ICOT Technical Report: TR-260

TR-260

Inductive Inference of Logic Programs based on Algebraic Semantics

by
Y. Sakakibara (Fujitsu)

May, 1987

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Inductive Inference of Logic Programs based on Algebraic Semantics

by

Yasubumi SAKAKIBARA*

* International Institute for Advanced Study of Social Information Science, FUJITSU LIMITED, 140 Miyamoto, Numazu, Shizuoka 410-03, JAPAN

Abstract In this paper we will present a new inductive inference algorithm for a class of logic programs, called linear monadic logic programs, in the sense that it is different from the Shapiro's Model Inference System. It is known that a set of trees (or terms) is rational if and only if it can be computed by a linear monadic logic program, and that the rational set of trees is recognized by tree automata. On the other hand, several efficient inference algorithms for finite automata are developed. Then we will extend them to an inference algorithm for tree automata and use it to get an efficient inductive inference algorithm for linear monadic logic programs. The correctness, time complexity and several comparisons of the algorithm with the Model Inference System will be shown.

1. Introduction

The study of inductive inference of logic programs was initially and mostly done by E.Shapiro and his work is known by the *Model Inference System* [8,9]. He devises a program that infers first order sentences (Horn clauses) from examples of their logical consequences. The target of the inference is an Herbrand model. Thus Shapiro's algorithm (especially the diagnosis algorithm) deeply depends on the

theory of predicate logic and logic programming. In the theory of logic programming, the least model $\cap M(LP)$ of a logic program LP is taken as the mathematical semantics, called model-theoretic semantics, for it. This semantics provides the denotation of a predicate symbol P in a logic program LP :

$D(P) = \{(t_1,...,t_n) : P(t_1,...,t_n) \in \cap M(LP)\}$.

D(P) is the denotation of P as determined by model-theoretic semantics. Thus model-theoretic semantics gives a nice characterization of the set of terms computed by a logic program.

On the other hand, algebraic semantics which connects between the theory of tree languages and the semantics of programming languages is now well known and recently introduced to logic programming in [7]. It studies the use of tree languages in the semantics of logic programming. Intuitively, a term is a tree, whose nodes are labeled by symbols in such a way that the arity of the label of each node is equal to the degree of that node. Then the set of terms computed by a logic program LP can be viewed as a tree language. That is to say, the denotation of P, $D(P)=\{t : P(t) \in \cap M(LP)\}$, is a tree language. From the result in [7], a set of trees is rational iff it can be computed by a linear monadic logic program, where a *rational* set of trees is a set of trees which can be recognized by some tree automaton T_A and a *linear monadic logic program* is a class of logic programs defined by syntactic restrictions such that predicate symbols are monadic, the height of terms involved is less than or equal to 1 and the variables in a term must be distinct. Therefore, the denotation of P can be written as $D(P)=\{t : t$ is accepted by a tree automaton T_A about P in LP$\}$. Based on such an algebraic semantics, we can establish a new inductive inference schema of logic programs so that the problem of inductive inference of logic programs is reduced to the problem of inductive inference of tree automata which accept tree languages computed by the logic programs. Then we can get an efficient inductive inference method of logic programs which is extended from the one of automata. In this setting, the problem will be set so that the inductive inference algorithm can

identify in the limit a class of logic programs, linear monadic logic programs, such that the denotation of P computed by it is equal to the one in the unknown model.

2. Basic definitions of tree

Definition Let N be the set of positive integers. Dom is a *tree domain* iff it satisfies

a) $Dom \subseteq N^*$ and Dom is finite,

b) Dom is prefix-closed, i.e. if $a, b \in N^*$ and $ab \in Dom$ then $a \in Dom$,

c) $am \in Dom$ implies $an \in Dom$ for $1 \leqq n \leqq m$, $n \in N$.

A *direct successor* (*direct predecessor*) of a node x is a node y, where $y = xa$ ($ya = x$) for $a \in N$. The *frontier* of Dom is the set of all nodes in Dom which have no direct successors.

Definition The *depth* of $a \in Dom$ is defined recursively as

$depth(a) = 0$ if $a = \varepsilon$

$depth(ax) = depth(a) + 1$ for $x \in N$.

If t is a tree domain, then $depth(t) = \max\{depth(i) : i \in t\}$.

Definition ([2]) A *ranked alphabet* is a pair (Γ, ρ) consisting of a finite set Γ and a mapping $\rho : \Gamma \to N$ which defines the *rank* of any symbol f in Γ. For such a set Γ, we denote by Γ_n the set $\{f \in \Gamma : \rho(f) = n\}$ for $n \geqq 0$. In many cases the symbols in Γ_n will be considered as *function symbols*. The rank of a function symbol is called its *arity* and a symbol of arity 0 is called a *constant symbol*.

Definition ([2]) A *tree* over a finite ranked alphabet Γ is a mapping $t : Dom \to \Gamma$, which labels the nodes of the tree domain Dom. Furthermore we require the following condition which concerns the rank function : if $t(a) = f$ of arity $k \geqq 0$, then for $i \in N$, $ai \in Dom(t)$ iff $1 \leqq i \leqq k$. Let Γ^T be the set of all trees over Γ.

Definition If a, b, $b' \in U$ and $b = ab'$, then $b/a = b'$. If $t \in \Gamma^T$, then the *subtree* of t at a, where a is in the domain of t ($a \in Dom(t)$), is defined as $t/a = \{(x, b) : (ax, b) \in t\}$. The

replacement of the subtree at a with a tree u is defined as $t(a \leftarrow u) = \{(b, x) : t(b) = x \text{ and } a \not< b\} \cup \{(ay, x) : u(y) = x \text{ and } y \in Dom(u)\}$. The *replacement (substitution) of terminal nodes* labeled $c \in \Gamma$ with a tree u is defined as $t(c \leftarrow u) = \{(b, x) : t(b) = x \text{ and } x \neq c\} \cup \{(dy, x) : t(d) = c, u(y) = x \text{ and } y \in Dom(u)\}$.

Definition Let \$ be a new symbol of arity 0 that we add to Γ. $(\Gamma \cup \{\$\})^T$ denotes the set of all trees over $\Gamma \cup \{\$\}$. Especially we are interested in the subset Sub of $(\Gamma \cup \{\$\})^T$ which is the set of all trees $t \in (\Gamma \cup \{\$\})^T$ such that t exactly contains one \$-symbol. We use the notation $\Gamma_\$^T$ for the Sub. For trees $t \in \Gamma^T$ and $s \in \Gamma_\$^T$, we define an operation $\cdot$ to attach s to the node labeled \$ of t by $s \cdot t = s(\$ \leftarrow t)$ (like concatenation of strings).

3. Tree automaton and linear monadic logic program

Definition ([10]) A *deterministic (frontier to root) tree automaton* over Γ is a 4-tuple $T_A = (Q, \Gamma, \delta, F)$, where

a) Q is a nonempty finite set of states,

b) Γ is a nonempty finite tree labels,

c) $\delta = (\delta_0, \delta_1, ..., \delta_m)$ is a state transition function such that

$$\delta_k : \Gamma_k \times Q^k \rightarrow Q \quad (k = 0, 1, ..., m),$$

d) $F \subseteq Q$ is the set of final states.

If δ is a state transition function from $\Gamma_k \times Q^k$ to 2^Q, then T_A is *nondeterministic*. δ can be extended to Γ^T by letting :

$$\delta(f(t_1, ..., t_k)) = \delta_k(f, \delta(t_1), ..., \delta(t_k)) \quad \text{for } k > 0 \text{ and } f \in \Gamma_k,$$
$$= \delta_0(f) \quad \text{for } k = 0 \text{ and } f \in \Gamma_0.$$

The tree t is *accepted* by T_A iff $\delta(t) \in F$. The set of trees accepted by T_A is the subset $L(T_A)$ of Γ^T defined as : $L(T_A) = \{t : \delta(t) \in F\}$. A subset L of Γ^T is called *rational* iff there exists some automaton T_A such that $L = L(T_A)$.

Proposition 3.1 ([10]) Nondeterministic frontier to root tree automata are no more powerful than deterministic frontier to root tree automata.

Given a rational set L_R, by the above proposition, there always exists the minimum state deterministic tree automaton which accepts L_R.

For the definitions of logic program and the semantics of it, we use the notations of [3]. Especially, we reserve the predicate symbol P for the inferring predicate. We state some useful results about semantics of logic programs.

Proposition 3.2 Let LP be a logic program. The least Herbrand model for LP which is the intersection of all Herbrand models for LP exists.

We denote it by $\cap M(LP)$.

Proposition 3.3 Let LP be a logic program. If $A_0 \leftarrow A_1,...,A_n$ $(n \geqq 0)$ is a ground instance of a clause in LP and $A_1,...,A_n \in \cap M(LP)$, then $A_0 \in \cap M(LP)$. Conversely, if $A_0 \in \cap M(LP)$, then there is a ground instance $A_0 \leftarrow A_1,...,A_n$ $(n \geqq 0)$ of a clause in LP such that $A_1,...,A_n \in \cap M(LP)$.

Definition ([7]) A *linear monadic logic program* is a logic program in which all predicate symbols are monadic and all the terms occurring in atomic formulas belong to one of the following two forms :

a) x_i $(i \in N)$

b) $f(x_{i_1},...,x_{i_m})$ with $f \in \Gamma_m$, $\{i_1,...,i_m\} \subseteq N$ the i_k being pairwise distinct.

Now we state very important theorem from [7] which connects a linear monadic logic program with a tree automaton.

Proposition 3.4 ([7]) A set of trees is rational iff it can be computed by a linear monadic logic program.

By the results of logic programs in [3], we can restate the above theorem as follows.

Corollary 3.5 If LMLP is a linear monadic logic program and P is a predicate symbol in LMLP, then the set of trees $\{t : P(t) \in \cap M(LMLP)\}$ is rational. Conversely, if a set of trees T is rational, then there is a linear monadic logic program LMLP such that $T = \{t : P(t) \in \cap M(LMLP)\}$.

Definition (A) Let $T_A = (Q, \Gamma, \delta, F)$ be a tree automaton. We define a set of predicate symbols $R = \{R_q : q \in Q\}$ in one-to-one correspondence with the set of states of the T_A. To code the computation of T_A, we need a clause for each transition. So, for each $f \in \Gamma n$ and each n-tuple of states $(q_1,...,q_n)$, we define the clause $C_{f,q_1,...,q_n}$ as :

$$C_{f,q_1,...,q_n} = R_{\delta(f,q_1,...,q_n)}(f(x_1,...,x_n)) \leftarrow R_{q_1}(x_1),...,R_{q_n}(x_n).$$

Another set of clauses is necessary to take care of the set of final states. So, for each $q \in F$, we define the clause C'_q as :

$$C'_q = P(x) \leftarrow R_q(x).$$

Proposition 3.6 Let $T_A = (Q, \Gamma, \delta, F)$ be a tree automaton and LMLP be a corresponding linear monadic logic program in the above sense. Then $R_q(t) \in \cap M(LMLP)$ iff $\delta(t) = q$. Furthermore, $P(t) \in \cap M(LMLP)$ iff $\delta(t)$ is in F.

(Proof) We prove it by induction on the depth of t. Suppose first that the depth of t is 0, i.e. $t = a \in \Gamma_0$. By the definition of $C_{f,q_1,...,q_n}$, there is a clause $R_{\delta(a)}(a) \leftarrow$ in LMLP. Then clearly $R_{\delta(a)}(a) \in \cap M(LMLP)$. If $\delta(a) = q$, then $R_q(a) = R_{\delta(a)}(a) \in \cap M(LMLP)$. Conversely if $R_q(a) \in \cap M(LMLP)$, since T_A is deterministic (so δ is deterministic), $\delta(a) = q$.

Next suppose that the result holds for all trees with depth at most h. Let t be a tree of depth $h+1$, so that $t = f(u_1,...,u_n)$ for some trees $u_1,...,u_n$ with depth at most h and some $f \in \Gamma_n$. For the if part, assume that $\delta(t) = q$. By the definition of δ, $\delta(t) = \delta(f(u_1,...,u_n)) = \delta(f,\delta(u_1),...,\delta(u_n)) = q$. By the definition of the clause $C_{f,q_1,...,q_n}$, there is a clause $R_{\delta(f,\delta(u_1),...,\delta(u_n))}(f(x_1,...,x_n)) \leftarrow R_{\delta(u_1)}(x_1),...,R_{\delta(u_n)}(x_n)$ in LMLP. For $1 \leq i \leq n$, by the induction hypothesis, $R_{\delta(u_i)}(u_i) \in \cap M(LMLP)$ iff $\delta(u_i) = \delta(u_i)$. The right-hand side of this statement is obviously true. Thus $R_{\delta(u_i)}(u_i) \in \cap M(LMLP)$, and so $R_{\delta(f,\delta(u_1),...,\delta(u_n))}(f(u_1,...,u_n)) \in \cap M(LMLP)$. Then

$R_q(t) = R_{\delta(t)}(t)$, by the assumption,

$= R_{\delta(f(u_1,...,u_n))}(f(u_1,...,u_n))$

$= R_{\delta(f,\delta(u_1),...,\delta(u_n))}(f(u_1,...,u_n))$, by the definition of δ.

Hence $R_q(t) \in \cap M(LMLP)$.

For the only-if part, assume that $R_q(t) \in \cap M(LMLP)$. Then $R_q(f(u_1,...,u_n)) \in \cap M(LMLP)$. For $R_q(f(u_1,...,u_n))$, there is a ground instance $R_{\delta(f,q_1,...,q_n)}(f(u_1,...,u_n)) \leftarrow R_{q_1}(u_1),...,R_{q_n}(u_n)$ of a clause in LMLP such that $\delta(f,q_1,...,q_n) = q$ and $R_{q_1}(u_1),...,R_{q_n}(u_n) \in \cap M(LMLP)$. By the induction hypothesis, $\delta(u_i) = q_i$ $(1 \leqq i \leqq n)$. Then

$\delta(t) = \delta(f(u_1,...,u_n))$

$= \delta(f,\delta(u_1),...,\delta(u_n))$, by the definition of δ,

$= \delta(f,q_1,...,q_n)$

$= q$.

This completes the induction.

Furthermore, if $\delta(t)$ is in F, there is a final state q_f in F such that $\delta(t) = q_f$. Then by the above result, $R_{q_f}(t) \in \cap M(LMLP)$, and by the definition of C'_q, $P(t) \in \cap M(LMLP)$. Conversely if $P(t) \in \cap M(LMLP)$, there is a ground instance $P(t) \leftarrow R_q(t)$ of a clause in LMLP such that $R_q(t) \in \cap M(LMLP)$ and q is a final state. By the above result, $\delta(t) = q$, and hence $\delta(t)$ is in F.

□

By the above result, in the inductive inference schema of linear monadic logic program, we have only to consider inferring a linear monadic logic program of the form in definition (A).

4. Predicate characterization matrix

Definition A *set of test predicates* S is a finite set of trees of Γ^T. The *set of test clauses* is defined to be $X(S) = \Gamma(S^*) - S = \{f(\hat{u}) : f \in \Gamma_i, \hat{u} \in S^i$, and $f(\hat{u}) \notin S$ for $i \geqq 0\}$. A *set of experiments* E is a finite set of trees of $\Gamma_\$^T$. S is called *subtree-closed* if $s \in S$ implies all

subtrees of s are elements of S. E is called *$-prefix-closed with respect to* S if $e \in E$ except \$ implies there exists an e' in E such that $e = e' \cdot f(s_1,...,s_{i-1},\$,s_i,...,s_{n-1})$ for some $f \in \Gamma_n$, $s_1,...,s_{n-1} \in S$ and i $(1 \leqq i \leqq n)$.

Definition A predicate characterization matrix is a triple (S, E, M) where M is a matrix with labeled rows and columns such that

1) The rows are labeled with the elements of $S \cup X(S)$.

2) The columns are labeled with the elements of E.

3) Each entry of M is either 0 or 1.

4) If $s_i, s_j \in S \cup X(S)$ and $e_i, e_j \in E$ and $e_i \cdot s_i = e_j \cdot s_j$, then the (s_i, e_i) and (s_j, e_j) positions in M must have the same entry.

The *data contained in* M is $D(M) = \{(e \cdot s, y) : s \in S \cup X(S), e \in E$, and the entry of M is $y \in \{0, 1\}\}$. Thus we can regard D(M) as a finite function mapping $E \cdot (S \cup X(S))$ to $\{0, 1\}$. If s is an element of $(S \cup X(S))$, then *row(s)* denotes the finite function f from E to $\{0, 1\}$ defined by $f(e) = D(M)(e \cdot s)$.

Definition A predicate characterization matrix is called *closed* if every row(x) of test clause $x \in X(S)$ is identical to some row(s) of test predicate $s \in S$. A predicate characterization matrix is called *consistent* if whenever s_1 and s_2 are test predicates of S such that $row(s_1)$ is equal to $row(s_2)$, for all $f \in \Gamma_n$ and $u_1,...,u_{n-1} \in S$, $row(f(u_1,...,u_{i-1},s_1,u_i,...,u_{n-1}))$ is equal to $row(f(u_1,...,u_{i-1},s_2,u_i,...,u_{n-1}))$ for $0 \leqq i \leqq n$ $(n \geqq 0)$.

The ideas of the closed, consistent predicate characterization matrix and the algorithm using this are essentially the extensions of Angluin's ones [1] (the extension from string automata to tree automata and so to linear monadic logic programs). A sequence of following lemmas and theorems are guided by those Angluin's results. The idea of the characterization matrix is also related to the work by Gold [4].

Definition Let (S, E, M) be a closed, consistent predicate characterization matrix such that E contains \$. The *constructed linear monadic logic program* $LMLP_M$ over Γ from (S, E, M) is defined with predicate set Pre, calling predicate P, and the set of clauses $LMLP_M$ as follows.

$Pre = \{R_{row(s)}(x) : s \in S\}$,

$LMLP_M = \{P(x) \leftarrow R_{row(s)}(x) : s \in S \text{ and } D(M)(s) = 1\}$

$\cup \{R_{row(f(s_1,...,s_n))}(f(x_1,...,x_n)) \leftarrow R_{row(s_1)}(x_1),...,R_{row(s_n)}(x_n) : f \in \Gamma_n, n > 0\}$

$\cup \{R_{row(a)}(a) \leftarrow : a \in \Gamma_0\}$.

Lemma 4.1 Suppose that (S, E, M) is a closed, consistent predicate characterization matrix such that S is subtree-closed and E is \$-prefix-closed with respect to S. For the constructed linear monadic logic program $LMLP_M$ and for every s in $(S \cup X(S))$, $R_{row(s)}(s) \in \cap M(LMLP_M)$.

(Proof) We prove it by induction on the depth of s. Suppose first that the depth of s is 0, i.e., $s \in \Gamma_0$. Since $R_{row(s)}(s) \leftarrow$ by the definition of $LMLP_M$, the result is clearly true. Next suppose that the result holds for all trees in $(S \cup X(S))$ with depth at most h. Let t in $(S \cup X(S))$ have depth h + 1, so that $t = f(s_1,...,s_n)$ for some $s_1,...,s_n$ in $(S \cup X(S))$ with depth at most h and some f in Γ_n. Since S is subtree-closed, $s_1,...,s_n$ must be in S. Then

$R_{row(t)}(t) \in \cap M(LMLP_M)$

iff $R_{row(f(s_1,...,s_n))}(f(s_1,...,s_n)) \in \cap M(LMLP_M)$

iff $R_{row(s_1)}(s_1),...,R_{row(s_n)}(s_n) \in \cap M(LMLP_M)$,

by the definition of $LMLP_M$ and proposition 3.3.

By the induction hypothesis, $R_{row(s_1)}(s_1),...,R_{row(s_n)}(s_n) \in \cap M(LMLP_M)$. Hence $R_{row(t)}(t) \in \cap M(LMLP_M)$ is true.

□

Lemma 4.2 Suppose that (S, E, M) is a closed, consistent predicate characterization matrix such that S is subtree-closed and E is \$-prefix-closed with respect to S. For

the constructed linear monadic logic program $LMLP_M$ and for any tree t over Γ, there is exactly one function value row(s) such that $R_{row(s)}(t) \in \cap M(LMLP_M)$ and $s \in S$.

(Proof) We prove it by the induction on the depth of t. Suppose first that the depth of t is 0, i.e. $t = a \in \Gamma_0$. By the definition of $LMLP_M$, for $a \in \Gamma_0$, row(a) is exactly one function value such that $R_{row(a)}(a) \in \cap M(LMLP_M)$ and $a \in S$. Next suppose that the result holds for all trees with depth at most h. Let t be a tree of depth $h+1$, so that $t = f(u_1,...,u_n)$ for some trees $u_1,...,u_n$ with depth at most h and some f in Γ_n. There are several clauses of the form : $R_{row(f(v_1,...v_n))}(f(x_1,...,x_n)) \leftarrow R_{row(v_1)}(x_1),...,R_{row(v_n)}(x_n)$ in $LMLP_M$. However by the induction hypothesis, for each u_i ($1 \leqq i \leqq n$), there is exactly one function value, say $row(s_i)$, such that $R_{row(s_i)}(u_i) \in \cap M(LMLP_M)$ and $s_i \in S$. Since (S, E, M) is consistent, there is only one clause of the form : $R_{row(f(s_1,...s_n))}(f(x_1,...,x_n)) \leftarrow R_{row(s_1)}(x_1),...,R_{row(s_n)}(x_n)$ in $LMLP_M$. Thus $row(f(s_1,...,s_n))$ is exactly one function value such that $R_{row(f(s_1,...s_n))}(f(u_1,...,u_n)) \in \cap M(LMLP_M)$, and since (S, E, M) is closed, $row(f(s_1,...,s_n))$ is equal to row(s) for some s in S. Hence there is exactly one function value row(s) such that $R_{row(s)}(t) \in \cap M(LMLP_M)$ and $s \in S$.

□

Lemma 4.3 (replacement) Suppose that (S, E, M) is a closed, consistent predicate characterization matrix such that S is subtree-closed and E is \$-prefix-closed with respect to S and that $LMLP_M$ is the constructed linear monadic logic program. Suppose that $R_{row(s)}(t) \in \cap M(LMLP_M)$, $R_{row(s')}(t') \in \cap M(LMLP_M)$ and $row(s) = row(s')$ for s, s' in $(S \cup X(S))$ and trees t, t' over Γ. For e in E, $P(e \cdot t) \in \cap M(LMLP_M)$ iff $P(e \cdot t') \in \cap M(LMLP_M)$.

(Proof) We prove it by induction on the depth of \$ in e. When e is \$, if $P(e \cdot t) = P(t) \in \cap M(LMLP_M)$, then there is a ground instance $P(t) \leftarrow R_{row(s_0)}(t)$ of a clause $P(x) \leftarrow R_{row(s_0)}(x)$ in $LMLP_M$ such that $R_{row(s_0)}(t) \in \cap M(LMLP_M)$ and $s_0 \in S$. By lemma 4.2, $row(s_0) = row(s)$. By the assumption, $row(s_0) = row(s')$ and $R_{row(s')}(t') \in \cap M(LMLP_M)$. Hence $P(t') \in \cap M(LMLP_M)$. Interchanging the roles of s and s' and of t and t', we obtain the converse.

Next suppose that the result holds for all e in E where the depth of \$ is at most h. Let e be an element of E where the depth of \$ is h+1. Since E is \$-prefix-closed with respect to S, $e = e' \cdot f(s_1,...,s_{i-1},\$,s_i,...,s_{n-1})$ for some $f \in \Gamma_n$, $s_1,...,s_{n-1} \in S$, i $(1 \leqq i \leqq n)$ and some e' in E where the depth of \$ is h. Since (S, E, M) is closed, there is some s_0 in S such that $row(s_0) = row(s)$. Then $R_{row(s_0)}(t) \in \cap M(LMLP_M)$ and by lemma 4.1, $R_{row(s_1)}(s_1),...,R_{row(s_{n-1})}(s_{n-1}) \in \cap M(LMLP_M)$. By the definition of $LMLP_M$, there is a clause of the form

$R_{row(f(s_1,...,s_{i-1},s_0,s_i,...,s_{n-1}))}(f(x_1,...,x_n)) \leftarrow R_{row(s_1)}(x_1),...,R_{row(s_0)}(x_i),...,R_{row(s_{n-1})}(x_n)$

in $LMLP_M$ and so $R_{row(f(s_1,...,s_{i-1},s_0,s_i,...,s_{n-1}))}(f(s_1,...,s_{i-1},t,s_i,...,s_{n-1})) \in \cap M(LMLP_M)$. Since $row(s_0) = row(s')$ and $R_{row(s')}(t') \in \cap M(LMLP_M)$,

$R_{row(f(s_1,...,s_{i-1},s_0,s_i,...,s_{n-1}))}(f(s_1,...,s_{i-1},t',s_i,...,s_{n-1})) \in \cap M(LMLP_M)$.

By the induction hypothesis, $P(e' \cdot f(s_1,...,s_{i-1},t,s_i,...,s_{n-1})) \in \cap M(LMLP_M)$ iff $P(e' \cdot f(s_1,...,s_{i-1},t',s_i,...,s_{n-1})) \in \cap M(LMLP_M)$. Therefore $P(e \cdot t) \in \cap M(LMLP_M)$ iff $P(e \cdot t') \in \cap M(LMLP_M)$.

□

Theorem 4.4 Suppose that (S, E, M) is a closed, consistent predicate characterization matrix such that S is subtree-closed and E is \$-prefix-closed with respect to S. Then the constructed linear monadic logic program $LMLP_M$ agrees with the data in M. That is, for every tree s in $(S \cup X(S))$ and e in E, $P(e \cdot s) \in \cap M(LMLP_M)$ iff $D(M)(e \cdot s) = 1$.

(Proof) We prove it by induction on the depth of \$ in e. When e is \$ and s is any element of $(S \cup X(S))$, by lemma 4.1, $R_{row(e \cdot s)}(e \cdot s) = R_{row(s)}(s) \in \cap M(LMLP_M)$. If s is in S, then by the definition of $LMLP_M$, $P(x) \leftarrow R_{row(s)}(x)$ in $LMLP_M$ iff $D(M)(s) = 1$. Hence $P(s) \in \cap M(LMLP_M)$ iff $D(M)(s) = 1$. If s is in X(S), then since (S, E, M) is closed, $row(s) = row(s')$ for some s' in S, and $P(x) \leftarrow R_{row(s')}(x)$ in $LMLP_M$ iff $D(M)(s') = 1$, and so $P(x) \leftarrow R_{row(s)}(x)$ in $LMLP_M$ iff $D(M)(s) = 1$. Hence $P(s) \in \cap M(LMLP_M)$ iff $D(M)(s) = 1$.

Next suppose that the result holds for all e in E where the depth of \$ is at most h.

Let e be an element of E where the depth of \$ is h + 1. Since E is \$-prefix-closed with respect to S, $e = e' \cdot f(s_1,...,s_{i-1},\$,s_i,...,s_{n-1})$ for some $f \in \Gamma_n$, $s_1,...,s_{n-1} \in S$, i $(1 \leq i \leq n)$ and some e' in E where the depth of \$ is h. For any element s of $(S \cup X(S))$, since (S, E, M) is closed, there is an element s' in S such that row(s) = row(s'). By lemma 4.1, $R_{row(s)}(s) \in \cap M(LMLP_M)$ and $R_{row(s')}(s') \in \cap M(LMLP_M)$. Then by replacement lemma 4.3,

$P(e \cdot s) \in \cap M(LMLP_M)$

iff $P(e \cdot s') \in \cap M(LMLP_M)$

iff $P(e' \cdot f(s_1,...,s_{i-1},\$,s_i,...,s_{n-1}) \cdot s') \in \cap M(LMLP_M)$

iff $P(e' \cdot f(s_1,...,s_{i-1},s',s_i,...,s_{n-1})) \in \cap M(LMLP_M)$.

By the induction hypothesis,

$P(e' \cdot f(s_1,...,s_{i-1},s',s_i,...,s_{n-1})) \in \cap M(LMLP_M)$ iff $D(M)(e' \cdot f(s_1,...,s_{i-1},s',s_i,...,s_{n-1})) = 1$.

Since row(s) = row(s') and (S, E, M) is consistent,

$row(f(s_1,...,s_{i-1},s',s_i,...,s_{n-1})) = row(f(s_1,...,s_{i-1},s,s_i,...,s_{n-1}))$

and hence $D(M)(e' \cdot f(s_1,...,s_{i-1},s',s_i,...,s_{n-1})) = D(M)(e' \cdot f(s_1,...,s_{i-1},s,s_i,...,s_{n-1}))$,

and since $e' \cdot f(s_1,...,s_{i-1},\$, s_i,...,s_{n-1}) = e$ is in E, $D(M)(e' \cdot f(s_1,...,s_{i-1},s,s_i,...,s_{n-1}))$ $= D(M)(e \cdot s)$. Therefore $P(e \cdot s) \in \cap M(LMLP_M)$ iff $D(M)(e \cdot s) = 1$.

□

For the following result, for a tree automaton $T_A = (Q, \Gamma, \delta, F)$ we extend δ to $(\Gamma \cup Q)^T$ by letting : $\delta(q) = q$ for $q \in Q$, where Q is considered as a set of 0-ary constant symbols. In this definition, if $q = \delta(s)$ for $q \in Q$ and $s \in \Gamma^T$, then $\delta(t(x \leftarrow q)) = \delta(t(x \leftarrow s))$ for $t \in \Gamma^T$ and $x \in Dom(t)$.

Theorem 4.5 Suppose that (S, E, M) is a closed, consistent predicate characterization matrix such that S is subtree-closed and E is \$-prefix-closed with respect to S. Suppose that the constructed linear monadic logic program $LMLP_M$ from (S, E, M) has n predicates. If $T_A = (Q, \Gamma, \delta, F)$ is any tree automaton which agrees with the data in M that has n or fewer states and $LMLP_{T_A}$ is a corresponding linear monadic logic program in the sense of definition (A), then $LMLP_M$ is isomorphic to $LMLP_{T_A}$.

(Proof) We prove it by exhibiting an isomorphism ϕ. First define, for each q in Q, row(q) to be the finite function f from E to {0, 1} such that f(e)=1 iff δ(e·q) is in F. Since T_A agrees with the data in M, for each s in (S∪X(S)) and each e in E, δ(e·s) is in F iff D(M)(e·s)=1, so row(δ(s)) is equal to row(s) in (S, E, M). Hence as s ranges over all of S, row(δ(s)) ranges over all the elements of Q, so T_A must have at least n states, i.e., it must have exactly n states. Thus, for each s in S there is a unique q in Q such that row(s)=row(q), namely, δ(s). Next define for each s in S, ϕ(row(s)) to be δ(s). This mapping is one-to-one and onto, We must verify that it preserves the clauses. For each $s_1,...,s_n$ in S and $f \in \Gamma_n$, let s be an element of S such that row($f(s_1,...,s_n)$) =row(s). Then

$$R_{\phi(row(f(s_1,...,s_n))}(x) = R_{\phi(row(s)}(x)$$
$$= R_{\delta(s)}(x)$$

Also,

$$R_{\delta(f,\delta(s_1),...,\delta(s_n))}(x) = R_{\delta(f(s_1,...,s_n))}(x)$$

Since δ(s) and $\delta(f(s_1,...,s_n))$ have identical row values, namely row(s) and row($f(s_1,...,s_n)$), they must be the same state of T_A. Hence the mapping ϕ carries the clause $R_{row(f(s_1,...s_n))}(f(x_1,...,x_n)) \leftarrow R_{row(s_1)}(x_1),...,R_{row(s_n)}(x_n)$ in $LMLP_M$ to the clause $R_{\delta(f,\delta(s_1),...,\delta(s_n))}(f(x_1,...,x_n)) \leftarrow R_{\delta(s_1)}(x_1),...,R_{\delta(s_n)}(x_n)$ in $LMLP_{T_A}$. Since if $P(x) \leftarrow R_{row(s)}(x)$ for some s in S, then D(M)(s)=1, so since ϕ(row(s)) is mapped to a state q with row(q)=row(s), it must be that q is in F and hence $P(x) \leftarrow R_q(x)$. Conversely, if row(s) is mapped to a state q such that $P(x) \leftarrow R_q(x)$ is in $LMLP_{T_A}$, then since q is in F and row(q)=row(s), D(M)(s)=1, so $P(x) \leftarrow R_{row(s)}(x)$ is in $LMLP_M$. So we conclude that the mapping ϕ preserves the clauses.

□

5. Inductive inference algorithm for linear monadic logic program

First we confirm the inductive inference schema of linear monadic logic program. The problem is to identify the denotation of the predicate P in the unknown model.

That is, in our setting the problem is to infer a linear monadic logic program LMLP such that the denotation of P in $\cap M(LMLP)$ is equal to the one in the unknown model.

(Algorithm of inductive inference for linear monadic logic program)

Input : An oracle EX() for a sufficient set of examples (or facts of ground atoms) of the predicate P in the unknown linear monadic logic program LMLP,

An oracle MEMBER(P(t)) on a ground atom P(t) as input for a membership query to output 1 or 0 according to whether P(t) is true in $\cap M(LMLP)$,

Output : A sequence of conjectures of linear monadic logic program,

Procedure :

$S := \varnothing$; $E := \{\$\}$; $LMLP := \varnothing$; $Examples := \varnothing$;

do forever

add an example EX() to Examples;

while there is an negative example $-P(t) \in Examples$ such that $LMLP \vdash P(t)$

or there is an positive example $+P(t) \in Examples$ such that $LMLP \nvdash P(t)$;

add t and all its subtrees to S;

extend (S, E, M) to $E \cdot (S \cup X(S))$ using MEMBER;

repeat

if (S, E, M) is not consistent

then find s_1 and s_2 in S, $f \in \Gamma_n$, $u_1,...,u_{n-1} \in S$, $e \in E$, and i $(1 \leqq i \leqq n)$ such that $row(s_1)$ is equal to $row(s_2)$ and

$D(e \cdot f(u_1,...,u_{i-1},s_1,u_i,...,u_{n-1})) \neq D(e \cdot f(u_1,...,u_{i-1},s_2,u_i,...,u_{n-1}))$;

add $e \cdot f(u_1,...,u_{i-1},\$,u_i,...,u_{n-1})$ to E;

extend (S, E, M) to $E \cdot (S \cup X(S))$ using MEMBER;

if (S, E, M) is not closed;

then find $f(\hat{u}) \in X(S)$ for $\hat{u} \in S^n$ and $f \in \Gamma_n$ such that $row(f(\hat{u}))$ is different from row(s) for all $s \in S$;

add $f(\hat{u})$ to S;

```
        extend (S, E, M) to E·(S∪X(S)) using MEMBER;
    until (S, E, M) is closed and consistent;
    LMLP := LMLP_M;
  end;
  output LMLP;
end.
```

In the above algorithm, the operation of "extend (S, E, M) to E·(S∪X(S)) using MEMBER" is the operation to extend D(M) by asking membership queries for missing elements. We call an example t presented by the oracle EX a *counter-example* when the last conjecture $LMLP_M$ does not agree with t.

6. Correctness and complexity

To see that the algorithm is correct, i.e. the algorithm identifies a linear monadic logic program LMLP in the limit such that $\{t : P(t) \in \cap M(LMLP)\}$ is the denotation of P by the unknown model, it is enough for us to show that the constructed predicate characterization matrix (S, E, M) during the running of the algorithm is a closed, consistent one such that S is subtree-closed and E is \$-prefix-closed with respect to S, and that the while loop of the algorithm is executed at most in a finite time during the running of the algorithm.

Lemma 6.1 Let (S, E, M) be a predicate characterization matrix such that S is subtree-closed and E is \$-prefix-closed with respect to S. Let n be the number of different values of row(s) for s in S. Any deterministic tree automaton which agrees with the data in M must have at least n states.

(Proof) Let $T_A = (Q, \Gamma, \delta, F)$ be a deterministic tree automaton which agrees with the data in M. Suppose that s_1 and s_2 are elements of S such that $row(s_1)$ and $row(s_2)$ are distinct. Then there exists e in E such that $D(M)(e \cdot s_1) \neq D(M)(e \cdot s_2)$. Since T_A agrees with the data in M, exactly one of $\delta(e \cdot s_1)$ and $\delta(e \cdot s_2)$ is in F. Thus $\delta(s_1)$ and $\delta(s_2)$ must

be distinct states because T_A is deterministic. Since $\delta(s)$ takes on at least n different values as s ranges over S, T_A must have at least n states.

□

Lemma 6.2 The while loop of the algorithm is executed at most in a finite time during the running of the algorithm.

(Proof) Let n be the number of states in the minimum state deterministic tree automaton for the denotation of the predicate P in the unknown model. Firstly we will show that whenever a predicate characterization matrix (S, E, M) is not consistent or not closed, the number of distinct values row(s) for s in S must increase. If (S, E, M) is not consistent, then since two previously equal row values, $row(s_1)$ and $row(s_2)$, are no longer equal after E is augmented, the number of distinct values row(s) for s in S must increase by at least one. If (S, E, M) is not closed and a tree $f(\hat{u})$ is added to S, then since $row(f(\hat{u}))$ is different from row(s) for all s in S before S is augmented, the number of distinct values row(s) must increase by at least one.

Next we will show that whenever a tree t and all its subtree are added to S and (S, E, M) is extended because $LMLP_M$ does not agree with t, the extended predicate characterization matrix (S', E', M') is always not consistent or not closed. If a conjecture $LMLP_M$ is found to be incorrect by the example t, then since the minimum state tree automaton T_A is correct for the data in M and $LMLP_{T_A}$ is inequivalent to $LMLP_M$ (since they disagree on t), by theorem 4.5, T_A must have at least one more state. By lemma 6.1, the number of distinct row values in a predicate characterization matrix which is correct for the data in M and t must be same as the number of states in T_A. Thus (S', E', M') must be not consistent or not closed to increase the number of distinct row values.

Then by these and lemma 6.1, a counter-example is added to S at most n times during the running of the algorithm. Thus, the while loop is executed at most in a finite time.

□

By the above result, it follows that the algorithm makes at most a finite number of conjectures.

Lemma 6.3 The conjectures which the algorithm makes are correct for the facts known by the oracles EX and MEMBER.

(Proof) We will show that each predicate characterization matrix (S, E, M) during the running of the algorithm is a closed, consistent one such that S is subtree-closed and E is \$-prefix-closed with respect to S. In the algorithm, there are three operations which extend the row or the column of (S, E, M). When t and all its subtrees are added to S, S obviously remains subtree-closed. If (S, E, M) is not consistent, then for some $f \in \Gamma_n$, $u_1,...,u_{n-1} \in S$, $e \in E$, and i $(1 \leqq i \leqq n)$, $e \cdot f(u_1,...,u_{i-1},\$,u_i,...,u_{n-1})$ is added to E. In this case, E remains \$-prefix-closed with respect to S. If (S, E, M) is not closed, then for some $\hat{u} \in S^n$ and $f \in \Gamma_n$, $f(\hat{u})$ is added to S. In this case, S remains subtree-closed. Since the repeat loop is repeated as long as (S, E, M) is not closed and consistent, by lemma 6.2, each constructed (S, E, M) must eventually be closed and consistent. Thus each constructed (S, E, M) during the running of the algorithm is a closed, consistent one such that S is subtree-closed and E is \$-prefix-closed with respect to S. Then by theorem 4.4, the conjectures of linear monadic logic program which the algorithm makes are correct for the facts known by the oracles EX and MEMBER.

□

Now we conclude the following theorem.

Theorem 6.4 The algorithm identifies in the limit a linear monadic logic program LMLP such that $\{t : P(t) \in \cap M(LMLP)\}$ is equal to the denotation of P by the unknown model.

Next we will analyse the time complexity of the algorithm. By lemma 6.2, the while loop of the algorithm is executed at most in a finite time. Then how much time does the while loop consume during the running of the algorithm. That depends

partly on the size of the examples t presented by the oracle EX. We will analyze the running time of the while loop as a function of n, the number of states in the minimum tree automaton for the denotation of the predicate P in the unknown model, and m, the maximum size of any counter-examples presented by EX during the running of the algorithm, where the *size* of an example is the number of symbols in its textual representation. We will show that its running time is bounded by a polynomial in m and n. Let k be the cardinality of the alphabet Γ and d be the maximum arity of the function symbols in Γ. We may assume $d \geq 1$.

Whenever (S, E, M) is discovered to be not closed, one element is added to S. Whenever (S, E, M) is discovered to be not consistent, one element is added to E. For each counter-example t of size at most m presented by the oracle EX, at most m subtrees are added to S. Since the predicate characterization matrix is discovered to be not consistent at most $n-1$ times, the total number of trees in E cannot exceed n. Since the predicate characterization matrix is discovered to be not closed at most $n-1$ times, and there can be at most n counter-examples, the total number of strings in S cannot exceed $n+mn$. Thus, the maximum cardinality of $E \cdot (S \cup X(S))$ is at most

$$n((n+mn)+k(n+mn)^d) = O(m^d n^{d+1}).$$

Now we consider the operations in the while loop executed by the algorithm. Checking the predicate characterization matrix to be closed and consistent can be done in time polynomial in the size of the matrix and must be done at most n times. Adding a tree to S or E requires at most $O(m^d n^d)$ membership queries to extend the matrix. When the predicate characterization matrix is closed and consistent, $LMLP_M$ may be constructed in time polynomial in the size of the matrix, and this must be done at most n times. A counter-example requires the addition of at most m subtrees to S, and this can be also happen at most n times.

Therefore, the total time which the while loop consumes during the running of the algorithm can be bounded by a polynomial function of m and n.

7. Concluding remarks

We remark on related work. Shapiro's Model Inference System (MIS for short) [8,9] is the excellent and only existing system to infer logic programs or Herbrand models in first order logic using the concept of identification in the limit defined by Gold [4]. MIS can infer a whole class of logic programs, but ours only for a restricted class of logic programs. However our algorithm has several unique features compared with MIS. (1) As we mentioned in the introduction, our algorithm is based on algebraic semantics and the target of the inference is a tree language computed by a logic program, and hence it is different from Shapiro's approach and is not model inference. (2) In general, it is not easy to analyse the time complexity of inductive inference algorithm, and neither in MIS. We have shown in the last section the time complexity of our algorithm in the very clear manner. (3) Our algorithm is based on the constructive method, while MIS is based on the enumerative method, where the constructive method systematically use examples to construct the conjecture and the enumerative method use them to select a conjecture in enumeration. It is said that the constructive method is in general more efficient than the enumerative method. (4) In our algorithm, the predicate symbol P and its interpretation are only given as the observational language and the oracle, and any information about the hypothesis language is not given. The algorithm automatically generates other predicates whenever they are needed. However in MIS, all predicates used to construct the conjectures and those intended interpretations must also be given as the hypothesis language and the oracle, and this is often referred to as the problem about *theoretical terms* of MIS, as pointed out in [6].

Finally we will briefly mention some application of our algorithm. It is known that the frontier set of a rational set of trees is a context free language and vice versa. Based on this fact, our algorithm can be applied to inferring parsers of context free languages from the *shapes* of their structural descriptions. In the case of context free

grammars, the shapes of their structural descriptions mean the shapes of the derivation trees. Such structural descriptions are called *skeletons*. Thus a skeleton is a kind of tree whose interior nodes have no label. A tree automaton which recognizes a set of skeletons is called a *skeletal automaton*. According to [6], given a context free language L over the alphabet Σ, there is a skeletal automaton S_A such that the frontier set of the skeletal set accepted by S_A is exactly L, and conversely the frontier set of the skeletal set accepted by any skeletal automaton is a context free language. Then we will use our algorithm to infer the unknown skeletal automaton from their skeletal examples and construct a parser corresponding to the unknown context free grammar. In this case, the construction of a conjecture from the matrix (S, E, M) is only changed into the construction of a parser. For example, the constructed parsing prolog program over Σ using difference-lists from (S, E, M) is defined with predicate set Pre, calling predicate Sentence, and the set of clauses $PARSER_M$ as follows.

$Pre = \{R_{row(s)}(x,x') : s \in S\}$,

$PARSER_M = \{Sentence(x_0,x_1) \leftarrow R_{row(s)}(x_0,x_1) : s \in S \text{ and } D(M)(s) = 1\}$

$\cup \{R_{row((s_1,\ldots,s_n))}(x_0,x_n) \leftarrow R_{row(s_1)}(x_0,x_1),\ldots,R_{row(s_n)}(x_{n-1},x_n) : n > 0\}$

$\cup \{R_{row(a)}([a|x],x) \leftarrow : a \in \Sigma\}$.

The detail discussions about this work will appear elsewhere.

This is part of the work in the major R&D of the Fifth Generation Computer Project, conducted under program set up by MITI.

References

[1] Angluin,D., Learning regular sets from queries and counter-examples, *Yale DCS TR-464*, 1986.

[2] Courcelle,B., Fundamental properties of infinite trees, *TCS 25*, 95-169 (1983).

[3] van Emden,M.H., Kowalski,R.A., The semantics of predicate logic as a programming language, *JACM 23*, 733-742 (1976).

[4] Gold,E.M., Complexity of automaton identification from given data, *Information and Control 37*, 302-320 (1978).

[5] Ishizaka,H., Inductive inference of regular languages based on model inference, to appear in *Logic Programming Conference '87*, Tokyo, 1987.

[6] Levy,L.S., Joshi,A.K., Skeletal structural descriptions, *Information and Control 39*, 192-211 (1978).

[7] Marque-Pucheu,G., Rational set of trees and the algebraic semantics of logic programming, *Acta Informatica 20*, 249-260 (1983).

[8] Shapiro,E., Algorithmic program debugging, MIT Press, 1983.

[9] Shapiro,E., Inductive inference of theories from facts, *Yale DCS TR-192*, 1981.

[10] Thatcher,J.W., Tree automaton : An informal survey, in *Currents in the Theory of Computing*, Aho,A.V., Ed., Prentice-Hall, 1973.