TR-678

Derivation of Efficient Logic Programs by Synthesizing New Predicates

by

T. Kawamura

August, 1991

Derivation of Efficient Logic Programs by Synthesizing New Predicates

Tadashi KAWAMURA
Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan
tkawamur@icot.or.jp

Abstract

This paper shows a strategy of logic program transformation based on unfold/fold rules. In unfold/fold transformation, efficiency is often improved by performing folding steps, and folding steps are often allowed by synthesizing new auxiliary predicates. Since atoms in the body of a clause used to fold should be more general than atoms to be folded, new predicates are often found by generalizing atoms in the body of the clauses. Our method synthesizes new predicates automatically with suitable generalization steps in many cases. An extension of this method to incorporate goal replacement transformation is also shown.

1 Introduction

Program transformation is an important technique to derive correct and efficient programs. Unfold/fold transformation is one of the well-known program transformation techniques and its effectiveness was first demonstrated by Burstall and Darlington for functional programs[2]. In the field of logic programming, Tamaki and Sato proposed unfold/fold rules for definite clause programs which preserve equivalence of programs[17]. The correctness of unfold/fold rules was further investigated in successive works[5, 6, 15].

Though unfold/fold rules provide a very powerful methodology for program development, the application of those rules needs to be guided by strategies to obtain efficient programs. Recently, Pettorossi and Proietti showed some theoretical limitations to the mechanization of unfold/fold transformation[11, 12, 13]. However, in spite of those limitations, it will be worthwhile investigating the transformation strategies further.

In unfold/fold transformation, the efficiency improvement is mainly due to finding a recursive definition of a predicate, by performing folding steps. Introduction of auxiliary predicates often allows folding steps. Pettorossi and Proietti also proposed several transformation strategies which suggest the new predicates to be defined[11, 12, 13]. Further, they showed that their strategies are successful for any logic program by performing suitable generalization steps, which generalize the atoms to be folded for synthesizing new predicates[12, 13].

This paper shows a strategy of logic program transformation based on unfold/fold rules. New predicates are synthesized automatically to perform folding. A generalization technique is adopted in the synthesis process. An extension of this method to incorporate goal replacement transformation is also shown.

The rest of this paper is organized as follows. Section 2 describes program transformation rules and formalizes the transformation process. Section 3 describes a program transformation procedure which synthesizes new predicates automatically. Section 4 shows an extension of our method. Section 5 discusses the relations to other works and Section 6 gives a conclusion.

In the following, familiarity with the basic terminologies of first order logic is assumed[7]. A program is a set of definite clauses. As syntactical variables, $X, Y, \dots$ are used for variables, A, B for atoms, and K, L for multisets of atoms, possibly with primes and subscripts. In addition, θ is used for substitutions, and $A\theta$ for the atom obtained from atom A by applying substitution θ.

2 Logic Program Transformation

In this section, preliminary notions of our logic program transformation are described. At first, unfold/fold transformation rules are shown below [17].

Definition 2.1 Unfolding

Let P_i be a program, C be a clause in P_i, A be an atom in the body of C, and $C_1, C_2, \dots, C_k$ be all the clauses in P_i whose heads are unifiable with A, say by mgu's $\theta_1, \theta_2, \dots, \theta_k$. Let C'_i be the result of applying θ_i after replacing A in C with the body of C_i. Then $P_{i+1} = (P_i - \{C\}) \cup \{C'_1, C'_2, \dots, C'_k\}$. C is called the *unfolded clause*, $C_1, C_2, \dots, C_k$ are called the *unfolding clauses* and $C'_1, C'_2, \dots, C'_k$ are called the *results of unfolding C at A by $C_1, C_2, \dots, C_k$*.

Definition 2.2 Folding

Let P_i be a program, C be a clause in P_i of the form $A \leftarrow K, L$, and D be a clause of the form $B \leftarrow K'$, where K, K' and L are multisets of atoms. Suppose that there exists a substitution θ such that $K'\theta = K$ holds. Let C' be a clause of the form $A \leftarrow B\theta, L$. Then $P_{i+1} = (P_i - \{C\}) \cup \{C'\}$. C' is called the *result of folding C by D*.

Note that when applying folding, some conditions have to be satisfied to preserve the least Herbrand model of programs. See [17] for details.

Next, to formalize transformation processes, several notions are defined.

Definition 2.3 Descendant Clause and Ancestor Clause

Let P be a program, C be a clause in P, and P' be a program obtained from P by successively applying unfolding to P. A clause C' in P' is called a *descendant clause* of C when

(a) C' is identical to C, or

(b) C' is the result of unfolding a descendant clause of C.

Conversely, C is called an *ancestor clause* of C'.

Definition 2.4 U-selection Rule

A rule that determines what transformation should be applied to a program is called a *selection rule*. Let P be a program and C be a clause in P. A selection rule R is called a *U-selection rule for P rooted on C* when R always selects unfolding such that the unfolded clause is a descendant clause of C. C is called the *root clause for R*(or *of the transformation*.) A program obtained from P by successively applying unfolding according to R is called a program *obtained from P via R*.

Definition 2.5 Definition Clause Set

Let P be a program. A clause D is called a *definition clause for P* when all the predicates appearing in D's body are defined in P and the predicate of D's head does not appear in P. A set of clauses $\mathcal{D}$ is called a *definition clause set for P* when every element D of $\mathcal{D}$ is a definition clause for P and the predicate of D's head appears only once in $\mathcal{D}$.

Definition 2.6 Closed Program

Let P be a program, C be a clause in P, $\mathcal{D}$ be a definition clause set for P, and R be a U-selection rule for P rooted on C. Let P' be a program obtained from P via R. P' is said to be *closed with respect to* triple $< P, C, \mathcal{D} >$ when every non-unit descendant clause C' of C in P' satisfies one of the following:

(a) C' can be folded by a clause in $\mathcal{D} \cup \{C\}$.

(b) There exists an atom in the body of C' that is not unifiable with the head of any clause in P.

P is said to be *closed with respect to* C and $\mathcal{D}$ when there exists a closed program with respect to $< P, C, \mathcal{D} >$ and for every clause D in $\mathcal{D}$ there exists a closed program with respect to $< P \cup \{D\}, D, \mathcal{D} >$.

Our framework is a slight modification of the one which Pettorossi and Proietti has proposed with the notion of unfolding tree[11, 12, 13]. Our framework directly corresponds to the transformation process proposed by Tamaki and Sato[17].

Now, we can formalize our problem as follows: for given program P and clause C in P, find a finite definition clause set $\mathcal{D}$ such that P is closed with respect to C and $\mathcal{D}$.

Note that it is always possible to find a trivial answer for this problem. For example, for every predicate p appearing in P, construct a definition clause such that its body consists of only one atom whose predicate is p and whose arguments are all distinct variables, and its head is identical to its body except for the predicate symbol. A definition clause set consisting of those clauses merely renames predicates, but it satisfies the above problem. Obviously, efficiency of the program is not improved by using such a trivial set. In the next section, we propose a method to find a (possibly non trivial) set of new predicate definitions.

3 A Transformation Method by Synthesizing New Predicates

In this section, we describe a procedure to derive efficient programs by synthesizing new predicates.

3.1 Atomic Closure for Unfolding

Our idea is as follows. If we can predict what atoms appear in the bodies of clauses during a transformation process, we can specify clauses to fold them. Though an infinite number of distinct atoms may appear, we can represent them by a finite number of atoms, by regarding an atom as a set consisting of all its instances. This idea is formalized as follows.

Definition 3.1 Atomic Closure for Unfolding

Let P be a program, K be a multiset of atoms, D be a clause of the form $H \leftarrow K$, where H is an atom whose predicate does not appear in P nor K, and R be a U-selection rule for

$P \cup \{D\}$ rooted on D. Let P' be a program obtained from $P \cup \{D\}$ via R. A finite set of atoms S' is called an *atomic closure for unfolding of P with respect to K via R* when for every descendant clause of D in P', every atom appearing in its body is an instance of an atom in S'. A finite set of atoms S is called an *atomic closure for unfolding of P with respect to K* when S is an atomic closure of P with respect to K via any U-selection rule for P rooted on D.

In the following, we often say simply 'atomic closure' for 'atomic closure for unfolding'. A similar notion was given in [8] to show the correctness of partial evaluation.

Note that it is easy to construct a trivial atomic closure. For example, construct an atom whose arguments are all distinct variables for every predicate appearing in a program. A set consisting of those atoms is an atomic closure of the program. However, we are not interested in such a trivial set.

A generalization technique is required to represent an infinite number of atoms by a finite number of atoms. We use the notion of least general generalization(*lgg* for short)[10]. Let A and B be atoms. We write $A \preceq B$ when there exists a substitution θ such that $A\theta = B$ holds. Let S be a set of atoms. Then, atom A is an lgg of S when (a) for every atom B in S, $A \preceq B$ and (b) if A' is an atom such that $A' \preceq B$ for every atom B in S, then $A' \preceq A$. An algorithm to compute lgg was also shown in [10].

Further, we classify atoms by their unifiability with the heads of the clauses in a program. The atoms which can be unified with the different heads of clauses often behave in a different manner. Thus, the classification seems to be reasonable.

Definition 3.2 Unifiable Clause Set

Let P be a program and A be an atom. The set of all the clauses in P whose head is unifiable with A is called the *unifiable clause set of A with respect to P*.

Now, we show a naive procedure to calculate an atomic closure w.r.t. an atom.

Procedure 3.1 *Getting an atomic closure of a program w.r.t. an atom*
procedure atomic-closure ;
Input: P : a program and A_0 : an atom ;
Output: a set of pairs consisting of an atom and a set of clauses ;

```
$i := 0$ ;
$S_0 := \{(A_0, U_0)\}$, where $U_0$ be the unifiable clause set of $A_0$ w.r.t. $P$ ;
repeat
  begin
    $i := i + 1$ ;
    $S_i := S_{i-1}$ ;
    for every element $(A, U)$ in $S_i$ do
      begin
        let $\{C_1, \ldots, C_n\}$ be the unifiable clause set of $A$ w.r.t. $P$ ;
        let $\theta_k$ be an mgu of $A$ and the head of $C_k$ for $k = 1, \ldots, n$ ;
        for every atom $B$ appearing in the bodies of $C_1\theta_1, \ldots, C_n\theta_n$ do
          begin
            let $U_B$ be the unifiable clause set of $B$ w.r.t. $P$ ;
            if there exists an element of the form $(B', U_B)$ in $S_i$
            then $S_i := S_i - \{(B', U_B)\} \cup \{(B_{new}, U_B)\}$,
                  where $B_{new}$ is an lgg of $B$ and $B'$ ;
            else  $S_i := S_i \cup \{(B, U_B)\}$ ;
        end
    end
  end
until $S_i$ is identical to $S_{i-1}$ ;
return $S_i$ ;
```

A procedure similar to ours was given in [1] for the purpose of partial evaluation.

Next, we show the termination property and soundness of Procedure 3.1.

Theorem 3.1 For any program P and atom A_0 given as inputs, Procedure 3.1 always terminates.

Proof. Note that the number of possible unifiable clause sets is finite since they are subsets of P. Then, the number of the elements of every set of pairs S_i appearing in Procedure 3.1 is bounded. Suppose that S_i contains n distinct pairs for some i. Note that each atom appearing in the atom-part of an element of S_j is identical to or more general than the corresponding atom in S_i for $j > i$. Since for any atom A, the number of atoms which is more general than A is finite, one of the following holds.

(a) There exists a finite integer $k(> i)$ s.t. $S_k = S_{k+1}$ and S_k also contains n elements.

(b) There exists a finite integer $l(> i)$ s.t. S_{l-1} contains n elements and S_l contains $m(> n)$ elements.

In case (a), the procedure terminates. Consider case (b). The above discussion holds repeatedly for S_l whenever case (b) occurs. The number of the elements of S_l increases when case (b) occurs. Since the number of the elements is bounded, case (b) occurs for a finite number of times. Thus, eventually case (a) occurs and the procedure terminates. $\square$

Theorem 3.2 Suppose Procedure 3.1 returns a set of pairs S consisting of an atom and a set of clauses for given program P and atom A given as inputs. Let S_A be a set consisting of atoms appearing in the atom-part of every element in S. Then, S_A is an atomic closure of P w.r.t. A.

Proof. The theorem is proved by induction on the number of iterations on the repeat loop in Procedure 3.1. Let $C_1, \ldots, C_n$ be clauses in P whose heads are unifiable with A, say by mgu's $\theta_1, \ldots, \theta_n$, and $B_1, \ldots, B_m$ be all the atoms appearing in the bodies of $C_1\theta_1, \ldots, C_n\theta_n$. Let $S_P(B_i)$ be any atomic closure of P w.r.t. B_i for $i = 1, \ldots, m$. Then, there exists an atomic closure $S_P(A)$ of P w.r.t. A s.t.

$$S_P(A) = \{A\} \cup S_P(B_1) \cup \ldots \cup S_P(B_m) \tag{1}$$

Suppose that an atom represents a set of all its instances. Let T be a set of atoms and $U_P(T)$ be the set of pairs consisting of every atom in T and its unifiable clause set w.r.t. P. Let $F_P(T)$ be the set of all the atoms appearing in the atom-part of the set obtained from $U_P(T)$ and P by the repeat loop of Procedure 3.1. Then,

$$F_P(T \cup T') \supseteq F_P(T) \cup F_P(T') \tag{2}$$

holds since some atoms in $F_P(T)$ ($F_P(T')$) may be generalized by computing lgg with atoms in $F_P(T')$ ($F_P(T)$) when computing $F_P(T \cup T')$. Further, evidently

$$F_P(T) \supseteq F_P(T') \quad \text{if } T \supseteq T' \tag{3}$$

holds. By applying F_P to $\{A\}$, the following holds.

$$F_P(\{A\}) \supseteq \{A, B_1, \ldots, B_m\} \tag{4}$$

From (2), (3) and (4),

$$\begin{aligned} F_P^{j+1}(\{A\}) &\supseteq F_P^j(\{A, B_1, \ldots, B_m\}) \\ &\supseteq F_P^j(\{A\}) \cup F_P^j(\{B_1\}) \cup \ldots F_P^j(\{B_m\}) \\ &\supseteq \{A\} \cup F_P^j(\{B_1\}) \cup \ldots F_P^j(\{B_m\}) \end{aligned} \tag{5}$$

From Theorem 3.1, there exists a finite integer k s.t. $F_P^k(\{B_i\}) = F_P^{k-1}(\{B_i\})$ holds for $i = 1, \ldots, m$. By the induction hypothesis,

$$F_P^k(\{B_i\}) \supseteq S_P(B_i) \tag{6}$$

holds. Thus, from (1), (5) and (6), $F_P^{k+1}(A) \supseteq S_P(A)$ holds, which means $F_P^{k'+1}(A)$ is an atomic closure of P w.r.t. A. □

Evidently, an atomic closure of a program w.r.t. a multiset of atoms K is the union of the atomic closures of the program w.r.t. all the elements of K.

Example 3.1 Let P be a program as follows :
C_1 : rev([],Z,Z).
C_2 : rev([A|X],Y,Z) ← rev(X,Y,[A|Z]).
An atomic closure S of P w.r.t. rev(X,Y,Z) is calculated by Procedure 3.1 as follows. First, a set S_0 ={(rev(X,Y,Z),$\{C_1, C_2\}$)} is given. Next, rev(X,Y,[A|Z]) is obtained from rev(X,Y,Z) and C_2. But this atom is generalized into rev(X,Y,Z) by computing lgg, because it is also unifiable with C_1 and C_2. Hence, S_0 is not changed in this step. As a result, S ={rev(X,Y,Z)}.

Note that in Example 3.1, an infinite number of distinct atoms appear when applying unfolding successively, that is, rev(X,Y,[A|Z]),rev(X,Y,[B,A|Z]),⋯. Those atoms are represented by the single atom rev(X,Y,Z).

Example 3.2 Let P be a program as follows :
C_1 : subseq([],L).
C_2 : subseq([X|L],[X|M]) ← subseq(L,M).
C_3 : subseq([X|L],[Y|M]) ← subseq([X|L],M).
C_4 : csub(X,Y,Z) ← subseq(X,Y), subseq(X,Z).
An atomic closure S of P w.r.t. subseq(L,M) is calculated by Procedure 3.1 as follows:
S_0 = {(subseq(L,M),$\{C_1, C_2, C_3\}$)}
S_1 = {(subseq(L,M),$\{C_1, C_2, C_3\}$),(subseq([X|L],M),$\{C_2, C_3\}$)}
$S_2 = S_1$
Then, S ={subseq(L,M),subseq([X|L],M)}.

Note that in Example 3.2, S_0 is also an atomic closure. However, considering the unifiability with the heads of the clauses, subseq([X|L],M) is also included. In fact, when transforming the definition of 'csub' into a recursive one, subseq([X|L],M) will appear in the body of a new predicate definition.

3.2 Synthesis of New Predicates by Using Unfolding Closure

In this subsection, we propose a transformation procedure which incorporates a method to find a (possibly non-trivial) set of new predicate definitions. To synthesize a new predicate, we have to predict what multisets of atoms appear in the body of clauses during a transformation process. (Note that atoms in a multiset may share some terms.) We consider a method to construct such multisets from an atomic closure.

Definition 3.3 Cover of Multiset

Let K be a multiset of atoms and S be a set of atoms. A multiset of atoms L is called a *cover of K by S* when
(a) there exists a substitution θ such that $L\theta = K$, and
(b) for every element A of L, S includes a variant of A.

Note that for any finite multiset of atoms K and any set of atoms S, the number of distinct covers of K by S is finite. For our purpose, it is sufficient to get a cover by an atomic closure. Now, after giving one more definition, we will show a procedure to get a cover.

Definition 3.4 Minimally General Atom

Let A be an atom and S be a set of atoms. Then, an atom B in S is called a *minimally general atom of A in S* when (a) $B \prec A$ and (b) for any atom B' in S such that $B' \preceq A$, $B \not\preceq B'$.

Procedure 3.2 *Getting a cover of a multiset*

```
procedure cover-of-multiset :
Input :  $K = \{A_1, \dots, A_n\}$ : a multiset of atoms and $S$ : a set of atoms ;
Output :  $K' = \{B_1, \dots, B_n\}$ : a multiset of atoms ;

select a minimally general atom $A'_i$ of $A_i$ in $S$ for $i = 1, \dots, n$ ;
construct atom $B_i$ from $A_i$ for $i = 1, \dots, n$ as follows :

    for every argument $t$ in $A_i$, do one of the following:

    (1) if all the variables in $t$ appear only once in $K$, then replace $t$ in $A_i$ by the
        corresponding argument in $A'_i$ ;
    (2) if a variable in $t$ appears in another place in $K$, then
        (2.1) leave $t$ in $A_i$ as it is, if $t$ is a variant of the corresponding argument of
              $A'_i$. ;
        (2.2) replace $t$ in $A_i$ by the corresponding argument in $A'_i$, otherwise ;

return $K' = \{B_1, \dots, B_n\}$ ;
```

Proposition 3.3 Suppose that for multiset K of atoms and set S of atoms given as inputs, Procedure 3.2 returns a multiset of atoms K'. Then, K' is a minimally general cover of K by S, i.e., there does not exist a multiset of atoms L which satisfies the following :
(a) L is a cover of K by S and
(b) there exists a substitution θ such that $L = K'\theta$.

Proof. Obvious. □

Example 3.3 Let S be the set {reverse(L_1,M_1), append(N_2,X_2,M_2), append([],M_3,N_3), append([X_4],M_4,N_5)}. Then, a cover of {reverse(L_0,N_0), append(N_0,[X_0],M_0)} by S obtained by Procedure 3.2 is {reverse(L,N), append(N,X,M)}.

Next, we show our program transformation procedure. A clause C is called a *terminal clause of* program P when C is a unit clause or there exists an atom in C's body that is not unifiable with the head of any clause in P.

Procedure 3.3 *program transformation*

```
procedure transform ;
Input : $P_0$ : a program, $C_0$ : a clause in $P$, and $k$ : a finite integer ;
Output : $P$ : a program ;

let $S$ be an atomic closure of $P_0$ w.r.t. the body of $C_0$ ;
$\mathcal{D} := \{C_0\}$ ;
while there exist clauses in $\mathcal{D}$ that are not marked 'selected' do
  begin
    select an unmarked clause $D$ from $\mathcal{D}$ and mark $D$ 'selected' ;
    $P := P \cup \{D\}$ ;
    $P' := \{D\}$ ;
    while $P'$ includes non-terminal clauses do
      begin
        select a non-terminal clause $C$ from $P'$ ;
        $P' := P' - \{C\}$ ;
        if $C$ can be folded by a clause $D'$ in $\mathcal{D}$
          then $P := P - \{C\} \cup \{C'\}$, where $C'$ is the result of folding $C$ by $D'$
                 (folding may be applied to $C$ successively, if possible) ;
          else if execute one of the following nondeterministically ;
                   (1) $(P, \mathcal{D})$ := define($P, C, \mathcal{D}, S, k$) ;
                   (2) $(P, P')$ := unfold($P, C, S, P'$) ;
      end
  end
return $P$ ;
```

```
procedure define ;
Input : $P$ : a program, $C$ : a clause, $\mathcal{D}$ : a set of clauses, $S$ : a set of atoms,
        and $k$ : an integer ;
Output : $P_{new}$ : a program and $\mathcal{D}_{new}$ : a set of clauses ;

suppose that $C$ is of the form $A \leftarrow K, L$, where the number of $K$'s elements is less than $k$ ;
let $K'$ be a cover of $K$ by $S$ ;
synthesize a new clause $D$ of the form $B \leftarrow K'$, where $B$ is an atom such that its predicate
does not appear in $P \cup \mathcal{D}$ and it contains all the variables appearing in $K'$ as its arguments ;
let $C'$ be the result of folding $C$ by $D$ ;
$P_{new} := P - \{C\} \cup \{C'\}$ ;
$\mathcal{D}_{new} := \mathcal{D} \cup \{D\}$ ;
return $P_{new}$ and $\mathcal{D}_{new}$ ;
```

```
procedure unfold ;
Input : $P$ : a program, $C$ : a clause, S : a set of atoms, and $P'$ : a set of clauses ;
Output : $P_{new}$ : a program and $P'_{new}$ : a set of clauses ;

select an atom $A$ from $C$'s body s.t. there exists a minimally general atom $A'$ of $A$ in $S$ which
is not marked with $C$'s ancestor clause (if there is no such atom, unfolding is forbidden) ;
mark $A'$ with $C$ ;
let $C_1, \ldots, C_m$ be the results of unfolding $C$ at $A$ by the clauses in $P$ ;
$P_{new} := P - \{C\} \cup \{C_1, \ldots, C_m\}$ ;
$P'_{new} := P' \cup \{C_1, \ldots, C_m\}$ ;
return $P_{new}$ and $P'_{new}$ ;
```

Note that Procedure 3.3 does not depend on the method of calculating atomic closures nor the method of calculating covers. Any atomic closures and covers are available. Note also that an atomic closure is used to restrict the application of unfolding.

Theorem 3.4 When any program P, clause C_0 in P and integer k are given as inputs, Procedure 3.3 always terminates for any nondeterministic choice in it.

Proof. First, we show that only a finite number of new predicates can be defined. Note that the body of each new predicate definition is constructed from the atoms appearing in an atomic closure of P w.r.t. the body of C_0, allowing some atoms in the body to share some terms. Since the atomic closure is a finite set and the number of terms appearing in it is also finite, the number of distinct new predicates is finite.

Next, we show that unfolding can not be applied an infinite number of times. Consider the application of unfolding to a clause C. Let S be an atomic closure used in the transformation process. Suppose that n elements of S are not marked with ancestor clauses of C. Let C' be a clause in the results of unfolding C. Then, when applying unfolding to C', $n-1$ elements of S are not marked. By induction on the number of unmarked elements of S, unfolding can be applied to each of C''s descendant clauses only a finite number of times. As only a finite number of clauses are generated as the result of unfolding, unfolding can be applied to C's descendant clauses only a finite number of times.

Note that the above properties hold for any nondeterministic choice in Procedure 3.3. In the inner while-loop in the procedure transform, the number of clauses in P' is reduced when folding is applied and may be increased by applying unfolding. Then, since unfolding can be applied to each clause only a finite number of times, the number of clauses in P' is bounded. Thus, the inner while-loop eventually terminates. In the outer while-loop in the procedure transform, the number of clauses in $\mathcal{D}$ is bounded by the number of new predicate definitions, which is finite as shown above. Thus, the outer while-loop also eventually terminates. Hence, Procedure 3.3 always terminates for any nondeterministic choice in it. $\square$

The root clause and all definition clauses are eventually folded if they are not transformed into terminal clauses. For nondeterministic choices in the procedure, different choices can derive different programs. It is important to investigate which choice can derive an efficient program, although we do not discuss this problem in this paper.

Example 3.4 Let C_1, C_2, C_3 and C_4 be clauses in Example 3.2. Let P be a program consisting of $\{C_1, C_2, C_3, C_4\}$. Recall that an atomic closure S of P w.r.t. subseq(L,M) is {subseq(L,M),(subseq([A|L],M)}, as shown in Example 3.2. Suppose that C_4 is the root of the transformation. By unfolding the first atom in C_4's body, the following clauses are obtained.

C_5 : csub([],Y,Z) ← subseq([],Z).
C_6 : csub([A|X],[A|Y],Z) ← subseq(X,Y),subseq([A|X],Z).
C_7 : csub([B|X],[A|Y],Z) ← subseq([B|X],Y),subseq([B|X],Z).

Mark subseq(L,M) in S with C_4. By unfolding C_5 further, clause C_8 is obtained, where

C_8 : csub([],Y,Z).

By folding C_7 by C_4, clause C_9 is obtained, where

C_9 : csub([B|X],[A|Y],Z) ← csub([B|X],Y,Z).

Since folding can not be applied to C_6, a new predicate is synthesized to fold C_6. Procedure 3.2 gives {subseq(X,Y),subseq([A|X],Z)} as a cover of C_6's body by S. Then, new predicate 'cs' is synthesized as follows.

D : cs(A,X,Y,Z) ← subseq(X,Y),subseq([A|X],Z).

By folding C_6 by D, clause C_{10} is obtained, where

C_{10} : csub([A|X],[A|Y],Z) ← cs(A,X,Y,Z).

Now all the descendant clauses of C_4 have been dealt with. Next, D is regarded as the root clause. By unfolding the second atom of D's body and then applying folding, we can obtain the following clauses.

D_1 : cs(A,X,Y,[A|Z]) ← csub(X,Y,Z).
D_2 : cs(A,X,Y,[B|Z]) ← cs(A,X,Y,Z).

As a result, program $P' = \{C_8, C_{10}, D_1, D_2\}$ is obtained.

Pettorrosi and Proietti pointed out that a definition clause may be trivial when the atoms in its body do not share any terms[11, 12, 13]. Our method may construct such a definition clause since some shared terms may be replaced by new terms in (2.2) in Procedure 3.2. In general, it is very difficult to avoid such definition clauses because some shared variables may be instantiated an infinite number of times by unfolding, which may produce an infinite number of distinct multisets of atoms.

However, if the depth of every term appearing in the bodies of clauses during a transformation process is known to be finite, only a finite number of distinct multisets can appear in the body when the sizes of those multisets are bounded. Thus, (2) in Procedure 3.2 can be modified as follows.

(2) If a variable in t appears in another place in K, then leave t in A_i as it is.

Proietti and Pettorossi showed some classes of programs that satisfy the above restriction[12]. Our modified method can be applied to those classes of programs. Note that we can weaken the restriction for programs because we do not care about the depth of non-shared terms.

4 An Extension - Incorporating Goal Replacement

Tamaki and Sato proposed goal replacement transformation that is consistent with unfold/fold transformation [17]. In many cases, efficient programs can be derived by unfold/fold transformation combined with goal replacement. In this section, we extend our framework to incorporate goal replacement. First, goal replacement transformation is described following [17].

Definition 4.1 Replacement Rule

A *replacement rule* is a rule of the form :

$\exists X_1, \ldots, X_m K \rightarrow \exists Y_1, \ldots, Y_n L$

where K and L are conjunction of atoms and $X_1, \ldots, X_m, Y_1, \ldots, Y_n$ are distinct variables.

Example 4.1 The following is a replacement rule r which represents the associativity of 'append'.

r : $\exists$ LM (append(L,M,LM),append(LM,N,LMN))
$\rightarrow \exists$ MN (append(M,N,MN),append(L,MN,LMN))

Definition 4.2 Goal Replacement

Let P_i be a program, C be a clause in P of the form $A \leftarrow K, K'$ and r be a replacement rule of the form $\exists X_1, \ldots, X_m L \rightarrow \exists Y_1, \ldots, Y_n L'$. Suppose that there exists a substitution θ such that $K = L\theta$. Then, $P_{i+1} := P_i - \{C\} \cup \{C''\}$, where C'' is a clause of the form $A \leftarrow L'\theta, K'$. C is called the *applied clause*, each atom in K is called the *applied atom*. r is called the *applying rule* and C'' is *the result of the application of r to C*.

Note that when applying replacement rules, several conditions have to be satisfied to preserve the least Herbrand model of programs. See [17] for details.

The notion of atomic closure for unfolding can be easily extended to that of *atomic closure for unfolding and goal replacement*. Now, we redefine some notions.

Definition 4.3 Descendant Clause and Ancestor Clause *(revised)*

Let P be a program, C be a clause in P, and $\mathcal{R}$ be a set of replacement rules. Let P' be a program obtained from P by successively applying unfolding or replacement rules in $\mathcal{R}$ to P. A clause C' in P' is called a *descendant clause* of C when

(a) C' is the result of unfolding or applying a replacement rule in $\mathcal{R}$ to C or

(b) C' is the result of unfolding or applying a replacement rule in $\mathcal{R}$ to a descendant clause of C.

Conversely, C is called an *ancestor clause* of C'.

Definition 4.4 UR-selection Rule

Let P be a program, C be a clause in P, and $\mathcal{R}$ be a set of replacement rules. A selection rule R is called a *UR-selection rule for P with $\mathcal{R}$ rooted on C* when R always selects unfolding or goal replacement by rules in $\mathcal{R}$ such that the unfolded/applied clause is a descendant clause of C. C is called the *root clause for R*(or *of the transformation*.) A program obtained from P by successively applying unfolding or replacement rules in $\mathcal{R}$ according to R is called a program *obtained from P with $\mathcal{R}$ via R*.

Definition 4.5 Closed Program*(revised)*

Let P be a program, C be a clause in P, $\mathcal{R}$ be a set of replacement rules, $\mathcal{D}$ be a finite set of definition clauses for P, and R be a UR-selection rule for P with $\mathcal{R}$ rooted on C. Let P' be a program obtained from P via R. P' is said to be *closed with respect to* quadruplet $< P, C, \mathcal{D}, \mathcal{R} >$ when every non-unit descendant clause C' of C in P' satisfies one of the following :

(a) C' can be folded by a clause in $\mathcal{D} \cup \{C\}$.

(b) There exists an atom in the body of C' that is not unifiable with the head of any clause in P.

P is said to be *closed with respect to* triple $< C, \mathcal{D}, \mathcal{R} >$ when there exists a closed program with respect to $< P, C, \mathcal{D}, \mathcal{R} >$ and for every clause D in $\mathcal{D}$ there exists a closed program with respect to $< P \cup \{D\}, D, \mathcal{D}, \mathcal{R} >$.

Again, our problem is formalized as follows : for given program P, clause C in P and set $\mathcal{R}$ of replacement rules, find a finite definition clause set $\mathcal{D}$ such that P is closed with respect to $< C, \mathcal{D}, \mathcal{R} >$.

Definition 4.6 Atomic Closure for Unfolding and Goal Replacement

Let P be a program, K be a multiset of atoms, $\mathcal{R}$ be a set of replacement rules, D be a clause of the form $H \leftarrow K$, where H is an atom whose predicate does not appear in P, and R be a UR-selection rule for $P \cup \{D\}$ with $\mathcal{R}$ rooted on D. Let P' be a program obtained from $P \cup \{D\}$ with $\mathcal{R}$ via R. A finite set of atoms S' is called an *atomic closure for unfolding and goal replacement of P with respect to K with $\mathcal{R}$ via R* when for every descendant clause of D in P', every atom appearing in its body is an instance of an atom in S'. A finite set of atoms S is called an *atomic closure for unfolding and goal replacement of P with respect to K with $\mathcal{R}$* when S is an atomic closure for unfolding and goal replacement of P with respect to K with $\mathcal{R}$ via any UR-selection rule for P with $\mathcal{R}$ rooted on D.

In the following, we say simply 'extended atomic closure' for 'atomic closure for unfolding and goal replacement'.

Again, we consider an extended atomic closure which is concerned with the unifiability with the heads of the clauses in a program. After giving a definition as a preliminary, we will show a naive method to get such an extended atomic closure.

Definition 4.7 Compatible Multiset

Let K be a multiset of atoms and S be a set of atoms. A multiset of atoms L is called a *compatible multiset with K from S* when

(a) there exists a substitution θ such that $K\theta = L\theta$ and

(b) for each element A in L, S includes a variant of A.

Note that for any finite multiset of atoms K and any finite set of atoms S, the number of distinct compatible multisets with K from S is finite.

Suppose that procedure 'atomic-closure' in Procedure 3.1 is modified to be applied to a set of pairs consisting of an atom and its unifiable clause set, by using the repeat loop in the procedure only.

Procedure 4.1 *Getting an extended atomic closure*
procedure extended-atomic-closure ;
Input : P : a program, A : an atom and R : a set of replacement rules ;
Output : a set of pairs consisting of an atom and a set of clauses ;

```
$i := 0$ ;
$S_0 := S$ ;
repeat
  begin
    $i := i + 2$ ;
    $S_{i-1}$ :=atomic-closure($S_{i-2}$,$P$) ;
    $S_i$ :=closure-for-goal-replacement($S_{i-1}$,$P$,$\mathcal{R}$) ;
  end
until $S_i$ is identical to $S_{i-2}$
return $S_i$ ;
```

procedure closure-for-goal-replacement :
Input : S : a set of pairs consisting of an atom and a set of clauses, P : a program, and R : a set of replacement rules ;
Output : a set of pairs consisting of an atom and a set of clauses ;

```
$i := 0$ ;
$S_0 := S$ ;
repeat
  begin
    $i := i + 1$ ;
    $S_i := S_{i-1}$ ;
    for every rule $K \to L$ in $R$ do
      begin
        let $S_{atom}$ be a set of all the atoms appearing in the atom-part of an element of $S_i$ ;
        let $S_K$ be the set consisting of all the compatible multiset with $K$ from $S_{atom}$ ;
        for every element $K'$ in $S_K$ do
          begin
            let $\theta$ be an mgu of $K$ and $K'$ ;
            for every atom $A$ appearing in $L\theta$ do
              begin
                let $U$ be the unifiable clause set of $A$ w.r.t. $P$ ;
                if there exists an element of the form $(A', U)$ in $S_i$
                  then $S_i := S_i - \{(A', U)\} \cup \{(B, U)\}$, where $B$ is an lgg of $A$ and $A'$ ;
                  else $S_i := S_i \cup \{(A, U)\}$ ;
              end
          end
      end
  end
until $S_i$ is identical to $S_{i-1}$ ;
return $S_i$ ;
```

Procedure 4.1 always terminates and an extended atomic closure can be easily obtained. (This is shown in a similar way to Theorems 3.1 and 3.2.)

Note that the procedure 'closure-for-goal-replacement' tries to apply goal replacement to any possible combination of atoms. During the actual transformation process, some of the combinations may not appear or some of the applications may be impossible. Thus, an extended atomic closure obtained by Procedure 4.1 may include superfluous atoms.

Example 4.2 Let P be a program as follows:

C_1 : reverse([],[]).
C_2 : reverse([X|L],M) ← reverse(L,N),append(N,[X],M).
C_3 : append([],M,M).
C_4 : append([X|L],M,[X|N]) ← append(L,M,N).

Let $\mathcal{R}$ be a set consisting of only one replacement rule r, where

r : ∃ LM (append(L,M,LM),append(LM,N,LMN))
→ ∃ MN (append(M,N,MN),append(L,MN,LMN)).

An extended atomic closure of P w.r.t. C_2's body with $\mathcal{R}$ is calculated by Procedure 4.1 as follows. By applying 'atomic-closure' to P and C_2's body, a set S_0 is obtained, where

$S_0 = \{$(reverse(L,N),$\{C_1, C_2\}$),(append(N,[X],M),$\{C_3, C_4\}$)$\}$.

Now the procedure 'closure-for-goal-replacement' is applied to P,S_0 and $\mathcal{R}$. First, the only possible application of r to the atoms in S_0 is as follows.

append(L,[X],M),append(M,[Y],N) → append([X],[Y],M_1),append(L,M_1,N)

Then, a set S_1 is obtained, where

$S_1 = \{$(reverse(L_1,N_1),$\{C_1, C_2\}$), (append(N_2,X_2,M_2),$\{C_3, C_4\}$),
(append([X_3],[Y_3],N_3),$\{C_4\}$)$\}$.

Next, r can be applied to the atoms appearing in S_1 in the following four ways.

append(L,M,L′),append(L′,N,N′) → append(M,N,M′),append(L,M′,N′)
append([X],[Y],Z),append(Z,M,N) → append([Y],M,Z′),append([X],Z′,N)
append(L,M,[X]),append([X],[Y],N) → append(M,[Y],M′),append(L,M′,N)
append([X],[Y],[U]),append([U],[V],Z) → append([Y],[V],W),append([X],W,Z)

(Obviously, the fourth application is incorrect since append([X],[Y],[U]) is unsatisfiable. We can ignore such incorrect applications in some cases. However, we do not discuss the problem here.) Then, a set S_2 is obtained, where

$S_2 = \{$(reverse(L_1,N_1),$\{C_1, C_2\}$), (append(N_2,X_2,M_2),$\{C_3, C_4\}$),
(append([X_3],M_3,N_3),$\{C_4\}$)$\}$.

Further application of the above procedure does not change S_2. Thus, S_2 is the output of 'closure for goal replacement'. Next, by applying 'atomic-closure' to P and S_2, a set S_3 is obtained, where

$S_3 = \{$(reverse(L_1,N_1),$\{C_1, C_2\}$), (append(N_2,X_2,M_2),$\{C_3, C_4\}$),
(append([],M_3,N_3),$\{C_3\}$), (append([X_4],M_4,N_4),$\{C_4\}$)$\}$.

The result of applying 'closure-for-goal-replacement' to P,S_3 and $\mathcal{R}$ is identical to S_3 itself. Then, an extended atomic closure of P w.r.t. C_2's body with $\mathcal{R}$ obtained by Procedure 4.1 is $\{$reverse(L_1,N_1), append(N_2,X_2,M_2), append([],M_3,N_3), append([X_4],M_4,N_4)$\}$.

Next, we show our transformation procedure incorporating goal replacement.

Procedure 4.2 *program transformation*

```
procedure transform ;
Input : $P_0$ : a program, $C_0$ : a clause in $P$, $\mathcal{R}$ : a set of replacement rules,
        and $k$ : a finite integer ;
Output : $P$ : a program ;

let $S$ be an extended atomic closure of $P_0$ w.r.t. the body of $C_0$ with $\mathcal{R}$ ;
$\mathcal{D} := \{C_0\}$ ;
while there exist clauses in $\mathcal{D}$ that are not marked 'selected' do
  begin
    select an unmarked clause $D$ from $\mathcal{D}$ and mark $D$ 'selected' ;
    $P := P \cup \{D\}$ ;
    $P' := \{D\}$ ;
    while $P'$ includes non-terminal clause do
      begin
        select a non-terminal clause $C$ from $P'$ ;
        $P' := P' - \{C\}$ ;
        if $C$ can be folded by a clause $D'$ in $\mathcal{D}$
          then $P := P - \{C\} \cup \{C'\}$, where $C'$ is the result of folding $C$ by $D'$ ;
          else if execute one of the following nondeterministically ;
                   (1) $(P, \mathcal{D})$ := define($P, C, \mathcal{D}, S, k$) ;
                   (2) $(P, P')$ := unfold($P, C, S, P'$) ;
                   (3) $(P, P')$ := goal-replacement($P, C, \mathcal{R}, S, P'$) ;
      end
  end
return $P$ ;
```

```
procedure goal-replacement ;
Input : $P$ : a program, $C$ : a clause, $\mathcal{R}$ : a set of replacement rules, S : a set of atoms,
        and $P'$ : a set of clauses ;
Output : $P_{new}$ : a program and $P'_{new}$ : a set of clauses ;

suppose that $C$ is of the form $A \leftarrow L, L'$ ;
select a replacement rule $r$ of the form $K \rightarrow K'$ from $\mathcal{R}$ s.t.
(a) there exists a substitution $\theta$ s.t. $K = L\theta$, and
(b) there exists a multiset of atoms $L_0$ s.t.
    (b-1) for every element $A$ of $L$, $L_0$ includes an atom $A'$ which is a minimally general
          atom of $A$ in $S$, and
    (b-2) $L_0$ is not marked with an ancestor clause of $C$ and $r$ ;
(If there does not exist such $r$ in $\mathcal{R}$, goal replacement is forbidden.)
mark $L_0$ with $C$ and $r$ ;
let $C'$ be the result of the application of $r$ to $C$ ;
$P_{new} := P - \{C\} \cup \{C'\}$ ;
$P'_{new} := P \cup \{C'\}$ ;
return $P_{new}$ and $P'_{new}$ ;
```

The procedures define and unfold are the ones given in Procedure 3.3.

Note that Procedure 4.2 always terminates. (This is shown in a similar way to Theorem 3.4.)

Example 4.3 Let P be the program and $\mathcal{R}$ be the set of replacement rules in Example 4.2. Suppose that C_2 is selected as a root clause. Let S be an extended atomic closure S of P w.r.t. C_2's body with $\mathcal{R}$ obtained in Example 4.2. Recall that a cover of C_2's body by S is {reverse(L,N),append(N,X,M)}, as shown in Example 3.3. Then, a new predicate 'rev' is defined as follows.

```
rev(L,M,N,X) ← reverse(L,N),append(N,X,M).
```

This definition is essentially the same as the one used in a well-known example transforming an $O(n^2)$ list reversal algorithm to a linear one. Our method allows the usual transformation steps. Then, the following program is obtained.

```
rev([],X,[],X).
rev([A|L],M,N,X) ← rev(L,M,N',[A|X]).
```

Obviously, the third argument of 'rev' is redundant. A method to remove redundant arguments called 'truncation' is shown in [3].

Note that the definition of 'rev' is automatically derived, while it was given manually in most of previous works. It is well-known that list processing programs using 'append' can be often optimized by adding an extra argument as an accumulator. The transformation of list reversal is an example of such optimization. Example 4.3 shows that the optimization process can be mechanized by our method.

5 Discussion

As described in section 2, our framework of logic program transformation is similar to that of Pettorossi and Proietti[11, 12, 13]. They showed the effectiveness of the generalization strategy[12, 13]. The purpose of the generalization is to bound: (a) the depth of terms in the body of a clause (vertical bound) and (b) the length of a chain of atoms in the body of a clause which share variables (horizontal bound). New predicates are defined to fold the generalized atoms. However, as Pettorossi and Proietti pointed out, the application of the generalization should be restricted. The unrestricted use of generalization reduces the number of variables shared among atoms and often fails to improve efficiency of programs. They showed a solution to the problem, which gives the vertical and horizontal bounds.

They also proposed a notion of program projections[11]. Projections generate unary programs by noticing only one argument of a predicate and abstracting others. The program projection is also regarded as a restricted use of generalization.

Our method also incorporates a generalization technique. In many cases, Procedures 3.1 and 4.1 can produce appropriately generalized atoms. Some arguments of an atom are abstracted when they are not concerned with the unifiability with the heads of clauses. We believe that our method is another possible solution to the generalization technique.

The work described here has common features to partial evaluation, rather than the con ventional works on transformation strategies(cf. [14]). In the researches of partial evaluation, recent works investigated the use of folding(or similar rules) [4, 9] and the restricted use of goal replacement[16]. We hope that further investigation in this direction will contribute to the research of the advanced program development.

6 Conclusion

A logic program transformation method has been described. This method is based on unfold/fold transformation and synthesizes new predicates automatically to derive a recursive definition of a predicate. An extension to incorporate goal replacement has also been described.

Acknowledgments

I would like to thank Kazuhiro Fuchi for giving me the opportunity to do this research. I also wish to thank Koichi Furukawa and Ryuzo Hasegawa for their advice.

References

[1] Benkerimi. K. and J. W. Lloyd, "A Partial Evaluation Procedure for Logic Programs", Proc. of the 1990 North American Conference on Logic Programming, pp.343-358, Austin, 1990.

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

[3] Fribourg, L., "Extracting Logic Programs from Proofs that Use Extended Prolog Execution and Induction", Proc. of 7th International Conference on Logic Programming, pp.685-699, Jerusalem, 1990.

[4] Futamura, Y., and K. Nogi, "Generalized Partial Computation", in Partial Evaluation and Mixed Computation, eds. D. Bjørner, A. P. Ershov and N. D. Jones, pp.133-151, North-Holland, 1988.

[5] Kanamori, T. and H. Fujita, "Unfold/Fold Logic Program Transformation with Counters", Presented at U.S-Japan Workshop on Logic of Programs, Honolulu, 1987. Also a preliminary version appical Report TR-179, 1986.

[6] Kawamura, T. and T. Kanamori, "Preservation of Stronger Equivalence in Unfold/Fold Logic Program Transformation", in Theoretical Computer Science Vol.75, Nos. 1/2, pp.139-156, 1990.

[6] Kawamura, T. and T. Kanamori, "Preservation of Stronger Equivalence in Unfold/Fold Logic Program Transformation", in Theoretical Computer Science Vol.75, Nos. 1/2, pp.139-156, 1990.

[7] Lloyd, J. W., "Foundations of Logic Programming", Springer-Verlag, 2nd Edition, Berlin, Heidelberg, New York, 1987.

[8] Lloyd. J. W. and J. C. Shepherdson,"Partial Evaluation in Logic Programming", Technical Report CS-87-09. Department of Computer Science. University of Bristol, 1987.

[9] Owen, S., "Issues in the partial evaluation of meta-interpreters", in Meta-Programming in Logic Programming, eds. H. D. Abramson and M. H. Rogers, MIT Press, pp.319-340, 1989.

[10] Plotkin, G.D., "A note on inductive generalization", in Machine Intelligence 5, eds. B. Meltzer and D. Michie, Elsevier North-Holland, pp.153-163, New York, 1970.

[11] Pettorossi, A. and M. Proietti, "Decidability Results and Characterization of Strategies for the Development of Logic Programs", Proc. of 6th International Conference on Logic Programming, pp.539-553, Lisboa, 1989.

[12] Proietti, M. and A. Pettorossi, "Construction of Efficient Logic Programs by Loop Absorption and Generalization", Proc. of the Second Workshop on Meta-programming in Logic, pp.57-81, Leuven, 1990.

[13] Proietti, M. and A. Pettorossi, "Synthesis of Eureka Predicates for Developing Logic Programs", Proc. of 3rd European Symposium on Programming, Copenhagen, LNCS 432, Springer-Verlag, pp.307-325,1990.

[14] Seki, H. and Furukawa, K, "Notes on Transformation Techniques for Generate and Test Logic Programs", 4th IEEE Symp. on Logic Programming, San Francisco, pp.215 223.

[15] Seki, H., "Unfold/Fold Transformation of Stratified Programs", Proc. of 6th International Conference on Logic Programming, pp.554-568, Lisboa, 1989.

[16] Takeuchi, A. and H. Fujita, "Competitive Partial Evaluation - Some Remaining Problems of Partial Evaluation", An International Journal on New Generation Computing, Vol.6, Nos. 2,3. pp.259 278, 1988.

[17] Tamaki, H. and T. Sato, "Unfold/Fold Transformation of Logic Programs", Proc. of 2nd International Logic Programming Conference, pp.127–138, Uppsala, 1984.

The work described here has common features to partial evaluation, rather than the conventional works on transformation strategies(cf. [14]). In the researches of partial evaluation, recent works investigated the use of folding(or similar rules) [4, 9] and the restricted use of goal replacement[16]. We hope that further investigation in this direction will contribute to the research of the advanced program development.

6 Conclusion

A logic program transformation method has been described. This method is based on unfold/fold transformation and synthesizes new predicates automatically to derive a recursive definition of a predicate. An extension to incorporate goal replacement has also been described.

Acknowledgments

I would like to thank Kazuhiro Fuchi for giving me the opportunity to do this research. I also wish to thank Koichi Furukawa and Ryuzo Hasegawa for their advice.

References

[1] Benkerimi, K. and J. W. Lloyd, "A Partial Evaluation Procedure for Logic Programs", Proc. of the 1990 North American Conference on Logic Programming, pp.343-358, Austin, 1990.

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

[3] Fribourg, L., "Extracting Logic Programs from Proofs that Use Extended Prolog Execution and Induction", Proc. of 7th International Conference on Logic Programming, pp.685-699, Jerusalem, 1990.

[4] Futamura, Y., and K. Nogi, "Generalized Partial Computation", in Partial Evaluation and Mixed Computation, eds. D. Bjørner, A. P. Ershov and N. D. Jones, pp.133-151, North-Holland, 1988.

[5] Kanamori, T. and H. Fujita, "Unfold/Fold Logic Program Transformation with Counters", Presented at U.S-Japan Workshop on Logic of Programs, Honolulu, 1987. Also a preliminary version appical Report TR-179, 1986.

[6] Kawamura, T. and T. Kanamori, "Preservation of Stronger Equivalence in Unfold/Fold Logic Program Transformation", in Theoretical Computer Science Vol.75, Nos. 1/2, pp.139-156, 1990.

[7] Lloyd, J. W., "Foundations of Logic Programming", Springer-Verlag, 2nd Edition, Berlin, Heidelberg, New York, 1987.

[8] Lloyd, J. W. and J. C. Shepherdson,"Partial Evaluation in Logic Programming", Technical Report CS-87-09, Department of Computer Science, University of Bristol, 1987.

[9] Owen, S., "Issues in the partial evaluation of meta-interpreters", in Meta-Programming in Logic Programming, eds. H. D. Abramson and M. H. Rogers, MIT Press, pp.319-340, 1989.

[10] Plotkin, G.D., "A note on inductive generalization", in Machine Intelligence 5, eds. B. Meltzer and D. Michie, Elsevier North-Holland, pp.153-163, New York, 1970.

[11] Pettorossi, A. and M. Proietti, "Decidability Results and Characterization of Strategies for the Development of Logic Programs", Proc. of 6th International Conference on Logic Programming, pp.539-553, Lisboa, 1989.

[12] Proietti, M. and A. Pettorossi, "Construction of Efficient Logic Programs by Loop Absorption and Generalization", Proc. of the Second Workshop on Meta-programming in Logic, pp.57-81, Leuven, 1990.

[13] Proietti, M. and A. Pettorossi, "Synthesis of Eureka Predicates for Developing Logic Programs", Proc. of 3rd European Symposium on Programming, Copenhagen, LNCS 432, Springer-Verlag, pp.307-325,1990.

[14] Seki, H. and Furukawa, K, "Notes on Transformation Techniques for Generate and Test Logic Programs", 4th IEEE Symp. on Logic Programming, San Francisco, pp.215-223.

[15] Seki, H., "Unfold/Fold Transformation of Stratified Programs", Proc. of 6th International Conference on Logic Programming, pp.554-568, Lisboa, 1989.

[16] Takeuchi, A. and H. Fujita, "Competitive Partial Evaluation - Some Remaining Problems of Partial Evaluation", An International Journal on New Generation Computing, Vol.6, Nos. 2,3, pp.259-278, 1988.

[17] Tamaki, H. and T. Sato, "Unfold/Fold Transformation of Logic Programs", Proc. of 2nd International Logic Programming Conference, pp.127-138, Uppsala, 1984.