TR-096

Verification of Prolog Programs Using an Extension of Execution

Tadashi Kanamori and Hirohisa Seki
(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

Verification of Prolog Programs Using An Extension of Execution

Tadashi KANAMORI,Hirohisa SEKI

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

Abstract

In this paper we describe an approach to verification of Prolog programs taking advantages of characteristics of Prolog. The most important feature in our approach is the use of an extension of execution, which is a generalization of the conventional Prolog interpreter. We use the extended execution to verify that a specification S in a class of first order formulas,called S-formulas, is a logical consequence of the completion of a program P. This approach is (1) simple because we need only an extention of the Prolog interpreter, (2) understandable because we put the execution and the verification into a single logical framework based on the natural deduction and (3) wasteless because we carry out verification without unnecessary explicit strengthening of P. We show how the extended execution works for the same example in the Boyer and Moore Theorem Prover (BMTP).

Keywords : Program Verification,Prolog,Natural Deduction,Theorm Proving.

Contents

1. Introduction

Logic programming is getting attracted because of its clear semantics. It is expected that this virtue makes verification (and other meta-processing and manipulation) of programs simpler and easier. But so far it has not yet been fully investigated what is really good in verification except a few works [5],[9],[22].

In this paper we show the verification and the execution of Prolog programs can be on a same axis by introducing an extension of execution and it actually makes the verification simple,understandable and wasteless.

After summarizing preliminary materials in section 2, we present a framework of verification of Prolog programs in section 3. In section 4,we describe a class of deductions, which is a generalization of the behavior of Prolog interpreter. In section 5, we show the extended execution can play a role of first order inference and be integrated into provers with induction like Boyer-Moore Theorem Prover (BMTP) by very simple exmaples. Then we present an example for comparison, which is intensionally the same as is used in Boyer and Moore [4]. 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),positive and negative literals, formula,substitution, most general unifier (mgu) and so on. We also assume knowledge about semantics of Prolog such as completion,minimum Herbrand model and transformation T of Herbrand interpretations (see [1],[6],[7],[8],[11]). We follow the syntax of DEC-10 Prolog [17]. 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 for substitutions, $\mathcal{F}_{\mathcal{G}}(\mathcal{H})$ for a replacement of *all* occurences of a formula $\mathcal{G}$ in a formula $\mathcal{F}$ with $\mathcal{H}$ and $\mathcal{F}_{\mathcal{G}}[\mathcal{H}]$ for a replacement of an occurence of a formula $\mathcal{G}$ in a formula $\mathcal{F}$ with $\mathcal{H}$.

2.1. Polarity of Subformulas

We generalize the distinctions of positive and negative goals. The *positive* and *negative subformulas* of a formula $\mathcal{F}$ are defined as follows (see [18],[16],[15]).

(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

∀ X,W (label(W) ∧ordered(X) ⊃∃Y insert(X,W,Y)).

Then $\exists Y\, insert(X, W, Y)$ is a positive 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 (list(A)⊃∀B ∃C append(A,B,C)).
Then A and B are both free variables, while C 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 (list(A)⊃∀B ∃C append(A,B,C)).
Then S is an S-formula, because free variables A and B are quantified outside $\exists C$ and C appears only in the positive atom $append(A, B, C)$. 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 (list(A)⊃∀B ∃C append(A,B,C))
is represented by a goal formula
list(A)⊃append(A,B,?C).
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.

Remark: This representation corresponds to backward application of ∀-introduction and ∃-introduction to positive $\forall X\,\mathcal{G}$ and $\exists X\,\mathcal{G}$ and forward application of ∀-elimination and ∃-elimination to negative $\forall X\,\mathcal{G}$ and $\exists X\,\mathcal{G}$ in the nartural deduction.

2.3. Manipulation of Goal Formulas

Lastly we introduce two manipulations of goal formulas.

One is an application of a class of substitutions. A substitution σ for G is called a *deciding substitution* when σ instanciates no free variable in G.

Example 2.3.1: Let S be

∀ A,B,U ((list(A) ⊃∃C append(A,B,C)) ⊃(list(A) ⊃∃C append([U|A],B,C)))
Then the goal formula of S is
(list(A) ⊃append(A,B,C)) ⊃(list(A) ⊃append([U|A],B,?C))
The most general common instance of $append([U|A], B, ?C)$ and the head of the second definite clause for *append* is obtained by a deciding substitution $\sigma = <?C \Leftarrow [U|?C'] >$. $\sigma(G)$ represents an S-formula
∀ A,B,U ((list(A) ⊃∃C append(A,B,C)) ⊃(list(A) ⊃∃C' append([U|A],B,[U|C'])))

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.

$\neg true \rightarrow false$,	$\neg false \rightarrow true$,
$true \wedge G \rightarrow G$,	$false \wedge G \rightarrow false$,
$G \wedge true \rightarrow G$,	$G \wedge false \rightarrow false$,
$true \vee G \rightarrow true$,	$false \vee G \rightarrow G$,
$G \vee true \rightarrow true$,	$G \vee false \rightarrow G$,
$true \supset G \rightarrow G$,	$false \supset G \rightarrow true$,
$G \supset true \rightarrow true$,	$G \supset false \rightarrow \neg G$.

Example 2.3.2: Let G_1 and G_2 be
(true ⊃reverse(C,A)) ⊃(true ∧append(C,[U],B) ⊃reverse(B,[U|A]))
(false ⊃reverse(C,A)) ⊃(false ∧append(C,[U],B) ⊃reverse(B,[U|A])).
Then $G_1\downarrow$ is $reverse(C, A) \supset (append(C, [U], B) \supset reverse(B, [U|A]))$ and $G_2\downarrow$ is $true$.

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 set of definite clauses whose head is with a unary predicate defining a data structure. (**type** in our verification system is being corresponded to *shell* in BMTP.) Procedures are defined following the syntax of DEC-10 Prolog [17],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 "B :- $B_1, B_2, \ldots, B_m$" ($m \geq 0$) 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 quantifications may be followed by a type qualifier : p (e.g.,:list above). $\forall X : p\mathcal{F}$ and $\exists X : p\mathcal{F}$ are abbreviations of $\forall X(p(X) \supset \mathcal{F})$ and $\exists X(\mathcal{F} \wedge p(X))$ respectively.

3.3. Formulation 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 first order inference and some induction when proof theoretically speaking. (See section 5 for induction.)

The most important difference between our verification system and BMTP is that specifications in BMTP are quantifier-free (i.e. universal) formulas while ours are S-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 $\forall X, Y(condition(X, Y) \supset p(X, Y))$ with respect to a program $p(X, Y) :- q(X, Z), r(Z, Y)$. Then we must prove $\forall X, Y(condition(X, Y) \supset \exists Z(q(X, Z) \wedge r(Z, Y))$ substantially.

4. An Extension of Execution

If we follow the previous formulation of verification, it is necessary to generalize the Prolog execution somehow so that we can perform first order inference from P^* on S-formulas. Our logic system needs carefull treatment of quantifiers, i.e. distinction of free variables and undecided variables (cf.[2]). Moreover it needs appropriate processings of logical connectives other than $\wedge$, because S has a more complicated form than usual execution goals (cf.[16] and Schütte [19]). Our logic system consists of the following seven inference rules. (See the following explanation for their notations.) Each rule says that subgoals in S-formulas above the line are generated from goals in S-formulas below the line. We assume variables in specification S are renamed appropriately so that there occurs no variable names conflict.

$\wedge$-deletion
$$\frac{G_H[H_1] \quad G_H[H_2] \quad \cdots \quad G_H[H_k]}{G_+[H_1 \wedge H_2 \wedge \cdots \wedge H_k]}$$

$\vee$-deletion
$$\frac{G_H[H_1] \quad G_H[H_2] \quad \cdots \quad G_H[H_k]}{G_-[H_1 \vee H_2 \vee \cdots \vee H_k]}$$

$\supset$-deletion
$$\frac{G_H[\neg H_1] \qquad G_H[H_2]}{G_-[H_1 \supset H_2]}$$

DCI $\dfrac{\sigma_1(G_A[\wedge_{i=1}^{m_1} B_{1i}]) \downarrow \quad \sigma_2(G_A[\wedge_{i=1}^{m_2} B_{2i}]) \downarrow \quad \cdots \quad \sigma_k(G_A[\wedge_{i=1}^{m_k} B_{ki}]) \downarrow}{G_+[A]}$

NFI $\dfrac{\sigma_1(G_A[\wedge_{i=1}^{m_1} B_{1i}]) \downarrow \quad \sigma_2(G_A[\wedge_{i=1}^{m_2} B_{2i}]) \downarrow \quad \cdots \quad \sigma_k(G_A[\wedge_{i=1}^{m_k} B_{ki}]) \downarrow}{G_-[A]}$

simplification $\dfrac{\sigma(G)_A(true) \downarrow \qquad \sigma(G)_A(false) \downarrow}{G}$

oracle decision $\dfrac{\sigma(G)}{G}$

4.1. Case Splitting

$\wedge$ may appear in more complicated ways in goal formulas.

$\wedge$-Deletion

Let G be a goal formula. When H is an outermost positive subformula of the form $H_1 \wedge H_2 \wedge \cdots \wedge H_k$ $(k > 1)$ and each undecided variable $?X$ appearing in H_i appaers only in H_i $(1 \leq i \leq k)$, we generate new k AND-goals $G_H[H_1], G_H[H_2], \ldots, G_H[H_k]$.

Example 4.1.1: Let S be

∀ A,B,C (append(A,B,C)∧list(C) ⊃∃D reverse(A,D)∧∃E reverse(B,E)).

Then the goal formula of S is

append(A,B,C)∧list(C) ⊃reverse(A,?D)∧reverse(B,?E).

By applying ∧-deletion, we have 2 AND-goals

append(A,B,C)∧list(C) ⊃reverse(A,?D).

append(A,B,C)∧list(C) ⊃reverse(B,?E).

Remark: ∧-deletion corresponds to backward application of ∧-introduction in the natural deduction.

One of the new logical connectives in goal formulas not appearing in usual execution goals is ∨.

∨-Deletion

Let G be a goal formula. When H is an outermost negative subformula of the form $H_1 \vee H_2 \vee \cdots \vee H_k$ $(k > 1)$ and each undecided variable $?X$ appearing in H_i appaers only in H_i $(1 \leq i \leq k)$, we generate new k AND-goals $G_H[H_1], G_H[H_2], \ldots, G_H[H_k]$.

Example 4.1.2: Let S be of the form

∀ S,T ((S=T ∨S<T ∨T<S) ⊃(···)).

Then the goal formula of S is

(S=T ∨S<T ∨T<S) ⊃(···).

By applying ∨-deletion, we have 3 AND-goals

S=T ⊃(···).

S<T ⊃(···).

T<S ⊃(···).

Remark: ∨-deletion corresponds to forward application of ∨-elimination in the natural

deduction.

Another important logical connective is $\supset$.

$\supset$-Deletion
Let G be a goal formula. When H is an outermost negative subformula of the form $H_1 \supset H_2$ and each undecided variable $?X$ appearing in H_i appaers only in H_i $(1 \leq i \leq 2)$, we generate new AND-goals $G_H[\neg H_1]$ and $G_H[H_2]$.

Example 4.1.3: Let S be
$\forall$ U,B,C,D_1,D_2 ((append(B,C,D_2) $\supset D_1 = D_2$) $\supset$(append(B,C,D_2) $\supset$[U|D_1]=[U|D_2])).
Then the goal formula of S is
(append(B,C,D_2) $\supset D_1 = D_2$) $\supset$(append(B,C,D_2) $\supset$[U|D_1]=[U|D_2]).
By applying $\supset$-deletion, we have AND-goals
$\neg$ append(B,C,D_2) $\supset$(append(B,C,D_2) $\supset$[U|D_1]=[U|D_2]).
$D_1 = D_2$ $\supset$(append(B,C,D_2) $\supset$[U|D_1]=[U|D_2]).

Reamrk: $\supset$-deletion corresponds to forward application of $\supset$-elimination with $\perp c$ in the natural deduction.

4.2. Definite Clause Inference

We generalize the execution of positive goals using the polarity.

Definite Clause Inference(DCI)
Let A be a positive atom in a goal formula G and "B :- $B_1, B_2, \ldots, B_m$" be any definite clause in P. When A is unifiable with B by a deciding mgu σ, we generate a new OR-goal $\sigma(G_A[B_1 \wedge B_2 \wedge \cdots \wedge B_m]) \downarrow$. ($B_1 \wedge B_2 \wedge \cdots \wedge B_m$ is *true* when $m = 0$.) All newly introduced variables are treated as fresh undecided variables.

Example 4.2.1: Let S be
$\forall$ A,B,U ((reverse(A,B)$\supset$reverse(B,A))$\supset$(reverse(A,[U|B])$\supset$reverse([U|B],A))).
Then the goal formula of S is
(reverse(A,B) $\supset$reverse(B,A)) $\supset$(reverse(A,[U|B]) $\supset$reverse([U|B],A))
We can apply DCI to $reverse([U|B], A)$ and it is replaced with $reverse(A, ?C) \wedge append(?C, [U], B)$.
Note the internal variable is treated as an undecided variable $?C$.

Example 4.2.2: When S is an existential formula of the form $\exists Y_1 Y_2 \cdots Y_m (A_1 \wedge A_2 \wedge \cdots \wedge A_k)$, i.e. of the form of usual execution goals, the goal formula of S is $?\text{-}A_1, A_2, \ldots, A_k$. (The juxtaposition delimited by "," denotes conjunction and $?\text{-}G$ denotes the goal formula obtained by replacing every variable Y in G with $?Y$.) Then usual execution is applied to $?\text{-}A_1, A_2, \ldots, A_k$.

Remark: DCI correspond to using "if" part of formulas in P^* as assumptions in the natural deduction. Soundness of DCI is guaranteed most easily by replacing equivalence with equivalence using P^* first and then constructing a proof of G from proofs of $\sigma(G_A[B_1 \wedge B_2 \wedge \cdots \wedge B_m])$ using $\vee$-introduction.

4.3. "Negation as Failure" Inference

We also generalize the execution of negative goals using the polarity.

"Negation as Failure" Inference(NFI)

Let A be a negative atom in a goal formula G and "B :- $B_1, B_2, \ldots, B_m$" be any definite clause in P. When A is not unifiable with B for any definite clause in P, we generate a goal $G_A[false] \downarrow$. When A is unifiable with B for some definite clause in P, we generate new AND-goals of $\sigma(G_A[B_1 \wedge B_2 \wedge \cdots \wedge B_m]) \downarrow$ for all such definite clauses. ($B_1 \wedge B_2 \wedge \cdots \wedge B_m$ is *true* when $m = 0$.) All newly introduced variables are treated as fresh free variables. (Note that A always includes only free variables and σ may be any mgu without restriction.)

Example 4.3.1: Let S be

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

Then the goal formula of S is

(reverse(A,B) ⊃reverse(B,A)) ⊃(reverse([U|A],B) ⊃reverse(B,[U|A]))

We can apply NFI to $reverse([U|A], B)$ and it is replaced with $reverse(A, C) \wedge append(C, [U], B)$. Note the internal variable is treated as a free variable C.

Example 4.3.2: Let S be a specification $\forall X(human(X) \supset \exists Z\, mother(Z, X))$ and the program P for *human* consists of k unit clauses $human(t(X_1))., human(t(X_2))., \ldots, human(t(X_k))$. When NFI is applied to the goal formula $human(X) \supset mother(?Z, X)$, the free variable X is instanciated in k-ways and k AND-goals $mother(?Z_1, t(X_1)), mother(?Z_2, t(X_2)), \ldots, mother(?Z_k, t(X_k))$ are generated. Note $?Z$ is not shared among these AND-goals.

Remark: NFI correspond to using "only if" part of formulas in P^* as assumptions in the natural deduction. Soundness of NFI is guaranteed most easily by replacing equivalence with equivalence using P^* first and then constructing a proof of G from proofs of $\sigma(G_A[B_1 \wedge B_2 \wedge \cdots \wedge B_m])$ using ∨-elimination.

4.4. Simplification

We sometimes simplify goal formulas assuming an atom *true* or *false* (cf.[16]).

Simplification

Let G be a goal formula. When $A_1, A_2, \ldots, A_m$ be positive atoms and $A_{m+1}, A_{m+2}, \ldots, A_n$ be negative atoms unifiable to A by a deciding mgu σ ($0 \leq m \leq n$), we generate new AND-goals $\sigma(G)_A(true) \downarrow$ and $\sigma(G)_A(false) \downarrow$.

In the following examples, σ are both $<>$ and undecided variables are not instanciated. For more general simplifications with instanciation of undecided variables, see 5.3.

Example 4.4.1: Let G be a goal formula

(add(X,Y,Z) ⊃add(Y,X,Z)) ⊃(add(X,Y,Z) ⊃add(Y,s(X),s(Z)))

of an S-formula

∀ X,Y,Z ((add(X,Y,Z) ⊃add(Y,X,Z)) ⊃(add(X,Y,Z) ⊃add(Y,s(X),s(Z)))).

Because $\sigma = <>$ is a deciding substitution and unifies the positive atom $add(X, Y, Z)$ and the negative atom $add(X, Y, Z)$, we generate new AND-goals

(true ⊃add(Y,X,Z)) ⊃(true ⊃add(Y,s(X),s(Z))) ↓,

(false ⊃add(Y,X,Z)) ⊃(false ⊃add(Y,s(X),s(Z))) ↓,

i.e., $add(Y, X, Z) \supset add(Y, s(X), s(Z))$ and *true*. (This inference corresponds to generate

(Y+X)+1=Y+(X+1)

from

X+Y=Y+X ⊃(X+Y)+1=Y+(X+1)

in functional programs, i.e. using the equation $X + Y = Y + X$ in premise and throwing

it away. This is called *cross-fertilization* in BMTP [4].)

Example 4.4.2: Let G be a goal formula
(reverse(A,B) ⊃reverse(B,A)) ⊃(reverse(A,C) ∧append(C,[U],B) ⊃reverse(B,[U|A]))
of an S-formula
∀ A,B,C,U ((reverse(A,C) ⊃reverse(C,A)) ⊃((reverse(A,C)∧append(C,[U],B))⊃reverse(B,[U|A]))).
Because $\sigma = <>$ is a deciding substitution and unifies the positive atom $reveres(A, C)$ and the negative atom $reverse(A, C)$, we generate new AND-goals
(true ⊃reverse(C,A)) ⊃(true ∧append(C,[U],B) ⊃reverse(B,[U|A])) ↓,
(false ⊃reverse(C,A)) ⊃(false ∧append(C,[U],B) ⊃reverse(B,[U|A])) ↓,
i.e., $reverse(C, A) \supset (append(C, [U], B) \supset reverse(B, [U|A]))$ and *true*. (This inference corresponds to infer
reverse(C)=A ⊃reverse(append(C,[U]))=[U|A]
from
reverse(reverse(A))=A ⊃reverse(append(reverse(A),[U]))=[U|A]
in functional programs, i.e. replacement of the special term $reverse(A)$ with a variable C. This is called *generalization* in BMTP [4].)

Remark: Simplification performs the role of inference rules in the natural deduction not mentioned so far. It corresponds to discharging of assumptions at ⊃-introduction. It also corresponds to application of $\perp_C$, because the use of $\perp_C$ is equivalent to additional axioms of all formulas of the form $\mathcal{F} \vee \neg \mathcal{F}$ (which is more similar to the Gentzen's original system).

4.5. Oracle Decision

The last inference rule is never done automatically in our verification system.

Oracle Decision
When ?X is in a goal formula G and $\sigma = <?X \Leftarrow t>$ is a deciding substitution, we generate a new goal $\sigma(G)$. All newly introduced variables are treated as fresh undecided variables.

Remark: Oracle decision corresponds to resolving the ambiguity in forward application of ∀-elimination and backward application of ∃-introduction in the natural deduction.

5. Examples

In this section, we show how the extended execution is used in verification of Prolog programs.

5.1. First Order Inference by Extended Execution

First we show the simplest first order inference performed by the extended execution. Let us prove the following *th*1 (cf. Kowalski [14],p.223).

theorem(th1).
∀ U ¬ append([ ],[U],[ ]).
end.

The extended execution proceeds as follows.

¬ append([ ],[U],[ ])

⇓ NFI for append([ ],[U],[ ]) (there is no unifiable head and $\neg false \downarrow$ is *true*)
true

This concludes "$P^* \vdash$th1".

5.2. Inductive Proof with Extended Execution

Secondly we show the use of the extended execution with induction. Let us prove the following *th*2 (cf.Kowalski [14] pp.221-222).

theorem(th2).
∀ A:list append(A,[ ],A).
end.

Before describing the verification process, we explain about the computational induction following Clark [7] p.75-76. The *list* relation is the smallest set of terms that includes [] and that,for any term s,includes $[s|t]$ whenever it includes t. Hence,suppose $Q(A)$ is a formula with free variable A. For any Herbrand interpretation, $Q(A)$ will denote some set of terms. If this set includes [ ],i.e.

Q([ ])

is true,and if it includes $[s|t]$ whenever it includes t, i.e.,

∀ A,U (Q(A)⊃Q([U|A]))

is true,then the set $Q(A)$ includes all terms in the *list* relation.In other words,

∀ A (list(A)⊃Q(A))

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

$$\frac{Q([\,]) \qquad \forall A,U\ (Q(A) \supset Q([U|A]))}{\forall A\ (\mathrm{list}(A) \supset Q(A))}$$

Let $Q(A)$ be $append(A, [\,], A)$.

Base Case

The subgoal $Q([\,])$ is represented by a goal formula $append([\,], [\,], [\,])$.

append([ ],[ ],[ ])
⇓ DCI for append([ ],[ ],[ ])
true

Induction Step

The subgoal $Q(A) \supset Q([U|A])$ is represented by a goal formula $append(A, [], A) \supset append([U|A], [], [U|A])$.

append(A,[ ],A) ⊃ append([U|A],[ ],[U|A]))
⇓ DCI for append([U|A],[ ],[U|A])
append(A,[ ],A) ⊃ append(A,[ ],A)
⇓ simplification w.r.t. append(A,[ ],A) and append(A,[ ],A)
true

This concludes "$P^* \vdash th2$".

5.3. An Example for Comparison

A well-known property of "*reverse*" is described as follows (Boyer and Moore [4]).

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

Let us prove *reverse-reverse* using the extended execution and the computational induction. The same discussion for *reverse* relation holds as for the *list* relation in the previous section. We have a computational induction scheme as follows.

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

Now let $Q(A, B)$ be *reverse*(B, A). (We omit the details how Q is found.See [12].)

Base Case

The subgoal $Q([\,],[\,])$ is represented by a goal formula *reverse*$([\,],[\,])$. The extended execution of *reverse*$([\,],[\,])$ proceeds as follows.

reverse([ ],[ ])
⇓ DCI for reverse([ ],[ ])
true

Induction Step

The subgoal $\forall U, A, B, C(Q(A, C) \wedge$*append*$(C, [U], B) \supset Q([U|A], B))$ is represented by a goal formula *reverse*$(C, A) \wedge$*append*$(C, [U], B) \supset$*reverse*$(B, [U|A])$. Now let *new-p*(L, M, N, X) be a procedure defined by

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

Then the *new-p* relation is also computed by the following program. (This is justified by the Tamaki-Sato's transformation [21]. For lack of space we omit the details.See [12]).

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

The theorem to be proved is now

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

and the computational induction scheme is

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

Now let $Q(A, B, C, U)$ be *reverse*$(B, [U|A])$. By applying computaional induction,we have two goals.

Base Case (Deeper Level)

The subgoal $\forall U Q([\,], [U], [\,], U)$ is represented by a goal formula *reverse*$([U], [U])$.

reverse([U],[U])
⇓ DCI for reverse([U],[U])
reverse([ ],?C) ∧append(?C,[U],[U])
⇓ DCI for reverse([ ],?C)

append([ ],[U],[U])
⇓ DCI for append([ ],[U],[U])
true

Induction Step (Deeper Level)

The subgoal $\forall A, B, C, U, A_1, V(Q(A_1, B, C, U) \wedge append(A_1, [V], A) \supset Q(A, [V|B], [V|C], U))$ is represented by a goal formula $reverse(B, [U|A_1]) \wedge append(A_1, [V], A) \supset reverse([V|B], [U|A])$

reverse(B,[U|A_1]) ∧ append(A_1,[V],A) ⊃ reverse([V|B],[U|A])
⇓ DCI for reverse([V|B],[U|A])
reverse(B,[U|A_1]) ∧ append(A_1,[V],A) ⊃ reverse(B,?A_2) ∧ append(?A_2,[V],[U|A])
⇓ DCI for append(?A_2,[V],[U|A])
reverse(B,[U|A_1]) ∧ append(A_1,[V],A) ⊃ reverse(B,[U|?A_1]) ∧ append(?A_1,[V],A)
⇓ simplification w.r.t. reverse(B,[U|A_1]) and and reverse(B,[U|?A_1])
append(A_1,[V],A) ⊃ append(A_1,[V],A)
⇓ simplification w.r.t. append(A_1,[V],A) and append(A_1,[V],A)
true

This concludes "$P^* \vdash$ *reverse-reverse*".

6. Discussions

Our approach is similar to one by Tärnlund,Haridi et al [9],[10],[22] which uses the natural deduction directly. They accomodate various manipulations of programs into a monolithic logical framework. It is human-oriented and keeps intuitive information each formula has so that the quality of human interface in interactive systems is not degraded. It also has an advantage of the chance to utilize the result about normal proof constructions. But it is different from ours in following 5 respects.

(1) Our inferences are sound.

Their proof construction is not necessarily sound and need to check whether the resultant proof tree is a true proof tree after having constructed it (see [10]). For lack of space, we omitted the formal discussions of soundness of our extended execution. But our inferences are sound, because we restricted attentions to S-formulas. Moreover we conjecture a completeness that any S-formula S is provable by extended execution if and only if S is a logical consequence of P^*.

(2) Our approach is based on unification much more strongly.

Their approach is closer to the original natural deduction. Our approach is based on it, but it has a flavor much more similar to the usual execution in Prolog, because main inferences are based on unification. (Especially simplification is not used in their approach.) This makes equational inferences completely implicit (except cases $=$ is used in P) and performs several steps of equational inference in one step.

(3) Our proof construction is in linear format.

Their proof tree construction is faithfull to the normal proof construction in the natural deduction. Corresponding to charging and discharging of assumptions, their proof construction changes its mode between "forward" and "backward" (see [10]). In our approach,instead

of explicit charging and discharging of separated assumptions, we keep assumptions and conclusions in a single formula of the form $\mathcal{F} \supset \mathcal{G}$. (In this point our approach is more similar to the sequent calculus.cf.[2].) This makes it possible to construct the corresponding proof trees (in the natural deduction) bi-directionally and proceed in a linear format.

(4) Our system is verification dedicated and controlled by many heuristics.

Their system is considered as an extension of the execution with respect to general formula programs. Ours is an extension with respect to definite clause programs. We take advantages of the fact fully that the completion P^* of a definite clause program P consists of formulas of a special form (though we do not do explicit strengthening to P^*) and it disturbs the quantification relations in the generated subgoals so little that the inferences for verification can be kept rather simple. Actual applications of the extended execution are controlled by many BMTP-like heuristics in our verification system [4],[13]. This can be considered as a kind of meta-inference ([20],Bowen and Kowalski [3]).

(5) Our inferences are integrated into a proof system with induction.

For lack of space we omitted the details how induction formulas are generated automatically. When there is no way to resort to in verification, we apply inductions and generate new induction goals [12] as BMTP resorts to the well-founded induction. But in many cases we can apply the de Bakker and Scott's computational induction, which skips several steps of inferences and generates more processed goals than naive structural inductions. For example,in the proof of *reverse-reverse*, the subgoal in **Induction Step** generated by naive structural induction is

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

and we need to apply NFI and simplification before applying the deeper level induction. In addition,we need not to guarantee termination of predicates in theorems to be proved because we employ the semantics based on the minimum Herbrand model = the least fixpoint of a transformation T of Herbrand interpretations (see Clark [7] pp.75-76).

8. Conclusions

We have shown how the interpreter of Prolog can be extended to execute more general formulas and how it can be utilized to verify specifications of Prolog programs. This extended execution is an element of our verification system for Prolog programs under development.

Acknowledgements

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] Bowen,K.A.,"Programming with Full First-Oreder Logic", Machine Intelligence 10 (J.E.Hayes, D.Michie and Y-H.Pao Eds), pp.421-440,1982.
[3] Bowen,K.A. and R.A.Kowalski,"Amalgamating Language and Metalanguage in Logic Programming",in Logic Programming (K.L.Clark and S-Å.Tärnlund Eds),Academic Press,1980.
[4] Boyer,R.S. and J.S.Moore,"Computational Logic",Academic Press,1979.
[5] 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.
[6] Clark,K.L.,"Negation as Failure",in Logic and Database (H.Gallaire and J.Minker Eds),pp.293-302,1978.
[7] Clark,K.L.,"Predicate Logic as a Computational Formalism", Chap.4, Research Monograph : 79/59,TOC,Imperial College,1979.
[8] 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.
[9] Hansson,A. and S-Å.Tärnlund,"A Natural Programming Calculus", Proc.6th International Joint Conference on Artificial Intelligence, pp.348-355,1979.
[10] Haridi,S. and D.Sahlin,"Evaluation of Logic Programs Based on Natural Deduction",Proc.2nd Workshop on Logic Programming,1983.
[11] Jaffar,J.,J-L.Lassez and J.Lloyd,"Completeness of the Nagation as Failure Rule", Proc. IJCAI83, Vol.1, pp.500-506, 1983.
[12] Kanamori,T.and H.Fujita,"Formulation of Induction Formulas in Verification of Prolog Programs",ICOT Technical Report,TR-094,1984.
[13] Kanamori,T.and K.Horiuchi,"Type Inference in Prolog and Its Applications",ICOT Technical Report,TR-095,1984.
[14] Kowalski,R.A.,"Logic for Problem Solving",Chap.10-12,North Holland,1980.
[15] Manna,Z.and R.Waldinger,"A Deductive Approach to Program Synthesis",ACM TOPLAS Vol.2,No.1,pp.90-121,1980.
[16] Murray,N.V.,"Completely Non-Clausal Theorem Proving", Artificial Intelligence, Vol.18,pp.67-85,1982.
[17] 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,1979.
[18] Prawitz,D.,"Natural Deduction,A Proof Theoretical Study", Almqvist & Wiksell,Stockholm,1965.
[19] Schütte,K., "Proof Theory",(translated by J.N.Crossley),Springer Verlag,1977.
[20] Stering,L. and A.Bundy,"Meta-Level Inference and Program Verification", in 6th Automated Deduction (W.Bibel Ed), Lecture Notes in Computer Science 138, pp.144-150,1982.
[21] Tamaki,H. and T.Sato,"Unfold/Fold Transformation of Logic Programs", Proc.2nd International Logic Programming Conference,pp.127-138,1984.
[22] Tärnlund,S-Å.,"Logic Programming Language Based on A Natural Deduction System",UPMAIL Technical Report,No.6,1981.