Inductive Inference of Prolog Programs with Linear Data Dependency from Positive Data

Arimura, Hiroki
Department of Artificial Intelligence, Kyushu Institute of Technology

Shinohara, Takeshi
Department of Artificial Intelligence, Kyushu Institute of Technology

出版情報：RIFIS Technical Report. 70, 1993-05-14. Research Institute of Fundamental Information Science, Kyushu University
バージョン：
権利関係：

RIFIS-TR-CS-70

RIFIS Technical Report

Inductive Inference of Prolog Programs with Linear Data Dependency from Positive Data

Hiroki Arimura
Takeshi Shinohara

May 14, 1993

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

E-mail: arim@ai.kyutech.ac.jp Phone: 0948-29-7638

Inductive Inference of Prolog Programs with Linear Data Dependency from Positive Data

Hiroki ARIMURA **Takeshi SHINOHARA**

Department of Artificial Intelligence

Kyushu Institute of Technology

Kawazu 680-4, Iizuka 820, JAPAN

arim@ai.kyutech.ac.jp

Abstract In this paper, we study inductive inference of a subclass of Prolog programs from positive examples. The subclass, called *linearly covering programs*, allows shared variables occurring only in the bodies of a clause, which are excluded in the class of linear Prologs already known to be inferable from positive data. In a linearly covering programs, any data passing between subgoals preserves the total size of data contents from a data-dependency constraint. Using Shinohara's work on the inferability of concept defining framework, we prove that for every fixed $k,m > 0$, if the length of subgoals in a clause is bounded by a k, the class of linearly covering programs containing at most m definite clauses is inferable from positive data, without any oracle for auxiliary predicates. The result gives a partial answer to the theoretical term problem in model inference. Furthermore, we show that the restriction on the length of bodies is necessary for the inferability.

1. Introduction

In this paper, we study the inductive inferability of a class of Prolog programs. The problem considered in this paper is to identify an unknown Prolog program from the examples of the least Herbrand model of the unknown program. The problem is called *inductive inference of Prolog programs* or *model inference*. In particular, this paper deals with the problem of *inductive inference of Prolog programs from positive data*, where examples are given as an infinite sequence of *positive examples*, that is, ground atoms which is true in the least Herbrand model of the unknown program.

Since the study of Model Inference System (MIS, for short) by Shapiro [Sha82], inductive inference of Prolog programs has been considered as a framework to unify concept learning and automated program synthesis. In these practical applications we want to apply inductive learning mechanism, negative examples are often hard to obtain. Therefore, it is natural requirement that inference mechanism works only from positive data.

On the other hand, there is a difficulty, called *theoretical term generation problem*, in inference of Prolog programs from examples [Ban88, Ish90, Mug90]. Assume that we infer a Prolog program that defines reversal of a list from an infinite sequence

rev([];[]), rev([a,b];[b;a]), rev([a,b,c];[c,b,a]), . . .

of atoms that is true in the intended meaning of the predicate *rev*(,). The difficulty is that to infer a whole program,

```
rev([]; []).
rev([A | B]; D) ←
        rev(B; C),
        append(C, [A]; D).
```

where the semicolon ";" is used to separate input and output arguments of a predicate, we have to discover an auxiliary predicate *append*(, ,) and reconstruct the meaning of *append*(, ,) from examples of *rev*(,). The problem is closely related to the inferability from positive data.

For the reasons described above, studies of inference from positive data is important from practical point of view. In the early stage of studies on inductive inference from positive data, Gold [Gol67] showed that even the class of regular languages, which is a very simple class of languages, is not inferable from positive data. This indicates that the whole class of Prolog programs is not inferable from positive data. Shinohara [Shi91] showed that the class of *linear Prolog*, which was introduced by Shapiro for his MIS, is inferable from positive data when the number of clauses in a program is bounded. The result also says that programs in the class are inferable from positive examples only of a target predicate. It gives a partial answer to the theoretical term generation problem.

However, linear programs are too restricted for practical applications in automated program synthesis. Linear programs have a restriction on their syntax of clauses, called variable-boundedness, that any variable occurring in the body of a clause must occurs in the head. Because of the syntactic restriction, linear programs do not allow the use of shared variables to pass intermediate data computed by a subgoal on other subgoals.

In this paper, we propose another subclass of Prolog programs, called *linearly covering programs*, which allows shared variables occurring only in the bodies of a clause. In the class, predicates are augmented with a declaration that indicates *input and output mode.*

First, in Section 2, we give basic definitions on logic programming, inductive inference and other concepts and notations that are used in later sections. We introduce Shinohara's *concept defining frameworks* [Shi91] that we use to prove the main result. Next, in Section 3, we define the class of linearly covering programs

through a data-dependency constraint on input and output arguments as any data passing between subgoals in a clause does not increase the total size of data contents.

From the constraint, programs P in the class have a number of good properties. The problem of deciding whether a program P is in the class is polynomial time decidable. Any succeeded SLD-derivation from a goal with ground input arguments returns ground output arguments. Further, the size of output is at most that of input. There is an effective decision for the membership of a ground atom in the least Herbrand model $M(P)$. In Section 3 and the next Section 4, we prove these good properties for the class.

Then, in Section 4, we show that if the maximum length of subgoals in a clause is bounded by a fixed constant $k > 0$, the class is inferable from positive examples of a target predicate. Furthermore, we show that the restriction on the length of bodies is necessary for the inferability. Finally, we summarize the results in Section 5.

2. Preliminaries

First, we introduce our language for logic programs and their semantics. Definitions and terminology that are not given here may be found in Lloyd [Llo87].

The *alphabet of a first order language* is a triple $L = (\Pi, \Sigma, X)$, where Π, Σ and X are disjoint sets and Π and Σ are finite. We call elements of Π, Σ, X *predicate symbols, function symbols, variables*, respectively. Each symbols in Π and Σ is associated with a nonnegative integer called an *arity*.

A *term* is a variable, a function symbol of arity 0, or an expression of the form $f(t_1,...,t_n)$, where f is a function symbol of arity $n \geq 0$ and $t_1,...,t_n$ are terms. We denote sequences of terms by bold face letters $\mathbf{s},\mathbf{t},\mathbf{u},\mathbf{t}_1,\mathbf{t}_2,...$.

In practical applications of logic programs, predicate symbols are often augmented with a mode declaration. For example, a mode declaration append(+,+,-) means that the first and the second argument of a predicate append are used as input and the third argument is used as output. For simplicity, we assume that all input arguments are preceding output arguments, and we write an atom with a mode declaration using a semicolon ";" as a punctuation symbol between input and output arguments.

Definition. An atom is an expression of the form $p(s_1,...,s_m;t_1,...,t_n)$ or $p(\mathbf{s};\mathbf{t})$, where sequences of terms $\mathbf{s}=s_1,...,s_m$ and $\mathbf{t}=t_1,...,t_n$ are called the *input* and the *output arguments*, respectively.

A *goal* is a clause of the form $\leftarrow A_1,...,A_n$ and a *definite clause* is a clause of the form $A \leftarrow A_1,...,A_n$ $(n \geq 0)$, where n is called the body-length. A *program* is a finite set

P of definite clauses. We denote by $G_1 + G_2$ the concatenation of goals G_1 and G_2. Note that we do not distinguish two goals with different order of atoms.

Let L be an alphabet of first order language and P be a program. Then, we denote by $B(L)$ the *Herbrand base*, the set of all the atoms on L, by $M(P)$ the *least Herbrand model* of P, and by $M(p, P)$ the meaning of an n-ary predicate p defined by P,

$$M(p, P) = \{\ p(t_1,\ldots,t_n) \mid\ p(t_1,\ldots,t_n) \in M(P)\ \}.$$

SLD-resolution gives the procedural semantics of logic programs [Llo87].

Theorem 2.1. ([Llo87]) For an atom a, $a \in M(P)$ iff there is a SLD-refutation from $\leftarrow a$.

Next, we define the multiset inclusion $\subseteq$ *and* a mapping $[\cdot]$ from terms to multisets of symbols. Using a data-dependency based on the preorder induced by them, we introduce a subclass of logic programs in the next section.

Multisets are corrections of objects in which an object can occur more than once. We denote multisets by lower case bold face letters $\mathbf{x},\mathbf{y},\mathbf{x}_1,\mathbf{x}_2,\ldots$. For an object x and a multiset $\mathbf{x}$, we denote by $Oc(x,\mathbf{x})$ the *number of occurrences* of x in $\mathbf{x}$. For example, if $\mathbf{x}=\{x,x,x,y\}$, then $Oc(x,\mathbf{x})=3$, $Oc(y,\mathbf{x})=1$ and $Oc(z,\mathbf{x})=0$. We define the inclusion as $\mathbf{x}\subseteq\mathbf{y}$ if $Oc(x,\mathbf{x})-Oc(x,\mathbf{y})\geq 0$ for any object x, the *sum* $\mathbf{x}+\mathbf{y}$ as the multiset for which $Oc(x,\mathbf{x}+\mathbf{y})=Oc(x,\mathbf{x})+Oc(x,\mathbf{y})$. The size $|\mathbf{x}|$ of a multiset $\mathbf{x}$ is the number of total occurrences of objects in $\mathbf{x}$. For ordinary sets, relations $\in$, $\subseteq$ and operations $\cup$, $\cap$ are defined by usual manner.

The size $|t|$ of a term t is the total number of occurrences of variables and function symbols in t. For a sequence $\mathbf{t}=t_1,\ldots,t_n$ of terms, $|\mathbf{t}|=|t_1|+\cdots+|t_n|$. We assume a special symbol $\mathbf{1}$ that is not contained in X. Let $\mathbf{X}$ be the class of all the multisets consisting of elements of $X\cup\{\mathbf{1}\}$.

Definition. A multiset $[t]$ of symbols associated with a term t is defined as follows:

(1) if t is a variable x, $[t]=\{x\}$, and

(2) if $t=f(t_1,\ldots,t_n)$ with $f\in\Sigma$, $[t]=\{\mathbf{1}\}+[t_1]+\cdots+[t_n]$.

(3) For a sequence $\mathbf{t}=t_1,\ldots,t_n$ of terms, $[\mathbf{t}]=[t_1]+\cdots+[t_n]$.

Example. For terms $s=f(x,y)$, $t=f(a,b)$ and $u=[a,x]=.(a,.(x,[]))$, the corresponding multisets are $[s]=\{\mathbf{1},x,y\}$, $[t]=\{\mathbf{1},\mathbf{1},\mathbf{1}\}$ and $[u]=\{\mathbf{1},\mathbf{1},\mathbf{1},x,\mathbf{1},\}$, respectively.

For sequences of terms, the preorder induced by the inclusion over $\mathbf{X}$ generalizes one based on size of terms, which is used to characterize linear Prologs in [Shi91].

Lemma 2.2. Let s and t be sequences of terms. Then,

(1) $[|\mathbf{s}|]\subseteq[|\mathbf{t}|]$, iff

(2) $|\mathbf{s}| \leq |\mathbf{t}|$ and $Oc(x,\mathbf{s}) \leq Oc(x,\mathbf{t})$ for every variable x, where $Oc(x,\mathbf{u})$ is the number of occurrences of $\mathbf{x}$ in $\mathbf{u}$.

Finally, we formalize learning of Prolog programs as identification of formal languages from positive data in the limit. We give a brief review of basic definitions and results on inductive inference according to [Ang79, Shi91].

Let U and E be sets, whose elements are called *objects* and *expressions*, respectively. A *concept* is a subset $R \subseteq U$. A *formal system* is a finite subset $\Gamma \subseteq E$. A *semantic mapping* is a mapping M from formal systems to concepts. When $M(\mathrm{G}) = R$, we say a formal system G *defines* a concept R.

Definition. A *concept defining framework* is a triple (U, E, F) of a universe U of objects, a universe E of expressions, and a semantic mapping F.

In the case of inference of Prolog programs, for example, U is the Herbrand base $B(L)$ of a first order language L, E is a class of definite clauses and $\mathrm{F}(\mathrm{G})$ is the least Herbrand model $M(\mathrm{G})$ of a program $\Gamma \subseteq E$. From now on, we fix a concept defining framework (U, E, F).

Definition. Let $\Gamma_1, \Gamma_2, \ldots$ be a recursive enumeration of all the formal systems. Then, a class $C = \{\Phi(\Gamma_1), \Phi(\Gamma_2), \ldots,\}$ of concepts is said to be an *indexed family of recursive concepts* if there is an algorithm that, given an object $w \in U$ and a formal system $\Gamma \subseteq E$, decides whether $w \in \Phi(\Gamma)$.

Definition. A positive presentation s of a concept R is an infinite sequence $w_1, w_2, \ldots$ of objects such that $\{w_n ; n \geq 1\} = R$.

An *inference machine* is an effective procedure M that requests a string and produces a conjecture at a time. Given a positive presentation $\sigma = w_1, w_2, \ldots$, M generates an infinite sequence $g_1, g_2, \ldots$ of conjectures as follows: it starts with the empty sample $S_0 = \varnothing$. When M makes the n-th request $(n > 0)$, an object w_n is added to the sample. Then, M reads the current sample $S_n = \{w_1, \ldots, w_n\}$ and adds a conjecture g_n to the end of the sequence of conjectures; any conjecture g_n must be a formal system.

Definition. A class of concepts C is said to be *identifiable in the limit from positive data* if there is an inference machine M such that for any $R \in C$ and any positive presentation s of R, there is some g such that for all sufficiently large n, the n-th conjecture g_n is identical to g and $\Phi(g) = R$.

Hereafter, we simply say *inferable from positive data* instead of identifiable in the limit from positive data. The first result on inductive inference from positive data, which was obtained by Gold [Gol67] in 1967, is negative one. The following is a modified version of the result.

Theorem 2.3. ([Gol67]) If a class of concepts C is *superfinite*, that is, C contains infinitely many finite concepts

$$R_1 \subseteq R_2 \subseteq \ldots \subseteq R_n \subseteq \ldots \quad (n > 0)$$

and at least one infinite concept R_∞ that contains all of them, then C is not identifiable from positive data.

From Gold's theorem, we can see that even the class of regular languages is not inferable from positive data. The result made most of researchers in the area disappointed. The first result that indicated the possibility of inductive inference form positive data was given by Angluin [Ang79]. Angluin characterized a class that is inferable from positive data and presented a number of classes inferable from positive data.

Shinohara [Shi91] showed the following theorem by extending a characterization shown by Angluin. A semantic mapping Φ is *monotonic* if $\Gamma \subseteq \Gamma'$ implies $\Phi(\Gamma) \subseteq \Phi(\Gamma')$. A formal system G is *reduced with respect to* $S \subseteq U$ if $S \subseteq \Phi(\Gamma)$ but $S \not\subseteq \Phi(\Gamma')$ for any proper subset G' of G.

Definition. A concept defining framework (U, E, F) has *bounded finite thickness* if it is monotonic and the set

$$\{\, M(\mathrm{G}) \mid \#\Gamma \le m \text{ and } \mathrm{G} \text{ is reduced wit respect to } S \,\}$$

consists of finitely many concepts for any finite set $S \subseteq U$ and any $m > 0$.

Theorem 2.4. ([Shi91]) Let a concept defining framework (U, E, F) has bounded finite thickness. Then, the class $C^m = \{\, M(\mathrm{G}) \mid \Gamma \subseteq E \text{ and } \#\Gamma \le m \,\}$ is inferable from positive data for every $m > 0$.

3. Linearly Covering Programs

In this section, we introduce a class of logic programs, called linearly covering programs as a target class of inference. An *argument goal* is a triple $B=(G,\mathbf{x},\mathbf{y})$, where G is a goal and $\mathbf{x},\mathbf{y} \in \mathbf{X}$.

Definition. A *block* is an augmented goal $B=(G,\mathbf{x},\mathbf{y})$ defined recursively as follows.

(3.1) For the empty goal $\varnothing$ and $\mathbf{x} \supseteq \mathbf{y}$, $B=(\varnothing,\mathbf{x},\mathbf{y})$ is a block called an *empty block*.

(3.2) For an atom $p(\mathbf{s};\mathbf{t})$, $B=(\{p(\mathbf{s};\mathbf{t})\},[\mathbf{s}],[\mathbf{t}])$ is a block called an *atomic block*.

(3.3) If $(G_1,\mathbf{x}_1,\mathbf{y}_1)$ and $(G_2,\mathbf{x}_2,\mathbf{y}_2)$ are blocks, then $B=(G_1+G_2,\mathbf{x}_1+\mathbf{x}_2,\mathbf{y}_1+\mathbf{y}_2)$ is a block called a *parallel block*.

(3.4) If $(G_1,\mathbf{x},\mathbf{y})$ and $(G_2,\mathbf{y},\mathbf{z})$ are blocks, then $B = (G_1+G_2,\ \mathbf{x},\ \mathbf{z})$ is a block called a *series block*.

Intuitively, we can think of a block $B=(G,\mathbf{x},\mathbf{y})$ as a network of pipes along which fluid can flow, $\mathbf{x}$ as the source and $\mathbf{y}$ as the sink of the network. We can construct a larger network by combining two network modules in parallel or in series.

Definition. A definite clause $C = p(\mathbf{s};\mathbf{t}) \leftarrow G$ is *linearly covering* if $(G,[\mathbf{s}],[\mathbf{t}])$ is a block, and a program is *linearly covering* if its definite clauses are all *linearly covering*.

Example 3.1. Clauses defined as

```
reverse([A | B]; D)←
        reverse(A; C),
        add_last(C, A; D).

merge_sort(A; F)←
        split(A; B, C),
        merge_sort(B; D),
        merge_sort(C; E),
        merge(D, E; F).
```

are linearly covering. In the clause that defines reverse, the variable C occurs only in bodies of the clause.

We can efficiently decide from syntax whether a program is linearly covering.

Theorem 3.1. ([Ari93]) There is a polynomial time algorithm that, given a program P, decides whether P is linearly covering.

Lemma 3.2. Let $B=(G,\mathbf{x},\mathbf{y})$ be a block and G' be a resolvent of G and a linearly covering clause C by the mgu θ. Then, $B'=(G',\mathbf{x}\theta,\mathbf{y}\theta)$ is a block.

Proof. The result is proved by induction on constructions of blocks. n

Lemma 3.3. Let P be a linearly covering program and $B=(G,\mathbf{x},\mathbf{y})$ be a block. If there is a refutation from G with the answer θ, then $\mathbf{x}\theta \supseteq \mathbf{y}\theta$ and $\mathbf{x}\theta \supseteq [\mathbf{s}]\theta \supseteq [\mathbf{t}]\theta$ for each atom $p(\mathbf{s}\ ;\ \mathbf{t}) \in G$

Proof. Assume that there is a refutation from G with the answer θ. Then, repeated applications of Lemma 3.2 immediately show that $\mathbf{x}\theta \supseteq \mathbf{y}\theta$. Let $p(\mathbf{s}\ ;\ \mathbf{t})$ be an atom in G. Then, we can construct a refutation with the answer θ' for G' from the refutation for

G, where G' is a subgoal of G such that (1) $p(\mathbf{s}\,;\mathbf{t}) \in G'$, and (2) $(G', \mathbf{x}, [\mathbf{s}])$ and $(G', \mathbf{x}, [\mathbf{t}])$ are blocks. Thus, the result is proved by using the preceding result . n

Definition. A program P has *ground input and output* (for short, of ground I/O) property if for any goal $G = \leftarrow p(\mathbf{s}\,;\mathbf{t})$ such that the input arguments $\mathbf{s}$ is ground, if there is a SLD-refutation from G, then the answer substitution θ makes the output arguments $\mathbf{t}\theta$ ground.

Theorem 3.4. A linearly covering program has ground I/O property.
Proof. Let $G = \leftarrow p(\mathbf{s}\,;\mathbf{t})$ be a goal, where $\mathbf{s}$ is ground. Thus, $[\mathbf{s}]\theta$ is of the form $\{\mathbf{1},\ldots,\mathbf{1}\}$. If there is a SLD-refutation from G with the answer θ, then $[\mathbf{s}]\theta \supseteq [\mathbf{t}]\theta$ by Lemma 3.3. Therefore, $[\mathbf{t}]\theta$ is also of the form $\{\mathbf{1},\ldots,\mathbf{1}\}$ and is ground. n

4. Main result

In this section, we show the inferability of the class of linearly covering programs containing at most m definite clauses of body-length at most k for every $k,m > 0$. Furthermore, we show that the restriction on the length of bodies is necessary for the inferability.

Theorem 4.1. ([Ari93]) For the class of linearly covering programs, there is a recursive function f from ground goals and programs to positive integers for which for any ground atom $a \in B(P)$ and any program P in the class,

(1) $a \in M(P)$, iff
(2) there is a SLD-refutation from $\leftarrow a$ of length at most $f(a,P)$.

Lemma 4.2. Let $\Gamma_1, \Gamma_2, \ldots$ be any recursive enumeration of linearly covering programs with at most k bodies. Then, the class $C = \{M(\Gamma_1), M(\Gamma_2), \ldots\}$ of concepts is an indexed family of recursive concepts.
Proof. By Theorem 4.1, the least Herbrand model $M(P)$ is recursive for any linearly covering programs P. Thus, from Theorem 3.1, a recursive function can enumerate all the linearly covering programs on a given first order language. n

Definition. Let P be a program. A *supporting clause* for P is a ground instance $C = a \leftarrow a_1,\ldots,a_m$ $(m \geq 0)$ of a clause in P for which $\{a_1,\ldots,a_m\} \subseteq M(P)$.

Lemma 4.3. Let P be a linearly covering program. Then, for any supporting clause $a \leftarrow b_1,\ldots,b_m$ for P, the following (1) and (2) hold.

(1) $|a / I| \geq |b_i / I|$ for any $1 \leq i \leq m$.
(2) $|a / I| \geq |a / O|$ and $|b_i / I| \geq |b_i / O|$ for any $1 \leq i \leq m$.

Proof. By Lemma 3.2

Now, we consider a concept defining framework (U,E,Φ) defined as $U = B_L$, E is the set of all linearly covering clauses of body-length at most k, and $\Phi(\Gamma) = M(\Gamma)$ for a fixed $k > 0$.

Lemma 4.4. Let $S \subseteq B(L)$ be finite and P be a linearly covering program that is reduced with respect to S. Then, for any $A_0 \leftarrow A_1, \ldots, A_n$ $(n \geq 0)$ in P and any $0 \leq i \leq n$,

$$|A_i| \leq 2 \cdot \max\{|w| ; w \in S\}.$$

Proof. For every clause $C = A_0 \leftarrow A_1, \ldots, A_n$ in P, if P is reduced with respect to S, then there is a supporting clause $c = a_0 \leftarrow a_1, \ldots, a_n$ for P such that c is a ground instance of C and $a_0 \in S$. If A is an instance of an atom B, then $|A| \geq |B|$. Thus, the result follows from Lemma 4.3. n

Since there are finitely many atoms smaller than a fixed size except renaming of variables, we have the following lemma from Lemma 4.4.

Lemma 4.5. For any finite set $S \subseteq B(L)$ and any $m > 0$, the set

$$\left\{ M(\Gamma) \;\middle|\; \begin{array}{c} \Gamma \text{ is a set of linearly covering clauses of body-length at most } k, \\ \#\Gamma \leq m, \text{ and } \Gamma \text{ is reduced with respect to } S \end{array} \right\}$$

consists of finitely many Herbrand models.

Lemma 4.6. Let $k > 0$, $U = B_L$, E be the set of all linearly covering clauses of body-length at most k, and $\Phi(\Gamma) = M(\Gamma)$. Then, (U,E,Φ) is a concept defining framework that has bounded finite thickness.

From the lemma and Theorem 2.4, we have the main result of the paper.

Theorem 4.7. For any $k,m > 0$, the class of least Herbrand models defined by linearly covering programs consisting of at most m clauses of body-length at most k is inferable from positive data.

Next, we consider the problem of identifying a Prolog program P from positive examples of a target predicate p without oracles for other auxiliary predicates. The problem is called *theoretical term generation problem* [Ban88, Ish90, Mug90]. We deal with the problem by modifying a concept defining framework (U,E,Φ) as U is the set of all the ground instances of a target predicate p, E is the same as in the original, and $\Phi(\Gamma) = M(p, P)$. For the modification, Lemma 4.4 is again true. Thus, we can prove Lemma 4.6 in a similar manner. Hence, we have the following.

Theorem 4.8. Let p be a fixed predicate symbol. Then, for any $k,m>0$, the class of meanings of p defined by linearly covering programs consisting of at most m clauses of body-length at most k is inferable from positive examples of p, without providing positive examples of any predicates other than p.

Finally, we show that the restriction on the body-length of clauses for the target class in Theorem 4.7 is necessary even for inference of least Herbrand models.

Theorem 4.9. Let L be a first order language such that $B(L)$ is infinite, P contains at least two nonzero arity predicates and for one of them, its arity is more than or equal to two. Then, for any $m \geq 4$, the class of least Herbrand models defined by linearly covering programs consisting of at most m clauses can not be inferable from positive data.

Proof. Since $B(L)$ is infinite, we can assume without loss of generality that P = { p(_), q(_,_) } and S = {a, f(_,_)}. Let Q = {q(x,x),q(f(a,x),x)}. Then, we define an infinite sequence $P_0,\ldots,P_n,\ldots(n \geq 0)$ and the limit P_∞ as

$$P_n = \{ \mathtt{p}(\mathtt{x}_n) \leftarrow \mathtt{q}(\mathtt{x}_n,\mathtt{x}_{n-1}),\ldots,\mathtt{q}(\mathtt{x}_1,\mathtt{a}) \} \cup Q,$$

$$P_\infty = \left\{ \begin{array}{l} \mathtt{p}(\mathtt{a}) \leftarrow \\ \mathtt{p}(\mathtt{x}) \leftarrow \mathtt{q}(\mathtt{x},\mathtt{y}),\mathtt{p}(\mathtt{y}) \end{array} \right\} \cup Q.$$

We write a term $\overbrace{\mathtt{f}(\mathtt{a},\mathtt{f}(\mathtt{a},\ldots,\mathtt{f}(\mathtt{a},\mathtt{a})\ldots))}^{n}$ as a^{n+1}. Then, these programs P_n define an infinite increasing sequence $M(P_n) = \{ a^{i+1} ; 0 \leq i \leq n \}$ $(n \geq 0)$ of the finite models. On the other hand, P_∞ defines the limit $M(P_\infty) = \bigcup_{n \geq 0} M(P_n)$ of the sequence. Thus, any concept class that contains these models is superfinite. Hence, the result immediately follows from Theorem 2.3. n

5. Conclusion

In this paper, we proposed a subclass of Prolog programs, called *linearly covering programs* and showed that if the maximum length of subgoals in a clause is bounded by a fixed constant $k > 0$, the class is inferable from positive examples only of a target predicate. Further, we showed that the restriction on the length of bodies is necessary.

In recent years, a number of inference systems are proposed in the area of *inductive logic programming* concerning to theoretical term generation problem [Ban88, Mug90]. From our main result, we know the existence of an effective inference algorithm that infers a linearly covering program defining an unknown target relation by receiving positive examples of the target relation. Such a inference algorithm is an ideal for concept learning and program synthesis in inductive logic programming. Therefore, studies on realization of inference algorithm based on this work may be interesting.

For practical application, efficiency is important. In [AIS92], the authors studied an efficient inference algorithm that infers a small subclass of linear Prologs from positive data. The study on efficient algorithms for subclasses of linearly covering programs is a future problem.

Plümer [Plu90] studies the procedural semantics of a slightly wider class of Prolog programs than linearly covering programs based on linear predicate inequalities.

Acknowledgements

The first author is grateful to Prof. Setsuo Arikawa for his constant encouragement. He also would like to thank to Prof. Setsuko Otsuki and Prof. Akira Takeuchi for their constant support and to Hiroki Ishizaka for fruitful discussion on inductive inference of Prolog programs.

References

[Ang79] Angluin, D. : Finding pattern common to a set of strings, In Proc. 11th STOC, (1979), 130--141.

[AIS92] Arimura, H., Ishizaka, H. and Shinohara, T. : Polynomial time inference of a subclass of context-free transformations. In Proc. 5th Annual ACM Workshop on Computational Learning Theory, (1992), 136--143.

[Ari93] Arimura, H. : Depth-bounded inference for Nonterminating Prologs. To appear in Bulletin of Informatics and Cybernetics, (1993).

[Ban88] Banerji, R. B. : Learning Theories in a Subset of a Polyadic Logic. In Proc. first Annual ACM Workshop on Computational Learning Theory, (1988), 281--295.

[Gol67] Gold, E. M. : Languages Identification in the Limit, Information and control, 10, (1967), 447--474.

[Ish90] Ishizaka, H. : Polynomial Time Learnability of Simple Deterministic Languages, Machine Learning, 5 (2), (1990), 151--164.

[Llo87] Lloyd J. W. : Foundations of Logic Programming , Springer Verlag, second edition, (1987).

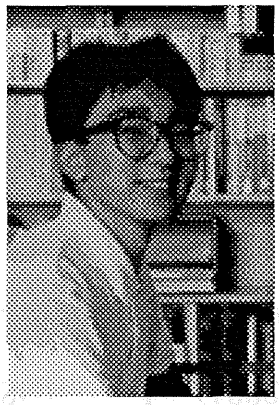
[Mug90] Muggleton, S. : Inductive Logic Programming, In Arikawa, S. et al, editor, Proc. the First International Workshop on Algorithmic Learning Theory, (1990), 42--62.

[Plu90] Plümer, L. : Termination Proofs for Logic Programs based on Predicate Inequalities, In Warren, D. H. and Szeredi, H., editors, Proc. of the seventh International Conference on Logic Programming, The MITpress, (1990), 634--648.

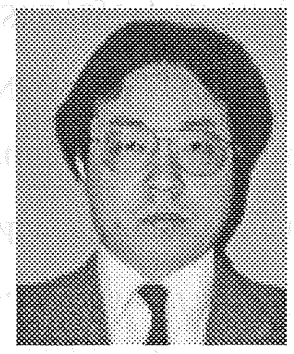
[Sha82] Shapiro, E. Y. : Algorithmic Program Debugging, MIT Press, (1982).

[Shi91] Shinohara, T. : Inductive Inference of Monotonic Formal Systems From Positive Data, New Generation Computing, 8, OHMSHA, LTD. and Springer-Verlag, (1991), 371--384.

About the Author



Hiroki Arimura (有村博紀) was born in Fukuoka on June 7, 1965. He received the B.S. degree in 1988 in Phisics, and the M.S. degree in 1990 in Information Systems from Kyusyu University. Presently, he is an Assistant of Kyushu Institute of Technology. His research interests are in logic programming and computational learning theory.



Takeshi Shinohara (篠原　武) was born in Fukuoka on January 23, 1955. He received the B.S. in 1980 from Kyoto University, and the M.S. degree and the Dr. Sci. from Kyushu University in 1982, 1986, respectively. Currently, he is an Associate Professor of Department of Artificial Intelligence, Kyushu Institute of Technology. His present interests include information retrieval, string pattern matching algorithms and computational learning theory.

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