Model Inference Using Bidirectional Refinements

Kawasaki, Youji
Department of Information Systems, Interdisciplinary Graduate School, of Engineering Science, Kyushu University

Shinohara, Takeshi
Department of Artificial Intelligence Kyushu Institute of Technology

Arikawa, Setsuo
Research Institute of Fundamental Information Science Kyushu University

https://doi.org/10.5109/3116

出版情報：Bulletin of informatics and cybernetics. 24 (1/2), pp.1-13, 1988-12-29. Research Association of Statistical Sciences
バージョン：
権利関係：

RIFIS-TR-CS-10

RIFIS Technical Report

Model Inference Using Bidirectional Refinements

Youji Kawasaki
Takeshi Shinohara
Setsuo Arikawa

December 29, 1988

Research Institute of Fundamental Information Science
Kyushu University 33
Fukuoka 812, Japan
E-mail: shino@iizuka.kyutech.junet Phone: 0948(28)5551

Model Inference Using Bidirectional Refinements

Youji KAWASAKI [†]

Department of Information Systems,
Interdisciplinary Graduate School of Engineering Science,
Kyushu University 39, Fukuoka 812, Japan

Takeshi SHINOHARA

Department of Artificial Intelligence,
Kyushu Institute of Technology, Iizuka 820, Japan

and

Setsuo ARIKAWA

Research Institute of Fundamental Information Science,
Kyushu University 33, Fukuoka 812, Japan

Abstract

Model inference is an inductive inference of theories from their models. In this paper, we propose a method of model inference for logic programs using both refinements in direction from general to specific and the opposite. By our method we can identify the target program from a program analogous to the target.

1 Introduction

Inductive inference problem is formalized as a process to identify an unknown rule from the examples of the facts implied by the rule. Gold[2] discussed inductive inference of languages and gave the criteria of successful inference widely known as the notion of "*identification in the limit.*" Blum and Blum[1] showed that any inferable class of recursive functions is characterized by a complexity measure. Model inference is an inductive inference introduced by Shapiro[8, 9], in which theories are inferred from the facts, that is, their models.

† Presently at NEC Corporation.
Mailing address: T. Shinohara
Department of Artificial Intelligence, Kyushu Institute of Technology, Iizuka 820, Japan
E-mail: shino@iizuka.kyutech.junet

The most naive, and essential in a sense, method of inductive inference is so called "*enumerative method*" or "*identification by enumeration.*" The method enumerates all possible hypothesis and outputs the first one that can explain all examples given so far. Clearly the enumerative method does not work efficiently. Shapiro[8, 9] discussed inductive inference of first order theories from their models and implemented the model inference system MIS. A feature of the method adopted in MIS is that it modifies the current hypothesis to obtain the next correct hypothesis. The notion of "*refinement*" is originally introduced by him, and it is used to modify hypothesis. By using refinement we can avoid to enumerate some of incorrect hypotheses. The initial hypothesis of MIS is $\{\Box\}$ representing a contradiction, and it is refined in direction from general to specific.

Laird[5, 6] discussed refinements in more abstract way. He considered refinements in direction from general to specific but also in the reverse direction. We call the former *downward* refinements and the latter *upward.* Ishizaka[4] pointed out that MIS lacks naturalness in the initial hypothesis and the direction of refinement and presented more efficient and natural method of model inference which utilizes the notion of least generalization by Plotkin.

No matter which direction of downward and upward is used, it is somewhat unnatural to adopt a refinement in one direction. In this paper we propose an inductive inference method using refinements in both directions and apply it to logic programs. The theory of analogy formalized by Haraguchi[3] suggests us to find a similar program to the target and adopt it as the initial hypothesis. When we have a program analogous to the target and try to modify it, we do not know which direction of refinement should be applied. Further, in some cases, we can not reach any correct program by refining the initial program in one direction. The method we propose can be applied to model inference problems in such a situation.

2 Preliminaries

In this section, we present some basic definitions on inductive inference and logic programs.

2.1 Logic Programs

First we briefly review logic programs and related notions according to Lloyd [7]. We assume basic terminologies on first order predicate logics. Throughout in this paper we assume that a first order language L has finitely many predicate symbols and function (including constant) symbols.

A *program clause* is a definite clause in L of the form

$$A \leftarrow B_1, B_2, \dots, B_n \ (n \geqq 0),$$

where A, B_1, B_2, ... , B_n are atomic formulas. We will use the word "atom" to abbreviate "atomic formula." The atom A of the program clause above is called the *head*. The sequence B_1, B_2, ... , B_n of atoms is called the *body*. If a program clause has no body, that is, if $n=0$, then it is called a *unit clause*. A *logic program* or simply *program* is a finite set of program clauses. A *goal clause* is a clause of the form

$$\leftarrow B_1, B_2, \dots, B_n \ (n \geqq 0).$$

We call a goal clause with $n=0$ a *empty clause*, and denote it by $\square$. A *Horn clause* is a program clause or a goal clause.

In this paper we deal with pure-Prolog as a logic programming language. We adopt the notation of DEC-10 Prolog. That is, variable symbols are denoted by capitalized names like X, Y, and predicate, function and constant symbols are denoted names starting with lower case letters like p, f, a. A term [] denotes a empty list, a term $[x_1, x_2, \dots x_n \mid y]$ denotes a list such that the first n elements are $x_1, x_2, \dots x_n$ and the remaining list is y. We take the set $\{\square\}$ as a special program.

The *Herbrand base* of L is the set of all ground atoms in L and it is denoted by B_L. A subset of B_L is called an *Herbrand interpretation*. An *Herbrand model* of a program P is an Herbrand interpretation that is a model of P. As is well-known, any program P has a unique least Herbrand model that is equal to the set of ground atoms implied by P.

2.2 Inductive Inference Problems and Refinements

Shapiro's model inference is defined as an inductive inference of first order theories[8, 9]. Laird[5, 6] discussed inductive inference problems more generally, and showed the usefulness of refinements. In this section we define inductive inference problems and related notions according to Laird.

Definition 2.1 An *inductive inference problem* is a 6-tuple $(D, d_0, \mathcal{E}, h, ASK, EX)$, where

· D is a finite or countable set of objects partially ordered by $\geqq$,

· d_0 is an element in D,

· $\mathcal{E}$ is a countable set of expressions,

· $h\colon \mathcal{E} \rightarrow D$ is a mapping from $\mathcal{E}$ onto D,

· ASK is an oracle which answers 1 if $h(e_1) \geqq h(e_2)$, 0 otherwise for any pair $(e_1, e_2) \in \mathcal{E} \times \mathcal{E}$, and

· *EX* is an oracle which returns a signed expression $+e$ or $-e$, if $d_0 \geqq h(e)$ or not, respectively.

D is called a *semantic domain* of objects, d_0 is called a *target*. When $h(e)=d$, e is called an *expression* of d and d is called a *semantics* of e. We denote the answers of *ASK* and *EX* by $ASK(e_1, e_2)$ and $EX()$, respectively. We call $+e$ a *positive example*, $-e$ a *negative example*.

Definition 2.2 The oracle *EX* gives a *sufficient presentation* of d_0 if the set $\{ e \in \mathcal{E} \mid h(e) \geqq x$ for all positive example $+x$ given by *EX* and $h(e) \not\geqq x$ for any negative example $-x \} = \{ e \in \mathcal{E} \mid h(e)=d_0 \}$.

An *inductive inference machine* is an effective procedure that receives inputs from time to time and produces outputs from time to time. An inductive inference machine *M identifies* d_0 *in the limit* if the sequence of outputs produced by *M* converges to e such that $h(e)=d_0$ whenever any sufficient presentation of d_0 is given by *EX*. An inductive inference machine *M identifies the set D in the limit* if *M* identifies any $d_0 \in D$ in the limit. The notion of "identification in the limit" is introduced by Gold[2], and it is widely accepted as a reasonable criterion of successful inductive inference.

If the set $\mathcal{E}$ is recursively enumerable and the oracle *EX* gives a sufficient presentation, we can easily solve the inductive inference problem by using a simple method called *enumerative method* or *generate and test*. Hereafter we assume the oracle *EX* gives a sufficient presentation. Such a simple method, however, does not work efficiently. If we have a binary relation on $\mathcal{E}$ that is reflected by h to the semantic relation $\geqq$, we can solve the problem more efficiently using it.

Definition 2.3 Let $\geqq_{\varepsilon}$ be an ordering of $\mathcal{E}$, $\geqq$ be a partial ordering of D, and $h: \mathcal{E} \to D$ be a mapping from $\mathcal{E}$ onto D. Then h is said to be an *order homomorphism* if $h(e_1) \geqq h(e_2)$ whenever $e_1 \geqq_{\varepsilon} e_2$.

Let the mapping h be an order homomorphism, and let e be the current hypothesis in an inference process. If we know $h(e) \geqq h(x)$ for some negative example $-x$, then we can neglect all expressions e' such that $e' \geqq_{\varepsilon} e$. Because $h(e') \geqq h(e) \geqq h(x)$ and $h(e')=d_0$ contradict $d_0 \not\geqq h(x)$. Further, if we know $h(e) \not\geqq h(x)$ for some positive example $+x$, then we need not examine any expression e' such that $e \geqq_{\varepsilon} e'$. Hereafter we assume the mapping h is an order homomorphism with respect to $\geqq$.

Definition 2.4 A *downward refinement* is a finitely axiomatizable binary relation ρ on $\mathcal{E}$ such that $e_1 \, \rho \, e_2$ implies $h(e_1) \geqq h(e_2)$. An *upward refinement* is a finitely axiomatizable binary relation γ on $\mathcal{E}$ such that $e_1 \, \gamma \, e_2$ implies $h(e_2) \geqq h(e_1)$. We denote the set $\{e' \mid e \, \rho \, e'\}$ by $\rho(e)$ and $\{e' \mid e' \, \gamma \, e\}$ by $\gamma(e)$. Similarly we denote the set $\{ e' \mid e \, \rho^* \, e'\}$ by $\rho^*(e)$ and $\{ e' \mid e' \, \gamma^* \, e\}$ by $\gamma^*(e)$, where ρ^* and γ^* are the reflexive transitive closures of ρ and γ, respectively.

Definition 2.5 A downward refinement ρ is called *complete for* $e \in \mathcal{E}$ if $h(\rho^*(e)) = \{ d \mid h(e) \geqq d\}$. An upward refinement γ is called *complete for* $e \in \mathcal{E}$ if $h(\gamma^*(e)) = \{ d \mid d \geqq h(e)\}$. A refinement is called simply *complete* if it is complete for any expression $e \in \mathcal{E}$.

Laird showed that an inductive inference problem can be solved whenever $\mathcal{E}$ is recursively enumerable, the oracle *EX* gives sufficient examples, and a complete refinement is available. As he pointed out, however, his method for general case does not seem to be natural, since it obtains expressions by not only refining but also enumerating. He also showed that some conditions on refinement are useful to make the inference method more efficient.

When $\mathcal{E}$ has a *top element* e_0 such that $h(e_0) \geqq h(e)$ for all $e \in \mathcal{E}$, any semantic object $d \in D$ can be obtained by repeatedly refining e_0 downward using a complete refinement ρ, and therefore we need not enumerates all expressions. Further, if the refinement ρ is *locally finite*, that is, if $\rho(e)$ is finite for any $e \in \mathcal{E}$, a simple queuing mechanism suffices us to obtain all refined expressions. The following procedure is given by Laird. Note that the existence of the top element e_0 in $\mathcal{E}$ and a locally finite complete downward refinement ρ is used to simplify the inference procedure. In the dual case, that is, in case of upward refinement, the similar method is applicable. More detailed discussions are found in the literatures[5, 6].

Procedure 1. (Inference by Downward Refinement with a Top Expression)

```
Input:   A recursively enumerable set $\mathcal{E}$ of expressions.
         A locally finite complete downward refinement $\rho$.
         A top element $e_0 \in \mathcal{E}$.
         An oracle $ASK$.
         An oracle $EX$ giving a sufficient presentation of $d_0$.
Output:  A sequence of expressions $H_1, H_2, \ldots$ , such that $H_i$ is correct for the first $i$
         examples given by $EX$.
Method: begin
           $Q \leftarrow$ emptyqueue
```

```
    $S \leftarrow$ emptyset (holding the set of examples)
    $H \leftarrow e_0$ (Start with the top element)
    do forever
    begin
      $S \leftarrow S \cup EX()$
      while $ASK(H, e) = 1$ for some $-e \in S$ or
              $ASK(H, e) = 0$ for some $+e \in S$ ($H$ is incorrect) do
      begin
        if $ASK(H, e) = 1$ for some $-e \in S$ and
          ASK$(H, e) = 1$ for all $+e \in S$ then
          Add $\rho(H)$ to the tail of $Q$ (queuing refined expressions of $H$)
        Remove the head element of $Q$, and let it be $H$
      end
      Output $H$
    end
  end.
```

3 Model Inference Using Bidirectional Refinements

Here we introduce a method of model inference for logic programs using bidirectional refinements. First we sketch the method in the context that bidirectional refinements should naturally be needed.

3.1 Model Inference Based on Analogy

We start this section with overviewing Shapiro's model inference method MIS [8, 9]. The initial hypothesis adopted by MIS is a top element $\{\square\}$ which represents a contradiction. MIS refines the current hypothesis downward, that is, enumerates logic programs in direction from general to specific. This feature might be somewhat strange in some case where we are inductively learning in a common sense, as Ishizaka[4] also pointed out. Ishizaka proposed an inference method which utilizes the notion of least generalization by Plotkin. As we have seen in the previous section, Laird[5, 6] focused on refinement and introduced some interesting refinements. Although he discussed not only downward refinement but also upward refinement, any inference procedure presented by him uses a refinement in one direction.

No matter which direction of refinements we use, it is not natural to start an inference with a program $\{\square\}$ or $\{\}$ as the initial hypothesis . In fact, to solve a problem using programs we will first try to find the target program in the library, and if we fail to find the exact program, then try to find a program analogous to the target and modify it . When we start with a program p_0 analogous to the target, we do not know in which direction, downward or upward, refinement should be applied to p_0, and in some cases we can not reach any correct program by refining p_0 in one direction. The method we will introduce here uses refinements of both directions at a time to deal with such a initial hypothesis. We can observe the difference between refinements in one direction and those in both directions from the following two examples of inferring a program to append two lists.

Example 3.1 (Refinement in Downward Direction)

Let take a program {□} as the initial hypothesis, and refine it downward. The following is a possible sequence of programs from {□} to the target. To avoid ambiguity we punctuate clauses in the set notation by semicolon.

{□}
{append(X, Y, Z)}
{append(X, Y, Z); append($[A|X], Y, [A|Z]$)}
{append($[], Y, Z$); append($[A|X], Y, [A|Z]$)}
{append($[], Y, Y$); append($[A|X], Y, [A|Z]$)}
{append($[], Y, Y$); append($[A|X], Y, [A|Z]$)←append(X, Y, Z)}

Example 3.2 (Refinement in Downward and Upward Directions)

Consider the following program to catenate terms as a program analogous to the target.

{cat($[], A, [A]$); cat($[U|X], Y, [U|Z]$)←cat(X, Y, Z)}

Let take the program obtained by replacing a predicate symbol "cat" by "append" as the initial hypothesis. Then refine it upward and then downward.

{append($[], A, [A]$); append($[U|X], Y, [U|Z]$)←append(X, Y, Z)}
{append($[], A, B$); append($[U|X], Y, [U|Z]$)←append(X, Y, Z)}
{append($[], A, A$); append($[U|X], Y, [U|Z]$)←append(X, Y, Z)}

Thus we can reach the target program by using refinements of both directions.

3.2 Refinements for Logic Programs

In the framework by Laird we should define a domain D of semantic objects, the set $\mathcal{E}$ of expressions, and an order homomorphism h: $\mathcal{E} \rightarrow D$. Let $\mathcal{E}$ be a set of logic programs, and D be the set of Herbrand interpretations. Let h be a mapping such that $h(e)$ is the set of ground atoms satisfied by a logic program e, that is, the minimal Herbrand model of e. We assume that the ordering of D is the ordinal set inclusion. Note that $\mathcal{E}$ has a maximum element $\{\square\}$, whose semantics is the set of all ground atoms, that is, the Herbrand base.

Definition 3.1 A *most general term* is a constant or a term of the form $f(x_1, x_2, \dots, x_n)$ and a *most general atom* is an atomic formula of the form $p(x_1, x_2, \dots, x_n)$, where f is an n-place function symbol, p is an n-place predicate symbol and $x_1, x_2, \dots, x_n$ are mutually different variable symbols.

Now we define a downward refinement and an upward refinement for logic programs, which are essentially the restrictions of Laird's refinements for more general class of logic programs. It should be noted that our class of logic programs is closed under the refinements defined below.

Definition 3.2 Let $P = \{ c_i \mid i=1, \dots, n \}$ be a logic program. Then $\rho(P)$ is the set of logic programs obtained by one of the following operations. For each operation i), $\rho_i(P)$ denotes the set of logic programs obtained by the operation i).

1) Delete a clause c_i.
2) Add the resolvent of c_i and c_j (i and j are possibly the same) to P.
3) Unify two variables in c_i, and add the result to P.
4) Substitute a most general term for a variable in c_i, and add the result to P.
5) If a clause c_i has a head then append a most general atom to the body, otherwise append it as the head.

Example 3.1 Let $P = \{p(X, Y) \leftarrow q(X);\ q(f(Z)) \leftarrow r(Z)\}$. Then $\rho(P)$, we have just defined, contains the following programs.

1) $\{p(X, Y) \leftarrow q(X)\}$
2) $\{p(X, Y) \leftarrow q(X);\ q(f(Z)) \leftarrow r(Z);\ p(f(A), B) \leftarrow r(A)\}$
3) $\{p(X, Y) \leftarrow q(X);\ q(f(Z)) \leftarrow r(Z);\ p(A, A) \leftarrow q(A)\}$
4) $\{p(X, Y) \leftarrow q(X);\ q(f(Z)) \leftarrow r(Z);\ q(f(g(A, B))) \leftarrow r(g(A, B))\}$
5) $\{p(X, Y) \leftarrow q(X);\ q(f(Z)) \leftarrow r(Z);\ q(f(A)) \leftarrow r(A), p(B, C)\}$

Theorem 1. ρ is a complete downward refinement for the set $\mathcal{E}$ of logic programs.

Proof. It is clear that ρ is a downward refinement, that is, $P_2 \in \rho(P_1)$ implies $h(P_1) \supseteq h(P_2)$.

We can easily prove the completeness of refinement ρ in a similar way to Laird's downward refinement for clause-form sentences. So, here we only give a brief sketch how a logic program equivalent to P_2 can be obtained by refining a logic program P_1, when assuming $h(P_1) \supseteq h(P_2)$.

If P_1 is inconsistent, then we can get empty clause $\square$ by using ρ_2 repeatedly. Since $\rho^*(\{\square\})$ contains any program, we have $P_2 \in \rho^*(P_1)$. Therefore we assume the consistency of P_1 without loss of generality. Let P_1 be a consistent program, $P_2 = \{ c_i \mid i=1, \dots , n \}$, and $h(P_1) \supseteq h(P_2)$. Then clearly $h(P_1) \supseteq h(\{c_i\})$ for any $i=1, \dots , n$. If we can show $P_1 \cup \{c_1\} \in \rho^*(P_1)$, then we can also show $P_1 \cup P_2 \in \rho^*(P_1)$ inductively, by deleting all clauses in P_1 from $P_1 \cup P_2$ we can find P_2 in $\rho^*(P_1)$. Therefore it is sufficient for us to show $P_1 \cup \{c\} \in \rho^*(P_1)$ for any clause c such that $h(P_1) \supseteq h(\{c\})$ and c is not a tautology.

Let $c = p(\bar{x}) \leftarrow q_1(\bar{x}), q_2(\bar{x}), \dots , q_n(\bar{x})$, where $\bar{x}$ denotes all variables in c. Then

$$\sim c = \exists \bar{x}(\sim p(\bar{x}) \wedge q_1(\bar{x}) \wedge q_2(\bar{x}) \wedge \dots \wedge q_n(\bar{x})).$$

Let $\bar{s}$ be Skolem constants corresponding to $\exists \bar{x}$ and substitute $\bar{x}$ by $\bar{s}$. The resulted formula is

$$\sim c[\bar{s}] = \sim p(\bar{s}) \wedge q_1(\bar{s}) \wedge q_2(\bar{s}) \wedge \dots \wedge q_n(\bar{s}).$$

Since we assume $h(P_1) \supseteq h(\{c\})$, there exists a derivation of the empty clause $\square$ from P_1 and $\sim c[\bar{s}]$, that is, from $P_1 \cup \{\leftarrow p(\bar{s}); q_1(\bar{s}) \leftarrow; q_2(\bar{s}) \leftarrow; \dots ; q_n(\bar{s}) \leftarrow\}$. Consider the resolution proof tree, as in Fig.1, whose nodes are labeled by clauses. If a clause c_3 is derived from c_1 and c_2 in a derivation, then corresponding nodes N_1, N_2 and N_3 are labeled by c_1, c_2 and c_3, respectively, and N_1 and N_2 are children of N_3. The label of every leaf is a clause in $P_1 \cup \{\leftarrow p(\bar{s}); q_1(\bar{s}) \leftarrow; q_2(\bar{s}) \leftarrow; \dots ; q_n(\bar{s}) \leftarrow\}$. The root is labeled by the empty clause $\square$. We call such a label of node N an *r-clause* and denote it by $r(N)$.

Based on the resolution proof tree, we construct a refinement path from P_1 to $P_1 \cup \{c\}$. First we label each node by an additional clause called x-clause. The *x-clause* of node N, denoted by $x(N)$, is defined inductively from leaves to the root.

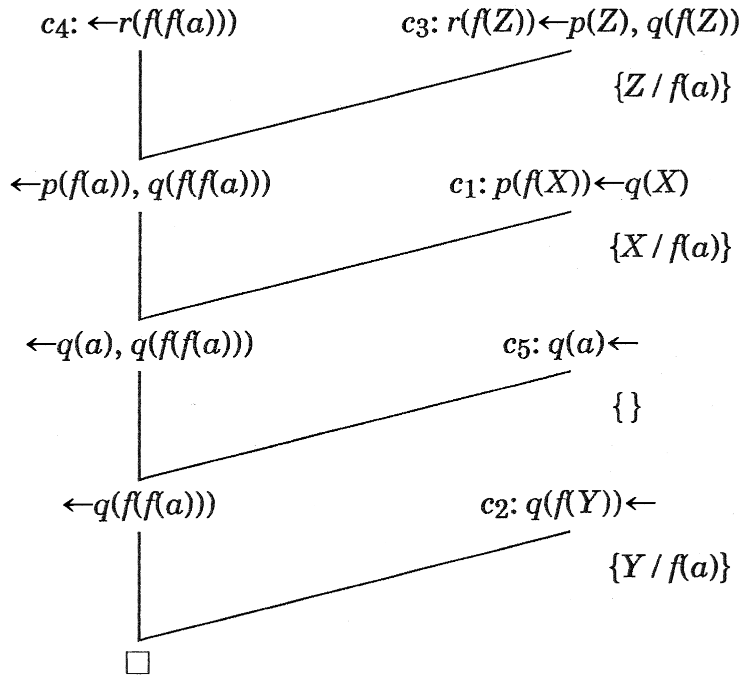
a)If N is a leaf, then $x(N)[\bar{s}] = r(N)$.

b)If exactly one child node N_1 of N is a leaf such that $r(N_1) \in \sim c[\bar{s}]$, then $x(N)[\bar{s}] = \theta[x(N_2)]$, where N_2 is another child of N and θ is the unifier used to resolve $r(N_1)$ and $r(N_2)$.

c) If no child of N has any clause in $\sim c[\bar{s}]$ as its r-clause, then $x(N)[\bar{s}]$ is the resolvent of $x(N_1)[\bar{s}]$ and $x(N_2)[\bar{s}]$ using the same atom and unification as in the derivation of $r(N)$ from $r(N_1)$ and $r(N_2)$.

d) If both of the children of N are labeled by r-clauses from $\sim c[\bar{s}]$, then $r(N)$ should be the empty clause $\square$, it contradicts the assumption that c is consistent. Therefore we can ignore this case.

In Fig.2 x-clauses of the tree in Fig.1 are illustrated, where underlined clauses are from $\sim c$ and marked clauses by b) and c) are defined by the respective rule.



$P_1 = \{c_1\!: p(f(X))\leftarrow q(X);$
$\quad c_2\!: q(f(Y))\leftarrow;$
$\quad c_3\!: r(f(Z))\leftarrow p(Z), q(f(Z))\,\}$

$c = r(f(f(U)))\leftarrow q(U), s(U, V)$
$\sim c[\bar{s}] = \{\, c_4\!: \leftarrow r(f(f(a)));$
$\quad c_5\!: q(a)\leftarrow;$
$\quad c_6\!: s(a, b)\}$

Fig. 1. A Resolution Proof Tree

We can easily observe that each x-clause is obtained by using ρ, the x-clause of the root node is a subclause of c, and therefore the program $P_1 \cup \{c\}$ is constructed by the refinement. For example, for the logic program $P_1 = \{\, c_1\!: p(f(X))\leftarrow q(X);\ c_2\!: q(f(Y))\leftarrow;\ c_3\!: r(f(Z))\leftarrow p(Z), q(f(Z))\,\}$ and the program clause $c = r(f(f(U)))\leftarrow q(U), s(U, V)$ in Fig.1 and Fig.2, the refinement ρ_3 can add

$$x_1 = c_3[Z/f(U)] = r(f(f(U)))\leftarrow p(f(U)), q(f(f(U))),$$

then ρ_2 may add the resolvent of x_1 and c_1

$$x_2 = r(f(f(U)))\leftarrow q(U), q(f(f(U))),$$

and the resolvent of x_2 and c_2

$$x_3 = r(f(f(U)))\leftarrow q(U),$$

ρ_5 can add a clause

$$x_4 = r(f(f(U)))\leftarrow q(U), s(U, V) = c,$$

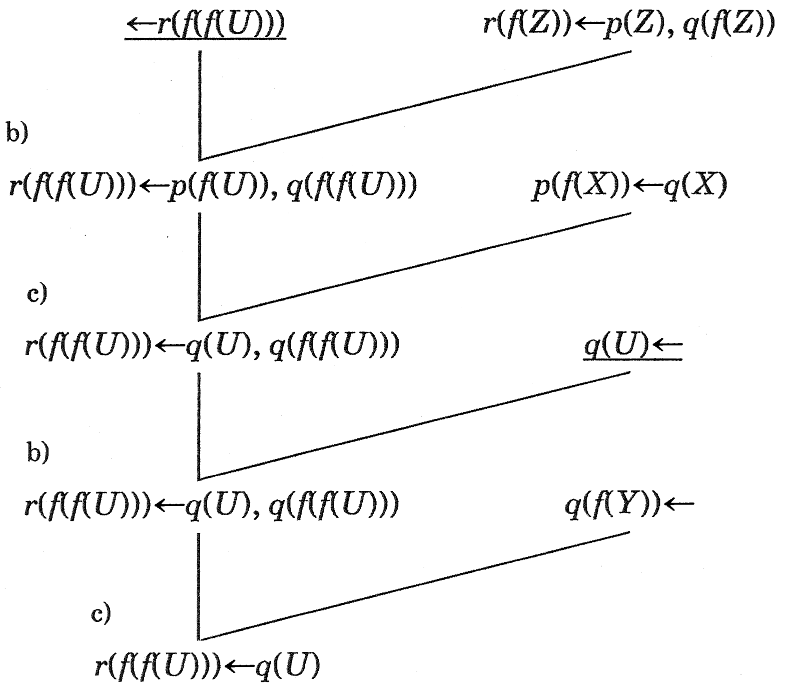

←r(f(f(U)))
r(f(Z))←p(Z), q(f(Z))
b)
r(f(f(U)))←p(f(U)), q(f(f(U)))
p(f(X))←q(X)
c)
r(f(f(U)))←q(U), q(f(f(U)))
q(U)←
b)
r(f(f(U)))←q(U), q(f(f(U)))
q(f(Y))←
c)
r(f(f(U)))←q(U)

Fig. 2. *X*-clauses of a Resolution Proof Tree

which is the result of appending a most general atom $s(U, V)$ to the body of x_3. Thus we can get a program $\{c_1, c_2, c_3, x_1, x_2, x_3, x_4 = c\}$ in $\rho^*(P_1)$. Since we can delete any clause by ρ_1, finally we have the logic program $\{c_1, c_2, c_3, c\}$. □

Definition 3.3 Let $P = \{c_i \mid i=1, \dots, n\}$ be a logic program. Then $\gamma(P)$ is the set of logic programs obtained by one of the following operations.

1) Add a ground clause to P.
2) Let $c_i = A \leftarrow A_1, A_2, \dots, A_n$. Select an atom B arbitrarily, and replace c_i by two clauses
   $A \leftarrow A_1, A_2, \dots, A_n, B$ and
   $B \leftarrow A_1, A_2, \dots, A_n$. (anti-resolution)
3) Replace some occurrences of a variable in c_i by a new variable. (anti-unification)
4) Replace some occurrences of a most general term t in c_i such that no variable in t occurs other than in t by a new variable. (anti-substitution)
5) If a clause c_i has a body then remove an atom from the body, otherwise remove the head (the result of removing head is the empty clause □).

Example 3.2 Let $P = \{p(X, Y) \leftarrow q(X);\ q(f(Z)) \leftarrow r(f(Z))\}$. Then $\gamma(P)$ contains the following programs.

1) $\{p(X, Y)\leftarrow q(X);\ q(f(Z))\leftarrow r(f(Z));\ r(a)\leftarrow r(f(a)), p(f(f(b)))\}$
2) $\{p(X, Y)\leftarrow q(X);\ q(f(Z))\leftarrow r(f(Z)), r(f(a));\ r(f(a))\leftarrow r(f(Z))\}$
3) $\{p(X, Y)\leftarrow q(U);\ q(f(Z))\leftarrow r(f(Z))\}$ (Note that the first occurrence of X is not replaced.)
4) $\{p(X, Y)\leftarrow q(X);\ q(f(Z))\leftarrow r(U)\}$
5) $\{p(X, Y)\leftarrow;\ q(f(Z))\leftarrow r(f(Z))\}$

We can easily show the completeness of γ also in a similar way to Laird.

Theorem 2. γ is a complete upward refinement for the set $\mathcal{E}$ of logic programs.

3.3 Inference Procedures

Now we present our inference procedure using two refinements ρ and γ. In the procedure, we use a dovetailing technique to refine programs where $\rho(P, n)$ and $\gamma(P, n)$ denote the programs obtained right after the n-th computation step of $\rho(P)$ and $\gamma(P)$, respectively for a program P and an positive integer n. Note that $\rho(P, n)$ and $\gamma(P, n)$ may not be any program for a particular n.

Procedure 2. (Model Inference Using Bidirectional Refinements)

```
Input:   A logic program $P_0 \in \mathcal{E}$.
         An oracle $ASK$.
         An oracle $EX$ giving a sufficient presentation of the target $d_0$.
Output:  A sequence of logic programs $H_1, H_2, \ldots$, such that $H_i$ is correct for the
         first $i$ examples given by $EX$.
Method: begin
          $Q :=$ emptyqueue
          $S :=$ emptyset
          $H := p_0$
          do forever
          begin
            $S := S \cup EX()$
            while $ASK(H, e)=0$ for some $+e \in \mathcal{E}$ or
                    $ASK(H, e)=1$ for some $-e \in \mathcal{E}$ do
            begin
              if $ASK(H, e)=0$ for some $+e \in \mathcal{E}$ then
                    Add ['u', $H$, 1] to $Q$ (Dovetail upward refinement)
              if $ASK(H, e)=1$ for some $-e \in \mathcal{E}$ then
                    Add ['d', $H$, 1] to $Q$ (Dovetail downward refinement)
```

```
        $H := NEXT()$
      end
      Output $H$
    end
  end
  where $NEXT()$ is:
    repeat
      Remove the head element of $Q$, and let it be $[w, P, n]$
      Add $[w, P, n+1]$ to $Q$ (to continue the dovetailing)
      if $w =$ 'u' then
          $NEXT := \gamma(P, n)$
      else
          $NEXT := \rho(P, n)$
    until $NEXT$ has a value.
```

We can easily show that the inference procedure above correctly infers the target program.

Theorem 3. Procedure 2 identifies d_0 in the limit.

Dovetailing in Procedure 2 seems to make the inference process inefficient. To simplify the procedure we need some restrictions on refinements. As we have seen in Section 2, when the refinement is locally finite the dovetailing mechanism can be replaced by a simple queuing one. Using one of the upward and downward refinements, we need the completeness of the refinement to gualantee the proceduer correctly infers. On the contrast, using bidirectional refinements at a time, we need not the completeness of the both refinements. From our observation the following theorem is obvious.

Theorem 4. Procedure 2 identifies d_0 in the limit if the downward refinement ρ is complete and the upward refinement γ is reachable to the top element $\{\Box\}$.

Our downward refinement ρ is locally finite since $\mathcal{E}$ contains at most finitely many function symbols and predicate symbols. On the other hand, our upward refinement is not locally finite under the assumption. Because the number of ground clauses to be added by γ_1 is infinite and the number of atoms we can select in γ_2 is also infinite. Since we apply upward refinement to a program P only when P can not explain some positive example, the ground clause to be added might be one of such examples. Even if we always select a most general atom in anti-resolution, the atom can be refined to any atom by downward refinement. Thus we have a modified upward refinement.

Definition 3.4 Let $p = \{ c_i \mid i=1, \dots , n \}$ be a logic program, and S be the set of examples given by EX so far. Then $\hat{\gamma}(p)$ is the set of logic programs obtained by one of the following operations.

1) Add a ground atom e such that $+e \in S$ and $ASK(p, e)=0$ to p.
2) Let $c_i = A \leftarrow A_1, A_2, \dots , A_n$. Select a most general atom B arbitrarily, and replace c_i by two clauses
   $A \leftarrow A_1, A_2, \dots , A_n, B$ and
   $B \leftarrow A_1, A_2, \dots , A_n$. (anti-resolution)
3) Replace some occurrences of a variable in c_i by a new variable. (anti-unification)
4) Replace some occurrences of a most general term t in c_i such that no variable in t occurs other than in t by a new variable. (anti-substitution)
5) If a clause c_i has a body then remove an atom from the body, otherwise remove the head (the result of removing head is the empty clause $\Box$).

Theorem 5. $\hat{\gamma}$ is a locally finite (but not complete) upward refinement. $\hat{\gamma}$ is reachable to $\{\Box\}$.

Now we have a simplified model inference procedure corresponding to Procedure 1 in section 2.

Procedure 3. (Model Inference Using Locally Finite Bidirectional Refinements)

Input: A logic program $p_0 \in \mathcal{E}$.
An oracle ASK.
An oracle EX giving a sufficient presentation of the target d_0.

Output: A sequence of logic programs H_1, H_2, ... , such that H_i is correct for the first i examples given by EX.

Method:
```
begin
    $Q$ := emptyqueue
    $S$ := emptyset
    $H$ := $p_0$
    do forever
    begin
      $S := S \cup EX()$
      while $ASK(H, e)=0$ for some $+e \in \mathcal{E}$ or
              $ASK(H, e)=1$ for some $-e \in \mathcal{E}$ do
    begin
      if $ASK(H, e) = 0$ for some $+e \in \mathcal{E}$ then
          Add $\gamma(H)$ to the tail of $Q$
      if $ASK(H, e) = 1$ for some $-e \in \mathcal{E}$ then
          Add $\rho(H)$ to the tail of $Q$
      Remove the head element of $Q$, and let it be $H$
    end
    Output $H$
  end.
```

4 Concluding Remarks

We have discussed the method of model inference using bidirectional refinements. By the method we can identify a target program from an initial program analogous to the target. However, we have not yet an essential problem on what criteria we should select an analogous program to the target. Although we improve the efficiency by restricting refinements to be locally finete, further studies should be needed to apply our method to practical problems.

References

[1] Blum, L., Blum, M.: Toward a Mathematical Theory of Inductive Inference, Inf. & Contr. vol.28, 125 - 155, 1975.

[2] Gold, E.M.: Language Identification in the Limit, Inf. & Contr. vol.10, 447 - 474, 1967.

[3] Haraguchi, M.: Towards a Mathematical Theory of Analogy, Bull. Inf. & Cybern., vol.21, 29 - 56, 1985.

[4] Ishizaka, H.: Model Inference Incorporating Generalization, Proc. Symp. Software Sci. Engineering, Kyoto, 1986.

[5] Laird, P.D.: Inductive Inference by Refinement, YALEU / DCS / TR - 376, 1986.

[6] Laird, P.D.: Learning From Good Data and Bad, YALEU / DCS / TR - 551, 1987.

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

[8] Shapiro, E.Y.: Inductive Inference of Theories From Facts, YALEU / DCS / TR -192, 1981.

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