ICOT Technical Memorandum: TM-0649

TM-0649

Unfold/Fold Transformation of Stratified Programs

by
H. Seki

December, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Unfold/Fold Transformation of Stratified Programs

(Extended Abstract)*

Hirohisa SEKI

Institute for New Generation Computer Technology

1-4-28, Mita, Minato-ku, Tokyo 108, Japan

Abstract

This paper describes several extensions of Tamaki-Sato's [TS84] unfold/fold transformation of definite programs. We first propose unfold/fold rules preserving also the finite failure set (by SLD-resolution) of a definite program, which the original rules by Tamaki and Sato do not. Then, we show that our unfold/fold rules can be extended to ones for stratified programs, and prove that both the success set and the finite failure set (by SLDNF-resolution) of a stratified program are preserved. Furthermore, preservation of equivalence of the perfect model semantics [Prz] is also discussed.

Key Words: Unfold/Fold Transformation, Equivalence of Programs, SLDNF-resolution, Finite Failure, Stratified Programs, Perfect Model Semantics

1 Introduction

Program transformation provides a powerful methodology for program development, especially for derivation of an efficient program preserving the same meaning (semantics) as that of an original (possibly inefficient) program. Thus, one of the most important properties of program transformation is preservation of equivalence (Maher [Mah86] investigated various formulations of equivalence for logic programs).

Tamaki and Sato proposed an excellent framework for unfold/fold transformation of logic programs [TS84]. Their transformation rules preserve the equivalence of a definite program in the sense of the least Herbrand model. Kawamura and Kanamori [KK88] recently proved that Tamaki-Sato's transformation preserves also the *success set* of a program, that is, a transformed program has the same computed answer substitution as that of the original program for any goal. Thus, the transformation rules by Tamaki and Sato seem to be sufficient, at least as far as positive information inferred from a program is concerned.

In general, however, their transformation does not always preserve the finite failure set (by SLD-resolution) of a definite program. The evaluation of a goal in a transformed program might not be terminating, even if the evaluation of that goal is finitely failed in the original program. Thus, when

*The full version of the paper is to appear in [Sek88a] and [Sek88b] which contain complete proofs.

we are interested in negative information inferred from a program and Clark's Negation as Failure rule [Cla78] is used, their transformation is not sufficient. Furthermore, when we consider an extension of their rules to a general logic program where the body of a clause may contain negative literals, the failure to preserve the finite failure of a program would lead to failure to preserve positive information inferred from the program.

In this paper we propose unfold/fold rules which also preserve the finite failure set of a definite program. Then, we extend them to a stratified program and show that our transformation preserves both the success set and the finite failure set (by SLDNF-resolution) of a given stratified program. Furthermore, preservation of equivalence of transformation in the perfect model semantics [Prz] is also discussed.

The organization of this paper is as follows. after summarizing preliminaries, section 2 gives transformation rules which preserve the finite failure set of a definite program. In section 3, we extend them to stratified programs. In section 4, we discuss transformation rules which preserve the perfect model semantics. Finally, a summary of this work and a discussion of related work are given in section 5.

Throughout this paper, we assume the reader is familiar with the basic concept of logic programming, and the terminology follows in [Llo84]. As notation, variables are denoted by $X, Y, \cdots$, and atoms by $A, B, \cdots$. Multisets of atoms are denoted by $L, K, M, \cdots$. Furthermore, $\theta, \sigma, \cdots$ are used for substitutions.

2 Unfold/fold Transformation

2.1 Preliminaries: Rules of Transformation

This section describes Tamaki-Sato's unfold/fold transformation for definite programs [TS84]. The following descriptions of transformation rules are borrowed mainly from [KK88].

Definition 2.1 Initial (Definite) Program

An initial (definite) program P_0 is a definite program satisfying the following conditions:

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

Example 2.1 Let $P_0 = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$, where

C_1 : $p(0,0)$.

C_2 : $p(X,Y) \leftarrow eq(X,Y), p(X,Y)$.

C_3 : $q(0,0)$.

C_4 : $q(X,Y) \leftarrow plus1(X,Y), q(X,Y)$.

C_5 : $eq(X,X)$.

C_6 : $plus1(X,s(X))$.

C_7 : $r(X,Y) \leftarrow p(X,Y), q(X,Y)$.

and P_{old}= $\{C_1, C_2, C_3, C_4, C_5, C_6\}$, P_{new}= $\{C_7\}$. Thus, 'r' is a new predicate, while the other predicates are old predicates. □

We call an atom A a *new atom* (*old atom*) when the predicate of A is a new predicate (old predicate), respectively.

Definition 2.2 Unfolding

Let P_i be a program, C be a clause in P_i of the form : $H \leftarrow A, L$. Furthermore, let $C_1, \cdots, C_k$ be 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 mgu's, say, θ_j, for each j $(1 \geq j \geq k)$. Let C'_j $(k \geq j \geq 1)$ 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, $P_{i+1} = (P_i - \{C\}) \cup \{C'_1, \cdots, C'_k\}$. C is called the *unfolded clause* and $C_1, \cdots, C_k$ are called the *unfolding clauses*. □

Example 2.2 (Continued from Example 2.1) By unfolding C_7 at atom '$p(X, Y)$' in its body, program P_1=$\{C_1, C_2, C_3, C_4, C_5, C_6, C_8, C_9\}$ is obtained, where

C_8 : $r(0,0) \leftarrow q(0,0)$.

C_9 : $r(X,Y) \leftarrow eq(X,Y), p(X,Y), q(X,Y)$. □

Definition 2.3 Folding

Let C be a clause in P_i of the form: $A \leftarrow K, L$ and D be a clause in P_{new}[1] of the form: $B \leftarrow K'$. Suppose that there exists a substitution θ satisfying the following conditions:

(F1) $K'\theta$=K

(F2) Let $X_1, \cdots, X_j, \cdots, X_m$ be internal variables of D, namely they appear only in the body K' of D (but not in B). Then, each $X_j\theta$ appears neither in A nor L, and furthermore $X_i\theta \neq X_j\theta$ if $i \neq j$.

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

Example 2.3 (Continued from Example 2.2) By folding the body of C_9 by C_7, program P_2=$\{C_1, C_2, C_3, C_4, C_5, C_6, C_8, C_{10}\}$ is obtained, where

C_{10} : $r(X,Y) \leftarrow eq(X,Y), r(X,Y)$. □

2.1.1 Previous Results

Definition 2.4 Transformation Sequence

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

[1] Note that D is not necessarily in P_i.

For the above unfold/fold transformation, Tamaki and Sato proved the following result [TS84].

Theorem 2.1 [Tamaki-Sato 84] The least Herbrand model M_{P_i} of any program P_i in a transformation sequence starting from initial program P_0, is identical to that of P_0. □

Recently, Kawamura and Kanamori [KK88] showed that Tamaki-Sato's transformation preserves also *answer substitutions* for any given top-level goals.

Definition 2.5 Success Set

Let P be a (definite) program. The set of all the atom-substitution pairs (A, σ) such that there exists an successful SLD-derivation for $P \cup \{\leftarrow A\}$ with computed answer σ, is called the *success set* of P, and denoted by $SS(P)$. □

Theorem 2.2 [Kawamura-Kanamori 88] The success set $SS(P_i)$ of of any program P_i in a transformation sequence starting from initial program P_0, is identical to that of P_0. □

Example 2.4 (Continued from Example 2.1, 2.2)

Since $r(0,0) \in M(P_0)$ holds, $r(0,0)$ is also in $M(P_2)$ from Theorem 2.1. More precisely, $(r(X,Y), \sigma = \{X/0, Y/0\})$ is in $SS(P_0)$, thus that pair is also in $SS(P_2)$ from Theorem 2.2. □

2.2 Modified Folding Rule and Preservation of FF

2.2.1 Modified Folding Rule

In this paper, the finite failure (FF) set of a program is also considered.

Definition 2.6 Finite Failure (FF) Set

Let P be a (definite) program. The set of all atoms A such that there exists a finitely failed SLD-tree for $P \cup \{\leftarrow A\}$, is called the *(SLD) finite failure set* of P, and denoted by $FF(P)$. □

The *partial correctness* of the transformation *wrt FF* is easily shown.

Proposition 2.1 (Partial Correctness wrt FF) Let $P_0, \cdots, P_N$ be a transformation sequence. Then, $FF(P_N) \subseteq FF(P_0)$ for all $N \geq 0$.

Proof: Let G be a definite goal, and suppose that $P_N \cup G$ has a finitely failed SLD-tree. From the soundness of SLD-resolution [Cla78], $comp(P_N) \vdash G$. It is easily to see that $comp(P_0) \vdash comp(P_N)$ holds[2]. Thus, G is also a logical consequence of $comp(P_0)$. Then, from the completeness of SLD-resolution [JLL83], $P_0 \cup G$ has a finitely failed SLD-tree. □

Tamaki-Sato's unfold/fold transformation, however, does *not* preserve the *total correctness wrt FF*. That is, $FF(P_0) \subseteq FF(P_i)$ for all i $(N \geq i > 0)$ does not hold in general.

[2] Note that this converse does not hold in general, that is, $comp(P_N) \not\vdash comp(P_0)$.

Example 2.5 (Continued from Example 2.1, 2.2)

The failure set of the original program P_0 is *not* preserved. For example, $r(s(0), s(0)) \in FF(P_0)$, while $r(s(0), s(0))$ is not contained in $FF(P_2)$. In fact, any SLD-derivation for $P_2 \cup \{\leftarrow r(s(0), s(0))\}$ is *infinite*. Thus, $FF(P_0) \not\subseteq FF(P_2)$. □

We now give a modified transformation rule which preserves also the total correctness wrt FF. In order to specify such a rule, we need several definitions.

Definition 2.7 Inherited Atom

Let $P_0, \cdots, P_N$ be a transformation sequence starting from P_0, and C be a clause in P_i $(N \geq i \geq 0)$ whose head is a new atom. Then, an atom in the body of C is called an atom *inherited from* P_0 if one of the following conditions is satisfied:

(i) C is a clause in P_{new}. Then, each atom in the body of C is *inherited from* P_0.

(ii) Let C be the result of unfolding in P_i. Suppose that C_+ in P_{i-1} is the unfolded clause of the form: $A \leftarrow B, B_1, \cdots, B_n$, and C_- in P_{i-1} is one of the unfolding clauses of the form: $B' \leftarrow K$. Thus, C is of the form: $A\theta \leftarrow K\theta, B_1\theta, \cdots, B_n\theta$, where θ is an mgu of B and B'. Then, each atom $B_j\theta$ $(1 \leq j \leq n)$ in C is *inherited from* P_0, if B_j in C_+ is inherited from P_0.

(iii) Let C be the result of folding in P_i. Suppose that C_+ in P_{i-1} is the folded clause of the form: $A \leftarrow K, B_1, \cdots, B_n$, and D in P_{new} is the folding clause of the form: $B \leftarrow K'$. Thus, C is of the form: $A \leftarrow B\theta, B_1, \cdots, B_n$, where θ is an mgu such that $K'\theta = K$. Then, each atom B_j $(1 \leq j \leq n)$ in C is *inherited from* P_0, if B_j in C_+ is inherited from P_0. □

Intuitively, an inherited atom is (a possibly instantiated version of) an atom such that it was in the body of some clause in P_{new} and no unfolding has been applied to it.

Example 2.6 In Example 2.1, both '$p(X,Y)$' and '$q(X,Y)$' in the body of C_7 are inherited atoms. In the body of clause C_9 (Example 2.2), atom '$q(X,Y)$' is inherited from P_0, while neither '$eq(X,Y)$' nor '$p(X,Y)$' is inherited from P_0. □

Now, we can define a *modified* folding rule.

Definition 2.8 Folding (modified)

Let C and D be defined similarly in **Definition 2.3**, namely, C is a clause in P_i of the form: $A \leftarrow K, L$ and D be a clause in P_{new} of the form: $B \leftarrow K'$. Suppose that there exists a substitution θ satisfying the following conditions:

- (F1), (F2) and (F3) are the same as those defined in **Definition 2.3**.
- **(F4')** Either the predicate of A is an old predicate, or there is no atom in K which is inherited from P_0. □

Example 2.7 (Continued from Example 2.2)

Consider clause C_9 in Example 2.2. As is noted in Example 2.6, atom '$q(X,Y)$' in its body is inherited from P_0, thus the modified folding does not allow to fold it by C_7.

Instead, by unfolding C_9 at atom '$q(X,Y)$' in its body, program P_2^m=$\{C_1, C_2, C_3, C_4, C_5, C_6, C_8, C_{11}, C_{12}\}$ is obtained, where

C_{11} : $r(0,0) \leftarrow eq(0,0), p(0,0)$.

C_{12} : $r(X,Y) \leftarrow eq(X,Y), p(X,Y), plus1(X,Y), q(X,Y)$.

Now, atom '$q(X,Y)$' in the body of C_{12} is not inherited from P_0, so that the modified folding is now applicable to C_{12}. That is, by folding the body of C_{12} by C_7, program P_3^m=$\{C_1, C_2, C_3, C_4, C_5, C_6, C_8, C_{11}, C_{13}\}$ is obtained, where

C_{13} : $r(X,Y) \leftarrow eq(X,Y), plus1(X,Y), r(X,Y)$. □

Hereafter, except in section 4, by *folding* we mean the *modified* folding defined in Definition 2.8, and by a *transformation sequence*, we mean the one obtained by applying either unfolding or the modified folding.

2.2.2 Preservation of FF for Definite Clauses

In this subsection, we show that the unfold/fold transformation (using the modified folding) guarantees the total correctness wrt FF for definite programs. We need one more definition and a lemma.

Definition 2.9 P_{new}-expansion

Let A be an atom and L be a sequence of atoms. L is called a *P_{new}-expansion of A*, denoted by $\overline{A}$, if the following conditions are satisfied:

- When A is an old atom, L is A itself.
- When A is a new atom, L is either A, or a sequence of atoms '$B_1\theta, \cdots, B_n\theta$' such that there exists a clause in P_{new} of the form: $A_0 \leftarrow B_1, \cdots, B_n$ and θ is an mgu of A and A_0.

Similarly, let M be a sequence of atoms of the form: $G_1, \cdots, G_k$. Then, L is called a *P_{new}-expansion of M*, denoted by $\overline{M}$, if L=$\overline{G_1}, \cdots, \overline{G_k}$.

Example 2.8 (Continued from Example 2.1) Since $p(X,Y)$ is an old atom, a P_{new}-expansion of $p(X,Y)$ is itself. On the other hand, a P_{new}-expansion of $r(0,Y)$ is either itself, or a sequence of atoms '$p(0,Y), q(0,Y)$'. □

Lemma 2.1 (P_0-simulation of SLD-derivation in P_N)

Let $P_0, \cdots, P_N$ be a transformation sequence. Let G be a goal, and suppose that there exists an SLD-derivation Dr for $P_N \cup \{G\}$, $G_0 = G, \cdots, G_k, \cdots$ using input clauses in P_N and substitutions $\theta_1, \cdots, \theta_k, \cdots$. Then, there exists an SLD-derivation Dr_0 for $P_0 \cup \{G\}$, $F_0 = G, \cdots, F_l, \cdots$ using input clauses in P_0 and substitutions $\sigma_1, \cdots, \sigma_l, \cdots$, satisfying the following conditions:

F_0 : $\leftarrow$ $r(s(0), s(0))$

| C_7

F_1 : $\leftarrow$ $\underline{p(s(0), s(0))}, q(s(0), s(0))$

| C_2

F_2 : $\leftarrow$ $eq(s(0), s(0)), p(s(0), s(0)), \underline{q(s(0), s(0))}$

| C_4

F_3 : $\leftarrow$ $eq(s(0), s(0)), p(s(0), s(0)),$
$\underline{plus1(s(0), s(0))}, q(s(0), s(0))$

|

fail

G_0 : $\leftarrow$ $r(s(0), s(0))$

| C_{13}

G_1 : $\leftarrow$ $eq(s(0), s(0)), \underline{plus1(s(0), s(0))}, r(s(0), s(0))$

|

fail

Figure 1: P_0-simulation (left) of an SLD-derivation for $P_3^m \cup \{\leftarrow r(s(0), s(0))\}$ (right)

(i) For each k ($k \geq 0$), there exists some l (≥ 0) such that $F_l\sigma_1 \cdots \sigma_l$ is an P_{new}-expansion of $G_k\theta_1 \cdots \theta_k$, and

(ii) the restriction of $\sigma_1 \cdots \sigma_l$ to the variables in G is the same as that of $\theta_1 \cdots \theta_k$.

(iii) *(fairness)* Furthermore, if the SLD-derivation $G_0 = G, \cdots, G_k, \cdots$ is fair, then so is the SLD-derivation for $F_0 = G, \cdots, F_l, \cdots$.

Dr_0 is called a *P_0-simulation of Dr*.

Notes on the proof: The proof is done by induction on both the length of a transformation sequence N and a length of SLD-derivation k. In order to show the fairness in (iii), the folding condition ($F4'$) is essential. For the lack of space, we omit the proof (see [Sek88a]). □

Example 2.9 Consider an SLD-derivation Dr_3 for $P_3^m \cup \{G_0 = \leftarrow r(s(0), s(0))\}$, where P_3^m was given in Example 2.7. See the right-hand side in Figure 1. Dr_3 has a P_0 -simulation $F_0 = G_0, \cdots, F_1, \cdots, F_3$, which is shown in the left-hand side in the figure (underlined atoms mean selected atoms). Note that F_3 is a P_{new}-expansion of G_1. □

Proposition 2.2 (Total Correctness wrt FF) Let $P_0, \cdots, P_N$ be a transformation sequence. Then, $FF(P_0) \subseteq FF(P_N)$ for all $N \geq 0$.

Proof:

For the simplicity of explanation, we assume here that G is a ground atom (a more general case is shown in **Proposition 3.3**). Suppose that an SLD-tree of $P_0 \cup \{\leftarrow G\}$ is finitely failed. Suppose further that $P_N \cup \{\leftarrow G\}$ has a *fair* SLD-tree which is not finitely failed. Obviously, any SLD-derivation $P_N \cup \{\leftarrow G\}$ never succeeds ; otherwise, a P_0-simulation of such derivation would also succeed, which is a contradiction. Let BR be any non-failed infinite branch in the fair SLD-tree for $P_N \cup \{\leftarrow G\}$. From

Lemma 2.1, there exists a fair SLD-derivation Dr_0 for $P_0 \cup \{\leftarrow G\}$ which is a P_0-simulation of BR. Thus, Dr_0 is a non-failed fair infinite derivation. From the result by Lassez and Maher [LM84], G is in the SLD finite failure set of P_0 iff every fair SLD-tree for $P_0 \cup \{\leftarrow G\}$ is finitely failed. Thus, Dr_0 should be finitely failed, which is a contradiction. □

3 Unfold/Fold Transformation for Stratified Programs

3.1 Preliminaries

We now consider an extension of the unfold/fold transformation from definite programs to stratified programs.

Definition 3.1 Stratified Program [ABW88]

A general logic program, P, is *stratified* if its predicates can be partitioned into levels so that, in every program clause, $p \leftarrow L_1, ..., L_n$, the level of every predicate in a positive literal is less than or equal to the level of p and the level of every predicate in a negative literal is less than the level of p. □

Throughout this paper, we assume that the levels of a stratified program are 1, ..., r for some integer r, where r is the *minimum* number satisfying the above definition. In this case, P is said to have the maximum level r and is denoted $P = \mathcal{P}^1 + ... + \mathcal{P}^r$, where $\mathcal{P}^i$ is a set of clauses whose head predicates have level i. Note that $\mathcal{P}^1$ is a set of definite clauses. When L is a literal whose predicate has level i, we denote it $level(L) = i$. Furthermore, the *stratum* [Prz] of a goal is defined as follows. For any positive atom A, let $stratum(A) = level(A)$ and $stratum(\neg A) = stratum(A) + 1$. Suppose that G is a goal of the form: $\leftarrow L_1, \cdots, L_n$, where $n \geq 0$ and L_i's are literals. Then, $stratum(G)$ is 0 if G is empty, and $max\{stratum(L_i) \ : \ 1 \leq i \leq n\}$, otherwise.

As we did in the previous section, we have to define an initial program, unfolding/folding and a transformation sequence for stratified programs. Although they are almost the same as the previous ones, we impose further restrictions on an initial stratified program.

Definition 3.2 Initial (Stratified) Program

An initial (stratified) program P_0 is a *stratified* program satisfying the following conditions:

- (I1) and (I2) are the same as ones defined in **Definition 2.1**, and
- **(I3)** For each new predicate, its definition consists of exactly one clause.
- **(I4)** Furthermore, the body of each clause in P_{new} contains no negative literal. □

The above condition (I3) guarantees that a stratified program is also stratified after the unfold/fold transformation as shown in the below (Proposition 3.1). On the other hand, the condition (I4) is due to the fact that we do not employ such "unfolding" as it is applicable to a *negative* literal in the body of a clause. Thus, if a clause C in P_{new} contained a negative atom ' $\sim A$' in its body, then, after applying unfolding (possibly several times), C would be unfolded into a clause, say, C', where (possibly

instantiated version of) $\sim A$ in the body of C' would remain as an inherited atom from P_0. Thus, it would prevent us from applying the folding rule to C'. On the other hand, it is needless to say that the body of a clause in P_{old} can contain negative literals.

The unfolding, the (modified) folding and a transformation sequence are the same as those defined in Definition 2.2, Definition 2.8 and Definition 2.4, respectively.

At first, we have to confirm that our unfold/fold transformation preserves a stratification of an initial program.

Proposition 3.1 (Preservation of Stratification) Let $P_0, \cdots, P_N$ be a transformation sequence. Then, if P_0 is a stratified program, then so is P_i $(N \geq i \geq 0)$.

Proof: Let p be a new predicate, and let $C \in P_{new}$ be its definition of the form: $p \leftarrow L$. Then, we define the level of p by $level(p)=max\{level(B_j) \mid B_j \in L\}$. Then, the proposition is obvious from the definitions of unfolding and folding. □

3.2 Partial Correctness of Transformation

The success set (SS) and the finite failure (FF) set of a stratified program are defined similarly to those of a definite program. That is, SS (FF) of a stratified program is defined by replacing "SLD-derivation (SLD-tree)" in Definition 2.5 (Definition 2.6) with "SLDNF-derivation (SLDNF-tree)", respectively.

In this subsection, we show the partial correctness of our transformation wrt both SS and FF.

Proposition 3.2 (Partial Correctness wrt SS and FF)

Let $P_0, \cdots, P_N$ be a transformation sequence. Then,

(SS) : If $SS(P_i)=SS(P_0)$, then $SS(P_{i+1}) \subseteq SS(P_i)$ for $i=0, \cdots, N-1$.

(FF) : If $FF(P_i)=FF(P_0)$, then $FF(P_{i+1}) \subseteq FF(P_i)$ for $i=0, \cdots, N-1$.

Notes on the proof: We first note that we used the completeness of SLD-resolution for the proof wrt FF in Proposition 2.1. In this case, however, we cannot resort to the completeness of SLDNF-resolution, since we do not assume such conditions as allowedness and strictness ([ABW88]) which guarantees its completeness ([CL87]). Thus, we show the above two properties (SS) and (FF) by a more direct proof, based on mutual induction on the stratum of a goal (see [Sek88a] for the complete proof). □

3.3 Total Correctness of Transformation

3.3.1 Total Correctness wrt FF

We now show the total correctness of our unfold/fold transformation. We prove the total correctness wrt FF first. Due to the partial correctness wrt FF, it is easy to show that **Lemma 2.1** replacing "SLD-derivation" in it with "SLDNF-derivation" also holds for stratified programs. That is,

Lemma 3.1 (P_0**-simulation of SLDNF-derivation in** P_N)

Let $P_0, \cdots, P_N$ be a transformation sequence. Let G be a goal, and suppose that there exists an SLDNF-derivation Dr for $P_N \cup \{G\}$, $G_0 = G, \cdots, G_k, \cdots$ using input clauses in P_N and substitutions $\theta_1, \cdots, \theta_k, \cdots$. Then, there exists an SLDNF-derivation Dr_0 for $P_0 \cup \{G\}$, $F_0 = G, \cdots, F_l, \cdots$ using input clauses in P_0 and substitutions $\sigma_1, \cdots, \sigma_l, \cdots$, satisfying the following conditions:

(i) For each k ($k \geq 0$), there exists some l (≥ 0) such that $F_l\sigma_1 \cdots \sigma_l$ is an P_{new}-expansion of $G_k\theta_1 \cdots \theta_k$, and

(ii) the restriction of $\sigma_1 \cdots \sigma_l$ to the variables in G is the same as that of $\theta_1 \cdots \theta_k$.

(iii) *(fairness)* Furthermore, if the SLDNF-derivation $G_0 = G, \cdots, G_k, \cdots$ is fair, then so is the SLDNF-derivation for $F_0 = G, \cdots, F_l, \cdots$.

Dr_0 is called a *P_0-simulation of Dr*. □

Now we can show the total correctness wrt FF.

Proposition 3.3 (Total Correctness wrt FF)

Let $P_0, \cdots, P_N$ be a transformation sequence, where P_0 is an initial stratified program. Then, $FF(P_0) \subseteq FF(P_N)$ for all $N \geq 0$.

Outline of the proof: Suppose that an SLDNF-tree of $P_0 \cup \{\leftarrow A\}$ is finitely failed. Obviously, any SLDNF-derivation $P_0 \cup \{\leftarrow A\}$ never succeeds. Furthermore, it does not flounder, from the proposition shown by Shepherdson [She84] which says that, if a query Q flounders under a computation rule, then it cannot fail under *any* computation rule.

Suppose that $P_N \cup \{\leftarrow A\}$ has a *fair* SLDNF-tree which is *not* finitely failed. Let BR_N be any non-failed branch in that fair SLDNF-tree for $P_N \cup \{\leftarrow A\}$.

From **Lemma 3.1**, there exists a *fair* SLDNF-derivation BR_0 for $P_0 \cup \{\leftarrow A\}$ which is a P_0-simulation of BR_N. BR_0 neither succeeds nor flounders as is noted in the above. Thus, BR_0 is a non-failed fair infinite derivation. Then, we can show that $comp(P_0) \cup \{\exists A\}$ has a model, using similar methods in the proofs of completeness of Negation as Failure rule by [Llo84], [CL87], which is a contradiction. □

3.3.2 Total Correctness wrt SS

Finally, we state the total correctness wrt SS.

Proposition 3.4 (Total Correctness wrt SS)

Let $P_0, \cdots, P_N$ be a transformation sequence, where P_0 is an initial stratified program. Then, $SS(P_0) \subseteq SS(P_N)$ for all $N \geq 0$.

Notes on the proof: The proof can be done along almost the same line as ones given by [TS84], [Tam87] or [KK88], except the handling of negative literals. However, it follows immediately from the total correctness wrt FF in **Proposition 3.3** (the complete proof is found in [Sek88a]).

4 On Preservation of Perfect Model Semantics

The semantics we have considered is somewhat operational, in that the success set and the finite failure set of a stratified program are given by specific procedures such as SLD(NF)-resolution. In this section, we consider more declarative semantics, that is, the standard (minimal Herbrand) model M_P by Apt, Blair and Walker [ABW88] and Van Gelder [VG86], or more generally, the *perfect model semantics* for stratified programs introduced by Przymusinski [Prz].

It seems to be a more direct extension from Tamaki-Sato's original unfold/fold rules to consider transformation rules preserving the equivalence of M_P or the perfect model semantics, since their framework preserves the least Herbrand model for a definite program. Recall that, Tamaki-Sato's unfold/fold transformation does not preserve the finite failure set. However, from the point of the perfect model semantics, it makes no problems, since a goal : "$\leftarrow G$" which has neither a successful SLD-derivation nor a finite failed SLD-tree is simply considered to be false. For the lack of space, we assume the familiarity with the perfect model semantics (see [Prz]), and we state only results (for further detail, see [Sek88b]).

Definition 4.1 Initial Program

An initial program P_0 is a *stratified* program satisfying the following conditions:

- (I1), (I2) and (I3) are the same as ones defined in Definition 3.2. □

Thus, condition (I4) in Definition 3.2 is unnecessary.

The unfolding rule and the folding rule are the same as those defined in Definition 2.2 and Definition 2.3, respectively. Note that we do not have to consider the modified folding rule. A transformation sequence is also defined similarly to Definition 2.4.

Then, we have the following proposition.

Theorem 4.1 (Preservation of Perfect Model Semantics)

The perfect model semantics of any program P_i in a transformation sequence starting from initial program P_0, is identical to that of P_0. □

Notes on the proof: First, we fix a pre-interpretation (e.g., [Llo84]) J of a program P_0. Note that every perfect model M is *supported* (see [Prz]), that is, for every J-ground atom A in M there exists a J-ground instance of a clause from a program such that its head is equal to A, all positive premises belong to M and none of the negative premises belongs to M. From this property, we can consider a ground "proof tree" for any $A \in M$. Then, the proposition follows from the similar discussions in [TS84] and [Tam87], where correctness proofs are based on the manipulation of ground finite proof trees. □

5 Conclusion

There are several studies on equivalence-preserving transformation for logic programs. Tamaki and Sato's result [TS84] and its elaboration by Kawamura and Kanamori [KK88] are already described in section

2.1.1. Maher extensively studied various formulation of equivalence for definite programs [Mah86]. In that paper, he considered a transformation system similar to that of Tamaki and Sato, and stated that his unfold/fold rules preserve logical equivalence of completions, while, as is stated in section 2.2.1, those of Tamaki-Sato do not preserve it. Kanamori and Horiuchi [KH87] proposed a framework for transformation and synthesis based on generalized unfold/fold rules. Their system was shown to preserve the minimum Herbrand model semantics, but the finite failure set is not preserved in general.

Compared with previous work, the contributions of this paper will be summarized as follows :

1) The modified folding rule for a definite program has been proposed.

   The unfolding rule together with the modified folding rule has been shown to preserve the finite failure set (by SLD-resolution) of a program as well as the success set. This guarantees a safer use of Tamaki-Sato's transformation when negation as failure rule is used.

2) The unfold/fold rules for stratified programs have been proposed.

   The modified folding rule has made it possible to extend the applicability of unfold/fold transformation rules to a stratified program, so that they preserve both the success set and the finite failure set of a stratified program by SLDNF-resolution.

3) Preservation of equivalence of the perfect model semantics has been discussed.

   We have shown that unfold/fold rules by Tamaki and Sato can be extended to rules for a stratified program and it preserve the equivalence of the perfect model semantics.

Acknowledgement

This work is based on the result by Tamaki and Sato, and the successor work by Kawamura and Kanamori. We would like to express deep gratitude to them for their stimulative work. The idea of the modified folding arose from discussions with Kazunori Ueda and Tadashi Kanamori.

References

[ABW88] K.R. Apt, H. Blair, and A. Walker. Towards A Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148, Morgan Kaufmann, 1988. Los Altos, CA.

[CL87] L. Cavedon and J. W. Lloyd. *A Completeness Theorem For SLDNF-Resolution*. Technical Report CS-87-06, Computer Science Department. University Walk, Bristol, 1987.

[Cla78] K.L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Database*, pages 293–322, Plenum Press, 1978.

[JLL83] J. Jaffar, J.-L. Lassez, and J. W. Lloyd. Completeness of the Negation as Failure Rule. In *IJCAI-83*, pages 500–506, Karlsruhe, 1983.

[KH87] T. Kanamori and K. Horiuchi. Construction of Logic Programs Based on Generalized Unfold/Fold Rules. In *Proceedings of the Fourth International Conference on Logic Programming*, pages 744–768, Melbourne, 1987.

[KK88] T. Kawamura and T. Kanamori. *Preservation of Stronger Equivalence in Unfold/Fold Logic Program Transformation.* ICOT Technical Report, ICOT, 1988. also to appear in FGCS'88.

[Llo84] J.W. Lloyd. *Foundations of Logic Programming.* Springer, 1984.

[LM84] J.-L. Lassez and M.J. Maher. Closures and Fairness in the Semantics of Programming Logic. *Theoretical Computer Science*, 29:167–184, 1984.

[Mah86] M.J. Maher. Equivalences of Logic Programs. In *Proceedings of the Third International Conference on Logic Programming*, pages 410–424, London, 1986. also in Foundations of Deductive Databases and Logic Programming, (edited by Minker, J.), pp. 627-658, Morgan Kaufmann, 1988.

[Prz] T.C. Przymusinski. On the Declarative and Procedural Semantics of Logic Programs. submitted for publication. Its extended abstract appears in 5th International Conference Symposium on Logic Programming, Seattle, 1988.

[Sek88a] H. Seki. *Unfold/Fold Transformation of Stratied Programs.* ICOT Technical Report, ICOT, 1988. in preparation.

[Sek88b] H. Seki. *Unfold/Fold Transformation of Stratied Programs in the Perfect Model Semantics.* ICOT Technical Report, ICOT, 1988. in preparation.

[She84] J.C. Shepherdson. Negation as Failure: A Comparison of Clark's Completed Data Base and Reiter's Closed World Assumption. *J. Logic Programming*, 1:51–79, 1984.

[Tam87] H. Tamaki. *Program Transformation in Logic Programming*, pages 39–62. Kyoritsu Pub. Co., 1987. in Japanese.

[TS84] H. Tamaki and T. Sato. Unfold/Fold Transformation of Logic Programs. In *Proceedings of the Second International Logic Programming Conference*, pages 127–138, Uppsala, 1984.

[VG86] A. Van Gelder. Negation as Failure Using Tight Derivations for General Logic Programs. In *Proc. 1986 Symposium on Logic Programming*, pages 127–138, IEEE Computer Society, 1986.