A Generalization of the Least General Generalization

Arimura, Hiroki
Department of Artificial Intelligence Kyushu Institute of Technology

Ishizaka, Hiroki
FUJITSU LABORATORIES, IIAS

Shinohara, Takeshi
Department of Artificial Intelligence Kyushu Institute of Technology

Otsuki, Setsuko
Department of Artificial Intelligence Kyushu Institute of Technology

出版情報：RIFIS Technical Report. 63, 1992-08-20. Research Institute of Fundamental Information Science, Kyushu University
バージョン：
権利関係：

RIFIS-TR-CS-63

RIFIS Technical Report

A Generalization of the Least General Generalization

Hiroki Arumura
Hiroki Ishizaka
Takeshi Shinohara
Setsuko Otsuki

August 20, 1992

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

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

A Generalization of the Least General Generalization

Hiroki ARIMURA[†]
arim@ai.kyutech.ac.jp

Hiroki ISHIZAKA[‡]
hiro@iias.flab.fujitsu.co.jp

Takeshi SHINOHARA[†]
shino@ai.kyutech.ac.jp

Setsuko OTSUKI[†]
otsuki@ai.kyutech.ac.jp

† Department of Artificial Intelligence
Kyushu Institute of Technology
680-4, Kawazu, Iizuka 820, Japan
‡ FUJITSU LABORATORIES, IIAS
140 Miyamoto, Numazu 410-03, Japan

Abstract

In this paper, we describe 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. Given a finite set of gound terms, the k-mmg algorithm generalizes the examples by at most k terms, while Plotkin's algorithm does by a single term. We apply this algorithm to problems in inductive logic programming. We also show that this method is applicable to a problem of knowledge discovery in databases.

1 Introduction

Inductive inference is a process to guess or identify an unknown general rule from its examples. An inference machine receives finite examples and produces a kind of a generalization of them as a hypothesis. Thus, we can consider such an inference process as a generalization.

Recently, many studies are developed on knowledge discovery in databases [PSF91]. Usually, we think of a database as a collection of positive examples drawn from some rule in the real world. To discover such a rule in the database, an inductive inference from only positive examples rather than from both positive and negative examples is naturally used.

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. For example, when an unknown rule to be inferred is represented by a single term, an algorithm developed by Plotkin in 1970 [Plo70], independently by Reynolds [Rey70], efficiently finds the minimal generalization of given examples. This generalization is called the *least general generalization* (*lgg*, for short). From the reason, the least general generalization algorithm plays an important role in model inference systems [Sha81, Ish88] and inductive logic programming [Mug90].

On the other hand, when some rules in question are too complex to be represened 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 generalizaion 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 basic notations of tree pattens, their languages and generalizations. Then, we show the property, called compactness with respect to containment, of the class of languages defined by at most k tree patterns. In Section 3, we describe a *k-mmg* algorithm that finds a *k-mmg* in polynomial time. 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. We show results obtained from our previous works [ASO91a, AIS92, IAS92] on inductive inferability of the subclasses only from positive data using the *k-mmg* algorithm. In Section 5, we show the method is also applicable to the problem of discovering rules that characterize a concept in a database.

2 Preliminaries

In this section, we introduce basic notions and notations on tree patterns, their languages, generalizations. Notations and terminology that are not defined here may be found in Lloyd [Llo84].

Let Σ be a finite alphabet and X be a countable set of variables that is disjoint from Σ. A *tree pattern over* Σ is an ordered tree p whose nodes are labelled with symbols in $\Sigma \cup X$, where if a node α of p is labelled with an n-ary function symbol in Σ ($n > 0$), then α has n children, and if α is labelled with a variable or a 0-ary function symbol in Σ, then it is a leaf. Tree patterns are identified with *terms* or *atoms* in formal logic and logic programming. A *ground tree pattern*, that is, tree pattern containing no variables is called a *tree*. By $\mathcal{T}_\Sigma$ we denote a set of all trees over Σ. We use equality symbol $=$ as syntactic identity. We define a relation $\leq'$ over tree patterns as $p \leq' q$ iff $p = q\theta$ for some substitution θ. When $p \leq' q$, p is an *instance* of q, or q is a *generalization* of p. We define $p <' q$ iff $p \leq' q$ but $q \not\leq' p$, and $p \equiv' q$ iff $p \leq' q$ and $q \leq' p$. If $p \equiv q$, then we say p is *equivalent* to q. Note that $p \equiv' q$ iff $p = q\theta$ for some renaming θ of variables. We do not distinguish equivalent tree patterns from each other. The *size* $|p|$ of a tree pattern p is the number of occurrences of symbols in p. The *size* $||S||$ of a finite set $S \subseteq \mathcal{T}_\Sigma$ is the total size of the trees in S. We denote by $\sharp S$ the number of elements of S.

For a tree pattern p, a *language defined by* p is the set $L(p) = \{ w \in \mathcal{T}_\Sigma \mid w \leq' p\}$ of all ground instances of p. A set $L \subseteq \mathcal{T}_\Sigma$ is called a *tree pattern language* iff $L = L(p)$ for some tree pattern p.

Lemma 1 ([LMM88]) *$L(p) \subseteq L(q)$ iff $p \leq' q$.*

Let $k > 0$ be any fixed integer. A *union defined by a set $\{p_1, \ldots, p_k\}$ of at most k tree patterns*, denoted by $L(\{p_1, \ldots, p_k\})$, is a union $L(p_1) \cup \cdots \cup L(p_k)$. We refer to the class of unions of at most k tree pattern languages as $\mathcal{TPL}^k_\Sigma$ and to the class of sets of at most k tree patterns as $\mathcal{TP}^k_\Sigma$. We may omit the subscript Σ if it is clear from context. The class $\mathcal{TPL}^k$ is *compact with respect to containment* iff for any $L \in \mathcal{TPL}^1$ and any union $L_1 \cup \cdots \cup L_k \in \mathcal{TPL}^k$,

$$L \subseteq L_1 \cup \ldots \cup L_k \iff L \subseteq L_i \text{ for some } 1 \leq i \leq k.$$

Lassez et. al.[LM86] showed that if Σ is an infinite alphabet, then the class $\mathcal{TPL}^*$ of any finite unions of tree pattern languages over Σ is compact with respect to containment. Now, we refine their result for a finite Σ.

Theorem 2 ([ASO91b]) Let Σ be an alphabet. For any $k > 0$, if $\sharp\Sigma > k$, then the class $\mathcal{TPL}^k_\Sigma$ is compact with respect to containment.

We introduce a notion of multiple generalizations. Let $k > 0$ be an integer and S be a set of trees. A set $\bar{p} \in \mathcal{TP}^k$ is a *k-minimal multiple generalization* (*k-mmg*) of S iff $L(\bar{p})$

is a minimal language containing S within the class $\mathcal{TPL}^k$, that is, (1) $S \subseteq L(\bar{p})$ and (2) $L \subset L(\bar{p})$ implies $S \not\subseteq L$ for any $L \in \mathcal{TPL}^k$. For a set $S \subseteq \mathcal{T}$, a set $\bar{p} \in \mathcal{TP}^k$ is *reduced with respect to* S iff $S \subseteq L(\bar{p})$, and $S \not\subseteq L(\bar{q})$ for any proper subset $\bar{q}$ of $\bar{p}$. For any *k-mmg* $\bar{p}$ of S, there is a reduced set $\bar{q}$ with respect to S such that $L(\bar{p}) = L(\bar{q})$.

From Lemma 1, the notion of 1-*mmg* coincides with the least general generalization [Plo70, Rey70, LMM88]. We write 1-*mmg* of S as $lgg(S)$. For a finite set $S \subseteq \mathcal{T}$, the *least general generalization algorithm* [Plo70] computes $lgg(S)$ in polynomial time in $||S||$. The algorithm is also called the *anti-unification algorithm* in [Rey70]. Computation of least general generalization allows fast efficient parallel algorithms. Delcher et al. [DK90] describes an algorithm that runs in $O(\log^2 n)$ time using $O(N/\log^2 n)$ processors on EREW PRAM model.

3 Finding minimal multiple generalizations

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      p̄(= {p_1, ..., p_k}) := REDUCED(k, S);
4      if p̄ is found then
5          for each i = 1, ..., k do  /* tightening process */
6              let S_i := S − L(p̄ − {p_i});
7              replace p_i in p̄ by q_i ≡' lgg(S_i);
8          output p̄;
9      else output MMG(k − 1, S);
```

Let $S \subseteq \mathcal{T}$ be a finite set. Then, $lgg(S)$ is uniquely determined in polynomial time in $||S||$. Intuitively, even for a fixed $k > 0$, S may have exponentially many reduced *k-mmg*'s, since the number of all the partitions into k subsets is $k^{\sharp S}$. Pitt et al. showed that the consistency problem for DNF formulas is hard to compute [PV88]. By a reduction from the problem, we can actually show that the set of all the reduced *k-mmg*'s of S cannot be computed in polynomial time in $||S||$ unless $P = NP$ even for fixed $k \geq 2$. In contrast, as shown in this section, one of the *k-mmg*'s of S can be found in polynomial time under the assumption of the compactness with respect to containment. Throughout this section, we assume that $\sharp\Sigma > k$ to guarantee the compactness of $\mathcal{TPL}^k$.

Now, we give an algorithm $MMG(k, S)$ in Figure 1 that computes a reduced *k-mmg* $\bar{p}$ of S. The algorithm first tries to find a set of exactly k tree patterns reduced with

respect to S as a candidate of a *k-mmg*. The following Theorem 3 will be proved by the last part of this section. Let k be a fixed positive integer.

Theorem 3 ([ASO91a]) *For any finite set $S \in \mathcal{T}$, a reduced set of exactly k tree patterns with respect to S can be found in polynomial time with respect to $||S||$ if it exists.*

By theorem 3, Algorithm 1 finds the reduced set at Line 3. The language defined by the set may not be a minimal one, however we can find a minimal language containing S as one of its subsets. To find a minimal language, the algorithm tries to make $\bar{p}$ narrower by replacing each pattern $p \in \bar{p}$ by more specific pattern $lgg(S - L(\bar{p} - \{p\}))$. We call this process *tightening*. A reduced set $\bar{p}$ with respect to S is *of normal form with respect to S* if there is no $p \in \bar{p}$ such that $lgg(S - L(\bar{p} - \{p\})) <' p$. A normal form is a fixed point of the tightening process, furthermore, it is a *k-mmg* of S.

Lemma 4 *After executing the lines from Line 1 to Line 7 in Algorithm 1, any $\bar{p} \in \mathcal{TP}^k$ computed at Line 8 is of normal form with respect to S.*

Lemma 5 *If $\bar{p} \in \mathcal{TP}^k - \mathcal{TP}^{k-1}$ is reduced with respect to S, then there is no $\bar{q} \in \mathcal{TP}^{k-1}$ such that $S \subseteq L(\bar{q}) \subset L(\bar{p})$.*

Lemma 6 *If $\bar{p} \in \mathcal{TP}^k - \mathcal{TP}^{k-1}$ is of normal form with respect to S, then $\bar{p}$ is a k-mmg of S.*

From these lemmas, we show the main result of this section.

Theorem 7 ([ASO91a]) *Let k be a positive integer and Σ be an alphabet such that $\sharp\Sigma > k$. Then, for any finite set $S \in \mathcal{T}$ of trees, a k-mmg of S can be computed in polynomial time in $||S||$.*

The key to an efficient algorithm for computing *k-mmg* is to find a reduced set in polynomial time, which dominates the time complexity of total computation. As it can be easily seen, there is a trivial algorithm to find a reduced set of k tree patterns based on deviding S into k subsets. The algorithm enumerates every possible partitions $S_1, \ldots, S_k$ of S and checks whether $\bar{p} = \{lgg(S_1), \ldots, lgg(S_k)\}$ is reduced with respect to S. However, this simple method does not work efficiently, because the number of all the partitions of S is exponential in $\sharp S$.

To achieve efficiency, we adopt another approach. In Algorithm 2, we give a procedure $REDUCED(k, S)$ that computes a reduced set of exactly k tree patterns with respect to a given set S. We pay attention only to the combinations of k distinct trees selected from S instead of all the partitions of S. The number of such combinations is at most $(\sharp S)^k$. Assume that $\bar{p} = \{p_1, \ldots, p_k\}$ is reduced with respect to S. Then, $S_i = S - \bigcup_{j \neq i} L(p_j)$ is not empty for $1 \leq i \leq k$. Therefore, we can select k distinct trees $w_1, \ldots, w_k$ from

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

Input: A positive integer k and a finite set $S \subseteq \mathcal{T}$.
Output: A reduced set of exactly k tree patterns with respect to S.
Procedure:

```
1  for each combination w_1, ..., w_k ∈ S of k distinct trees do
2      for each 1 ≤ i ≤ k do
3          for each 1 ≤ j ≤ k such that j ≠ i do
4              T_j := 2-MaxTree(w_i, w_j);
5          A_i := { gci({q_1, ..., q_{k-1}}) ∈ TP | (q_1, ..., q_{k-1}) ∈ T_{h_1} × ... × T_{h_{k-1}} },
6          where {h_1, ..., h_{k-1}} = {1, ..., k} − {i};
7      for each combination p̄ = {p_1, ..., p_k} where p_i ∈ A_i for every 1 ≤ i ≤ k do
8          if S ⊆ L(p̄) then output p̄;
```

$S_1, \ldots, S_k$, respectively. For each $i = 1, \ldots, k$, let A_i be a set of maximally general tree patterns whose languages include $\{w_i\}$ but exclude $\{w_1, \ldots, w_k\} - \{w_i\}$, that is,

$$A_i = \left\{ q \in \mathcal{TP} \;\middle|\; \begin{array}{l} L(q) \cap \{w_1, \ldots, w_k\} = \{w_i\}, \text{ and } L(q) \subset L(p) \text{ implies} \\ L(p) \cap \{w_1, \ldots, w_k\} \neq \{w_i\} \text{ for any } p \in \mathcal{TP}. \end{array} \right\}.$$

Then, there exists $q_i \in A_i$ such that $L(p_i) \subseteq L(q_i)$. Therefore, $\{q_1, \ldots, q_k\}$ is reduced because $w_i \in S - \bigcup_{j \neq i} L(q_j)$. If each A_i can be computed in polynomial time, then we can find a reduced set with respect to S in polynomial time.

For disjoint finite sets $Pos, Neg \subseteq \mathcal{T}$, we call $p \in \mathcal{TP}$ a *maximal consistent tree pattern with Pos and Neg* iff (1) p is *consistent* with Pos and Neg, i.e., $Pos \subseteq L(p)$ and $Neg \cap L(p) = \emptyset$, and (2) $L(p) \not\subset L(q)$ for any consistent $q \in \mathcal{TP}$ with Pos and Neg. For instance, $f(x, a)$ and $f(x, x)$ are maximal consistent tree patterns with $\{f(a, a)\}$ and $\{f(a, b)\}$. By 2-$MaxTree(w, w')$ we denote the set of maximal consistent tree pattern with $Pos = \{w\}$ and $Neg = \{w'\}$, and by $MaxTree(w, \{w_1, \ldots, w_m\})$ the set of maximal consistent tree pattern with $Pos = \{w\}$ and $Neg = \{w_1, \ldots, w_m\}$. We denote by $gci(p_1, \ldots, p_n)$ the *greatest common instance* of $\{p_1, \ldots, p_n\}$, that is, the tree pattern obtained by unifying all the tree patterns $p_1, \ldots, p_n$.

If we can efficiently compute the set $MaxTree(w, \{w_1, \ldots, w_m\})$, then we can efficiently find a reduced set. By the following lemmas, its subproblem of computing the set 2-$MaxTree(w, w')$ can be solved in polynomial time in $||S||$ by the procedure shown in Algorithm 3.

Lemma 8 ([ASO91b]) *Let w^+, w^- be distinct trees and p be a maximal tree pattern consistent with $\{w^+\}$ and $\{w^-\}$. Then, p satisfies either (1) or (2) below. (See Figure 1)*

(1) For some node α in p, all leaves in p other than α are labeled by variables, and all

internal nodes in p are ancestors of the node α. Moreover, all variables that occur in p are mutually distinct.

(2) For some distinct leaves α_1, α_2 in p, all leaves in p are labeled by variables, and all internal nodes in p are ancestors of either α_1 or α_2. Moreover, only α_1, α_2 are leaves in p that are labeled by the same variable, and other leaves in p are labeled by mutually distinct variables.

Lemma 9 ([ASO91b]) *Given $w^+, w^- \in \mathcal{T}$, Algortihm 3 computes the set of all the maximal consistent tree patterns with $\{w^+\}$ and $\{w^-\}$ in polynomial time. The number of trees in the solution T by the algorithm is $O(|w^+|^2)$.*

Algorithm 3: The algorithm 2-$MaxTree(w^+, w^-)$

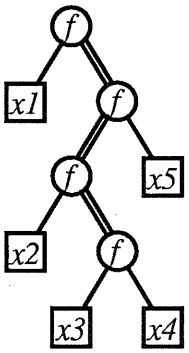
Input: A pair of distinct trees $w^+, w^- \in \mathcal{T}$.
Output: The set of all the maximal consistent tree patterns with $\{w^+\}$ and $\{w^-\}$.
Procedure:
$T := \emptyset$;
for each node α in w^+ such **do**
  if $w^+(\alpha) \in \Sigma$ and $w^+(\alpha) \neq w^-(\alpha)$ **then**
    $T := T \cup \{p\}$, where p is the term satisfying the condition (1) of Lemma 8;
for each pair α_1, α_2 of nodes in w^+ **do**
  if $w^+/\alpha_1 = w^+/\alpha_2$ but $w^-/\alpha_1 = w^-/\alpha_2$ **then**
    $T := T \cup \{p\}$, where p is the term satisfying the condition (2) of Lemma 8;
for each pair $p, q \in T$ **do**
  if $L(p) \subset L(q)$ **then** $T := T - \{p\}$;
output T;

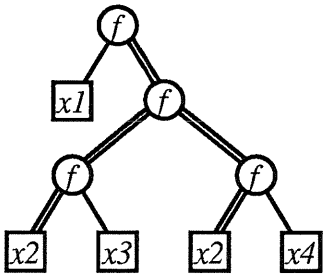
Where $w(\alpha)$ is the label of the node α of w, and w/α is the subtree of w whose root is α.

We have the following relationship between $MaxTree$ and 2-$MaxTree$.

$$MaxTree(w, \{w_1, \ldots, w_{k-1}\}) \subseteq \left\{ q \in \mathcal{TP} \;\middle|\; \begin{array}{l} q = gci(q_1, \ldots, q_{k-1}) \text{ for some } (q_1, \ldots, q_{k-1}) \in \\ 2\text{-}MaxTree(w, w_1) \times \ldots \times 2\text{-}MaxTree(w, w_{k-1}) \end{array} \right\}.$$

The number of elements of 2-$MaxTree(w, w_i)$ is proportional to the square of $|w|$ by Lemma 8 and we can compute the gci in polynomial time. Thus, it is clear that





$f(x_1, f(f(x_2, f(x_3, x_4)), x_5))$ $f(x_1, f(f(x_2, x_3), f(x_2, x_4)))$

Figure 1: Possible forms of maximal tree patterns consistent with $\{w^+, w^-\}$

the algorithm *REDUCED* computes the right hand side of the equation above after polynomial steps in the lines from Line 2 to Line 6 of Algorithm 2.

Lemma 10 ([ASO91a]) *There is an algorithm that, given disjoint sets* $\{w_0\}, \{w_1, \ldots, w_k\} \subseteq \mathcal{T}$, *computes a finite set* A *that contains all the maximal consistent tree patterns with* $\{w_0\}$ *and* $\{w_1, \ldots, w_k\}$ *in polynomial time in* $|w_0|$. *The number of the members in* A *is* $O(|w_0|^{2k})$.

Hence, we prove Theorem 3 shown in the beginning of this section, which describes the correctness of the algorithm *REDUCED*. The algorithm runs in time $O(n^{2k^2-k+1})$ in the total size $n = ||S||$ of the input S.

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 [Ish88] or CIGOL [MB88], 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 were obtained from our previous works [ASO91a, AIS92, IAS92] on inductive inferability of several subclasses of logic programs from positive facts using the *mmg* algorithm. Shinohara showed that a class of *linear Prologs* [Shi90] 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 [Llo84]. 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.

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 [Ang80] 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* [Wri89a, Wri89b]. Shinohara [Shi90] 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 7, it is also clear that Algorithm 4 produces each hypothesis in polynomial steps in $||S||$. Thus we have the following theorem.

Theorem 11 ([ASO91a]) *If $\sharp(\Pi \cup \Sigma) > k$ 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 [Sha81] 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:

$$C_0 = p(s_1, \ldots, s_m).$$
$$C_1 = p(t1, \ldots, t_m) \leftarrow p(x_1, \ldots, x_m).$$

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

$$S = \{app([], [], []), app([b], [a], [b, a]), app([a], [], [a]), app([], [a], [a]), app([a, b], [c, d], [a, b, c, d])\}$$

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_{FB}^{uniq} 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_{FB}^{uniq} 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-emtpy 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_{FB}^{uniq}, Algorithm 5 may need to transform P^* into a CFT_{FB}^{uniq} that has the same least Herbrand model with P^*. The transformation φ performs such model preserving generalization.

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

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

Since the class $\mathcal{CFT}_{FB}^{uniq}$ 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 $\mathcal{CFT}_{FB}^{uniq} \cup 2\text{-}\mathcal{UCP}$. 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 finding a consistent minimal hypothesis as described in the next subsection. However, from the uniqueness of 2-*mmg* for the model of CFT_{FB}^{uniq}, 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_{FB}^{uniq} $\varphi(P^*)$ can be finished in polynomial time in $||S||$. Hence we can obtain the following theorem.

Theorem 12 ([AIS92]) *If $|\Sigma| > 2$ 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 [IAS92]. The algorithm concentrates its attention 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. In [LW90], Lange and Wiehagen 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 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 [PSF91]. 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 kind of generalization of databases. Since a database

Table 1: A university database [CCH91]

Name	Category	Major	Birth_Place	GPA
Anderson	M.A.	History	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

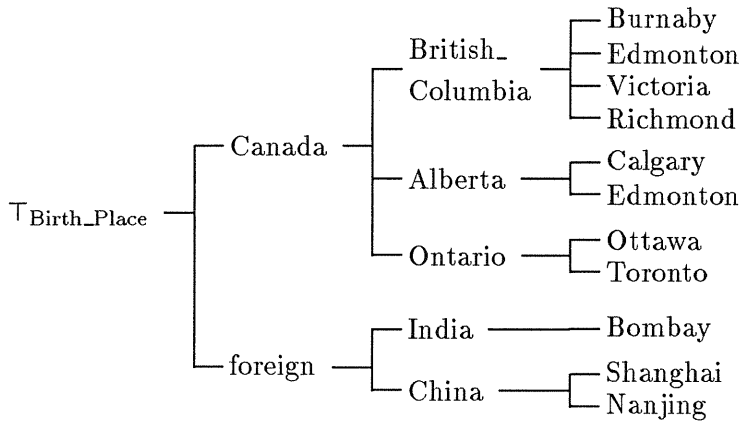


Figure 2: A conceptual hierarchy H_V over Birth_Place [CCH91]

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.

Cai et al. [CCH91] proposed a hueristics algorithm to discover characteristic rules. Their algorithm LCHR generalizes a database by using a given conceptual hierarchy. We show an example of databases in Table 1 extracted from Cai et al. [CCH91], which consists of tuples concerning to informations of graduate students. In Figure 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.

Table 2: A generalized relation obtained by the algorithm LCHR

Major	Birth_Place	GPA
science	Canada	excellent
science	foreign	good

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 thereshold k, the algorithm terminates and outputs the resulting generalized database as a chracteristic rule.

Table 2 is a generalized database obtained from the database in Table 1 by LCHR, where two columns for Name and Category that have no useful information are removed [CCH91]. The generalized database computed by LCHR characterizes the given database by a kind of clustering. The paper [CCH91] reported that the algorithm LCHR runs efficiently in the size of the input database and finds rules characterizing the database well.

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 substituing 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.

We first formalize the notion of databases with conceptual hierarchy. Let $\mathcal{V}$ be a class of mutually disjoint finite sets, we refer to the members in $\mathcal{V}$ as *domains*. Let $A_1, \ldots, A_n$ $(n > 0)$ be *attributes* associated with domains $V_1, \ldots, V_n \in \mathcal{V}$. A *database* D over attributes $A_1, \ldots, A_n$ $(n > 0)$ is a finite set of *tuples* $(t_1, \ldots, t_n) \in V_1 \times \ldots \times V_n$. A *conceptual hierarchy* over a domain V is a tree $H_V = (\{\top_V\} \cup C_V \cup V, \sqsubseteq)$ such that $\top_V$ is the root, C_V is the set of the internal nodes other than the root, and V is the set of the leaves. Elements of C_V are called *class names*. H_V represents an IS_A hierarchy, and $c \sqsubseteq c'$ means c IS_A c'. We say c' is the parent of c iff $c \sqsubseteq c'$.

To apply the *k-mmg* algorithm directly to the problem, we represent attribute values in D together with the conceptual hierarchy H_V by typed terms. We give the definition of typed terms of H_V. Let $H_V = (\{\top_V\} \cup C_V \cup V, \sqsubseteq)$ be a conceptual hierarchy, and X_c be a set of variables for each $c \in C_V$. We assume $X_c \cap X_{c'} = \emptyset$ if $c \neq c'$. Then, for each $c \in \{\top_V\} \cup C_V$, we define a set $\mathcal{TP}_c$ of *terms of sort* c as follows. If c is the parent of $v \in V$, then $v \in \mathcal{TP}_c$. If c is the parent of $c' \in C_V$ and $t \in \mathcal{TP}_{c'}$, then $c'(t) \in \mathcal{TP}_c$. If $x \in X_c$ for some $c \in C_V$, then $x \in \mathcal{TP}_c$. We denote by $\mathcal{T}_c$ the set of ground terms in $\mathcal{TP}_c$.

All attribute value and concept name, respectively, in H_V are associated with ground

Table 3: The transformed database in Step 1 of the method based on the *k-mmg*

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

terms and terms in $\mathcal{TP}_{\mathsf{T}_V}$ in a unique way. Thus, we write $\mathcal{TP}_V$ and $\mathcal{T}_V$ instead of $\mathcal{TP}_{\mathsf{T}_V}$ and $\mathcal{T}_{\mathsf{T}_V}$. We explain the correspondence by the following example. In the conceptual hierarchy of Figure 2, the sets of terms of sorts Alberta and Birth_Place, respectively, are

$$\mathcal{TP}_{\text{Alberta}} = \{\text{Calgary}, \text{Edmonton}, x, \ldots\},$$
$$\mathcal{TP}_{\mathsf{T}_{\text{Birth\_Place}}} = \{\text{foreign(China(Shanghai))}, \text{foreign(China}(y)), \text{foreign}(z), \ldots\},$$

where $x \in X_{\text{Alberta}}, y \in X_{\text{China}}, z \in X_{\text{foreign}}$. Then, the associated term with the attribute value Shanghai is foreign(China(Shanghai)), and the associated term with the concept name China is foreign(China(y)).

We represent a database over attributes $A_1, \ldots, A_n$ by a set of tuples $(t_1, \ldots, t_n)$ of typed terms in $\mathcal{T}_{V_1}, \ldots, \mathcal{T}_{V_n}$, respectively, where V_i associates A_i. A *generalized database* is a set of tuples $(p_1, \ldots, p_n)$ of typed terms in $\mathcal{TP}_{V_1}, \ldots, \mathcal{TP}_{V_n}$, respectively. According to the correspondence beetween concept names and typed terms described above, we can naturally define the transformation from databases to sets of tuples of typed terms.

For a tuple $p = (p_1, \ldots, p_n) \in \mathcal{TP}_{V_1} \times \ldots \times \mathcal{TP}_{V_n}$, we define the *language defined by* p, denoted by $L(p)$, as

$$L(p) = \{\, t \in \mathcal{T}_{V_1} \times \ldots \times \mathcal{T}_{V_n} \mid t \text{ is a ground instace of } p\}.$$

The *language* $L(D)$ *defined by a generalized database* D is defined by $L(D) = \bigcup_{p \in D} L(p)$. By $\mathcal{TPL}^k$, we denote the class of languages defined by generalized databases with at most k tuples. Note that if the conceptual hierarchy is finite, then languages in $\mathcal{TPL}^k$ are finite languages.

For the typled pattern languages, we can also define the notion of compactness with respect to containment and can show the corresponding theorem to Theorem2; if any internal node in a conceptual hierarchy H_V has more than k children, then the class $\mathcal{TPL}^k$ has compactness with respect to containment. We assume the compactness of languages.

Now, we describe our method of computing minimally generalized databases. The method consists of the following three steps. Let $k = 2$ be the threshold. We explain our method by the previous example.

Table 4: Computed sets in Step 2 of the method based on the *k-mmg*

A reduced set with respect to S

Major	Birth_Place	GPA
x_1	Canada(x_2)	x_3
y_1	foreign(y_2)	y_3

A generalized database computed by the *k-mmg* algorithm

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

Table 5: A generalized relation computed in Step 3 of the method based on the *k-mmg*

Major	Birth_Place	GPA
science	Canada	excellent
science	foreign	good

Assume that we are given the threshold $k = 2$, the database D shown in Table 1 and the corresponding conceptual hierarchy including Figure 2. The computation proceeds in the following way.

Step 1. Transform the given database to a set S of tuples of typed terms replacing all the attribute values v in D by the corresponding terms. The table in Table 3 represents the resulting set of tuples. We think of each tuple as an atom with the empty predicate symbol.

Step 2. Compute a *k-mmg* of the resulting set S of atoms. First, the algorithm find a reduced set with respect to S shown in Table4. Once the algorithm have a reduced set, it computes a one of *k-mmg*, shown in Table 4, minimizing the set by the tightening process.

Step 3. Transform the obtained *k-mmg* into the generalized database by using the inverse transformation of one used in Step 1.

We can easily see that the generalized database computed by our method corresponds to what the LCHR algorithm finds. In the work in [CCH91], 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. Under the assumption of the compactness with respect to containment, our method efficiently computes one of minimally generalized database in the sense of typed tree pattern languages.

Unfortunately, the *k-mmg* algorithm may not find all of the answer. 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.

References

[AIS92] Hiroki Arimura, Hiroki Ishizaka, and Takeshi Shinohara. Polynomial time inference of a subclass of context-free transformations. In *Proceedings of 5th Annual ACM Workshop on Computational Learning Theory*, pp.136–143, 1992.

[Ang80] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[ASO91a] Hiroki Arimura, Takeshi Shinohara, and Setsuko Otsuki. A polynomial time algorithm for finite unions of tree pattern languages. In *Proc. of the 2nd International Workshop on Nonmonotonic and Inductive Logics*, 1991. To appear in LNCS, Springer.

[ASO91b] Hiroki Arimura, Takeshi Shinohara, and Setsuko Otsuki. Polynomial time inference of unions of two tree pattern languages. *IEICE trans. Inf. and Syst.*, E75-D(7):426–434, 1992.

[CCH91] Y. Cai, N. Cercone, and J. Han. 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, 1991.

[DK90] Arthur L. Delcher and Simon Kasif. Efficient parallel term matching and anti-unification. In D. H. D. Warren and P. Szeredi, editors, *Logic Programming: Proceedings of the Seventh International Conference*, pp. 355–369. The MIT Press, 1990.

[IAS92] Hiroki Ishizaka, Hiroki Arimura, and Takeshi Shinohara. Efficient inductive inference of primitive prologs from positive data. In S. Doshita, K. Furukawa and T. Nishida, editors, *Proc. ALT '92*, pp. 135–146, 1991.

[Ish88] Hiroki Ishizaka. Model inference incorporating generalization. *Journal of Information Processing*, 11(3):206–211, 1988.

[LW90] S. Lange and R. Wiehagen, Polynomial-time Inference of Pattern Languages, *New Generation Computing* **8** (4), 361–370, 1991.

[LMM88] J-L. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pp. 587–625. Morgan Kaufmann, 1988.

[LM86] J-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3:301–317, 1986.

[Llo84] John W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1984.

[MB88] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proc. 5th International Conference on Machine Learning*, pp. 339–352, 1988.

[Mug90] Stephen Muggleton. Inductive logic programming. In S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori, editors, *Proc. ALT '90*, pp. 42–62. Ohmsha, 1990.

[Plo70] Gordon D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pp. 153–163. Edinburgh University Press, 1970.

[PSF91] G. Piatetsky-Shapiro and W. J. Frawley, editors. *Knowledge Discovery in Databases.* AAAI Press/The MIT Press, 1991.

[PV88] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *JACM*, 35(4):965–984, 1988.

[Rey70] John C. Reynolds. 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, 1970.

[Sha81] Ehud Y. Shapiro. Inductive inference of theories from facts. Technical Report 192, Yale University Computer Science Dept., 1981.

[Shi90] Takeshi Shinohara. 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, 1990.

[Wri89a] Keith Wright. 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, 1989.

[Wri89b] Keith Wright. *Inductive Inference of Pattern Languages.* PhD thesis, University of Pittsburgh, 1989.