TM-0966

A Comparative Study of the Well-founded and the Stable Model Semantics: Transformation's Viewpoint

by

H. Seki (Mitsubishi)

December, 1990

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Comparative Study of the Well-founded and the Stable Model Semantics: Transformation's Viewpoint (Extended Abstract)

Hirohisa SEKI

E-mail: seki@sys.crl.melco.co.jp

Central Research Lab., Mitsubishi Electric Corp.

8-1-1, Tsukaguchi-Honmachi, Amagasaki, Hyogo, JAPAN 661

1 Introduction

We give a comparative study of two major semantics for general logic programs, i.e., the well-founded semantics [VGRS88] and the (two-valued) stable model semantics [GL88], from the viewpoint of program transformation.

Program transformation and partial evaluation have been considered to be a useful methodology for program development and their usefulness has been shown in various applications (e.g., [BD77], [Fut71] and [Ers77]). It will be therefore a useful and interesting question to examine whether or not each of the two semantics is amenable to the program transformation developed so far. We will consider specifically unfold/fold transformation, with an attention paid to its preservation of equivalence. Tamaki and Sato proposed an elegant 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. Recently, Seki [Sek89] gave an extension of the unfold/fold transformation rules to stratified programs [ABW87], where not only the success set and the finite failure set (by SLDNF-resolution) of a given stratified program but also the perfect model semantics [Prz86] of the program is shown to be preserved.

In this paper, we first specify the rules for unfold/fold transformation of *general* logic programs. We then introduce a *reduction rule* which can be considered to be an instance of partial evaluation. The main result of the paper is that the well-founded semantics is preserved for both the unfold/fold rules and the reduction rule, whereas the stable model semantics is also preserved for the unfold/fold rules but it is *not* necessarily so for the reduction rule. This implies that the stable model semantics is not so "stable" from the viewpoint of program transformation and that it requires a more careful treatment for its preservation than the well-founded semantics.

2 Unfold/fold Transformation

2.1 Preliminaries: Rules of Transformation

This section describes a framework of unfold/fold transformation of *general* logic programs, which is defined along the same lines as those in [TS84], except that programs to which transformation is applied are now general logic programs. In the following, variables are denoted by $X, Y, \cdots$, and literals by $A, B, \cdots$. Multisets of atoms are denoted by $L, K, M, \cdots$, and $\theta, \sigma, \cdots$ are used for substitutions.

Definition 2.1 Initial Program

An *initial* program P_0 is a *general* 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}. □

New predicates are considered to be those introduced by "Definition Rule" in the literature [BD77]. They are supposed to be given at the beginning of transformation in our framework. We call an atom, A, a ***new atom*** (an ***old atom***) when the predicate of A is a new predicate (an old predicate), respectively.

Definition 2.2 Unfolding

Let P_i be a program and C a clause in P_i of the form: $H \leftarrow A, L$. Suppose that $C_1, \cdots, C_k$ 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, say, θ_j, for each j $(1 \leq j \leq k)$.

Let C'_j $(1 \leq j \leq k)$ 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, $P_{i+1} = (P_i - \{C\}) \cup \{C'_1, \cdots, C'_k\}$. C is called the *unfolded clause* and $C_1, \cdots, C_k$ are called the *unfolding clauses*. □

Definition 2.3 Folding

Let C be a clause in P_i of the form: $A \leftarrow K, L$ and D a clause in $P_{new}$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, appearing only in the body K' of D but not in B. Then, each $X_j\theta$ is a variable in C such that it appears in none of A, L and $B\theta$. 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 v 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. □

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

3 Preservation of the Well-Founded Semantics

We now show that the unfold/fold transformation preserves the well-founded semantics for general logic programs. For the lack of space, we assume that readers are familiar with the definitions and basic terminologies wrt the well-founded semantics, which are found in [VGRS88],[VG89] and [Prz89]. We denote the well-founded semantics of a general logic program P by $WFS(P)$.

Proposition 3.1 (Preservation of the Well-Founded Semantics) [Sek90]

The well-founded semantics $WFS(P_i)$ of any program P_i ($i \geq 0$) in a transformation sequence starting from initial program P_0, is identical to $WFS(P_0)$. □

Moreover, it is shown that the *dynamic stratification* (see [Prz89]) of each atom is also preserved [Sek90]. Note that the above proposition has covered the previous result by Tamaki and Sato [TS84] for definite programs and the one by Seki [Sek89] for stratified programs.

Now, we introduce another transformation rule called a *reduction rule*. When no negative premise appears in the body of each clause, the reduction rules are considered to be special cases of the *goal replacement rule* studied in [TS84]. Although the rule seems to be quite simple, it is useful for examining the well-founded semantics of a given program.

Definition 3.1 Reduction Rule

Let P_i be a program and C a clause in P_i of the form: $H \leftarrow A, L$. Then,

- let $P_{i+1} = (P_i - \{C\}) \cup \{H \leftarrow L\}$, if, for every ground instantiation θ, $A\theta$ is true in $WFS(P_i)$.
- let $P_{i+1} = P_i - \{C\}$, if, for every ground instantiation θ, $A\theta$ is false in $WFS(P_i)$. □

We call $A\theta$ a *target literal* of the reduction rule.

It is easy to see that unfold/fold transformation together with the reduction rule preserves the well-founded semantics.

Proposition 3.2 [Sek90]

Let $P_0, \cdots, P_N$ be a sequence of transformation where unfolding and folding together with the reduction rule are applied. Then, the well-founded semantics of any program P_N is identical to that of P_0. □

In this case, the dynamic stratification of each atom is not necessarily preserved (see the following example).

Example 3.1 [BF88]

Consider the following program P^{BF}:

$$\begin{aligned} father(a,b)&. \\ father(b,c)&. \\ p(a)&. \\ p(Y) &\leftarrow father(X,Y), \sim p(X) \end{aligned}$$

Note that P^{BF} is not locally stratified. However, we can apply unfolding to the last rule at $father(X,Y)$, obtaining the following program P_1^{BF}:

$$\begin{aligned} father(a,b)&. \\ father(b,c)&. \\ p(a)&. \\ p(b) &\leftarrow \sim p(a) \\ p(c) &\leftarrow \sim p(b) \end{aligned}$$

Since the last two rules can be further simplied by the reduction rule, the original program P^{BF} is reduced to the following equivalent but much simplified (definite!) program:

$$\begin{aligned} father(a,b)&. \\ father(b,c)&. \\ p(a)&. \\ p(c)&. \end{aligned}$$

□

The following result is derived as a corollary of Proposition 3.2. It is the well-founded semantics' counterpart of the result by Bidoit-Froidevaux [BF88], where they considered default theories:

Corollary 3.1 [BF88], [Sek90]

Let P_0 be a (not necessarily locally stratified) logic program and let $P_0, \cdots, P_N$ be a sequence of transformation using unfolding and the reduction rule. Suppose that P_N is a locally stratified program. Then, the well-founded semantics of P_0 is equivalent to the perfect model semantics of P_N. □

4 Preservation of the Stable Model Semantics

We now show that the unfold/fold transformation also preserves the (two-valued) stable model semantics by Gelfond and Lifschitz [GL88].

Proposition 4.1 (Preservation of the Stable Model Semantics) [Sek90]

Let $P_0, \cdots, P_N$ be a transformation sequence starting from an initial program P_0. Then, for any $i\,(i \geq 0)$, P_i has a stable model M if and only if so does P_0. □

The *reduction rule* (wrt the stable model semantics) is defined as in Definition 3.1, simultaneously replacing $WFS(P_i)$ in Definition 3.1 by M, where M is an arbitrary but fixed stable model of an initial program P_0. The reduction rule defined so, however, does not always preserve the stable model semantics in general.

Example 4.1 [VGRS88] Consider the following program P.

$$\begin{array}{lcl} a & \leftarrow & \neg b \\ b & \leftarrow & \neg a \\ p & \leftarrow & \neg p \\ p & \leftarrow & \neg b \end{array}$$

Then, P has a unique stable model, $M = <\{p,a\};\{b\}>$. Since p is true in M, we apply the reduction rule to the third clause of P, obtaining the following program P_1:

$$\begin{array}{lcl} a & \leftarrow & \neg b \\ b & \leftarrow & \neg a \\ p & \leftarrow & \neg b \end{array}$$

Now, P_1 has two stable models; $M_1 = <\{p,a\};\{b\}>$ and $M_2 = <\{b\};\{a,p\}>$. Thus, P_1 has no unique stable model. □

The above example implies that we have to be careful to apply program transformation based on the reduction rule as far as the stable model semantics is concerned.

A *safe* reduction rule (wrt the stable model semantics) is defined to be a reduction rule such that its target literal $A\theta$ is either true or false in $WFS(P_0)$. The following proposition gives a safe condition of applying the reduction rule.

Proposition 4.2 [Sek90]

Let $P_0, \cdots, P_N$ be a sequence of transformation starting from an initial program P_0, where unfolding and folding together with the *safe* reduction rule are applied. Then, for any i ($i \geq 0$), P_i has a stable model M if and only if so does P_0. □

5 Concluding Remarks

There have been several studies on equivalence-preserving transformation of logic programs. Tamaki and Sato's result [TS84] and its extension to stratified programs [Sek89] are already described in section 2. Maher extensively studied various formulations 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 those of Tamaki-Sato do not preserve it in general. 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 it is applicable to rather narrow class of programs and not to general logic programs. In a very recent paper, Gardner and Shepherdson [GS] proposed a framework for unfold/fold transformation of normal programs and they showed that their transformation preserves procedural equivalence based on SLDNF-resolution, as opposed to the well-founded semantics in this paper. It should be noted that their unfold/fold rules are not comparable with our version, since their folding rule [GS] specifies that, when a program P_{i+1} is obtained from P_i by folding $C \in P_i$ by D, D should be in P_i, while, in our framework like [TS84], D is not necessarily in P_i.

The results reported in this paper will be summarized as follows :

1) We have considered a framework for unfold/fold transformation of general logic programs and shown that the rules of unfold/fold transformation preserve both the well-founded semantics and the stable model semantics.

   The framework has eliminated those syntactic restrictions imposed so far in previous work such as [TS84] and [Sek89], thereby giving a natural extension of those work.

2) We have introduced the reduction rule. When used together with unfold/fold transformation, it has been shown to be a useful and powerful deduction rule so that it derives the well-founded semantics' counterpart of the result by Bidoit-Froidevaux [BF88] in default theories.

3) We have shown that the well-founded semantics is always preserved for unfold/fold transformation together with the reduction rule, whereas the stable model semantics is not always so. Since the reduction rule is so simple and straightforward, it seems to be quite a natural expectation that a semantics should be preserved for the reduction rule. The stable model semantics, however, does not satisfy this requirement in general. Several researchers (e.g. [VGRS88] and [Prz90]), have argued that the stable model semantics does not always give an intuitive model of a general logic program. Our result gives another justification for it from the viewpoint of program transformation.

Acknowledgement

This research was done as a part of the Fifth Generation Computer Systems project of Japan. The author would like to thank Dr. K. Fuchi (Director of ICOT) and Dr. R. Hasegawa (Chief of ICOT 5th Laboratory) for the opportunity of doing this research. The author would like to thank Mark Wallace who let him know the work by Bidoit and Froidevaux. Finally, the author

would like to thank Teodor Przymusinski whose comment has initiated this work.

References

[ABW87] 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*, pp. 89–148. Morgan Kaufmann, 1987. Los Altos, CA.

[BD77] R.M. Burstall and J. Darlington. A Transformation System for Developing Recursive Programs. *J. ACM*, Vol. 24, No. 1, pp. 44–67, 1977.

[BF88] N. Bidoit and C. Froidevaux. More on Stratified Default Theories. In *Proc. of European Conference on Artificial Intelligence*, pp. 492–494, 1988.

[Ers77] A. P. Ershov. On the partial computation principle. *Information Processing Letters*, Vol. 6, No. 2, pp. 38–41, 1977.

[Fut71] Y. Futamura. Partial evaluation of computation process – an approach to a compiler-compiler. *Systems, Computers, Controls*, Vol. 2, No. 5, pp. 45–50, 1971.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the Fifth Logic Programming Symposium and Conference*, pp. 1070–1080. MIT Press, 1988.

[GS] P. A. Gardner and J. C. Shepherdson. Unfold/Fold Transformations of Logic Programs. submitted for publication.

[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*, pp. 744–768, Melbourne, 1987.

[Mah86] M.J. Maher. Equivalences of Logic Programs. In *Proceedings of the Third International Conference on Logic Programming*, pp. 410–424, London, 1986. also in Foundations of Deductive Databases and Logic Programming, (edited by Minker, J.), pp. 627-658, Morgan Kaufmann, 1987.

[Prz86] T. C. Przymusinski. On the Semantics of Stratified Deductive Databases. In J. Minker, editor, *Proc. of Workshop on Foundations of Deductive Databases and Logic Programming*, pp. 433–443, 1986. Washington, DC.

[Prz89] T. C. Przymusinski. Every Logic Program Has a Natural Stratification and an Iterated Least Fixed Point Model. In *Proc. Eighth ACM Symposium on Principles of Database Systems*, pp. 11-21, 1989.

[Prz90] T. C. Przymusinski. Extended Stable Model Semantics for Normal and Disjunctive Programs. In *Proc. Seventh International Conference on Logic Programming*, pp. 459–479, 1990.

[Sek89] H. Seki. Unfold/Fold Transformation of Stratified Programs. ICOT Technical Report TR-536, ICOT, 1989. also to appear in J. of Theoretical Computer Science. Its extended abstract appeared in the Sixth ICLP, 1989.

[Sek90] H. Seki. Unfold/Fold Transformation of General Logic Programs. ICOT Technical Report, ICOT, 1990. in preparation.

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

[VG89] A. Van Gelder. The Alternating Fixpoint of Logic Programs with Negation. In *Proc. Eighth ACM Symposium on Principles of Database Systems*, pp. 1–10, 1989.

[VGRS88] A. Van Gelder, K. Ross, and J. S. Schlipf. Unfounded Sets and Well-Founded Semantics for General Logic Programs. In *Proc. Seventh ACM Symposium on Principles of Database Systems*, pp. 211–230, 1988.