View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Elsevier - Publisher Connector

JOURNAL OF COMPUTER AND SYSTEM SCIENCES 21, 46–62 (1980)

Finding Patterns Common to a Set of Strings*

DANA ANGLUIN

Department of Mathematics, University of California, Santa Barbara, California 93106

Received July 1979

Assume a finite alphabet of *constant symbols* and a disjoint infinite alphabet of *variable symbols*. A *pattern* is a non-null finite string of constant and variable symbols. The *language* of a pattern is all strings obtainable by substituting non-null strings of constant symbols for the variables of the pattern. A *sample* is a finite nonempty set of non-null strings of constant symbols. Given a sample S, a pattern p is *descriptive of* S provided the language of p contains S and does not properly contain the language of any other pattern that contains S. The computational problem of finding a pattern descriptive of a given sample is studied. The main result is a polynomial-time algorithm for the special case of patterns containing only one variable symbol (possibly occurring several times in the pattern). Several other results are proved concerning the class of languages generated by patterns and the problem of finding a descriptive pattern.

1. INTRODUCTION

The problem that we consider in this paper is one of finding a "pattern" that describes a given finite set of strings. For example, $xx0$ describes {10100, 2342340, 000} and xyx^r describes {2896982, 426324, 11246394211}, where the superscript r denotes reversal.

Our notion of a "pattern" is a simple one; a pattern is just a concatenation of constant symbols and variable symbols, for example, $xx0$ or $x2yx3y$. (Other possible notions of "pattern" are discussed in Section 7, including reversal.) A pattern generates an associated set of strings, or language, consisting of all strings obtained by substituting non-null strings of constants for the variables of the pattern. For example, substituting 00 for x and 45 for y in the pattern $x2yx3y$ generates the string 0024500345. Some other strings in the language of this pattern are 123133, 66216631, 1233313333.

Given a finite set S of non-null strings, there are generally a number of patterns that generate all the strings in S, in particular, the trivial pattern x is always applicable. Among the patterns that generate all the strings in S we distinguish some as *descriptive* of S. A pattern p is descriptive of S if it generates all the elements of S and moreover no other pattern q both generates all of the elements of S and generates a strict subset of the language of p. Intuitively, no pattern q gives a strictly "closer fit" to the sample S than p does.

* This research was supported by the National Science Foundation under Grant MCS77-11360. A preliminary version of these results was presented at the 11th Annual ACM Symposium on Theory of Computing, Atlanta, Georgia, May 1979. Current address of author: Computer Science Department, Yale University, New Haven, Connecticut 06520.

0022-0000/80/040046-17$02.00/0

Copyright © 1980 by Academic Press, Inc.
All rights of reproduction in any form reserved.

We study the computational problem of finding a pattern descriptive of S, given a finite set S of non-null strings. Section 2 gives formal definitions of these notions and others. Section 3 studies the properties of the languages generated by patterns; for these languages the equivalence problem is decidable in linear time, the membership problem is decidable but NP-complete, the containment problem is open. Section 4 studies the general computational problem of finding a pattern descriptive of S; the problem is effectively solvable and a variant of it is NP-hard. An example is given to illustrate the nonuniqueness of patterns descriptive of S. Section 5 describes a particular way of representing the set of patterns "induced" in a string by substrings of it; the representation uses certain finite automata called pattern automata. Section 6 describes an algorithm that runs in polynomial time to find descriptive one-variable patterns. Section 7 contains remarks on areas in need of further work.

The emphasis of this work is somewhat different from work on pattern matching, either in the sense of the Knuth, Morris, and Pratt linear-time algorithm to search for occurrences of one string as a substring of another [8], or in the sense of the powerful pattern-matching facilities in the language SNOBOL [5]. Instances of these latter problems generally consist of a specification of a pattern and a string to match to (or test for membership in) the pattern. We instead specify a set of strings and ask that an algorithm find a particular pattern from the universe of patterns that satisfies some criterion of "goodness of fit" to the sample strings.

The primary motivation for the questions studied in this paper is an attempt to understand in formal terms the process of inductive inference, that is, the process of hypothesizing general rules from specific examples. We hope to find efficient procedures to perform such generalization in well-defined ways for relatively natural domains. We would argue that the polynomial-time algorithm that we present for finding a one-variable pattern that is descriptive of a given finite sample of strings is one such procedure. As the phenomena of inductive inference become better understood, many more such procedures will undoubtedly be discovered.

2. Definitions

Σ is a finite alphabet containing at least two symbols. The set of all finite strings of symbols from Σ is denoted Σ^*. The set of all finite non-null strings of symbols from Σ is denoted Σ^+. A *sample* is any finite nonempty subset of Σ^+.

$X = \{x_1, x_2, ...\}$ is a countable set of symbols disjoint from Σ. Elements of X are called *variables*. A *pattern* is any finite non-null string of symbols from $\Sigma \cup X$. The set of all patterns is denoted P_*. The *length* of a pattern p is just the number of symbols composing it, and is denoted $|p|$. The concatenation of two patterns p and q is denoted either pq or $p \cdot q$.

In order to discuss computations with patterns as inputs or outputs, we choose a particular concrete representation of patterns as strings over the finite alphabet $\Sigma \cup \{(,), 0, 1, ..., 9\}$, where we assume that $(,) \notin \Sigma$. A pattern p will be represented by replacing each variable x_i by its index i written in decimal notation and enclosed in parentheses. Thus if $p = x_2x_{16}34x_2$ then its representation is (2)(16)34(2). (Note that the length of p still refers to the abstract string, so $|p| = 5$ in this case.)

$\mathscr{H}$ denotes the set of all nonerasing homomorphisms (with respect to concatenation) of P_* to itself. An element of $\mathscr{H}$ that is the identity when restricted to Σ is called a *substitution*. A substitution that is a bijection of X to itself when restricted to X is called a *renaming* of variables. If $u_1, u_2,..., u_k$ are symbols from $\Sigma \cup X$ and $p_1, p_2,..., p_k$ are patterns, then we write $[p_1/u_1, p_2/u_2,..., p_k/u_k]$ for the homomorphism that maps u_i to p_i for $i = 1, 2,..., k$ and maps every other element of $\Sigma \cup X$ to itself. The application of this homomorphism to a string is by convention written to the right of the string.

If p and q are patterns then we say p and q are *equivalent*, denoted $p \equiv' q$, if and only if there exists a renaming of variables f such that $p = f(q)$. (In Section 3 we see that $\equiv'$ is indeed an equivalence relation, and coincides with the notion of language equivalence.) We define another binary relation: $p \leqslant' q$ if and only if for some substitution f, $p = f(q)$. (The intuitive reading of $\leqslant'$ may be taken to be "is less general than.") We note that since a substitution is a nonerasing homomorphism, if $p \leqslant' q$ then $|p| \geqslant |q|$.

For any pattern p, the *number of variables in* p is the number of distinct positive integers i such that the variable x_i occurs in p. For each $k = 0, 1,...,$ we let P_k denote the set of patterns p such that the number of variables occurring in p is k. The elements of P_1 are called *one-variable patterns*. Clearly $P_0 = \Sigma^+$ and $P_* = \bigcup_{k=0}^{\infty} P_k$.

A pattern p is in *canonical form* provided that if k is the number of variables in p, then the variables occurring in p are precisely $\{x_1, x_2,..., x_k\}$ and, further, for every i with $1 \leqslant i < k$, the leftmost occurrence of x_i in p is to the left of the leftmost occurrence of x_{i+1} in p. That is, the variables in p constitute an initial segment of X and are "introduced" in increasing order from left to right. Clearly, for any pattern p, there exists a unique pattern denoted $\hat{p}$ that is in canonical form such that $\hat{p} \equiv' p$. (It is not difficult to see that $\hat{p}$ may be computed in linear time from p. Since $p \equiv' q$ if and only if $\hat{p} = \hat{q}$, this gives us a linear-time method for testing whether $p \equiv' q$.)

If p is a pattern, the *language of* p, denoted $L(p)$, is $\{s \in \Sigma^+: s \leqslant' p\}$. Thus if $s \in L(p)$, $|s| \geqslant |p|$. If $s \in L(p)$, we also say that *p generates s*. A key notion for this paper is that of a pattern being a "good description" of a sample. Formally, a pattern p is *descriptive of* a sample S provided $S \subseteq L(p)$ and for every pattern q such that $S \subseteq L(q)$, $L(q)$ is not a proper subset of $L(p)$. That is, $L(p)$ is minimal in the set-containment ordering among all pattern languages containing the sample. We also define a restriction of this notion. If Q is any set of patterns, we say p is *descriptive of S within Q* provided $S \subseteq L(p)$ and for every $q \in Q$ such that $S \subseteq L(q)$, $L(q)$ is not a proper subset of $L(p)$.

EXAMPLE 2.1. Suppose $\Sigma = \{0, 1\}$ and $X = \{x, y,...\}$. Then $x0x01 \leqslant' yyx$ because $yyx[1/x, x0/y] = x0x01$. $L(xx) = \{ww: w \in \Sigma^+\}$, so $100100 \in L(xx)$ and $101 \notin L(xx)$. Clearly $L(x) = \Sigma^+$ and $L(101) = \{101\}$.

We refer the reader to [1] for definitions of the notions of polynomial-time computability and NP-completeness. The notion of finite automaton that we use in Section 5 and 6 is that of a deterministic incompletely specified finite-state acceptor [7].

The cardinality of a set C will be denoted $|C|$.

3. The Pattern Languages

In this section we consider the questions of deciding membership, equivalence, and containment for the pattern languages. We begin with some technical results.

Lemma 3.1. *For all patterns p and q,*

(a) *$\leqslant'$ is transitive,*

(b) *$p \leqslant' q$ implies $L(p) \subseteq L(q)$,*

(c) *$p \equiv' q$ if and only if $p \leqslant' q$ and $q \leqslant' p$.*

Proof. The relation $\leqslant'$ is transitive because the composition of two substitutions is a substitution, and this immediately implies part (b). If $p \equiv' q$ then since a renaming of variables is a special case of a substitution, $p \leqslant' q$ and $q \leqslant' p$. Conversely, if $p \leqslant' q$ and $q \leqslant' p$ then there exist substitutions f and g such that $p = f(q)$ and $q = g(p)$, so $p = f(g(p))$. Thus, since p is of finite length, g must map the variables in p to variables and must be injective on the variables of p. Hence there exists a renaming of variables h such that $h(p) = g(p) = q$ so $p \equiv' q$. ▌

Let p be any pattern. Fix two distinct letters $a, b \in \Sigma$. We define a particular non-null finite subset of $L(p)$ which we denote by $S(p)$. Consider the substitutions defined for all positive integers i, j:

$$\begin{aligned} f_a(x_i) &= a, \\ f_b(x_i) &= b, \\ g_j(x_i) &= a \qquad \text{if } \ i = j, \\ &= b \qquad \text{otherwise}. \end{aligned}$$

Let $S(p)$ consist of the set of strings $\{f_a(p), f_b(p), g_1(p), g_2(p), \ldots\}$. If p contains no variables, then $S(p) = L(p) = \{p\}$. If p contains one variable then $S(p)$ consists of just two strings, and if p contains $k \geqslant 2$ variables then $S(p)$ consists of $k + 2$ strings. Clearly $S(p) \subseteq L(p)$, and may be constructed in time polynomial in the length of the representation of p.

Lemma 3.2. *Let p and q be patterns such that $|p| = |q|$, and $S(p) \subseteq L(q)$. Then $p \leqslant' q$.*

Proof. Note that every element $s \in S(p)$ has $|s| = |p| = |q|$. Let m and n be positive integers not exceeding $|p|$. By construction of $S(p)$, p has a constant $c \in \Sigma$ at position m if and only if every $s \in S(p)$ has the constant c at position m. Also, if p has two different variables at positions m and n, then there is a string $s \in S(p)$ that has different constant symbols at positions m and n. Since $S(p) \subseteq L(q)$, for every $s \in S(p)$ there exists a substitution h such that $s = h(q)$. Since $|s| = |q|$, h must map each variable of q to a string of length one. Now suppose q has the constant $c \in \Sigma$ at some position n. Then $h(q) = s$ must have c at position n, and since this holds for all $s \in S(p)$, p must have c at position n. Suppose q has the variable x_i at two positions m and n. Then $h(q) = s$ must have the same (constant) symbol at positions m and n, and since this holds for all $s \in S(p)$, p must have the same symbol (either a constant or a variable) at positions m and n. Thus the set of positions of x_i in q are all occupied by the same symbol, say $g(x_i)$, in p. We may extend g to be a substitution such that $p = g(q)$, so $p \leqslant' q$. ▌

As direct corollaries of this result we have:

COROLLARY 3.3. *For any pattern p, p is descriptive of the sample $S(p)$.*

Proof. Assume to the contrary that there exist patterns p and q such that $S(p) \subseteq L(q)$ and $L(q) \subsetneq L(p)$. Then we must have both $|p| \geqslant |q|$ and $|q| \geqslant |p|$, so $|p| = |q|$. By Lemma 3.2, $p \leqslant' q$, so $L(p) \subseteq L(q)$, a contradiction. ▌

COROLLARY 3.4. *Let p and q be any patterns such that $|p| = |q|$. Then $L(p) \subseteq L(q)$ if and only if $p \leqslant' q$.*

Proof. The "if" part is implied by Lemma 3.1(b). If $L(p) \subseteq L(q)$ then $S(p) \subseteq L(q)$, so Lemma 3.2 gives $p \leqslant' q$. ▌

Equivalence

THEOREM 3.5. *For all patterns p and q, $L(p) = L(q)$ if and only if $p \equiv' q$.*

Proof. If $L(p) = L(q)$ then $|p| \geqslant |q|$ and $|q| \geqslant |p|$, so $|p| = |q|$. Then by two applications of Corollary 3.4, $p \leqslant' q$ and $q \leqslant' p$, so $p \equiv' q$ by Lemma 3.1(c). Conversely, if $p \equiv' q$ then $p \leqslant' q$ and $q \leqslant' p$ by Lemma 3.1(c), so $L(p) \subseteq L(q)$ and $L(q) \subseteq L(p)$ by Lemma 3.1(b), and thus $L(p) = L(q)$. ▌

Since the equivalence of two patterns may be tested in time linear in the representations of the two patterns, this theorem implies a linear-time algorithm for deciding whether two patterns generate the same language.

Membership

THEOREM 3.6. *The problem of deciding whether $s \in L(p)$ for an arbitrary string $s \in \Sigma^*$ and pattern p is NP-complete.*

Proof. We first note that the problem of deciding whether $p \leqslant' q$ given arbitrary patterns p and q is in NP; a nondeterministic Turing machine may simply guess a substring of p for each variable appearing in q and check whether the implied substitution yields p from q. (The same reasoning may be applied to see that if there is a fixed constant bound k on the number of different variables occurring in q, then there is a deterministic algorithm that runs in time $O(n^{2k+1})$ to decide whether $p \leqslant' q$ that works by enumerating all k-tuples of substrings of p and testing the resulting substitutions.) Thus if $s \in \Sigma^*$ and p is a pattern, the problem of deciding whether $s \in L(p)$ is in NP. To see that this problem is complete in NP, we exhibit a polynomial-time reduction to it of the problem of deciding whether a propositional formula in conjunctive normal form with three literals per clause is satisfiable. Since this latter problem is NP-complete [1], this will prove the theorem.

Let ϕ be a given propositional formula with three literals per clause. Assume the variables of ϕ are $V_1, V_2, \ldots, V_n$ and the clauses are $C_1, C_2, \ldots, C_m$, where each clause is an ordered set of three terms each of which is either a variable or the complement of a variable. Since $|\Sigma| \geqslant 2$, we may choose two distinct symbols from Σ, say 0 and 1 for concreteness. We shall construct from ϕ a pattern p containing the $2(m+n)$ variables x_i, y_i, z_j, u_j for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$. Define for $j = 1, 2, \ldots, m$ and $k = 1, 2, 3$

$$\begin{aligned} f(j,k) &= x_i \qquad \text{if the } k\text{th literal in } C_j \text{ is } V_i, \\ &= y_i \qquad \text{if the } k\text{th literal in } C_j \text{ is } \bar{V}_i. \end{aligned}$$

Let

$$p = 0p_1 0p_2 \cdots 0p_n 0q_1 0q_2 \cdots 0q_m 0r_1 0r_2 \cdots 0r_m 0,$$

where

$$\begin{aligned} p_i &= x_i y_i \\ q_j &= f(j,1) \cdot f(j,2) \cdot f(j,3) \cdot z_j \\ r_j &= z_j u_j \end{aligned}$$

for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$. Now we construct a related string

$$s = 0s_1 0s_2 \cdots 0s_n 0t_1 0t_2 \cdots 0t_m 0w_1 0w_2 \cdots 0w_m 0,$$

where

$$\begin{aligned} s_i &= 111 \\ t_j &= 1111111 \\ w_j &= 1111 \end{aligned}$$

for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$. It is clear that there is a deterministic polynomial time algorithm to construct p and s from ϕ. We shall now show that $s \in L(p)$ if and only if ϕ is satisfiable, which will complete the proof of the theorem.

Suppose first that ϕ is satisfiable. Then let $\alpha\colon \{V_1, V_2, \ldots, V_n\} \rightarrow \{0, 1\}$ be an assignment satisfying ϕ. We define a substitution h such that $s = h(p)$ as follows. For each $i = 1, 2, \ldots, n$, if $\alpha(V_i) = 1$ then let $h(x_i) = 11$ and $h(y_i) = 1$, while if $\alpha(V_i) = 0$ then let $h(x_i) = 1$ and $h(y_i) = 11$. In any case $h(p_i) = h(x_i y_i) = 111 = s_i$ for each i. Consider some j with $1 \leqslant j \leqslant m$. Suppose for concreteness that C_j is $(V_1 + \bar{V}_2 + V_4)$, so that q_j is $x_1 y_2 x_4 z_j$. Since α satisfies ϕ, it satisfies C_j, so $\alpha(V_1) = 1$ or $\alpha(V_2) = 0$ or $\alpha(V_4) = 1$. Hence at least one of x_1, y_2, x_4 is assigned the string 11 by h, so $h(x_1 y_2 x_4)$ is four, five, or six 1's. Define $h(z_j)$ to be three, two, or one 1's and $h(u_j)$ to be one, two, or three 1's, respectively, so that

$$h(q_j) = h(x_1 y_2 x_4 z_j) = 1111111 = t_j$$

and

$$h(r_j) = h(z_j u_j) = 1111 = w_j .$$

Since j was arbitrary, we see that we may extend h to a substitution such that $s = h(p)$, so $s \in L(p)$.

Conversely, suppose that $s \in L(p)$. Then there is a substitution h such that $s = h(p)$. Since s and p contain the same number of 0's, h must assign some non-null string of 1's to each variable in p. Thus for each $i = 1, 2,..., n$, $h(x_i y_i) = 111$, so either $h(x_i) = 11$ and $h(y_i) = 1$ or vice versa. Define $\alpha(V_i) = 1$ if $h(x_i) = 11$ and 0 otherwise, for $i = 1, 2,..., n$. We show that the assignment α satisfies ϕ. Let j be arbitrary, with $1 \leqslant j \leqslant m$. Suppose for concreteness that $C_j = (V_1 + \bar{V}_2 + V_4)$. Since $h(z_j u_j) = h(r_j) = 1111$, $h(z_j)$ is one, two, or three 1's. Since $q_j = x_1 y_2 x_4 z_j$ and $h(q_j) = 1111111$, it must be that $h(x_1 y_2 x_4)$ is four, five, or six 1's. Thus at least one of x_1, y_2, x_4 must be assigned the string 11 by h. By definition of α, this implies that $\alpha(V_1) = 1$ or $\alpha(V_2) = 0$ or $\alpha(V_4) = 1$, so α satisfies clause C_j. Since j was arbitrary, we conclude that α satisfies ϕ, and ϕ is indeed satisfiable. ▌

COROLLARY 3.7. *The problem of deciding whether $p \leqslant' q$ given arbitrary patterns p and q is* NP-*complete.*

Containment

Recall from Lemma 3.1(b) that $p \leqslant' q$ implies $L(p) \subseteq L(q)$. Corollary 3.4 shows that in the special case when $| p | = | q |$, the converse also holds. To see that the converse does not hold in general, we consider the following example.

EXAMPLE 3.8. Let $\Sigma = \{0, 1\}$. Let $p = 0x10xx1$ and $q = xxy$. Then clearly $p \not\leqslant' q$. Suppose $s \in L(p)$. Then $s = 0t10tt1$ for some non-null string $t \in \Sigma^*$. Either $t = 0u$ or $t = 1u$ for some string $u \in \Sigma^*$. In the first case, $s = 00u100u0u1 = q[0/x, u100u0u1/y]$. In the second case, $s = 01u101u1u1 = q[01u1/x, u1/y]$. Thus in either case, $s \in L(q)$. Hence $L(p) \subseteq L(q)$.

This example may easily be generalized to apply to any finite cardinality for Σ, based on the $| \Sigma |$ cases for the first letter of the substituted string t. For example, for $\Sigma = \{0, 1, 2\}$ we may take $p = 0x10xx20x10xxx2$ and $q = xxy$. Another special case in which the converse of Lemma 3.1(b) holds is the following.

LEMMA 3.9. *Let p be any pattern and let q be a one-variable pattern. Then $L(p) \subseteq L(q)$ if and only if $p \leqslant' q$.*

Proof. Lemma 3.1(b) gives the "if" direction. Suppose $L(p) \subseteq L(q)$. Suppose q contains k occurrences of the variable x, and write q in the form $q_0 x q_1 x \cdots q_{k-1} x q_k$, where each $q_i \in \Sigma^*$. Since $S(p) \subseteq L(q)$, p must be of the form $q_0 p_1 q_1 p_2 \cdots q_{k-1} p_k q_k$ for some patterns $p_1, p_2,..., p_k$ all having a common length $l = | p_i |$. Furthermore, if f_a, b_b, g_1, g_2,... are the substitutions defined before Lemma 3.2, $f_a(p_i) = f_a(p_j)$, $f_b(p_i) = f_b(p_j)$, and $g_r(p_i) =$

$g_r(p_j)$ for all $1 \leqslant i, j \leqslant k$ and $r \geqslant 1$. From this we may easily conclude that $p_i = p_j$, for all $1 \leqslant i, j \leqslant k$, so $p = q[p_1/x]$ and $p \leqslant' q$. ▌

In summary, we may effectively decide whether $L(p) \subseteq L(q)$ given p and q in the special cases when either $|p| = |q|$ or q is a one-variable pattern, using Corollary 3.4, Lemma 3.9, and the effective decidability of $p \leqslant' q$. In the general case, however, the question of whether there is an effective procedure to decide whether $L(p) \subseteq L(q)$ given p and q appears to be unresolved. Some recent work of Makanin [9] showing the solvability of systems of equations in free semigroups may shed some light on this problem.

Some Other Properties

THEOREM 3.10. *The class of pattern languages is incomparable with the class of regular languages and with the class of context-free languages. The class of pattern languages is not closed under any of these operations: union, complement, intersection, Kleene plus, homomorphism, or inverse homomorphism. It is closed under concatenation and reversal.*

Proof. The only finite pattern languages are the singleton subsets of Σ^+. Thus some regular languages are not pattern languages. The pattern language $L(xx)$ is not context-free [7]. Suppose for concreteness Σ contains the distinct symbols 0 and 1. Then $L(0) \cup L(1)$, $\neg L(0)$, $L(0) \cap L(1)$, and $(L(0))^+$ are not pattern languages (another, less trivial, example for intersection is given in the next section). If we define $h(b) = 0$ for all $b \in \Sigma$ and extend h to be a (nonerasing) homomorphism on Σ^*, then $h(L(x)) = (L(0))^+$, which is not a pattern language. If $h(0) = 1$ and $h(1) = 11$ then $h^{-1}(111) = \{01, 10, 000\}$, which is not a pattern language.

Given two patterns p and q, we rename (if necessary) the variables that occur in q to obtain a pattern q' which is equivalent to q and contains no variables in common with p. Then we have $L(p) \cdot L(q) = L(p \cdot q')$, so the concatenation of pattern languages is a pattern language. Also, it is not difficult to see that $(L(p))^r = L(p^r)$, where the superscript r denotes reversal, so the reversal of a pattern language is a pattern language. ▌

4. FINDING A DESCRIPTIVE PATTERN

We now consider the following computational problem: given a sample S, find a pattern p that is descriptive of S.

THEOREM 4.1. *There is an effective procedure which, given a sample S as input, outputs a pattern p that is descriptive of S.*

Proof. The procedure is defined as follows. Given an arbitrary sample S, let $l \geqslant 1$ denote the minimum length of any string in S. Enumerate the finitely many canonical patterns p with $|p| \leqslant l$, and for each one test whether $S \subseteq L(p)$. (This uses the decidability of membership for the pattern languages, and the fact that S is an explicitly given finite set.) Let C denote the resulting set of canonical patterns p of length $\leqslant l$ such that $S \subseteq L(p)$. C is a nonempty finite set. Let m denote the maximum length of any element of

C and let C' denote the set of elements of C of length m. Then C' is a nonempty finite set. Using the decidability of $\leqslant'$, find and output any element p that is minimal in C' with respect to the partial ordering $\leqslant'$.

This procedure is clearly effective. We must see that the output pattern p is descriptive of S. Let q be any pattern such that $S \subseteq L(q)$. Then $|q| \leqslant l$ and $\hat{q} \in C$. If $|q| < |p|$ then $L(q) \not\subseteq L(p)$. If $|q| = |p|$ then $\hat{q} \in C'$. Then by the choice of p, either $p = \hat{q}$, so $L(p) = L(q)$, or $\hat{q} \nleqslant' p$. By Corollary 3.4, $\hat{q} \nleqslant' p$ and $|q| = |p|$ implies $L(q) \not\subseteq L(p)$. Hence in any case, $L(q)$ is not a proper subset of $L(p)$, so p is descriptive of S. ∎

Elsewhere [2] we have shown that the procedure of Theorem 4.1 may be used to construct an inference machine that correctly identifies the class of pattern languages in the limit from positive data. Furthermore, this machine may be arranged so that when a guess is made, it is consistent with the sample read in so far, and a guess is not changed unless it fails to be consistent with some new sample string.

The algorithm described in the above proof is time-consuming; the number of patterns enumerated may grow exponentially in the length of input sample, and the tests performed on the patterns are in the general case NP-complete problems. In Section 6 we give a polynomial-time algorithm for this problem in the special case of one-variable patterns. Below we shall give some partial evidence for the difficulty of the general case.

Observe that the algorithm described in the proof of Theorem 4.1 actually solves the following problem: given a sample S, find a pattern p of the maximum possible length that is descriptive of S. The reason for this is that we do not know whether containment is a decidable problem for the pattern languages, hence we maneuver so that the special case of Corollary 3.4 is applicable. For this stronger version of the basic problem, we are able to show the following.

THEOREM 4.2. *If $P \neq NP$ then there is no polynomial-time algorithm to solve the following problem: given a sample S, find a pattern p of maximum possible length that is descriptive of S.*

Proof. Suppose there exists an algorithm A that runs in polynomial time and is such that for any sample S, A on input S outputs a pattern p of the maximum possible length that is descriptive of S. We shall use A to construct a polynomial-time algorithm to decide whether $s \in L(p)$ for an arbitrary string $s \in \Sigma^*$ and pattern p. Since this latter problem is NP-complete (Theorem 3.6), this will imply $P = \mathrm{NP}$, proving the theorem.

Let a string s and pattern p be given. In polynomial time we may construct the sample $S = \{s\} \cup S(p)$. (The sample $S(p)$ is defined before Corollary 3.2 in Section 3.) Run A on input S and denote the output by q. Since $S \subseteq L(q)$, $|q| \leqslant |p|$. Clearly if $q \equiv' p$ then $s \in L(p)$. If $q \not\equiv' p$, there are two cases to consider. The first case is that $|q| < |p|$; since q has maximum possible length among all patterns generating S, it must be that $S \not\subseteq L(p)$. But by construction $S(p) \subseteq L(p)$, so we must have $s \notin L(p)$. The second case is that $|p| = |q|$. Since $S(p) \subseteq L(q)$, we may apply Lemma 3.2 to obtain $p \leqslant' q$. Hence $L(p) \subsetneq L(q)$. Since q is descriptive of S, it must be that $S \not\subseteq L(p)$, so $s \notin L(p)$. Thus we have shown that $p \equiv' q$ if and only if $s \in L(p)$. We may test whether $p \equiv' q$ in linear time, which concludes the construction. ∎

We have been unable to prove this result for the original problem, without the restriction to a longest descriptive pattern. One approach is to try to replace $S(p)$ by a sample $T(p) \subseteq L(p)$ such that for any q, $T(p) \subseteq L(q)$ implies $L(p) \subseteq L(q)$. It is not difficult to see that such a sample must exist for any pattern p. However, it is also the case that there is an algorithm to decide the containment of pattern languages if and only if there is an algorithm to compute such a sample $T(p)$ for any pattern p. Since the containment problem is open, this approach seems infeasible at present.

We give an example to illustrate the fact that there may be a pattern $q \not\equiv' p$ that is descriptive of $S(p)$. This also illustrates the nonuniqueness of the patterns descriptive of a sample, and of the fact that the intersection of two pattern languages is not in general a pattern language.

EXAMPLE 4.3. Let $\Sigma = \{0, 1\}$, $p = x010y$, and $q = xxy$. Then $S(p) = \{00100, 10101, 00101, 10100\}$. Clearly $S(p) \subseteq L(q)$. By Corollary 3.3, p is descriptive of $S(p)$. To see that q is descriptive of $S(p)$, consider any pattern p_1 such that $S(p) \subseteq L(p_1)$. Then $|p_1| \leqslant 5$. Consider the following cases for the length of p_1.

(i) $|p_1| < 3$. Since $|q| = 3$ this implies that $L(p_1) \not\subseteq L(q)$.

(ii) $|p_1| = 3$. Suppose that $L(p_1) \subseteq L(q)$. Then by Corollary 3.4, $p_1 \leqslant' q$. But since $S(p) \subseteq L(p_1)$, p_1 must begin with some variable u and end with some distinct variable z. Thus the substitution to obtain p_1 from q must map x to u and y to z, so $p_1 = uuz$, and $L(p_1) = L(q)$. Thus in this case either $L(p_1) \not\subseteq L(q)$ or $L(p_1) = L(q)$.

(iii) $|p_1| = 4$. Since $S(p) \subseteq L(p_1)$, p_1 must begin with some variable u and end with some distinct variable z. The two middle symbols of p_1 cannot be equal for then we would have 01, 010, or 10 in $L(xx)$, a contradiction. By enumerating the 21 canonical patterns of length 4 satisfying these two constraints, and retaining those that generate $S(p)$, we find that p_1 must be equivalent to one of the nine patterns $uvwz$, $uvuz$, $uzvz$, $uv0z$, $uv1z$, $u0vz$, $u1vz$, $u01z$, $u10z$. The language generated by each of these nine patterns is a proper superset of $L(p)$, and $100101 \in L(p) - L(q)$, so in this case $L(p_1)$ is not a proper subset of $L(q)$.

(iv) $|p_1| = 5$. Then by Lemma 3.2, $p \leqslant' p_1$, so $L(p) \subseteq L(p_1)$. Since $100101 \in L(p) - L(q)$, in this case also $L(p_1)$ is not a proper subset of $L(q)$.

Hence in all cases, $L(p_1)$ is not a proper subset of $L(q)$ and we conclude that q is also descriptive of $S(p)$. ▌

5. A REPRESENTATION OF INDUCED PATTERNS

In this section we describe a particular method of representing the patterns "induced" in a string by a collection of substrings of it. This representation is used in the next section to construct a polynomial-time algorithm to find a one-variable pattern descriptive of a given sample. With this application in view, and also for clarity of presentation, we give the construction and results for the one-variable case, and indicate how they may be generalized.

One-Variable Pattern Automata

Let s, $t \in \Sigma^+$, where t is a substring of s. Let $P(s; t)$ denote the set of all one-variable patterns p such that the one variable occurring in p is x and $p[t/x] = s$. For example, if $s = 1101110$ and $t = 11$ then $P(s; t) = \{x01110, 110x10, 1101x0, x0x10, x01x0\}$.

We construct a particular finite automaton denoted by $A(s; t)$ to recognize $P(s; t)$. The states of $A(s; t)$ are all ordered pairs (i, j) such that $0 \leqslant i, j$ and $i + j \mid t \mid \leqslant \mid s \mid$. The initial state is $(0, 0)$. The final states are all states (i, j) such that $j \geqslant 1$ and $i + j \mid t \mid = \mid s \mid$. The state (i, j) signifies that in the input string, i constant symbols and j occurrences of x have been read so far. The transitions of the machine are given by:

$\delta((i, j), b) = (i + 1, j)$ if the symbol at position $1 + i + j \mid t \mid$ of s is b.
$\delta((i, j), x) = (i, j + 1)$ if there is an occurrence of t as a substring of s beginning at position $1 + i + j \mid t \mid$ of s.

For the example given above of $S = 1101110$ and $t = 11$, the reachable part of $A(s; t)$ is depicted in Fig. 1. We note that $A(s; t)$ is in general not reduced; the reason for this choice will become clearer in connection with the Intersection Theorem (5.2).

THEOREM 5.1. *The language recognized by* $A(s; t)$ *is precisely* $P(s; t)$, *that is,* $L(A(s; t)) = P(s; t)$.

Proof. Let $p = s_0 x s_1 x \cdots s_{n-1} x s_n$, where $n \geqslant 1$ and each $s_j \in \Sigma^*$. Then $p \in L(A(s; t))$ if and only if for all $j = 0, 1, \ldots, n$ the substring of length $\mid s_j \mid$ beginning at position $j \cdot \mid t \mid + \sum_{k=0}^{j-1} \mid s_k \mid$ of s is s_j, and the substring of length $\mid t \mid$ beginning at position $j \cdot \mid t \mid + \sum_{k=0}^{j} \mid s_k \mid$ is t, which is true if and only if $p[t/x] = s$, i.e., $p \in P(s; t)$. ∎

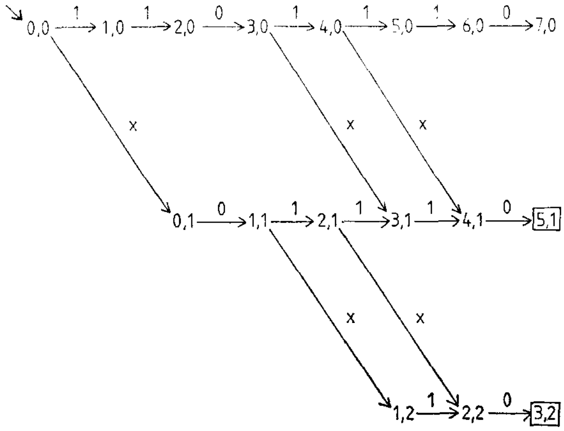

FIG. 1. The automaton $A(1101110; 11)$.

Let $A_i = \langle Q_i, q_0, \delta_i, F_i\rangle$ for $i = 1, 2$ be two finite automata over the alphabet Σ with the same initial state q_0, where Q_i is the set of states, δ_i is the transition function, and F_i is the set of final states of A_i. Define $A_1 \subseteq A_2$ if and only if $Q_1 \subseteq Q_2$, $F_1 \subseteq F_2$ and the function δ_2 is an extension of δ_1. Define $A_1 \cap A_2$ to be the finite automaton $A_3 = \langle Q_3, q_0, \delta_3, F_3\rangle$, where $Q_3 = Q_1 \cap Q_2$, $F_3 = F_1 \cap F_2$, and for all $q \in Q_3$ and $b \in \Sigma$, if $\delta_1(q, b)$ and $\delta_2(q, b)$ are defined and equal, then $\delta_3(q, b)$ is defined and equal to their common value. Clearly $A_1 \cap A_2 \subseteq A_1$ and $A_1 \cap A_2 \subseteq A_2$.

DEFINITION. A finite automaton A is a *one-variable pattern automaton* if and only if there exist $s, t \in \Sigma^+$, where t is a substring of s, and $A \subseteq A(s; t)$.

Intuitively speaking, a one-variable pattern automaton is obtained from some $A(s; t)$ by (optionally) erasing some of the states, transitions, and/or final-markers. The result for pattern automata is the following Intersection Theorem:

THEOREM 5.2. *Let A_1 and A_2 be one-variable pattern automata. Then $L(A_1) \cap L(A_2) = L(A_1 \cap A_2)$.*

Proof. Suppose $A_i = \langle Q_i, q_0, \delta_i, F_i\rangle$ for $i = 1, 2, 3$ and $A_3 = A_1 \cap A_2$. Suppose $p \in L(A_3)$. By induction we may see that for every prefix p' of p, $\delta_3(q_0, p') = \delta_1(q_0, p') = \delta_2(q_0, p')$. Since $\delta_3(q_0, p) \in F_3$ and $F_3 = F_1 \cap F_2$, both A_1 and A_2 accept p, i.e., $p \in L(A_1) \cap L(A_2)$. Conversely, suppose $p \in L(A_1) \cap L(A_2)$, where $p = u_1u_2 \cdots u_n$ and each u_i is either x or an element of Σ. For each $i = 0, 1,..., n$ let $q_i^j = \delta_j(q_0, u_1u_2 \cdots u_i)$ for $j = 1, 2$. Then we show by induction that $q_i^1 = q_i^2$ for $i = 1, 2,... n$, since $q_0^1 = q_0^2 = q_0$ and $\delta_j(q_{i-1}, u_i)$ depends only on the value of q_{i-1} and whether u_i is x or an element of Σ. Thus $\delta_3(q_0, p)$ is defined and equal to $\delta_1(q_0, p) = \delta_2(q_0, p)$. Since $\delta_1(q_0, p) \in F_1$ and $\delta_2(q_0, p) \in F_2$, $\delta_3(q_0, p) \in F_3$, so $p \in L(A_1 \cap A_2)$. ▌

The point of this theorem is that our "redundant" representation of sets of patterns makes the problem of intersection easier. Recall that in general if M_1, M_2, M_3 are finite automata with m_1, m_2, m_3 states, respectively, and $L(M_3) = L(M_1) \cap L(M_2)$, then we may have $m_3 = m_1 \cdot m_2$; in our special case we have $m_3 \leqslant \min\{m_1, m_2\}$, and M_3 may be constructed in time proportional to the sum of the sizes of M_1 and M_2. The following example shows that the quadratic growth ($A(s; t)$ may have $\Omega(| s |^2)$ states) is in general necessitated by intersection.

EXAMPLE 5.3. Let $s_1 = (01)^{3n}0$, $t_1 = 010$, $s_2 = (01)^{2n}0$, $t_2 = 0$. Consider the set $R = P(s_1; t_1) \cap P(s_2; t_2)$. R consists of all strings from $((0 + x)1)^{2n}(0 + x)$ that contain exactly n occurrences of x. It is not difficult to see that any finite automaton to recognize R must contain at least $2n^2$ states.

k-Variable Pattern Automata

Let $k \geqslant 1$. Let $s, t_1, t_2,..., t_k \in \Sigma^+$, where each t_i is a substring of s. Define $P(s; t_1, t_2,..., t_k)$ to be the set of k-variable patterns p in canonical form such that $p[t_1/x_1,..., t_k/x_k] = s$. We may construct an automaton $A(s; t_1, t_2,..., t_k)$ to recognize

$P(s; t_1, t_2, \ldots, t_k)$ in a manner analogous to $A(s; t)$. States are $(k+1)$-tuples; $(i, j_1, j_2, \ldots, j_k)$ records the fact that i constants and j_r occurrences of x_r have been read in so far, for $r = 1, 2, \ldots, k$. A transition on $b \in \Sigma$ (or on the symbol x_m) depends on an occurrence of b (or t_m) at position $1 + i + \sum_{r=1}^{k} j_r \cdot | t_r |$ of s, and increments the i (or j_m) component of the state. Further details are left to the reader; the definition of a k-variable pattern automaton and Theorems 5.1 and 5.2 generalize straightforwardly.

6. Finding Descriptive One-Variable Patterns

In this section we restrict our attention to the class of one-variable patterns, which is denoted P_1. A pattern $p \in P_1$ is said to be *descriptive of* the sample S *within* P_1 provided $S \subseteq L(p)$ and for all $q \in P_1$, if $S \subseteq L(q)$ then $L(q)$ is not a proper subset of $L(p)$. We describe a polynomial-time algorithm to solve the following problem: given a sample S, find a one-variable pattern p that is descriptive of S within P_1. The algorithm uses the one-variable pattern automata described in the preceding section to construct a particular representation of all the one-variable patterns that generate S, and then selects an appropriate one to output.

For any one-variable pattern p, define $\tau(p)$ to be the triple of nonnegative integers (i, j, k) such that the number of occurrences of constants in p is i, the number of occurrences of variables in p is j, and the position of the leftmost occurrence of x in p is k. The algorithm partitions one-variable patterns p according to the value of $\tau(p)$.

Let an input sample $S = \{s_1, s_2, \ldots, s_m\}$ be given, where each $s_i \in \Sigma^+$ and $m \geqslant 2$. For each $r = 1, 2, \ldots, m$ let $P_1(s_r)$ denote the set of one-variable patterns p such that $s_r \in L(p)$. Fix r, $1 \leqslant r \leqslant m$. A triple (i, j, k) of non-negative integers is said to be *feasible for* s_r provided $0 \leqslant i < | s_r |$, $1 \leqslant j \leqslant | s_r |$, $1 \leqslant k \leqslant i + 1$, and $j \mid (| s_r | - i)$. The set of all triples feasible for s_r is denoted F_r. For each $(i, j, k) \in F_r$, we define a particular one-variable pattern automaton $B_r(i, j, k)$ as follows. Let t be the unique substring of s_r beginning at position k and of length $l = (| s_r | - i)/j$. To obtain $B_r(i, j, k)$, take $A(s_r; t)$ and remove any x transition leaving from a state $(u, 0)$ in which $u < k - 1$, remove the constant transition leaving from state $(0, k - 1)$, and remove all final states except (i, j).

Lemma 6.1. *For each* $(i, j, k) \in F_r$, $L(B_r(i, j, k))$ *is precisely those* $p \in P_1(s_r)$ *such that* $\tau(p) = (i, j, k)$. *Consequently,*

$$\bigcup_{(i,j,k) \in F_r} L(B_r(i, j, k)) = P_1(s_r).$$

Proof. Let $(i, j, k) \in F_r$. If t is the unique substing of s_r of length $l = (| s_r | - i)/j$ beginning at position k, then clearly $L(A(s_r; t))$ contains all $p \in P_1(s_r)$ such that $\tau(p) = (i, j, k)$. Furthermore, the modifications made to $A(s_r; t)$ to obtain $B_r(i, j, k)$ guarantee that the first $k - 1$ symbols of any pattern $p \in L(B_r(i, j, k))$ are constant symbols, the kth symbol is x, and p contains i occurrences of constant symbols and j occurrences of x, i.e., $\tau(p) = (i, j, k)$. Since if $p \in P_1(s_r)$, $\tau(p)$ must be a triple that is feasible for s_r, the second assertion follows. ▌

LEMMA 6.2. *Let $F = \bigcap_{r=1}^{m} F_r$ and for each $(i, j, k) \in F$ let $B(i, j, k) = \bigcap_{r=1}^{m} B_r(i, j, k)$. Then*

$$\bigcap_{r=1}^{m} P_1(s_r) = \bigcup_{(i,j,k)\in F} L(B(i, j, k)).$$

Proof. Clearly $p \neq p'$ if $\tau(p) \neq \tau(p')$, so if $(i, j, k) \neq (i', j', k')$ then $L(B_r(i, j, k))$ and $L(B_u(i', j', k'))$ must be disjoint, for any $1 \leqslant r, u \leqslant m$. Hence, using Lemma 6.1,

$$\bigcap_{r=1}^{m} P_1(s_r) = \bigcap_{r=1}^{m} \bigcup_{(i,j,k)\in F_r} L(B_r(i, j, k))$$

m

$$= \bigcup_{(i,j,k)\in F} \bigcap_{r=1}^{m} L(B_r(i, j, k)).$$

By the Intersection Theorem, Theorem 5.2, $\bigcap_{r=1}^{m} L(B_r(i, j, k)) = L(\bigcap_{r=1}^{m} B_r(i, j, k)) = L(B(i, j, k))$, and the theorem follows. ▌

The set of one-variable patterns that generate S is precisely $\bigcap_{r=1}^{m} P_1(s_r)$, so the automata $B(i, j, k)$ for $(i, j, k) \in F$ are the representation we seek. The algorithm to solve the problem of finding descriptive one-variable patterns may now be described as follows. On input $S = \{s_1, s_2, \ldots, s_m\}$, $m \geqslant 2$, enumerate the set of *feasible triples* F, and for each $(i, j, k) \in F$, construct the automaton $B(i, j, k)$. Among all (i, j, k) in F such that $L(B(i, j, k)) \neq \varnothing$, select (i_0, j_0, k_0) to maximize $i + j$. Select and output any pattern $p \in L(B(i_0, j_0, k_0))$.

LEMMA 6.3. *The pattern p output by this algorithm is descriptive of S within P_1.*

Proof. Suppose to the contrary that q is a one-variable pattern with variable x, $S \subseteq L(q)$, and $L(q) \subsetneq L(p)$. Then by Lemma 3.9, $q \leqslant' p$, and consequently $|q| \geqslant |p|$. By Lemma 6.2, some $B(i, j, k)$ accepts q, and $|q| = i + j$. Since p is chosen to maximize $|p| = i_0 + j_0$, $|p| \geqslant |q|$. Hence we have $q \leqslant' p$ and $|p| = |q|$. Thus q is obtained from p by substituting either a variable or a constant for x in p, so q is either a constant string or equivalent to p, i.e., either $|L(q)| = 1$ or $L(q) = L(p)$, contradicting the assumption that $S \subseteq L(q)$ and $L(q) \subsetneq L(p)$. (Recall that $|S| \geqslant 2$.) ▌

Thus the algorithm we have described correctly finds a one-variable pattern descriptive of S within P_1. We now show that this algortihm may be implemented to run in polynomial time.

The model of computation we assume is a random-access computer [3] that performs a reasonable menu of operations each in unit time on registers of length $O(\log n)$ bits, where n is the input length. For each feasible triple $(i, j, k) \in F$ and each $r = 1, 2, \ldots, m$, we construct $B_r(i, j, k)$. This machine has $O(|s_r|^2)$ states and may easily be constructed in time proportional to its size. (Once we extract the substring t of length $(|s_r| - i)/j$ and starting at position k in s_r, we may locate all occurrences of t as a substring in s_r in time $O(|s_r|)$ using the linear pattern-matching algorithm of Knuth, Morris, and Pratt [8].

From this is is not difficult to construct $B_r(i, j, k)$.) To find $B(i, j, k)$ we must intersect the machines $B_1(i, j, k), \ldots, B_m(i, j, k)$. This may be done in time proportional to the sum of their sizes, so the whole computation of $B(i, j, k)$ may be done in time $O(M)$, where $M = \sum_{r=1}^{m} | s_r |^2$. We may bound the number of elements of F as follows. Let $l = \min\{| s_r |: 1 \leqslant r \leqslant m\}$. Let $G = \{(i, j): 1 \leqslant i + 1, j \leqslant l \text{ and } j \mid (l - i)\}$. Clearly $F \subseteq G \times \{1, 2, \ldots, l\}$. If $d(n)$ denotes the number of divisors of the integer n, then $| G | = \sum_{n=1}^{l} d(n)$, so by a theorem of Dirichlet on the average order of $d(n)$ [6, Theorem 320], we have $| G | = O(l \log l)$. This gives a bound of $O(l^2 \log l)$ for the cardinality of F. Clearly the time to enumerate the elements of F and to extract the final answer from the automata $B(i, j, k)$ will be dominated by the time to construct these automata, so we obtain the following.

Lemma 6.4. *The algorithm described above may be implemented to run in time $O(Ml^2 \log l)$ and space $O(M)$, where $M = \sum_{r=1}^{m} | s_r |^2$ and $l = \min\{| s_r |: 1 \leqslant r \leqslant m\}$. If $n = \sum_{r=1}^{m} | s_r |$, then the time is also $O(n^4 \log n)$ and the space is $O(n^2)$.*

We observe that the algorithm has good "incremental" behavior. Suppose for the sample $S = \{s_1, s_2, \ldots, s_m\}$ we have computed the set $\mathscr{B} = \{B(i, j, k): (i, j, k) \in F$ and $L(B(i, j, k)) \neq \varnothing\}$. If S is now augmented by a new string, say s_{m+1}, we may update $\mathscr{B}$ by finding each $B(i, j, k) \in \mathscr{B}$ such that (i, j, k) is feasible for s_{m+1}, constructing $B_{m+1}(i, j, k)$, and intersecting it with $B(i, j, k)$. If the resulting machine accepts at least one string, it is retained in $\mathscr{B}$, otherwise it is discarded. The time to update $\mathscr{B}$ is proportional to the maximum of the space used to represent $\mathscr{B}$ and $| s_{m+1} |^2$.

From Lemmas 6.3 and 6.4 we obtain

Theorem 6.5. *There exists a polynomial-time algorithm which given a sample S will output a one-variable pattern p that is descriptive of S within P_1.*

An Example

We next give an example to illustrate the operation of the algorithm. Let $S = \{1100, 110000, 110101\}$. There are fourteen feasible triples (i, j, k); we consider the two possible cases for j separately.

(a) $j = 1$. This is the case of a single occurrence of the variable. Each triple gives rise to just one pattern per string. (In practice it is probably more reasonable to treat $j = 1$ as a special case.) In this case there are ten triples:

	triple	1111	110000	110101	intersection
1.	(0, 1, 1)	x	x	x	x
2.	(1, 1, 1)	$x1$	$x0$	$x1$	—
3.	(1, 1, 2)	$1x$	$1x$	$1x$	$1x$
4.	(2, 1, 1)	$x11$	$x00$	$x01$	—
5.	(2, 1, 2)	$1x1$	$1x0$	$1x1$	—

	triple	1111	110000	110101	intersection
6.	(2, 1, 3)	$11x$	$11x$	$11x$	$11x$
7.	(3, 1, 1)	$x111$	$x000$	$x101$	—
8.	(3, 1, 2)	$1x11$	$1x00$	$1x01$	—
9.	(3, 1, 3)	$11x1$	$11x0$	$11x1$	—
10.	(3, 1, 4)	$111x$	$110x$	$110x$	—

(b) $j > 1$. The restrictions $j \mid (4 - i)$ and $j \mid (6 - i)$ yield $j = 2$ and $i = 0, 2$ as the only possibilities. This gives four more feasible triples. (Here, as above, we show $L(B_r(i, j, k))$ in place of $B_r(i, j, k)$ for perspicuity.)

	triple	1111	110000	110101	intersection
11.	(0, 2, 1)	$\{xx\}$	—	—	—
12.	(2, 2, 1)	$\{xx11, x1x1, x11x\}$	—	—	—
13.	(2, 2, 2)	$\{1xx1, 1x1x\}$	—	$\{1xx1\}$	—
14.	(2, 2, 3)	$\{11xx\}$	$\{11xx\}$	$\{11xx\}$	$\{11xx\}$

The selection procedure then prefers the (in this case unique) longest pattern: $11xx$.

7. Remarks

Several natural extensions of the notion of "pattern" spring to mind. One that seems to be possible to accommodate with only minor modifications in the foregoing approach is allowing reversal of variables. That is, we consider an auxiliary set of variables $\{x_1{}^r, x_2{}^r, \ldots\}$ and require that every substitution f have the property that $f(x_i{}^r)$ be the reversal of $f(x_i)$, for all i. For example, 10201, 1242421 $\in L(x2x^r)$. For the one-variable case, the fact that a string and its reversal have the same length preserves the reasoning about lengths. A similar approach should work for more obscure length-preserving operations, e.g., changing every letter of a string to 0. Another natural operation to consider is Kleene star; introduce variables $\{x_1^*, x_2^*, \ldots\}$ such that every substitution f must map $f(x_i^*)$ into $\{f(x_i)\}^*$ for every i. For example, 1021010, 323333 $\in L(x2x^*)$. This kind of operation, being not length-preserving, does not fit into the foregoing scheme of things, and seems to require a new approach. Alternation, or finite union, also seems like a reasonable candidate, but this would preclude the possibility of correct inference in the limit from positive data [2], and would require a completely different definition of what pattern should be produced for a given finite sample. (See [4] for another way of formalizing inference problems.)

The algorithm for finding a descriptive one-variable pattern in polynomial time may be partly generalized to the case of k-variable patterns, for $k > 1$, using k-variable pattern automata. The difficulty arises because there is no analogue of the result for one-variable patterns showing that a triple (i, j, k) feasible for a string s uniquely determines the corresponding substring t of s and allows us to construct a single pattern automaton to recognize all patterns p such that $s \in L(p)$ and $\tau(p) = (i, j, k)$. In particular, for $k > 1$ there may be two distinct substitutions yielding the same string from a given pattern, e.g., $0010012 = xxy[0/x, 10012/y] = xxy[001/x, 2/y]$. Thus, to represent the set of patterns with any given collection of characteristics, we must in general use a *set* of k-variable pattern automata, and the bounding arguments no longer apply. It would be of interest to know whether for $k = 2, 3, \ldots$, there is a polynomial-time algorithm to find a k-variable pattern descriptive of a given sample within P_k. It would also be of interest to know whether Theorem 4.2 could be strengthened so that the requirement of finding a *longest* descriptive pattern could be dropped.

Considering certain special cases of the problem solved by the algorithm to find descriptive one-variable patterns, we have been able to improve the running time (for example, the case in which $|s_r| = |s_u|$ for some $1 \leqslant r \neq u \leqslant m$ allows a much more direct computation). We hope that further study will provide insight for a uniform improvement in the result.

We note again that the question of whether it is in general effectively decidable whether $L(p) \subseteq L(q)$ given patterns p and q is to our knowledge open. This necessitates a few detours in the foregoing results; it would be of interest to know whether they are essential. Results on equations in free semigroups may prove relevant in this connection [9].

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, Mass., 1974.
2. D. Angluin, Inductive inference of formal languages from positive data, *Inform. Contr.*, in press.
3. D. Angluin and L. G. Valiant, Fast probabilistic algorithms for Hamiltonian circuits and matchings, *J. Comput. System Sci.* **18** (1979), 155–193.
4. E. M. Gold, Complexity of automaton identification from given data, *Inform. Contr.* **37** (1978), 302–320.
5. R. E. Griswold, J. F. Poage, and I. P. Polonsky, "The SNOBOL4 Programming Language," Prentice-Hall, Englewood Cliffs, N.J., 1972.
6. G. H. Hardy and E. M. Wright, "An Introduction to the Theory of Numbers," Oxford Univ. Press, London, 1968.
7. J. E. Hopcroft and J. D. Ullman, "Formal Languages and Their Relation to Automata," Addison-Wesley, Reading, Mass., 1969.
8. D. E. Knuth, J. H. Morris, and V. R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.* **6** (1977), 323–350.
9. G. S. Makanin, The problem of solvability of equations in a free semigroup, *Soviet Math. Dokl.* **18** (1977), 330–334.