数理解析研究所講究録
第 618 巻 1987 年 1-15

Model Inference Incorporating Generalization

Hiroki ISHIZAKA (石坂裕毅)
IIAS-SIS, FUJITSU LIMITED
140, Miyamoto, Numazu,
Shizuoka 410-03, Japan

ABSTRACT

This paper is concerned with a strategy for inductive inference of logic programs. In the strategy an inference machine infers the head of a program clause by the least generalization of positive facts. Hence, the inference machine has only to enumerate the body of a program clause. This makes the inference machine efficient. However, if the inference machine computes the least generalization of all positive facts, it may occur that the atom becomes too general for the head of a program clause. The atoms which fall in such a situation are said to be "too general". In the strategy the main problem is to decide whether the least generalization of positive facts is too general or not. In this paper we shall formulate this problem and discuss its decidability.

1. Introduction

Model inference is a kind of inductive inference based on the first order logic. In this paper we shall restrict the domain of model inference to logic programs. On the domain, an inference machine receives as its input examples concerning some unknown model which are pairs of a ground atom and its truth value in the model, then produces as its output a logic program whose least Herbrand model is equal to the model.

In this paper, we assume that a first order language $L$ with finitely many predicate and function symbols is given. We shall identify a model with a subset of the Herbrand base B_L for $L$. The pairs of a ground atom in B_L and its truth value in a model are called *facts* about the model. The ground atom which has a truth value '*True*' is called a *positive fact*, while the ground atom which has a truth value '*False*' is called a *negative fact.* From the point of view of automated program synthesizing by examples, positive facts mean correct input/output samples, and negative facts mean incorrect ones. An *enumeration of a model M* is an effective enumeration of facts about M: F_1, F_2, . . . where any ground atom in B_L occurs in a fact F_i for some $i>0$. The output produced by inference machine is called a *conjecture*. We say that a conjecture is *too strong* iff the conjecture implies some negative facts, and *too weak* iff it cannot imply some positive facts.

Ehud Shapiro designed an efficient strategy for the model inference algorithm in 1981 [6]. Although his strategy is essentially based on the enumeration technique [1, 3] designed by Gold, it makes enumeration efficient by taking full advantage of the semantic and syntactic properties of logic. He constructed the Model Inference System (*MIS*) [7] based on his strategy. *MIS* is expected to be applicable to knowledge acquisition in the knowledge processing system such as an expert system, automated program synthesizing or program debugging and so forth.

However, there are some shortcomings in his strategy. For example, in his strategy, the initial hypothesis is the most general one, and the inference machine gradually makes it less general. A general hypothesis is only refuted by negative facts. Therefore in his strategy, negative facts are more useful than positive facts. But it is very important for inductive inference to use much information from positive facts. In this paper, on the contrary, we consider a model inference strategy incorporating Plotkin's least generalization [5] in which information from positive facts is more eagerly utilized. In order to elaborate the strategy we construct a tentative system *GEMINI* (GEneralization aided Model INductive Inference system) based on the strategy.

In the strategy, the inference machine will infer the head of a program clause by taking the least generalization from positive facts. Therefore, it has only to enumerate the body of a program clause. This makes the inference machine efficient. However, if the inference machine computes the least generalization of all positive facts, it may occur that the atom to be the head of a program clause becomes too general. We call such an atom "too general". The main problem of our strategy is to decide whether a resultant atom is too general or not. In the present paper we shall formulate this problem and present a sufficient condition for its decidability. The reader is assumed to be familiar with rudiments of logic programs [4].

2. The Least Generalization

First, we recall the notion of the least generalization by Plotkin [5].

A *word* is a literal or a term. An *index* is a finite sequence (possibly empty) of integers: $<i_1, i_2, \ldots, i_n>$. Let W be a word and t be a subword of W. The *index* I of t in W is defined inductively as follows:

(1) If $W=t$ then $I=<>$.

(2) If $W=\phi(t_1, t_2, \ldots, t_m)$ and the index of t in t_{i_1} $(1 \leqq i_1 \leqq m)$ is $<i_2, \ldots, i_n>$ then $I=<i_1, i_2, \ldots, i_n>$.

For example, in $p(a,b,g(X,Y))$, the index of X is $<3, 1>$, the index of b is $<2>$. $W(I)$ denotes the subword of W which has an index I.

Let t_1 be a subword of W_1 and t_2 be a subword of W_2. We say that t_1 and t_2 $(t_1 \neq t_2)$ are *replaceable* in W_1 and W_2 iff:

(1) The index of t_1 in W_1 is equal to the index of t_2 in W_2 and

(2) t_1 and t_2 begin with different function symbols or at least one of them is a variable.

We say W_1 is *more general* than W_2 iff $W_1\sigma = W_2$ for some substitution σ, and we denote it by $W_1 \leqq W_2$. For example, $p(a,g(X,Y)) \leqq p(a,g(h(W),U))$.

Two words are *compatible* iff they are both terms or have the same predicate symbol and sign.

Definition 1: Let W be a word and K be a set of words. W is the *least generalization* of K iff:

(1) For every V in K, $W \leqq V$.

(2) If for every V in K, $W_1 \leqq V$, then $W_1 \leqq W$.

Theorem 2 (Plotkin): Every non-empty set has the least generalization iff any two words in the set are compatible. Furthermore, if W_1 and W_2 are compatible then **Algorithm 1** terminates and computes the least generalization of $\{W_1, W_2\}$.

In our strategy, an inference machine, such as *GEMINI*, infers the head of a program clause by computing the least generalization of positive facts. At each step of inference, there exist only a finite number of known positive facts. Hence, the above theorem ensures that the inference machine is able to compute the least generalization of known positive facts.

```
*Input*: Compatible words $W_1$, $W_2$.
*Output*: The least generalization of $\{W_1, W_2\}$.
*Algorithm*:
    $V_1 := W_1$;  $V_2 := W_2$;
    *while* there exist $t_1$ and $t_2$ replaceable in $V_1$, $V_2$ *do*;
        choose a variable $X$ which does not occur in $V_1$, $V_2$;
        *while* there exists $I$ such that $V_1(I) = t_1$, $V_2(I) = t_2$ *do*
            $V_1 := X$;  $V_2 := X$
    output $V_1$  $(= V_2)$
```

Algorithm 1

3. An Outline of the Strategy

In this section we state an outline of our strategy via a simple example of *GEMINI's* behavior. It is an example of synthesizing a logic program for "append":

append([X|Y], Z, [X|U]) ← append(Y, Z, U).
append([ ], X, X).

We take a pure-Prolog program as an example of the logic program, and follow the DEC-10 Prolog in notational conventions [2].

Basically, the way how to modify a current conjecture is as follows. If a current conjecture is too strong then it should be specialized. *GEMINI* specializes a conjecture by adding a condition to the body of a clause in the conjecture. If a conjecture is too weak, then it is generalized by generalizing a clause in the conjecture. *GEMINI* usually replaces the head of a clause by a least generalization of uncovered positive facts in order to generalize the clause. When the clause is unit, it is sufficient to replace its head. When the clause is not unit, however, transformations of variables in the body of the clause must be considered. In this version, *GEMINI* generalizes a clause with a non-empty body if the resultant clause satisfies some conditions. Although the problem about generalization of the non-unit clause is important, the present paper is not concerned with the problem.

Figure 1 below shows how *GEMINI* modifies conjectures. In fact, each conjecture is a logic program, that is, a set of definite clauses. As a matter of convenience, the figure only shows alternations of a clause in the conjecture.

$$\begin{tikzcd}
\text{An enumeration of a model} & \text{Conjectures} \\
\text{(1) } <ap([a,b],[c],[a,b,c]), True> \arrow[dd] & \\
{} \arrow[r] & \text{(a) } ap([a,b],[c],[a,b,c]). \\
\text{(2) } <ap([a],[\,],[a]), True> \arrow[dd] & \\
{} \arrow[r] & \text{(b) } ap([a/X], Y, [a/Z]). \\
\text{(3) } <ap([a],[b],[a]), False> \arrow[dd] & \\
{} \arrow[r] & \text{(c) } ap([a/X], Y, [a/Z]) \leftarrow ap(X, Y, Z). \\
\text{(4) } <ap([1],[\,],[1]), True> & \\
\vdots \arrow[r] & \text{(d) } ap([A/X], Y, [A/Z]) \leftarrow ap(X, Y, Z).
\end{tikzcd}$$

Figure 1: An outline of modifications of conjectures

In the figure, first, a positive fact *ap([a,b], [c], [a,b,c])* is given. If a current conjecture cannot imply some positive facts then *GEMINI* modifies the conjecture by generalizing it. At this stage since the conjecture is not yet produced, *GEMINI* simply returns the received positive fact intact as its conjecture.

At the next stage, a positive fact *ap([a], [ ], [a])* is given and the current conjecture (a) cannot imply it. Then *GEMINI* returns *ap([a/X], Y, [a/Z])* the least generalization of (a) and the positive fact as its conjecture.

Next, a negative fact *ap([a], [b], [a])* is given and the current conjecture (b) implies it. In such a case, *GEMINI* usually adds an atom to the body of a clause in the conjecture in order to specialize the conjecture. In fact, *GEMINI* executes this operation by enumerating bodies and searching for an appropriate one.

Again, a positive fact *ap([1], [ ], [1])* is given, and the current conjecture (c) cannot imply it. In this case, *GEMINI* replaces the head of a clause by the least generalization of the fact and the head. Thus we will obtain the intended program clause for "append".

GEMINI basically modifies a conjecture in the above way. Although **Figure 1** shows a successful example, *GEMINI* may be in trouble in some cases. In the following section, we shall present such an example and point out the main problem in our strategy.

4. A Problem in the Strategy

Figure 2 below shows a failure example of a modification of a conjecture.

$$\begin{tikzcd}
\text{An enumeration of a model} & & \text{Conjectures} \\
(1)\ <ap([a,b],[c],[a,b,c]), True> \arrow[dd] & & \\
 & {} \arrow[r] & (a)\ ap([a,b],[c],[a,b,c]). \\
(2)\ <ap([a],[\,],[a]), True> \arrow[dd] & & \\
 & {} \arrow[r] & (b)\ ap([a|X],Y,[a|Z]). \\
(3)\ <ap([\,],[a],[a]), True> \arrow[dd] & & \\
 & {} \arrow[r] & (c)\ ap(X,Y,[a|Z]). \\
(4)\ <ap([a],[b],[a]), False> & & \\
\vdots & {} \arrow[r] & (d)\ ap(X,Y,[a|Z]) \leftarrow ???
\end{tikzcd}$$

Figure 2: An example of too general atoms

The figure is different from **Figure 1** at stage (3). A positive fact *ap([ ], [a], [a])* given at the stage cannot be implied from the current conjecture (b). Hence *GEMINI* returns the least generalization of (b) and the fact as its new conjecture.

At the next stage (4), a negative fact *ap([a], [b], [a])* is given, which can be implied from the current conjecture (c). Hence *GEMINI* will try to find a body in order to specialize the clause in the conjecture (c). But, in this case, it will not be able to find an appropriate body to specialize the clause. Because, in a successful recursive clause for "append" program, the first argument of *ap*(pend) in the head of the clause should be a compound term. In this paper, such an atom is said to be *too general.*

The main problems of our strategy are as follows.

(1) How does an inference machine identify too general atoms?

(2) How does it modify the conjecture when it has identified them?
The first problem is closely related to the power of the device which enumerates bodies of a clause. For the second problem, the current *GEMINI* partitions a set of known positive facts by using too general atoms, then computes the least generalizations of each set successively and returns them as its new conjectures.

In the following, we shall further discuss the above-mentioned something more detail.

Suppose that a target model M is partitioned as in **Figure 3**, where M_1 contains the elements of M whose first argument is not *[ ]*, and M_2 others. In such a case, the least generalization of M_1 will be *ap([X/Y], Z, [X/U])*, and the one of M_2 will be *ap([ ], X, X)*. In fact, they are both the heads of the clauses in the successful program for "append" given in the previous section.

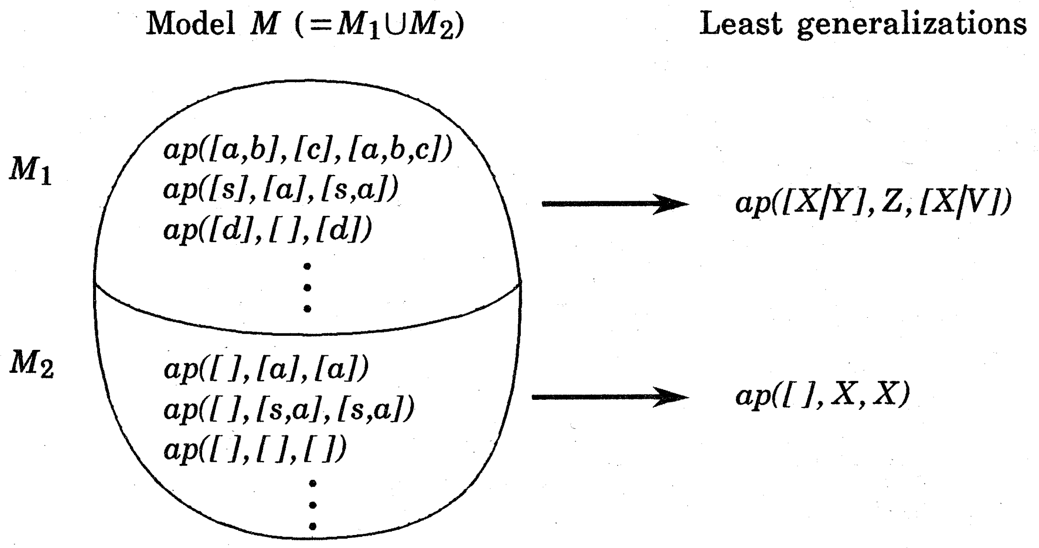

Figure 3: A partition of the model and the derived clause heads

In a practical model inference, however, the information about the boundary line in the model is not given. Hence the inference machine has to detect the boundary line for itself. *GEMINI* detects it using too general atoms. The way to do it is as follows.

For example, in **Figure 2**, the positive facts (1), (2), (3) are given and the least generalization of them becomes *ap(X, Y, [a/Z])*. Now suppose *GEMINI* can identify it to be too general in one way or another. Then *GEMINI* partitions the set of known positive facts into two subsets in which the least generalization for each subset is strictly less general than the too general atom *ap(X, Y, [a|Z])* (**Figure 4**). In the example, there are three

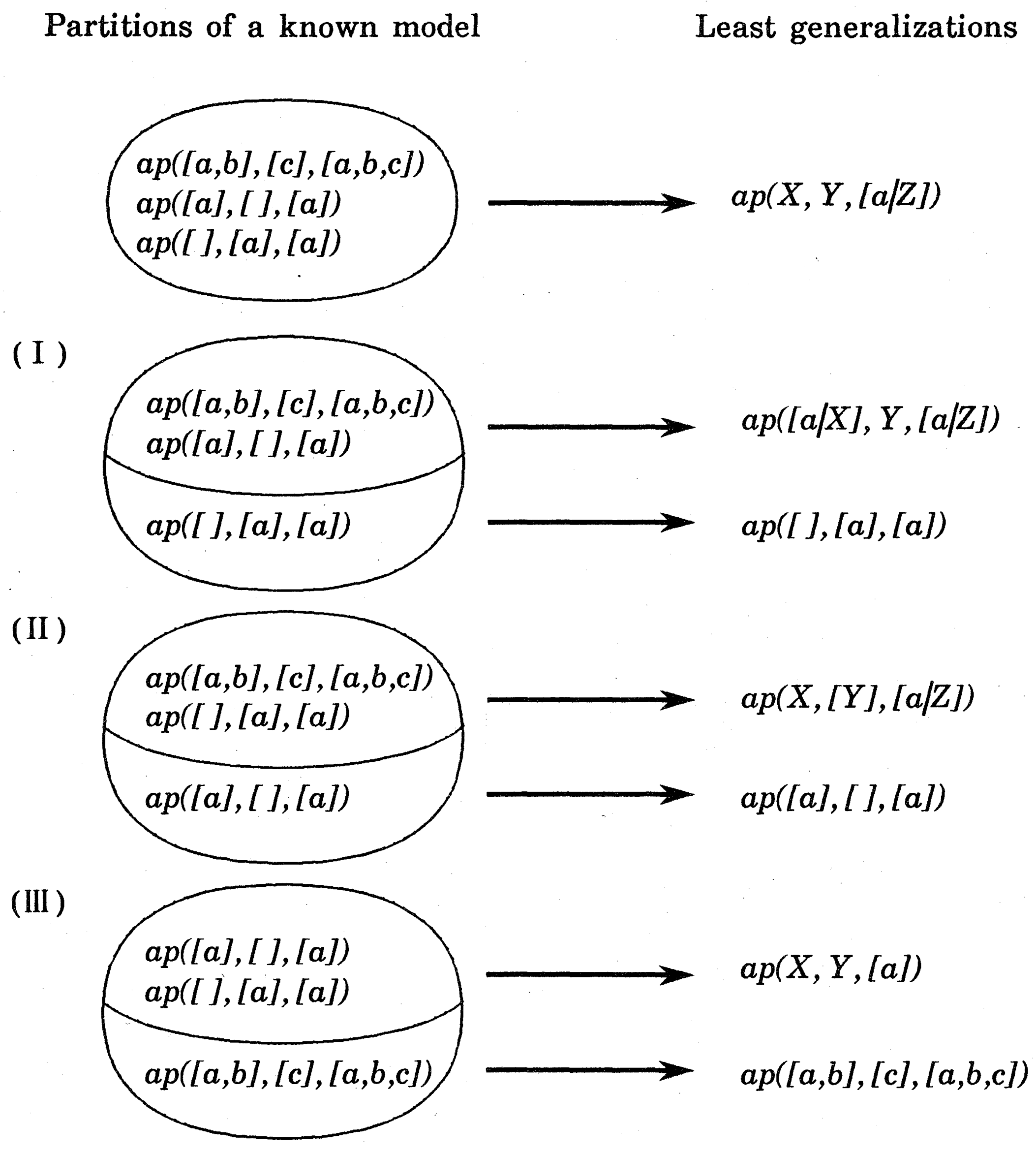

Figure 4: Partitions of the known model according to too general atom

cases. But in the second and the third cases, *ap(X, [Y], [a|Z])* and *ap(X, Y, [a])* are too general (because the both first arguments are not compound term). By the assumption, *GEMINI* will identify them as too general atoms. Hence *GEMINI* will reject these partitions and adopt the first partition eventually. In fact, this partition corresponds to the boundary line of the model in **Figure 3**.

In the result, the main problem of our strategy is to determine whether an inference machine can identify too general atoms or not. In the following two sections we shall formulate these problems and present conditions of its decidability.

5. The Head of a Program Clause

In this strategy, an inference machine takes the least generalization of a set of known positive facts as the head of a clause which should cover them. In general, however, the head of a clause does not need to be the least generalization. Hence, first, we shall show that the least generalization is sufficient to be the head of a clause.

Definition 3: Let M be a subset of B_L and A' be an ground atom. We say that a clause $A \leftarrow B_1, \ldots, B_n$ *covers* A' in M iff there is a substitution θ such that $A' = A\theta$ and $B_i\theta$ is in M for every i ($1 \leqq i \leqq n$).

M_C denotes the set of all ground atoms covered by the clause C in the model M. A clause C is *true* in M iff $M_C \subseteq M$, and *false* otherwise. A logic program P is *true* in M iff every clause in P is true in M.

Theorem 4: Let M be a model. For any clause C, there exists a clause C' whose head is the least generalization of M_C such that $M_{C'} = M_C$.

Proof: Suppose C be a clause of the form:

$$A \leftarrow B_1, B_2, \ldots, B_n. \quad (0 \leqq n)$$

and A' be the least generalization of M_C. Without loss of generality, we may assume that A' does not share variables with A. Since A' is the least generalization of M_C, we know that $A \leqq A'$. Hence there exists a substitution θ such that $A\theta = A'$, where we may assume that θ operates only variables occurring in A.

Let C' be $C\theta$. We show that $M_{C'} = M_C$.

For any α in M_C, there exists substitution σ such that $\alpha = A\sigma$ and $B_i\sigma$ is in M for $1 \leqq i \leqq n$, where we may assume σ operates only variables occurring in C. Since A' is the least generalization of M_C, there exists a substitution η such that $\alpha = A'\eta = A\theta\eta$, where we may assume that η operates only variables occurring in A'. Thus we have $\alpha = A\sigma = A\theta\eta$ for any α in M. From the assumptions for θ, σ and η, there exists a substitution η' such that $\theta\eta\eta' = \sigma$. Let $V(A)$ be a set of all variables occurring in A. Then η' obtained from σ by deleting any binding v/t for any v in $V(A)$. Hence there exists a substitution $\theta\eta\eta'$ such that, for any α in M_C, $\alpha = A\theta\eta\eta'$ and $B_i\theta\eta\eta'$ is in M for $1 \leqq i \leqq n$. Thus C' ($= C\theta$) covers α in M for any α in $M_{C'}$.

Conversely, for any α in $M_{C'}$, there exists a substitution σ such that $\alpha = A\theta\sigma$ and $B_i\theta\sigma$ is in M for $1 \leqq i \leqq n$. Thus C covers α in M for any α in $M_{C'}$. ∎

The above theorem shows that the head of a clause may be the least generalization of the set of ground atoms (a subset of a model) that should be covered by the clause. As we have seen in the previous section, *GEMINI* distinguishes such a subset by using too general atoms. Hence, *GEMINI* must be able to identify too general atoms. Now, we shall discuss the decidability of too general atoms.

6. Decidability of Too General Atoms

First, we shall define the notion of too general atoms.

Definition 5: Let A be an atom, P be a logic program and S be a non-empty set of logic programs.

(1) We say that A is *too general for* P iff for any clause C in P, $head(C) \leqq A$, where $head(C)$ denotes the head of C.

(2) We say that A is *too general for* S iff for any program P' in S, A is too general for P'.

For example, let P be the logic program:

$$insert(X, [Y|Z], [X,Y|Z]) \leftarrow X \leqq Y.$$
$$insert(X, [Y|Z], [Y|W]) \leftarrow Y \leqq X, insert(X, Z, W).$$
$$insert(X, [\,], X).$$

Then the atom such as "*insert(X, Y, [Z/W])*" or "*insert(X, [X/Z], [Y/W])*" is too general for P.

Now we shall define a clause enumerator, which produces candidates for target program. A *clause enumerator CE* is the device which receives an atom $A$ for its input and produces a sequence of the clauses of the form:

$$A \leftarrow B_1, B_2, \ldots, B_n. \quad (0 \leqq n)$$

as its output. *CE(A)* denotes the set of all clauses enumerated by *CE* for input $A$. $CE(A)_i$ denotes the $i$th clause in the enumeration. Furthermore, CE^* denotes the set of all clauses enumerated by *CE* for any input atom.

Definition 6: We say *CE* is *finite* iff *CE(A)* is finite for any atom $A$.

Definition 7: We say *CE* is *sufficient* iff for any atoms $A$ and A' if $A\theta = A'$ for some substitution θ which operates only variables occurring in $A$ then $CE(A)\theta \subseteq CE(A')$. Where $CE(A)\theta = \{C\theta | C \in CE(A)\}$.

Definition 8: We say *CE* is *complete for* a model $M$ iff there exists a finite subset $P$ of CE^* such that $M(P) = M$. Where $M(P)$ denotes the least Herbland model of $P$, that is, $M(P) = \cap\{M_i \subseteq B_L | P$ is true in $M_i\}$.

CE_M denotes a set of all finite subsets of CE^*, the least Herbland model of which is equal to $M$, that is, $CE_M = \{P \subseteq CE^* | P$ is finite and $M(P) = M\}$.

Lemma 9: Let $M$ be a model, $A$ be an atom and *CE* be a complete clause enumerator. If $A$ is too general for CE_M then any clause $C$ in *CE(A)* is false in $M$.

Proof: Assume that there exists a clause $C$ in *CE(A)* such that $C$ is true in $M$. Since *CE* is complete for $M$, there exists a program $P$ in CE^* such that $M(P) = M$. Put $P' = P \cup \{C\}$. Clearly $M(P') \supseteq M$. Since $C$ is true in $M$, P' is also true in $M$. Thus $M(P') \subseteq M$. Hence P' is in CE_M. Consequently, $A$ is not too general for CE_M. ∎

Theorem 10: Let $M$ be a model, $A$ be an atom and *CE* be finite and complete. Assume that **Algorithm 2** is given an enumeration of $M$. If $A$ is too general for CE_M then **Algorithm 2** terminates with output "$A$ is too general".

Input: An atom A and an enumeration of a model M.
Output: A sequence of clauses in $CE(A)$, or "A is too general"
Algorithm:

```
i:=1; S_True:={ }; S_False:={□};
repeat
    read the next fact <F, V>;
    S_V:=S_V∪{F};
    while there exists α in S_False such that CE(A)_i covers α in S_True do;
        i:=i+1;
        if CE(A)_i does not exist then
            output "A is too general"; HALT
    output CE(A)_i
forever
```

Algoritm 2

Proof: By **Lemma 9**, if A is too general for CE_M then every clause in $CE(A)$ is false in M. Hence, for any clause in $CE(A)$, the condition of the *while* loop will be satisfied after reading in some finite number of facts respectively, then the value of i will be increased. Since CE is finite, $CE(A)_i$ will be exhausted eventually. Thus **Algorithm 2** terminates with output "A is too general". ▮

Theorem 11: Let M be a model, A be an atom and CE be sufficient and complete. Assume that **Algorithm 2** is given an enumeration of M. If A is not too general for CE_M then the outputs of **Algorithm 2** converges a clause true in M.

Proof: Every clause $CE(A)_i$ which is false in M will be discarded at the *while* loop in some finite steps. But if $CE(A)_i$ is true in M then i will be never changed. Hence it is sufficient to prove that if A is not too general for CE_M then there exists a clause in $CE(A)$ which is true in M.

Since CE is complete for M, CE_M is not empty. If A is not too general for CE_M then there exists a program P in CE_M such that for some clause C in P, $head(C) \leqq A$. Let θ be a substitution which operates only variables occurring in $head(C)$ such that $head(C)\theta = A$. Since CE is sufficient, there exists a clause *C'* in *CE(A)* such that $C'=C\theta$, while *C* is true in *M* (=*M(P)*). Thus *C'* is true in *M*. ▮

The above two theorems show that the inference machine equipped with a finite, sufficient and complete clause enumerator can identify too general atoms. The sufficiency and the completeness of *CE* should be assumed necessarily in the enumeration strategy. The finiteness, however, seems to be so restricted. But we will be able to implement such a clause enumerator by restricting the size of the atom adding to the body of a clause, or the number of the occurrences of the same predicate symbol in the body.

7. Concluding Remarks

We have given an outline of model inference incorporating generalization, and discussed decidability on "too general" atoms. We have shown that the inference machine equipped with a finite, sufficient and complete clause enumerator can identify too general atoms. *GEMINI* is equipped with such a clause enumerator. *GEMINI* restricts an enumeration of whole clauses to one of bodies of a clause. The restriction leads to efficiency of an enumeration. For example, *GEMINI* can infer the program for the reverse of a list defined as follows:

reverse([X|Y], Z)←reverse(Y, U), append(U, [X], Z).
reverse([ ], [ ]).

This program cannot be inferred by *MIS*. Because in the first program clause, a compound term *[X]* occurs at the second argument of *append(__, __, __)* in the body, and the refinement operator employed in *MIS* [7] cannot produce such a clause. Of course the capacity of the refinement operator may be increased to enumerate such a clause. But the increase of the capacity will require a heavy cost of efficiency.

We conclude this paper by presenting some problems which should be considered for a future study.

(1) Partitions of a known model

In **Figure 4**, there are three partitions of a known model for three positive facts. In general, the number of partitions is $O(2^n)$ for n positive facts. Further, if the known model needs to be partitioned into three or more subsets then it will be more increasing. *GEMINI* tries to find out an appropriate partition among them by testing for the least generalization of each partitioned subset to be not too general. Hence the more the number of known positive facts will increase, the more *GEMINI* will be inefficient. But there are many partitions to be rejected without testing. For example, we shall denote a partition of M by (M_1, M_2), where M_1 and M_2 are subsets of M such that $M_1 \cup M_2 = M$ and $M_1 \cap M_2 = \phi$. Suppose there is some partition (M_1, M_2) of M such that the least generalization of M_1 is too general. Then, we know that every partition (M_1', M_2') such that $M_1 \subseteq M_1'$ should be rejected. Thus, we may expect to decrease the number of partitions of the known model.

(2) Generalization of a clause with non-empty body

As we have mentioned in section 3, the problem of generalization of a clause with non-empty body should be considered. Such a generalization cuts down much useless enumeration. For example, as in **Figure** 1, if the clause (c) is generalized to the clause (d) by a positive fact *ap([1], [ ], [1])*, then the enumeration of *CE(ap([A|X], Y, [A|Z]))* can be eliminated. We would like to investigate the problem of generalization of a clause with non-empty body by a positive fact.

ACKNOWLEDGEMENTS

The author wishes to acknowledge Prof. S. Arikawa, Kyusyu University, for many helpful suggestions and encouragement. He is also indebted to Dr. T. Kitagawa, the president of IIAS-SIS, and Dr. H. Enomoto, the director of IIAS-SIS, for their ceaseless support and advice.

REFERENCES

[1] Blum, L. ,Blum, M. : Toward a Mathematical Theory of Inductive Inference, Information and Control 28, 125-155, 1975.

[2] Clocksin, W. F. , Mellish, C. S. : Programming in Prolog, Springer-Verlag, 1981.

[3] Gold, E. M. : Language Identification in the Limit, Information and Control 10, 447-474, 1967.

[4] Lloyd, J. W. : Foundations of Logic Programming, Springer-Verlag, 1984.

[5] Plotkin, G. D. : A Note on Inductive Generalization, Machine Intelligence 6, Edinburgh University Press, 153-163, 1971.

[6] Shapiro, E. Y. : Inductive Inference of Theories From Facts, Technical Report 192, Yale University, 1981.

[7] Shapiro, E. Y. : Algorithmic Program Debugging, MIT Press, 1982.