数理解析研究所講究録
458巻 1982年 221-230

Polynomial Time Inference of Pattern Languages
(Extended Abstract)

by

Takeshi Shinohara

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

1. Introduction

In general, there have been known two kinds of inferences. One is from positive and negative data and the other is from only positive data. A general theory of inductive inference is found in literatures (for example, see Gold(1967) or Solomonoff(1964)). However it has been considered not so interesting to study inferences from positive data, since Gold(1967) proved a strong theorem which asserts that any class of languages over an alphabet Σ that contains every finite language together with at least one infinite language over Σ is not inferrable from positive data. For example, by his theorem, we can easily show that the class of regular sets is not inferrable from positive data. Recently Angluin(1980) gave a new life to the study of inference by characterizing the class of languages inferrable from positive data. She also gave interesting classes such as pattern languages.

A pattern is a nonempty finite string consisting of constants and variables. The language of a pattern is the set of all strings obtained by substituting any nonempty constant string for each variable in the pattern. Angluin(1979) proved that MINL calculation, which finds a minimal language containing a given nonempty finite set of strings, plays an important role in inference from positive data. She also proved that a special case of MINL for pattern languages, called ℓ-MINL, is computable. An ℓ-MINL calculation finds the longest pattern which represents a minimal pattern language containing a given finite set of strings.

The inferences, we deal with in this paper, are carried out by using ℓ-MINL calculation. The main interest of the present study is to find some classes of pattern languages whose inferences are not intractable and yet applicable to practical use.

For reasons of space, all the proofs of our results are omitted; they will be presented elsewhere (Shinohara, 1982).

2. Pattern Languages

First, we recall patterns and their languages in accordance with Angluin(1979), and then we introduce regular patterns and non-cross patterns. We also present a lemma which shows some basic properties of pattern languages.

Let Σ be a finite set of symbols containing at least two symbols and let $X = \{x_1, x_2, \ldots\}$ be a countable set of symbols disjoint from Σ. Elements of Σ are called constants and elements of X are called variables. A pattern is any nonempty finite string over $\Sigma \cup X$. The set of all patterns is denoted P. The language of a pattern p, denoted L(p), is the set of all strings obtained from the pattern p by substituting any nonempty string over Σ for each variable in p.

Now we define regular patterns and non-cross patterns.

i) A pattern p is called regular if each variable x in p appears exactly once in p.

ii) A pattern p is called non-cross if for each variable x in p there is no occurrence of y between the leftmost occurrence of x and the rightmost occurrence of x, where y is a variable not equal to x.

By definitions, if p is regular then clearly p is non-cross. It is easily shown that L(p) is a regular set iff p is regular. A pattern p which contains just one variable is a one-variable pattern by Angluin(1979). One-variable patterns are also non-cross. (See Fig. 1.)

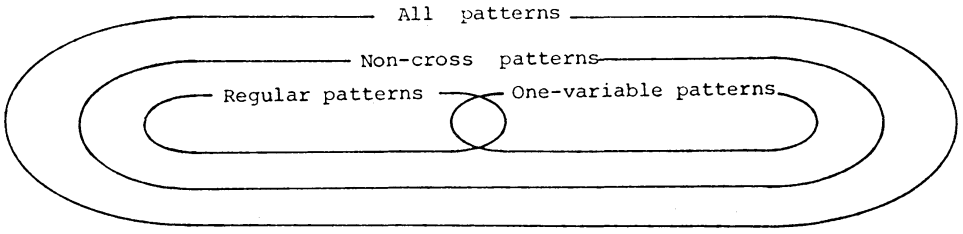

Fig. 1 Regular Patterns and Non-cross Patterns

Let f be a nonerasing homomorphism from P to P. If f(c) = c for any constant c, then f is called a substitution. If f is a substitution, f(x) is in X, and f(x) = f(y) implies x = y for any variables x and y, then f is called a renaming of variables. We define two binary relations on P as follows:

i) $p \equiv' q$ iff $p = f(q)$ for some renaming of variables f, and

ii) $p \leq' q$ iff $p = f(q)$ for some substitution f.

Note that we can define the language of p as $L(p)=\{w \in \Sigma^+ \mid w \leq' p\}$. These syntactic relations are characterized by the following lemma.

Lemma 1. (Angluin, 1979)

(1) For all patterns p and q, $p \equiv' q$ iff $L(p) = L(q)$.

(2) For all patterns p and q, if $p \leq' q$ then $L(p) \subseteq L(q)$, but the converse is not true in general.

(3) If p and q are patterns such that $|p| = |q|$, then $p \leq' q$ iff $L(p) \subseteq L(q)$.

We use a notation $[a_1/v_1, \ldots, a_k/v_k]$ to denote a substitution mapping each variable v_i to a string a_i and every other symbol to itself. When we apply this to a pattern, we put it to the right of the pattern, like p[a/x,b/y].

Examples. Let $\Sigma = \{0, 1, 2\}$ and let $X = \{x, y, \ldots\}$. Then p = x2x is a one-variable pattern, and its language is $L(p)=\{w2w \mid w \in \Sigma^+\}$. A pattern q = 0xy is regular, and a pattern r=xyx is not non-cross. Since p = r[2/y], we have $p \leq' r$. By Lemma 1, $L(p) \subseteq L(r)$.

3. An Upperbound of the Time Complexity of ℓ-MINL

In this section, we investigate the time complexity of ℓ-MINL calculation. ℓ-MINL is defined formally as follows (Angluin, 1979).

> ℓ-MINL = Given a nonempty finite set S of strings, find a pattern p (if any) of maximum possible length such that $S \subseteq L(p)$ and for no q do we have $S \subseteq L(q)$ and $L(q) \subsetneq L(p)$.

Theorem 1. The following procedure calculates ℓ-MINL(S) and it can be done by a deterministic polynomial time Turing machine with an oracle in NP, where S is a given nonempty finite set of strings and $w = a_1 \ldots a_m$ ($a_i \in \Sigma$) is one of the shortest strings in S.

```
p_1 := x_1...x_m ;
for i:=1 to m do
   begin
      q := p_i[a_i/x_i] ;
      j:=i+1 ;
      while S ⊄ L(q) and j ≤ m do
         begin
            q := p_i[x_j/x_i] ;
            j:=j+1
         end ;
      if S ⊆ L(q) then p_{i+1} := q
                  else p_{i+1} := p_i
   end ;
output p_{m+1} ;
halt ;
```

Fig.2 illustlates a computation of our ℓ-MINL procedure, where $\Sigma = \{0, 1, 2\}$, $X = \{x_1, x_2, x_3, \ldots\}$, $S = \{000, 12120\}$, and $w = 000$.

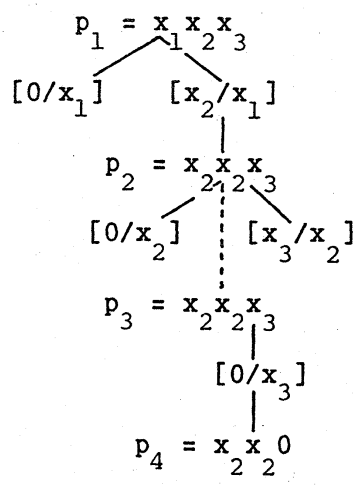

Fig. 2. Tree Search Method in ℓ-MINL

In Sections 4 and 5, we show that ℓ-MINL calculations for regular pattern languages and non-cross pattern languages are performed in polynomial time. The tree search method[1)] used by ℓ-MINL procedure above is shown to be applicable to these subclasses.

---

1) The original version of this method was invented by S. Miyano for regular pattern languages.

4. An ℓ-MINL Calculation for Regular Pattern Languages

Let $p = w_1 x_1 w_2 x_2 \ldots x_n w_{n+1}$ be a given regular pattern, where each w_i ($i = 1, 2, \ldots, n+1$) is any (possibly empty) string over Σ and $x_1, \ldots, x_n$ are distinct variables. Then we can construct a deterministic finite automaton DFA_1 which recognizes the language $\{w_1\}$ in $O(|w_1|)$ time. For each $i = 2, \ldots, n+1$ we can also construct a deterministic finite automaton DFA_i which recognizes the language represented by a regular expression $\Sigma^+ w$ in $O(|w_i|)$ time by using the method of pattern matching machine (Aho, et al., 1974). We can obtain the automaton DFA[p] which recognizes the language L(p), by catenating DFA_i and DFA_{i+1} for all $i = 1, \ldots, n$ as shown in Fig. 3.

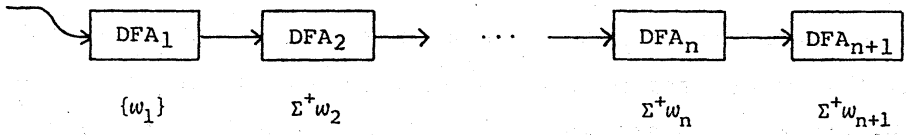

Fig. 3. Finite Automaton Recognizing a Regular Pattern Language

Lemma 2. For any regular pattern p, a deterministic finite automaton recognizing L(p) can be constructed in $O(|p|)$ time.

Lemma 3. For any regular pattern p and any string w, whether $w \in L(p)$ is decidable in $O(|p|+|w|)$ time.

Theorem 2. The following procedure calculates ℓ-MINL(S) for regular pattern languages in $O(m^2 n)$ time, where S is a given nonempty finite set of strings, $w = a_1 \ldots a_k$ ($a_i \in \Sigma$) is one of the shortest strings in S, $m = \max\{|w|;\ w \in S\}$ and $n = \mathrm{card}(S)$.

```
p1 := x1...xk ;
for i:=1 to k do
   begin
      q := pi[ai/xi] ;
      if S ⊆ L(q) then pi+1 := q
                  else pi+1 := pi
   end ;
output pk+1 ;
halt ;
```

5. An ℓ-MINL Calculation for Non-cross Pattern Languages

In this section, we show that ℓ-MINL for non-cross pattern languages is computable in polynomial time in the same way as in the previous section. To show that the membership for non-cross pattern languages is decidable in polynomial time, we use a two-way nondeterministic finite automaton with four heads (2NFA(4) for short). The definition and related concepts of two-way multihead nondeterministic finite automaton are found elswhere (for example, see Ibarra(1973)).

Lemma 4. For any non-cross pattern p, a 2NFA(4) which recognizes L(p) and has $O(|p|)$ states can be constructed in $O(|p|)$ time.

In general it is known that the acceptance of two-way nondeterministic pushdown automaton is decidable in polynomial time with respect to the length of the input string (Aho, et al., 1968). In our case, we may ignore the effect of pushdown store, and hence we just consider the effect of the number of states and the number of heads.

Lemma 5. For any non-cross pattern p and any string w, whether $w \in L(p)$ is decidable in polynomial time with respect to $|p|$ and $|w|$.

Theorem 3. The following procedure calculates ℓ-MINL for non-cross pattern languages in polynomial time, where S and w are similar to those in Theorem 2.

```
p_1 := x_1...x_k; j := 0 ;
for i := 1 to k do begin
    q := p_i[a_i/x_i] ;
    if S ⊈ L(q) then begin
        if j ≠ 0 then begin
            q := p_i[x_i/x_j] ;
            if S ⊆ L(q) then  p_{i+1} := q
                       else  p_{i+1} := p_i
        end ;
        j := i
    end
    else p_{i+1} := q
end ;
output p_{k+1} ; halt ;
```

6. Polynomial Time Inference from Positive Data

This section is devoted to discuss our main result.

Theorem 4. The classes of regular and non-cross pattern languages are plynomial time inferrable from positive data.

The ℓ-MINL calculation considered in the previous sections plays an important role in the discussion below. Throughout this section we omit the phrase "from positive data", hence for examle "inference" means "inference from positive data".

In order to prove the theorem we need some more preparation on inference. Consider an effective procedure M which requires inputs from time to time and produces outputs from time to time. Let $s = s_1, s_2, \ldots$ be an arbitrary infinite sequence, and let $g = g_1, g_2, \ldots$ be a sequence of outputs produced by M when inputs in s are successively given to M on requests. Then we say that M on input s converges to g_0 iff g is a finite sequence ending with g_0 or all but finitely many elements of g are equal to g_0.

Let $\mathcal{L} = L_1, L_2, \ldots$ be an indexed family of recursive languages, and let $s = s_1, s_2, \ldots$ be an arbitrary enumeration of some language L_i. Then M infers $\mathcal{L}$ iff M on input s converges to an index j with $L_j = L_i$. We call such a procedure M an inference machine and call its behavior an inference. We also call such indexed family $\mathcal{L}$ to be inferrable.

An inference is consistent iff a language L_{g_i} contains all inputs so far given whenever the machine produces g_i. An inference is conservative iff an output g_i from the machine is never changed unless L_{g_i} fails to contain the inputs. These two properties should be natural and valuable in inference problem. It is, however, known that inferrability does not always mean consistency and conservativeness (Angluin, 1979).

A class $\mathcal{L}$ is polynomial time inferrable iff there exists an inference machine M which infers $\mathcal{L}$ consistently and conservatively, and requests a new input in polynomial time (with respect to the inputs so far received) after the last input received. The behavior of such machine M is called a polynomial time inference.

The following theorem by Angluin (1979) shows the importance of ℓ-MINL calculation in inference. By this theorem, together with Lemma 3, Lamma 5, Theorem 2, and Theorem 3, we can easily prove our theorem 4.

Theorem 5. Let $\mathcal{L} = L_1, L_2, \ldots$ be a class of pattern languages for which ℓ-MINL is computable. Then the procedure Q below infers $\mathcal{L}$ consistently and conservatively.

```
procedure Q;
    g_1 := "none" ;  S_1 := ∅ ;
    for each input s_i do begin
        S_{i+1} := S_i ∪ {s_i} ;
        if s_i ∈ L_{g_i} then
            g_{i+1} := g_i
        else begin
            g_{i+1} := ℓ-MINL(S_{i+1}) ;
            output g_{i+1}
        end
    end
```

7. Discussions

Finally we discuss a practical application of our inference. In the department to which the author belongs, an information system SIGMA (Arikawa, et al., 1982) has been developed. Data dealt with in SIGMA are texts or strings of characters. S. Arikawa, the project leader, has proposed to add a data entry system with learning function to SIGMA. The present work is directly motivated by his idea.

At the stage of data entry from a keyboard, a user types, for example, some bibriographic data in the following way:

```
$
Author:  Angluin, D.
Title:   Inductive Inference of Formal Languages from Positive Data
Journal: Inform. Contr. 45
Year:    1980
$
Author:  Ibarra, O.H.
Title:   On Two-Way Multihead Automata
Journal: JCSS, 7
Year:    1973
$
...,
```

where $ is a special symbol to mark off records. A halfway through this entry process, the system will learn or infer a structure of records in a regular pattern of the form

Author: <w>Title: <x>Journal: <y>Year: <z>,

where <w>, <x>, <y>, and <z> are variables, and it will successively emit the constant parts "Author: ", "Title: ", "Journal: " and "Year: " as prompts. Then the user may only type the data corresponding to the variables <w>, <x>, <y>, and <z>.

The data entry system is a man-machine system. Hence the learning process should not be computationally hard. The present study guarantees that our system will satisfies the condition and will be practical.

In order to implement our data entry system with such learning function, a slight modification will naturally be needed. More detailed discussion will be given elsewhere.

ACKNOWLEDGEMENT

The author gratefully acknowledges the support and advice of Prof. Setsuo Arikawa. He also wishes to thank Mr. Satoru Miyano for his helpful suggestions on the tree search method and the membership problems considered in Sections 4 and 5, and Mr. Makoto Haraguchi for his useful comments.

REFERENCES

Aho, A.V., Hopcroft, J.E. and Ullman, J.D. (1968), Time and Tape Complexity of Pushdown Automaton Languages, Inform. Contr. 13, 186-206.

Aho, A.V., Hopcroft, J.E. and Ullman, J.D. (1974), "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, Mass.

Angluin, D. (1979), Finding Patterns Common to a Set of Strings, in "Proceeding, 11th Annual ACM Symposium on Theory of Computing," pp. 130-141.

Angluin, D. (1980), Inductive Inference of Formal Languages from Positive Data, Inform. Contr. 45, 117-135.

Arikawa, S. (1981), One-Way Sequential Search Systems and their Powers, Bull. Math. Stat., 19, 69-85.

Arikawa, S., Shinohara, T., Shiraishi, S. and Tamakoshi, Y. (1982), SIGMA - An Information System for Researchers Use, Bull. Informatics and Cybernetics, 20 (in press).

Gold, E.M. (1967), Language Identification in the Limit, Inform. Contr. 10, 447-474.

Hopcroft, J.E. and Ullman, J.D. (1969), "Formal Languages and their Relation to Automata," Addison-Wesley, Reading, Mass.

Ibarra, O.H. (1973), On Two-Way Multihead Automata, JCSS, 7, 28-36

Shinohara, T. (1982), Polynomial Time Inference of Pattern Languages and Its Application, in "Proceedings of The Seventh IBM Symposium on Mathematical Fundations of Computer Science," (to appear).

Solomonoff, R. (1964), A Formal Theory of Inductive Inference, Inform. Contr. 7, 1-22.