Machine Intelligence 13, Oxford University Press, 59-85, 1994.

A Generalization of the Least General Generalization

HirokiArimura, Takeshi Shinohara, Setsuko Otsuki

Department of ArtificialIntelligence
Kyushu Institute of Technology

Hiroki Ishizaka

FUJITSU LABORATORIES, ISIS

Abstract

In this chapter, we present a polynomial time algorithm, called a k-minimal multiple generalization (k-*mmg*) algorithm, where $k \geq 1$, and its application to inductive learning problems. The algorithm is a natural extension of the least general generalization algorithm developed by Plotkin and Reynolds. Given a finite set of ground first-order terms, the k-*mmg* algorithm generalizes the examples by at most k first-order terms, while Plotkin's algorithm does by a single first-order term. We apply the k-*mmg* algorithm to several learning problems in inductive logic programming, and knowledge discovery in databases.

1 Introduction

Inductive inference is a process to guess or identify an unknown general rule from its examples. An inference algorithm receives finite examples and produces a generalization of them as a hypothesis.

This paper concerns with inference only from positive examples. For example, recently, a number of studies are aimed at knowledge discovery in databases (Piatetsky-Shapiro and Frawley 1991). Usually, we think of a database as a collection of positive examples drawn from some rule that applies in the real world. To discover such a rule in the database, inductive inference from only positive examples rather than from both positive and negative examples is natural.

However, in general, any successful inference from positive data should avoid *overgeneralizations*. In other words, the key notion in inference from positive data is a *minimal* generalization. Consider the case when an unknown rule to be inferred is represented by a single first-order term. Then,

an algorithm developed by Plotkin (1970) and Reynolds (1970), efficiently finds the minimal generalization of given examples. This generalization is called the *least general generalization* (*lgg*, for short). For this reason, the least general generalization algorithm plays an important role in model inference systems (Shapiro 1981; Ishizaka 1988) and inductive logic programming (Muggleton 1990).

On the other hand, when some rules in question are too complex to be represented in a single term, the least general generalization algorithm cannot be directly applied because it may produce an overgeneralization. For example, assume that the following examples are given.

$$\left\{ \begin{array}{c} app([],[],[]),\ app([b],[a],[b,a]),\ app([a],[],[a]), \\ app([],[a],[a]),\ app([a,b],[c,d],[a,b,c,d]) \end{array} \right\}$$

Clearly, the least general generalization of them becomes a most general term $app(x,y,z)$. However, if we generalize the examples by a set of several terms instead of a single term, we might get a less general generalization. For any positive integer k, we call a minimal generalization by at most k terms a *k-minimal multiple generalization* (*k-mmg*, for short). For example, the pair

$$\{\, app([],X,X), app([A|X],Y,[A|Z]) \,\}$$

is a 2-*mmg* of the examples above. Since the notion of 1-*mmg* coincides with the one of *lgg*, *k-mmg* is a generalization of the least general generalization.

Clearly, there exist several *k-mmg*'s for a finite set of examples, while the *lgg* is unique. We say that a *k-mmg* is reduced with respect to a set of examples when it has no redundancy. We can show that the set of *all* the reduced k-mmg's for a set of examples is hard to compute. On the contrast, *one* of the k-mmg's can be found in polynomial time under the assumption of the compactness with respect to containment. This assumption ensures that containment relations are characterized by syntactic relations.

In Section 2, first we introduce *lgg* according to Plotkin (1970) and Reynolds (1970). Then we define *mmg* in Section 3 by generalizing *lgg* from a single first-order term to a set of first-order terms, and present a *k-mmg* algorithm that finds a *k-mmg* in polynomial time. The results in the section are obtained from our work (Arimura *et al.* 1991 In Section 4, we apply the *k-mmg* algorithm to problems in inductive logic programming. We introduce three subclasses of logic programs, unit clause programs, context-free transformations with a flat base and primitive Prologs, and show that these subclasses are polynomial time inferable from positive examples. The results in this section are obtained from previous works (Arimura *et al.* 1992b; Ishizaka *et al.* 1992). In Section 5, we show the method is also

applicable to discovery of a set of rules that characterize a concept in a database.

2 Preliminaries

For a finite set A, we denote by $\sharp A$ the number of the elements in A. Let Σ be a finite set of *function symbols* and X be a countable set of *variables* disjoint from Σ, where a mapping *arity* from function symbols to positive numbers is associated with Σ. We call Σ an *alphabet*. A *first-order term* (or a *term*) is either a variable, a 0-ary function symbol, or a string $f(t_1, \ldots, t_n)$ that is recursively constructed from an n-ary function symbol f and terms $t_1, \ldots, t_n$. A term is *ground* if it contains no variable. We denote by $\mathcal{TP}$ the set of first-order terms and by $\mathcal{T}$ the set of ground first-order terms. The *size of term* p is the total number $|p|$ of occurrences of function symbols and variable in p, and *the size of set* P *of terms* is $||P|| = \sum_{p \in P} |p|$.

A *substitution* is a mapping θ from terms to themselves satisfying that for any term $t = f(t_1, \ldots, t_n)$, $\theta(t) = f(\theta(t_1), \ldots, \theta(t_n))$. A set of replacement $\{x_1 := t_1, \ldots, x_n := t_n\}$ denotes the substitution that maps variable x_i to term t_i and any other variable to itself.

We define a binary relation $\leq'$ on $\mathcal{T}$ by $p \leq' q \iff p = \theta(q)$ for some substitution θ. If $p \leq' q$ then we say p is an *instance* of q, p is a *generalization* of q, p is more *specific* than q, or q is more *general* than p. We also define $<'$ by $p \leq' q \iff p \leq' q$ but $q \not\leq' p$. Let $\mathcal{TP}_{\equiv}$ be the set of representatives of the equivalence classes of $\mathcal{TP}$ modulo $\equiv$, where $p \equiv q \iff p \leq' q$ and $q \leq' p$. In Section 2 and Section 3, we write $\mathcal{TP}$ for $\mathcal{TP}_{\equiv}$ without extra notice.

Let P be a finite set of terms. If $P \subseteq L(p)$, then we say a term p is a *common generalization* of S. The *least general generalization* (*lgg*, for short) of P is a common generalization p of P such that $p \leq' q$ for any common generalization q of P. The *lgg* of P is computable in polynomial time in $||P||$ by the anti-unification algorithm (Plotkin 1970, Reynolds 1970).

The *language* defined by term p is the set $L(p) = \{ w \in \mathcal{T} \mid w \leq' p \}$, that is, the set of ground instances of p. A set L of ground terms is a *tree pattern language* if $L = L(p)$ for some p. By definition, $p \leq' q \Longrightarrow L(p) \subseteq L(q)$. In this case, the converse also holds.

Lemma 1. (Reynolds 1970) *If* $\sharp\Sigma > 1$, *then* $L(p) \subseteq L(q) \iff p \leq' q$.

By Lemma 1, we can define the *lgg* of S as a term that defines a minimal tree pattern language containing S with respect to set inclusion $\subseteq$.

3 Polynomial time k-mmg algorithm

In this section, we present a polynomial time algorithm to compute a k-minimal multiple generalization of a finite set of ground terms. Let k be a positive integer. A *k-multiple term* is a set P of at most k first-order terms, and the *language* defined by P is the set $L(P) = \bigcup_{p \in P} L(p)$. Two k-multiple generalizations P and Q are *equivalent* if they define the same language. We denote by $\mathcal{TP}^k$ the class of k-multiple terms. If $S \subseteq$

$L(P)$, we say P is a *k-multiple generalization* of S. A *k-minimal multiple generalization* (*k-mmg*, for short) of S is a k-multiple generalization P of S such that $L(Q) \not\subset L(P)$ for any k-multiple generalization Q of S. Note that there may be more than one *k-mmg* of S if $k > 1$, while the *lgg* of S is unique. For example, let $S = \{f(a,a), f(a,b), f(b,b)\}$ and $\Sigma = \{a, b, f(\cdot,\cdot)\}$. Then, $P_1 = \{f(x,x), f(a,b)\}$ and $P_2 = \{f(a,a), f(y,b)\}$ are both 2-*mmg* of S.

In this paper, we are concerning with an efficient algorithm that, given S, finds one of *k-mmg*'s of S. Indeed, it is reasonable to give up finding all the answer because we can show by a reduction from an NP-complete problem that it is hard to compute all the *k-mmg*'s of S in polynomial time.

A set P in $\mathcal{TP}^k$ is *reduced with respect to* S iff $S \subseteq L(P)$, but $S \not\subseteq L(Q)$ for any proper subset Q of P. Any *k-mmg* of S is equivalent to a member Q in $\mathcal{TP}^k$ that is reduced with respect to S, and there are only finitely many such Q's. Thus, by the definition of *mmg*, if the decision of "$L(P) \subseteq L(Q)$?" is computable (indeed, it is possible from Theorem 2 below), we can use an exhaustive search method to compute a *k-mmg* of S. However, this simple method may not efficiently work. Because even for a fixed $k > 0$, S may have exponentially many reduced *k-mmg*'s. Furthermore, the observation on hardness of computing all solutions mentioned above also leads to the difficulty.

Recall that k-multiple generalizations are defined through the class of languages defined by sets in $\mathcal{TP}^k$. Consider the following property of the class: for any terms $p, q_1, \ldots, q_m$ $(1 \leq m \leq k)$,

$$L(p) \subseteq L(q_1) \cup \cdots \cup L(q_m) \implies L(p) \subseteq L(q_i) \text{ for some } 1 \leq i \leq m.$$

We call this property the *compactness with respect to containment* of $\mathcal{TP}^k$. If Σ is sufficiently large, $\mathcal{TP}^k$ has this property. Lassez and Marriott (1986) showed the property in the case that Σ is infinite. The next theorem improves their result.

Theorem 2. (Arimura *et al.* 1992a) *Let $k > 0$ and Σ be an alphabet with $\sharp\Sigma > k$. Then the class $\mathcal{TP}^k$ has the compactness with respect to containment.*

The condition on $\sharp\Sigma$ above is necessary. For example, let $k = 2$ and Σ consists of two symbols $a, f(\cdot,\cdot)$. For terms $p = x$, $q_1 = a$ and $q_2 = f(x,y)$, we see $L(p) \subseteq L(q_1) \cup L(q_2)$ but none of q_1 and q_2 is a generalization of p.

Hereafter, we assume $\sharp\Sigma > k$ for the compactness. Then we can use Theorem 2 to show the necessary and sufficient condition for a k-mutiple generalization P consisting of exactly k terms to be a *k-mmg* of a finite set S. A k-multiple generalization P is said to be of *normal form with respect to* S iff p is the *lgg* of $S - L(P \setminus p)$ for any $p \in P$, where $P \setminus p$ is the set

Algorithm 1: The algorithm $MMG(k, S)$

Input: A positive integer k and a finite set $S \subseteq \mathcal{T}$.
Output: A k-*mmg* of S.
Procedure:
1 **if** $k = 1$ **then return** $lgg(S)$
2 **else**
3 $P(= \{p_1, \ldots, p_k\}) := REDUCED(k, S)$;
4 **if** P is found **then**
5 **for** each $i = 1, \ldots, k$ **do** /* tightening process */
6 replace p_i in P by $lgg(S - L(P \setminus p_i))$;
7 **return** P;
8 **else return** $MMG(k-1, S)$;

obtained by removing p from P. Assume that P is of normal form with respect to S, and also assume to contradict that there is some Q satisfying $S \subseteq L(Q) \subset L(P)$. If $\sharp Q < \sharp P$ then compactness shows that P is not reduced with respect to S; contradiction. On the other hand, if $\sharp Q = \sharp P$, then the compactness shows that P can not be of normal form. Hence, the following holds.

Theorem 3. *Let P be a mutiple generalization of S that is reduced with respect to S and $\sharp P = k$. Then, P is of normal form with respect to S iff P is k-mmg of S.*

Now, we give a polynomial time algorithm $MMG(k, S)$ in Algorithm 1 that finds a k-*mmg* based on a greedy search. First, the MMG starts from any k-multiple generalization of S that is reduced with respect to S. The next theorem ensures that such a candidate can be efficiently found (we prove the theorem in the latter half of this section).

Theorem 4. (Arimura *et al.* 1991) *For any $k > 0$ and any finite set S of ground terms, a set consisting of exactly k terms that is reduced with respect to S can be found in polynomial time with respect to $||S||$ if it exists.*

Once a general candidate is obtained, the algorithm searches k-mmg in the direction from general one to most specific one. The algorithm tries to make P more specific by replacing each component p in P by the more specific term $lgg(S - L(P \setminus p))$. We call this process *tightening*. Any fixed point of this tightening process is of normal form with respect to S.

Lemma 5. *In MMG in Algorithm 1, assume that a set of exactly k terms that is reduced with respect to S is found at Line 1. Then after executing the lines from Line 2 to Line 6, any P that MMG returns at Line 7 is of normal form with respect to S.*

Algorithm 2: The algorithm $REDUCED(k, S)$

Where gci S is the greatest common instance of a set of terms.

Input: A positive integer k and a finite set S of ground terms.
Output: A set of exactly k terms reduced with respect to S.
Procedure:

```
for each V ⊆ S with ♯V = k do
    P_1 := ∅, ..., P_k := ∅;
    for each v ∈ V do
        let {v_1, ..., v_{k-1}} := V \ v;
        for each q_1 ∈ MAXT(v, v_1), ..., q_{k-1} ∈ MAXT(v, v_{k-1}) do
            P_v := P_v ∪ gci{q_1, ..., q_{k-1}};
    end for;
    for each p_1 ∈ P_1 , ..., p_k ∈ P_k do
        P := {p_1, ..., p_k};
        if S ⊆ L(P) then          /* check whether reduced */
            return P;
    end for;
end for;
```

Hence, we show the main result of this section. Note that any finite set of ground terms has its k-*mmg* for any $k \geq 1$ (At worst, it is the *lgg* of S).

Theorem 6. (Arimura *et al.* 1991) *Let k be a positive integer and Σ be an alphabet with $\sharp\Sigma > k$. Then, for any finite set S of ground terms, a k-mmg of S can be computed in polynomial time in $||S||$.*

The key to an efficient k-*mmg* algorithm is to efficiently find a reduced k-multiple term, which dominates the time complexity of total computation. In the rest of the section, we present a polynomial time algorithm $REDUCED$ (in Algorithm 2) to find a reduced ck-mutiple term and show Theorem 4. In particular, the algorithm $REDUCED$ searches k-mutiple terms that define a maximal language within reduced k-terms with respect to S instead of all reduced k-multiple terms.

A *k-pivot* of a k-mutiple term P is a set V of just k ground terms in $L(P)$ such that there is some 1-to-1 correspondence f from V to P such that $v \in L(p_v)$ but for any $u \in V \setminus v$, $u \notin L(p_v)$, where p_v denotes the member $f(v)$ of $P = \{\, p_v \,|\, v \in V \}$. By definition, If P is a k-mutiple generalization of S with $\sharp P = k$, P is reduced with respect to S iff P has a k-pivot drawn from S. Let $\mathcal{R}(V)$ be the class of k-mutiple terms that has a k-pivot V, and max$\mathcal{R}(V)$ be the subclass consisting of maximal members of $\mathcal{R}(V)$ with respect to set inclusion on their languages. Then, we have the following lemma.

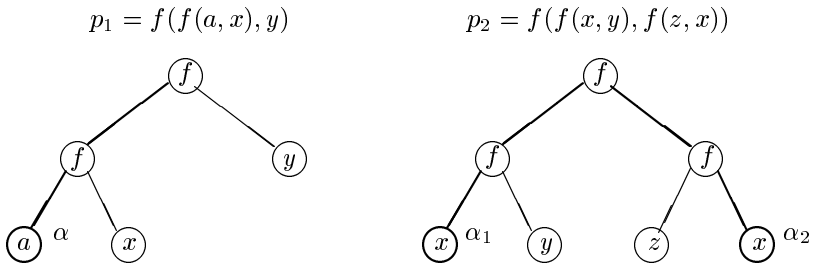

FIG. 1.1. Two possible shapes p_1 and p_2 of max trees consistent with positive term $f(f(a,a),f(b,a))$ and negative term $f(f(b,a),f(a,a))$. See Lemma 9.

Lemma 7. *Assume that $\sharp P = k$ and P is a k-mutiple term. If P defines a maximal language within reduced k-terms with respect to S, then P is a member of* $\max\mathcal{R}(V)$ *for some k-pivot V contained in S.*

Next, we show that how to compute members in $\max\mathcal{R}(V)$. We introduce the notion of consistent max trees. Let u_+, u_- be ground terms. Then, a term p is *consistent with positive term u_+ and a negative term u_-* if $u_+ \in L(p)$ but $u_- \notin L(p)$. We say p is a *max tree* consistent with u_+ and u_- if

- p is consistent with u_+ and u_-, and
- for any consistent term q with u_+ and u_-, $q \not<' p$.

To characterize $\max\mathcal{R}(V)$, we use the set $MAXT(u_+, u_-)$, the set of max trees consistent with u_+ and u_-.

Lemma 8. *Assume that $P = \{\, p_v \mid v \in V \,\}$ is k-multiple term in* $\max\mathcal{R}(V)$. *Then, each p_v is a maximally general term such that $v \in L(p_v)$ but for any $u \in V \setminus v$, $u \notin L(p_v)$. Moreover, each p_v is represented as the greatest common instance of some terms $\{\, q_u \mid u \in V \setminus v \}$ such that for every $u \in V \setminus v$, each q_u is a max tree consistent with positive v and negative u.*

By Lemma 8 above, we can see that the algorithm $REDUCED$ shown in Algorithm 2 finds a k-mutiple generalization of S that is reduced with respect to S.

Finally, we show that the set $MAXT(u_+, u_-)$ has at most polynomially many members, and it can be computed in polynomial time. A first-order term is identified with an ordered tree $tree(p)$ labelled by symbols in $\Sigma \cup X$ in standard way. A node α *touches* a path $\beta_1, \ldots, \beta_n$ if for some $1 \leq i \leq n$ either $\alpha = \beta_i$ or α is a child of β_i. The following lemma tells us the possible shapes of max trees (See FIG 1.1).

Algorithm 3: The algorithm $MAXT(t_+, t_-)$

Where $w(\alpha)$ is the label of the node α of w, and w/α is the subtree of w whose root is α. See Lemma 9 for 1-$branch(t_+, \alpha)$ and 2-$branch(t_+, \alpha_1, \alpha_2)$.

Input: Ground terms t_+, t_-.
Output: The set of max trees consistent with t_+ and t_-.
Procedure:
$T := \emptyset$;
for each node α in t_+ **do**
  if $t_+(\alpha) \in \Sigma$ and $t_+(\alpha) \neq t_-(\alpha)$ **then**
    $T := T \cup \{$ 1-$branch(t_+, \alpha)$ $\}$;
for each pair α_1, α_2 of nodes in t_+ **do**
  if $t_+/\alpha_1 = t_+/\alpha_2$ but $t_-/\alpha_1 = t_-/\alpha_2$ **then**
    $T := T \cup \{$ 2-$branch(t_+, \alpha_1, \alpha_2)$ $\}$;
for each pair $p, q \in T$ **do**
  if $L(p) \subset L(q)$ **then** $T := T - \{p\}$;
return T;

Lemma 9. *If p is a max tree with t_+ and t_-, then p is a generalization of t_+ such that tree(p) satisfies either 1 or 2:*

1. *there is a node α labelled by a function symbol such that every node touches the path from the root to α, and all leaves other than α are labelled by mutually distinct variables.*
2. *there are nodes α_1, α_2 labelled by the same variable, say x, such that every node touches a path from the root to α_1 or α_2, and all leaves other than α_1, α_2 are labelled by mutually distinct variables that are different from x.*

In Lemma 9, the choice of α and the choice of α_1, α_2 in t_+ determine the unique generalizations p of t_+ satisfying 1 and p satisfying 2, respectively. We refer to these p's as 1-$branch(t_+, \alpha)$ and 2-$branch(t_+, \alpha_1, \alpha_2)$, respectively. By Lemma 9, we can compute the set $MAXT(t_+, t_-)$ by the algorithm $MAXT$ shown in Algorithm 3 in polynomial time with respect to the size $|t_+|$.

By the observation above, the algorithm $REDUCED$ in Algorithm 2 runs in time $O(m^{2k^2+1}n^{k+1})$ with respect to m and n, where m is the maximum size of terms in S and n is the number of terms in S because the number of different choices of k-pivot V is bounded by n^k, $\sharp MAXT(t_+, t_-)$ is bounded by $2m^2$, and the greatest common instance of terms can be computed by unification algorithm in linear time in m. Hence, we can compute a k-mutiple generalization reduced with respect to S in polynomial time in $||S||$ by the algorithm $REDUCED$. Hence we have Theorem 4.

4 Applications in ILP

In this section, we describe several applications of the *mmg* algorithm in inductive logic programming (ILP, for short). In some inductive inference algorithms for logic programs such as GEMINI (Ishizaka 1988) or CIGOL (Muggleton and Buntin 1988), the least general generalization plays a very important role to infer heads of clauses. However, in general, a program consists of several clauses. In order to infer several heads using *lgg*, the inference algorithm has to divide a given set of positive examples (a finite subset of the least Herbrand model of a target program) into several appropriate subsets at first, then it can get candidates for heads of clauses by computing the *lgg* of each subset. This process, that is, dividing a set of positive examples appropriately then generalizing each obtained subset of examples, exactly corresponds to the *mmg* calculation. Hence, we believe that *mmg* is more useful than *lgg* in ILP.

The results we introduce here were obtained from our previous works (Arimura *et al.* 1991; Arimura *et al.* 1992b; Ishizaka *et al.* 1992) on inductive inferability of several subclasses of logic programs from positive facts using the *mmg* algorithm. Shinohara (1990) showed that a class of *linear Prologs* with at most k clauses is inferable from only positive facts. However, his result concerns with just inferability but not *efficiency*. We introduce three subclasses of linear Prologs, unit clause programs, context-free transformations with a flat base and primitive Prologs. Each class is *efficiently* inferable from only positive facts.

In this section, the reader is assumed to be familiar with rudiments of logic programs (Lloyd 1984). Furthermore, we assume that a first order language $\mathcal{L}$, that has finitely many predicate and function symbols (we regard a constant symbol as a 0-ary function symbol), is given. An atom, a term, a clause, a logic program (program, for short) and related notions are defined over $\mathcal{L}$. We denote the set of predicate symbols and function symbols of $\mathcal{L}$ by Π and Σ respectively. For a program P, $M(P)$ denotes the *least Herbrand model* of P.

The following definitions concerned with the notion of identification in the limit are based on (Gold 1967). An *inference algorithm* $\mathcal{A}$ is an algorithm that iterates a process "input request $\rightarrow$ computation $\rightarrow$ output". Let $g_1, g_2, \ldots$ be a sequence of outputs of $\mathcal{A}$ for an input sequence $e_1, e_2, \ldots$. We say that $\mathcal{A}$ *converges* to g for the input sequence $e_1, e_2, \ldots$ iff there exists $n \geq 1$ such that $g_i = g$ for any $i \geq n$.

An *enumeration* of a model M is a sequence $e_1, e_2, \ldots$ of elements in M such that every atom in M occurs as e_i for some $i \geq 1$. We say that $\mathcal{A}$ *identifies* a model M *in the limit from positive facts* iff $\mathcal{A}$ converges to a program P such that $M(P) = M$ for any enumeration of M. We say that $\mathcal{A}$ *identifies* a class of programs $\mathcal{P}$ *in the limit from positive facts* iff, for any $P \in \mathcal{P}$, $\mathcal{A}$ identifies $M(P)$ in the limit from positive facts.

Let $P_1, P_2, \ldots$ be a sequence of outputs of $\mathcal{A}$ for an enumeration $e_1, e_2, \ldots$ of a model M and S_i be the set $\{e_1, \ldots, e_i\}$. An inference algorithm $\mathcal{A}$ is *consistent* iff $S_i \subseteq M(P_i)$ for any i. An inference algorithm $\mathcal{A}$ is *conservative* iff $P_i = P_{i-1}$ for any i such that $e_i \in M(P_{i-1})$. An inference algorithm $\mathcal{A}$ is a *polynomial update time inference algorithm* iff there exists some polynomial f such that, for any stage i, after $\mathcal{A}$ feeds the input e_i it produces the output P_i in $f(||S_i||)$ steps. Any exponential update time inference algorithm can be converted into a cunning polynomial update time one, even if either consistency or conservativeness lacks. Hence, both conditions are necessary for validity of polynomial update time inference.

A class of programs $\mathcal{P}$ is said to be (*consistently, conservatively, polynomial update time*) *inferable* from positive facts iff there exists an (consistent, conservative, polynomial update time) inference algorithm that identifies $\mathcal{P}$ in the limit from positive facts. .

4.1 Unit clause programs

First we consider a class of very simple logic programs that consist of only unit clauses. We denote the class by $\mathcal{UCP}$ and a class of logic programs that consist of at most k unit clauses by k-$\mathcal{UCP}$. For any $P \in \mathcal{UCP}$, since each clause $C \in P$ is unit, it holds that $M(P) = \bigcup_{C \in P} L(C)$ where $L(C)$ is a set of all ground instances of C. Thus, if $\sharp(\Pi \cup \Sigma) > k$, then we can directly apply k-mmg algorithm to infer $\mathcal{UCP}$.

Algorithm 4: Inference algorithm for k-$\mathcal{UCP}$

Input: An enumeration of a model $M(P)$ where $P \in k$-$\mathcal{UCP}$.
Output: An infinite sequence of programs in k-$\mathcal{UCP}$.
Procedure:
  $H := \emptyset$; $S := \emptyset$;
  repeat
    read the next fact e; $S := S \cup \{e\}$;
    if $e \notin M(H)$ **then** $H := MMG(k, S)$;
    output H;
  forever

Angluin (1980) showed that if a target class has the property called *finite thickness*, that is, it contains only finitely many concepts including a given examples, then an inference algorithm that, at any stage, outputs a *minimal hypothesis consistent with given positive examples* can identify the class. A minimal hypothesis is a hypothesis that defines a minimal concept such as a minimal language or a minimal least Herbrand model, in our context, it corresponds to $MMG(k, S)$. Unfortunately, the class k-$\mathcal{UCP}$ does not have finite thickness when $k > 1$. However, Angluin's result can be extended to the class with the property called *finite elasticity* (Wright

1989a, 1989b). Shinohara (1990) showed that the class of linear Prologs with at most k clauses is inferable from positive facts by showing the class has finite elasticity. Since k-$\mathcal{UCP}$ is a subclass of linear Prologs, it also has finite elasticity. Thus Algorithm 3 identifies k-$\mathcal{UCP}$. Consistency and conservativeness of Algorithm 4 are trivial. From Theorem 6, it is also clear that Algorithm 4 produces each hypothesis in polynomial steps in $||S||$. Thus we have the following theorem.

Theorem 10. (Arimura *et al.* 1991) *Suppose that* $\sharp(\Pi \cup \Sigma) > k$. *Then the class* k-$\mathcal{UCP}$ *is consistently and conservatively polynomial update time inferable.*

4.2 Context-free transformations with a flat base

Next we consider a slightly complex subclass $\mathcal{CFT}^{uniq}_{FB}$ of context-free transformations ($\mathcal{CFT}$, for short). The class $\mathcal{CFT}$ was originally introduced by Shapiro in his study on MIS (Shapiro 1981) and includes a lot of non-trivial programs such as *append*, *plus* and so on. We introduce a restricted subclass $\mathcal{CFT}^{uniq}_{FB}$ of $\mathcal{CFT}$.

A *context-free transformation with a flat base* (CFT_{FB}) is a program that consists of two clauses C_0 and C_1:

$$\begin{aligned} C_0 &= p(s_1, \ldots, s_m). \\ C_1 &= p(t1, \ldots, t_m) \leftarrow p(x_1, \ldots, x_m). \end{aligned}$$

that satisfy the following conditions (a)–(c).

(a) Every argument s_i $(1 \leq i \leq m)$ of the head of C_0 is either a function symbol of arity 0 or a variable symbol.

(b) All arguments $x_1, \ldots, x_m$ of the body of C_1 are mutually distinct variables.

(c) For every $1 \leq i \leq m$, every argument x_i of the body of C_1 occurs exactly once in the term t_i of the head. Moreover, x_i does not occur in any argument t_j $(i \neq j)$ of the head.

A program P is a CFT^{uniq}_{FB} iff there exists at most one 2-mmg of $M(P)$. We denote the class of all CFT^{uniq}_{FB} prorams by $\mathcal{CFT}^{uniq}_{FB}$.

It seems the class $\mathcal{CFT}^{uniq}_{FB}$ is too restrictive. Furthermore, the class $\mathcal{CFT}^{uniq}_{FB}$ is defined according to the least Herbrand model of each element. Since the least Herbrand model of a program is an infinite set in general, the definition seems to be problematic. Fortunately, however, the class is decidable in polynomial time. That is, given a program P, we can decide whether P is in $\mathcal{CFT}^{uniq}_{FB}$ in polynomial time of $size(P)$, where $size(P)$ is the size of P as an expression. In fact, we can show that several non-trivial programs in $\mathcal{CFT}$ are still in $\mathcal{CFT}^{uniq}_{FB}$. For example, the following CFT's

are in $\mathcal{CFT}^{uniq}_{FB}$.

$$append([], X, X).$$
$$append([A|X], Y, [A|Z]) \leftarrow append(X, Y, Z).$$

$$suffix(X, X).$$
$$suffix(X, [A|Y]) \leftarrow suffix(X, Y).$$

$$plus(X, 0, X).$$
$$plus(X, s(Y), s(Z)) \leftarrow plus(X, Y, Z).$$

$$lesseq(0, X).$$
$$lesseq(s(X), s(Y)) \leftarrow lesseq(X, Y).$$

Algorithm 5 is an inference algorithm for $\mathcal{CFT}^{uniq}_{FB} \cup 2\text{-}\mathcal{UCP}$. Algorithm 5 does not change the current hypothesis H, if it is consistent with a newly given positive fact e. Suppose that

$$\begin{aligned} S \quad = \{ \quad & app([], [], []), app([b], [a], [b, a]), app([a], [], [a]), app([], [a], [a]), \\ & app([a, b], [c, d], [a, b, c, d])\} \end{aligned}$$

is the set of positive facts given so far and the current hypothesis cannot imply the last fact. First, the algorithm finds a pair of atoms

$$\{app([], X, X), app([A|X], Y, [A|Z])\}$$

by $MMG(2, S)$. Then it tries to find a hypothesis consistent with S by enumerating every CFT^{uniq}_{FB} with $\{app([], X, X), app([A|X], Y, [A|Z])\}$ as its heads. Actually, the search is done by enumerating every possible instance of CFT^{uniq}_{FB} with pair of atoms obtained by the 2-mmg algorithm as its heads, because the candidate heads are possibly less general than the heads of a target program. If such a program P^* containing a clause with non-empty body is found, the algorithm outputs it. Otherwise the algorithm simply outputs $MMG(2, S)$ as an approximation of the target model. Since P^* is an instance of a CFT^{uniq}_{FB}, Algorithm 5 may need to transform P^* into a CFT^{uniq}_{FB} that has the same least Herbrand model with P^*. The transformation φ performs such model preserving generalization.

Since the class $\mathcal{CFT}^{uniq}_{FB}$ is also a subclass of linear Prologs, it has finite elasticity. Hence, if Algorithm 5 outputs a minimal hypothesis consistent with S at any stage, it is ensured that the algorithm identifies $\mathcal{CFT}^{uniq}_{FB} \cup 2\text{-}\mathcal{UCP}$. As described in the next subsection, in general, there exist several 2-mmg's for a entire model $M(P)$ of a program P that contains a clause with non-empty body. If there exist several 2-mmg's of $M(P)$, then it becomes difficult to decide which 2-mmg is appropriate for the heads of a target program. The difficulty directly concerns with the difficulty of

Algorithm 5: Inference algorithm for $\mathcal{CFT}^{uniq}_{FB} \cup 2\text{-}\mathcal{UCP}$

Input: An enumeration of a model $M(P)$ where $P \in \mathcal{CFT}^{uniq}_{FB} \cup 2\text{-}\mathcal{UCP}$.
Output: An infinite sequence of programs in $\mathcal{CFT}^{uniq}_{FB} \cup 2\text{-}\mathcal{UCP}$.
Procedure:

```
H := ∅; S := ∅;
repeat
    read the next fact e; S := S ∪ {e};
    if e ∉ M(H) then
        {h0, h1} := MMG(2, S);
        find a hypothesis P* consistent with S
            whose heads are {h0, h1};
        if found then H := φ(P*);
        else H := {h0, h1};
    output H;
forever
```

finding a consistent minimal hypothesis as described in the next subsection. However, from the uniqueness of 2-*mmg* for the model of CFT^{uniq}_{FB}, it is possible to ensure that a consistent minimal hypothesis can be found by a very simple search as mentioned above. Consistency and conservativeness of Algorithm 5 is trivial. From the syntactical restriction on CFT_{FB}, the search of a consistent hypothesis P^* and the transformation P^* into a CFT^{uniq}_{FB} $\varphi(P^*)$ can be finished in polynomial time in $||S||$. Hence we can obtain the following theorem.

Theorem 11. (Arimura *et al.* 1992b) *Suppose that* $\sharp\Sigma > 2$. *Then the class* $\mathcal{CFT}^{uniq}_{FB} \cup 2\text{-}\mathcal{UCP}$ *is consistently and conservatively polynomial update time inferable.*

4.3 Primitive Prologs

Finally, we consider a class of programs called primitive Prologs. A *primitive Prolog* P is a program that satisfies following conditions (a)–(d):

(a) Only one unary predicate symbol appears in P.
(b) P consists of at most two clauses.
(c) If P consists of two clauses, then both heads of clauses have no common instance.
(d) Atoms appearing in the body of a clause are most general atoms as $p(x)$.
(e) Variables appearing in the body of a clause are mutually distinct and also appear in the head of the clause.

In a word, a primitive Prolog is a program of the form:

$$p(t[x_1, \ldots, x_m]) \leftarrow p(x_1), \ldots, p(x_m).$$
$$p(s).$$

where $x_1, \ldots, x_m$ are mutually distinct variables, $t[x_1, \ldots, x_m]$ is any term containing the variables $x_1, \ldots, x_m$, s is any term and $L(t[x_1, \ldots, x_m]) \cap L(s) = \emptyset$. We denote a class of primitive Prologs by $\mathcal{PP}$.

Although the class $\mathcal{PP}$ is so restricted, there exist a primitive Prolog P that has several 2-mmg's of its model $M(P)$. For example, consider the following primitive Prolog P.

$$p([a, b, a]).$$
$$p([b|X]) \leftarrow p(X).$$

For the least Herbrand model

$$M(P) = \{p([a, b, a]), p([b, a, b, a]), p([b, b, a, b, a]), p([b, b, b, a, b, a]), \ldots\},$$

there exist two kinds of 2-mmg of $M(P)$:

$$\{p([a, b, a]), p([b, X, Y, Z|W])\} \quad \text{and} \quad \{p([b, a, b, a]), p([X, b, Y|Z])\}.$$

Actually the former is an instance of the heads of the program P. Hence, if an inference algorithm selects the former pair as the heads of a hypothesis, it will be able to identify the target program. However, if the inference algorithm selects the latter pair, then it may produce an overgeneralized hypothesis and fail to identify the target program consistently and conservatively.

Let P_1 be an instance

$$p([a, b, a]).$$
$$p([b, X, Y, Z|W]) \leftarrow p([X, Y, Z|W]).$$

of P with the former 2-mmg as its heads and P_2 be be a program consists of the atoms in the latter 2-mmg. We know that P_1 is a correct hypothesis but P_2 is an overgeneralized one. However the algorithm is given only positive facts and both of P_1 and P_2 are consistent with every fact. Hence, in order to achieve conservative inference, the algorithm has to decide which program has a smaller model. That is, to construct a consistent and conservative polynomial update time inference algorithm for the class $\mathcal{PP}$, a model containment problem for primitive Prologs (P_1 can be easily transformed into the original primitive Prolog) should be solved efficiently. Unfortunately the problem is still open.

Although the problem of consistent and conservative polynomial update time inferability of $\mathcal{PP}$ is also still open, we showed an interesting polynomial update time algorithm that identifies $\mathcal{PP}$ consistently but not conservatively (Ishizaka *et al.* 1992). The algorithm concentrates its at-

Table 1.1 A university database (Cai *et al.* 1991)

Name	Category	Major	Birth_Place	GPA
Anderson	M.A.	Physics	Vancouver	3.5
Fraser	M.S.	Physics	Ottawa	3.9
Gupta	Ph.D	Math	Bombay	3.3
Liu	Ph.D.	Biology	Shanghai	3.4
Monk	Ph.D.	Computing	Victoria	3.8
Wang	M.S.	Statistics	Nanjing	3.2

GPA = grade point average

tention on a minimal size fact given so far to find a unit clause in a target program. This simple idea works well. Whenever a target primitive Prolog consists of a unit clause and a recursive clause with non-empty body as the previous example,

$$p([a,b,a]).$$
$$p([b|X]) \leftarrow p(X).$$

a fact $p([a,b,a])$ of minimal size in the model $M(P)$ should be an instance of the unit clause. Using this property, the algorithm can identify the correct heads in several 2-mmg's in the limit working inconservatively. Lange and Wiehagen (1991) showed an interesting inference algorithm that inconsistently runs in polynomial update time and infers pattern languages from positive examples. Our idea is similar to theirs.

5 Application in knowledge discovery in databases

In this section, we describe an application of the k-mmg algorithm to the problem of discovering knowledge in databases.

5.1 Attribute-oriented induction

There are different two approaches in discovery of knowledge in databases (Piatetsky-Shapiro and Frawley 1991). One is, given positive examples and negative examples, to find *classification rules*, that is, rules separating positive examples from negative examples. Another is, given positive examples, to find *characteristic rules*, that is, rules characterizing the concept represented by the positive examples.

The latter one is closely related to inductive inference from positive examples and can be thought as computing a generalization of databases. Since a database usually contains only positive examples of a relation in the real world, we believe discovery algorithms that learn concepts only from positive examples are useful. Thus, we deal with discovery of characteristic rules from positive examples in this section.

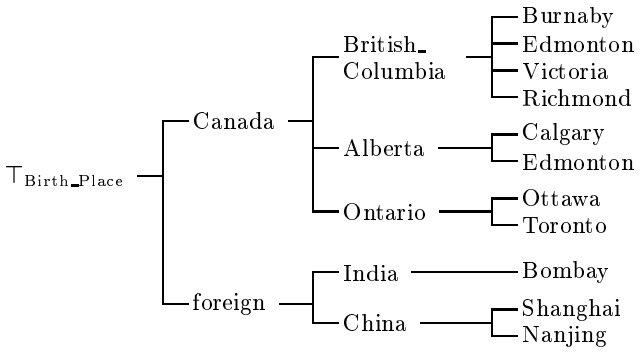

FIG. 1.2. A concept hierarchy $\mathcal{D}$ for attribute Birth_Place (Cai *et al.* 1991)

Cai *et al.* (1991) proposed a heuristics algorithm to discover characteristic rules. Their algorithm LCHR generalizes a database by using a given conceptual hierarchy. In Table 1.1, we show an example of databases extracted from (Cai *et al.* 1991), which consists of tuples concerning to information of graduate students. In Figure 1.2, we show the corresponding conceptual hierarchy concerning to the attribute Birth_Place by a tree-like structure. The tree-like structure shows that a concept placed near the root is more general than one placed near the leaves.

Given a small positive integer k, called a *threshold*, a database and the corresponding conceptual hierarchy, the algorithm LCHR iterates the following process. It nondeterministically selects a tuple from the database and replaces some of attribute values by more general values with respect to the conceptual hierarchy. In this process, attribute values to be substituted should be carefully selected from tuples to avoid overgeneralization. Successive applications of this process eventually decrease the number of different tuples in the database. If the number of tuples becomes less than or equal to the threshold k, the algorithm terminates and outputs the resulting generalized database as a characteristic rule.

Table 1.2 is a generalized database obtained from the database in Table 1.1 by LCHR, where two columns for Name and Category that have no useful information are removed (Cai *et al.* 1991). The generalized database computed by LCHR characterizes the given database by a kind of clustering. The paper (Cai *et al.* 1991) reported that the algorithm LCHR runs efficiently in the size of the input database and finds rules characterizing the database well.

Table 1.2 A generalized database obtained by the algorithm LCHR

Major	Birth_Place	GPA
science	Canada	excellent
science	foreign	good

5.2 Applying the *k-mmg* to attribute-oriented induction

As seen before, the algorithm LCHR finds a generalized database as specific as possible by careful applications of substituting concept values. In fact, we can see that what LCHR tries to find is a minimal multiple generalization in the sense of databases with conceptual hierarchy. In the remaining part, the method of application of *k-mmg* to attribute-oriented induction by the previous example.

First, we formalize the notion of generalized databases with conceptual hierarchy. A *concept hierarchy* or *domain* is a pair $(\mathcal{D}, \leq)$ of a set $\mathcal{D}$ of objects and a partial order $\leq$ on D. We assume that $(\mathcal{D}, \leq)$ is a tree with the greatest element $\top$. The relation $\leq$ is called *generalization relation*. Hereafter, we write $\mathcal{D}$ for $(\mathcal{D}, \leq)$ if it is clear from context. A ground object is a minimal element of $\mathcal{D}$. Assume countably many attributes $a_1, a_2, \ldots$ and the corresponding domains $\mathcal{D}_1, \mathcal{D}_2, \ldots$. Then, a *scheme* is a m-tuple $\mathcal{R} = (a_1, \ldots, a_m)$ of attributes. For the corresponding tuple $(\mathcal{D}_1, \ldots, \mathcal{D}_m)$ of domains, a *generalized tuple* on $\mathcal{R}$ is a m-tuple $p = (o_1, \ldots, o_m)$ such that $o_1 \in \mathcal{D}_1, \ldots, o_m \in \mathcal{D}_m$. The *value* of a_i in p is o_i $(1 \leq i \leq m)$. A *generalized database* on $\mathcal{R}$ is a finite set R of generalized tuples on $\mathcal{R}$, equivalently, a subset of $\mathcal{D}_1 \times \cdots \times \mathcal{D}_m$. If R consists only of ground objects, then we say R is a *ground* database. Usual relational databases are ground database. We denote by $\mathcal{DB}_{\mathcal{R}}$ the class of generalized databases on R.

Then, we introduce a generalization relation between generalized databases by extending $\leq$ on concept hierarchies. A tuple p is *more general than* q if for every attribute a_i $(1 \leq i \leq m)$, the value of a_i in p is more general than that of a_i in q. A generalized database P is *more general than* Q, denoted by $Q \sqsubseteq P$, if for every q in Q, there is some p in P that is more general than q. Careful reader may find that the relation $(\mathcal{DB}_{\mathcal{R}}, \sqsubseteq)$ is the generalization relation used in the framework of Cai *et al.* (1991), and what their heuristics algorithm searches for is k-minimal multiple generalizations, where k is the threshold.

Let E be a ground database that the learning algorithm is given, and $k > 0$ be the predetermined threshold value. Our method consists of the following three stages.

1. Transform ground database Q on the scheme R into a set S of ground first-order terms that represent tuples in Q.

2. Apply *k-mmg* algorithm to S. It finds a k-minimal multiple generalization P of S in polynomial time in the size of $||S||$.
3. Transform P to the corresponding generalized database R on $\mathcal{R}$. R is minimally generalized database of Q.

To apply the *k-mmg* algorithm directly to the problem, we represent a concept hierarchy $(\mathcal{D}, \leq)$ by a specific class $\mathcal{TP}_{\mathcal{D}}$ of first-order terms with generalized relation $\leq'$. For each objects o in $\mathcal{D}$, the alphabet $\Sigma_{\mathcal{D}}$ of $\mathcal{TP}_{\mathcal{D}}$ contains a function symbol o, where o is 0-ary if it is a ground element in $\mathcal{D}$, and 1-ary otherwise.

Recall that we identified a first-order term p on $\Sigma_{\mathcal{D}}$ with a ordered tree $tree(p)$ labelled by symbols in $\Sigma_{\mathcal{D}}$ in Section 3. The mapping $tree$ maps a term on $\Sigma_{\mathcal{D}}$ to a tree consisting of a single path from the root to the leaf. We use this correspondence to define a mapping τ from $\mathcal{D}$ to $\mathcal{TP}_{\mathcal{D}}$. Hereafter, we consider $\mathcal{D}$ as an ordered tree labelled by objects in $\mathcal{D}$. Let o be an object in $\mathcal{D}$ and π be a branch from the root $\top$ of $\mathcal{D}$ to o. If o is not a ground object, we add a node labelled by a variable, say x, to π as the child of o.

Then, the single path π labelled by function symbols (and possibly a variable) determines the unique first-order term p such that $tree(p)$ is the ordered tree π. Intuitively, a path $\pi = o_1, o_2, \ldots, o_n$ is mapped to a term $o_1(o_2(\ldots(o_n)\ldots))$. Consider the concept hierarchy shown in FIG 1.2. Objects China and Shanghai have the following correspondence with first-order terms.

$$\begin{array}{l} o = \text{China} \\ o' = \text{Shanghai} \end{array} \iff \begin{array}{l} \tau(o) = \top(\text{foreign}(\text{China}(x))) \\ \tau(o') = \top(\text{foreign}(\text{China}(\mathit{Shanghai}))) \end{array}$$

For a generalized tuple $p = (o_1, \ldots, o_m)$, we define the corresponding first-order term by $\tau(p) = db(\tau(o_1), \ldots, \tau(o_m))$ using a special m-ary function symbol db, and the multiple first-order term by $\tau(R) = \{\, \tau(p) \mid p \in R\}$. Let $\mathcal{TP}_D = \{\, \tau(o) \mid o \in \mathcal{D}\}$ and $\mathcal{TP}^*_{\mathcal{R}} = \{\, \tau(R) \mid R \in \mathcal{DB}_{\mathcal{R}}\}$.

Then, $(\mathcal{D}, \leq)$ and $(\mathcal{TP}_{\mathcal{D}}, \leq')$ are order isomorphic. Moreover, $(\mathcal{TP}_{\mathcal{D}}, \leq'$) is closed under generalizations. Thus, if we take a relation $\sqsubseteq'$ defined by $Q \sqsubseteq P$ if for every q in Q, there is some p in P such that $q \leq' p$, $\mathcal{DB}_{\mathcal{R}}$ and $\mathcal{TP}^*_{\mathcal{R}}$ become isomorphic. By careful observation of the result in Section 3, a reader can see that *k-mmg* algorithm computes a minimal k-multiple term P containing a given finite set S of ground terms with respect to $\sqsubseteq'$.

Hence, we can efficiently finds a minimally general database R in $\mathcal{DB}_{\mathcal{R}}$ containing a ground database E under threshold k as follows. First, we transform E to a set of first order terms $S = \tau(E)$, then compute a *k-mmg* P of S, and again transform P to a generalized database $R = \tau^{-1}(P)$. We show an example of computation in Table 1.3

Table 1.3

Computing a minimally generalized database from a given ground database E on the scheme $\mathcal{R} = (\text{Major}, \text{Birth\_Place}, \text{GPA})$ by *k-mmg* algorithm, where the threshold $k = 2$ and E is given in Table 1.1.

Step 1. First, transform a ground database E to the set S of ground first-order terms by mapping τ.

A set S of ground first-order terms:

Major	Birth_Place	GPA
$\top$(science(Physics))	$\top$(Canada(B_C(Vancouver)))	$\top$(excellent(3.5))
$\top$(science(Physics))	$\top$(Canada(Ontario(Ottawa)))	$\top$(excellent(3.9))
$\top$(science(Math))	$\top$(foreign(India(Bombay)))	$\top$(good(3.3))
$\top$(science(Biology))	$\top$(foreign(China(Shanghai)))	$\top$(good(3.4))
$\top$(science(Computing))	$\top$(Canada(B_C(Victoria)))	$\top$(excellent(3.8))
$\top$(science(Statistics))	$\top$(foreign(China(Nanjing)))	$\top$(good(3.2))

Step 2.a. Apply *k-mmg* algorithm to S: It searches a k-mutiple terms reduced with respect to S.

A k-multiple term Q reduced with respect to S:

Major	Birth_Place	GPA
$\top(x_1)$	$\top$(Canada(x_2))	$\top(x_3)$
$\top$(science(y_1))	$\top(y_2)$	$\top(y_3)$

Step 2.b. Apply *k-mmg* algorithm to S: It computes k-minimal mutiple generalization P by tightening Q with respect to S.

A 2-mimimal multiple generalization P of S:

Major	Birth_Place	GPA
$\top$(science(x_1))	$\top$(Canada(x_2))	$\top$(excellent(x_3))
$\top$(science(y_1))	$\top$(foreign(y_2))	$\top$(good(y_3))

Step 3. Transform P to the corresponding generalized database R on $\mathcal{R}$ by mapping τ^{-1}. Then, R is minimally general generalized database obtained from E with respect to the order $(\mathcal{DB}_{\mathcal{R}}, \sqsubseteq)$

A transformed generalized database R:

Major	Birth_Place	GPA
science	Canada	excellent
science	foreign	good

We can easily see that the generalized database computed by our method (FIG 1.3) corresponds to what the LCHR algorithm finds (FIG 1.2). In the work in (Cai *et al.* 1991), the correctness and the time complexity of the algorithm LCHR are not clear. On the other hand, we can estimate those of our method from the correctness and the time complexity of the *k-mmg* algorithm.

Unfortunately, the *k-mmg* algorithm may not find all of the answers. This means that our method may loss some of characteristic rules that Cai et al's method can find. Thus, the study of a *k-mmg* algorithm that can find any *k-mmg*'s nondeterministically seems to be interesting from the view point of knowledge discovery in databases.

Bibliography

1. Angluin, D. (1980) Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135.
2. Arimura, H., Shinohara, T. and Otsuki, S. (1991) A polynomial time algorithm for finding finite unions of tree pattern languages. In *Proc. of the 2nd International Workshop on Nonmonotonic and Inductive Logic.* LNAI 659, pp. 118–131. Springer, 1993.
3. Arimura, H., Shinohara, T. and Otsuki, S. (1992a) Polynomial time inference of unions of two tree pattern languages. *IEICE trans. Inf. and Syst.*, E75-D(7):426–434.
4. Arimura, H., Ishizaka, H. and Shinohara, T. (1992b) Polynomial time inference of a subclass of context-free transformations. In *Proceedings of 5th Annual ACM Workshop on Computational Learning Theory*, pp.136–143.
5. Cai, Y., Cercone, N. and Han, J. (1991) Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pp. 213–228. AAAI Press/The MIT Press.
6. Gold, E. M. (1967) Language Identification in the Limit. *Information and Control*, 10:447–474.
7. Ishizaka, H. (1988) Model inference incorporating generalization. *Journal of Information Processing*, 11(3):206–211.
8. Ishizaka, H., Arimura, H. and Sinohara, T. (1992) Efficient inductive inference of primitive prologs from positive data. In S. Doshita, K. Furukawa and T. Nishida, editors, *Proc. ALT '92*, pp. 135–146.
9. Lange, S. and Wiehagen, R. (1991) Polynomial-time inference of arbitrary pattern languages, *New Generation Computing*, 8(4):361–370.
10. Lassez, J-L. and Marriott, K. (1986) Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3:301–317.

11. Lloyd, J. W. (1984) *Foundations of Logic Programming.* Springer-Verlag.

12. Muggleton, S. (1990) Inductive logic programming. In S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori, editors, *Proc. ALT '90*, pp. 42–62. Ohmsha.

13. Muggleton, S. and Buntine, W. (1988) Machine invention of first-order predicates by inverting resolution. In *Proc. 5th International Conference on Machine Learning*, pp. 339–352.

14. Piatetsky-Shapiro, G. and Frawley, W. J. editors. (1991) *Knowledge Discovery in Databases.* AAAI Press/The MIT Press.

15. Pitt, L. and Valiant, L. G. (1988) Computational limitations on learning from examples. *JACM*, 35(4):965–984.

16. Plotkin, G. D. (1970) A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pp. 153–163. Edinburgh University Press.

17. Reynolds, J. C. (1970) Transformational systems and the algebraic structure of atomic formulas. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pp. 135–151. Edinburgh University Press.

18. Shapiro, E. Y. (1981) Inductive inference of theories from facts. Technical Report 192, Yale University Computer Science Dept..

19. Shinohara, T. (1990) Inductive inference of monotonic fomal systems from positive data. In S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori, editors, *Proc. ALT '90*, pp. 339–351. Ohmsha.

20. Wright, K. (1989a) Identification of unions of languages drawn from an identifiable class. In *Proceedings of 2nd Annual Workshop on Computational Learning Theory*, pp. 328–333. Morgan Kaufmann.

21. Wright, K. (1989b) *Inductive Inference of Pattern Languages.* PhD thesis, University of Pittsburgh.