ICOT Technical Report: TR-094

TR-094

Formulation of Induction Formulas in Verification of Prolog Programs

Tadashi Kanamori and Hiroshi Fujita
(Mitsubishi Electric Corp.)

December, 1984

ICOT

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

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Formulation of Induction Formulas in Verification of Prolog Programs

Tadashi KANAMORI,Hiroshi FUJITA

Mitsubishi Electric Corporation
Central Research Laboratory
Tsukaguchi-Honmachi 8-1-1
Amagasaki,Hyogo,JAPAN 661

Abstract

In this paper we describe how induction formulas are formulated in verification of Prolog programs. The same problem for well-founded induction was investigated by Boyer and Moore in their verification system for terminating LISP programs (BMTP). The least fixpoint semantics of Prolog provides an advantage that computational induction is easily applicable and generates simpler induction schemes. We investigate how the computational induction is applied to a class of first order formulas called S-formulas, in which specifications are described in our verification system. In addition,we show how an equivalence preserving transformation can be utilized to merge induction schemes into simpler one when more than two induction schemes are suggested.

Keywords : Program Verification,Induction,Prolog,Program Transformation.

Contents

1. Introduction

The intimacy of Prolog to first order logic is expected to bring advantages to first order inference in verification ([11],[17]). But how about induction, which plays an important role in verification of programs? The least fixpoint semantics of Prolog suggests fixpoint inductions. But how can we find such induction schemes from theorems to be proved? In addition,when more than two different induction schemes are suggested, how can we manage them?

In this paper we answer these questions. So far the same problem was investigated by Boyer and Moore [3] in their verification system for terminating LISP programs (BMTP) where they adopted quantifier-free formulas as specifications and applied well-founded induction. The least fixpoint semantics of Prolog provides an advantage that computational induction is applicable staying within first order logic and moreover it generates simpler induction schemes [5]. Here we investigate how the computational induction is applied to a class of first order formulas called S-formulas, in which specifications are described in our verification system. In addition,we show how the Tamaki-Sato's Prolog program transformation can be utilized to merge computational induction schemes into more simpler one when more then two induction schemes are suggested. Because the Tamaki-Sato's transformation preserves equivelence, justification of the manipulation of induction schemes is immediate and easy to grasp the meaning.

After summarizing preliminary materials in section 2 and a framework of verification in section 3, we describe in section 4 how computational induction schemes are found from theorems to be proved. In section 5, we show how induction schemes are merged into a more appropriate one when more than two induction schemes are suggested. Lastly in section 6 we discuss the relations to other works and our actual verification system.

2. Preliminaries

In the followings, we assume familiarity with the basic terminologies of first order logic such as term,atom(atomic formula), formula,substitution, most general unifier (mgu) and so on. We also assume knowledge about semantics of Prolog such as completion P^*, minimum Herbrand model M_0 and transformation T of Herbrand interpretations (see [1],[5],[7],[10]). We follow the syntax of DEC-10 Prolog [15]. Variables appearing in the head of a definite clause are called *head variables*. Other variables are called *internal variables*. As syntactical variables, we use X, Y, Z for variables, s, t for terms, A, B for atoms and $\mathcal{F}, \mathcal{G}, \mathcal{H}$ for formulas possibly with primes and subscripts. In addition,we use σ, τ, μ, ν for substitutions and $\mathcal{F}_{\mathcal{G}_1,\mathcal{G}_2,\ldots,\mathcal{G}_n}[\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n]$ for a replacement of an occurence of $\mathcal{G}_i$ in $\mathcal{F}$ with $\mathcal{H}_i$ for all $1 \leq i \leq n$. An atom $p(X_1, X_2, \ldots, X_n)$ is said to be *in general form* when $X_1, X_2, \ldots, X_n$ are distinct variables.

2.1. Polarity of Subformulas

We generalize the distinctions of positive and negative subformulas. The *positive* and *negative subformula* of a formula $\mathcal{F}$ is defined as follows (Murray [14]).

(a) $\mathcal{F}$ is a positive subformula of $\mathcal{F}$.

(b) When $\neg\mathcal{G}$ is a positive (negative) subformula of $\mathcal{F}$, then $\mathcal{G}$ is a negative (positive) subformula of $\mathcal{F}$.

(c) When $\mathcal{G} \wedge \mathcal{H}$ or $\mathcal{G} \vee \mathcal{H}$ is a positive (negative) subformula of $\mathcal{F}$, then $\mathcal{G}$ and $\mathcal{H}$ are positive (negative) subformulas of $\mathcal{F}$.

(d) When $\mathcal{G} \supset \mathcal{H}$ is a positive (negative) subformula of $\mathcal{F}$, then $\mathcal{G}$ is a negative (positive) subformula of $\mathcal{F}$ and $\mathcal{H}$ is a positive (negative) subformula of $\mathcal{F}$.
(e) When $\forall X \mathcal{G}, \exists X \mathcal{G}$ are positive (negative) subformulas of $\mathcal{F}$, then $\mathcal{G}_X(t)$ is a positive (negative) subformula of $\mathcal{F}$.

Example 2.1: Let $\mathcal{F}$ be
∀ A,B (reverse(A,B) ⊃reverse(B,A)).
Then $reverse(A, B)$ is a negative subformula of $\mathcal{F}$.

2.2. S-formulas and Goal Formulas

Let $\mathcal{F}$ be a closed first order formula. When $\forall X \mathcal{G}$ is a positive subformula or $\exists X \mathcal{G}$ is a negative subformula of $\mathcal{F}$, X is called a *free variable* of $\mathcal{F}$. When $\forall Y \mathcal{H}$ is a negative subformula or $\exists Y \mathcal{H}$ is a positive subformula of $\mathcal{F}$, Y is called an *undecided variable* of $\mathcal{F}$. In other words, free variables are variables quantified universally and undecided variables are those quantified existentially when $\mathcal{F}$ is converted to prenex normal form.

Example 2.2.1: Let $\mathcal{F}$ be
∀ A,V,C (append(A,[V],C) ⊃∃B reverse(C,[V|B])).
Then A, V, C are all free variables, while B is an undecided variable.

A closed first order formula S is called an *S-formula* when
(a) no free variable in S is quantified in the scope of quantification of an undecided variable in S and
(b) no undecided variable appears in negative atoms of S.

In other words, S-formulas are formulas convertible to prenex normal form $\forall X_1, X_2, \ldots, X_n \exists Y_1, Y_2, \ldots, Y_m \mathcal{F}$ and no $Y_1, Y_2, \ldots, Y_m$ appears in negative atoms of $\mathcal{F}$. Note that S-formulas include both universal formulas $\forall X_1, X_2, \ldots, X_n \mathcal{F}$ and usual execution goals $\exists Y_1, Y_2, \ldots, Y_m (A_1 \wedge A_2 \wedge \cdots \wedge A_k)$.

Example 2.2.2: Let S be
∀ A,V,C (append(A,[V],C) ⊃∃B reverse(C,[V|B])).
Then S is an S-formula, because free variables A, V, C are quantified outside $\exists B$ and B appears only in the positive atom $reverse(C, [V|B])$. A universal formula $\forall A, B(reverse(A, B) \supset reverse(B, A))$ and an execution goal $\exists C\, append([1, 2], [3], C)$ are also S-formulas.

A formula G obtained from an S-formula S by replacing free variable X with X, undecided variable Y with $?Y$ and deleting all quantifications is called a *goal formula* of S. Note that S can be uniquely restorable from G. In the followings, we use goal formulas instead of original S-formulas with explicit quantifiers. Goal formulas are denoted by F, G, H.

Example 2.2.3: An S-formula
∀ A,V,C (append(A,[V],C) ⊃∃B reverse(C,[V|B])).
is represented by a goal formula
append(A,[V],C)⊃reverse(C,[V|?B]).
A universal formula $\forall A, B(reverse(A, B) \supset reverse(B, A))$ and an execution goal $\exists C\, append([1, 2], [3], C)$ are represented by $reverse(A, B) \supset reverse(B, A)$ and $append([1, 2], [3], ?C)$ respectively.

2.3. Manipulation of Goal Formulas

Lastly we introduce two manipulations of goal formulas. One is an application of a class

of substitutions. To avoid variable name's conflict, we introduce a class of substitutions. A substitution σ is called a *substitution away from* A when σ instanciates each free variable X in A to t such that every variable in t is a fresh free variable not in A.

Example 2.3.1: $< A \Leftarrow [X|L], B \Leftarrow M >$ is a substitution away from $reverse(A, B)$ and unifies it with $reverse([X|L], M)$, where X, L and M are considered fresh free variables. $< C \Leftarrow [X|L], U \Leftarrow Y, B \Leftarrow [X|N] >$ is a substitution away from $append(C, [U], B)$ and unifies it with $append([X|L], [Y], [X|N])$ where X, L, Y and N are considered fresh free variables.

Another manipulation is a *reduction* of goal formulas with logical constants $true$ and $false$. The *reduced form* of a goal formula G,denoted by $G\downarrow$, is the normal form in the reduction system defined as follows.

$$\begin{array}{ll} \neg true \rightarrow false, & \neg false \rightarrow true, \\ true \wedge G \rightarrow G, & false \wedge G \rightarrow false, \\ G \wedge true \rightarrow G, & G \wedge false \rightarrow false, \\ true \vee G \rightarrow true, & false \vee G \rightarrow G, \\ G \vee true \rightarrow true, & G \vee false \rightarrow G, \\ true \supset G \rightarrow G, & false \supset G \rightarrow true, \\ G \supset true \rightarrow true, & G \supset false \rightarrow \neg G. \end{array}$$

Example 2.3.2: Let G_1 and G_2 be $false \supset reverse(B, A)$ and $true \supset reverse(B, A)$. Then $G_1\downarrow$ is $true$ and $G_2\downarrow$ is $reverse(B, A)$.

3. Framework of Verification of Prolog Programs

3.1. Programming Language

We introduce **type** construct into Prolog to separate definite clauses defining data structures from others defining procedures,e.g.,

```
type.
    list([ ]).
    list([X|L]) :- list(L).
end.
```

The body of **type** is a conjunction of definite clauses whose head is with a unary predicate defining a data structure. (Type predicates in our verification system are being corresponded to *shells* in BMTP.) Procedures are defined by following the syntax of DEC-10 Prolog [15],e.g.,

```
append([ ],K,K).
append([X|L],M,[X|N]) :- append(L,M,N).
reverse([ ],[ ]).
reverse([X|L],M) :- reverse(L,N),append(N,[X],M).
```

Throughout this paper, we study pure Prolog consisting of definite clauses and consider a finite set of definite clauses P as their conjunction. We assume variables in each definite clause are renamed at each use so that there occurs no variable names conflict.

3.2. Specification Language

The main construct of our specification language is **theorem** to state a theorem to be proved,e.g.,

theorem(halting-theorem-for-append).
∀ A:list,B ∃ C append(A,B,C).
end.

The body of **theorem** must be a closed S-formula. Any variable X in quantification may be followed by a type qualifier : p (e.g.,:list above). $\forall X : p\mathcal{F}$ and $\exists X : p\mathcal{F}$ are abreviations of $\forall X(p(X) \supset \mathcal{F})$ and $\exists X(\mathcal{F} \wedge p(X))$ respectively.

3.3. Framework of Verification

Let S be a specification in an S-formula, M_0 be the minimum Herbrand model of P and P^* be the completion of P. We adopt a formulation that verification of S with respect to P is to show $M_0 \models S$ when model theoretically speaking and to prove S from P^* using the first order inference and some induction when proof theoretically speaking.

The most important difference between our system and BMTP is that specifications in BMTP are quantifier-free (i.e. universal) formulas while ours are general first order formulas. Though we prove quantifier-free specifications of the form $\forall X_1, X_2, \ldots, X_n\ (A_1 \wedge A_2 \wedge \cdots \wedge A_m \supset A_0)$ in most cases, the consideration of existential quantifiers is inevitable because of the effects of internal variables in Prolog. For example, suppose we prove $\forall X, Y\,(\mathit{condition}(X, Y) \supset p(X, Y))$ with respect to a program $p(X, Y) :- q(X, Z), r(Z, Y)$. Then we must prove $\forall X, Y\,(\mathit{condition}(X, Y) \supset \exists Z(q(X, Z) \wedge r(Z, Y))$ substantially.

4. Generation of Computational Induction Schemes

4.1. Computational Induction

For pure Prolog, the transformation T of Herbrand interpretations is always continuous and there holds $\bigcup_{i=0}^{\infty} T^i(\emptyset) = M_0$. This suggests use of fixpoint induction to prove $M_0 \models \psi$. Here we explain it rather intuitively following Clark [5] p.75-76.

Example 4.1.1: Let *reverse* be a relation defined by the previous definite clauses. The *reverse* relation is the smallest set of pairs of terms that includes a pair ([ ], [ ]) and that,for any term s,includes $([s|t_1], t_2)$ whenever it includes (t_1, t) and $(t, [s], t_2)$ is in the *append* relation. Hence,suppose $Q(A, B)$ is a formula with free variable A, B. For any Herbrand interpretation, $Q(A, B)$ will denote some binary relation over terms. If this relation includes ([ ], [ ]),i.e.

Q([ ],[ ])

is true,and if it includes $([s|t_1], t_2)$ whenever it includes (t_1, t) and $(t, [s], t_2)$ is in *append* relation, i.e.,

∀ A,B,C,U (Q(A,C)∧append(C,[U],B)⊃Q([U|A],B))

is true,then the relation $Q(A, B)$ includes all the pairs of terms in the *reverse* relation.In other words,

∀ A,B (reverse(A,B)⊃Q(A,B))

is true of the *reverse* relation and such $Q(A, B)$.Hence we get the following computational induction scheme

$$\frac{\mathrm{Q}([\,],[\,]) \qquad \forall \mathrm{A,B,C,U}\ (\mathrm{Q(A,C)} \wedge \mathrm{append(C,[U],B)} \supset \mathrm{Q([U|A],B)})}{\forall \mathrm{A,B}\ (\mathrm{reverse(A,B)} \supset \mathrm{Q(A,B)})}$$

This is a Prolog version of the de Bakker and Scott's computational induction [2]. In order to apply such an induction, we need to know (a) how we select the key atom ($reverse(A, B)$ above), (b) what conditions we have to observe for soundness and (c) how we can make the generated induction schemes simple.

First of all, we introduce a concept "dominant atom". An atom A in a goal formula G is said to be *dominant* in G when G is equivalent to a goal formula of the form $A \supset H$. Note that A is dominant in G iff A is negative and $G_A[false] \downarrow$ is $true$. Then H is $G_A[true] \downarrow$.

Example 4.1.2: Let G be $reverse(A, B) \supset reverse(B, A)$. Then the left $reverse(A, B)$ is dominant in G, but the right $reverse(B, A)$ is not.

4.2. Inducible Definite Clause

A definite clause is said to be *inducible* to A when,for any ground instance of the definite clause such that the head is a ground instance of A, any recursive call in the body is also a ground instance of A. (Hence any non-recursive definite clause is always inducible. Any definite clause with a head nonunifiable with A is also inducible.)

When every definite clause in P is inducible to A, A is said to be *closed with respect to* P. This means that the set of ground atoms in M_0 of the form of instance of A is computable by the instances of inducible definite clauses. Note that an atom $p(X_1, X_2, \ldots, X_n)$ in general form is always closed.

Example 4.2.1: Let the atom A be $append(A, [U], C)$. Then A is closed. This means that $\{append(t_1, [t_2], t_3) \mid t_1, t_2$ and t_3 are ground terms$\} \bigcap M_0$ is computable by some instances of definite clauses,i.e.,

```
append([ ],[Y],[Y]).
append([X|L],[Y],[X|N]) :- append(L,[Y],N).
```

The inducibility can be checked as follows.

(a) Check whether the head B_0 is unifiable with A by a substitution for A away from A (see 2.3). If it is, decompose the mgu to $\sigma \circ \tau_0$ where σ is the restriction to variables in B_0 and τ_0 is the restriction to variables in A. If it is not,the definite clause is inducible to A.

(b) Check whether each instance of the recursive call in the body $\sigma(B_i)$ is an instance of A and if it is,compute the instanciation τ_i. If it is not,the definite clause is not inducible to A.

The set of all instances of definite clauses by σ is called *instanciated program* for A.

Example 4.2.2: Let the atom A be $append(A, [U], C)$. Then the first head $append([ ], L, L)$ is unifiable with $append(A, [U], C)$ by $< L \Leftarrow [Y] > \circ < A \Leftarrow [ ], U \Leftarrow Y, C \Leftarrow [Y] >$. The second head $append([X|L], M, [X|N])$ is unifiable with $append(A, [U], C)$ by $< M \Leftarrow [Y] > \circ < A \Leftarrow [X|L], U \Leftarrow Y, C \Leftarrow [X|N]$ and the instance $append(L, [Y], N)$ in the body is also an instance of $append(A, [U], C)$ by $< A \Leftarrow L, U \Leftarrow Y, C \Leftarrow N >$.

4.3. Generation of Induction Schemes

Let G be a goal formula, $A \Rightarrow p(t_1, t_2, \ldots, t_n)$ be a dominant atom in G and Q be $G_A[true] \downarrow$. Let "B_0 :- $B_1, B_2, \ldots, B_m$" be a definite clause inducible to A. By $\sigma(B_0$:-

$B_1, B_2, \ldots, B_m)_p(Q)$ we denote a formula obtained by replacing $\sigma(B_i) = p(s_1, s_2, \ldots, s_n)$ in $\sigma(B_1 \wedge B_2 \wedge \cdots \wedge B_m \supset B_0)$ with $\tau_i(Q)$ when B_0 is unifiable with A and *true* otherwise.

We generate an induction scheme as follows. All free variables are quantified universally at the outermost, which keeps the generated subgoals within S-formulas.

$$\frac{\sigma_1(\mathrm{B_{10}} \text{ :- } \mathrm{B_{11}},\mathrm{B_{12}},\ldots,\mathrm{B_{1m_1}})_p(\mathrm{Q}),\ \sigma_2(\mathrm{B_{20}} \text{ :- } \mathrm{B_{21}},\mathrm{B_{22}},\ldots,\mathrm{B_{2m_2}})_p(\mathrm{Q}),\ \ldots,\ \sigma_k(\mathrm{B_{k0}} \text{ :- } \mathrm{B_{k1}},\mathrm{B_{k2}},\ldots,\mathrm{B_{km_k}})_p(\mathrm{Q})}{G}$$

Example 4.3.1: Suppose we prove a theorem

theorem(reverse-reverse).
∀A,B (reverse(A,B) ⊃reverse(B,A)).
end.

Then $reverse(A, B)$ is dominant. Let $Q(A, B)$ be $(true \supset reverse(B, A)) \downarrow\!= reverse(B, A)$. The induction scheme generated is the one in example 4.1,i.e.,

$$\frac{\text{reverse([ ],[ ])} \qquad \forall \text{A,B,C,U (reverse(C,A) } \wedge \text{append(C,[U],B) } \supset \text{reverse(B,[U|A]))}}{\forall \text{A,B (reverse(A,B) } \supset \text{reverse(B,A))}}$$

4.4. Generalization

Must we always give up applications of computational induction when some definite clause is not inducible?

Example 4.4.1: Let A be $reverse(A, [V|B])$. Then "$reverse([X|L], M)$:- $reverse(L, N)$, $append(N, [X], M)$" is not inducible to A, because the instance of the recursive call by σ in the definition above may have the form $reverse(L, N)$ which is not necessarily an instance of $reverse(A, [V|B])$.

A set of replacement $\mathcal{M}$ of occurrences of subterm in A with fresh free variables is called a *generalization mask* when no different subterm is replaced with a same fresh free variable. An atom obtained from A by replacing the occurrence of t_{i_k} with a fresh free variable X_{i_k} for all $< t_{i_k} \Leftarrow X_{i_k} > \in \mathcal{M}$ is called *generalization* of A with respect to $\mathcal{M}$ and denoted by $A_{\mathcal{M}}$. Then $X_{i_1} = t_{i_1} \wedge X_{i_2} = t_{i_2} \wedge \cdots \wedge X_{i_k} = t_{i_m}$ is called the *generalization equation*. (When $\mathcal{M}=\emptyset$, $A_\emptyset$ is A itself and the generalization equation is *true*.)

Example 4.4.2: $\mathcal{M}$={ $[V|B] \Leftarrow B'$ } is a generalization mask of $reverse(A, [V|B])$ in
∀ A,V,B (reverse(A,[V|B]) ⊃member(V,A)).
The generalization of $reverse(A, [V|B])$ w.r.t. $\mathcal{M}$ is $reverse(A, B')$ where B' is a fresh free variable. The generalization equation is $B' = [V|B]$. Intuitively this corresponds to modify the theorem to
∀ A,V,B,B' (reverse(A,B')∧B'=[V|B] ⊃member(V,A)).

A definite clause B_0 :- $B_1, B_2, \ldots, B_m$ is said to be *inducible with* $\mathcal{M}$ *to* A when it is inducible to $A_{\mathcal{M}}$.

Example 4.4.3: Let A be $reverse(A, [V|B])$ and $\mathcal{M}$ be { $[V|B] \Leftarrow B'$ }. Then "$reverse([X|L], M)$:- $reverse(L, N)$, $append(N, [X], M)$" is inducible with $\mathcal{M}$ to A,because the head $reverse([X|L], M)$ is unifiable with $reverse(A, B')$ by $< > \circ < A \Leftarrow [X|L], B' \Leftarrow M >$ and the recursive call $reverse(L, N)$ in the body is also an instance of $reverse(A, B')$.

One naturally expect that $A_{\mathcal{M}}$ be the most specific one. An atom $\overline{A}$ is said to be a

closure of A with respect to P when

(a) $\overline{A}$ is closed with respect to P,
(b) A is an instance of $\overline{A}$ and
(c) $\overline{A}$ is an instance of any $\overline{A}'$ satisfying (a) and (b).

The closure is unique up to renaming and A is closed iff $A = \overline{A}$ modulo renaming. (See appendix for the proof of uniqueness and algorithm to compute the closure.) A generalization mask $\mathcal{M}$ corresponding to $\overline{A}$ is called a *computational induction mask*. Examples in 4.1—3 are cases with the mask $\emptyset$.

Example 4.4.4: Let A be $reverse(A, [V|B])$ and $\mathcal{M}$ be $\{[V|B] \Leftarrow B'\}$. Then the first head $reverse([\,], [\,])$ is unifiable with $reverse(A, B')$ by $< > \circ < A \Leftarrow [\,], B' \Leftarrow [\,] >$. The second definite clause is inducible with $\mathcal{M}$ to A. Moreover $reverse(A, B')$ is a closure of $reverse(A, [V|B])$ and $\mathcal{M}$ is a computational induction mask.

Then computational induction schemes are generated as are in 4.3 by replacing A with $A_{\mathcal{M}}$ and Q with $G_{\mathcal{A}}[e] \downarrow$ where e is the generalization equation.

Example 4.4.5: Let *member* be defined by the following programs.

```
member(X,[X|L]).
member(X,[Y|L]) :- member(X,L).
```

Suppose we prove

```
theorem(last-is-a-member).
   ∀A,V,B (reverse(A,[V|B]) ⊃memeber(V,A)).
end.
```

Note the dominant atom $reverse(A, [V|B])$. Because $\emptyset$ is not a computational induction mask, we can't apply the induction immediately without generalization. We need a non-empty mask $\mathcal{M}=\{\ [V|B] \Leftarrow B'\ \}$ and generated formulas are

∀ V,B ([ ]=[V|B] ⊃member(V,[ ])),
∀ U,V,A,B,C,D ((C=[V|B] ⊃member(V,A)) ∧append(C,[U],D) ⊃(D=[V|B] ⊃member(V,[U|A]))).

Note that the second subgoal remains within S-formulas. The first goal is reduced to *true* by the definition $Z = Z$. The second goal is reduced to

∀ U,V,A,B,C ((C=[V|B] ⊃member(V,A)) ∧append(C,[U],[V|B]) ⊃member(V,[U|A])).

and it is proved using the definition of *append*.

4.5. Examples of Induction Schemes

We show how computational inductions are applied by several examples.

Example 4.5.1: We need not to impose the termination condition, i.e. $p(t_1, t_2, \ldots, t_n)$ may be neither *true* nor *false* for some ground terms $t_1, t_2, \ldots, t_n$. This means we can separate M_0 from other models of P^* and show properties specific to M_0. Let p and q be defined by the following programs (Apt and van Emden [1]).

```
p(a) :- p(X),q(X).
p(s(X)) :- p(X).
q(b).
q(s(X)) :- q(X).
```

The execution of $? - p(s^1(a))$ never terminates, while those of $? - p(s^2(b))$ and $? - q(s^1(a))$ fail finitely. The minimum (and the maximum) Herbrand model is $M_0 = \{q(b), q(s(b)), q(s(s(b))), \ldots\}$. But we can't infer $\neg p(s^1(a))$ from the completion P^*. This is showen by existence of a non-

Herbrand model M. Let the domain of M be a set consisting of red nutural numbers,green natural numbers and blue integers and the interpretation of symbols on M be one interpreting a as red 0, b as green 0, s be a function mapping X to $X+1$ in the usual arithmetic, $p(i)$ is *true* iff i is either red natural number or blue integer and $q(j)$ is *true* iff j is either green natural number or blue integer. Then M is a model of P^* not isomorphic to M_0 and $p(s^i(a))$ is valid in M (cf.Jaffar et al [10]). Suppose we prove a theorem

theorem(p-does-not-hold).
∀A ¬p(A).
end.

which is valid in M_0 but not valid in M, hence not provable by first order inference from P^*. Note the dominant atom $p(A)$. Because each definite clause is inducible with $\emptyset$, we can apply the induction immediately and two generated formulas are

∀ A (false ∧q(A) ⊃false)
false ⊃false.

Both of them are trivially *true*.

Example 4.5.2: When a theorem includes a dominant atom with a type predicate, we have the effect to perform the usual structural inuction. Suppose we prove (cf.Kowalski [13] pp.221-222)

theorem(right-identity-of-append).
∀ A:list append(A,[ ],A).
end.

Note the dominant atom $list(A)$. Because each definite clause is inducible with $\emptyset$, we can apply the induction immediately and two generated formulas are

append([ ],[ ],[ ]),
∀ U,A (append(A,[ ],A) ⊃append([U|A],[ ],[U|A])).

Example 4.5.3: Theorems may include existentially quantified variables. Suppose we prove

theorem(last-can-be-first-of-reversed).
∀ A,V,C (append(A,[V],C) ⊃∃B reverse(C,[V|B])).
end.

Note the dominant atom $append(A,[V],C)$. Because each definite clause is inducible with $\emptyset$, we can apply the induction immediately and two generated formulas are

∀ V ∃ B reverse([V],[V|B]),
∀ U,A,V,C (∃ B reverse(C,[V|B]) ⊃∃ B reverse([U|C],[V|B])).

Note that the generated subgoals still remain within S-formulas.

5. Manipulation of Computational Induction Schemes

5.1. Mergible Schemes

In section 4,we cited examples with only one dominant atom intensionally. But sometimes more than two different induction schemes are suggested from theorems to be proved.

Example 5.1: Suppose we prove the second formula generated from *reverse-reverse*

theorem(first-last).
∀A,B,C,U (reverse(C,A) ∧append(C,[U],B) ⊃reverse(B,[U|A])).
end.

Then both $reverse(C,A)$ and $append(C,[U],B)$ are dominant. Let Q_1 and Q_2 be

Q_1(A,B,C,U) : append(C,[U],B) ⊃reverse(B,[U|A]),
Q_2(A,B,C,U) : reverse(C,A) ⊃reverse(B,[U|A]).

Then $reverse(C, A)$ and $append(C, [U], B)$ suggests two different induction schemes.

$$\frac{\forall B,U\ Q_1([\,],B,[\,],U) \quad \forall A,B,C,U,A_1,V,\ (Q_1(A_1,B,C,U) \wedge \mathrm{append}(A_1,[V],A) \supset Q_1(A,B,[V|C],U))}{\forall A,B,C,U\ (\mathrm{reverse}(C,A) \wedge \mathrm{append}(C,[U],B) \supset \mathrm{reverse}(B,[U|A]))}$$

$$\frac{\forall A,U\ Q_2(A,[U],[\,],U) \quad \forall A,B,C,U,V\ (Q_2(A,B,C,U) \supset Q_2(A,[V|B],[V|C],U))}{\forall A,B,C,U\ (\mathrm{reverse}(C,A) \wedge \mathrm{append}(C,[U],B) \supset \mathrm{reverse}(B,[U|A]))}$$

In the example above, the substitution to N in each scheme coincides and it suggests a possibility to apply both inductions simultaneously. In order to justify such a manipulation, we describe an equivalence preserving transformation in the next section.

5.2. Tamaki-Sato's Transformation

The Tamaki-Sato's transformation system is developed for Prolog programs based on unfold/fold transformations. The entire process proceeds as follows [16].

Transformation Process

P_0 :=the initial program ; $D_0 := \{\}$;
mark every clause in P_0 "foldable";
for $i := 1$ **to** arbitrary N
  apply any of the transformation rules to obtain P_i and D_i from P_{i-1} and D_{i-1};

Example 5.2.1: Before starting the transformation,the initial program is given,e.g.,

```
P0 : C1. append([ ],L,L).
     C2. append([X|L],M,[X|N]) :- append(L,M,N).
     C3. reverse([ ],[ ]).
     C4. reverse([X|L],M) :- reverse(L,N),append(N,[X],M).
```

and D_0 is initialized to $\{\}$.

The basic part of Tamaki-Sato's transformation system consists of three transformation rules, i.e.,definition,unfolding and folding.

Definition : Let C be a clause of the form $p(X_1, X_2, \ldots, X_n) :- A_1, A_2, \ldots, A_m$ where
(a) p is an arbitrary predicate appearing neither in P_{i-1} nor in D_{i-1},
(b) $X_1, X_2, \ldots, X_n$ are distinct variables,and
(c) $A_1, A_2, \ldots, A_m$ are atoms whose predicates all appears in P_0.
Then let P_i be $P_{i-1} \bigcup \{C\}$ and D_i be $D_{i-1} \bigcup \{C\}$. Do not mark C "foldable".

Example 5.2.2: Suppose we need the conjunction of atoms each induction scheme is accounting for in example 5.1. Then we introduce it by the following definition.

```
C5. new-p(L,M,N,X) :- reverse(N,L),append(N,[X],M).
```

Then $P_1 = \{\underline{C_1}, \underline{C_2}, \underline{C_3}, \underline{C_4}, C_5\}$ and $D_1 = \{C_5\}$. The underlines indicate "foldable" clauses.

Unfolding : Let C be a clause in P_{i-1},A be an atom in its body and $C_1, C_2, \ldots, C_k$ be all the clauses in P_{i-1} whose heads are unifiable with A. Let C'_i be the result of resolving C with C_i on A. Then let P_i be $(P_{i-1} - \{C\}) \bigcup \{C'_1, C'_2, \ldots, C'_k\}$ and D_i be D_{i-1}. Mark each C'_i "foldable" unless it is already in P_{i-1}.

Example 5.2.3 : When C_5 is unfolded at its first atom in the body, we obtain P_2=$\{\underline{C_1}, \underline{C_2}, \underline{C_3}, \underline{C_4}, \underline{C_6}, \underline{C_7}\}$ and D_2=$\{C_5\}$ where

C_6. new-p([ ],M,[ ],X) :- append([ ],[X],M).

C_7. new-p(L,M,[Y|N],X) :- reverse(N,L_1),append(L_1,[Y],L),append([Y|N],[X],M).

C_6 and C_7 are still unfoldable into

C'_6. new-p([ ],[X],[ ],X).

C'_7. new-p(L,[Y|M],[Y|N],X) :- reverse(N,L_1),append(L_1,[Y],L),append(N,[X],M).

and we get P_3=$\{\underline{C_1}, \underline{C_2}, \underline{C_3}, \underline{C_4}, \underline{C'_6}, \underline{C'_7}\}$ and D_3=$\{C_5\}$.

Folding : Let C be a clause in P_{i-1} of the form $A_0 :- A_1, A_2, \ldots, A_n$ and C_{folder} be a clause in D_{i-1} of the form $B_0 :- B_1, B_2, \ldots, B_m$. Suppose there is a substitution σ and a subset $\{A_{i_1}, A_{i_2}, \ldots, A_{i_m}\}$ of the body of C such that the following conditions hold.

(a) $A_{i_j} = \sigma(B_j)$ for $j = 1, 2, \ldots, m$,

(b) σ substitutes distinct variables for the internal variables of C_{folder} and moreover those variables occur neither in A nor in $\{A_1, A_2, \ldots, A_n\} - \{A_{i_1}, A_{i_2}, \ldots, A_{i_m}\}$,and

(c) C is marked "foldable" or $m < n$.

Then let P_i be $(P_{i-1} - \{C\}) \bigcup \{C'\}$ and D_i be D_{i-1} where C' is a clause with head A and body $(\{A_1, A_2, \ldots, A_n\} - \{A_{i_1}, A_{i_2}, \ldots, A_{i_m}\}) \bigcup \{\sigma(B)\}$. Let C inherit the mark of C.

Example 5.2.4 : By folding the whole body of C_7 by C_5, we obtain P_4=$\{\underline{C_1}, \underline{C_2}, \underline{C_3}, \underline{C_4}, \underline{C_6}, \underline{C_8}\}$ and D_4=$\{C_5\}$ where

C_8. new-p(L,[Y|M],[Y|N],X) :- new-p(L_1,M,N,X),append(L_1,[Y],L).

The most important property of this transformation system is stated as follows [16].

Equivalence Preservation Theorem

P_N is equivalent to $P_0 \bigcup D_N$ in the minimum Herbrand model semantics,i.e., the minimum Herbrand models of them are identical.

Example 5.2.5: Through the previous transformation process, we reach a program

P_4 : C_1. append([ ],L,L).

C_2. append([X|L],M,[X|N]) :- append(L,M,N).

C_3. reverse([ ],[ ]).

C_4. reverse([X|L],M) :- reverse(L,N),append(N,[X],M).

C'_6. new-p([ ],[X],[ ],X).

C_8. new-p(L,[Y|M],[Y|N],X) :- new-p(L_1,M,N,X),append(L_1,[Y],L).

which is equivalent to

P_0 : C_1. append([ ],L,L).

C_2. append([X|L],M,[X|N]) :- append(L,M,N).

C_3. reverse([ ],[ ]).

C_4. reverse([X|L],M) :- reverse(L,N),append(N,[X],M).

D_4 : C_5. new-p(L,M,N,X) :- reverse(N,L),append(N,[X],M).

5.3. Derivation of Merged Schemes

We keep each induction scheme in a triple $(A, Q, \{\sigma_i(C_i)\})$, i.e.,an atom A accounting for, a goal formula Q and an instanciated program $\{\sigma_i(C_i)\}$.

Example 5.3.1: Suppose we are trying to prove the theorem *first-last*. Before merging, we have two induction schemes I and J whose instanciated programs are

I : reverse([ ],[ ]).

reverse([X|L],M) :- reverse(L,N),append(N,[X],M).

J : append([ ],[X],[X]).
append([X|L],[Y],[X|M]) :- append(L,[Y],M).

An induction scheme J accounting for $q(s_1, s_2, \ldots, s_m)$ is said to be *mergible to* an induction scheme I accounting for $p(t_1, t_2, \ldots, t_n)$ (cf. [3] pp.191-194) when

(a) $p(t_1, t_2, \ldots, t_n)$ and $q(s_1, s_2, \ldots, s_m)$ share at least one free variable,
(b) Every recursive definite clause of I is mergible to only one definite clause of J and
(c) No two recursive definite clauses of I is mergible to a same definite clause of J.

where a recursive definite clause $A_0 :- A_1, A_2, \ldots, A_\alpha$ is said to be *mergible* to a recursive definite clause $B_0 :- B_1, B_2, \ldots, B_\beta$ when the following conditions are satisfied. Let A_0 be $\tau_0(p(t_1, t_2, \ldots, t_n))$ and B_0 be $= \tau'_0(q(s_1, s_2, \ldots, s_m))$.

(a) τ_0 and τ'_0 substitute an identical term to each common variable of $p(t_1, t_2, \ldots, t_n)$ and $q(s_1, s_2, \ldots, s_m)$,
(b) There is a one-to-one correspondence between recursive calls in the bodies such that the corresponding τ_i and τ'_j substitute an identical term to each common variable of $p(t_1, t_2, \ldots, t_n)$ and $q(s_1, s_2, \ldots, s_m)$.

Example 5.3.2: The previous two induction schemes are ovbiously mergible, because for the second clauses,

(a) When $reverse(C, A)$ is unified with $reverse([X|L], M)$ by $\tau_0 = < C \Leftarrow [X|L], A \Leftarrow M >$, $append(C, [U], B)$ is unifiable with $append([X|L], [Y], [X|N])$ by $\tau'_0 = < C \Leftarrow [X|L], B \Leftarrow [X|N], U \Leftarrow Y >$
(b) There is only one recursive calls $reverse(L, N)$ and $append(L, [Y], N)$ in each second definite clause and they are instances of $reverse(C, A)$ by $\tau_1 = < C \Leftarrow L, A \Leftarrow N >$ and of $append(C, [U], B)$ by $\tau'_1 = < C \Leftarrow L, B \Leftarrow N, U \Leftarrow Y >$.

The mergibility guarantees that a transformation sequence like one stated in example 5.2.1—5 is applicable. Note that the transformation is a routine and we can apply it mechnically. When two schemes are mergible, we derive a new scheme as follows.

(a) Define $new\text{-}p(X_1, X_2, \ldots, X_l)$ by the conjunction of $p(t_1, t_2, \ldots, t_n)$ and $q(s_1, s_2, \ldots, s_m)$.
(b) Unfold at $p(t_1, t_2, \ldots, t_n)$ and $q(s_1, s_2, \ldots, s_m)$ once in the bodies of definite clauses.
(c) Fold if possible.

Then the obtained definite clauses represent a new scheme with $new\text{-}p(X_1, X_2, \ldots, X_l)$ and $Q = G_{p(t_1, t_2, \ldots, t_n), q(s_1, s_2, \ldots, s_m)}[true, true] \downarrow$.

Example 5.3.3: As was shown in example 5.2.1—5, we can obtain a definite clause program for *new-p* from the previous two induction schemes

new-p([ ],[X],[ ],X).
new-p(L,[Y|M],[Y|N],X) :- new-p(L_1,M,N,X),append(L_1,[Y],L).

and the theorem to be proved is now

∀ A,B,C,U (new-p(A,B,C,U)⊃reverse(B,[U|A])).

Hence the new scheme accounting for *new-p* represents

$$\frac{\forall U\ reverse([U],[U]) \quad \forall A,B,C,U,A_1,V\ (reverse(B,[U|A_1]) \wedge append(A_1,[V],A) \supset reverse([V|B],[U|A]))}{\forall A,B,C,U\ (reverse(C,A) \wedge append(C,[U],B) \supset reverse(B,[U|A]))}$$

Merging of compuatational induction schemes sometimes results in surprisingly simple schemes.

Example 5.3.4: Suppose we prove a theorem

theorem(equivalence-of-flatten-and-mc-flatten).
∀W,A,B,C (flatten(W,A) ∧mc-flatten(W,B,C) ⊃append(A,B,C)).
end.

where *flatten* and *mc-flatten* are procedures collecting leaves of S-expressions defined as follows [3]. (*mc-flatten*$(W,[],A)$ performs the same task as *flatten*(W,A) more efficiently.)

```
flatten(X,[X]) :- atom(X).
flatten([X|Y],L) :- flatten(X,L1),flatten(Y,L2),append(L1,L2,L).
mc-flatten(X,M,[X|M]) :- atom(X).
mc-flatten([X|Y],M,N) :- mc-flatten(Y,M,L),mc-flatten(X,L,N).
```

Then by defining a new predicate by

```
new-p(Z,L,M,N) :- flatten(Z,L),mc-flatten(Z,M,N).
```

and applying the transformations,we obtain a new definition

```
new-p(X,[X],M,[X|M]) :- atom(X).
new-p([X|Y],L,M,N) :- append(L1,L2,L),new-p(X,L1,L3,N),new-p(Y,L2,M,L3).
```

The scheme accounting for *new-p*(W,A,B,C) is

$$\frac{\begin{array}{c}\forall U,B\ (\mathrm{atom}(U) \supset \mathrm{append}([U],B,[U|B])) \\ \forall A,B,C,A_1,A_2,A_3\ (\mathrm{append}(A_1,A_2,A) \wedge \mathrm{append}(A_1,A_3,C) \wedge \mathrm{append}(A_2,B,A_3) \supset \mathrm{append}(A,B,C))\end{array}}{\forall W,A,B,C\ (\mathrm{flatten}(W,A) \wedge \mathrm{mc\text{-}flatten}(W,B,C) \supset \mathrm{append}(A,B,C))}$$

The second formula is the associativity of *append* for A_1,A_2 and B.

6. Discussions

Computational induction is due to de Bakker and Scott [2]. Its use in mechanical verification is investigated by Weyrauch and Milner [18] and Gordon etc [8]. The simplest form of its use in Prolog was pointed out by Clark [5]. (Integration of induction into a logic programming system as an inference rule is investigated by Hagiya and Sakurai [9].) But our use is more general in the sense that atoms accounted for need not be in general form $p(X_1,X_2,\ldots,X_n)$,e.g. *Example 4.4.5* and *4.5.3*. Our method to merge computational induction schemes is new as far as we know.

In our verification system, if we can apply computational inductions,we prefer them. But when we can't because of the conditions and heuristic constraint, we resort to well-founded induction. Formulation of well-founded induction is done similarly with BMTP-like heuristics [12].

7. Conclusions

We have shown how to formulate computational induction formulas in verification of Prolog programs and its advantages. Compared to BMTP,use of computational induction (a) need not guarantee termination by any well-founded ordering, (b) does not require the restriction that procedures be terminating, (c) generate induction goals 1 less than BMTP-like well-founded inductions, (d) generate simpler induction formulas more first order inferences are already performed to and (e) accomodate naive structural inductions and well-founded inductions in many cases. This method to formulate induction formulas is an element of our verification system for Prolog programs under development.

Acknowledgements

The authors would like to express deep gratitude to Dr.T.Sato (Electrotechnical Laboratory) and Dr.H.Tamaki (Ibaraki University) for their stimiulative and perspicuous works.

Our verification system is a subproject of Fifth Generation Computer System(FGCS) "Intelligent Programming System". The authors would like to thank Dr.K.Fuchi (Director of ICOT) for the chance of this research and Dr.K.Furukawa(Chief of ICOT 2nd Laboratory) and Dr.T.Yokoi(Chief of ICOT 3rd Laboratory) for their advices and encouragements.

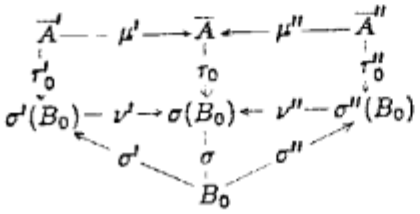
References

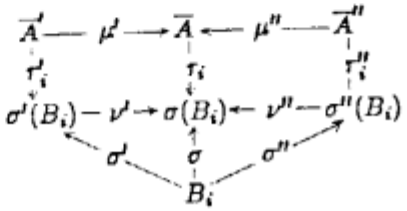
[1] Apt,K.R. and M.H.van Emden,"Contribution to the Theory of Logic Programming",J.ACM, Vol.29,No.3,pp 841-862,1982.
[2] de Bakker,J.W. and D.Scott,"A Theory of Programs",Unpublished Notes, IBM Seminar,Vienna,1969.
[3] Boyer,R.S. and J.S.Moore,"Computational Logic",Chap.14-15, Academic Press,1979.
[4] Burstall,R.,"Proving Properties of Programs by Structural Induction",Comput.J.,12,No.1.,pp.41-48,1969.
[5] Clark,K.L.,"Predicate Logic as a Computational Formalism", pp.75-76, Research Monograph : 79/59, TOC,Imperial College,1979.
[6] Clark,K.L. and S-Å.Tärnlund,"A First Order Theory of Data and Programs",in Information Processing 77 (B.Gilchrist Ed), pp 939-944,1977.
[7] van Emden,M.H. and R.A.Kowalski,"The Semantics of Predicate Logic as Programing Language",J.ACM,Vol.23,No.4,pp 733-742,1976.
[8] Gordon,M.J.,A.J.Milner and C.P.Wadsworth,"Edinburgh LCF — A Mechanized Logic of Computation",Lecture Notes in Computer Science 78,Springer,1979.
[9] Hagiya,M. and T.Sakurai,"Foundation of Logic Programming Based on Inductive Definition",New Generation Computing,2,pp.59-77,1984.
[10] Jaffar,J.,J-L.Lassez and J.Lloyd,"Completeness of the Nagation as Failure Rule", Proc. IJCAI83, Vol.1, pp.500-506, 1983.
[11] Kanamori,T.and H.Seki,"Verification of Prolog Programs Using An Extension of Execution", ICOT Technical Report,TR-093,1984.
[12] Kanamori,T.and K.Horiuchi,"Type Inference in Prolog and Its Applications",ICOT Technical Report,TR-095,1984.
[13] Kowalski,R.A.,"Logic for Problem Solving",Chap 10-12,North Holland,1980.
[14] Murray,N.V.,"Completely Non-Clausal Theorem Proving", Artificial Intelligence, Vol.18,pp.67-85,1982.
[15] Pereira,L.M.,F.C.N.Pereira and D.H.D.Warren,"User's Guide to DECsystem-10 Prolog", Occational Paper 15,Dept.of Artificial Intelligence, Edinburgh,1978.
[16] Tamaki,H. and T.Sato,"Unfold/Fold Transformation of Logic Programs", Proc.2nd International Logic Programming Conference,pp.127-138,1984.
[17] Tärnlund,S-Å.,"Logic Programming Language Based on A Natural Deduction System",UPMAIL Technical Report,No.6,1981.
[18] Weyrauch,R.W. and R.Milner,"Program Correctness in A Mechanized Logic", Proc.1st USA-Japan Computer Conference,1972.

Appendix. Closure of Atom

Theorem Closure is unique up to renaming.

Proof: Suppose A has two closures $\overline{A}'$ and $\overline{A}''$. Then from the condition (b), they are unifiable. Let its most general instance be $\overline{A} = \mu'(\overline{A}') = \mu''(\overline{A}'')$. Suppose a head of a recursive definite clause B_0 is unifiable with $\overline{A}$ by an m.g.u. $\tau_0 \circ \sigma$. Hence $(\tau_0 \circ \mu') \circ \sigma$ is a unifier of $\overline{A}'$ and B_0 and $(\tau \circ \mu'') \circ \sigma$ is a unifier of $\overline{A}''$ and B_0. Because $\overline{A}'$ and $\overline{A}''$ are closures of A, B_0 is unifiable with $\overline{A}'$ by an m.g.u. $\tau'_0 \circ \sigma'$ and unifiable with $\overline{A}''$ by an m.g.u. $\tau''_0 \circ \sigma''$. This means that for some ν' and ν'', $\sigma = \nu' \circ \sigma' = \nu'' \circ \sigma''$. For all i such that B_i is a recursive call, $\sigma(B_i) = \nu' \circ \sigma'(\overline{A}') = \nu' \circ \tau'_i(\overline{A}') = \nu'' \circ \sigma''(\overline{A}'') = \nu'' \circ \tau''_i(\overline{A}'')$. Hence $\sigma(B_i)$ is a common instance of $\overline{A}'$ and $\overline{A}''$. Because $\mu' \circ \mu''$ is an m.g.u. of $\overline{A}'$ and $\overline{A}''$, there exists a substitution τ_i such that $\sigma(B_i) = \tau_i \circ \mu'(\overline{A}') = \tau_i \circ \mu''(\overline{A}'')$. Then τ_i satisfies the condition of the closedness and $\overline{A}$ is a closure of A. Because of the condition (c), $\overline{A}'$ and $\overline{A}''$ are variants. Hence the closure is unique up to renaming.





Computation of $\overline{p(t_1, t_2, \ldots, t_n)}$

$i := -1$; $A_0 := p(t_1, t_2, \ldots, t_n)$; $P_0 :=$ the set of recursive definite clauses defining p;
repeat
  $i := i + 1$; select a recursive definite clause C in P_i fairly;
  if the head of C is unifiable with A_i
  then $A_{i+1} := closure1(A_i, C)$; $P_{i+1} := P_i - \{C\}$;
  else $A_{i+1} := A_i$; $P_{i+1} := P_i$;
until all heads of definite clauses in P_i are not unifiable with A_i
return A_{i+1}

$closure1(A, "B_0 :- B_1, B_2, \ldots, B_m")$;
$i := -1$; $A_0 := A$;
repeat forever
  $i := i + 1$;
  let $\tau_0 \circ \sigma$ be an m.g.u. of A_i and B_0
  where τ_0 and σ are the restrictions to A_i and B_0;
  let $B'_0, B'_1, B'_2, \ldots, B'_i$ be variants of atoms with p in $\sigma(B_0 :- B_1, B_2, \ldots, B_m)$
  without shared variable by an appropriate renaming;
  $B :=$ most specific common generalization of $B'_0, B'_1, B'_2, \ldots, B'_i$;
  if B is an instance of A_i **then** return A_i **else** $A_{i+1} := B$;

where most specific common generalization is the dual of most general common instanciation, i.e., E is a most specific common generalization of $E_1, E_2, \ldots, E_i$ when
(a) $E_1, E_2, \ldots, E_i$ are instances of E,
(b) E is an instance of any E' satisfying (a).
It is easily obtained by comparing the corresponding subexpressions.