathcal{N}_{1,1}, \ldots, \mathcal{N}_{1,k}$ is the sequence of negative literals derived by $L_{1,1}, \ldots, L_{1,k}$ in Dr_j. But D' must be a folder clause by definition of folding. Hence there must be a clause C' in P_i with a ground instance $L \text{:-} L_{1,1}, \ldots, L_{1,k}, L_2, \ldots, L_n$ (this is the folded clause corresponding to D'). Now, by lemma 1 we have $L_l \rightsquigarrow_{P_i} \mathcal{N}_l$ for all $2 \leq l \leq n$. Also since $I1(P_j) \wedge I1(P_i)$ therefore $L_{1,l} \rightsquigarrow_{P_i} \mathcal{N}_{1,l}$ for all $1 \leq l \leq k$. Therefore the ground clause $L \text{:-} L_{1,1}, \ldots, L_{1,k}, L_2, \ldots, L_n$ appears at the root of a positive ground derivation in P_i. As P_i is a weight consistent program, this derivation must be weakly weight consistent. Hence

$$\begin{aligned}
w(L, \mathcal{N}) &\leq \gamma^i_{hi}(C') + \sum_{1 \leq l \leq k} w(L_{1,l}, \mathcal{N}_{1,l}) + \sum_{2 \leq l \leq n} w(L_l, \mathcal{N}_l) \\
&\leq \gamma^i_{hi}(C') - \gamma^j_{lo}(D') + w(L_1, \mathcal{N}_1) + \sum_{2 \leq l \leq n} w(L_l, \mathcal{N}_l) \\
&\leq \gamma^i_{hi}(C') - \gamma^j_{lo}(D') + \sum_{1 \leq l \leq n} w(L_l, \mathcal{N}_l) \\
&\leq \gamma^{i+1}_{hi}(C) + \sum_{1 \leq l \leq n} w(L_l, \mathcal{N}_l)
\end{aligned}$$

This completes the proof. □