ON THE CORRECTNESS OF UNFOLD/FOLD TRANSFORMATION OF NORMAL AND EXTENDED LOGIC PROGRAMS

Chandrabose ARAVINDAN AND Phan Minh DUNG[1]

We show that the framework for unfold/fold transformation of logic programs, first proposed by Tamaki & Sato and later extended by various researchers, preserves various non-monotonic semantics of normal logic programs, esp. preferred extension, partial stable models, regular model, and stable theory semantics. The primary aim of this research is to adopt an uniform approach for every semantics of normal programs, and that is elegantly achieved through the notion of semantic kernel. Later, we show that this framework can also be applied to extended logic programs, preserving the answer set semantics.

§1. Introduction

The use of unfold/fold transformation in synthesis of logic programs has been realized as early as 1977 [11,23] and a framework for this transformation was proposed in 1984 by Tamaki and Sato [44]. Later, the relationship between unfold/fold transformation and partial deduction has been studied, indicating that unfold/fold transformation can also be used for program specialization and optimization [28,43]. The primary requisite of an unfold/fold transformation system is its ***correctness***: it should preserve the meaning of the original program, i.e., the original and transformed programs should be equivalent wrt the chosen semantics.

[1]Address for correspondence: Computer Science Program, School of Advanced Technologies, Asian Institute of Technology, P.O. Box 2754, Bangkok 10501, Thailand.
Internet: {arvind,dung}@cs.ait.ac.th

The original unfold/fold transformation system of Tamaki and Sato [44] is meant for definite programs, and is shown to be correct wrt the least Herbrand model semantics of a definite program. Later, the same system was shown to preserve answer substitutions, by Kawamura and Kanamori [27]. The correctness of the transformation technique, when applied to normal programs, is closely related to how we understand the negation in a program. So far, numerous non-monotonic approaches to handle negation have been proposed, and the relationship among them is well studied [e.g. 2,10,12,13,16,17,20,25,26,30,37,38,45,47,48]. Often, we find researches that study the correctness of an unfold/fold transformation system wrt a *particular* semantics of negation. For example, Maher showed that the transformation system is correct wrt perfect model semantics in [31,33] and studied the correctness wrt stable model and well-founded model semantics in [32]; in [19] Gardner and Shepherdson showed that it preserves the Clark's completion semantics; Seki extended the framework and studied the correctness wrt stable model semantics in [41], wrt perfect model semantics in [42] and wrt well-founded model semantics in [43]; and correctness wrt Fitting's semantics is studied by Bossi & Etalle in [5]. [36] provides a good survey of program transformation systems. Apart from the correctness, another important property of a transformation system is to preserve the termination behavior of the original program [4,6]. Addressing this, in [6], Bossi and Etalle showed that unfold/fold transformation system preserves the acyclicity, when applied to acyclic programs.

Recent researches in semantics of logic programs, propose new non-monotonic semantics for negation. For example, Dung suggested preferred extension semantics in [12], Kakas & Mancarella introduced stable theory semantics in [25], partial stable models semantics has been forwarded by Sacca & Zaniolo in [39], and in [47] You and Yuan proposed regular model semantics to overcome the shortcomings of stable model semantics. Moreover, works like [e.g. 9,15,35] indicated the links between these semantics and various non-monotonic logics of AI. Therefore, it is important to study the correctness of unfold/fold transformation wrt these new semantics also, and this paper addresses this issue.

One might individually study the correctness of unfold/fold transformation system wrt each of the non-monotonic semantics of negation. But the proof of the correctness will be cumbersome and tedious, as one has to provide proof for every semantics. Instead we wish to exploit the relationship among all these semantics. In this paper, we first observe that to show the correctness of unfold/fold transformation system wrt various semantics of negation, it is enough if the correctness wrt the semantic kernel [16,17] of a normal program is shown. Later, we prove that the unfold/fold framework preserves the semantic kernel of a normal program and obtain previous and new results of the field as corollaries of our main theorem.

Apart from the correctness results for normal programs, we also show that it is easy to apply unfold/fold transformation techniques to extended logic programs that include extended negation [e.g. 1,14,18,21,22,24,29,34]. In fact, we observe that the unfold/fold transformation framework of normal

programs can be applied to extended logic programs, without any modification. Addressing the correctness of the transformation of extended logic programs, we again exploit the relationship among various semantics of extended logic programs, and observe that the framework preserves answer set semantics [21], generalized well-founded semantics [18], and argumentation semantics [14] of extended logic programs.

The rest of the paper is organized as follows: In section 2, we review the basic unfold/fold transformation system proposed by Tamaki and Sato and extended by Seki, Maher, and Gardner & Shepherdson. The concepts of semantic kernel of a normal logic program, with an important theorem showing how the semantic kernel is related to various other semantics, are provided in section 3. The unfold/fold transformation is then shown to preserve the semantic kernel in section 4, where we obtain various correctness results as corollaries. Section 5 is devoted to study how unfold/fold transformation of section 2 can be applied to extended logic programs. Finally, section 6 concludes the paper with some remarks.

Throughout this paper, we assume the reader's familiarity with basic notions of logic programming as provided in [e.g. 30]. We also assume that the reader is familiar with various non-monotonic semantics of logic programs such as preferred extension semantics [12], regular model semantics [47], stable theory semantics [25], and partial stable models [39]. In the sequel, we represent variables by X,Y,Z; atoms by A,B,H; sequences of literals by K, L; clauses by C,D,F; substitutions by θ,γ; and programs by P,Q. All these symbols may be subscripted and/or primed as necessary.

§2. Unfold/fold transformation

The unfold/fold transformation framework described below, was originally proposed by Tamaki and Sato for definite programs and later extended for normal programs by various researchers like Seki, Maher, Gardner & Shepherdson, and others. In fact, the transformation system of [19,31,32,33] slightly differs from that of [41,42,43,44] in the definition of folding, and in this paper we consider the correctness of both the systems. In the sequel, we provide the basic notions of unfold/fold transformation, taken from [19,31,32,33,41,42,43,44], and the interested readers are referred to these references for more information and examples.

Definition 2.1 (Initial Program). An *initial program* P_0 is a normal logic program satisfying the following conditions:

(I1) P_0 is divided into two disjoint sets of clauses, P_{new} and P_{old}. The predicates defined in P_{new} are called *new predicates*, while those defined in P_{old} are called *old predicates*.

(I2) The new predicates appear neither in P_{old} nor in the bodies of the clauses in P_{new}. ▤

Definition 2.2 (Unfolding). Let P_i be a normal program and C a clause in P_i of the form: $H \leftarrow A,L$. Suppose that $C_1,...,C_k$ are all the clauses in P_i such that C_j is of the form: $A_j \leftarrow K_j$ and A_j is unifiable with A, by an mgu θ_j for each j ($1 \le j \le k$). Let C_j' ($1 \le j \le k$) be the result of applying θ_j after replacing A in C with the body of C_j, namely, $C_j'=H\theta_j \leftarrow K_j\theta_j,L\theta_j$. Then $P_{i+1} = (P_i - \{C\}) \cup \{C_1',...,C_k'\}$. C is called the *unfolded clause* and $C_1,...,C_k$ are called the *unfolding clauses*. A is called the *selected atom* (in unfolding). ▤

As mentioned earlier, there are two ways to fold a clause in a program. In the sequel, the folding of [41,42,43,44] is referred to as TSS-folding and that of [19,31,32,33] is referred to as MGS-folding. These two operations differ in where the folding clause is coming from.

Definition 2.3 (TSS-Folding). Let C be a clause in P_i of the form: $A \leftarrow K,L$ and D a clause in P_{new} (not necessarily in P_i) of the form: $B \leftarrow K'$. Suppose that there exists a substitution θ satisfying the following conditions:

(F1) $K'\theta = K$

(F2) Let $X_1,...,X_j,...,X_m$ be internal variables of D, namely, appearing only in the body K' of D but not in B. Then, each $X_j\theta$ is a variable in C s.t. it appears in none of A,L, and $B\theta$. Furthermore, $X_j\theta \neq X_{j'}\theta$ if $j \neq j'$.

(F3) D is the only clause in P_{new} whose head is unifiable with $B\theta$.

(F4) Either the predicate of A is an old predicate, or C is the result of applying unfolding at least once to a clause in P_0.

Then, let C' be the clause: $A \leftarrow B\theta,L$, and let P_{i+1} be $(P_i - \{C\}) \cup \{C'\}$. C is called the *folded* clause and D is called the *folding* clause. ▤

Definition 2.4 (MGS-Folding). Let C be a clause in P_i of the form: $A \leftarrow K,L$ and D a clause in P_i of the form: $B \leftarrow K'$. Suppose that there exists a substitution θ satisfying the following conditions:

(F1) $K'\theta = K$

(F2) Let $X_1,...,X_j,...,X_m$ be internal variables of D, namely, appearing only in the body K' of D but not in B. Then, each $X_j\theta$ is a variable in C s.t. it appears in none of A,L, and $B\theta$. Furthermore, $X_j\theta \neq X_{j'}\theta$ if $j \neq j'$.

(F3) D is the only clause in P_i whose head is unifiable with $B\theta$.

(F4) C is different from D.

Then, let C' be the clause: $A \leftarrow B\theta,L$, and let P_{i+1} be $(P_i - \{C\}) \cup \{C'\}$. C is called the *folded* clause and D is called the *folding* clause. ▤

In this paper, we consider the correctness of both these folding operations and in the sequel, we simply write folding when we do not want to differentiate between these two operations.

Definition 2.5 (Transformation Sequence). Let P_0 be an initial program and P_{i+1} ($i \geq 0$) a program obtained from P_i by applying either unfolding or folding. Then, the sequence of programs $P_0,P_1,...,P_N$ is called a *transformation sequence starting from* P_0. ▤

Various correctness results have been obtained so far, for the unfold/fold transformation system described above. Initially, Tamaki and Sato showed that this transformation is correct for definite programs wrt least Herbrand model semantics [44]. Later Kawamura and Kanamori obtained a stronger result for definite programs in [27], stating that the set of all computed answer substitutions[2] of definite programs are preserved by this transformation. In [31], Maher showed that the transformation system with MGS-folding also preserves the least Herbrand model semantics. For normal programs, correctness of unfold/fold transformation (with MGS-folding) has been shown wrt Clark's completion semantics in [19,33], perfect model semantics in [33,42], stable model semantics in [32,41] and well-founded model semantics in [43]. These previous correctness results are summarized by the following theorem.

Theorem 2.1 (Previous correctness results)

(A) The least Herbrand model of any program P_i in a transformation sequence starting from initial definite program P_o, is identical to that of P_0 [31,44].

(B) The set of all computed answer substitutions of any program P_i in a transformation sequence starting from initial definite program P_0, is identical to that of P_0 [27].

(C) The Clark's completion semantics of any program P_i in a transformation sequence (that does not use TSS-folding and no rule unfolds itself) starting from initial normal program P_0, is identical to that of P_0 [19,33].

(D) The perfect model semantics of any program P_i in a transformation sequence starting from initial stratified program P_0, is identical to that of P_0 [33,42].

(E) The stable model semantics of any program P_i in a transformation sequence starting from initial normal program P_0, is identical to that of P_0 [32,41].

(F) The well-founded model semantics of any normal program P_i in a transformation sequence starting from initial normal program P_0, is identical to that of P_0 [43]. ▤

§3. Semantic Kernel of a normal program

In this section, we review the concepts of semantic kernel and show how it is related to other semantics of normal logic programs. To capture the intended meaning of a normal logic program in

[2][27] defines a computed answer substitution of a goal G wrt program P as a pair (G,θ) s.t. there exists a proof tree of G wrt P with answer substitution θ.

a more natural way, in [16,17] Dung et. al. defined the *semantic kernel*[3] of a normal program. The idea starts with the concept of a quasi-interpretation, which is formally defined below.

Definition 3.1 (Quasi-interpretation). A *quasi-interpretation* I is a set of ground program clauses of the form, $A \leftarrow not\ B_1, \ldots, not\ B_n \quad n \geq 0$, where A, B_i are ground atoms. The set of quasi-interpretation is denoted by QI. It is clear that QI is a complete lattice wrt set inclusion. ▤

Definition 3.2

Let C be a ground clause	$A \leftarrow not\ B_1, \ldots, not\ B_n, A_1, \ldots, A_m$	$n \geq 0, m \geq 0$
and let C_i be ground clauses	$A_i \leftarrow K_i$	$1 \leq i \leq m$
Then $T_C(C_1,\ldots,C_m)$ is the clause	$A \leftarrow not\ B_1, \ldots, not\ B_n, K_1, \ldots, K_m$	

C is said to be the *generating clause* for $T_C(C_1,\ldots,C_m)$. ▤

Definition 3.3 The transformation[4] S_P on quasi-interpretations is defined as follows:

$S_P : QI \rightarrow QI$

$S_P(I) = \{ T_C(C_1,\ldots,C_m) \mid C \in G_P \text{ and } C_i \in I,\ 1 \leq i \leq m \}$ where G_P stands for the set of all ground instantiations of every clause of P. ▤

Lemma 3.1 S_P is ω-continuous. ▤

Definition 3.4 (Semantic Kernel). Let $SK^n(P) = S_p^n(\phi)$. Now the least fixed point of S_p is given by $SK(P) = \bigcup \{SK^n(P) \mid n \geq 1\}$. SK(P) is called as the *semantic kernel* of P. ▤

We now come to the important question of how the semantic kernel of a normal logic program is related to various other semantics proposed in the literature so far. We formally present the relationship by means of the following theorem.

Theorem 3.1 Let P be a normal logic program and SK(P) be its semantic kernel. Then the following results hold:

(A) If P is a definite program, then SK(P) is same as the least Herbrand model of P.
(B) If P is a stratified program, then P and SK(P) have the same perfect model semantics.
(C) P and SK(P) have the same stable model(s).
(D) P and SK(P) have the same well-founded model(s).
(E) P and SK(P) have the same preferred extension semantics.

---

[3] in [16,17], it was simply referred to as least fixpoint denoted as LFP

[4] in [16,17], it was denoted as T_P. We have changed it to S_P here to avoid confusion with the classical T_P operator of logic programming.

(F) P and SK(P) have the same regular model semantics.

(G) P and SK(P) have the same stable theory semantics.

(H) P and SK(P) have the same set of partial stable models.

PROOF See Appendix A. ▤

REMARK 3.1 Results (A) and (B) follow directly from (C) or (D). (C) and (D) have been proved in [17] and [16] respectively. (E), (F), (G), and (H) are new results. ▤

REMARK 3.2 Preferred extension semantics was first introduced in [12]. Regular model semantics was introduced in [47], and partial stable models in [39]. [8,26,48] have extensively studied the relationship among these three semantics and shown that they are equivalent. Stable theory semantics was introduced in [25]. ▤

REMARK 3.3 In [38], Pryzymusinski introduced stationary semantics for disjunctive and normal logic programs. In the case of normal logic programs, it coincides with the well-founded model semantics and hence preserved by unfold/fold transformation. More than that, the stationary expansions of normal programs are also preserved, and this follows from (E) and the results of [8]. ■

§4. Correctness of unfold/fold transformation

In this section we show that the unfold/fold transformation, as described in section 2, preserves the semantic kernel of a normal program. For the sake of clarity, we divide the proof into two parts: the first part shows that the semantic kernel of any program in a transformation sequence is contained in that of the initial program; and the second part shows that the converse is also true. The first part is divided into two cases, one for folding and another for unfolding, while the second part is already implied by a lemma in [43].

Theorem 4.1 (Main Theorem: Unfold/fold transformation preserves the semantic kernel). Let $P_0,...,P_N$ be a transformation sequence. Then the semantic kernel of any program P_j $(0 \le j \le N)$ in a transformation sequence is identical to that of P_0.

PROOF This is proved by induction on j.

Base: j = 0

Obviously, $SK(P_0) = SK(P_0)$

Induction: Assume that the lemma is true for $j \le k$. We have to show that it is true for j = k+1, i.e. $SK(P_0) = SK(P_{k+1})$.

We prove this in two parts: (1) $SK(P_{k+1}) \subseteq SK(P_0)$ and (2) $SK(P_0) \subseteq SK(P_{k+1})$

PART (1): $SK(P_{k+1}) \subseteq SK(P_0)$

Since $SK(P_0) = SK(P_k)$ (from the induction assumption), the required result follows from showing $SK(P_{k+1}) \subseteq SK(P_k)$. There are two cases to consider, reflecting the fact that P_{k+1} may be obtained from P_k by folding or unfolding operation.

CASE A: P_{k+1} is obtained by folding operation

This case is proved by showing the following proposition

$\forall i$: If $C \in SK^i(P_{k+1})$ then $C \in SK(P_k)$

This proposition is proved by induction on i.

Base: i=0

The proposition follows obviously, since $SK^0(P_{k+1}) = \phi$.

Induction: Assuming that the proposition is true for $i \leq I$, we have to show that it holds when $i = I+1$. (where I is a natural number)

$C \in SK^{I+1}(P_{k+1})$. We have to show that $C \in SK(P_k)$.

Let C' be the generating clause of C, and let it be of the form $H \leftarrow B_1,...,B_n,\text{not } B'_1,...,\text{not } B'_m$.

Let C' be a ground instantiation of $C'' \in P_{k+1}$.

Case (i) : $C'' \in P_k$

$C \in SK^{I+1}(P_{k+1})$. Hence, $\forall B_r\ (1 \leq r \leq n)$: $\exists D_r \in SK^I(P_{k+1})$ s.t. $C = T_{C'}(D_1,...,D_n)$. From the inner induction assumption, we have that $\forall r\ (1 \leq r \leq n)$: $D_r \in SK(P_k)$. Since $C'' \in P_k$, it follows that $C \in SK(P_k)$.

Case (ii) : $C'' \notin P_k$

$C \in SK^{I+1}(P_{k+1})$. Hence, $\forall B_r\ (1 \leq r \leq n)$: $\exists D_r \in SK^I(P_{k+1})$ s.t. $C = T_{C'}(D_1,...,D_n)$. From the inner induction assumption, we have that $\forall r\ (1 \leq r \leq n) : D_r \in SK(P_k)$. From the outer induction assumption, it follows that $\forall r\ (1 \leq r \leq n) : D_r \in SK(P_0)$. In this case (CASE A), $C'' \in P_{k+1}$ is the result of folding. Let us consider TSS-folding first.

Let $F \in P_k$ be the folded clause and $D \in P_{new}$ be the folding clause. Further let C'' be of the form, $H \leftarrow B\theta$, K'; D be of the form, $B \leftarrow K$; and F be of the form, $H \leftarrow K\theta$, K'. Let $C' = C''\gamma$, where γ is a ground instantiation. Without any loss of generality, we can assume that $head(D_1) = B\theta\gamma$. From the fact that $D_1 \in SK(P_0)$ and D is the only clause in P_{new} (hence also the only clause in P_0) whose head unifies with $B\theta$, it follows that: $\exists C_{11},...,C_{1m} \in SK(P_0)$ s.t. $D_1 = T_{D\theta\gamma}(C_{11},...,C_{1m})$. From the outer induction assumption, it also follows that $C_{11},...,C_{1m} \in SK(P_k)$. From the conditions of folding, we have that $C = T_{F\gamma}(C_{11},...,C_{1m},D_2,...,D_n)$. Since all these clauses $C_{11},..., C_{1m}, D_2, ...,D_n$ are present in $SK(P_k)$, it follows that $C \in SK(P_k)$.

The proof is very much similar, in case C'' is obtained by MGS-folding. Let $F \in P_k$ be the folded clause and $D \in P_k$ be the folding clause. Further let C'' be of the form, $H \leftarrow B\theta$, K'; D be of the form, $B \leftarrow K$; and F be of the form, $H \leftarrow K\theta$, K'. Let $C' = C''\gamma$, where γ is a ground instantiation. Without any loss of generality, we can assume that $head(D_1) = B\theta\gamma$. From the fact that $D_1 \in SK(P_k)$ and D is the only clause in P_k whose head unifies with $B\theta$, it follows that: $\exists C_{11},...,C_{1m} \in SK(P_k)$ s.t. $D_1 = T_{D\theta\gamma}(C_{11},...,C_{1m})$. From the conditions of folding, we have that $C = T_{F\gamma}(C_{11},...,C_{1m},D_2,...,D_n)$. Since all these clauses $C_{11},..., C_{1m}, D_2, ...,D_n$ are present in $SK(P_k)$, it follows that $C \in SK(P_k)$.

CASE B: P_{k+1} is obtained by an unfolding operation

The case is proved by showing the following proposition:

$\forall i$: If $C \in SK^i(P_{k+1})$ then $C \in SK(P_k)$

The above proposition is proved by induction on i.

Base: i=0

The proposition follows obviously, since $SK^0(P_{k+1}) = \phi$

Induction: Assuming that the proposition is true for every $i \leq I$, we have to show that it holds for i=I+1. (where I is a natural number)

$C \in SK^{I+1}(P_{k+1})$. We have to show that $C \in SK(P_k)$.

Let C' be the generating clause of C, and let it be of the form $H \leftarrow B_1,...,B_n,$not $B'_1,...,$not B'_m. Let C' be a ground instantiation of $C'' \in P_{k+1}$.

Case (i) : $C'' \in P_k$

$C \in SK^{I+1}(P_{k+1})$. Hence $\forall B_r$ $(1 \leq r \leq n)$: $\exists D_r \in SK^I(P_{k+1})$ s.t. $C = T_{C'}(D_1,...,D_n)$. From the inner induction assumption, we have that $\forall r$ $(1 \leq r \leq n)$: $D_r \in SK(P_k)$. Since $C'' \in P_k$, it follows that $C \in SK(P_k)$

Case (ii) : $C'' \notin P_k$

$C \in SK^{I+1}(P_{k+1})$. Hence, $\forall B_r$ $(1 \leq r \leq n)$: $\exists D_r \in SK^I(P_{k+1})$ s.t. $C = T_{C'}(D_1,...,D_n)$. From the inner induction assumption, we have that, $\forall r$ $(1 \leq r \leq n)$: $D_r \in SK(P_k)$. Since $C'' \notin P_k$, it is clear that it is the result of unfolding. From the definition of unfolding, it is clear that there exists a clause C_U in P_k, unfolding which by a C_K in P_k, results in C'' in P_{k+1}. Now, there exists a ground instantiation of C_U of the form $H \leftarrow B_1,..., B_q, B'',$ not $B'_1,...,$not B'_r where B'' is a ground instantiation of the unfolded literal. There also exists a ground instantiation of C_K of the form $B'' \leftarrow B_{q+1},...,B_n,$not $B'_{r+1},...,$ not B'_m. Now, using the fact that $\forall r$ $(1 \leq r \leq n)$: $D_r \in SK(P_k)$, it is not difficult to see that $C \in SK(P_k)$.

PART (2): $SK(P_0) \subseteq SK(P_{k+1})$

This part follows from Lemma 4.2 (Preservation of P_0 - derivation lemma) of [43]. Note that this lemma holds for MGS-folding also (remark 4.2 of [43]). ▤

REMARK 4.1 Though mentioned already, we would like to highlight the fact that the above theorem is valid irrespective of whether TSS-folding or MGS-folding is used. ▤

The following two corollaries follow immediately from the above theorem and the theorem 3.1 of the last section. The first corollary states the results that have been obtained before in the field, while the second one presents new contributions to the field.

Corollary 4.1 (Previous results)

(A) The least Herbrand model of any definite program P_i in a transformation sequence starting from an initial definite program P_0, is identical to that of P_0.
(B) The perfect model semantics of any stratified program P_i in a transformation sequence starting from an initial stratified program P_0, is identical to that of P_0.
(C) The well-founded model semantics of any normal program P_i in a transformation sequence starting from an initial normal program P_0, is identical to that of P_0.
(D) The stable model semantics of any normal program P_i in a transformation sequence starting from an initial normal program P_0, is identical to that of P_0. ▤

Corollary 4.2 (New results)

(A) The preferred extension semantics of any normal program P_i in a transformation sequence starting from an initial normal program P_0, is identical to that of P_0.
(B) The regular model semantics of any normal program P_i in a transformation sequence starting from an initial normal program P_0, is identical to that of P_0.
(C) The stable theory semantics of any normal program P_i in a transformation sequence starting from an initial normal program P_0, is identical to that of P_0.
(D) The partial stable models semantics of any normal program P_i in a transformation sequence starting from an initial normal program P_0, is identical to that of P_0. ▤

§5. Unfold/fold transformation of extended logic programs

Recognizing the lack of expressiveness of traditional logic programming, especially in dealing with incomplete information, Gelfond & Lifschitz and Kowalski & Sadri have proposed to extend logic programming with classical negation [21,29]. Since then, various semantics and applications of extended logic programming have been reported [e.g. 1,14,18,22,24,34]. We do not provide details of

these works here and the interested readers are referred to [1,14,18,21,22,24,29,34] for more information and examples.

In this section, we show that the unfold/fold transformation system of section 2, can be directly applied to the extended logic programs preserving answer set semantics of [21] which is briefly recalled in Appendix B. We also observe that generalized well-founded semantics [18], and argumentation semantics [14] of extended logic programs are also preserved by the unfold/fold transformation. Instead of studying the correctness wrt each individual semantics, we again exploit the relationship among them. To achieve this, we first extend the notion of semantic kernel to extended logic programs and obtain a result which is very similar to that of theorem 3.1. As we show later, the preservation of semantic kernel of an extended logic program by unfold/fold transformation, follows immediately from our main theorem 4.1, and thus the required correctness results. In the sequel, the symbol "not" stands for traditional logic programming negation (i.e. negation as failure), while the classical negation is represented by "¬".

Before considering semantic kernel of an extended logic program, we formally define what we mean by extended logic program and how it can be transformed into a normal logic program. These definitions are basically from [21].

Definition 5.1 (Extended logic program). An *extended program clause* is of the form $L_0 \leftarrow L_1,...,L_m,\text{not } L_{m+1},...,\text{not } L_{m+n}$ where each L_i is an atom or classical negation of an atom. An *extended logic program* is a set of extended program clauses. ▤

Definition 5.2 (Positive normal form). Let P be an extended logic program. For any predicate p occurring in P, let p' be a new predicate of the same arity. The atom p'(...) is referred to as the *positive form* of the negative literal ¬p(...). Every positive literal is, by definition, its own positive form. Let L^+ stand for the positive form of a literal L. Then the *positive normal form of an extended logic program* P, denoted as P^+, is a normal logic program obtained by replacing every literal L occurring in P by its positive form L^+. ▤

Definition 5.3 (Semantic Kernel of an extended logic program). Let P be an extended logic program and P^+ be its positive normal form. Let $SK(P^+)$ be the semantic kernel of the normal program P^+. Let Q be an extended logic program obtained from $SK(P^+)$ by replacing every occurrence of a positive form p'(...) in $SK(P^+)$ by its original form ¬p(...). Then Q is referred to as the *semantic kernel*, denoted as SK(P), of the extended logic program P. ▤

Theorem 5.1 Let P be an extended logic program and SK(P) be its semantic kernel. Then P and SK(P) have the same answer set(s).

PROOF See Appendix B. ▤

The unfold/fold transformation can be easily carried out on an extended logic program. Consider an extended logic program P and its positive normal form P^+. Since P^+ is a normal logic program, the unfold/fold transformation of section 2 and the correctness results of section 4 are valid for P^+. Thus we can apply folding and unfolding operations on P^+, obtaining a new normal program Q^+. From the correctness result of unfold/fold transformation, we know that P^+ and Q^+ have the same semantic kernel. Now Q^+ can be easily transformed into an extended logic program Q by replacing all positive forms of literals by their original forms. It is now clear that P and Q have the same semantic kernel, and the correctness results follow immediately from theorem 5.1. This is formalized in the sequel.

Definition 5.4 : (Initial Extended Program). An *initial extended program* P_0 is an extended logic program satisfying the following conditions:

(I1) P_0 is divided into two disjoint sets of clauses, P_{new} and P_{old}. The predicates defined in P_{new} are called *new predicates*, while those defined in P_{old} are called *old predicates*.

(I2) The new predicates appear neither in P_{old} nor in the bodies of the clauses in P_{new}. ▤

It is easy to see that the positive form of an initial extended program P_0 is an initial normal program satisfying the conditions of definition 2.1. This is formalized by the following lemma.

Lemma 5.1 The positive normal form of an initial extended program, is an initial program. ▤

Definition 5.5 : (Extended transformation sequence). Let P_0 be an initial extended logic program and P^+_0 be its positive normal form. Let $P^+_0, P^+_1, \ldots, P^+_N$ be a transformation sequence starting from the normal program P^+_0. Let P_i $(1 \leq i \leq N)$ be an extended program obtained by replacing every positive form of a literal in P^+_i by its original form. Then $P_0, P_1, \ldots, P_N$ is referred to as an *extended transformation sequence* starting from P_0. ▤

Theorem 5.2 (Unfold/fold transformation preserves the semantic kernel of extended logic programs). The semantic kernel of any extended program P_i in an extended transformation sequence starting from an initial extended program P_0 is identical to that of P_0.

PROOF Follows immediately from the main theorem 4.1 ▤

Corollary 5.1 (Correctness result for unfold/fold transformation of logic programs). The answer set semantics of any extended logic program P_i in an extended transformation sequence starting from an initial extended program P_0, is identical to that of P_0. ▤

REMARK 5.1 Various other semantics for extended logic programs have been proposed, such as generalized well-founded and argumentation semantics. It is easy to show the correctness results for unfold/fold transformation of extended logic programs wrt these semantics also. Let

P be an extended logic program and SK(P) be its semantics kernel. Then the following results also hold:

(A) P and SK(P) have the same generalized well-founded semantics.

(B) P and SK(P) have the same argumentation semantics.

Now, from Theorem 5.1, it follows that the generalized well-founded semantics and argumentation semantics of any extended logic program P_i in an extended transformation sequence starting from an initial extended program P_0, are, respectively, identical to those of P_0.

▤

§6. Concluding Remarks

In this paper, we have addressed the correctness of unfold/fold transformation of normal logic programs. Since the correctness depends on the semantics of normal programs and there are various semantics for normal programs, without any general consensus on which is the best, the trend is to show the correctness of program transformation wrt a particular semantics. In this paper, we have emphasized that a deep understanding of relationship among various semantics of normal programs, should be used to show the correctness of program transformation. In this line, we have demonstrated that to show the correctness of unfold/fold program transformation wrt various popular semantics of normal programs, it is enough to show the correctness wrt semantic kernel. This enabled us to obtain correctness results wrt most of the semantics of normal programs.

There are quite a few other semantics for logic programs based on non-Herbrand models, such as s-Semantics [7], and non-ground stable and well-founded semantics [13,46]. In [3], Bossi and Cocco studied the correctness of program transformation wrt s-Semantics for definite programs, but the correctness is yet to be studied when s-Semantics is extended to normal logic programs. We believe that the methodology used in this paper could be extended (possibly using the notion of non-ground semantic kernel) to study the correctness of program transformation wrt these non-ground semantics also.

We have also shown that unfold/fold transformation can be easily extended to extended logic programming where, apart from negation as failure, classical negation is also allowed. We believe that this is an initial step to apply unfold/fold program transformation techniques to knowledge bases. We are now gaining insight into the links between semantics of normal logic programs and various non-monotonic reasoning frameworks such as Reiter's default logic, autoepistemic logic, Pollock's inductive defeasible logic [e.g. 9,15,35], and we hope that correctness results of this paper will be useful in optimizing knowledge bases.

Authors are thankful to H.N.Phien for his support and encouragement. Constructive and encouraging comments received from Maher, Martens, Sato, Seki, Shepherdson, and the anonymous referees are gratefully acknowledged. The first author would like to thank the Government of Japan for financially supporting his studies at the Asian Institute of Technology, Bangkok, Thailand.

References

1. Alferes,J.J., and L.M. Pereira, On logic program semantics with two kinds of negation, in: K. Apt (Ed.), *Proc. of the Int'l Joint Conf. and Symp. on Logic Programming*, MIT Press, 1992, pp. 574-588.
2. Apt,K.R., Blair,H.A., and Walker,A., Towards a theory of declarative knowledge, in: J. Minker (Ed.), *Foundations of deductive databases and logic programming*, Morgan Kaufmann Publisher Inc., 1988, pp. 89-148.
3. Bossi,A. and Cocco,N., Basic transformation operations which preserve computed answer substitutions of logic programs, *Journal of Logic Programming*, 16:47-87 (1993).
4. Bossi,A. and Cocco,N., Preserving universal termination through unfold/fold, in: G. Levi and M. Rodriguez-Artalejo (Eds.), *Proc. of international conference on Algebraic and Logic Programming*, LNCS 850, Springer-Verlag, 1994, pp. 269-286.
5. Bossi,A. and Etalle,S., More on unfold/fold transformations of normal programs: Preservation of Fitting's semantics, in: *Proc. of the fourth international workshop on Meta Programming in Logic (Meta '94)*, 1994.
6. Bossi,A. and Etalle,S., Transforming Acyclic Programs, to appear in: *ACM Transactions on Programming Languages and Systems*, 1994.
7. Bossi,A., Gabbrielli,M., Levi,G., and Martelli,M., The s-Semantics Approach: Theory and Applications, *Journal of Logic Programming* 19&20:149-197 (1994).
8. Brogi,A., Lamma,E., Mancarella,P., and Mello,P., Normal logic programs as open positive programs, in: K. Apt (Ed.), *Proc. of joint Intl. conf. and symp. on logic programming*, MIT Press, 1992, pp. 783-797.
9. Bondarenko,A., Toni,F., and Kowalski,R.A., An assumption-based framework for non-monotonic reasoning, in: *Proc. of the second Intl. workshop on logic programming and non-monotonic reasoning*, Lisbon, June'93, MIT Press, 1993.
10. Clark,K.L., Negation as Failure, in: H. Gallaire and J. Minker (Eds.), *Logic and Databases*, Plenum press, New York, 1978, pp. 293-322.
11. Clark,K.L. and Sickel,S., Predicate logic: a calculus for deriving programs, in: *Proc. of IJCAI '77*, 1977, pp. 419-420.
12. Dung,P.M., Negation as hypotheses: an abductive foundation for logic programming, in: K. Furukawa (Ed.), *Proceedings of the Eighth International Conference on Logic Programming*, MIT Press, 1991, pp. 3-17.

13. Dung,P.M., On the relations between stable and well-founded semantics of logic programs, *Theoretical Computer Science* 105:7-25 (1992).
14. Dung,P.M., An argumentation semantics for logic programming with explicit negation, in: *Proc. of the ICLP'93*, MIT Press, 1993.
15. Dung,P.M., On the acceptability of arguments and its fundamental role in non-monotonic reasoning and logic programming, in: *Proc. of IJCAI'93*, 1993.
16. Dung,P.M. and Kanchanasut,K., A natural semantics for logic programs with negation, in: C.E. Veni Madhavan (Ed.), *Foundations of software technology and theoretical computer science*, LNCS 405, Springer-Verlag, 1989, pp. 78-88.
17. Dung,P.M. and Kanchanasut,K., A fixpoint approach to declarative semantics of logic programs, in: E.L.Lusk and R.A.Overbeek (Eds.), *Proc. of the North American Conference on Logic Programming*, Vol.1, MIT Press, 1989, pp. 604-625.
18. Dung,P.M. and Ruamviboonsuk,P., Well-founded reasoning with classical negation, in: A.Nerode, W.Marek, and V.S.Subrahmanian (Eds.), *Proc. of the first Intl. Workshop on Logic Programming and Non-monotonic reasoning*, MIT Press, 1991, pp. 120-132.
19. Gardner,P.A. and Shepherdson,J.C., Unfold/Fold Transformations of Logic Programs, in: J.-L. Lassez and G.Plotkin (Eds.), *Computational Logic: Essays in Honor of Alan Robinson*, MIT Press, 1991, pp. 565-583.
20. Gelfond,M. and Lifschitz,V., The stable model semantics for logic programs, in: R.A.Kowalski and K.A.Bowen (Eds.), *Proc. of the 5th International Conf/Symp on Logic Programming*, MIT Press, 1988, pp. 1070-1080.
21. Gelfond,M. and Lifschitz,V., Logic programs with classical negation, in: D.H.D.Warren and P.Szeredi (Eds.), *Proc. of the 7th International Conference on Logic Programming*, MIT Press, 1990, pp. 579-597.
22. Gelfond,M. and Lifschitz,V., Representing actions in extended logic programming, Technical Report, Department of Computer Sciences, University of Texas at Austin, U.S.A., 1992.
23. Hogger,C.J., Derivation of logic programs, *Journal of the ACM* 28:372-392 (1981).
24. Inoue,K., Extended logic programs with default assumptions, in: K.Furukawa (Ed.), *Proc. of the 8th Intl. Conf. on Logic Programming*, MIT Press, 1991, pp. 490-504.
25. Kakas,T. and Mancarella,P., Negation as stable hypotheses, in: A.Nerode, W.Marek, and V.S.Subrahmanian (Eds.), *Proc. of the first Intl. Workshop on Logic Programming and non-monotonic reasoning*, MIT Press, 1991, pp. 275-288.
26. Kakas,T. and Mancarella,P., Preferred extensions are partial stable models, *Journal of Logic Programming* 14:341-348 (1992).
27. Kawamura,T. and Kanamori,T., Preservation of stronger equivalence in Unfold/Fold logic program transformation, *Theoretical Computer Science* 75:139-156 (1990).
28. Komorowski,J., Towards a Programming Methodology Founded on Partial Deduction, in: *Proc. of the ECAI'90*, Stockholm, Sweden, 1990.

29. Kowalski,R.A. and Sadri,F., Logic programs with exceptions, in: D.H.D.Warren and P.Szeredi (Eds.), *Proc. of the 7th Intl. Conf. on Logic Programming*, MIT Press, 1990, pp. 598-613.
30. Lloyd,J.W., *Foundations of Logic Programming*, Second Extended Edition, Springer-Verlag, 1987.
31. Maher,M.J., Correctness of a logic program transformation system, IBM Research Report RC 13496, IBM Thomas J. Watson Research Center, Yorktown Heights, U.S.A., 1987.
32. Maher,M.J., Reasoning about stable models (and other unstable semantics), Technical Report, IBM Thomas J. Watson Research Center, Yorktown Heights, U.S.A., 1990.
33. Maher,M.J., A transformation system for deductive database modules with perfect model semantics, Theoretical Computer Science 110:377-403 (1993).
34. Pereira,L.M. and Alferes,J.J., Well-founded semantics for logic programs with explicit negation, in: B. Neumann (Ed.), *European Conf. on AI*, John Wiley & Sons Ltd., 1992, pp. 102-106.
35. Pereira,L.M., Aparício,J.N., and Alferes,J.J., Non-monotonic reasoning with logic programming, *Journal of Logic Programming (Special Issue on Non-monotonic Reasoning)* 17(2,3,&4):227-263 (1993).
36. Pettorossi,A. and Proietti,M., Transformation of logic programs: Foundations and techniques, *Journal of Logic Programming*, 19&20:261-320 (1994).
37. Przymusinski,T.C., On the declarative semantics of deductive databases and logic programs, in: J.Minker (Ed.), *Foundations of deductive databases and logic programming*, Morgan Kaufmann Publisher Inc, 1988, pp. 193-216.
38. Przymusinski,T.C., Stationary semantics for disjunctive logic programs and deductive databases, *in: Proc. of DOOD'91*, Springer-Verlag, 1991.
39. Sacca,D. and Zaniolo,C., Stable models and non-determinism in logic programs with negation, in: *Proc. of PODS '90*, ACM, 1990, pp. 205-217.
40. Sato,T., Equivalence-preserving first-order unfold/fold transformation systems, *Theoretical Computer Science* 105:57-84 (1992).
41. Seki,H., A comparative study of the well-founded and the stable model semantics: transformation's viewpoint (extended abstract), In: *Proc. of the workshop on Logic Programming and Non-monotonic Logic*, Austin, Texas, November 1990, pp. 115-123.
42. Seki,H., Unfold/fold transformation of stratified programs, *Theoretical Computer Science* 86: 107-139 (1991).
43. Seki,H., Unfold/fold transformation of logic programs for the well-founded semantics, *The Journal of Logic Programming* 16:5-23 (1993).
44. Tamaki,H. and Sato,T., Unfold/fold transformation of logic programs, in: *Proc. of the Second International Conference on Logic Programming*, Uppsala, Sweden, 1984, pp. 127-138.
45. Van Gelder,A., Ross,K.A. and Schlipf,J.S., The well-founded semantics for general logic programs, *Journal of the ACM* 38:620-650 (1991).
46. Van Gelder,A. and Schlipf,J.S., Common-sense axiomatizations for logic programs, *Journal of Logic Programming* 17:161-195 (1993).

47. You,J.-H. and Yuan,L.Y., Three-valued formalization of logic programming: is it needed?, in: *Proc. of the 9th ACM symposium on Principles of Database Systems*, ACM press, 1990, pp. 172-182.
48. You,J.-H. and Yuan,L.Y., On the equivalence of semantics for normal logic programs, to appear in: *Journal of Logic Programming*.

Appendix A: Proof of theorem 3.1

As mentioned earlier in REMARK 3.1, the results (C) and (D) have been proved in [17] and [16] respectively. Results (A) and (B) follow directly from (C) or (D). Hence, we need to prove only the results (E), (F), (G), and (H).

Kakas and Mancarella showed that preferred extension semantics and partial stable model semantics are equivalent in [26]. In [48], You and Yuan studied the relationship among preferred extension, regular model and partial stable model semantics, and showed that they are equivalent. Thus, it is enough to prove one of (E), (F) or (H), and in the sequel we prove (E).

Dung's preferred extension semantics is centered around the concept of admissible hypotheses set, which is reproduced below. The reader is referred to [12,15] for more information on this semantics.

Definition A.1 Let P be a normal program, G_P its ground instantiation and H a set of ground negative literals. Then, we define:

$T_P\uparrow 0(H) = \phi$

$T_P\uparrow i+1(H) = \{A \mid \exists$ a clause $A \leftarrow Bd$ in G_P and $Bd \subseteq H \cup T_P\uparrow i(H)\}$

$T_P\uparrow \omega(H) = \cup_{i\geq 0} T_P\uparrow i(H)$

Abusing the notation, we also write $T_P(H)$ for $T_P\uparrow 1(H)$ when no confusion arises. ▤

Definition A.2 Let P be a normal program and H a set of ground negative literals. $P \cup H$ is said to be *admissible* iff $\forall$ not $p \in H$ the following proposition holds: for every set of ground negative literals E, if $p \in T_P\uparrow \omega(E)$, then $\exists$ not $q \in E$ s.t. $q \in T_P\uparrow \omega(H)$. $P \cup H$ is said to be a *preferred extension* of P iff $P \cup H$ is maximally (wrt set inclusion) admissible. ▤

The result (E), and hence the results (F) and (H), follow from the following lemma.

Lemma A.1 Let P be a normal program and SK(P) its semantic kernel. Let H be a set of ground negative literals. Then, $T_P\uparrow \omega(H) = T_{SK(P)}(H)$.

PROOF

The required result can be proved by proving the following proposition:

$\forall$(integer)i: $T_P\uparrow i(H) = T_{SK^i(P)}(H)$

The proposition is proved by induction on i. When i=0, the proposition follows obviously, as both sides evaluate to empty sets. For the induction phase, we need to prove that the proposition holds for i=I+1, when the proposition holds for all i≤I.

(1) $\forall$ground atom A: $A \in T_P\uparrow I+1(H) \rightarrow A \in T_{SK^{I+1}(P)}(H)$

$A \in T_P\uparrow I+1(H)$ implies that $\exists$ a ground clause $C \in G_P$ of the form $A \leftarrow A_1,...,A_n,\text{not } B_1,...,\text{not } B_m$ where all A_i's and not B_j's are in $H \cup T_P\uparrow I(H)$. From the induction assumption, it is clear that all A_i's are in $T_{SK^I(P)}(H)$. This means that there are clauses C_i's in $SK^I(P)$, whose head is A_i and the body is contained in H. It is now obvious that, $C \in G_P$ and all C_i's $\in SK^I(P)$ lead to a clause D in $SK^{I+1}(P)$, whose head is A and body is contained in H. So, $A \in T_{SK^{I+1}(P)}(H)$.

(2) $\forall$ground atom A: $A \in T_{SK^{I+1}(P)}(H) \rightarrow A \in T_P\uparrow I+1(H)$

$A \in T_{SK^{I+1}(P)}(H)$ implies that $\exists$ a ground clause $D \in SK^{I+1}(P)$, whose head is A and the body is contained in H. Suppose $D \in SK^I(P)$, then the required result follows from the induction assumption. If not, it is clear that $\exists$ clause $C \in G_P$ of the form: $A \leftarrow A_1,...,A_n,\text{not } B_1,...,\text{not } B_m$ s.t. every not B_j is in H, and for every A_i $\exists$ a clause C_i in $SK^I(P)$, whose head is A_i and the body is contained in H. From the induction assumption, it follows that all A_i's are in $T_P\uparrow I(P)$. Hence, every literal in the body of C is in $T_P\uparrow I(H) \cup H$, and so $A \in T_P\uparrow I+1(H)$. ▤

Theorem 3.1

(E) A normal program P and its semantic kernel SK(P) have same preferred extension semantics.

(F) A normal program P and its semantic kernel SK(P) have same regular model semantics.

(H) A normal program P and its semantic kernel SK(P) have same partial stable model semantics.

PROOF From lemma A.1 it is clear that P and SK(P) have same set of admissible hypotheses (ground negative literals) sets. (E) follows from this. (F) follows from (E) and the results of [48]. (H) follows from (E) and the results of [26,48]. ▤

The stable theory semantics of Kakas and Mancarella [25] is based on the concept of weak stability, which is reproduced below. More information of this semantics can be obtained from [25].

Definition A.3 Let P be a normal program and H a set of ground negative literals. H is said to be consistent with P iff there exists no negative literal not A $\in$ H s.t. $A \in T_P\uparrow\omega(H)$.

T(P,H), the program induced by H, is the program obtained from G_P by deleting all negative literals not A, s.t. not A $\in$ H, from the body of every clause of G_P.

H is said to be weakly stable, if for all H' s.t. H' is consistent with T(P,H), $H \cup H'$ is consistent with P. ▤

Theorem 3.1 (G) A normal program P and its semantic kernel SK(P) have same stable theory semantics.

PROOF First of all, observe that SK(T(P,H)) = T(SK(P),H). Now, from lemma A.1, it is clear that P and SK(P) have same set of weakly stable hypotheses sets, and thus same stable theory semantics. ▤

Appendix B: Answer set semantics

Answer set, an extension of stable model semantics, for extended logic programs was introduced by Gelfond & Lifschitz in [21]. In the sequel, we recall the semantics as defined in [21].

Let HB stand for the Herbrand base and let $HB^* = HB \cup \{ \neg A \mid A \in HB \}$.

Definition B.1 Let P be an extended logic program that does not contain any negation as failure literal "not" and G_P its ground instantiation. Then, the *answer set* of P, denoted by $\alpha(P)$, is the smallest subset S of HB^* s.t.

i) for any rule $L_0 \leftarrow L_1,...,L_m$ from G_P, if $\{L_1,...,L_m\} \subseteq S$, then $L_0 \in S$;

ii) if S contains a pair of complementary literals (i.e. for some ground atom A, S contains both A and ¬A), then $S = HB^*$. ▤

Definition B.2 Let P be an extended logic program and G_P its ground instantiation. For any set $S \subset HB^*$, let P^S be the extended program obtained from G_P by deleting

i) each rule that has a literal not L in its body with $L \in S$; and

ii) all literals of the form not L in the bodies of the remaining rules.

Clearly P^S does not contain not, so that its answer set $\alpha(P^S)$ is already defined in B.1. Now, S is called an *answer set* of P iff $S = \alpha(P^S)$. ▤

Proposition B.3 If an answer set of an extended logic program P is inconsistent, then P has exactly one answer set which is HB^*. ▤

Proposition B.4 A consistent set $S \subset HB^*$ is an answer set of P iff S^+ is an answer set (which is also a stable model) of P^+. ▤

Theorem 5.1 An extended logic program P and its semantic kernel SK(P) have the same answer set(s).

PROOF Follows from Proposition B.4 and Theorem 3.1 (C). ▤