Transforming Acyclic Programs

ANNALISA BOSSI
Università di Padova
and
SANDRO ETALLE
Università di Padova and CWI

An unfold/fold transformation system is a source-to-source rewriting methodology devised to improve the efficiency of a program. Any such transformation should preserve the main properties of the initial program: among them, termination. In the field of logic programming, the class of acyclic programs plays an important role in this respect, since it is closely related to the one of terminating programs. The two classes coincide when negation is not allowed in the bodies of the clauses.

We prove that the Unfold/Fold transformation system defined by Tamaki and Sato preserves the acyclicity of the initial program. From this result, it follows that when the transformation is applied to an acyclic program, then the finite failure set for definite programs is preserved; in the case of normal programs, all major declarative and operational semantics are preserved as well. These results cannot be extended to the class of left-terminating programs without modifying the definition of the transformation.

Categories and Subject Descriptors: D.1.6 [**Programming Techniques**]: Logic Programming; F.3.2 [**Logics and Meaning of Programs**]: Semantics of Programming Languages—*denotational and operational semantics*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*logic programming*; I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program transformation*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving —*logic programming*

General Terms: Language, Theory

Additional Key Words and Phrases: Acyclic programs, termination, terminating programs

1. INTRODUCTION

1.1 Motivation

In this article we focus on the unfold/fold transformation system proposed by Tamaki and Sato [1984].

This work has been partially supported by "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo" CNR grant 89.00026.69.
Authors' addresses: A. Bossi, Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, 35131 Padova, Italia; email: bossi@zenone.math.unipd.it; S. Etalle, CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands; email:etalle@cwi.nl.

As the literature shows, a lot of research has been devoted to proving the correctness of the system with respect to the various semantics proposed for logic programs (e.g., see Aravidan and Dung [1993], Kawamura and Kanamori [1988], Seki [1990; 1991; 1993], and Tamaki and Sato [1984]). However the question of the consequences of the transformation on the (universal) termination of the program has not yet been tackled.

Recall that a program is called "*terminating*" if all its SLDNF derivations starting in a ground goal are finite.

Here we follow the approach to termination of Apt and Bezem [1991]. They investigate the class of *acyclic* programs (introduced by Cavedon [1991]) and prove that it is closely related to the one of terminating programs. In fact we have that every acyclic program is terminating [Apt and Bezem 1991] and that every definite, terminating program is acyclic [Bezem 1993]; however, when negation is allowed in the bodies of the clauses, there are programs that are terminating but not acyclic. This is caused either by the presence of floundering derivations or by the fact that, since nonground negative literals might not be selected, some infinite branches of the search tree cannot be explored; see Apt and Bezem [1991] for examples.

We prove that when the initial program of an unfold/fold transformation sequence is acyclic, then the resulting program is acyclic as well.

This has some obvious consequences on the preservation of termination and some semantic repercussions. For definite programs, the transformation preserves the Finite Failure Set. In fact, since acyclic programs are terminating, and since definite programs cannot flounder, their Finite Failure Set coincides with the complement of their success set. For programs with negation, the transformation preserves all the major formalisms, namely, Fitting's model, 2- and 3-valued ground logical consequence of the completion, and, in the nonfloundering cases, the operational semantics based on the SLDNF-resolution: when the program is acyclic they all coincide and thus are preserved by the transformation.

1.2 Structure of the Article

Sections 2 and 3 contain the preliminaries on terminating and acyclic programs and on the Tamaki-Sato unfold/fold transformation system. In Section 4 we prove that the transformation preserves the acyclicity of the initial program; we also discuss the case in which the initial program is left-terminating. In Section 5 we give a brief summary of the semantic properties of acyclic programs and show that they are preserved through the transformation.

1.3 Preliminaries

We assume that the reader is familiar with the basic concepts of logic programming; throughout we use the standard terminology of Loyd [1987] and Apt [1990]. We consider *normal programs*, that is, finite collections of *normal rules*, $A \leftarrow L_1, \ldots, L_m$, where A is an atom and $L_1, \ldots, L_m$ are literals. We say that a clause is *definite* if the body contains only positive

literals (atoms); a definite program is then a program consisting only of definite clauses. Symbols with a $\sim$ on top denote tuples of objects; for instance, $\tilde{x}$ denotes a tuple of variables $x_1, \ldots, x_n$, and $\tilde{x} = \tilde{y}$ stands for $x_1 = y_1 \wedge \ldots \wedge x_n = y_n$. We also adopt the usual logic programming notation that uses "," (comma) instead of $\wedge$; hence a conjunction of literals $L_1 \wedge \ldots \wedge L_n$ will be denoted by $L_1, \ldots, L_n$ or by $\tilde{L}$. Finally, we denote by $Var(E)$ the set of all the variables in an expression E, and by $Ground(P)$ the set of ground instances of the clauses of a program P.

2. TERMINATION

The following notion is crucial.

Definition 2.1. A program is called *terminating* iff all its SLDNF-derivations starting from a ground goal are finite.

Hence, terminating programs are the ones whose SLDNF-trees starting in a ground goal are finite. We now present the approach to the issue of termination followed by Apt and Bezem [1991].

Acyclic Programs

Acyclic programs form a natural subclass of the locally stratified ones; they were introduced by Cavedon [1991] and have been further studied by Apt and Bezem [1991]. To give their definition, first we need the following notion.

Definition 2.2. Let P be a program; a *level mapping* for P is a function $|\ |: B_P \rightarrow \mathbf{N}$ from ground atoms to natural numbers.

For an atom A, $|A|$ denotes the level of A. Following Apt and Bezem [1991], we extend this definition to ground literals by letting $|\neg A| = |A|$.

Definition 2.3. Let $|\ |$ be a level mapping.

—A clause is *acyclic wrt* $|\ |$ iff for every ground instance $A \leftarrow L_1, \ldots, L_k$ of it, and for each i, $|A| > |L_i|$;
—A program P is *acyclic wrt* $|\ |$ iff all its clauses are. P is called *acyclic* if it is acyclic wrt some level mapping.

Following Bezem [1993], we introduce the concept of boundedness, which applies also to nonground atoms.

Definition 2.4. Let $|\ |$ be a level mapping. A literal L is called *bounded* wrt $|\ |$ if $|\ |$ is bounded on the set $[L]$ of ground instances of L. A goal is called *bounded* wrt $|\ |$ iff all its literals are.

Example 2.5 [*Apt and Pedreschi* 1993]. Consider the program *member*.

$$P = \{\ member(X, [Y|Xs]) \leftarrow member(X, Xs).$$
$$member(X, [X|Xs]).\ \}$$

We adopt the standard list notation and define the function $|\ |_l$, called *listsize*, which assigns natural numbers to ground terms as follows:

—$|t|_l = 1$ if t is not of the form $[x_1|x_s]$ (this takes also care of the case $t = [\ ]$).
—$|[x_1|x_s]|_l = 1 + |x_s|_l$.

We can define now the level mapping $|\ |$ for the *member* program: $|member(t, s)| = |s|_l$. It is easy to see that program *member* is acyclic wrt $|\ |$ and that if l is a list (by this we mean $l = [x_1, \ldots, x_n]$, where the x_i's need not be ground), then $member(t, l)$ is a bounded atom.

Now we can relate acyclic and terminating programs.

THEOREM 2.6 [APT AND BEZEM 1991]. *Let P be a program and G be a goal. If there exists a level mapping $|\ |$ such that P is acyclic wrt $|\ |$ and G is bounded wrt $|\ |$ then all SLDNF-derivations of $P \cup \{G\}$ are finite.*

Since ground goals are bounded, this implies the following.

THEOREM 2.7 [APT AND BEZEM 1991]. *If P is an acyclic program then P is terminating.*

Apt and Bezem [1991] state that the converse of Theorem 2.7 holds in the case that no SLDNF-derivation starting in a ground goal contains a goal with a nonground negative literal in it, and that since that condition is quite constraining, the result itself is too weak to be formalized. However, it is significant at least for the case that we restrict our attention to definite programs; in fact in Bezem [1993] we find the following:

THEOREM 2.8 [BEZEM 1993]. *Let P be a definite program; then P is terminating iff P is acyclic.*

From the procedural point of view, acyclic programs enjoy the following important property: the two most prominent approaches, namely, the SLDNF-resolution (see Lloyd [1987] and Apt [1990]) and the SLS resolution from Przymusinski [1989], coincide when applied to acyclic programs. For the semantic properties of acyclic programs refer to Section 5.

3. UNFOLD / FOLD TRANSFORMATIONS

We give now the definition of the unfold/fold transformation sequence that was first given by Tamaki and Sato [1984] for definite programs and then used by Seki [1990; 1993] for normal programs. Here we present it as it is in Seki [1993]. All definitions are given modulo reordering of the bodies of the clauses, and standardization apart is always assumed.

Definition 3.1 (*Initial Program*). We call a normal program P_0 an *initial program* if the following two conditions are satisfied:

(I1) P_0 is divided into two disjoint sets $P_0 = P_{new} \cup P_{old}$;
(I2) All the predicates which are defined in P_{new} occur neither in P_{old} nor in the bodies of the clauses in P_{new}.

The predicates defined in P_{new} are called *new* predicates, while those defined in P_{old} are the *old* predicates. Clauses in P_{new} will also be referred to as *defining* clauses.

Example 3.2 (*Part* 1) [*Seki* 1993]. Let P_0 be the following program

$$
\begin{array}{lll}
P_0 = DB \cup \{ \quad c_1: & path(X,[X]) & \leftarrow node(X). \\
c_2: & path(X,[X|Xs]) & \leftarrow arc(X,Y), path(Y,Xs). \\
c_3: & goodlist([\,]). & \\
c_4: & goodlist([X|Xs]) & \leftarrow \neg\, bad(X), goodlist(Xs). \\
c_5: & goodpath(X,Xs) & \leftarrow path(X,Xs), goodlist(Xs). \quad \}
\end{array}
$$

where predicates *node*, *arc*, and *bad* are defined in DB by a set of unit clauses. Predicate $goodpath(X, Xs)$ can be employed for finding a path Xs starting from the node X which does not contain "bad" nodes. Let $P_{old} = \{c_1, \ldots, c_4\} \cup DB$ and $P_{new} = \{c_5\}$; thus *goodpath* is the only new predicate.

Unfolding an atom in the body of a clause consists in applying a resolution step to the considered atom in all possible ways. This operation is basic to all the transformation systems.

Definition 3.3 (*Unfold*). Let cl: $A \leftarrow H, \tilde{K}$. Be a clause of a normal program P, where H is an atom. Let $\{H_1 \leftarrow \tilde{B}_1, \ldots, H_n \leftarrow \tilde{B}_n\}$ be the set of clauses of P whose heads unify with H, by mgu's $\{\theta_1, \ldots, \theta_n\}$.

—*Unfolding H in cl* consists of substituting cl with $\{cl'_1, \ldots, cl'_n\}$, where, for each i, $cl'_i = (A \leftarrow \tilde{B}_i, \tilde{K})\theta_i$. $unfold\ (P, cl, H) \stackrel{\text{def}}{=} P \setminus \{cl\} \cup \{cl'_1, \ldots, cl'_n\}$.

Example 3.2 (*Part* 2). By unfolding the atom $path(X, Xs)$ in the body of c_5, we obtain

$$
\begin{array}{lll}
c_6: & goodpath(X,[X]) & \leftarrow node(X), goodlist([X]). \\
c_7: & goodpath(X,[X|Xs]) & \leftarrow arc(X,Y), path(Y,Xs), goodlist([X|Xs]).
\end{array}
$$

Both clauses can be further unfolded (c_6 twice); the resulting clauses are

$$
\begin{array}{lll}
c_8: & goodpath(X,[X]) & \leftarrow node(X), \neg\, bad(X).. \\
c_9: & goodpath(X,[X|Xs]) & \leftarrow arc(X,Y), path(Y,Xs), \\
 & & \neg\, \mathrm{bad(X)}, \mathrm{goodlist(Xs)}.
\end{array}
$$

Let $P_1 = \{c_1, \ldots, c_4, c_8, c_9\} \cup DB$.

Folding is the inverse of unfolding when one single unfolding is possible. It consists in substituting an atom A for an equivalent conjunction of literals $\tilde{K}$ in the body of a clause c. This operation is used in the transformation systems in order to simplify unfolded clauses and to detect implicit recursive definitions. In the literature there are different definitions for this operation. This is due to the fact that it does not always preserve the declarative semantics, and thus its use must be restricted by some applicability conditions. Depending on the approach, such conditions can be either a constraint on how to sequentialize the operations while transforming the program

[Kawamura and Kanamori 1988; Tamaki and Sato 1984], or can be expressed in terms of semantic properties of the program, independently from its transformation history [Bossi and Cocco 1993; Maher 1987].

In the method proposed by Tamaki and Sato [1984], the transformation sequence and the folding operation are defined in terms of each other.

Definition 3.4 (*Transformation Sequence*). A *transformation sequence* is a sequence of programs $P_0, \ldots, P_n$, $n \geq 0$, such that each program P_{i+1}, $0 \leq i < n$, is obtained from P_i by unfolding or folding a clause of P_i.

Definition 3.5 (*Folding*). Let $P_0, \ldots, P_i$, $i \geq 0$, be a transformation sequence, $c : A \leftarrow \tilde{K}', \tilde{J}$. a clause in P_i and $d : D \leftarrow \tilde{K}$. a clause in P_{new}. Let Y be the set of variables of $\tilde{K}'$ not in A, $\tilde{J}$, and X be the set of all the variables occurring in the clause d. If there exists a substitution τ whose domain is the set X, such that the following conditions hold:

(F1) $\tilde{K}\tau = \tilde{K}'$;

(F2) τ renames with distinct variables in Y the variables in $\tilde{K}$ not in D;

(F3) d is the only clause in P_{new} whose head is unifiable with D; and

(F4) one of the following two conditions holds:
  (1) the predicate in A is an old predicate; or
  (2) c is the result of at least one unfolding in the sequence $P_0, \ldots, P_i$;

then *folding* $D\tau$ *in* c *in* P_i consists of substituting c' for c in P_i, where

$$head(c') \stackrel{\text{def}}{=} A$$

$$body(c') \stackrel{\text{def}}{=} D\tau, \tilde{J}.$$

$$fold(P_i, D\tau, c) \stackrel{\text{def}}{=} (P_i \setminus \{c\}) \cup \{c'\}.$$

Example 3.2 (*Part* 3). We can now fold the body of c_9, using c_5 as folding clause; the resulting program is $P_2 = DB \cup \{c_1, \ldots, c_4, c_{10}\}$, where c_{10} is the following clause:

$$c_{10}: \quad goodpath(X, [X|Xs]) \leftarrow arc(X, Y), \neg\, bad(X), goodpath(Y, Xs).$$

Notice that because of this operation the definition of *goodpath* is now recursive.

The transformation enjoys the following important properties.

THEOREM 3.6. *Let* $P_0, \ldots, P_n$ *be a transformation sequence.*

(1) *If* P_0 *is a definite program then*
  —*The least Herbrand models of the initial and final programs coincide* [*Tamaki and Sato* 1984]
  —*The computed answers substitution semantics of the initial and final programs coincide* [*Kawamura and Kanamori* 1988]

(2) *If* P_0 *is a normal program, then*
  —*The Stable models of the initial and final programs coincide* [*Seki* 1990]

—*The Well-Founded models of the initial and final programs coincide* [*Seki* 1993]
—*Under a further mild assumption on the initial program; if the initial program is stratified then the final program is stratified, and their Perfect models coincide* [*Seki* 1989].
—*The Semantic Kernels of the initial and final program coincide; this implies also that the Stable model semantics, the preferred extension semantics, the stationary semantics, and the stable theory semantics of the initial and the final programs coincide* [*Aravidan and Dung* 1993].

MODIFIED FOLDING

In order to make this article more self-contained, we have to mention that the transformation does not preserve the Finite Failure Set of the initial (definite) program. More precisely, we have that the Finite Failure Set of the final program is contained in the one of the initial program, but, in general, not vice-versa. This is shown by the following example.

Example 3.7. Let P_0 be the following program:

$$P_0 = \{ \quad c_1: \quad p \quad \leftarrow q, h(X). \qquad c_2: \quad h(s(X)) \quad \leftarrow h(X). \quad \}$$

Here we use the following partition: $P_{new} = \{c_1\}$, $P_{old} = \{c_2\}$; notice that there is no definition for predicate q, so the queries $P \cup \{\leftarrow q\}$ and $P \cup \{\leftarrow p\}$ will always fail. Now if we unfold atom $h(X)$ in the body of the first clause, we obtain a renaming of the clause itself, namely:

$$P_1 = \{c_2\} \cup \{c_3: \quad p \leftarrow q, h(Y).\}$$

c_3 satisfies condition (F4.2), so it can be folded, using c_1 as folding clause. The resulting program is:

$$P_2 = \{c_2\} \cup \{c_4: \quad p \leftarrow p.\}$$

Now the query $P_2 \cup \{\leftarrow P\}$ does not terminate.

The problem of the correctness of the operation with respect to the finite failure set was pointed out by Seki, who modified the applicability conditions of the folding operation as follows:

Definition 3.8 (*Modified Folding*) [*Seki* 1991]. The *modified* folding operation is defined exactly as in Definition 3.5, with the exception of condition (F4.2), which is replaced by the following:

(F4.2′) All the atoms in $\bar{K}'$ are the result of some previous unfold operation.

This definition first appeared in Seki [1989]. It is easy to see that when (F4.2′) holds, then (F4.2) holds as well, hence that the *modified folding* operation enjoys all the properties that were proven for the folding operation. Seki proved that modified folding preserves the Finite Failure Set of a definite program [Seki 1989; 1991]; later, on a work that extends this definition to full first-order programs [Sato 1990], Sato proved the correctness of the system with respect to Kunen's semantics.

4. TRANSFORMING ACYCLIC PROGRAMS

We show now that if the initial program of a transformation sequence is acyclic then the resulting program is acyclic as well. We do this by showing that there exists a level mapping with respect to which every program in the transformation sequence is acyclic.

Notation

Let $P_0, \ldots, P_n$ be the transformation sequence we are considering. Since P_0 is acyclic, then it is acyclic wrt some level mapping, say $\| \ \|$; moreover, there in no loss of generality in assuming that $\| \ \|$ does not take value zero on any atom. Let *nf* be the number of foldings that are going to be performed in the sequence (which we assume greater than zero), and let *maxbody* be the maximum number of literals that a body of a clause of P_0 contains, augmented by one. We also suppose that $maxbody > 1$, since it is not possible to perform any unfold or fold operation on a program consisting solely of unit clauses.

We now define a new level mapping $| \ |$ for P_0.

Definition 4.1. Let P_0 be acyclic wrt the level mapping $\| \ \|$. The level mapping $| \ |$ is defined as follows. Let A be a ground atom.

—If A is an *old* atom then we let $|A| = nf \cdot maxbody^{\|A\|}$.

— If A is an *new* atom then we distinguish two subcases:

(a) If A unifies with the head of only one clause of P_{new}, $N \leftarrow B_1, \ldots, B_n$, suppose that $A = N\theta$, since $B_1, \ldots, B_n$ are *old* atoms, we have that $| \ |$ is already defined on their ground instances, so we set

$$|A| = |N\theta| = sup\{\textstyle\sum_{i=1}^{n} |B_i\theta\gamma| \mid Dom(\gamma) = Var(B_1\theta, \ldots, B_n\theta)\} + 1.$$

(b) (This case is of no relevance for the proof, as, because of condition (F3), we are interested in computing the level mapping of atoms that unify with the head of only one clause of P_{new}; but we have to extend $| \ |$ in a consistent way.) If A unifies with the head of a (nonunit) set of clauses $\{N_1 \leftarrow B_{1,1}, \ldots, B_{1,n(1)}, \ldots, N_j \leftarrow B_{j,1}, \ldots, B_{j,n(j)}\} \subseteq P_{new}$, suppose that $A = N_i\theta_i$; we define

$$|A| = sup\{\textstyle\sum_{k=1}^{n(i)} |B_{i,k}\theta_i\gamma|\} + 1$$

where i ranges in $[1, \ldots, j]$ and γ ranges over the ground substitutions whose domain is $Var(B_{i,1}\theta_i, \ldots, B_{i,n(i)}\theta_i)$

Here the *sup* of an empty set is assumed to be 0. $| \ |$ is obviously a level mapping, since it is defined and finite on each ground atom.

In order to prove that each of the programs in the transformation sequence is acyclic wrt $| \ |$, we need the following simple but technical lemma.

LEMMA 4.2. *For nonzero integers* $nf, n, n_1, \ldots, n_k$, *if* $1 < k < maxbody$ *then*

$$if\ n > sup\{n_1, \ldots, n_k\},\ then\ nf \cdot maxbody^n > nf + \textstyle\sum_{j=1}^{k} nf \cdot maxbody^{n_j}.$$

PROOF.

$$nf + \sum_{j=1}^{k} nf \cdot maxbody^{n_j} \leq nf + nf \cdot k \cdot maxbody^{sup\{n_j\}}$$

Since $k < maxbody$

$$\leq nf + nf \cdot (maxbody - 1) \cdot maxbody^{sup\{n_j\}}$$
$$= nf + nf \cdot maxbody^{sup\{n_j\}+1} - nf \cdot maxbody^{sup\{n_j\}}.$$

Since $maxbody > 0$ and $n > sup\{n_j\}$,

$$\leq nf \cdot maxbody^{n} + nf - nf \cdot maxbody^{sup\{n_j\}}$$
$$= nf \cdot maxbody^{n} + nf \cdot (1 - maxbody^{sup\{n_j\}}).$$

Since all integers are nonzero and $maxbody > 1$, we have $1 - maxbody^{sup\{n_j\}} < 0$. This proves the lemma. □

LEMMA 4.3. *For each P_i in the transformation sequence the level mapping $|\ |$ of Definition 4.1 satisfies the following*:

(a) for each ground instance of a *defining* clause $H \leftarrow B_1, \ldots, B_k.$,

$$|H| > |B_1| + \cdots + |B_k|;$$

(b) for any other clause $H \leftarrow B_1, \ldots, B_k.$ in $Ground(P_i)$,

$$|H| > |B_1| + \cdots + |B_k| + nf_i.$$

Where for each i, nf_i is the number of folding operations that will be performed in the sequence from P_i to P_n.

PROOF. The proof proceeds by induction on the index i.

Base Case P_0. Let $c : H \leftarrow B_1, \ldots, B_k.$ be a clause of $Ground(P_0)$. If $k = 0$ then the result holds trivially. So we assume $k > 0$. We have to distinguish two cases:

If H is a *new* predicate, then c is an instance of a *defining* clause, and condition (a) is then trivially satisfied by the definition of $|\ |$.

If H is an *old* predicate, then, since $\|H\| > sup\{\|B_j\|\}$ and since $1 < k < maxbody$, the result follows from Lemma 4.2.

Induction Step P_{i+1}. For those clauses that P_i and P_{i+1} have in common, the result follows from the inductive hypothesis and the fact that $nf_{i+1} \leq nf_i$. Hence we can focus on those clauses that were introduced or modified in the last transformation step (from P_i to P_{i+1}). We distinguish upon the operation that has been used for going from P_i to P_{i+1}.

Unfolding. Let

$d : H \leftarrow B', L_1, \ldots, L_h.$ be the unfolded clause, and

$c : B \leftarrow B_1, \ldots, B_k.$ be one of the unfolding ones.

Let also $\theta = mgu(B, B')$, then the resulting clause is

$$H\theta \leftarrow B_1\theta, \ldots, B_k\theta, L_1\theta, \ldots, L_h\theta.$$

Since $nf_{\iota+1} = nf_\iota$, in order to prove the thesis, we have to prove that, for each γ

$$|H\theta\gamma| > |B_1\theta\gamma| + \ldots + |B_k\theta\gamma| + |L_1\theta\gamma| + \ldots + |L_h\theta\gamma| + nf_\iota. \quad (1)$$

We have to distinguish two cases:

First we suppose that d is a *defining* clause. Then B is an old predicate, and clause c satisfies condition (b); hence

$$|B\theta\gamma| > |B_1\theta\gamma| + \ldots + |B_k\theta\gamma| + nf_\iota.$$

On the other hand, clause d satisfies condition (a), hence

$$|H\theta\gamma| > |B'\theta\gamma| + |L_1\theta\gamma| + \ldots + |L_h\theta\gamma|.$$

Since $B'\theta\gamma = B\theta\gamma$ this proves (1).

Second, we consider the case in which d is not a *defining* clause. Hence d satisfies condition (b), and we have that

$$|H\theta\gamma| > |B'\theta\gamma| + |L_1\theta\gamma| + \cdots + |L_h\theta\gamma| + nf_\iota.$$

Since clause c must satisfy either (a) or (b), we have also that

$$|B\theta\gamma| > |B_1\theta\gamma| + \ldots + |B_k\theta\gamma|.$$

Since $B'\theta\gamma = B\theta\gamma$ this proves again (1).

Folding. Suppose that:

$$c\colon H \leftarrow B'_1, \ldots, B'_k, L_1, \ldots, L_h. \text{ is the folded clause of } P_\iota,$$
$$d\colon N \leftarrow B_1, \ldots, B_k \text{ is the folding clause of } P_{new}.$$

Hence $(B'_1, \ldots, B'_k) = (B_1, \ldots, B_k)\tau$, and $H \leftarrow N\tau, L_1, \ldots, L_h.$ is the clause we add to $P_{\iota+1}$.

By (F4), c is not a *defining* clause; hence its ground instances have to satisfy condition (b), that is, for each γ, $|H\gamma| > |B'_1\gamma| + \ldots + |B'_k\gamma| + |L_1\gamma| \ldots + |L_h\gamma| + nf_\iota$. Since $(B'_1, \ldots, B'_k) = (B_1, \ldots, B_k)\tau$, this implies that, for each γ,

$$|H\gamma| > |B_1\tau\gamma| + \ldots + |B_k\tau\gamma| + |L_1\gamma| \ldots + |L_h\gamma| + nf_\iota,$$

where τ is a renaming on the variables in $W = Var(B_1, \ldots, B_k) \setminus Var(N)$. Let $Z = W\tau$, by the assumptions in (F2), $Var(H, L_1, \ldots, L_h) \cap Z = \varnothing$. Hence we can split γ into two independent orthogonal substitutions: $\gamma = \gamma_{|Z}\gamma_{|\overline{Z}}$, where $\gamma_{|Z}$ is γ restricted to Z, and $\gamma_{|\overline{Z}}$ is γ restricted to the complement of Z. And we have that, for each γ,

$$|H\gamma_{|\overline{Z}}| > |B_1\tau\gamma_{|\overline{Z}}\gamma_{|Z}| + \ldots + |B_k\tau\gamma_{|\overline{Z}}\gamma_{|Z}| + |L_1\gamma_{|\overline{Z}}| + \ldots + |L_h\gamma_{|\overline{Z}}| + nf_\iota.$$

Since this holds for any choice of $\gamma_{|Z}$, for each γ

$$|H\gamma_{|\overline{Z}}| > sup\left\{\Sigma_{i=1}^{k}|B_i\tau\gamma_{|\overline{Z}}\eta| \mid Dom(\eta) = Z\right\} + |L_1\gamma_{|\overline{Z}}| + \ldots + |L_h\gamma_{|\overline{Z}}| + nf_i.$$

Now by (F3) d is the only clause whose head unifies with $N\tau$; it follows that, by the definition of $|\ |$, $|N\tau\gamma_{|\overline{Z}}| = sup\{\Sigma_{i=1}^{k}|B_i\tau\eta|\} + 1$; hence we have that, for each γ,

$$|H\gamma_{|\overline{Z}}| > |N\tau\gamma_{|\overline{Z}}| + |L_1\gamma_{|\overline{Z}}| + \ldots + |L_h\gamma_{|\overline{Z}}| + nf_i - 1.$$

Now the variables of Z do not occur in any atom of this clause, we have that, for each γ

$$|H\gamma| > |N\tau\gamma| + |L_1\gamma| + \ldots + |L_h\gamma| + nf_i - 1.$$

Since this is a folding step, $nf_{i+1} < nf_i$, and hence we have that (b) is satisfied in P_{i+1}. □

This implies immediately the desired conclusion:

COROLLARY 4.4. *Let $P_0, \ldots, P_n$ be a transformation sequence; then*

(a) *if P_0 is acyclic then P_n is. In the case that P_0 is a definite program, this can be restated as follows:*

(b) *if P_0 is definite and terminating, then P_n is.*

PROOF. It follows at once from Lemma 4.3 □

TRANSFORMING LEFT-TERMINATING PROGRAMS

One would like condition (b) in Corollary 4.4 to hold also in the case of *left-terminating* programs, which are those programs whose LDNF (SLDNF with leftmost selection rule) derivations starting in a ground goal are finite. *Left-terminating* programs form an important superclass of the terminating programs, and, as pointed out by Apt and Pedreschi [1993], there are natural left-terminating programs that are not terminating. However, left-termination is not preserved by the transformation system. In fact, if we consider the three programs P_0, P_1, P_2 of Example 3.7, we have that P_0 and P_1 are left-terminating, while P_2 is not.

In general left-termination is not preserved even when Seki's (more restrictive) *modified* folding operation is used. This is shown by the following example.

Example 4.5. Let P_0, be the following program:

$$\begin{array}{lll} P_0 = \{ \ c_1\colon \ d(X) & \leftarrow h(X), q(X). \\ \quad c_2\colon \ p & \leftarrow q(X), h(X). \\ \quad c_3\colon \ q(s(0)). & \\ \quad c_4\colon \ h(s(X)) & \leftarrow h(X). \qquad \} \end{array}$$

where we adopt the following partition: $P_{new} = \{c_1\}$, $P_{old} = \{c_2, c_3, c_4\}$. It is easy to verify that the program is left-terminating. Since the head of c_2 is an old predicate (and then (F4.1) is satisfied), we can fold $q(X), h(X)$ in the body of c_2. The resulting program is

$$P_1 = \{c_1, c_3, c_4\} \cup \{c_5: \quad p \leftarrow d(X)\}.$$

Now the goal $P_1 \cup \{\leftarrow p\}$ originates an infinite LDNF-derivation.

In this case the problem is due to the fact that the definition of transformation sequence is given modulo reordering of the bodies of the clauses, and the operation of reordering itself does not preserve left-termination.

It can be argued then that what we have to do is to start by adopting the *modified* folding instead of the one of Tamaki-Sato and by restating the definition of unfolding and folding so that the order of the literals in the bodies of the clauses is taken into account. That is indeed a possible approach; however a fold operation so defined would be of far more limited applicability than the present one; this holds not only because the *modified folding* is more restrictive than the ordinary one, but mainly because we would have to require that the literals that are going to be folded are all found next to each other in the exact same sequence as in the body of the folding clause. This is often not the case, in particular, when the folded clause is the result of some previous unfold operation; notice that this is what happens in Example 3.2.

Nevertheless, we can relax the requirement of the acyclicity of the initial program, by exploiting the result in a modular way. First we need the following definition.

Definition 4.6. Let $P_0, \ldots, P_n$ be a transformation sequence, and let $P_0 = Q_0 \cup R$. We say that the transformation is performed *within* Q_0 if there exist programs $Q_1, \ldots, Q_n$ such that, for each i,

—$P_i = Q_i \cup R$;
—No clause of R is used as a folding or unfolding clause.

Now we have to use the concept of *acceptable* programs, introduced by Apt and Pedreschi [1993]. Here the notation becomes more cumbersome as the notion of acceptability is bound both to a level mapping and to a (not necessarily Herbrand) model. For the definition we refer to Apt and Pedreschi [1993]. Informally, acceptable are to left-terminating programs what acyclic are to terminating ones; in fact Apt and Pedreschi [1993] prove that, in cases of nonfloundering programs, the classes of acceptable and of left-terminating programs coincide.

Part (a) of Corollary 4.4 can then be restated as follows.

PROPOSITION 4.7. *Let $P_0, \ldots, P_n$ be a transformation sequence. Suppose that P_0 is acceptable wrt the level mapping $|\ |$ and the model M. If there exists a program $Q_0 \subseteq P_0$ such that Q_0 is acyclic wrt $|\ |$ and the transformation is performed within Q_0, then each P_i is acceptable.*

PROOF. It is a standard extension of the proof of Lemma 4.3. □

That is, if the initial program is acceptable (with respect to some model and some level mapping) and if the transformation is performed *within* a subset of P_0 which is also acyclic (with respect to the same level mapping), then the resulting program is acceptable (hence left-terminating) as well.

5. SEMANTIC CONSEQUENCES

5.1 Preliminaries: Three-Valued Model Semantics

In this section we refer to a fixed but unspecified language $\mathscr{L}$ that we assume contains all the functions symbols and the predicate symbols of the programs that we consider. We refer also to the usual Clark's completion definition, *Comp*(P) [Clark 1978], which consists of the completed definition of each predicate together with CET, Clark's Equality Theory, which is needed in order to interpret " = " correctly. When working with 3-valued logic, the same definition applies, with the only difference that the connective $\leftrightarrow$, used in the completed definitions of the predicates, is replaced with $\Leftrightarrow$, Łucasiewicz's operator of "having the same truth value." In this context, we have that a *three-valued* (or *partial*) *interpretation* is a mapping from the ground atoms of $\mathscr{L}$ into the set {*true*, *false*, *undefined*}.

We can now give the definition of Fitting's operator [Fitting 1985].

Definition 5.1.1. Let P be a normal program, I a three-valued interpretation, A a ground atom; $\Phi_P(I)$ is the three-valued interpretation defined as follows:

—A is *true* is $\Phi_P(I)$, iff there exists a clause c: $A \leftarrow \tilde{L}$. in *Ground*(P) such that $\tilde{L}$ is *true* in I;

—A is *false* in $\Phi_P(I)$, iff for all clauses c: $A \leftarrow \tilde{L}$. in *Ground*(P), $\tilde{L}$ is *false* in I.

We adopt the standard notation: $\Phi_P^{\uparrow 0}$ is the interpretation that maps every ground atom into the value *undefined*, $\Phi_P^{\uparrow \alpha+1} = \Phi_P(\Phi_P^{\uparrow \alpha})$, $\Phi_P^{\uparrow \alpha} = \bigcup_{\delta < \alpha} \Phi_P^{\uparrow \delta}$, when α is a limit ordinal. Φ_P is a monotonic operator. It follows that its Kleene's sequence is monotonically increasing and that it converges to the least fixpoint of Φ_P. Hence there always exists an ordinal α such that $lfp(\Phi_P) = \Phi_P^{\uparrow \alpha}$. Since Φ_P is monotone but not continuous, α could be greater than ω.

Φ_P characterizes the three-valued semantics of *Comp*(P); in fact Fitting [1985] shows that the three-valued models of P are exactly the fixpoints of Φ_P; it follows that any program has a least three-valued Herbrand model. This model is usually referred to as Fitting's model.

5.2 Semantics of Acyclic Programs

From the point of view of declarative semantics, acyclic programs enjoy various relevant properties. Before stating them, we need to introduce some domain closure axioms, often referred to as "weak domain closure axioms."

Definition 5.2.1. $DCA_{\mathscr{L}}$ is the axiom $\exists \tilde{y}_1(x = f_1(\tilde{y}_1)) \vee \ldots \vee \exists \tilde{y}_r(x = f_r(\tilde{y}_r))$. where $f_1, \ldots, f_r$ are all the function symbols in the language $\mathscr{L}$ and $\tilde{y}_i$ are tuples of variables of the appropriate arity.

Now we summarize some of the semantic properties of acyclic programs. For the definition and the properties of the Well-Founded model model semantics we refer to Gelder et al. [1988].

THEOREM 5.2.2. *Let P be an acyclic program, and let $M = \Phi_P^{\uparrow \omega}$. Then M is total, that is, no atom is undefined in it; moreover*

(i) *M is the unique fixpoint of Φ_P; hence it is the unique three-valued (and also two-valued) Herbrand model of $Comp(P)$ and coincides with Fitting's model of P.*
(ii) *M coincides with the Well-Founded model of P;*
(iii) *M coincides with the set of ground atomic logical consequences of $Comp(P) \cup DCA_{\mathscr{L}}$ in 2- and 3-valued logic;*
(iv) *for all ground atoms A such that no SLDNF-derivation of $P \cup \{\leftarrow A\}$ flounders,*
—A is true in M iff there exists an SLDNF-refutation for $P \cup \{\leftarrow A\}$;
—A is false in M iff $P \cup \{\leftarrow A\}$ has a finitely failed SLDNF tree.

PROOF. The fact that M is total and statement (i) are consequences of Lemma 2.6 and Theorem 4.4 in Apt and Bezem [1991]; more general statements are also proven in Apt and Pedreschi [1993], where the case of *acceptable* programs is considered; (ii) is a consequence of (i) and the fact that the Well-Founded model is also a three-valued model of $Comp(P)$ [Gelder et al. 1988]; (iii) and (iv) are consequences of Theorem 4.4 in Apt and Bezem [1991]. □

5.3 Semantics of Transformed Programs

An immediate consequence of Theorem 5.2.2 is the following.

LEMMA 5.3.1. *Let $P_0, \ldots, P_n$ be a transformation sequence; suppose that P_0 is acyclic; then $\Phi_{P_0}^{\uparrow \omega} = \Phi_{P_n}^{\uparrow \omega}$.*

PROOF. By Theorem 5.2.2, for each i, the Well-Founded model of P_i coincides with $\Phi_{P_i}^{\uparrow \omega}$, and by Proposition 4.1 in Seki [1993], the Well-Founded models of P_0 and P_n coincide. □

Because of Theorem 5.2.2, Corollary 4.4 has also some semantic consequences, the most relevant of which are:

COROLLARY 5.3.2. *Let $P_0, \ldots, P_n$ be a iransformation sequence; suppose that P_0 is acyclic; then*

(a) *the Fitting's models of P_0 and of P_n coincide;*
(b) *the set of ground logical consequences of $Comp(P_0) \cup DCA_{\mathscr{L}}$ and of $Comp(P_n) \cup DCA_{\mathscr{L}}$ coincide;*

(c) *for all ground atoms A such that no SLDNF-derivation of* $P_0 \cup \{\leftarrow A\}$ *and of* $P_n \cup \{\leftarrow A\}$ *flounders,*
—*there exists an SLDNF-refutation for* $P_0 \cup \{\leftarrow A\}$ *iff there exists one for* $P_n \cup \{\leftarrow A\}$,
—*all SLDNF trees for* $P_0 \cup \{\leftarrow A\}$ *are finitely failed iff all SLDNF trees for* $P_n \cup \{\leftarrow A\}$ are; in particular we have that

(d) *if* P_0 *is definite, then its Finite Failure Set coincides with the one of* P_n.

This shows that if the initial program is acyclic, then the transformation enjoys most of the properties that were proven for Seki's more restrictive modified folding. In some situations this can be useful for relaxing the applicability of the folding operation.

ACKNOWLEDGMENTS

The authors express their gratitude to K. R. Apt and to Maurizio Gabbrielli for their useful suggestions.

REFERENCES

APT, K. R. 1990. Introduction to logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B. *Formal Models and Semantics*. Elsevier, Amsterdam.

APT, K. R. AND BEZEM, M. 1991. Acyclic programs. *New Gen. Comput. 9*, 3, 335–363.

APT, K. R. AND PEDRESCHI, D. 1993. Reasoning about termination of pure Prolog programs. *Inf. Comput. 106*, 1, 109–157.

ARAVIDAN, C. AND DUNG, P. M. 1993. On the correctness of unfold/fold transformation of normal and extended logic programs. Tech. Rep., Div. of Computer Science, Asian Inst. of Technology, Bangkok, Thailand.

BEZEM, M. 1993. Strong termination of logic programs. *J. Log. Program. 15*, 1, 79–97.

BOSSI, A. AND COCCO, N. 1993. Basic transformation operations which preserve computed answer substitutions of logic programs. *J. Log. Program. 16*, 1, 47–87.

CAVEDON, L. 1991. Acyclic programs and the completeness of SLDNF-resolution. *Theor. Comput. Sci. 86*, 1, 81–92.

CLARK, K. L. 1978. Negation as failure rule. In *Logic and Data Bases*, H. Gallaire and G. Minker, Eds. Plenum Press, New York, 293–322.

FITTING, M. 1985. A Kripke-Kleene semantics for logic programs. *J. Logic. Program. 2*, 4, 295–312.

GELDER, A. V., ROSS, K., AND SCHLIPF, J. S. 1988. Unfounded sets and the well-founded semantics for general logic programs. In *Proceedings of the 7th ACM Symposium on Principles of Database Systems*. ACM, New York, 211–230.

KAWAMURA, T. AND KANAMORI, T. 1988. Preservation of stronger equivalence in unfold/fold logic programming transformation. In *Proceedings of the International Conference on Fifth Generation Computer Systems*. Institute for New Generation Computer Technology, Tokyo, 413–422.

LLOYD, J. W. 1987. *Foundations of Logic Programming*. 2nd ed. Springer-Verlag, Berlin.

MAHER, M. 1987. Correctness of a logic program transformation system. IBM Res. Rep. RC13496, T. J. Watson Research Center, Yorktown Heights, N.Y.

PRZYMUSINSKI, T. 1989. Every logic program has a natural stratification and an iterated least fixed point model. In *Proceedings of the 8th Symposium on Principles of Database Systems*. ACM, New York, 11–21.

SATO, T. 1990. An equivalence preserving first order unfold/fold transformation system. In the *2nd International Conference on Algebraic and Logic Programming* (Nancy, France, Oct.) Lecture Notes in Computer Science, vol. 463. Springer-Verlag, Berlin, 175–188.

SEKI, H. 1993. Unfold/fold transformation of general logic programs for the Well-Founded semantics. *J. Logic Program. 16*, 1, 5–23.

SEKI, H. 1991. Unfold/fold transformation of stratified programs. *Theor. Comput. Sci. 86*, 1, 107–139.

SEKI, H. 1990. A comparative study of the well-founded and stable model semantics: Transformation's viewpoint. In the *Workshop on Logic Programming and Non-Monotonic Logic* (Austin, Tex., Oct.), D. P. W. Marek, A. Nerode, and V. Subrahmanian, Eds., Association for Logic Programming and Mathematical Sciences Inst., Cornell Univ., 115–123.

SEKI, H. 1989. Unfold/fold transformation of stratified programs. In the *6th International Conference on Logic Programming*, G. Levi and M. Martelli, Eds. MIT Press, Cambridge, Mass., 554–568.

TAMAKI, H. AND SATO, T. 1984. Unfold/fold transformations of logic programs In *Proceedings of the 2nd International Conference on Logic Programming* (Uppsala Univ.), 127–139.

Received December 1993, revised February 1994; accepted March 1994