知識工学と人工知能　54－4
（１９８７・９・１８）

Ｐｒｏｌｏｇにおける再帰式の正当性に関するある十分条件

櫻井彰人、　元田　浩

（株）日立製作所基礎研究所

確定節によって表現される形式、とくに再帰形式（確定節の頭部の述語がその本体部にも現われる）が確定節のある集合から導かれる（その集合に対し正当である）ための十分条件を示す。一般に確定節の正当性を証明するには数学的帰納法を用いる必要があるが、数学的帰納法を自動的に適用するには帰納仮説の発見、証明過程での仮説の使用など、通常発見的に行なわれる推論を自動化しなければならない問題がある。本報告で提示する十分条件は、それが実際に満足されているか否かどうかは数学的帰納法を用いずに確かめることができる。そして、結果的に数学的帰納法が必要になる証明を、数学的帰納法を用いずに証明することを可能とするものである。

Sufficient Conditions for Validity of Recursion Formula in Prolog

Akito SAKURAI　　*Hiroshi MOTODA*
Advanced Research Laboratory, Hitachi Ltd.
1-280 Kokubunji, Tokyo 185, JAPAN

Abstract: A new approach for proving validity of recursive formuli written in definite clauses is developed based on resolution and definite clauses as functions. The main problem, when automatically proving it, is to develop strategy for inductive proof, or how to find out an appropriate induction scheme and how to direct deductions.
We propose a method to prove some properties on definite clauses without induction, which eventually satisfy sufficient conditions for formuli to hold.
We first prove sufficient conditions stated in terminology of functions, and then convert them to the ones stated in resolution equivalent to unfold transformation. We also prove a Prolog version of the fixed point induction and extend it.

1. Introduction

It is not an easy task to prove a logical sentence, when it requires us to use mathematical induction or structural induction. This is because the induction is different from other inference rules in that we have to search around heuristically for an appropriate induction variable and hypothesis. The first succesfull trial was done by R.S.Boyer and J.S.Moore [2] in their verification system for LISP functions. They combined some induction heuristics and a set of simple rewrite rules of LISP, and proved a wide variety of theorems about recursive LISP functions. Recently, Kanamori and Fujita [5] have built a verification system for Prolog. Since Prolog has a better semantic foundation than LISP, it seems the former is advantageous to build such a system. But the fact that Prolog is freer to construct structures, in contrast to LISP where only list structure is available, makes it difficult to make up in advance some induction scheme to be used in seek of inductive proof. It also forces us to search among many possible induction hypotheses, many of which lead us to nothing. Kanamori and Fujita used a fixed point induction scheme proposed by K.L.Clark [3] combined with Tamaki and Sato's Prolog program tranformation [8] and their generalization method. But still we have to adopt many heuristics to conduct inductive proofs.

The purpose of this paper is to present some useful theorems on sufficient conditions for validity of definite clauses. The conditions, when applied to problems, do not require inductive proofs to be verified. The theorems are based on functional property of definite clauses and are proved using model theoretic semantics and fixed point semantics. We also formally prove and extend notion of the fixed point induction scheme introduce by K.L.Clark [3].

Due to limitations of time and space, in this paper we present only restricted cases. After summarizing preliminary materials in section 2, we present the theorems in section 3. In section 4, we show relations between resolution and composition of functions that are alias of definite functions. In section 5, we define concept of predicate substitution which is used in section 6, where computational induction scheme is proved and extended. These relations are fully utilized in section 7 where four examples are given to depict the usefulness and easiness of application of the theorems.

2. Preliminaries

In this section we first give necessary definitions on terminology of logic programming, and then give notions of logic programs as functions. Note that we use terminologies not in general but in quite restricted meaning. General and thorough treatments will be found in Van Emden and Kowalski [4] and Apt and Van Emden [1] for logic programming and J.-L.Lassez and M.J.Maher [7] for treatment of logic programs as functions.

Syntax. We distinguish the sets of variables, function symbols and predicate symbols. A constant is a 0-ary function symbol. A term is a variable, a constant or of the form $f(t_1,\ldots,t_n)$ where f is an n-ary function symbol and $t_1,\ldots,t_n$ are terms. An atomic formula (or atom in short) is of the form $P(t_1,\ldots,t_n)$ where P is an n-ary predicate symbol and $t_1,\ldots,t_n$ are terms. An atom is ground if it has no occurrence of variable. A definite clause is a clause of the form $A \leftarrow B_1,\ldots,B_n \quad n \geq 0$ where $A, B_1,\ldots,B_n$ are atomic formulae and variables occurring in them are universally quantified. A is called the head and $B_1,\ldots,B_n$ is called the body. A definite clause with only one atom in its body is said of singleton-body. A definite clause is ground if it has no occurrence of variables. A program is a finite set of definite clauses. A substitution σ is an operation defined on an atom which replaces simultaneously all occurrences of given varibles with given terms. The result is called an instance and denoted by $A\sigma$ for an atom A. A substitution is ground if the result is ground. This naturally extends to a set of atoms, clauses and programs.

Model-Theoretic Semantics. The Herbrand base of a program P is the set of all ground atoms containing no function or predicate symbols other than those occurring in P. An interpretation is a subset of the Herbrand base. The truth value is defined with respect to a given interpretation I as follows.

- A ground atom A is true in I iff $A \in I$
- A ground clause $A \leftarrow B_1,\ldots,B_n$ is true in I iff A is true in I or at least one of the B_i's is not true in I.
- A clause $A \leftarrow B_1,\ldots,B_n$ is true in I iff all of its ground instances are true in I.
- A program P is true in I iff all of its definite clauses are true in I.

A model for P is an interpretation in which P is true. P is satisfiable if P has a model. An atom or a definite clause C is a logical consequence of a program P, iff $P \cup \{\neg C\}$ is not satisfiable, that is, has no model. We denote this by $P \models C$. A set of definite clauses Q is a logical consequence of a program P, iff $P \cup \{\neg Q\}$ is not satisfiable, that is, has no model, where $\{\neg Q\}$ is an "or" connection of negation of clauses in Q. We denote this similaryly by $P \models Q$. It is proved that $P \models Q$ iff all the definite clauses in Q is true in any model of P. M_P is the least Herbrand model for P, that is, the intersection of all Herbrand models for P. M_P is proved to be equal to $\{A; A \text{ is a ground atom and } P \models A\}$. If $P \subseteq Q$ for programs P and Q, then $M_P \subseteq M_Q$.

Fixed Point Semantics. With a program P we associate a function T_P from interpretations to interpretations in the following way,

$$A \in T_P(I) \quad \text{iff} \quad A \leftarrow B_1,\ldots,B_n \ (n \geq 0) \text{ is a ground instance of a clause in } P \text{ and } B_1,\ldots,B_n \in I$$

It is proved that T_P is continuous in the lattice theoretic sense considering the power set of the Herbrand base as a base set of the lattice with inclusion among subsets as its partial order. That is, $T_P(\bigcup_{i=0}^{\infty} I_i) = \bigcup_{i=0}^{\infty} T_P(I_i)$ for any increasing sequence $\{I_i\}_{i=0,\ldots}$ of interprretations. It is also proved that an interpretation I is a model for P iff $T_P(I) \subseteq I$.

Definite Clauses as Functions. A fact is a definite clause without body predicates. A rule is a definite clause with at least one body predicate. With a set of rules R, we associate a function $[R]$ from interpretations to interpretations in the following way,

$$A \in [R](I) \quad \text{iff} \quad A \leftarrow B_1, \ldots, B_n (n > 0) \text{ is a ground instance of a clause in } R \text{ and } B_1, \ldots, B_n \in I$$

Id is the identity function from interpretations to interpretations. $[R_1]+[R_2]$ is defined as $([R_1]+[R_2])(I) \stackrel{\text{def}}{=} [R_1](I) \cup [R_2](I)$. $[R_1] \circ [R_2]$ is defined as $([R_1] \circ [R_2])(I) \stackrel{\text{def}}{=} [R_1]([R_2](I))$. $[R]^n$ is defined inductively as $[R]^0 \stackrel{\text{def}}{=} Id$ and $[R]^{n+1} \stackrel{\text{def}}{=} [R] \circ [R]^n$. It is easily shown $[R]^n + [R]^n = [R]^n$. It is proved that $[R]$ is monotonous, that is, if $I \subseteq J$ then $[R](I) \subseteq [R](J)$. $[R]$ is proved continuous as T_P is. With a set of rules R, we associate another function $[\![R]\!]$ from interpretations to interpretations in the following way

$$A \in [\![R]\!](I) \quad \text{iff} \quad \exists n \geq 0 \quad A \in [R]^n(I)$$

Obviously $[\![R]\!] = \sum_{i=0}^{\infty}([R]+Id)^i = \lim_{i \to \infty}([R]+Id)^i$. Note that in J.-L.Lassez and M.J.Maher [7], a program P is a set of rules and their $[P]$ is equivalent to our $[\![P]\!]$. The following properties are proved.

- $[\![R]\!]$ is monotonous, that is, if $I \subseteq J$ then $[\![R]\!](I) \subseteq [\![R]\!](J)$.
- $[\![R]\!]$ is continuous and idempotent (*i.e.*, $[\![R]\!] \circ [\![R]\!] = [\![R]\!]$).
- Let R be a set of all the rules in a program P and F be a set of all the facts in P. An interpretation I is a model for P iff $[R](I) \subseteq I$ and $M_F \subseteq I$.
- $M_P = [\![R]\!](M_F)$.

3. The First Type Conditions

Throughout this section P denotes a program which consists of a set of rules R and a set of facts F, and R' denotes a set of rules to be proved under the program P, with all the predicates defined in P. Q also denotes a program.

Proposition 3.1. *For any program* $P' = P'_1 \cup P'_2$, $P \models P'$ *iff* $P \models P'_1$ *and* $P \models P'_2$.

Proposition 3.2. $P \models R'$ *iff* $[R'](M_P) \subseteq M_P$.

Proposition 3.3. $M_P \subseteq M_Q$ *if* $M_F \subseteq M_Q$ *and* $Q \models R$

Proposition 3.4. $P \models R'$ *if rules in* R' *are of singleton-body and* $\forall n \exists m [R'][R]^n(M_F) \subseteq ([R]+Id)^m(M_F)$.

Proposition 3.5. $P \models R'$ *if rules in* R *and* R' *are of singleton-body and* $\forall n \; \exists m \; [R'][R]^n(M_F) \subseteq ([R]+Id)^m([R']+Id)(M_F)$, *and* $[R'](M_F) \subseteq [\![R]\!](M_F)$.

Proposition 3.6. $P \models R'$ *if rules in* R *and* R' *are of singleton-body and there exists a positive integer* N *such that* $\forall n \leq N \; \exists m \; \forall p \; [R'][R]^n[R]^p(M_F) \subseteq ([R]+Id)^m([R']+Id)[R]^p(M_F)$, *and* $[R'](M_F) \subseteq [\![R]\!](M_F)$.

Theorem 3.7. $P \models R'$ *if rules in* R *and* R' *are of singleton-body and there exists a positive integer* N *st.*

$\exists m \; \forall p \; [R'][R]^N[R]^p(M_F) \subseteq ([R]+Id)^m([R']+Id)[R]^p(M_F)$,
$\forall n \leq N-1 \; \exists m \; [R'][R]^n(M_F) \subseteq ([R]+Id)^m([R']+Id)(M_F)$, *and* $[R'](M_F) \subseteq [\![R]\!](M_F)$.

Corollary 3.8. $P \models R'$ *if rules in* R *and* R' *are of singleton-body and* $[R]$ *and* $[R']$ *is commutable, and* $[R'](M_F) \subseteq [\![R]\!](M_F)$.

4. Resolution and Rule Composition

In the argument of the sections 2 and 3, we defined a function composition as a succesive application of functions. This is enough for theories, but not for applications. We define a rule composition and relate it to function composition. In the following of this section, R_1 and R_2 mean sets of rules.

Definition 4.1. $R_1 \circ R_2$ *is defined as a set of rules obtained from all the rules in* R_1 *resolving them at all their body predicates by all the rules in* R_2. *If any rule in* R_1 *has a body predicate which has no resolvable rule in* R_2, *then no rules in* $R_1 \circ R_2$ *are obtained from the rule.*

Proposition 4.2. $[R_1 \circ R_2] = [R_1] \circ [R_2]$

Definition 4.3. *Id is also used to denote a set of rules of the form* $p(X_1, \ldots, X_n) \leftarrow p(X_1, \ldots, X_n)$ *for all the predicates* p *appearing in all the programs under consideration.*

Definition 4.4. R^n *for a set of rules* R *is inductively defined as* $R^0 \stackrel{\text{def}}{=} Id$ *and* $R^{n+1} \stackrel{\text{def}}{=} R \circ R^n$.

Definition 4.5. "+" *is used instead of* "$\cup$" *for a set of rules when in the context of functions, that is,* $R_1 + R_2 = R_1 \cup R_2$.

Definition 4.6. *Two rules or two facts are equal to each other when they are literally the same or they become literally the same after the variables appearing in them are systematically renamed, i.e., in each rule or fact different ones remain different and the same ones remain the same. These ones are said to be "variant" to each other.*

Definition 4.7. *A rule or a fact* C *is a member of a set of rules, or a set of facts or a program* R, *iff there exists a member* R_i *in* R *and a substitution* σ *so that* C *is literally equal to* $R_i\sigma$. *A set of facts, a set of rules or a program* R_1 *is a subset of a set of facts, a set of rules or a program* R_2 *iff any element of* R_1 *is a member of* R_2 *in the above stated meaning.*

Proposition 4.8. *$R_1 \circ (R_2 + Id)$ is equal to a set of rules obtained from all the rules in R_1 resolving them at none, some or all of their body predicates by all the rules in R_2.*

Note that in general $R_1 \circ (R_2 + Id) \neq R_1 \circ R_2 + R_1$, instead $R_1 \circ (R_2 + Id) \supseteq R_1 \circ R_2 + R_1$ holds.

Proposition 4.9. *The following equalities hold.*

$$(R_1 + Id) \circ R_2 = R_1 \circ R_2 + R_2$$
$$(R_1 + Id) \circ (R_2 + Id) = R_1 \circ (R_2 + Id) + (R_2 + Id)$$
$$[R_1 + R_2] = [R_1] + [R_2]$$
$$[R]^n = [R^n]$$
$$[R_1 \circ (R_2 + Id)] = [R_1] \circ [R_2 + Id]$$
$$[(R_1 + Id) \circ R_2] = [R_1 + Id] \circ [R_2]$$
$$[(R_1 + Id) \circ (R_2 + Id)] = [R_1 + Id] \circ [R_2 + Id]$$

Porposition 4.10. *If $R_1 \subseteq R_2$ then $R_1(I) \subseteq R_2(I)$ for any interpretation I.*

We define $R \circ F$ for a set of rules R and a set of facts F in a similar way. Then we have similar propositions as follows.

Definition 4.11. *$R \circ F$ is defined as a set of facts obtained from all the rules in R resolving them at all their body predicates by all the facts in F. If any rule in R has a body predicate which has no resolvable fact in F, then no facts in $R \circ F$ are obtained from the rule.*

Proposition 4.12. $[R](M_F) = M_{R \circ F}$

Proposition 4.13. $[R](M_F) \subseteq [R'](M_{F'})$ *if* $R \circ F \subseteq R' \circ F'$.

Proposition 4.14. $[R'](M_F) \subseteq [\![R]\!](M_F)$ *if* $\exists m\ R' \circ F \subseteq (R + Id)^m \circ F$.

Using the definitions and propositions in this section, the theorems and corollaries proved in section 3 are easily rewritten to the ones which are more easy to apply.

Proposition 4.15. *$P \models R'$ if rules in R and R' are of singleton-body and there exists a positive integer N such that* $\forall n \leq N\ \exists m\ R' \circ R^n \subseteq (R + Id)^m \circ (R' + Id)$, and $\exists m\ R' \circ F \subseteq (R + Id)^m \circ F$.

Theorem 4.16. *$P \models R'$ if rules in R and R' are of singleton-body and there exists a positive integer N such that*

$\exists m\ R' \circ R^N \subseteq (R + Id)^m \circ (R' + Id)$,
$\forall n \leq N-1\ \exists m\ R' \circ R^n \circ F \subseteq (R + Id)^m \circ (R' + Id) \circ F$, *and*
$\exists m\ R' \circ F \subseteq (R + Id)^m \circ F$.

Corollary 4.17. *$P \models R'$ if rules in R and R' are of singleton-body and*
R and R' are commutable, i.e., $R \circ R' = R' \circ R$, and $\exists m\ R' \circ F \subseteq (R + Id)^m \circ F$.

5. Predicate Substitution

In this section, we introduce a notion of predicate substitution. This will be used in the next section to justify a method to be used in proving $\forall X_1, \ldots, X_n\ p(X_1, \ldots, X_n) \rightarrow q(X_{i_1}, \ldots, X_{i_n})$. The method is to replace all the occurrences of a predicate p by a predicate q in a definition program of p, and prove it.

Definition 5.1. *$\vartheta = \{p(X_1, \ldots, X_n)/q(X_{i_1}, \ldots, X_{i_n})\}$ is a* predicate substitution *(or* p-substitution *in short) of p by q if p and q are Prolog predicates with the same number of arguments, $X_1, \ldots, X_n$ are distinct variables and $(i_1, \ldots, i_n)$ is a permutaion of $(1, \ldots, n)$. When P is a program, a set of rules or a set of facts, $P\vartheta$ means the one obtained from P by replacing every occurence of $p(X_1, \ldots, X_n)$ in P by $q(X_{i_1}, \ldots, X_{i_n})$, leaving no occurence of p in $P\vartheta$. When P is a program, $M_P\vartheta$ means the set of ground atoms obtained from M_P, the least model of P, by replacing every occurence of $p(X_1, \ldots, X_n)$ in M_P by $q(X_{i_1}, \ldots, X_{i_n})$, leaving no occurence of p in M_P.*

Definition 5.2. *The inversion of ϑ is defined as $\vartheta^{-1} = \{q(X_{i_1}, \ldots, X_{i_n})/p(X_1, \ldots, X_n)\}$ when $\vartheta = \{p(X_1, \ldots, X_n)/q(X_{i_1}, \ldots, X_{i_n})\}$. Obviously ϑ^{-1} is a predicate substitution of q by p.*

Proposition 5.3. *The following properties hold for any predicate substitution ϑ.*

For any ground atom a, $a = a\vartheta\vartheta^{-1}$
For any atom A, $A = A\vartheta\vartheta^{-1}$
For any definite clause F, $F = F\vartheta\vartheta^{-1}$
For any program P, $P = P\vartheta\vartheta^{-1}$
For any definite clause F and ground substitution σ, $F\sigma\vartheta = F\vartheta\sigma$
For any interpretation I, $I = I\vartheta\vartheta^{-1}$
For any ground atom a and interpretation I, $a \in I$ iff $a\vartheta \in I\vartheta$
For any interpretations I and J, if $I \subseteq J$ then $I\vartheta \subseteq J\vartheta$

Proposition 5.4. *$([R_i](I))\vartheta = [R_i\vartheta](I\vartheta)$, where R_i is a rule, I is an interpretation and ϑ is a predicate substitution.*

Proposition 5.5. $M_P\vartheta = M_{P\vartheta}$.

In the above discussion we only considered a predicate substitution with same number of variables. But in some cases we need consider other cases, that is, substitution of predicates with different number of variables. In the follwing, we introduce notions of redundant and incomplete predicate substitution. To distinguish the predicate substitution defined already from the one defined here, we call the former a complete predicate substitution, if necessary.

Definition 5.6. *$\vartheta = \{p(X_1,\ldots,X_n)/q(X_{i_1},\ldots,X_{i_m})\}$ is a* redundant predicate substitution *(or* redundant p-substitution *in short) of p by q if p and q are Prolog predicates, $n < m$, $X_{i_1},\ldots,X_{i_m}$ are distinct variables and $(i_1,\ldots,i_m)$ is a permutaion of $(1,\ldots,m)$. Variables $X_{n+1},\ldots,X_m$ in ϑ are called redundant variables. When P is a set of facts, a set of rules or a program, $P\vartheta$ means the one obtained from P by replacing every occurence of $p(X_1,\ldots,X_n)$ in P by $q(X_{i_1},\ldots,X_{i_m})$, leaving no occurence of p in $P\vartheta$, and introducing new variables within a definite clause for redundant variables in each occurence of p. When P is a program, $M_P\vartheta$ means the set of ground atoms obtained from M_P, the least model of P, by replacing every occurence of $p(X_1,\ldots,X_n)$ in M_P by $q(X_{i_1},\ldots,X_{i_m})$, instanciating the redundant variables by all possible ground substitution, leaving no occurence of p in M_P.*

Definition 5.7. *$\vartheta = \{p(X_1,\ldots,X_n)/q(X_{i_1},\ldots,X_{i_m})\}$ is an* incomplete predicate substitution *(or an* incomplete p-substitution *in short) of p by q if p and q are Prolog predicates, $n > m$, $X_1,\ldots,X_n$ are distinct variables and $(i_1,\ldots,i_m)$ is a permutaion of a subsequence of $(1,\ldots,n)$. When P is a program, a set of rules or a set of facts, $P\vartheta$ means the one obtained from P by replacing every occurence of $p(X_1,\ldots,X_n)$ in P by $q(X_{i_1},\ldots,X_{i_m})$, leaving no occurence of p in $P\vartheta$. When P is a program, $M_P\vartheta$ means the set of ground atoms obtained from M_P, the least model of P, by replacing every occurence of $p(X_1,\ldots,X_n)$ in M_P by $q(X_{i_1},\ldots,X_{i_m})$, leaving no occurence of p in M_P.*

Definition 5.8. *The inversion of ϑ is defined as $\vartheta^{-1} = \{q(X_{i_1},\ldots,X_{i_m})/p(X_1,\ldots,X_n)\}$ when $\vartheta = \{p(X_1,\ldots,X_n)/q(X_{i_1},\ldots,X_{i_m})\}$. Obviously ϑ^{-1} is an incomplete predicate substitution of q by p when ϑ is a redundant predicate substitution, and ϑ^{-1} is a redundant predicate substitution of q by p when ϑ is an incomplete predicate substitution.*

The Proposition 5.3, Proposition 5.4 and Proposition 5.5 still hold for redundant substitutions, but not for incomplete substitutions.

6. The Second Type Conditions

The sufficient conditions presented in the section 3 and 4 (and their extended versions where longer bodies are allowed) are easily verified in many applications. But there are simple cases where these methods do not work. For example, a case to prove $q(X,Y) \leftarrow p(X,Y)$ where in p only the first argument is the recursion variable (that is, the first argument of the predicate in the body of the difinition clause is a substructure of the first argument in its head) whereas in q only the second argument is the recusion variable, is the one.

In this section, we consider another type of sufficient conditions specific for the validity of $\forall X_1,\ldots,X_n\ p(X_1,\ldots,X_n) \rightarrow q(X_{i_1},\ldots,X_{i_n})$. Note that we could consider this formula as a definite clauses with $q(X_1,\ldots,X_n)$ as its head and $p(X_1,\ldots,X_n)$ as its body, and proceed to prove that sufficint conditions stated in previouly mentioned theorems and corollaries are satisfied. But as we said before this would not work for some cases.

The basic idea to handle this is similar to that of Clark [3] and Kanamori and Fujita [5], in which it is stated rather intuitively without proof. The idea is, to prove $p(\ldots) \rightarrow q(\ldots)$ it suffices to pick up all the definition clauses of p, to replace all the occurences of p in them with q, and to prove the replaced clauses. This idea, based on the fixed point induction, seems quite simple and obvious in simple cases as they exemplified, but is not so trivial in application to more general cases that we need some care.

In the following we restate the idea and justify it.

Let us consider a program defining p, q and all other necessary predicates under consideration. Let P (or Q) be a program obtained from the above mentioned program by deleting the clauses whose head predicate symbol is q (or p, respectively). Then the program mentioned is equal to $P \cup Q$. It is easily observed that $M_{P\cup Q} = M_P \cup M_Q$ means that p (or q) is defined without any knowledge of q (or p, respectively), or that p and q are defined without referring to one another.

In the remaining of this section, the above defined P, Q, p, q, $(X_1,\ldots,X_n)$, $(X_{i_1},\ldots,X_{i_n})$ are used in the above stated meaning.

Proposition 6.1. *Suppose $M_{P\cup Q} = M_P \cup M_Q$. Then, $\forall X_1,\ldots,X_n\ p(X_1,\ldots,X_n) \rightarrow q(X_{i_1},\ldots,X_{i_n})$, iff $\forall X_1,\ldots,X_n\ \ p(X_1,\ldots,X_n) \in M_P \rightarrow q(X_{i_1},\ldots,X_{i_n}) \in M_Q$.*

The following theorem is a Prolog version with slight modifications of the fixed point induction.

Theorem 6.2. *Suppose $M_{P\cup Q} = M_P \cup M_Q$. Then, $\forall X_1,\ldots,X_n\ p(X_1,\ldots,X_n) \rightarrow q(X_{i_1},\ldots,X_{i_n})$, iff $M_{P\vartheta} \subseteq M_Q$, where ϑ is a predicate substitution of p by q.*

Corollary 6.3. *Suppose $M_{P\cup Q} = M_P \cup M_Q$. Then, $\forall X_1,\ldots,X_n\ p(X_1,\ldots,X_n) \leftrightarrow q(X_{i_1},\ldots,X_{i_n})$, iff $M_{P\vartheta} = M_Q$, where ϑ is a predicate substitution of p by q.*

Theorem 6.4. *Suppose $M_{P\cup Q} = M_P \cup M_Q$. Then, $\forall X_1,\ldots,X_n\ p(X_1,\ldots,X_n) \rightarrow q(X_{i_1},\ldots,X_{i_n})$, if $M_{F\vartheta} \subseteq M_Q$ and $Q \models R\vartheta$, where ϑ is a predicate substitution of p by q, P consists of a set of rules R and a set of facts F.*

Next, we consider a sufficient condition for the validity of $\forall X_i$'s $p(X_1,\ldots,X_n) \rightarrow q(X_{i_1},\ldots,X_{i_m})$. The argument to follow is to redundant or incomplete substitutions, as what the argument above is to complete substitutions. As is easily forseen, we could find sufficient conditions for the case $n < m$, but not for the case $n > m$. We restate the theorems here without proofs.

Proposition 6.5. *Suppose* $M_{P\cup Q} = M_P \cup M_Q$, $n < m$ *and* $(i_1,\ldots,i_m)$ *is a permutation of* $(1,\ldots,m)$. *Then,* $\forall X_1,\ldots,X_m\ p(X_1,\ldots,X_n) \to q(X_{i_1},\ldots,X_{i_m})$,
iff $\forall X_1,\ldots,X_m\ \ p(X_1,\ldots,X_n) \in M_P \to q(X_{i_1},\ldots,X_{i_m}) \in M_Q$.

Theorem 6.6. *Suppose* $M_{P\cup Q} = M_P \cup M_Q$. *Then,* $\forall X_1,\ldots,X_m\ p(X_1,\ldots,X_n) \to q(X_{i_1},\ldots,X_{i_m})$, *iff* $M_{P\vartheta} \subseteq M_Q$, *where* ϑ *is a redundant predicate substitution of* p *by* q.

Theorem 6.7. *Suppose* $M_{P\cup Q} = M_P \cup M_Q$. *Then,* $\forall X_1,\ldots,X_m\ p(X_1,\ldots,X_n) \to q(X_{i_1},\ldots,X_{i_m})$, *if* $M_F\vartheta \subseteq M_Q$ *and* $Q \models R\vartheta$, *where* ϑ *is a redundant predicate substitution of* p *by* q, P *consists of a set of rules* R *and a set of facts* F.

7. Applications

7.1 The First Example: To Get $(x+1)+y$ from $x+(y+1)$

This example is to show a Prolog version of if $x+(y+1)=(x+y)+1$ then $(x+1)+y=(x+y)+1$

The problem stated in Prolog is to show $P \models R'$ where

$$P:\begin{cases} sum(X,0,X). \\ sum(X,s(Y),s(Z)) \leftarrow sum(X,Y,Z). \end{cases} \qquad R':\ sum(s(X),Y,s(Z)) \leftarrow sum(X,Y,Z).$$

Utilizing the corollary, we reduce the problem to showing the followings.

- $R' \circ \{sum(X,0,X).\} \subseteq \{sum(X,0,X).\}$
- $\{sum(s(X),Y,s(Z)) \leftarrow sum(X,Y,Z).\}$ and $\{sum(X,s(Y),s(Z)) \leftarrow sum(X,Y,Z).\}$ are commutable.

The correctness of the above is clear, since
$(R' \circ \{sum(X,0,X).\} \subseteq \{sum(X,0,X).\})$ $R' \circ \{sum(X,0,X).\} = \{sum(s(X),0,s(X)).\} \subseteq \{sum(X,0,X).\}$
(commutativity) $R' \circ \{\text{the rule in } P\} = \{sum(s(X),s(Y),s(s(Z))) \leftarrow sum(X,Y,Z).\}$
$\{\text{the rule in } P\} \circ R' = \{sum(s(X),s(Y),s(s(Z))) \leftarrow sum(X,Y,Z).\}$

7.2 The Second Example: Sequence of a's and b's

Let us prove the equivalence of the following two programs.

$$P_1:\begin{cases} s(0). \\ s(a(0)). \\ R_1:\begin{cases} s(a(a(X))) \leftarrow s(X). \\ s(a(b(X))) \leftarrow s(X). \\ s(b(X)) \quad\ \leftarrow s(X). \end{cases} \end{cases} \qquad P_2:\begin{cases} s(0). \\ s(b(0)). \\ R_2:\begin{cases} s(b(b(X))) \leftarrow s(X). \\ s(b(a(X))) \leftarrow s(X). \\ s(a(X)) \quad\ \leftarrow s(X). \end{cases} \end{cases}$$

The proof proceeds informally as follows. In the following, $\{\!\!\{$ and $\}\!\!\}$ are used to denote { and } for R' for the sake of clarity. We will show only the case $P_1 \models P_2$.

- $s(0) \Rightarrow s(0)$. Hence, $P_1 \models s(0)$.
- $s(0), s(b(X)) \leftarrow s(X) \Rightarrow s(b(0))$. Hence, $P_1 \models s(b(0))$.
- Next, consider if R_1 and R_2 satisfies the condition of Theorem 4.16
  - $\{\!\!\{s(b(b(X))) \leftarrow s(X)\}\!\!\} = \{s(b(X)) \leftarrow s(X)\}^2$.
  - Set $R_2' = \{s(b(a(X))) \leftarrow s(X)\} \cup \{s(a(X)) \leftarrow s(X)\}$.
    - $\{\!\!\{s(b(a(X))) \leftarrow s(X)\}\!\!\} \circ \{s(a(a(X))) \leftarrow s(X)\} =$
      $\{s(b(X)) \leftarrow s(X)\} \quad \circ \{s(a(a(X))) \leftarrow s(X)\} \circ \{\!\!\{s(a(X)) \leftarrow s(X)\}\!\!\}$
    - $\{\!\!\{s(b(a(X))) \leftarrow s(X)\}\!\!\} \circ \{s(a(b(X))) \leftarrow s(X)\} =$
      $\{s(b(X)) \leftarrow s(X)\} \quad \circ \{s(a(a(X))) \leftarrow s(X)\} \circ \{s(b(X)) \leftarrow s(X)\}$
    - $\{\!\!\{s(b(a(X))) \leftarrow s(X)\}\!\!\} \circ \{s(b(X)) \leftarrow s(X)\} = \{s(b(X)) \leftarrow s(X)\} \circ \{s(a(b(X))) \leftarrow s(X)\}$
    - $\{\!\!\{s(a(X)) \leftarrow s(X)\}\!\!\} \circ \{s(a(a(X))) \leftarrow s(X)\} = \{s(a(a(X))) \leftarrow s(X)\} \circ \{\!\!\{s(a(X)) \leftarrow s(X)\}\!\!\}$
    - $\{\!\!\{s(a(X)) \leftarrow s(X)\}\!\!\} \circ \{s(a(b(X))) \leftarrow s(X)\} = \{s(a(a(X))) \leftarrow s(X)\} \circ \{s(b(X)) \leftarrow s(X)\}$
    - $\{\!\!\{s(a(X)) \leftarrow s(X)\}\!\!\} \circ \{s(b(X)) \leftarrow s(X)\} = \{s(a(b(X))) \leftarrow s(X)\}$

    Therefore we get $R_2' \circ R_1 \subseteq (R_1^3 + R_1^2 + R_1) + (R_1^2 + R_1) \circ R_2'$.
  Hence, $R_2' \circ R_1 \subseteq (R_1 + Id)^3 \circ (R_2' + Id)$
- The last thing is to consider $R_2 \circ \{s(0)., s(a(0))\}$. Easily,
  $R_2 \circ \{s(0)., s(a(0))\} = \{s(b(b(0)))., s(b(a(0)))., s(a(0))., s(b(b(a(0))))., s(b(a(a(0))))., s(a(a(0))).\}$
  $\subseteq ([R_1] + Id)^2 \circ \{s(0)., s(a(0)).\}$

7.3 The Third Example: Rev–rev Problem

The next example is the "rev-rev" problem [2] stated in Prolog [5]. The problem is to show $P \models R_1$ where

$$P: \begin{cases} rev([\,],[\,]). \\ rev([A|X],Z) \leftarrow rev(X,Y), lins(A,Y,Z). \\ lins(A,[\,],[A]). \\ lins(A,[B|Y],[B|Z]) \leftarrow lins(A,Y,Z). \end{cases} \qquad R_1: \; rev(X,Y) \leftarrow rev(Y,X).$$

Let us assume two rev's on each side of R_1 to be different symbol but have same definition scheme. To show it clear, let us rewrite rev in the premise of R_1 as rev_P and add definition clauses of rev_P to P. Then the problem is rewritten as showing $\forall X, Y \; rev_P(X,Y) \rightarrow rev(Y,X)$ under the program

$$\begin{cases} rev([\,],[\,]). \\ rev([A|X],Z) \leftarrow rev(X,Y), lins(A,Y,Z). \\ rev_P([\,],[\,]). \\ rev_P([A|X],Z) \leftarrow rev_P(X,Y), lins(A,Y,Z). \end{cases} \quad \begin{cases} lins(A,[\,],[A]). \\ lins(A,[B|Y],[B|Z]) \leftarrow lins(A,Y,Z). \end{cases}$$

Using Theorem 6.4, the problem is reduced to showing $M_F\vartheta \subseteq M_Q$ and $Q \models R_2\vartheta$ where $\vartheta = \{rev_P(X,Y)/rev(Y,X)\}$ and

$$R_2: \begin{cases} rev_P([A|X],Z) \leftarrow rev_P(X,Y), lins(A,Y,Z). \\ lins(A,[B|Y],[B|Z]) \leftarrow lins(A,Y,Z). \end{cases} \qquad Q: \begin{cases} rev([\,],[\,]). \\ rev([A|X],Z) \leftarrow rev(X,Y), lins(A,Y,Z). \\ lins(A,[\,],[A]). \\ lins(A,[B|Y],[B|Z]) \leftarrow lins(A,Y,Z). \end{cases}$$

$$F: \begin{cases} rev([\,],[\,]). \\ lins(A,[\,],[A]). \end{cases}$$

Doing some simplification, we are going to solve

$$P: \begin{cases} rev([\,],[\,]). \\ rev([A|X],Z) \leftarrow rev(X,Y), lins(A,Y,Z). \\ lins(A,[\,],[A]). \\ lins(A,[B|Y],[B|Z]) \leftarrow lins(A,Y,Z). \end{cases} \qquad R': \; rev(X,[A|Z]) \leftarrow lins(A,Y,X), rev(Y,Z).$$

Since R' does have two atoms in its body, we cannot apply Theorem 4.16. We give a brief sketch of a proof here, because the method to be used here will be in the forthcoming paper.
($R' \diamond P \subseteq P^\circ \circ R' + P^\circ$)

$$\begin{aligned} R' \diamond \{&lins(A,[\,],[A])., lins(A,[B|Y],[B|Z]) \leftarrow lins(A,Y,Z).\} \\ &\diamond \{rev([\,],[\,])., rev([A|X],Z) \leftarrow rev(X,Y), lins(A,Y,Z).\} \\ = \{&rev([A],[A])., rev([C|X],[A|Z]) \leftarrow lins(A,U,X), rev(U,V), lins(C,V,Z).\} \end{aligned}$$

and

$$\begin{aligned} \{&rev([A|X],Z) \leftarrow rev(X,Y), lins(A,Y,Z).\} \\ &\diamond \{lins(A,[B|Y],[B|Z]) \leftarrow lins(A,Y,Z).\} \diamond R' \\ = \{&rev([A|X],[B|Z]) \leftarrow lins(B,V,X), rev(V,U), lins(A,U,Z).\} \end{aligned}$$

7.4 The Fourth Example: Last-is-a-member

This is an example of the problem that we have to use many heuristics without knowing Theorem 6.7 [5]. The problem is to show $\forall A, V, B \; rev(A,[V|B]) \rightarrow member(V,A)$ where rev and $member$ are defined as

$$P: \begin{cases} rev([\,],[\,]). \\ rev([A|X],Z) \leftarrow rev(X,Y), lins(A,Y,Z). \\ lins(A,[\,],[A]). \\ lins(A,[B|Y],[B|Z]) \leftarrow lins(A,Y,Z). \end{cases} \qquad R: \begin{cases} member(A,[A]). \\ member(A,[B|X]) \leftarrow member(A,X). \end{cases}$$

In the following, we show very brief and informal proof. The strategy is, first, to obtain a program to define a predicate $\exists B \, rev(A,[V|B])$, then, to apply Theorem 6.7 to it.

By Tamaki and Sato's unfold/fold transformation [8], we obtain a program defining

$p(V,A,B) \stackrel{\text{def}}{=} rev(A,[V|B])$ as $\begin{cases} p(V,[V],[\,]). \\ p(V,[A|X],Z) \leftarrow p(V,X,Y), lins(A,Y,Z). \end{cases}$

Our aim is, by transforming this, to obtain $p(V,A) \stackrel{\text{def}}{=} \exists B \, p(V,A,B)$ by a goal deletion and unfold/fold transformation. The transformation in brief proceeds as follows.

The initial program is $P_0: \begin{cases} C_1: p(A,[A],[\,]). \\ C_2: p(A,[B|X],Z) \leftarrow p(A,X,Y), lins(B,Y,Z). \\ C_3: lins(A,[\,],[A]). \\ C_4: lins(A,[B|Y],[B|Z]) \leftarrow lins(A,Y,Z). \end{cases}$

First definition is $D_1: \{C_5: p(A,X) \leftarrow p(A,X,Y)\}$ Then $P_1 = P_0 \cup \{C_5\}$.

Unfold C_5 by C_1 and C_2, we get $\begin{cases} C_6: p(A,[A]). \\ C_7: p(A,[B|X]) \leftarrow p(A,X,Y), lins(B,Y,Z). \end{cases}$

At this point, $P_2 = P_0 \cup \{C_5, C_6, C_7\}$ and $D_2 = D_1$. Clearly if we prove $M_{P_0} \models p(A,X,Y) \rightarrow \exists Z\, lins(B,Y,Z)$ then by a goal deletion and a folding we will get the aimed result from C_7 C_7' : $p(A,[B|X]) \leftarrow p(A,X)$. In order to apply Theorem 6.7 to prove the implication we first obtain programs defining

$$q(Y) \stackrel{\text{def}}{=} \exists A, X\, p(A,X,Y) \qquad l(B,Y) \stackrel{\text{def}}{=} \exists Z\, lins(B,Y,Z)$$

and then show $q(Y) \rightarrow l(B,Y)$.

The tranformation to obtain the programs proceeds schemetically as follows.

$$q(Y) \stackrel{\text{unfold}}{\Longrightarrow} \begin{cases} q([\,]). \\ q(Z) \leftarrow p(A,X,Y), lins(B,Y,Z). \end{cases} \stackrel{\text{fold}}{\Longrightarrow} q(Z) \leftarrow q(Y), lins(B,Y,Z).$$

$$l(B,Y) \stackrel{\text{unfold}}{\Longrightarrow} \begin{cases} l(B,[\,]). \\ l(B,[C|Y]) \leftarrow lins(B,Y,Z). \end{cases} \stackrel{\text{fold}}{\Longrightarrow} l(B,[C|Y]) \leftarrow l(B,Y).$$

Now let us go to prove $q(Y) \rightarrow l(B,Y)$. Adopting Theorem 6.7, what we have to show is

$$\{\text{definition of } l\} \models \begin{cases} l(D,[\,]). \\ l(D,Z) \leftarrow l(E,Y), lins(B,Y,Z). \end{cases}$$

The first clause is obvious. The second one is proved by showing $l(D,Z) \leftarrow l'(Z)$ where

$$l'(Z) \stackrel{\text{def}}{=} \exists B,E,Y\, ll(B,E,Y,Z). \quad ll(B,E,Y,Z) \stackrel{\text{def}}{=} l(E,Y), lins(B,Y,Z).$$

Again by unfold/fold tranformation, we get

$$ll(B,E,Y,Z) \stackrel{\text{unfolds}}{\Longrightarrow} \begin{cases} ll(B,E,[\,],[B]). \\ ll(B,E,[C|Y],[C|Z]) \leftarrow l(E,Y), lins(B,Y,Z). \end{cases} \stackrel{\text{fold}}{\Longrightarrow}$$

$$l'(Z) \stackrel{\text{unfold}}{\Longrightarrow} \begin{cases} l'([C]). \\ l'([C|Z]) \leftarrow ll(B,E,Y,Z). \end{cases} \stackrel{\text{fold}}{\Longrightarrow} \begin{matrix} ll(B,E,[C|Y],[C|Z]) \leftarrow ll(B,E,Y,Z). \\ l'([C|Z]) \leftarrow l'(Z). \end{matrix}$$

$l(D,Z) \leftarrow l'(Z)$ is proved by showing $\{\text{definition of } l\} \models \begin{cases} l(D,[C]). \\ l(D,[C|Z]) \leftarrow l(D,Z). \end{cases}$

Since the second clause is the same as the second one of the definition and the first clause is obtained by applying the second definition clause to the first definitions one, this obviously holds.

Hence, $M_{P_0} \models p(A,X,Y) \rightarrow \exists Z\, lins(B,Y,Z)$. Consequently, $p(X,Y)$ is defined by $\begin{cases} p(V,[V]). \\ p(V,[A|X]) \leftarrow p(V,X). \end{cases}$

Using Theorem 6.4, $\forall V, X\, p(V,X) \rightarrow member(V,X)$ if $R \models \begin{cases} member(V,[V]). \\ member(V,[A|X]) \leftarrow member(V,X). \end{cases}$

Obviously this holds.

8. Conclusion

We have shown some sufficient conditions for the validity of definite clauses, related them to resolutions between rules and shown their usefulness by examples.

The inductive proof which does not explicitly use induction is called inductionless induction [6]. The theorems we stated are not powerful to provide full inductionless induction on definite clauses, but will be the first step to implement it.

9. Acknowledgement

We would like to express our thanks to Mr. Yutaka Takuma, general manager of Naka Works, Hitachi Ltd., to Dr. Eiichi Maruyama, general manager of Advanced Research Laboratory, Hitachi Ltd., and to Mr. Masahiro Mori, manager of System and Software Design Department of Naka Works, Hitahci Ltd. for their giving us to conduct the research. We also thank Mrs. Mariko Sakurai for her encouragement and typing.

References

[1] K.R. Apt and M.H. van Emden, Contribution to the theory of logic programming, *J. ACM* **29** (1982) 841–862

[2] R.S. Boyer and J.S. Moore, Proving Theorems About LISP Functions, *J. ACM* **22** (1975) 129–144

[3] K.L. Clark, Predicate Logic as a Computational Formalism, *Research Monograph: 79/59*, TOC Imperial College (1979) 75–76

[4] M.H. van Emden and R.A. Kowalski, The semantics of predicate logic as a programming language, *J. ACM* **23** (1976) 733–742

[5] T. Kanamori and H. Fujita, Formulation of induction fomulas in verification of Prolog programs, *ICOT Technical Report TR-094* (1984)

[6] D. Kapur and D.R. Musser, Proof by Consistency, *Artificail Intelligence* **31** (1987) 125–157

[7] J.-L. Lassez and M.J. Maher, Closures and fairness in the semantics of programming logic, *Theoretical Computer Science* **29** (1984) 167–184

[8] H. Tamaki and T. Sato, Unfold/fold transformation of logic programs, *Proc. 2nd International Logic Programming Conference* (1984) 127–138