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

Abhik Roychoudhury[1], K. Narayan Kumar[1,2], C.R. Ramakrishnan[1], and I.V. Ramakrishnan[1]

[1] Dept. of Computer Science, SUNY Stony Brook, Stony Brook, NY 11794, USA.
{abhik,kumar,cram,ram}@cs.sunysb.edu
[2] Chennai Mathematical Institute, 92 G.N. Chetty Road, Chennai, India.
kumar@smi.ernet.in

Abstract. Unfold/fold transformation systems for logic programs have been extensively investigated. Existing unfold/fold transformation systems for normal logic programs allow only Tamaki-Sato style folding using clauses from a previous program in the transformation sequence: i.e., they fold using a single, non-recursive clause. 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 semantics of negation including the well-founded model and stable model semantics.

1 Introduction

Unfold/fold transformation systems for logic programs have been used for and automated deduction [6, 15], and program specialization and optimization [2, 3, 8, 13]. Normal logic programs consist of definitions of the form $A{:-}\ \phi$ where A is an atom and ϕ is a boolean formula 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.

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 [12].

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*.

An unfold/fold transformation system for definite logic programs was first described in a seminal paper by Tamaki and Sato [18]. 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 [10] 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 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 [1, 11, 16, 17] are either extensions of Maher's reversible transformation system [10] or the original Tamaki-Sato system [18]. Even for definite logic programs, irreversible transformations of programs were, until recently, restricted to either folding using non-recursive clauses (see [5]) or a single recursive clause (see [7, 19]). In [14] we proposed a transformation framework for definite logic programs which generalized the above systems by permitting folding using multiple recursive clauses. Construction of such a general transformation system for *normal* logic programs has remained open. Below, we describe a solution to this problem.

Overview of the 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). The transformations of [14] associates *counters* with program clauses (*a la* Kanamori and Fujita [7]) to determine the applicability of fold and unfold transformations. In this paper, we extend this book-keeping to accommodate negative literals. We show that this extension is sufficient to guarantee that the resulting transformation system preserves a variety of semantics 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 [4] that preserving the *semantic kernel* of a program is sufficient to guarantee the preservation of the different semantics for negation listed above. 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).

Program P_0

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

Program P_1

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

Program P_2

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

Fig. 1. Example of an unfold/fold transformation sequence

The unfold transformation, following that in [1, 11, 16, 17], does not unfold a negative literal. As a result, the negative literals may participate in a transformation, but are by themselves unchanged. 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.

This 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 observation indicates how a transformation system originally designed for definite logic programs (such as the one we proposed in [14]) can be readily adapted for normal logic programs.

2 The Transformation System

Below we present our *unfold* and *fold* transformations for normal logic programs. In the following we assume familiarity with the standard notions of terms, substitutions, unification, atoms, literals. 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.

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 [7]; the separation of *hi* and *lo* permits us to store estimates of the counter values in the presence of disjunctive folding.

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 $C_l \equiv A$:– $L_{l,1}, \ldots, L_{l,n_l}, L'_1, \ldots, L'_n$ and $D_l \equiv B_l$:– $K_{l,1}, \ldots, K_{l,n_l}$. 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. $\forall 1 \leq l \leq m$ σ_l substitutes the internal variables 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. □

It must be noted that the conditions in the above definitions are present in one form or another in every irreversible fold/unfold transformation system. In fact, the first four conditions of the folding rule are essential for any correct application of folding, including reversible folding. Finally, the fifth condition of the rule is also present, in different forms, in every irreversible system, to govern sound applications of folding following a series of unfoldings.

An Example: The following example (derived from [5]) illustrates the use of our basic unfold/fold transformation system.

```
C1 : in_position(X,L) :- in_odd(X,L), ¬ even(X).  (1,1)
C2 : in_position(X,L) :- in_even(X,L), ¬ odd(X).  (1,1)
C3 : in_odd(X,[X|L]).                            (1,1)
C4 : in_odd(X,[Y,Z|L]) :- in_odd(X,L).           (1,1)
C5 : in_even(X,[Y,X|L]).                         (1,1)
C6 : 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. The `odd/1` and `even/1` predicates are encoded in the usual way and are not shown.
Unfolding `in_odd(X,L)` in clause C_1 we get the following clauses for `in_position/1`

```
C7 : in_position(X,[X|L]) :- ¬ even(X).          (2,2)
C8 : in_position(X,[Y,Z|L]) :- in_odd(X,L), ¬ even(X).  (2,2)
```

Unfolding `in_even(X,L)` in C_2 yields the following clauses for `in_position/1`:

```
C9 : in_position(X,[Y,X|L]) :- ¬ odd(X).         (2,2)
C10 : 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 clauses for `in_position/1`.

```
C7 : in_position(X,[X|L]) :- ¬ even(X).          (2,2)
C9 : in_position(X,[Y,X|L]) :- ¬ odd(X).         (2,2)
C11 : 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 [1, 16, 17] nor in reversible transformation systems [11].

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 [14], we can make the counters range over use a tuple of integers, and obtain a system that is strictly more powerful than the existing Tamaki-Sato-style systems [18, 19, 7, 16, 17, 5, 1]. The construction parallels that of the SCOUT system in [14]; 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 introduce the notion of positive ground derivations and show that it is preserved by the transformations. Secondly, we show that preserving positive ground derivations is equivalent to preserving semantic kernel [4]. 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, stable model, partial stable model, and stable theory semantics. We begin with a review of semantic kernels.

3.1 Semantic Kernel of a Program

Definition 1 (Quasi-Interpretation) [4, 1] *Given a normal logic program P, a quasi-interpretation of 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 2 *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 logical consequence operator T_P [9]. The semantic kernel of the program P is defined in terms of S_P as:

Definition 3 (Semantic Kernel) [4, 1] *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.,*

$$SK(P) = \bigcup_{n \in \omega} SK^n(P) \text{ where } SK^0(P) = \phi \text{ and } SK^{n+1}(P) = S_P(SK^n(P))$$

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 [Aravindan and Dung] *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 $\forall 0 \leq i < n$ 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 introduce the following notion of a positive ground derivation:

Definition 4 (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 ground instance of a clause in program P where $L_1, \ldots, L_n$ are the children of A in T.*

Thus, consider any positive ground derivation T in program P. Let the root of T be labeled with the ground literal L and let $\mathcal{N}$ be the *sequence of negative literals* 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 emph$L$ derives $\mathcal{N}$ in P, and denote such derivations by $L \rightsquigarrow_P \mathcal{N}$ (and $L \rightsquigarrow \mathcal{N}$ if P is obvious from the context). We overload this notation, often denoting existance of such derivations also by $L \rightsquigarrow_P \mathcal{N}$. 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 $L \rightsquigarrow L$.

We show that the transformations (rules 1 and 2) preserve the semantic kernel in two steps. First, we prove that the set of positive ground derivations is preserved across any transformation sequence. Then, we show that for any normal logic program P, every positive ground derivation in P corresponds to a clause in the semantic kernel $SK(P)$ and vice-versa. To prove that positive ground derivations are preserved, we will use the following definitions.

Definition 5 (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 6 (Weight of a pair) *Let $P_0, P_1, \ldots, P_n$ 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 $L \rightsquigarrow_{P_0} \mathcal{N}$ (the initial program).

Definition 7 (Weakly weight-consistent positive ground derivation) *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 sequence of negative literals derived from $A, L_1, ..., L_k$ in this derivation.*

Definition 8 (Strongly weight-consistent positive ground derivation) *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 sequence of negative literals derived from $A, L_1, ..., L_k$ in this derivation.

Definition 9 (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 derives $\mathcal{N}$ in $P_0 \Leftrightarrow L$ derives $\mathcal{N}$ in P_i).
- $I2(P_i) \equiv$ 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 the initial program P_0 is identical to the set of positive ground derivations of program P_i.

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

Lemma 2 (Preserving Weak Weight Consistency) *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.*

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 (Preservation of Positive Ground derivations) *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 *(i)* $L \rightsquigarrow_{P_i} \mathcal{N} \Rightarrow L \rightsquigarrow_{P_{i+1}} \mathcal{N}$, and *(ii)* 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 use 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

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

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

$$\begin{aligned} w(L,\mathcal{N}) &\geq \gamma^i_{lo}(C) + \sum_{1\leq l\leq k} w(L_l,\mathcal{N}_l) + \sum_{k+1\leq l\leq n} w(L_l,\mathcal{N}_l) \\ &\geq \gamma^i_{lo}(C) + w(B,\mathcal{N}_B) - \gamma^j_{hi}(D) + \sum_{k+1\leq l\leq n} w(L_l,\mathcal{N}_l) \qquad \cdots\cdots(*) \\ &\geq (\gamma^i_{lo}(C) - \gamma^j_{hi}(D)) + w(B,\mathcal{N}_B) + \text{No. positive literals in } L_{k+1},\ldots,L_n \\ &> w(B,\mathcal{N}_B) \text{ (by condition (5) of folding)} \end{aligned}$$

Hence there exists a strongly weight consistent derivation $B \rightsquigarrow_{P_{i+1}} \mathcal{N}_B$. We now construct Dr' with $L{:-}\ B, L_{k+1},\ldots,L_n$ at the root and then appending below 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 $\gamma^{i+1}_{lo}(C') \leq \gamma^i_{lo}(C) - \gamma^j_{hi}(D)$ since C and D are folded and folder clauses. Combining this with (*),

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

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 (Positive ground derivations and Semantic kernel) *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 $\mathcal{N}$ denote 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 strong induction on the weight (*i.e.* the number of internal nodes, refer definition 5) in 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. The complete proof appears in the appendix. □

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

Corollary 3 (Preservation of Semantic Kernel) *Let $P_0, P_1,\ldots,P_n$ 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 $\forall 0 \leq i < n\ SK(P_i) = SK(P_0)$.*

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 Unfolding/Folding) *Let $P_0, P_1, \ldots, P_n$ 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 $0 \leq i < n$ we have*
(1) If P_0 is a definite logic 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 semantics.
(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.

4 Discussion

In this paper we have presented an unfold/fold transformation system, which to the best of our knowledge, is the first to permit folding in the presence of recursion, disjunction, as well as negation. Apart from the transformation system, the details of the underlying correctness proof reveal certain interesting aspects generic to such transformation systems.

First of all, our proof exploits a degree of modularity that is inherent in the unfold/fold transformations for logic programs. Consider a modular decomposition of a definite logic program where each predicate is fully defined in a single module. Each module has a set of "local" predicates defined in the current module and a set of "external" predicates used (and not defined) in the current module. It is easy to see that Lemma 1, 2 and Theorem2 can be modified to show that unfold/fold transformations preserve the set of *local ground derivations* of a program. We say that $A \rightsquigarrow B_1, B_2, \ldots, B_n$ is a local ground derivation (analogous to a positive ground derivation), if each B_i contains an external predicate, and there is a proof tree rooted at A whose leaves are labeled with $B_1, \ldots, B_n$ (apart from true). Consequently, transformations of a normal logic program P, can be simulated by an equivalent positive program module Q obtained by replacing negative literals in P with new positive external literals. The newly introduced literals can be appropriately defined in a separate module. Thus any transformation system for definite logic programs that preserves local ground derivations also preserves the semantic kernels of normal logic programs.

Secondly, 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 [4] relating semantic kernels to various semantics for normal logic programs.

Semantic kernel is a fundamental concept in the study of model-based semantics. By it very nature, however, semantic kernels 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.

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. 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.
4. P.M. Dung and K. Kanchanasut. A fixpoint approach to declarative semantics of logic programs. *Proceeding of North American Conference on Logic Programming*, 1:604–625, 1989.
5. M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. In *Proceedings of PLILP, LNCS 844*, pages 340–354, 1994.
6. T. Kanamori and H. Fujita. Formulation of Induction Formulas in Verification of Prolog Programs. *Proceedings of International Conference on Automated Deduction (CADE)*, pages 281–299, 1986.
7. T. Kanamori and H. Fujita. Unfold/fold transformation of logic programs with counters. In *USA-Japan Seminar on Logics of Programs*, 1987.
8. 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.
9. J.W. Lloyd. *Foundations of Logic Programming, 2nd Ed.* Springer-Verlag, 1993.
10. M. J. Maher. Correctness of a logic program transformation system. Technical report, IBM T.J. Watson Research Center, 1987.
11. M. J. Maher. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science*, 110:377–403, 1993.
12. 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.
13. A. Pettorossi, M. Proietti, and S. Renault. Reducing nondeterminism while specializing logic programs. In *Proceedings of POPL*, pages 414–427, 1997.
14. A. Roychoudhury, K. Narayan Kumar, C.R. Ramakrishnan, and I.V. Ramakrishnan. A parameterized unfold/fold transformation framework for definite logic programs. *To appear in proocedings of Principles and Practice of Declarative Programming (PPDP)*, 1999.
15. A. Roychoudhury, K. Narayan Kumar, C.R. Ramakrishnan, and I.V. Ramakrishnan. Proofs by program transformations. *To appear in proceedings of Logic-based Program Synthesis and Transformation (LOPSTR)*, 1999.
16. H. Seki. Unfold/fold transformation of stratified programs. *In Theoretical Computer Science*, pages 107–139, 1991.
17. H. Seki. Unfold/fold transformation of general logic programs for well-founded semantics. *In Journal of Logic Programming*, pages 5–23, 1993.

18. H. Tamaki and T. Sato. Unfold/fold transformations of logic programs. In *Proceedings of International Conference on Logic Programming*, pages 127–138, 1984.
19. H. Tamaki and T. Sato. A generalized correctness proof of the unfold/ fold logic program transformation. Technical report, Ibaraki University, Japan, 1986.