Unfold/Fold Transformations For Definite Clause Programs*

Manolis Gergatsoulis and Maria Katzouraki

Institute of Informatics and Telecommunications
N.C.S.R 'Demokritos'
153 10 A. Paraskevi Attiki, Greece
e_mail: manolis@iit.nrcps.ariadne-t.gr

Abstract. An unfold/fold program transformation system which extends the unfold/fold transformations of H. Tamaki and T. Sato is presented in this paper. The system consists of unfolding, simultaneous folding, and generalization + equality introduction rules. The simultaneous folding rule permits the folding of a set of folded clauses into a single clause, using a set of folding clauses, while the generalization + equality introduction rule facilitates the application of the simultaneous folding rule by performing appropriate abstractions. A proof of the correctness of the proposed transformations in the sense of the least Herbrand model semantics of the program is also presented.

1 Introduction

Unfold/fold transformations were first proposed by R. Burstall and J. Darlington [BD77] in the context of a functional language. In the context of logic programming, H. Tamaki and T. Sato [TS84, TS86] formulated unfold/fold transformations for definite clause programs so as to preserve the equivalence of programs in the sense of the least Herbrand model semantics.

Many transformation systems for program optimization [Deb88, PP91], program specialization [BCD90], partial evaluation [LS91, PP93], and program synthesis [Sat90, ST84] are based on unfold/fold transformations.

An unfold/fold program transformation system which extends the unfold/fold transformations of Tamaki and Sato [TS84] is presented in this paper. The system consists of three transformation rules namely *unfolding*, *simultaneous folding*, and *generalization + equality introduction*. The main extension provided by our system is that the simultaneous folding rule permits the folding of a set of folded clauses into a single clause, using a set of folding clauses. The generalization + equality introduction rule widens the applicability of the folding rule. A proof of correctness of the transformations in the sense of the least Herbrand model semantics of the program is also presented.

Besides the uses of Tamaki and Sato's transformations in optimization of logic programs [Deb88, PP91] and in program synthesis [ST84], the transformations that we propose may also be proved useful in optimization of nondeterministic logic programs as shown in the examples that we present. The application of

* This paper appears in the Proceedings of the Sixth International Symposium "Programming Language Implementation and Logic Programming" (PLILP'94), Madrid, Spain Sept. 1994, Lecture Notes in Computer Science LNCS 844, Manuel Hermenegildo and Jaan Penjam (eds), pp 340-354, Springer-Verlag.

the transformations often reduces the total time needed to find all solutions to a query. This is because parts of the computations of the different solutions (or parts of the different paths through which a single solution is searched for) are merged in the transformed program.

The rest of this paper is organized as follows. After giving some preliminary definitions in section 2, the transformation rules are described in section 3. In section 4, the correctness of the transformations is proved. In section 5, the use of the generalization rule is shown. Finally, in sections 6 and 7, a conclusion is given and a comparison between our and related works is made.

2 Preliminaries

In the following, we assume familiarity with the basic terminologies of first order logic and logic programming [Llo87] such as term, atom, substitution, most general unifier, logical consequence, and least Herbrand model. In this section we give some definitions which are used in the following sections.

Definition 1 (*Definite Clause*). A *definite clause* is a formula of the form : $H \leftarrow B_1, ..., B_k.$ $(k \geq 0)$ where H, B_1,....,B_k are atoms and ',' stands for the logical conjunction operator.

Definition 2 (*Logic Program*). A *logic program* is a finite set of definite clauses.

Definition 3 (*Meaning of a Program*). Let P be a program. The *meaning* $M(P)$ of the program P is the set $M(P) = \{G : G$ is a ground atom which is a logical consequence of $P\}$.

Definition 4 (*Equivalent Programs*). Two programs P_1 and P_2 are said to be *equivalent* if $M(P_1) = M(P_2)$.

Definition 5 (*Internal Variable*). Let S and $S1$ be two sets of atoms such that $S1 \subset S$. We say that a variable X is an *internal variable* in $S1$ w.r.t. S if

(a) X occurs in every atom of $S1$ at least ones, and

(b) X occurs in no atom of the set $S - S1$.

Definition 6. Let C be a clause and p a predicate. We say that C *defines* p if p is the predicate of the head of C.

Definition 7 (*Instance*). An expression E' is an *instance* of an expression E, if there exists a substitution θ s.t. $E' = E\theta$.

Definition 8 (*Identical Clauses*). Two clauses are regarded as *identical* if one is obtained from the other through permutation of the body goals and renaming of variables.

In all clauses of this paper, variables are distinguished from constants by giving them uppercase names.

3 Unfold/fold transformations

The transformation rules are described in this section.

Definition 9 (*Initial Program*). An *initial program* P_0, is a definite clause program satisfying the following conditions :

(a) P_0 is divided into three disjoint sets of clauses, P_{new}, P_{old} and $P_{=}$ where $P_{=} = \{X = X\}$. The predicates defined in P_{new} are called *new predicates*, while the predicates defined in P_{old} are called *old predicates*.

(b) The new predicates appear neither in P_{old} nor in the bodies of the clauses[2] in P_{new}. The predicate '=' may appear in the bodies of the clauses of both P_{new} and P_{old} but not in their heads.

Example 1. Let $P_0 = \{C_1, C_2, C_3, C_4, C_5, C_6\} \cup S \cup P_{=}$ be an initial program, where

$/*$ *in_correct_position*(X, L) $\leftarrow$ *An element X is said to be in a correct position in the list L if X is in odd(even) position of the list and X is an odd(even) number.* $*/$

C_1 : $in\_correct\_position(X, L) \leftarrow inodd(X, L), odd(X).$
C_2 : $in\_correct\_position(X, L) \leftarrow ineven(X, L), even(X).$
C_3 : $inodd(X, [X|L]).$
C_4 : $inodd(X, [Y, Z|L]) \leftarrow inodd(X, L).$
C_5 : $ineven(X, [Y, X|L]).$
C_6 : $ineven(X, [Y, Z|L]) \leftarrow ineven(X, L).$

S contains the definitions of '*even*' and '*odd*', $P_{old} = \{C_3, C_4, C_5, C_6\} \cup S$, and $P_{new} = \{C_1, C_2\}$.

Definition 10 (*Generalization + Equality Introduction*). The application of the *generalization + equality introduction rule* to a clause C of the form

C : $H \leftarrow A_1, ..., A_n \quad (n \geq 0)$

consists in replacing C with a new clause C' of the form

C' : $GenH \leftarrow GenA_1, ..., GenA_n, X = t$

where $(GenH)\theta = H, (GenA_1)\theta = A_1, ..., (GenA_n)\theta = A_n$ and $\theta = \{X/t\}$.

Definition 11 (*Unfolding*). Let C be a clause in P_{l-1} of the form

C : $A \leftarrow A_1, ..., A_{i-1}, A_i, A_{i+1}, ..., A_n.$

and $C_1, C_2...., C_m$ be all clauses in P_{l-1}, whose heads are unifiable with A_i by most general unifiers (mgus) $\theta_1, \theta_2, ..., \theta_m$. The result of *unfolding C at* A_i is the set of clauses $\{C'_1, ..., C'_m\}$ such that, for each $j : (1 \leq j \leq m)$ if

C_j : $B_j \leftarrow B_{j_1}, ..., B_{j_h}. \quad (h \geq 0)$ and $B_j\theta_j = A_i\theta_j$, then
C'_j : $(A \leftarrow A_1, ..., A_{i-1}, B_{j_1}, ..., B_{j_h}, A_{i+1}, ..., A_n)\theta_j.$

Then, $P_l = (P_{l-1} - \{C\}) \cup \{C'_1, ..., C'_m\}$. C is called the *unfolded clause* and $C_1, ..., C_m$ are called the *unfolding clauses*. A_i is called the *unfolded atom*.

[2] We consider only non-recursive new definitions in this paper.

Example 2 (Continued from example 1). By unfolding C_1 at atom '$inodd(X, L)$' we obtain the program $P_1 = \{C_2, C_3, C_4, C_5, C_6, C_7, C_8\} \cup S \cup P_=$ where

C_7 : $in\_correct\_position(X, [X|L]) \leftarrow odd(X)$.

C_8 : $in\_correct\_position(X, [Y, Z|L]) \leftarrow inodd(X, L), odd(X)$.

By unfolding C_2 at atom '$ineven(X, L)$', we obtain the program $P_2 = \{C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}\} \cup S \cup P_=$ where

C_9 : $in\_correct\_position(X, [Y, X|L]) \leftarrow even(X)$.

C_{10} : $in\_correct\_position(X, [Y, Z|L]) \leftarrow ineven(X, L), even(X)$.

Definition 12 (*Primitive Folding*). Let C be a clause in P_{l-1} of the form :

C : $A_0 \leftarrow A_1, ..., A_n.$ $(n > 0)$

and D be a clause in P_{new}. Let D' be either D or the result of applying one or more times the generalization + equality introduction rule to D. Let D' be of the form :

D' : $B_0 \leftarrow B_1, ..., B_k.$ $(k > 0)$

Suppose that there exists a substitution θ satisfying the following conditions :

(a) $B_1\theta = A_{j_1}, B_2\theta = A_{j_2}, ..., B_k\theta = A_{j_k}$ where $j_1, j_2, ..., j_k$ are all different natural numbers between 1 and n.

(b) For each variable internal in the body of D', θ substitutes a distinct variable not appearing in $\{B_0\theta, A_0, A_1, ..., A_n\} - \{A_{j_1}, ..., A_{j_k}\}$.

(c) Either the predicate of C's head is an old predicate, or C is unfolded at least once in the sequence $P_0, P_1, ..., P_{l-1}$ at a predicate other than '='.

Then, *the result of the primitive folding of* C *using* D is the clause C' with head A_0 and body $\{B_0\theta\} \cup \{A_1, A_2, ..., A_n\} - \{A_{j1}, ..., A_{jk}\}$. C is called a *folded clause*, D is called a *folding clause* and $B_0\theta$ the *atom introduced by folding*.

Definition 13 (*Simultaneous Folding*). Let P_{l-1} a program, $S_{fd} = \{C_1, ..., C_m\}$ a set of m clauses $(m > 0)$ in P_{l-1} all defining a predicate p, and $S_{fg} = \{D_1, ..., D_m\}$ be a set of m clauses in P_{new} all defining a predicate q. Let f be a bijection (called a *folding bijection*) between S_{fd} and S_{fg} such that for each pair $(C_i, D_i) \in f$ the following hold:

(a) C_i can be folded (primitive folding) using D_i.

(b) Let S_P be the set of the primitive folding results. Then, S_P is a singleton (i.e. the primitive folding results are identical clauses).

(c) Let C_f be the clause in S_P. Then, A_f, the atom introduced by folding, is common to all primitive folding results, and no clause in $P_{new} - S_{fg}$ has head unifiable with A_f (we say that the folding is *set reversible*).

Then, the result of *simultaneously folding the set* S_{fd} *using* S_{fg} consists in replacing the set of clauses S_{fd} with the clause C_f, taking $P_l = (P_{l-1} - S_{fd}) \cup \{C_f\}$. S_{fd} is called the *folded set* while S_{fg} is called the *folding set*.

Example 3 (Continued from example 2). Simultaneously folding the set $S_{fd} = \{C_8, C_{10}\}$ using the set $S_{fg} = \{C_1, C_2\}$ we obtain the program $P_3 = \{C_3, C_4, C_5, C_6, C_7, C_9, C_{11}\} \cup S \cup P_=$ where :

C_{11} : $in\_correct\_position(X, [Y, Z|L]) \leftarrow in\_correct\_position(X, L)$.

We note here that this folding is not permitted in Tamaki-Sato's system [TS84, TS86]. Even the folding of C_8 with C_1 (or C_{10} with C_2) is not permitted since it violates their *reversibility condition* [TS86] (more than one clause in P_{new} can be used to unfold the atom introduced by folding). Tamaki & Sato's folding rule is obtained from our simultaneous folding by restricting the folding set and the folded set to singletons.

Definition14 (*Transformation Sequence***).** Let P_0 be an initial program and P_l $(l > 0)$ be a program obtained from P_{l-1} by applying unfolding, simultaneous folding or generalization + equality introduction rule. Then, the sequence of programs $P_0, P_1, ..., P_l$ is called a *transformation sequence starting from* P_0.

Example 4 (Continued from example 3). The sequence of programs P_0, P_1, P_2, P_3 in examples 1, 2 and 3 is a transformation sequence starting from P_0.

4 Correctness proof

The correctness of the transformation rules is proved in this section along the same line as in [TS84, TS86, KK90]. For this, we first introduce the notions of partial and total correctness.

Definition15 (*Partially Correct Transformation***).** Let P_0 be the initial program in a transformation sequence, and P_i a program obtained from P_0 by applying the transformation rules. The transformation is said to be *partially correct* when $M(P_i) \subseteq M(P_0)$.

Definition16 (*Totally Correct Transformation***).** Let P_0 be the initial program in a transformation sequence, and P_i a program obtained from P_0 by applying the transformation rules. The transformation is said to be *totally correct* when $M(P_i) = M(P_0)$.

In the proof of correctness that follows we are based on the notion of the *proof trees* [TS84] which characterize the least Herbrand model semantics.

Definition17 (*Proof Tree***).** Let P be a program and A a ground atom. A tree T_A whose nodes are labelled with ground atoms is called a *proof tree* (or simply *proof*) of A by P if the following conditions hold :

(a) A is the root label of T_A .

(b) Let $T_{A_1}, ..., T_{A_n}$ $(n \geq 0)$ be the immediate subtrees[3] of T_A, and $A_1, ..., A_n$ their root labels. Then, $A \leftarrow A_1, ..., A_n$ is a ground instance of a clause C in P.

(c) Each immediate subtree T_{A_i} of T_A is a proof tree of A_i by P.

C is called *the clause used at the root of* T_A. $T_{A_1}, ..., T_{A_n}$ are called the *immediate subproofs* of T_A. The number of nodes of T_A is called the *size* of T_A.

It is well known that for every ground atom A it holds that, $A \in M(P)$ if and only if there exists a proof tree of A by P.

[3] Notation : In all proofs of this paper an expression of the form T_A stands for 'a proof tree T of the ground atom A'.

4.1 Partial Correctness

In this section we prove partial correctness of the transformations.

Lemma 1. *Let P_i be a program, and C a clause in P_i. Let C' be a clause obtained from C either (a) by applying the generalization + equality introduction rule to C, or (b) by unfolding C at a body goal having '=' as predicate.*

Let $P_i' = (P_i - \{C\}) \cup \{C'\}$. Then, (i) $M(P_i) = M(P_i')$ and (ii) for every ground atom A : there exists a proof T in P_i with C used at its root iff there exists a proof T' of A in P_i' with C' used at its root.

Proof. (i) An isomorphism from the proof trees by P_i to the proof trees by P_i' is easily defined. In case of (a) we have : Let C be of the form

$C: \quad A_0 \leftarrow A_1, A_2, ..., A_n \quad (n > 0)$

Then, C' will be

$C': \quad A_0' \leftarrow A_1', A_2', ..., A_n', X = t$

such that $A_i'\theta = A_i$ for all $i : 0 \leq i \leq n$ where $\theta = \{X/t\}$. A proof tree T by P_i corresponds to itself in P_i' if no clause at the root of T or at the root of any subtree of T is an instance of C. Otherwise, T corresponds to a tree T' by P_i' such that for every subtree S of T with an instance $C\sigma$ of C used at its root, the corresponding subtree S' of T' has $C'\sigma$ at its root. It is easy to see that S' is taken from S by adding a node $(X = t)\sigma$ as a son to the root node of S.

In case of (b), T' is obtained by deleting the nodes of T of the form $(X = t)\sigma$ which correspond to the unfolded atoms.

(ii). Part ii is an immediate conclusion of the proof of part i.

Lemma 2. (Partial Correctness). *Let P_0,P_1,....,P_n be a transformation sequence. If $M(P_i) = M(P_0)$, then $M(P_i) \supseteq M(P_{i+1})$, for $i = 0, 1, ..., n-1$.*

Proof. Let A be a ground atom in $M(P_{i+1})$, T be a proof tree of A by P_{i+1} and C the clause, an instance of which, is used at the root of T. We construct a proof tree T' of A by P_i by induction on the structure of T, as follows :

case 1 : C is in P_i. Let C be of the form

$C: \quad A_0 \leftarrow A_1, A_2, ..., A_n \quad (n > 0)$

and $C\sigma$ the instance of C used at the root of T. Then $A = A_0\sigma$. Let $T_{A_1\sigma}, T_{A_2\sigma}$, ..., $T_{A_n\sigma}$ be T's immediate subproofs. By the induction hypothesis, there exist proof trees $T'_{A_1\sigma}, T'_{A_2\sigma}, ..., T'_{A_n\sigma}$ by P_i. A proof tree T' of A by P_i is obtained by putting a root node labeled with A over $T'_{A_1\sigma}, T'_{A_2\sigma}, ..., T'_{A_n\sigma}$.

case 2 : C is the result of generalizing a clause C' in P_i. From lemma $1(i)$, $M(P_i) = M(P_{i+1})$.

case 3 : C is the result of unfolding a clause C' in P_i. Let C' be of the form :

$C': \quad A_0 \leftarrow A_1, A_2, ..., A_n \quad (n > 0)$

and D be one of the unfolding clauses in P_i such that C is the result of unfolding C' using D. Let D be of the form

$D: \quad B_0 \leftarrow B_1, B_2, ..., B_m \quad (m \geq 0)$

Without loss of generality, we assume that D is applied to A_1. Then, A_1 and B_0 are unifiable, say by an mgu θ. Therefore $A_1\theta = B_0\theta$, and C is of the form :

C : $A_0\theta \leftarrow B_1\theta, ..., B_m\theta, A_2\theta, ..., A_n\theta$

Let $C\sigma$ be the instance of C used at the root of T, and $T_{B_1\theta\sigma}, ..., T_{B_m\theta\sigma}$, $T_{A_2\theta\sigma}, ..., T_{A_n\theta\sigma}$ be T's immediate subproofs by P_{i+1}. By the induction hypothesis, there are proof trees $T'_{B_1\theta\sigma}, ..., T'_{B_m\theta\sigma}, T'_{A_2\theta\sigma}, ..., T'_{A_n\theta\sigma}$ by P_i. Let $T'_{A_1\theta\sigma}$ be a proof tree obtained by putting a root node labelled with $A_1\theta\sigma$ over $T'_{B_1\theta\sigma}, ..., T'_{B_m\theta\sigma}$. By putting a new root labelled with A over $T'_{A_1\theta\sigma}, T'_{A_2\theta\sigma}, ...$, $T'_{A_n\theta\sigma}$ we obtain a proof T' of A by P_i.

case 4 : C is the result of simultaneously folding a set of clauses $S_{fd} = \{C'_1$, $...., C'_k\}$ in P_i using a set $S_{fg} = \{D_1, ..., D_k\}$ in P_{new}. Let C be of the form :

C : $A_0 \leftarrow A_1, ..., A_r, (B\theta)$

where $B\theta$ is the atom introduced by folding. Let $C\sigma$ be the instance of C at the root of T and $T_{A_1\sigma}, ..., T_{A_r\sigma}, T_{B\theta\sigma}$ be T's immediate subproofs by P_{i+1}. By the induction hypothesis, there exist proof trees $T'_{A_1\sigma}, ..., T'_{A_r\sigma}, T'_{B\theta\sigma}$ by P_i. From $M(P_i) = M(P_0)$, it follows that there exists a proof tree $T^0_{B\theta\sigma}$ of $B\theta\sigma$ by P_0. Since B has a new predicate, the clause used at the root of $T^0_{B\theta\sigma}$ is in P_{new}, and must be one of D_i $(1 \leq i \leq k)$ (see set reversibility condition, definition 13(c)). Let D'_i be the generalization of D_i used in folding. Let D'_i be of the form

D'_i : $B \leftarrow B_1, ..., B_m$ $(m > 0)$

From lemma $1(ii)$, there exists a proof of $B\theta\sigma$ by $(P_0 - \{D_i\}) \cup \{D'_i\}$ with D'_i used at its root, and from lemma $1(i)$, there exist proofs of $B_1\theta\sigma, ..., B_m\theta\sigma$ by P_0. Since $M(P_0) = M(P_i)$, there are also proofs $T'_{B_1\theta\sigma}, ..., T'_{B_m\theta\sigma}$ by P_i. From the definition of primitive folding, we have that $B_1\theta = A_{r+1}, ..., B_m\theta = A_{r+m}$. A proof T' of A by P_i is obtained by putting a root label A over $T'_{A_1\sigma}, ..., T'_{A_r\sigma}$, $T'_{B_1\theta\sigma}, ..., T'_{B_m\theta\sigma}$. The clause C'_j at the root of T', which has as instance

$C'_j\sigma$: $A_0\sigma \leftarrow A_1\sigma, ..., A_r\sigma, A_{r+1}\sigma, ..., A_{r+m}\sigma$

is in $S_{fd} \subseteq P_i$ (the clause corresponding to D_i through the folding bijection f).

4.2 Total Correctness

For the proof of total correctness we need some more definitions.

Definition 18 (*Weight of a Proof Tree*). Let P_0 be the initial program in a transformation sequence, A an atom in $M(P_0)$ and T a proof tree of A by P_0. Let s be the size of T, n the number of nodes (leafs) of T which have the predicate '=', and $sn = s - n$ (we call sn, the *net size* of T). Then, the *weight of the proof tree* T, denoted by $w_t(T)$, is defined as follows : $w_t(T) = sn - 1$ if A is defined in P_{new}, otherwise $w_t(T) = sn$ if A is defined in P_{old} or in $P_=$.

Definition 19 (*Weight of an Atom*). Let P_0 be the initial program in a transformation sequence, and A a ground atom in $M(P_0)$. The *weight of the atom* A, denoted by $w(A)$ is the minimum of the weights of the proof trees of A by P_0.

Lemma 3. *Let A be a ground atom in $M(P_0)$ and T a proof of A by P_0. If the predicate of A is '=' then (a) $w_t(T) = 0$, and (b) $w(A) = 0$.*

Proof. Obvious.

Definition 20 (*Weight Consistent Proof*). Let P_i be a program in a transformation sequence starting from P_0, A a ground atom in $M(P_i)$ and T a proof of A by P_i. Let C be the clause used at the root of T and $T_{A_1}, ..., T_{A_n}$ be T's immediate subproofs with root labels $A_1, ..., A_n$. T is said to be *weight consistent* if either C is a unit clause or

(1) $w(A) \geq w(A_1) + ... + w(A_n)$

(2) If the equality holds in (1), then folding condition (c) of definition 12 does not hold for C.

(3) $T_{A_1}, ..., T_{A_n}$ are also weight consistent.

Definition 21 (*Weight Consistent Program*). Let P_i be a program in a transformation sequence starting from P_0. We say that P_i is a *weight consistent program* if, every ground atom A in $M(P_i)$ has a weight consistent proof by P_i.

For the needs of the total correctness proof we define a binary relation over ground atoms as follows.

Definition 22. Let A and B be two ground atoms in $M(P_0)$. We say that $A >> B$ iff either (a) $w(A) > w(B)$, or (b) $w(A) = w(B)$ and A has a new predicate and B has an old predicate or '='.

Lemma 4. *$>>$ is a well founded relation over $M(P_0)$.*

Proof. Obvious.

We will use induction on the well founded relation '$>>$' in order to prove lemma 8 that follows.

Lemma 5. *Let P_i be a program, C a clause in P_i and C' a clause obtained from C by applying the generalization + equality introduction rule or unfolding C at a goal having '=' as predicate. Let $P_i' = (P_i - \{C\}) \cup \{C'\}$. Let $A \in M(P_i)$, then : A has a weight consistent proof by P_i iff A has a weight consistent proof by P_i'.*

Proof. An isomorphism is defined on the weight consistent proof trees by P_i and P_i' in the same way as in the proof of lemma 1. Addition or deletion of ground instances of '=' goals do not affect the relation between the weights of the nodes, and the weight consistency of the tree (see lemma 3).

Lemma 6. *For each ground atom $A \in M(P_0)$ there exits a weight consistent proof of A by P_0 (i.e. P_0 is weight consistent).*

Proof. It is easy to prove [GK94] by induction on the structure of the proof trees that for every ground atom $A \in M(P_0)$ the smallest proof of A by P_0 is weight consistent.

Lemma 7. *Let P_0 be an initial program in a transformation sequence and A a ground atom in $M(P_0)$. If either $w(A) = 1$ and A has an old predicate or $w(A) = 0$ and A has a new predicate or '=', then A has a weight consistent proof in every program in the transformation sequence.*

Proof. Base case : A has a weight consistent proof by P_0 (see lemma 6).

Induction hypothesis : A has a weight consistent proof T *by* P_i. We will prove that A also has a weight consistent proof by P_{i+1}.

Let C_0 be the clause used at the root of T of the form

$C_0 : \quad A_0 \leftarrow A_1, ..., A_n \quad (n \geq 0)$

and $T_{A_1}, ..., T_{A_n}$ be T's immediate subpoofs.

case 1 : A has an old predicate and $w(A) = 1$. Then C_0 is foldable, $1 > w(A_1) + ... + w_(A_n)$ and thus, $w(A_i) = 0$ for $i = 1, ..., n$. Then, each clause C_i, $(1 \leq i \leq n)$ at the roots of $T_{A_1}, ..., T_{A_n}$ is either a unit clause with a predicate in $P_{new} \cup P_{=}$, or its head has a new predicate and its body predicates are '='. In the later case, C_i must be not foldable (as T_{A_i} is weight consistent). We can see four cases :

case 1.1 : All clauses of T *are in* P_{i+1}. Then T is also a weight consistent proof by P_{i+1}.

case 1.2 : A clause C_i *used at the root of a subtree of* T *has been generalized or a body goal in* C_i *with predicate '=' has been unfolded giving a clause* C_i' *in* P_{i+1}. Lemma 5 ensures the existence of a weight consistent proof T' by P_{i+1}.

case 1.3 : A body goal A_i *of* C_0 *with a new predicate is unfolded.* We get a weight consistent proof of A by P_{i+1} by removing A_i from T and linking A to all childs of A_i (which are instances of '$X = X$').

case 1.4 : C_0 *is folded at some equalities giving* C_0' *in* P_{i+1}. Let D the clause in P_{new} used in the primitive folding of C_0, then, all predicates in D's body are '=', and thus only the generalization or unfolding of some equalities is applicable to D (or to a clause obtained from D). A clause D' obtained in this way is also in P_{i+1}. We get easily a weight consistent proof T' of A by P_{i+1} with C_0' at its root and D' at the root of an immediate subtree of T' (lemma 1).

case 2 : The predicate of A is new or '=', and $w(A) = 0$. Then, $0 \geq w(A_1) + ... + w(A_n)$, C is not foldable and $w(A_i) = 0$ for $i = 1, ..., n$. Thus, each A_i has '=' as predicate. The only transformations that we can apply to C is either to generalize it or unfold an equality predicate taking a clause C' in P_{i+1}, and thus lemma 5 ensures the existence of a weight consistent proof by P_{i+1}.

Lemma 8. *If* $M(P_0) = M(P_i)$ *and* P_i *is weight consistent then for each* $A \in M(P_i)$ *there is a weight consistent proof of* A *by* P_{i+1}.

Proof. Let A be a ground atom in $M(P_i)$ and T a weight consistent proof of A by P_i. We prove by induction on '$>>$' that there is also a proof T' of A by P_{i+1}.

Base cases : The lemma holds for the base cases (see lemma 7).

Induction hypothesis : *We suppose that for all atoms* $B \in M(P_i)$ *for which* $w(A) >> w(B)$ *there exists a weight consistent proof by* P_{i+1}.

Let C be a clause in P_i of the form

$C : \quad A_0 \leftarrow A_1, ..., A_n \quad (n \geq 0)$

and $C\sigma$ the instance of C at the root of T. Since P_i is weight consistent, then $w(A) \geq w(A_1\sigma) + ... + w(A_n\sigma)$. If the equality holds then, folding condition of definition 12(c) does not hold, A has a new predicate and all body atoms A_i have

old predicates or '='. In all cases, $A >> A_i\sigma$ for $i = 1, ..., n$. By the induction hypothesis, there are proofs $T'_{A_1\sigma}, ..., T'_{A_n\sigma}$ by P_{i+1}.

case 1 : C *is in* P_{i+1}. In this case, a (weight consistent) proof T' of A by P_{i+1} is constructed by putting a root label A over $T'_{A_1\sigma}, ..., T'_{A_n\sigma}$.

case 2 : C *is unfolded.* Suppose, without loss of generality, that C is unfolded at A_1 giving $C_1, ..., C_k$ $(k > 0)$ in P_{i+1}. Let $T_{B_1}, ..., T_{B_m}$ be the immediate subproofs of $T_{A_1\sigma}$ by P_i. By the induction hypothesis, there are also proofs $T'_{B_1}, ..., T'_{B_m}$ by P_{i+1}. A proof T' of A by P_{i+1} is constructed by putting a root label A over $T'_{B_1}, ..., T'_{B_m}, T'_{A_2\sigma}, ..., T'_{A_n\sigma}$. The clause at the root of T' is an instance of a clause C_j $(1 \leq j \leq k)$. From the weight consistency of T, we have

(1) $w(A) \geq w(A_1\sigma) + w(A_2\sigma) + ... + w(A_n\sigma)$ and

(2) $w(A_1\sigma) \geq w(B_1) + ... + w(B_m)$

Combining (1) and (2) we take

(3) $w(A) \geq w(B_1) + ... + w(B_m) + w(A_2\sigma) + ... + w(A_n\sigma)$

Equality in (3) holds only if it holds, in both (1) and (2). Then, folding condition of definition 12(c) must not hold neither for C nor for C_j. Therefore, A and A_1 must both have new predicates. Thus, folding condition holds in (1). This contradiction, proves that equality does not hold in (3), and T' is weight consistent.

case 3 : C *is folded.* Let C be of the form

C : $A_0 \leftarrow A_1, ..., A_r, A_{r+1}, ..., A_{r+m}$

and $C\sigma$ the instance of C at the root of T. Let $S_{fg} = \{D_1, ..., D_k\}$ be the folding set $(S_{fg} \subseteq P_{new})$, and C' the result of folding. Let, without loss of generality, $A_{r+1}, ..., A_{r+m}$ be the folded atoms. Then, C' must be of the form

C' : $A_0 \leftarrow A_1, ..., A_r, B\theta$

where $B\theta$ is the atom introduced by folding. Since folding condition holds for C, $w(A) > w(A_i\sigma)$ for $i = 1, ..., r + m$. By the induction hypothesis, there are proof trees $T'_{A_1\sigma}, ..., T'_{A_r\sigma}, T'_{A_{r+1}\sigma}, ..., T'_{A_{r+m}\sigma}$ by P_{i+1}. Let $D_j \in S_{fg}$ be the clause used in the primitive folding of C. Then, $A_{r+1}, ..., A_{r+m}$ is an instance of the body of D_j or an instance of the body of a generalization D'_j of D_j. Since $A_{r+1}\sigma, ..., A_{r+m}\sigma$ have proofs by P_i, they also have proofs by P_0 and therefore (see also lemma 1), $B\theta\sigma$ has a proof by P_0. From $M(P_0) = M(P_i)$, it follows that $B\theta\sigma$ has a proof by P_i. Since B has a new predicate and by the definition of weight consistency, we have that

(4) $w(B\theta\sigma) \leq w(A_{r+1}\sigma) + ... + w(A_{r+m}\sigma)$

In addition, since C is foldable we have that

(5) $w(A) > w(A_1\sigma) + ... + w(A_r\sigma) + w(A_{r+1}\sigma) + ... + w(A_{r+m}\sigma)$

Thus, $w(A) > w(B)$. By the induction hypothesis, $B\theta\sigma$ has a weight consistent proof $T'_{B\theta\sigma}$ by P_{i+1}. By putting a node A over $T'_{B\theta\sigma}, T'_{A_1\sigma}, ..., T'_{A_r\sigma}$, we take a proof T' for A by P_{i+1}. From (4) and (5) we see that T' is weight consistent.

case 4 : C *is generalized.* Lemma 5 ensures the existence of a weight consistent proof T' of A by P_{i+1}.

Lemma 9. *If* $M(P_0) = M(P_i)$ *and* P_i *is weight consistent then* $M(P_{i+1}) = M(P_i)$ *and* P_{i+1} *is also weight consistent.*

Proof. It immediatelly follows form lemmas 2 and 8.

Theorem 1. (Total Correctness). *Let P_0 be an initial program in a transformation sequence and P_i $(i > 0)$ a program obtained from P_0 by applying a sequence of transformation steps. Then, the least Herbrand model of P_i is identical to the least Herbrand model of P_0.*

Proof. It comes easily from lemmas 6 and 9.

5 The Generalization Rule

In this section we present a more complex example in which the use of generalization + equality introduction rule is shown.

Example 5. Let $P_{old} = \{C_1, C_2, C_3, C_4, C_5\}$ be the following program :

$/* \ one\_two\_sublist(S, L) \leftarrow S$ *is a sublist of the list* L *containing one or two elements.* $*/$

C_1 : $one\_two\_sublist([X], L) \leftarrow sublist([X], L)$.
C_2 : $one\_two\_sublist([X, Y], L) \leftarrow sublist([X, Y], L)$.
C_3 : $sublist([\,], L)$.
C_4 : $sublist([X|Xs], [X|L]) \leftarrow sublist(Xs, L)$.
C_5 : $sublist([X|Xs], [Z|L]) \leftarrow sublist([X|Xs], L)$.

P_{new} will be constructed during the transformation process that follows.
By unfolding C_1 at '$sublist([X], L)$' and C_2 at '$sublist([X, Y], L)$' we take :

C_6 : $one\_two\_sublist([X], [X|L]) \leftarrow sublist([\,], L)$.
C_7 : $one\_two\_sublist([X], [X1|L]) \leftarrow sublist([X], L)$.
C_8 : $one\_two\_sublist([X, Y], [X|L]) \leftarrow sublist([Y], L)$.
C_9 : $one\_two\_sublist([X, Y], [X1|L]) \leftarrow sublist([X, Y], L)$.

Unfolding C_6 at '$sublist([\,], L)$' we take

C_{10} : $one\_two\_sublist([X], [X|L])$.

Applying several times the generalization + equality rule to clauses $\{C_7, C_8, C_9\}$ we take [4].

C_{7G} : $one\_two\_sublist([X|W], [X1|L]) \leftarrow W = [\,], Z = X, Q = [\,], X1 = X1, sublist([Z|Q], L)$.
C_{8G} : $one\_two\_sublist([X|W], [X1|L]) \leftarrow W = [Y], Z = Y, Q = [\,], X1 = X, sublist([Z|Q], L)$.
C_{9G} : $one\_two\_sublist([X|W], [X1|L]) \leftarrow W = [Y], Z = X, Q = [Y], X1 = X1, sublist([Z|Q], L)$.

We now introduce the following nondeterministic definition for the new (Eureka) predicate 'new'.

D_1 : $new(W, X, X1, L) \leftarrow W = [\,], Z = X, Q = [\,], X1 = X1, sublist([Z|Q], L)$.

[4] We find the *least general generalization* of them and put the substitutions as calls to '='.

D_2 : $new(W, X, X1, L) \leftarrow W = [Y], Z = Y, Q = [\,],$
$X1 = X, sublist([Z|Q], L).$

D_3 : $new(W, X, X1, L) \leftarrow W = [Y], Z = X, Q = [Y],$
$X1 = X1, sublist([Z|Q], L).$

Clauses D_1, D_2, and D_3 are added to P_{new} [5]. We now fold simultaneously the clauses $S_{fd} = \{C_{7G}, C_{8G}, C_{9G}\}$ using the set $S_{fg} = \{D_1, D_2, D_3\}$ we take

C_{11} : $one\_two\_sublist([X|W], [X1|L]) \leftarrow new(W, X, X1, L).$

We now try to derive a recursive definition for the predicate 'new'. For this, we unfold clauses $\{D_1, D_2, D_3\}$ at the call to '$sublist$'. We take

C_{12} : $new(W, X, X1, [Z|L]) \leftarrow W = [\,], Z = X, Q = [\,],$
$X1 = X1, sublist(Q, L).$

C_{13} : $new(W, X, X1, [F|L]) \leftarrow W = [\,], Z = X, Q = [\,],$
$X1 = X1, sublist([Z|Q], L).$

C_{14} : $new(W, X, X1, [Z|L]) \leftarrow W = [Y], Z = Y, Q = [\,],$
$X1 = X, sublist(Q, L).$

C_{15} : $new(W, X, X1, [F|L]) \leftarrow W = [Y], Z = Y, Q = [\,],$
$X1 = X, sublist([Z|Q], L).$

C_{16} : $new(W, X, X1, [Z|L]) \leftarrow W = [Y], Z = X, Q = [Y],$
$X1 = X1, sublist(Q, L).$

C_{17} : $new(W, X, X1, [F|L]) \leftarrow W = [Y], Z = X, Q = [Y],$
$X1 = X1, sublist([Z|Q], L).$

By folding simultaneously $\{C_{13}, C_{15}, C_{17}\}$ with $\{D_1, D_2, D_3\}$ we take

C_{18} : $new(W, X, X1, [F|L]) \leftarrow new(W, X, X1, L).$

Simplifying C_{12}, C_{14} and C_{16} we take

$C_{12'}$: $new([\,], X, X1, [X|L]).$

$C_{14'}$: $new([Y], X, X, [Y|L]).$

$C_{16'}$: $new([Y], X, X1, [X|L]) \leftarrow sublist([Y], L).$

Therefore, $P_0 = \{C_1, C_2, C_3, C_4, C_5, D_1, D_2, D_3\} \cup P_{=}$ while the final program of the transformation sequence is $P_{final} = \{C_3, C_4, C_5, C_{10}, C_{11}, C_{12'}, C_{14'}, C_{16'}, C_{18}\} \cup P_{=}$. Collecting together the clauses of the final program we take [6] :

C_{10} : $one\_two\_sublist([X], [X|L]).$

C_{11} : $one\_two\_sublist([X|W], [X1|L]) \leftarrow new(W, X, X1, L).$

$C_{12'}$: $new([\,], X, X1, [X|L]).$

$C_{14'}$: $new([Y], X, X, [Y|L]).$

$C_{16'}$: $new([Y], X, X1, [X|L]) \leftarrow sublist([Y], L).$

C_{18} : $new(W, X, X1, [F|L]) \leftarrow new(W, X, X1, L).$

C_3 : $sublist([\,], L).$

C_4 : $sublist([X|Xs], [X|L]) \leftarrow sublist(Xs, L).$

C_5 : $sublist([X|Xs], [Z|L]) \leftarrow sublist([X|Xs], L).$

[5] The reader may notice that it is not necessary to define P_{new} at the begining of the transformation process. In practice, P_{new} is formed during the transformation process when new Eureka definitions are invented.

[6] This program could be further simplified by defining a new predicate by the clause $new1(Y, L) \leftarrow sublist([Y], L)$, use it to fold $C_{16'}$ and then finding a recursive definition for the predicate '$new1$' by unfolding and then folding.

6 Discussion

The motivation for this work is to find methods for removing the redundancy often presented in the execution of nondeterministic logic programs when 'similar' work is done in different derivation paths. A first step in this direction is to define correct transformation rules suitable for these optimizations.

It is widely known, that the use of Tamaki and Sato's [TS84] unfold/fold rules often results in significant improvements of logic programs. The source of these improvements is usually the shortening of the derivation paths due to the introduction of recursion through the use of the folding rule as well as to the resolution steps performed in compile time by unfolding. On the other hand, Tamaki and Sato's rules do not offer so much in limitation of 'similar' work done in different derivation paths. To our opinion, this is due to the fact that their folding rule is not the inverse of unfolding, since unfolding consists in replacing a single clause by a set of clauses (in general, more than one clauses), while in folding we always get a single clause from a single clause. This assymetry is, in a way, reinstated by our simultaneous folding rule. For this, we retain the unfolding rule and reformulate (generalize) the folding so as to permit the folding of a set of clauses with another set of clauses giving a single clause. The effect of the application of our simultaneous folding rule is shown in figure 1.

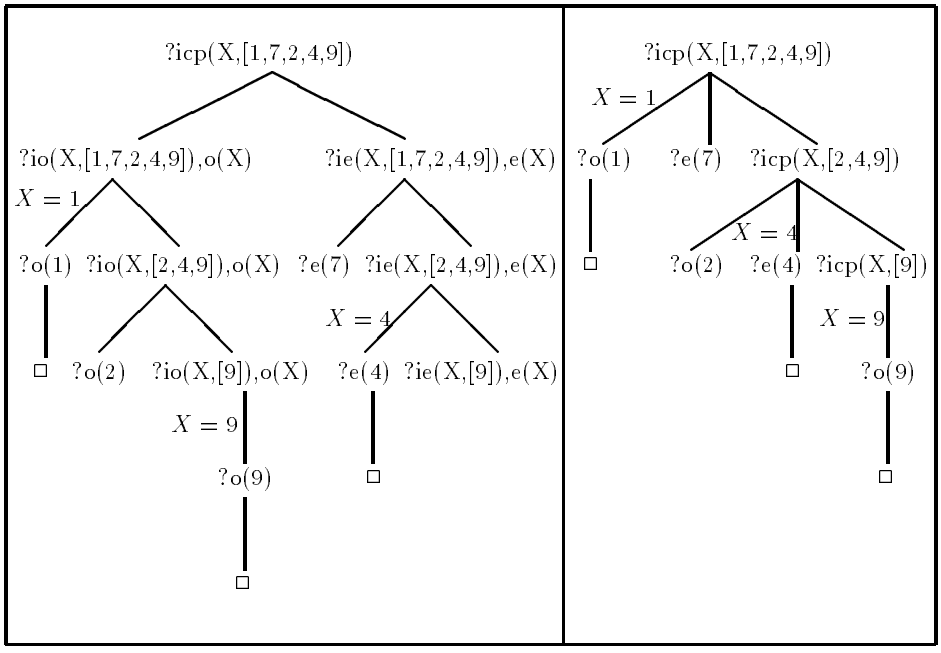

Fig. 1. SLD trees for the query '?$in\_correct\_position(X, [1, 7, 2, 4, 9])$' (we write '$icp$') with P_0 (see left tree) and the final program (right tree) of the example of section 3. (We write 'ie' for 'in_even', 'io' for 'in_odd', 'e' for '$even$' and 'o' for 'odd')

In this figure, the SLD-trees for a query to the initial and final programs of the examples of section 3, are shown. In the left tree, the derivation is split into two independent subtrees, one for the odd and the other for the even numbers. In the right tree, these subtrees have been merged. It is clear that applying simultaneous folding, different derivation paths with 'enough similarity' are merged, reducing in this way the total time and space needed for the production of all the solutions given by the procedure. Computer experiments demonstrate significant improvements in efficiency as a result of the application of the transformations. Even when a single solution is required, we may also have gains in efficiency since parts of the brances of the search tree traversed for finding the solution may have been merged by the application of the transformations.

Tamaki & Sato's folding rule is a special case of ours, since we obtained it by restricting the folding set and the folded set to singletons.

The role of the generalization + equality introduction rule is to perform the appropriate abstractions to clauses and reveal the similarity of them so as to permit the application of the simultaneous folding rule (possibly after the introduction of appropriate new definitions).

Our generalization + equality introduction rule is similar to the one proposed by M. Proietti and A. Pettorossi [PP90, PP93]. A difference is that we use this rule to generalize both head and/or body atoms of a clause while Proietti and Pettorossi use it to generalize only body atoms. The definition of the initial program is a slightly different of the one in Tamaki and Sato's work due to the inclusion of $P_{=}$.

As in Tamaki and Sato's program transformation system, our system becomes more powerful when combined with goal replacement rules as well as clause replacement rules. The incorporation of these rules as well as their applicability conditions seem to be similar to that presented in [TS84].

7 Conclusions

An unfold/fold program transformation system for definite clause programs, which generalize the rules proposed by Tamaki & Sato [TS84], is presented in this paper. The system consists of three rules namely unfolding, simultaneous folding and generalization + equality introduction. The simultaneous folding rule permits the folding of a set of folded clauses into a single clause, using a set of folding clauses, while the generalization + equality introduction rule facilitates the application of the simultaneous folding rule by performing appropriate abstractions. We prove the correctness of our transformations in the sense of the least Herbrand model semantics.

The strategies for invention of new definitions and the application of the transformation rules, the extension of the transformation rules to permit recursive new definitions as well as the work on proving other useful properties of our transformations are between our plans for future work.

Acknowledgement

I would like to thank the anonymous referees for their helpful comments on this paper.

References

[BCD90] A. Bossi, N. Cocco, and S. Dulli. A method for specializing logic programs. *ACM Transactions on Programming Languages and Systems*, 12(2):253–302, 1990.

[BD77] R. Burstall and J. Darlington. A transformation system for developing recursive programs. *J.ACM*, 24(1):44–67, Jan. 1977.

[Deb88] S. K. Debray. Unfold/fold transformations and loop optimization of logic programs. In *SIGPLAN' 88 Conference on Programming Language Design and Implementation*, pages 297–307, June 1988.

[GK94] M. Gergatsoulis and M. Katzouraki. Unfold/fold transformations for definite clause programs. Technical report 250/3/42, Institute of Informatics & Telecom. NCSR 'Demokritos', Febr. 1994.

[KK90] T. Kawamura and T. Kanamori. Preservation of stronger equivalence in unfold/fold logic program transformations. *Theoretical Computer Science*, 75:139–156, 1990.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[LS91] J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *J. Logic Programming*, 11(3 & 4):217–242, Oct./Nov. 1991.

[PP90] M. Proietti and A. Pettorossi. Synthesis of eureka predicates for developing logic programs. In *LNCS no. 432, Proc. of the 3rd European Symposium on Programming*, pages 306–325. Springer-Verlag, 1990.

[PP91] M. Proietti and A. Pettorossi. Unfolding-definition-folding, in this order for avoiding unnecessary variables in logic programs. In J. Maluszinski and M. Wirsing, editors, *LNCS no. 528, Proc. PLILP' 91*, pages 347–358. Springer-Verlag, 1991.

[PP93] M. Proietti and A. Pettorossi. The loop absorption and the generalization strategies for the development of logic programs and partial deduction. *The Journal of Logic Programming*, 16(1 & 2):123–162, May 1993.

[Sat90] T. Sato. An equivalence preserving first order unfold/fold transformation system. In *Lecture Notes in Computer Science (LNCS)*, 463, pages 175–188. Springer Verlag, 1990.

[ST84] T. Sato and H. Tamaki. Transformational logic program synthesis. In *International Conference of Fifth Generation Computer Systems*, pages 195–201, 1984.

[TS84] H. Tamaki and T. Sato. Unfold/fold transformations of logic programs. In *Second International Conference on Logic Programming*, pages 127–138, 1984.

[TS86] H. Tamaki and T. Sato. A generalized correctness proof of the unfold/fold logic program transformation. Technical Report No 86-4, Dept. of Information Science Ibaraki University, Japan, 1986.