

Copyright © 1976, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

AN APPLICATION OF THE THEORY OF COMPUTATIONAL COMPLEXITY
TO THE STUDY OF INDUCTIVE INFERENCE

by

Dana Angluin

Memorandum No. ERL-M586

17 March 1976

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TABLE OF CONTENTS

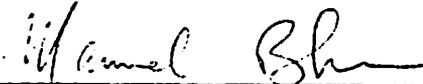
	<u>Page</u>
CHAPTER 1 INTRODUCTORY	
1.1 Introduction and the Basic Definition	1
1.2 Size of Descriptions in Different Representations, An Example	2
CHAPTER 2 FINDING A SMALLEST DETERMINISTIC AUTOMATON COMPATIBLE WITH GIVEN DATA	
2.1 Introduction and Example	8
2.2 Definitions, Notation	9
2.3 The Complexity of Finding Good-Guesses	11
2.4 Uniform-Complete Samples in Polynomial Time	12
2.5 A Further Extension of the Uniform-Complete Case Which is NP-hard	20
CHAPTER 3 FINDING A SMALLEST REGULAR EXPRESSION COMPATIBLE WITH GIVEN DATA	
3.1 Introduction and Example	32
3.2 Definitions, Notation, Conventions	33
3.3 Some Efficiently-Inferable Classes	37
3.4 Some Classes Which are Hard-to-infer	44
3.5 Inference of "Most Likely" Expressions	57
CHAPTER 4 SUMMARY AND ACKNOWLEDGMENTS	65
APPENDIX	67
REFERENCES	73

AN APPLICATION OF THE THEORY OF COMPUTATIONAL COMPLEXITY
TO THE STUDY OF INDUCTIVE INFERENCE

Ph.D. in
Engineering Science

Dana Angluin

Electrical Engineering
and Computer Sciences



Professor M. Blum
Chairman of Committee

ABSTRACT

We study the computational tractability of finding a deterministic finite automaton with a minimum number of states or a regular expression of minimum length compatible with given positive and negative samples of an unknown finite-state language. Some restrictions of these problems are shown to be efficiently solvable, others not.

Given a sample consisting of a finite set of strings in and a finite set of strings out of an unknown finite-state language, we define a "good guess" to be a smallest finite-state description which is compatible with the given information. (What language is guessed will in general depend on the descriptive system chosen for the guesses.)

Mark Gold [1974] has shown that finding a deterministic automaton with the minimum number of states compatible with an arbitrary sample is an NP-hard problem; that is, there will be no polynomial time algorithm for the problem if $P \neq NP$. We consider the problem restricted to "uniform-complete" samples -- samples which for some $k \geq 1$ contain all input strings of length $\leq k$ and no others -- and demonstrate a simple algorithm to find a smallest compatible deterministic automaton in polynomial time for such samples. Then we show

that for any $\epsilon > 0$, permitting $O(n^\epsilon)$ strings in an otherwise uniform-complete sample of total length n makes the problem again NP-hard. The construction given also shows that the problem remains NP-hard when restricted to "finite-language" or "definite" automata.

We also consider finding a regular expression of minimum length compatible with an arbitrary sample. We show that this problem is NP-hard in general, as are some variants using restricted sets of operators. We exhibit some still more restricted, syntactically-given sets of regular expressions in which minimum compatible expressions may be found in polynomial time for arbitrary samples. Finally, we define a "most likely" expression compatible with an arbitrary sample, and show that finding such an expression from given data is also an NP-hard problem.

CHAPTER 1

Introductory

1.1 Introduction and the Basic Definition

The work presented herein is a contribution to the study of "inductive inference", which is the process of constructing a general rule or hypothesis from a finite number of examples. This process has been of interest to philosophers and mathematicians since at least the time of Aristotle. The development in this century of the theory of computability permits the formulation and solution of some formerly inaccessible problems in inductive inference; see Gold [1967] and Blum & Blum [1975]. Some other papers in inductive inference are listed in the bibliography of this dissertation.

The particular focus of this work is the inference of a description of a finite-state language from a sample consisting of a finite set of positive examples and a finite set of negative examples. As an example, suppose that $L \subseteq \{0,1\}^*$ is a finite-state language which is unknown to us. If we are given the following sample of L :

<u>in L</u>	<u>not in L</u>
100	000
1010101	1101
1000	10001
1	0
101	1010

we might guess that L consists of all strings which begin with "1", followed either by zero or more repetitions of "0" or by zero or more repetitions of "01", in other words, that L is described by the regular expression: $1(0^* \vee (01)^*)$.

If we are given another sample, containing further examples of strings in L and not in L , we might find that our guess is incorrect. However, we can never conclude from any finite sample that our guess is a correct identification of L . Even so, we might want to define the notion of a "reasonable" or "good" guess for a given sample. One possible definition, answering to the demand for economy of description, is:

The Basic Definition: Given a sample S consisting of a pair of finite sets of strings, $S = \langle S_1, S_0 \rangle$, a good-guess for S is a smallest finite-state description D such that the language L denoted by D contains all of the strings in S_1 and excludes all of the strings in S_0 .

Clearly, we have to specify what we mean by a "finite-state description" and a "smallest" such in order for this definition to make sense.

The remaining section of this chapter presents an example showing that the languages inferred under this definition will in general depend upon the system of description chosen. Chapters 2 and 3 are devoted to a study of the computational tractability of finding good-guesses in classes of deterministic finite automata and regular expressions, respectively. Each chapter was written to be readable apart from the others; therefore some (hopefully not much) of the introductory material in each chapter is repeated in other chapters.

1.2 Size of Descriptions in Different Representations, An Example

There are a number of different formalisms available for denoting finite-state languages: deterministic, nondeterministic, or two-way automata, and regular expressions, among others. Meyer and Fischer

[1971] discuss "economy of description" in different systems of representation for finite-state languages, and present a number of examples of languages for which smallest representations are of greatly disparate sizes in two different formalisms. These results tend to suggest (correctly) that the system of description chosen will affect what languages are denoted by good-guesses for a given sample.

We will give one example illustrating this phenomenon, for deterministic and nondeterministic finite automata. First, some definitions:

Definition: Let X be a finite set. $|X|$ will denote the cardinality of X . If X is nonempty, X^* will denote the set of all finite strings of elements of X , and $u \cdot v$ will denote the concatenation of two elements $u, v \in X^*$. Λ will denote the null string in X^* . X^+ will denote the set of all non-null strings in X^* .

Definition: Fix two nonempty finite sets, X and Y , the input and output alphabets, respectively. A nondeterministic finite automaton, M , will be a quadruple $\langle Q, \delta, \lambda, I \rangle$ where Q is a nonempty finite set, the set of states of M , $I \subseteq Q$ is nonempty (the set of initial states of M), δ is a map from $Q \times X$ into the set of all subsets of Q , and λ is a map from $Q \times X$ into Y . We extend δ to a map $\hat{\delta}$ from $Q \times X^*$ into the set of all subsets of Q as follows:

$$\begin{aligned} \hat{\delta}(q, \Lambda) &= \{q\} && \text{for all } q \in Q \\ \hat{\delta}(q, u \cdot a) &= \bigcup_{q' \in \hat{\delta}(q, u)} \delta(q', a) && \text{for all } q \in Q, a \in X, u \in X^* . \end{aligned}$$

Definition: If M is a nondeterministic finite automaton and the output alphabet Y is the set $\{0,1\}$, then we define the language

accepted by M , denoted $L(M)$, by

$$L(M) = \{s \in X^+ \mid s = t \cdot a \text{ for some } t \in X^* \text{ and } a \in X, \\ \text{and } \lambda(q, a) = 1 \text{ for some } q \in \bigcup_{q' \in I} \delta(q', t)\} .$$

Definition: If M is a nondeterministic finite automaton, then M will be called deterministic if and only if $|I| = 1$, and $|\delta(q, a)| \leq 1$ for every $q \in Q$ and $a \in X$. (Sometimes deterministic is taken to require $|\delta(q, a)| = 1$, so this definition is a bit non-standard.)

Definition: If M is a nondeterministic finite automaton, the size of M , denoted $\text{size}(M)$, will be the cardinality of its state-set, i.e., $|Q|$.

We will now present a sample which distinguishes good-guesses in the class of all nondeterministic finite automata from good-guesses in the class of deterministic finite automata. Fix $X = \{0, 1, a\}$ and $Y = \{0, 1\}$. Let $S = \langle S_1, S_0 \rangle$ where S_1 and S_0 are given as follows:

S_1	S_0
0	1
11	a
10a	10
a1	1a
aa	100
1a0	101
1aa	a11

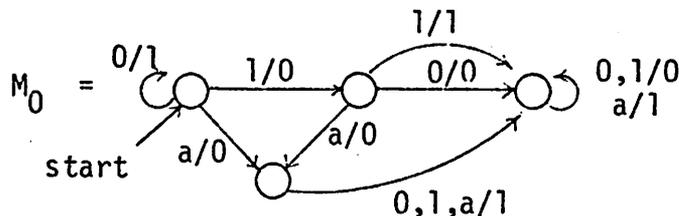
We will say that a nondeterministic finite automaton M is compatible with S provided that $S_1 \subseteq L(M)$ and $S_0 \subseteq X^* - L(M)$.

Proposition 1: If M is any deterministic finite automaton of minimum size which is compatible with S , and N is any nondeterministic finite automaton of minimum size which is compatible with S , then

$$L(M) \neq L(N).$$

Proof: Let M and N be deterministic and nondeterministic (respectively) automata of minimum size which are compatible with S . We will show that $a0 \in L(M)$ and $a0 \notin L(N)$, and conclude that $L(M) \neq L(N)$.

1. $\text{size}(M) \leq 4$. To see this, note that



is a deterministic finite automaton which is compatible with S , so the minimum size of any deterministic finite automaton compatible with S does not exceed 4 states.

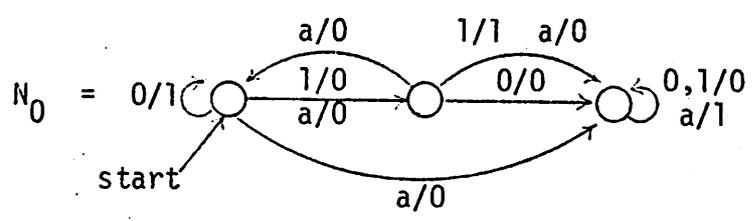
2. $\text{size}(M) \geq 4$. Let $M = \langle Q, \delta, \lambda, \{q_0\} \rangle$. If $\hat{\delta}(q_0, s)$ is non-empty, we will denote the unique state that it contains by $[s]$, for each $s \in X^*$. Since $0, 11, 10a, a1$ must all be in $L(M)$, we know that the sets $\hat{\delta}(q_0, \Lambda)$, $\hat{\delta}(q_0, 1)$, $\hat{\delta}(q_0, 10)$, and $\hat{\delta}(q_0, a)$ are nonempty. We have then the following information about $\lambda(q, x)$:

$q \backslash x$	0	1	a
$[A]$	1	0	0
$[1]$	0	1	0
$[10]$	0	0	1
$[a]$?	1	1

which shows that the four states $[\Lambda]$, $[1]$, $[10]$, $[a]$ are indeed distinct states of M , so $\text{size}(M) \geq 4$.

3. $a0 \in L(M)$. $\hat{\delta}(q_0, 1a)$ must be nonempty, since $1a0 \in L(M)$. But we must have $\lambda([1a], 0) = 1$ and $\lambda([1a], a) = 1$, so $[1a] \neq [\Lambda]$, $[1]$, $[10]$. By #1 and #2, $\text{size}(M) = 4$, and therefore $[1a] = [a]$. But then $\lambda([a], 0) = \lambda([1a], 0) = 1$, so $a0 \in L(M)$, as claimed.

4. $\text{size}(N) \leq 3$. Consider the machine



which is a nondeterministic finite automaton compatible with S . Thus the minimum size of a nondeterministic automaton which is compatible with S is at most 3 states.

5. $\text{size}(N) \geq 3$. Let $N = \langle Q, \delta, \lambda, I \rangle$. The set of initial states, I , may contain more than one element, but it must contain at least one state, say q_Λ , such that $\lambda(q_\Lambda, 0) = 1$, because $0 \in L(N)$. Since $1, a \notin L(N)$, we must also have $\lambda(q_\Lambda, 1) = 0$ and $\lambda(q_\Lambda, a) = 0$. Similarly, there must be a state, say q_1 , in $\bigcup_{q \in I} \hat{\delta}(q, 1)$ such that $\lambda(q_1, 1) = 1$, because $11 \in L(N)$. Also, $\lambda(q_1, 0) = \lambda(q_1, a) = 0$ because $10, 1a \notin L(N)$. Finally, there must be a state, say q_{10} , in $\bigcup_{q \in I} \hat{\delta}(q, 10)$ such that $\lambda(q_{10}, a) = 1$, and $\lambda(q_{10}, 0) = \lambda(q_{10}, 1) = 0$.

We summarize this information in a table for $\lambda(q, x)$:

q	x	0	1	a
q _Λ	1	0	0	0
q ₁	0	1	0	0
q ₁₀	0	0	0	1

and conclude that the states q_Λ , q_1 , and q_{10} are distinct states of N , so $\text{size}(N) \geq 3$, as claimed.

6. $a0 \notin L(N)$. By #4 and #5, the only states in N are the states q_Λ , q_1 , and q_{10} . Since $1, a \notin L(N)$, $q_1, q_{10} \notin I$, so I must be the singleton $\{q_\Lambda\}$. Because $q_1 \in \bigcup_{q \in I} \hat{\delta}(q, 1)$, we must have $q_1 \in \hat{\delta}(q_\Lambda, 1)$. Now assume contrary to #6 that $a0 \in L(N)$. This implies that $q_\Lambda \in \hat{\delta}(q_\Lambda, a)$. But if $q_\Lambda \in \hat{\delta}(q_\Lambda, a)$, then $q_1 \in \hat{\delta}(q_\Lambda, a1)$, so $a1 \in L(N)$, contradicting the fact that N is compatible with the sample S , and $a1 \in S_0$. Thus we conclude that $a0 \notin L(N)$, as claimed.

Since $a0 \in L(M)$, and $a0 \notin L(N)$, we have $L(M) \neq L(N)$. \square

Discussion

It is not difficult to see that any nondeterministic automaton, N , with 2 states is equivalent to some deterministic finite automaton with at most 3 states under the given definition of deterministic automaton. Thus, there will be no example of the phenomenon exhibited in Proposition 1 in which N contains only 2 states.

It would perhaps be of interest to study in more detail the effects of different systems of representation on the inferences made according to the Basic Definition. The remainder of the present work however is devoted to investigating the computational difficulty of finding such inferences for regular languages in two particular systems of representation: deterministic finite automata, and regular expressions. Related results concerning the difficulty of finding minimum-sized representations for Boolean functions compatible with given data may be found in Angluin [1976].

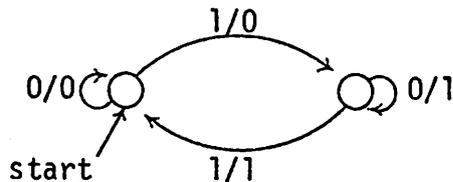
CHAPTER 2

Finding a Smallest Deterministic Automaton Compatible with Given Data2.1 Introduction and Example

In this chapter we are concerned with the problem of inferring a deterministic finite-state machine from a given sample of its behaviour. For example, given that:

Example 1: on input: 00010000
 M's response: 00001111
 on input: 1000001000
 M's response: 0111111000

we might guess that M is the machine:



Without a bound on the number of states in M, we can never be certain of identifying M correctly. One "reasonable" strategy would be to guess a machine with the minimum number of states which is consistent with the given sample; this answers to the notion of "economy of description". This strategy also leads to a correct inference "in the limit". That is, if we are presented with successively larger samples of the behaviour of an unknown machine, guaranteed to contain its response for every possible input string eventually, and we consistently guess a smallest machine in agreement with the given sample, we will eventually hit on the smallest machine equivalent to the unknown machine, having eliminated all candidates of the same size or smaller as inconsistent with the sample. (Work on inference in the

limit may be found in Gold [1967], Feldman [1972], and Blum & Blum [1975].) Other strategies also give correct inference in the limit, for example, Gold's polynomial time "padding" method (Gold [1974]).

In this chapter, we will investigate the computational problem of finding a smallest deterministic machine in agreement with given data.

2.2 Definitions, Notation

Definition: Let X be a finite set. $|X|$ will denote the cardinality of X . If X is nonempty, then X^* will denote the set of all finite strings of elements of X . Λ will denote the null string in X^* . X^+ will denote the set $X^* - \{\Lambda\}$. If $s, t \in X^*$, then $s \cdot t$ or simply st will denote the concatenation of s and t , and $|s|$ will denote the length of s .

We fix U and V , nonempty finite sets of symbols, the input and output alphabets, respectively.

Definition: A sample-function will be any function from a finite nonempty subset of U^+ into V .

Definition: A sample-function f will be called prefix-closed if for every u in the domain of f , if v is a nonnull prefix of u , then v is in the domain of f .

We will consider only prefix-closed sample-functions in the remainder of this chapter.

Definition: A sample S will be any finite nonempty subset of $U^+ \times V$ such that

1. $|u| = |v|$ for every pair $\langle u, v \rangle \in S$
2. there exists a sample-function f such that for any

$\langle a_1 a_2 \cdots a_k, b_1 b_2 \cdots b_k \rangle \in S$ ($k \geq 1$, $a_1, a_2, \dots, a_k \in U$ and $b_1, b_2, \dots, b_k \in V$) we have

$$f(a_1 a_2 \cdots a_i) = b_i \quad \text{for all } i = 1, 2, \dots, k.$$

The unique smallest such f will be called the sample-function presented by S .

Example 2: $S_1 = \{\langle 0101, 0011 \rangle, \langle 010011, 001110 \rangle\}$

is a sample presenting the sample-function:

$$f_1 = \{\langle 0, 0 \rangle, \langle 01, 0 \rangle, \langle 010, 1 \rangle, \langle 0101, 1 \rangle, \langle 0100, 1 \rangle, \langle 01001, 1 \rangle, \langle 010011, 0 \rangle\}$$

while

$$S_2 = \{\langle 010, 111 \rangle, \langle 0111, 1000 \rangle\}$$

is not a sample, since no function f can have both $f(01) = 1$ and $f(01) = 0$.

It is clear that a finite subset of $U^+ \times V$ can be tested to determine whether it is a sample in polynomial time.

Definition: A machine will be a fully-specified, deterministic, Mealy-model finite automaton with input alphabet U , output alphabet V , and a unique start-state. If M is a machine, and $u \in U^+$, we denote by $[u]_M$, or just $[u]$, the state that M is in after being started in its start-state and given the input u . If s is any state of M , $M(s, u)$ will denote the output string produced by M when started in state s and given input u . $M'(s, u)$ will denote the last symbol of $M(s, u)$. $M(u)$ and $M'(u)$ will be used as abbreviations for $M([A], u)$ and $M'([A], u)$ respectively.

Definition: A machine M is consistent with, or agrees with, a sample S if and only if $M(u) = f(u)$ for every u in the domain of f , where f is the sample-function presented by S .

Definition: If S is a sample, a good-guess for S will be any machine M with the minimum possible number of states which is consistent with S .

The machine of Example 1 is a good-guess for the sample given there, and any other good-guess for that sample must be equivalent to it. In general, however, there will be inequivalent good-guesses for a given sample.

2.3 The Complexity of Finding Good-Guesses

Gold [1974] has shown that finding a good-guess from a given sample is a computationally infeasible problem in general (provided $P \neq NP$). (We assume familiarity with the definitions and results concerning deterministic and non-deterministic polynomial-time-bounded computations, as developed in Cook [1971] and Karp [1972].) That is:

Theorem 1 (Gold): The problem of deciding, for a given sample S (which may be taken to be prefix-complete) and positive integer k , whether there exists a machine with $\leq k$ states which is consistent with S , is NP-complete, provided $|U|, |V| \geq 2$.

This may be interpreted as saying that enumerative techniques for finding a good-guess from a given sample are essentially the best we can expect at present, and no algorithm for the problem will work uniformly in polynomial time if $P \neq NP$. From a practical point of view, we might hope to find reasonable restrictions on the type of machine

to be inferred, or on the sample, which would permit us to find good-guesses more efficiently. In the case that the sample specifies an output for every input string of length $\leq k$ and no others, for some $k \geq 1$, we will see that there is a simple algorithm for finding a good-guess in polynomial time. We will then consider the effects of relaxing this condition on the sample.

2.4 Uniform-Complete Samples in Polynomial Time

In this section, we consider a rather strong condition on the sample, and show that good-guesses may be found in polynomial time for samples which satisfy this condition.

Definition: A sample S will be called k -uniform-complete if and only if the sample-function it presents is defined for precisely the set $U^k = \{u \mid u \in U^+ \text{ and } |u| \leq k\}$ for some integer $k \geq 1$.

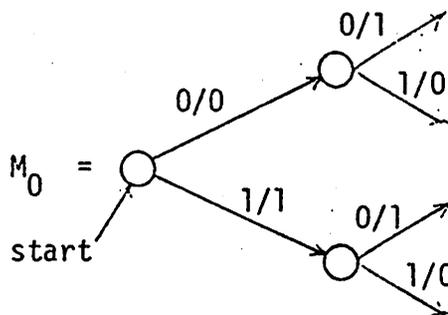
Thus a k -uniform-complete sample specifies input/output behaviour for all input strings of length $\leq k$ and no others, and consequently will be of size $O(k|U|^k)$.

Example 3: $S_0 = \{\langle 00,01 \rangle, \langle 01,00 \rangle, \langle 10,11 \rangle, \langle 11,10 \rangle\}$

is a 2-uniform-complete sample, if $U = \{0,1\}$.

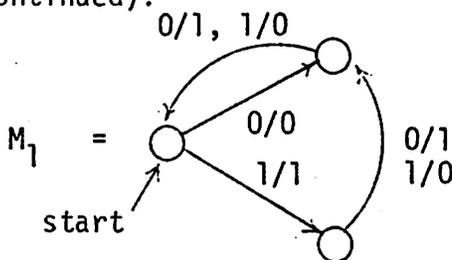
We may construct a partially-specified "tree machine" of $|U|^k - 1$ states in agreement with a k -uniform-complete sample, by assigning a state for every string of length $< k$.

Example 3 (continued): For S_0 we get the partially-specified machine



We might arbitrarily assign the unspecified transitions in the tree machine for S and find the minimal equivalent to the resulting fully-specified machine, e.g.,

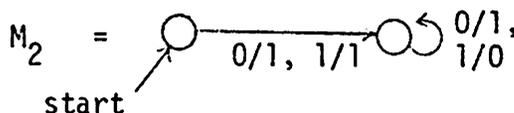
Example 3 (continued):



(M_1 is minimal because $M_1([\Lambda], 00) = 01$, $M_1([0], 00) = 10$, and $M_1([1], 00) = 11$.)

This procedure will not in general produce a good-guess for the sample S , as may be seen from the fact that M_2 is also in agreement with the S_0 of Example 3, where

Example 3 (continued):



However, there is an algorithm which is not too much more sophisticated than this which will always find a good-guess for a k -uniform-complete sample in polynomial time. It is a procedure which is called "contraction of a finite tree" by Trahtenbrot and Barzdin [1973], pp. 98-99, where it is given without proof.

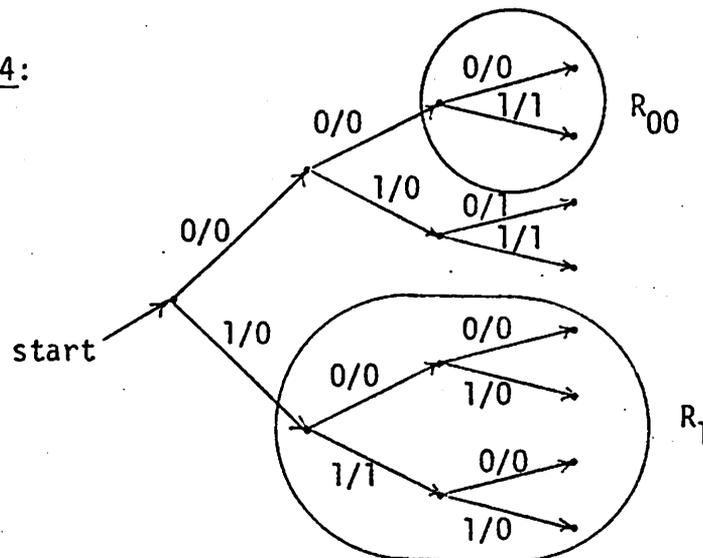
The idea of this procedure is to start with a state-set Q , initially $= \{\Lambda\}$, and to work breadth-first from the root to the leaves of the tree machine for S , adding a new state to Q only if it can be distinguished from all of the states currently in Q by information contained in the sample, and otherwise, identifying it with any state in Q from which it is indistinguishable by information contained in the sample. We will specify this algorithm and prove that it works.

Suppose we are given S , a k -uniform-complete sample for some $k \geq 1$. Let f denote the sample-function presented by S . For each $u \in U^k \cup \{\Lambda\}$, we let

$$R_u = \{ \langle z, f(uz) \rangle \mid z \in U^{k-|u|} \} .$$

(So if $|u| = k$, then $R_u = \emptyset$.) The set R_u represents the subtree of the tree machine for S which is rooted at the state $[u]$.

Example 4:



Definition: If $s, t \in U^k \cup \{\Lambda\}$ then we say that s can be merged to t if and only if $R_s \subseteq R_t$.

Thus $R_s \subseteq R_t$ means that the subtree rooted at $[s]$ is found embedded in the subtree rooted at $[t]$, for example $R_{00} \subseteq R_1$ in Example 4.

The following algorithm calculates a state-set $Q \subseteq U^k \cup \{\Lambda\}$. During execution, the set L contains those strings in $U^k \cup \{\Lambda\}$ which remain to be processed. The set T contains a record of identifications made: $\langle s, t \rangle \in T$ means that the state s is identified with the state t .

Algorithm A:

```

begin
  Q ← {Λ}; L ← Uk; T ← ∅;
  while L ≠ ∅ do
    begin
      s ← least(L);
      M ← {u | u ∈ Q and s can be merged to u};
      if M = ∅ then
        begin
          remove s from L and add it to Q;
        end
      else
        begin
          t ← anyelement(M);
          T ← T ∪ {⟨s, t⟩};
          remove s and all of its extensions from L;
        end
      end
    end
end.

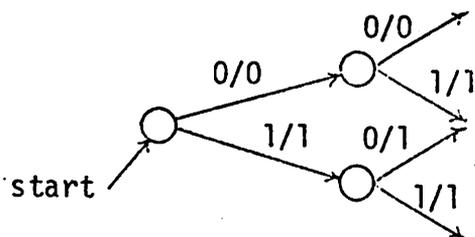
```

where the function "least" selects the lexically smallest of a finite

nonempty set of strings, and "anyelement" simply returns one element of a finite nonempty set.

We give an example of the execution of Algorithm A.

Example 5: Consider the 2-uniform-complete sample given by



After initialization, we have:

$$Q = \{\Lambda\}, \quad L = \{0,1,00,01,10,11\}, \quad T = \emptyset .$$

The first time through the "while" loop, state 0 is identified with state $\Lambda \in Q$, giving values:

$$Q = \{\Lambda\}, \quad L = \{1,10,11\}, \quad T = \{<0,\Lambda>\} .$$

The next time through, the state 1 is found to be distinguishable from $\Lambda \in Q$, and is added to the state-set:

$$Q = \{\Lambda,1\}, \quad L = \{10,11\}, \quad T = \{<0,\Lambda>\} .$$

In the third execution of the loop, 10 is found to be indistinguishable from both Λ and 1, and so is identified with either, say Λ :

$$Q = \{\Lambda,1\}, \quad L = \{11\}, \quad T = \{<0,\Lambda>, <10,\Lambda>\} .$$

In the final execution of the loop, 11 is identified with either Λ or 1, say Λ , resulting in final values:

$$Q = \{\Lambda, 1\}, \quad L = \emptyset, \quad T = \{\langle 0, \Lambda \rangle, \langle 10, \Lambda \rangle, \langle 11, \Lambda \rangle\} .$$

When Algorithm A terminates, we have for each $u \in U^k \cup \{\Lambda\}$ one of the three following possibilities:

i) $u \in Q$, every prefix of u is also in Q , and u cannot be merged to v for any $v \in Q$ such that $v < u$ (in the lexical ordering of strings);

ii) every proper prefix of u is in Q , but u itself is not in Q , and there is a unique $v \in Q$ such that u was identified with v , i.e., $\langle u, v \rangle \in T$;

iii) some proper prefix of u is not in Q , and u was therefore removed from L as an extension of a string in class (ii).

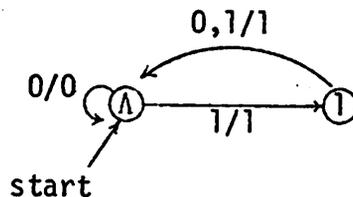
(In particular, if $u \in Q$, then $|u| \leq k-1$, because if $|u| = k$, then $R_u = \emptyset$ and u can be merged to $\Lambda \in Q$, so conditions (i)-(iii) guarantee that $u \notin Q$.)

We use the output set Q as the state-set of a machine M defined as follows: let $u \in Q$ and $a \in U$. Then $|u| \leq k-1$, so $u \cdot a \in U^k$. Thus we may define the output of M for state u and input a to be $f(u \cdot a)$. If $u \cdot a \in Q$, then we simply define the next state of M for state u and input a to be the state $u \cdot a$. Otherwise, $u \cdot a$ is a string of type (ii) above (because $u \in Q$ and $a \in U$), and there is a unique $v \in Q$ such that $\langle u \cdot a, v \rangle \in T$. In this case, we define the next state of M in state u with input a to be the state v . Λ will be the start-state of M .

Example 5 (continued): Applying this procedure to the final values

$$Q = \{\Lambda, 1\}, \quad L = \emptyset, \quad T = \{\langle 0, \Lambda \rangle, \langle 10, \Lambda \rangle, \langle 11, \Lambda \rangle\}$$

we obtain the machine:



Theorem 2: The machine M thus defined is a machine with the minimum possible number of states which is consistent with the input sample S .

Proof: (Recall from the definitions, p. 10, that $[u]_M$ denotes the state M reached from the start-state on input u , and $M'(u)$ denotes the last symbol of output produced by M on input u . $[u]_M$ will be written simply $[u]$ in what follows.)

It is clear from the definition of M that $[u] = u$ for all strings $u \in Q$. To show that M is consistent with S , we must show that $M'(u) = f(u)$ for all $u \in U^k$. If $a \in U$, then $M'(a) = f(a)$, by definition of M . Now suppose that $M'(v) = f(v)$ for all strings v which precede some string $u \in U^k$, with $|u| \geq 2$. Then $u = w \cdot a$ for some $a \in U$.

If $w \in Q$, then

$$\begin{aligned}
 M'(u) &= M'([w], a) \\
 &= M'(w, a) \\
 &= f(w \cdot a) \\
 &= f(u) .
 \end{aligned}$$

If $w \notin Q$, then $u = z \cdot b \cdot t$ where z is the longest prefix of u which is in Q , $b \in U$, and $t \in U^+$. Then $z \cdot b$ is a string of type (ii), so there is a unique $v \in Q$ such that $\langle z \cdot b, v \rangle \in T$. By the definition of M , we have $[z \cdot b] = v$. Hence

$$\begin{aligned}
M'(u) &= M'(z \cdot b \cdot t) \\
&= M'([z \cdot b], t) \\
&= M'(v, t) \\
&= M'(v \cdot t) \\
&= f(v \cdot t)
\end{aligned}$$

because $v < z \cdot b$, so $v \cdot t < z \cdot b \cdot t = u$, so our inductive assumption applies to $v \cdot t$. Since $\langle z \cdot b, v \rangle \in T$, we know that $z \cdot b$ can be merged to v , so $R_{z \cdot b} \subseteq R_v$. But $\langle t, f(z \cdot b \cdot t) \rangle \in R_{z \cdot b}$, so we must have $f(v \cdot t) = f(z \cdot b \cdot t) = f(u)$. Thus $M'(u) = f(u)$.

Hence in either case we get $M'(u) = f(u)$, so by induction we may conclude that this holds for every $u \in U^k$, i.e., that M is consistent with S .

To see that M has the minimum possible number of states among all machines consistent with S , we let N be any machine which is consistent with S . Let $u, v \in Q$ with $u \neq v$. Assume without loss of generality that $u < v$. Then from observation (i) above, we know that v cannot be merged to u , i.e., that $R_v \not\subseteq R_u$. Thus there is some element of R_v , say $\langle t, f(v \cdot t) \rangle$, with $1 \leq t \leq k - |v|$, which is not an element of R_u . Since $u < v$, $|u| \leq |v|$, so $1 \leq t \leq k - |u|$. Therefore, $\langle t, f(u \cdot t) \rangle \in R_u$, so we conclude that $f(v \cdot t) \neq f(u \cdot t)$. But since N is consistent with S , we have $N'(v \cdot t) = f(v \cdot t)$ and $N'(u \cdot t) = f(u \cdot t)$. Hence, the states $[u]_N$ and $[v]_N$ are distinguishable by the experiment t , that is, $[u]_N \neq [v]_N$. Since u and v were arbitrary distinct elements of Q , we conclude that N has at least $|Q|$ states, at least as many as the machine M . \square

Since the set L in Algorithm A is initially set to U^k , and each iteration of the "while" loop removes at least one element from L , it is not too difficult to see that the algorithm can be implemented to run in time polynomial in the size of the k -uniform-complete sample S .

An Extension of A: If S is a sample which would be k -uniform-complete with the addition of at most dk strings to the domain of its sample-function, then we could execute Algorithm A for each possible way of extending S to be k -uniform-complete and output a smallest among all the resulting machines; this would obviously be a good-guess for the sample S . For a fixed constant d , the number of possibilities we must try out is $|V|^{dk}$, which is polynomial in the size of S , provided $|U| \geq 2$ so that the whole procedure would run in time polynomial in the size of S . We have

Theorem 3: For any $d \geq 0$, we may find good-guesses in polynomial time for all samples S which are defined exactly on U^k less at most $d \cdot \log(|U^k|)$ strings.

2.5 A Further Extension of the Uniform-Complete Case Which is NP-Hard

This section is devoted primarily to a construction to demonstrate:

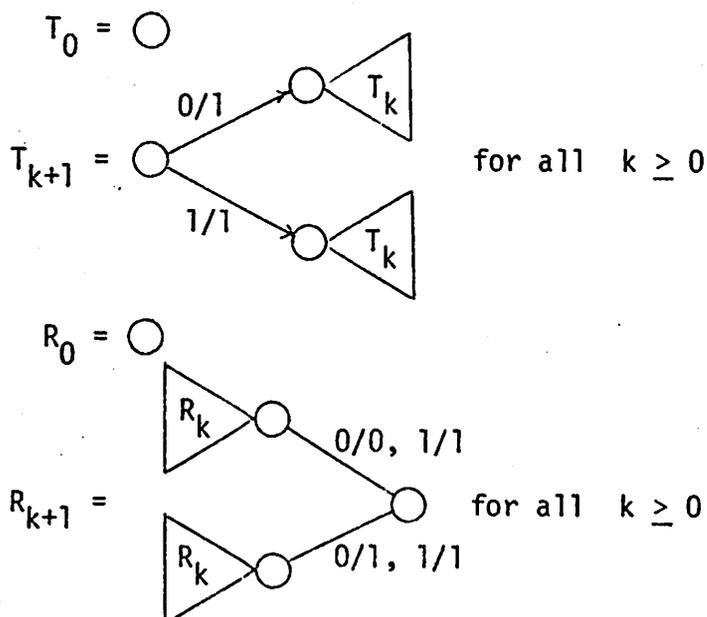
Theorem 4: For any $\epsilon > 0$, the problem of finding a good-guess for any sample S which is defined on U^k less at most $|U^k|^\epsilon$ strings for some $k \geq 1$, is NP-hard, provided $|U|, |V| \geq 2$.

This is to be contrasted with Theorem 3, which says that if the number of unspecified strings is bounded by $d \cdot \log(|U^k|)$ for some d , then the problem may be solved in polynomial time. What happens in the

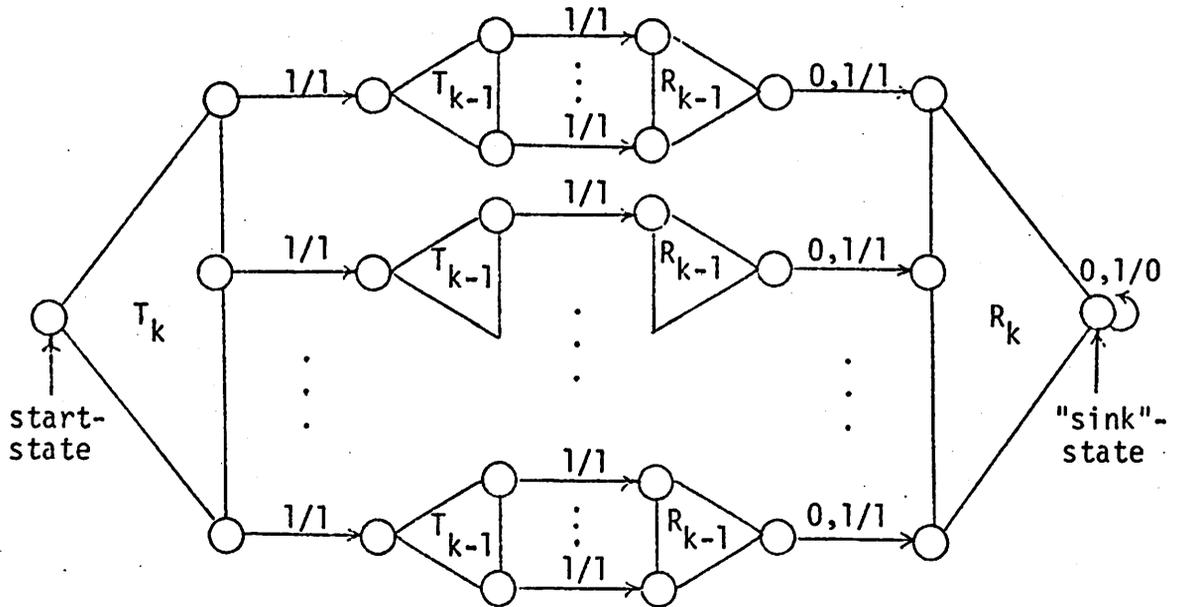
gap between $d \cdot \log(|U^k|)$ and $|U^k|^e$ is an open problem.

The reason that Gold's construction for the proof of Theorem 1 does not suffice to prove Theorem 4 is that he translates a propositional formula ϕ of m clauses and n variables into a sample containing input strings of length $k \geq \max\{m, n\}$. To make such a sample "nearly" k -uniform-complete in the sense of the theorem would make it of size at least $c \cdot 2^k$ for some constant c , which would not be polynomial in the size of ϕ , as required for the NP-reduction. Thus the primary purpose of this new construction is to keep the strings of the sample to length $O(\log k)$; the "e" may then be achieved by a standard sort of padding. The construction will also allow us to conclude that finding good-guesses is NP-hard even if we restrict the problem to "finite-language" or to "definite" machines [see p. 30].

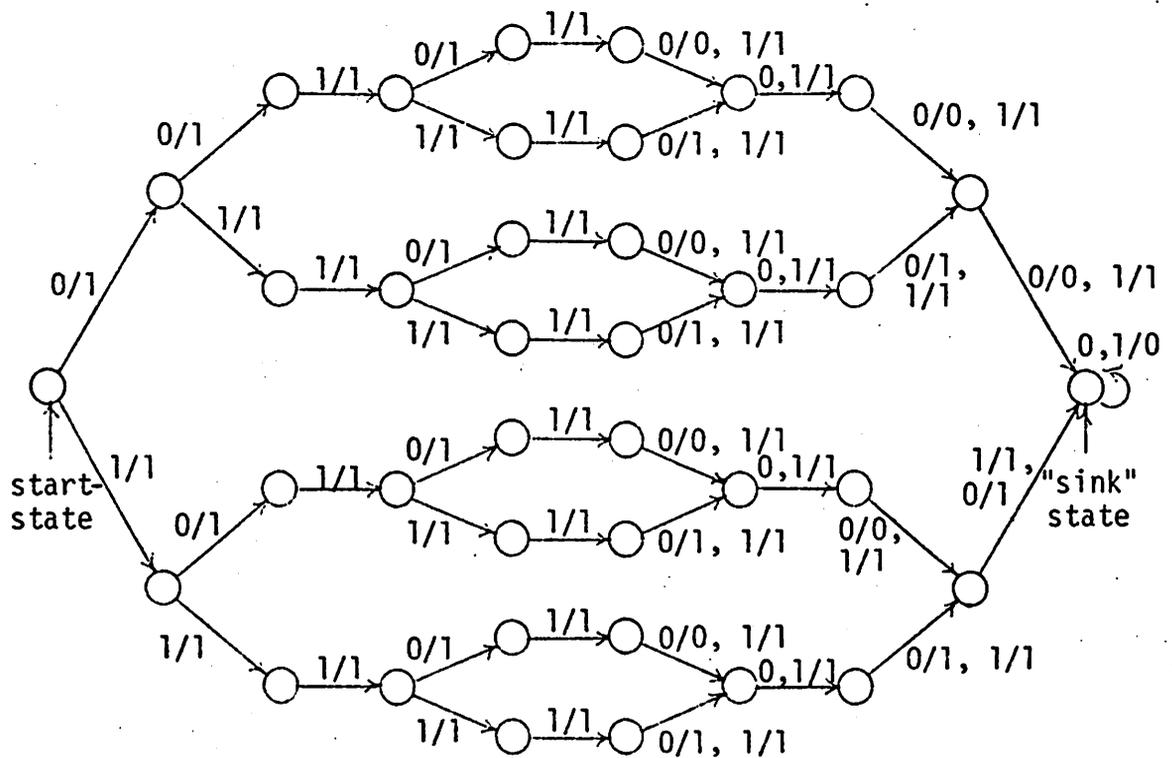
A Class of Machines: We first define a particular set of incompletely-specified machines. The definitions will be given pictorially:



and for all $k \geq 1$, M_k is the partially-specified machine:



Thus, for example, M_2 is the partially-specified machine:



M_k will have $2^{2k+1} + 2^{k+1} - 2$ states. We define the level of a state i in M_k to be the "distance" of state i from the start-state:

$$\text{level}(i) = \min \{ \ell \mid \exists u (|u| = \ell \text{ and } [u] = i) \}$$

The start-state will be at level 0 and the sink-state at level $4k+1$. The only unspecified transitions and outputs of M_k are the 0-transitions and 0-outputs from states at levels k and $2k$.

We choose a set Q_k of strings to represent the states of M_k . For each state i in M_k , $r(i)$ is taken to be the "southern route" from the start-state to i , i.e.,

$$r(i) = \max \{ u \mid |u| = \text{level}(i) \text{ and } [u] = i \}$$

where the maximum is taken in the lexical ordering of strings (with $0 < 1$). Then

$$Q_k = \{ r(i) \mid i \text{ is a state of } M_k \}$$

We note several facts that can be proved regarding M_k :

1. If $u \in Q_k$ then $M_k(u \cdot 1^{4k+2-|u|}) = 1^{4k+1} \cdot 0$.
2. If $u \in Q_k$ and $|u| \leq 2k$ then $M_k(u \cdot 1^{4k+1-2|u|} \cdot 0^{|u|}) = 1^{4k+1-|u|} \cdot u^r$, where u^r denotes the reverse of the string u .
3. If i is any state at level $\ell > 2k$ then $r(i)$ will be of the form $u \cdot 1^m$ where $|u| = 4k+1-\ell$ and $m = 4k+1-2|u|$.

The first fact allows us to distinguish two states at different levels of M_k ; the latter two allow us to distinguish between two states at the same level. We use these ideas to construct a sample:

let

$$E_k = \{e \mid e = u \cdot 1^{4k+2-|u|} \text{ for some } u \in Q_k, \text{ or} \\ e = u \cdot 1^{4k+1-2|u|} \cdot 0^{|u|} \text{ for some } u \in Q_k \text{ with } |u| \leq 2k\}$$

and

$$S_k = \{\langle e, M_k(e) \rangle \mid e \in E_k\} .$$

Claim: If M is any machine which is consistent with the sample S_k , then $[u]_M \neq [v]_M$ whenever $u, v \in Q_k$ and $u \neq v$.

(Thus in particular, M must have at least $2^{2k+1} + 2^{k+1} - 2$ distinct states.)

Proof: Let $u, v \in Q_k$ with $u \neq v$. Denote $[u]_M$ and $[v]_M$ by $[u]$ and $[v]$ simply. Since M is consistent with S_k , we must have

$$M([u], 1^{4k+2-|u|}) = 1^{4k+1-|u|} \cdot 0 ,$$

$$M([v], 1^{4k+2-|v|}) = 1^{4k+1-|v|} \cdot 0 .$$

Thus if $|u| \neq |v|$, say $|u| < |v|$, then we have $M([u], 1^{4k+2-|v|}) = 1^{4k+2-|v|}$, so $[u] \neq [v]$. If $|u| = |v|$, then if $|u| \leq 2k$, we have

$$M([u], 1^{4k+1-2|u|} \cdot 0^{|u|}) = 1^{4k+1-2|u|} \cdot u^r$$

$$M([v], 1^{4k+1-2|u|} \cdot 0^{|u|}) = 1^{4k+1-2|u|} \cdot v^r$$

so again $[u] \neq [v]$. In the last case, $|u| = |v| > 2k$, we have by Fact 3 above that

$$u = u_1 \cdot 1^m \text{ and } v = v_1 \cdot 1^m ,$$

where $|u_1| = |v_1| = 4k+1-|u|$, and $m = 4k+1-2|u_1|$. Thus

$$M([u], 0^{|u_1|}) = u_1^r$$

and

$$M([v], 0^{|u_1|}) = v_1^r,$$

so we conclude in this final case that $[u] \neq [v]$ once again. \square

Encoding a SATISFIABILITY Problem Using S_k

We now show how to augment the sample S_k to "represent" a propositional formula. Let ϕ be a propositional formula in conjunctive normal form (a conjunction of a set of clauses, each of which is a disjunction of some literals), for example $\phi = (A \vee \bar{B} \vee C)(\bar{A} \vee D)(B \vee \bar{C} \vee \bar{D})$. We may without loss of generality assume that each clause of ϕ contains only positive or only negative occurrences of variables. (If not, we may use the fact that $(p \vee q)r$ is satisfiable if and only if $(p \vee X)(q \vee \bar{X})r$ is satisfiable, provided that X is a variable that does not occur in p , q , or r , to transform the problem to meet this condition.)

If m and n are the numbers of clauses and variables in ϕ , respectively, we fix $k = \lceil \log(\max\{m, n\}) \rceil + 1$, so that $2^{k-1} \geq m, n$. Let

$$C = \{\text{the first } m \text{ binary strings of length } k-1\}$$

$$V = \{\text{the first } n \text{ binary strings of length } k-1\}$$

and put these into one-to-one correspondence with the clauses and variables of ϕ , respectively, so that we can talk about clause c and variables v for $c \in C$ and $v \in V$.

We use some of the 0-outputs from states of M_k at level $2k$ to encode the variable/clause incidence relation in ϕ :

$$A_1 = \{ \langle 1u1w0, 1^{2k} \delta \rangle \mid u, w \in \{0,1\}^*, |u| = |w| = k-1, \text{ and } \delta = 1 \text{ if and only if } u \in V, w \in C, \text{ and variable } u \text{ appears in clause } w \text{ of } \phi, \text{ else } \delta = 0 \} .$$

Then we constrain the 0-transitions from states $[0c]$ for $c \in C$ as follows:

$$A_2 = \{ \langle 0c01^{3k+2}, 1^{4k+2} 0 \rangle \mid c \in C \}$$

$$A_3 = \{ \langle 0c01^{3k} 0, 1^{4k+2} \rangle \mid c \in C \}$$

$$A_4 = \{ \langle 0c01c0, 1^{2k+2} \rangle \mid c \in C \}$$

$$A_5 = \{ \langle 0c00, 1^{k+1} \delta \rangle \mid c \in C \text{ and } \delta = 0 \text{ if clause } c \text{ contains only negative occurrences of variables, } \delta = 1 \text{ if clause } c \text{ contains only positive occurrences of variables} \} .$$

We set $S_k^* = S_k \cup A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5$. It is not too difficult to see that S_k^* is a sample, that it presents a well-defined sample-function.

Claim: There exists a machine of $2^{2k+1} + 2^{k+1} - 2$ states in agreement with the sample S_k^* if and only if the formula ϕ is satisfiable.

Proof: (\Rightarrow) Suppose M is a machine of $2^{2k+1} + 2^{k+1} - 2$ states which is consistent with S_k^* . Then, in particular, M is consistent with S_k , so M must contain exactly $2^{2k+1} + 2^{k+1} - 2$ states, namely $\{[u]_M \mid u \in Q_k\}$. We will denote $[u]_M$ by $[u]$ simply, in what follows. For each $v \in V$, let $t(v) = M([1v], 0)$, the 0-output of M in the state $[1v]$. We will show that t is an assignment of truth-values to the variables of ϕ which will satisfy ϕ . Let $c \in C$, then

$$M([0c0], 1^{3k+2}) = 1^{3k+1} 0 \text{ from agreement with } A_2,$$

$$M([0c0], 1^{3k} 0) = 1^{3k+1} \text{ from agreement with } A_3.$$

Taken together, these imply that $[0c0]$ must be a state in the lower half of the k^{th} level of M_k , that is, that $[0c0] = [1u]$ for some $u \in \{0,1\}^*$ with $|u| = k-1$. Then

$$\begin{aligned} M([1u], 1c0) &= M([0c0], 1c0) \\ &= 1^{k+1} \text{ from } A_4. \end{aligned}$$

But by A_1 , this implies that u is a variable which appears in clause c . Finally, by A_5

$$M([0c0], 0) = \begin{cases} 0 & \text{if } c \text{ contains only negative occurrences of} \\ & \text{variables} \\ 1 & \text{if } c \text{ contains only positive occurrences of} \\ & \text{variables} \end{cases}$$

and $t(u) = M([1u], 0) = M([0c0], 0)$, so the assignment of the value $t(u)$ to variable u makes clause c true. Since c was arbitrary, the assignment t makes all the clauses of ϕ true, i.e., satisfies ϕ .

(\Leftarrow) Conversely, suppose that ϕ is satisfiable. We choose an assignment $t: V \rightarrow \{0,1\}$ which satisfies ϕ , and for each clause $c \in C$, we pick a variable $v \in V$ which appears in clause c and which makes clause c true under the assignment t . (That is, $t(v) = 0$ if c contains only negative occurrences of variables, 1 otherwise.) We now specify the missing outputs and transitions of M_k in such a way that the resulting machine is consistent with S_k^* and has no more states than M_k .

1. The 0-output from state $[0c]$, where $c \in C$, will be "1"; the 0-transition will be to state $[1v]$, where $v \in V$ was the

variable picked to make clause c true under the assignment t .

2. The 0-output from the state $[1v]$ where $v \in V$ will be $t(v)$.

3. The 0-output from state $[1xly]$ will be "1" provided $x \in V$, $y \in C$, and variable x appears in clause y of ϕ , "0" otherwise, for all $x, y \in \{0,1\}^*$ such that $|x| = |y| = k-1$.

4. The rest of the missing transitions and outputs in M_k may be specified in any way within M_k . We note that in particular they may be chosen so that the resulting machine is a "finite-language" machine, that is, outputs "1" for only a finite number of possible input strings.

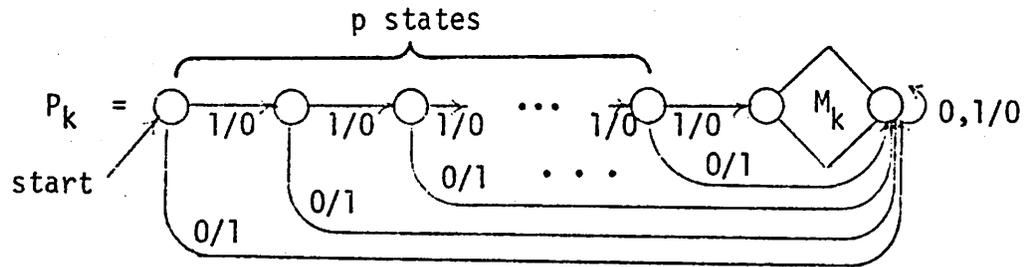
We call the machine that results from this procedure M_k^* ; we must see that it is consistent with S_k^* . M_k^* is an extension of M_k , and thus is consistent with S_k . Consistency with A_1 is by direct construction in item #3 above. Let c be any element of C . Then $M_k^*(0c0) = 1^{k+1}$ and $[0c0] = [1v]$, where v is the variable chosen to make clause c true under assignment t . This easily gives agreement with A_2, A_3, A_4 . Finally, $M_k^*([0c0], 0) = M_k^*([1v], 0) = t(v)$, and v was chosen so that $t(v)$ would agree with the sense of clause c . □

Analysis

Since $k = O(\log|\phi|)$, and the longest strings in S_k^* are of length $4k+3$, we can easily see that S_k^* will be of size polynomial in $|\phi|$. It is also not difficult to see that S_k^* can be constructed from ϕ in time polynomial in $|\phi|$. Thus we have given a polynomial-time reduction of SATISFIABILITY to the problem of finding good-guesses for the samples $S_k^*(\phi)$, and we conclude that the latter problem is

NP-hard, since SATISFIABILITY is complete in NP (Cook [1971]).

All that remains to complete the proof of Theorem 3 is to achieve the " ϵ "; this is done by "padding" the sample S_k^* to bury the effect of the at most 2^{4k+4} strings used in specifying ϕ . One way to do this is to add a preamble of length p to M_k , thus:



Each of the additional p states may be distinguished from the sink-state of M_k by its output under input "0", and from all the other states of M_k by its output under input "1". They may be distinguished from each other by their behaviour under the string 1^{p+1} .

We construct:

$$B_1 = \{ \langle s, P_k(s) \rangle \mid |s| \leq p+4k+3 \text{ and } s \neq 1^p t \text{ for all } t \in \{0,1\}^+ \}$$

$$B_2 = \{ \langle 1^p s, 0^p t \rangle \mid \text{for all } \langle s, t \rangle \in S_k^* \}$$

and define $S_k^{**} = B_1 \cup B_2$. S_k^{**} will be a sample containing input strings of maximum length $p+4k+3$, with at most 2^{4k+4} of them unspecified. We will have: ϕ is satisfiable if and only if there is a machine with $p+2^{2k+1}+2^{k+1}-2$ states in agreement with S_k^{**} . Given ϵ , $0 < \epsilon < 1$, we choose $p = \lceil (4k+4)/\epsilon \rceil$. Then the domain of S_k^{**} will be U_{p+4k+3} less at most $\lceil U_{p+4k+3} \rceil^\epsilon$ strings, as required for Theorem 3.

Restrictions on the Type of Machines

Definition: A machine M is a finite-language machine if and only if its output alphabet is $\{0,1\}$ and it gives the output "1" for only a finite number of input strings.

Definition: A machine M is definite if and only if there exists an integer $k \geq 0$ and a function f mapping the set of input strings of length k to the output alphabet such that for any input string u of length $\geq k$, $M'(u) = f(v)$, where v is the suffix of u of length k .

Note that a finite-language machine is necessarily definite. In the proof above, we observe that if ϕ is unsatisfiable, then there will be no machine of $\leq 2^{2k+1} + 2^{k+1} - 2$ states in agreement with S_k^* , but if ϕ is satisfiable, then the machine constructed from a satisfying instance for ϕ may be taken to be finite-language, and hence definite. Thus:

Corollary: The problem of finding a smallest finite-language (or definite) machine consistent with a given sample is NP-hard.

Notes

1. In the special case that the output alphabet is $\{0,1\}$ and we interpret the underlying machine as a finite-state acceptor, we might wish to present a k -uniform-complete sample as

$S_1 = \{\text{precisely those strings of length } \leq k \text{ which are accepted}\}$,

i.e., the subset of U^k which is accepted by the machine. S_1 may be of size polynomial rather than exponential in k for particular

families of machines. For this case, Algorithm A can be modified to run in time polynomial in the size of S_1 rather than in the size of U^k as follows: Initialize L to the set of all nonnull prefixes of strings in S_1 . For all $s, t \in L$, we consider $T_s = \{u \in U^+ \mid su \in S_1\}$ and say that s can be merged to t if and only if $|s| \geq |t|$ and T_s is identical to T_t for strings of length $\leq k - |s|$. In the specification of the machine M we may be compelled to introduce a "dead state" to guarantee that the machine is completely specified, but otherwise the construction carries over in a straightforward fashion. This modification of A may be of some practical interest.

CHAPTER 3

Finding a Smallest Regular Expression Compatible with Given Data3.1 Introduction

In this paper we consider the problem of "guessing" a regular expression to describe an unknown finite-state language, on the basis of a few examples of strings in the language and a few examples of strings not in the language. For example, given:

<u>in L</u>	<u>not in L</u>
011100	11111
1111110111	001111
111	11000
000111111	010

we might guess that the language L was described by the expression $(0 \vee 111)^*$.

Without additional kinds of information about L (for example, an upper bound on the length of an expression describing L), we can never be certain we have correctly identified L . However, we might still want to find a "reasonable" or "good" guess in this situation. We will define a "good guess" to be a regular expression of minimum length which is consistent with the given sample. This definition corresponds to a demand for succinctness in a scientific theory of given data; it also leads to a correct inference in the limit, in the sense defined by Mark Gold [1967].

For inference, regular expressions are interesting for two possibly related reasons. In simple cases, regular expressions correspond somewhat with natural language descriptions of inferences from samples of finite-state languages. Also, the strings composing a regular

expression (using the operators \vee , \cdot , $*$) appear as substrings of the positive strings of the samples. Both of these may provide insight into efficient constructive approaches to the inference of regular expressions. The "substrings" property is exploited in the classes of expressions constructed in Section 3.3.

One way to compute a shortest regular expression compatible with given data is to enumerate all regular expressions in order of size and test them for compatibility with the sample. This potentially exponential-time algorithm is the best we currently possess for finding shortest compatible expressions. Moreover, Theorem 2, in Section 3.4, will show that there is no polynomial-time algorithm for finding a shortest compatible expression for arbitrary given data (if $P \neq NP$).

This result motivates a search for computationally more tractable special cases of the problem of finding "good guesses" from given data. Ideally, we would like to characterize those features of the sample or of the expressions to be inferred which permit efficient inference. Perhaps such features would include formal correlates of our intuitions of "transparency" or "clues" or "structure" in the samples or expressions.

Toward this end, we present the beginnings of a study of the "inferrability" of various restricted classes of regular expressions, from arbitrary given data.

3.2 Definitions, Notation, Conventions

Definition: Σ will be a finite nonempty set, the alphabet. Λ will denote the null string, Σ^+ the set of all non-null strings over the alphabet Σ , and $\Sigma^* = \Sigma^+ \cup \{\Lambda\}$. If $s, t \in \Sigma^*$, the concatenation of s and t will be written st . If $A, B \subseteq \Sigma^*$ then

$$AB = \{s \mid s = tu \text{ for some } t \in A \text{ and } u \in B\}$$

and we define A^n as follows:

$$A^0 = \{\Lambda\}$$

$$A^{i+1} = A^i A \text{ for } i \geq 0.$$

Definition: The set of regular expressions over Σ will be the smallest set of strings E such that

1. $\Sigma \subseteq E$
2. $\underline{\Lambda}, \underline{\emptyset} \in E$ where $\underline{\Lambda}, \underline{\emptyset}$ are distinct symbols which are not in Σ .
3. Whenever $E, F \in E$ then

$$(E \cdot F) \in E$$

$$(E \vee F) \in E$$

$$(E)^* \in E \text{ where the symbols } (,), \cdot, \vee, * \text{ are not in } \Sigma.$$

The set of regular expressions over Σ will be denoted $\langle \Sigma, \vee, \cdot, * \rangle$.

The subset of $\langle \Sigma, \vee, \cdot, * \rangle$ consisting of strings which do not contain the symbol \vee will be denoted $\langle \Sigma, \cdot, * \rangle$; similarly for the other operators.

Definition: If E is a regular expression, we define the language of E , denoted $L(E)$, inductively, as follows:

1. $L(a) = \{a\}$ for all $a \in \Sigma$
2. $L(\underline{\Lambda}) = \{\Lambda\}$; $L(\underline{\emptyset}) = \emptyset$ (the null set)
3. $L((E \cdot F)) = L(E)L(F)$; $L((E \vee F)) = L(E) \cup L(F)$;
 $L((E)^*) = \bigcup_{i=0}^{\infty} (L(E))^i$

With respect to the language denoted, concatenation (" \cdot ") is

associative, and union (" \vee ") is associative and commutative. We will freely omit unnecessary parentheses and the concatenation symbol (" \cdot ") from regular expressions.

Definition: If E is a regular expression, its length, denoted $|E|$, is defined as follows:

1. $|a| = 1$ for $a \in \Sigma$
2. $|\underline{\Lambda}| = 1$ and $|\underline{\emptyset}| = 1$
3. $|(E \cdot F)| = |E| + |F|$; $|(E \vee F)| = |E| + |F| + 1$; $|(E)^*| = |E| + 1$

Note that parentheses and the concatenation symbol do not count towards the length of an expression, so that the length of any expression is simply the number of symbols from $\Sigma \cup \{\underline{\Lambda}, \underline{\emptyset}, \vee, *\}$ used in it. This definition is mostly for convenience in the proofs to follow; the results could be proved under different definitions of length.

Examples: Let $\Sigma = \{0,1\}$.

$$L((0 \vee 1)^*1) = \{\text{all strings of 0's and 1's which end in a 1}\}$$

$$|((0 \vee 1)(0 \vee 1))^*00| = 9$$

$$L((\underline{\emptyset})^*) = \{\underline{\Lambda}\}$$

$$L(((00)^*\underline{\emptyset}) = \underline{\emptyset}$$

$$|(0000)^*| = 5$$

Definition: A sample S over Σ will be an ordered pair $S = \langle S_1, S_0 \rangle$ where S_1, S_0 are finite subsets of Σ^* and $S_1 \cap S_0 = \emptyset$. S_1 will be called the positive part of S , and S_0 the negative part of S .

Definition: A regular expression E is consistent (or compatible) with a sample $S = \langle S_1, S_0 \rangle$ if and only if $S_1 \subseteq L(E)$ and $S_0 \cap L(E) = \emptyset$, that is, if and only if E generates all the strings in the positive part of S and none of the strings in the negative part of S .

Example: The sample presented in Example 1 is $S = \langle \{011100, 111110111, 111, 000111111\}, \{11111, 001111, 11000, 010\} \rangle$. The expression $(0 \vee 111)^*$ is compatible with this sample, the expressions 0^* and $(0 \vee 1)^*$ are not compatible with it.

Definition: If S is a sample and E is a set of regular expressions, then a regular expression E will be called a good-guess from E for S if and only if

1. $E \in E$;
2. E is compatible with S ;
3. for every $F \in E$, if F is compatible with S , then $|E| \leq |F|$.

If E is the set of all regular expressions, then we will simply say that E is a good-guess for S .

Definition: If E is a set of regular expressions, then E will be called efficiently-inferable if and only if there exists an algorithm A such that

1. A runs in polynomial time;
2. for every sample S ,

$$A(S) = \begin{cases} E & \text{where } E \text{ is a good-guess from } E \text{ for } S, \\ & \text{if any expressions from } E \text{ are compatible} \\ & \text{with } S \\ \text{"NONE"} & \text{otherwise} \end{cases}$$

Remark 1: If E and E' are efficiently-inferrable, then so is $E \cup E'$. (We simply run both algorithms and take a smallest answer, if any.)

Definition: If E is a set of regular expressions, then E will be called hard-to-infer if and only if $P \neq NP$ implies that E is not efficiently-inferrable.

(Note that efficiently-inferrable and hard-to-infer would simply be complements of each other if we could prove $P \neq NP$.)

Conventions: We assume straightforward encodings of strings, expressions, sets, integers as inputs and outputs of programs. We also assume familiarity with the definitions and results concerning sets recognizable in deterministic polynomial time (P) and nondeterministic polynomial time (NP), and the notions of NP-complete and NP-hard problems, as developed in Cook [1971] and Karp [1972].

3.3 Some Efficiently-Inferrable Classes

In this section we present some classes of regular expressions from which good-guesses may be found in polynomial time for arbitrary samples. We start with an example:

Let $\Sigma = \{0,1\}$.

Let $E = \{E \mid E = (x \vee y)^*$ for some $x, y \in \Sigma^*\}$. If we are given the sample $S = \langle S_1, S_0 \rangle$:

<u>S_1</u>	<u>S_0</u>
11001	011
10000	1000

we proceed to enumerate all substrings of the strings in the positive

part of the sample:

$$W = \{\Lambda, 0, 1, 00, 01, 10, 11, 000, 001, 100, 110, 0000, 1000, 1001, 1100, 10000, 1101\}$$

and for each pair $\langle r, s \rangle \in W \times W$, we form the expression $(rVs)^* \in E$ and test it for compatibility with S . A shortest regular expression found in this way which is compatible with S is $(1V00)^*$. It is easy to see that for any $(xVy)^* \in E$ which is compatible with S , both x and y must appear as substrings of the positive strings of S , so that $(1V00)^*$ will in fact be a shortest expression from E which is compatible with S , i.e., a good-guess from E for S .

The idea of this example may be formalized as follows:

Definition: Let x_1, x_2, x_3, \dots be an infinite sequence of new symbols. A regular form in k variables will be any regular expression over the alphabet $\Sigma \cup \{x_1, x_2, \dots, x_k\}$.

Definition: If $k \geq 0$ is an integer, then a k -substitution will be any function $f: \{x_1, x_2, \dots, x_k\} \rightarrow \Sigma^* \cup \{\emptyset\}$. For any set A of symbols disjoint from the x_i 's, we will implicitly consider f extended to map $(A \cup \{x_1, x_2, \dots, x_k\})^*$ to $(\Sigma \cup A \cup \{\emptyset\})^*$ by $f(a) = a$ for any $a \in A$, and $f(st) = f(s)f(t)$, for all strings s and t .

Definition: If F is a regular form in k variables, then a regular expression E is a substitution-instance of F if and only if there exists a k -substitution f such that E is the result of replacing in F every occurrence of x_i by $f(x_i)$ for $i = 1, 2, \dots, k$, i.e., $E = f(F)$.

Definition: If F is a regular form, then the set of regular expressions denoted by F , which will be denoted $E(F)$ and called a regular form class, is defined:

$$E(F) = \{E \mid E \text{ is a substitution-instance of } F\}$$

Examples:

$F = (x_1 \vee x_2)^*$ is a regular form in two variables.

$f: x_1 \rightarrow 00, x_2 \rightarrow 011$ is a 2-substitution.

$E = (00 \vee 011)^* = f(F)$ is a substitution-instance of F .

Let F be a regular form in k variables. We give an algorithm, A_F , for finding good-guesses from $E(F)$ for an arbitrary sample S .

Algorithm A_F :

input: $S = \langle S_1, S_0 \rangle$, a sample

output: C , a set of expressions compatible with the sample S

begin

$C \leftarrow \emptyset$;

$W \leftarrow \{s \mid s \text{ is a substring of some string from } S_1\}$;

for each k -tuple $\langle s_1, s_2, \dots, s_k \rangle \in (W \cup \{\emptyset\})^k$ do

begin

let f be the k -substitution $x_i \rightarrow s_i$ for $i = 1, 2, \dots, k$;

$E \leftarrow f(F)$;

if E is compatible with S , then $C \leftarrow C \cup \{E\}$;

end

end.

Claim: When A_F halts,

i) C will be a subset of the set of all expressions from $E(F)$ which are compatible with S .

ii) If any expression from $E(F)$ is compatible with S , then C will contain a smallest such.

Proof: An expression E is only added to C if it is a substitution-instance of F and is compatible with S , so (i) clearly holds. To prove (ii), suppose that $E \in E(F)$ is compatible with S . Since $E \in E(F)$, $E = f(F)$ for some k -substitution f . For each $i = 1, 2, \dots, k$, if $f(x_i)$ is a substring of some string from S_1 , then $f(x_i)$ will be placed in W , and so will appear in the i^{th} coordinate of $(W \cup \{\emptyset\})^k$. Otherwise, we need to show that we could map x_i to \emptyset and still have an expression compatible with S .

So, we let $g(x_j) = f(x_j)$ for $j \neq i$, and $g(x_i) = \emptyset$. We claim that $g(F)$ will be compatible with S . Since $|g(F)| \leq |f(F)|$, by iterating this procedure we will eventually arrive at an expression in C which has length not exceeding the length of $E = f(F)$, which will prove condition (ii). We will need:

Definition: If A is any set of strings, define

$$r(A) = \{s \mid s \in A \text{ and } s \text{ does not contain the symbol } \emptyset\}.$$

Note that $r(AB) = r(A)r(B)$ and $r(\bigcup_{i=1}^{\infty} A_i) = \bigcup_{i=1}^{\infty} r(A_i)$.

Lemma: $L(f(F)) = r(f(L(F)))$ for any regular form F in k variables, and any k -substitution f .

This result is not too difficult to prove by induction on regular forms. It says simply that if we remove the strings containing \emptyset from $f(L(F))$, the result will be precisely $L(f(F))$. It does not hold if the language of regular expressions is expanded to contain the "not" operator (\neg) , where $L(\neg E) = \Sigma^* - L(E)$.

Now, to see that $g(F)$ is compatible with S , we note that if $s \in L(g(F))$, then $s \in r(g(L(F)))$, so $s = g(\hat{s})$ for some $\hat{s} \in L(F)$, and $g(\hat{s})$ does not contain \emptyset . Then $f(\hat{s}) = g(\hat{s})$, so $s \in r(f(L(F)))$, so $s \in L(f(F))$. Conversely, if $s \in L(f(F))$, and $f(x_i)$ is not a substring of s , then $s \in r(f(L(F)))$, so $s = f(\hat{s})$ for some $\hat{s} \in L(F)$, and $f(\hat{s})$ does not contain \emptyset . Also, \hat{s} cannot contain x_i , otherwise $s = f(\hat{s})$ would contain $f(x_i)$ as a substring, so $g(\hat{s}) = f(\hat{s})$. Thus, $s \in r(g(L(F)))$, so $s \in L(g(F))$. Since $f(F)$ is compatible with S , and no string from S_1 contains $f(x_i)$ as a substring (by hypothesis), $g(F)$ is compatible with S . \square

Thus, for each $i = 1, 2, \dots, k$ we define

$$f'(x_i) = \begin{cases} f(x_i) & \text{if } f(x_i) \text{ is a substring of some string from } S_1 \\ \emptyset & \text{otherwise.} \end{cases}$$

Then $f'(F) \in C$, and $|f'(F)| \leq |f(F)|$, so if $E(F)$ contains any expressions compatible with S , then C will contain a shortest such. \square

To see that the Algorithm A_F will run in time polynomial in the length of the input, S , we note that if n is the total length of S , then the number of substrings of strings in S_1 , that is, $|W|$, will not exceed Kn^2 for some constant K . Thus, at most $(Kn^2 + 1)^k$ expressions $f(F)$ are tested for compatibility with S . The membership question: " $s \in L(E)$?" may be answered in time polynomial in the lengths of s and E (Aho, Hopcroft, and Ullman [1974]), so the whole procedure will run in time bounded by some polynomial in n , as claimed. (Note that k appears in the exponent of n in the analysis of A_F ; it is critical that the number of strings involved in defining

the set of expressions is fixed.) Thus:

Theorem 1: If F is a regular form, then we may find good-guesses from $E(F)$ for arbitrary samples in polynomial time, that is, $E(F)$ is efficiently-inferable.

Corollary 1.1: Theorem 1 and Remark 1 (Section 3.2) show that finite unions of regular form classes are efficiently-inferable.

However, there are efficiently-inferable sets of regular expressions which are not finite unions of regular form classes, for example:

$$E = \{E \mid E = (x^*y^*)^m \text{ for some } x, y \in \Sigma^* \text{ and } m \geq 1\}$$

Given a sample S of total length n , the maximum length of any string appearing in S is Kn for some constant K . We enumerate $\langle s_1, s_2 \rangle \in W \times W$, where W is the set of all substrings of strings from the positive part of the sample, and substitute $x \rightarrow s_1$, $y \rightarrow s_2$ into each of the regular forms: $x^*y^*, x^*y^*x^*y^*, \dots, (x^*y^*)^{Kn}$. Each of the resulting Kn expressions is tested for compatibility with S . If any expression from E is compatible with S , then this procedure will find a good-guess from E for S , since for all $x, y \in \Sigma^*$ and integers $p \leq q$, we have $L((x^*y^*)^p) \subseteq L((x^*y^*)^q)$ and if $|s| \leq p$ and $s \in L((x^*y^*)^q)$ then $s \in L((x^*y^*)^p)$.

In contrast to the preceding two results, if we define the class:

$$\mathcal{D} = \{E \mid \text{there exist } u, v, w, x \in \Sigma^* \text{ such that } E = E_1 E_2 \cdots E_k \text{ for some } k \geq 1 \text{ where each } E_i \text{ is either } (u)^*v \text{ or } w(x)^*\}$$

then \mathcal{D} will be hard-to-infer, as we shall show in Section 3.4.

For each of the efficiently-inferrable classes of regular expressions considered so far, there is some fixed bound to the number of distinct strings composing any expression in the class. To see that this condition is not necessary to efficient-inferrability, consider the following modification to add "finite patching" to the basic algorithm A_F :

When an expression E is tested for compatibility with the input sample S , if it happens that E is incompatible with S only because it fails to generate some strings, say s_1, s_2, \dots, s_t , from S_1 , the positive part of the sample, then we construct the expression:

$$E' = E \vee s_1 \vee s_2 \vee \dots \vee s_t$$

and add it to the set C of expressions compatible with S . Then C will no longer necessarily be a subset of $E(F)$, but this procedure will find a good-guess for S from the following augmented class:

$$E'(F) = \{E' \mid E' = E \vee s_1 \vee s_2 \vee \dots \vee s_r \text{ for some } r \geq 0, \text{ some } E \in E(F), \\ \text{and some } s_1, s_2, \dots, s_r \in \Sigma^*\}$$

$E'(F)$ will thus be efficiently-inferrable, though there is no bound to the number of strings composing expressions in the class $E'(F)$.

Discussion

The fragmentary positive results of this section present some sets of regular expressions which are efficiently-inferrable. The basic technique is a kind of "cutting and pasting" of the positive part of the sample -- the forms that prescribe the "pasting" are all unfortunately rather independent of the sample, in the examples presented. It would be nice to have further techniques for constructing

efficiently-inferable sets of regular expressions.

In the next section, we will see that the set of all regular expressions, as well as some restricted subsets of it, are hard-to-infer.

3.4. Some Classes Which are Hard-To-Infer

In this section we will present some sets of regular expressions which are hard-to-infer. The first set considered will be the set of all regular expressions: $\langle \Sigma, \vee, \cdot, * \rangle$, $|\Sigma| \geq 2$.

Theorem 2: Let $E = \langle \{0,1\}, \vee, \cdot, * \rangle$. Then E is hard-to-infer. In particular, the problem of finding a smallest regular expression compatible with an arbitrary sample is NP-hard.

Proof: The proof is a polynomial-time reduction of a known NP-complete problem, the SATISFIABILITY of propositional formulas in conjunctive normal form [Cook, 1971] to the problem of finding a good-guess for an arbitrary sample. So, suppose we are given a propositional formula ϕ in conjunctive normal form, e.g., $\phi = (X_1 \vee \bar{X}_3 \vee X_4) (\bar{X}_2 \vee \bar{X}_4) (\bar{X}_1 \vee X_2 \vee X_3)$, with clauses numbered 1 through m and variables numbered 1 through n . We consider a sample $S(\phi) = \langle S_1, S_0 \rangle$ as follows:

1. Let $q = (111000)^n$. Specify $q \in S_1$. (This is the only string that will be in the positive part of the sample; we have given it the name q for later reference.)

2. For every substring s of q , say $q = rst$, if s contains both 0's and 1's then specify $rsst \in S_0$.

3. For each $j = 1, 2, \dots, n$ specify $(111000)^{j-1} 10(111000)^{n-j} \in S_0$.

4. For $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$ let

$$F(i, j) = \begin{cases} 111000 & \text{if variable } j \text{ does not appear in clause } i \\ 1110 & \text{if variable } j \text{ occurs positively in clause } i \\ 1000 & \text{if variable } j \text{ occurs negatively in clause } i \end{cases}$$

and specify $F(i, 1)F(i, 2)\cdots F(i, n) \in S_0$.

Claim: ϕ is satisfiable if and only if there is a regular expression of length $\leq 5n$ which is compatible with $S(\phi)$.

Proof: (\Rightarrow) Suppose ϕ is satisfiable. Let $A: \{1, 2, \dots, n\} \rightarrow \{0, 1\}$ be an assignment of truth values to the variables of ϕ which satisfies ϕ . For each $j = 1, 2, \dots, n$ let

$$E(j) = \begin{cases} (1)^*000 & \text{if } A(j) = 1 \\ 111(0)^* & \text{if } A(j) = 0 \end{cases}$$

and $E = E_1E_2\cdots E_n$. Then $|E| = 5n$. To see that E is compatible with $S(\phi)$ note:

1'. $q \in L(E)$

2'. If $q = rst$ and s contains both 0's and 1's then $rsst \notin L((1^*0^*)^n)$ but $L(E) \subseteq L((1^*0^*)^n)$, so $rsst \notin L(E)$.

3'. $(111000)^{j-1}10(111000)^{n-j} \notin L(E)$ because $10 \notin L(E_j)$ for each $j = 1, 2, \dots, n$.

4'. Suppose to the contrary that for some i between 1 and m we have $F(i, 1)F(i, 2)\cdots F(i, n) \in L(E)$. Then for each j , if variable j occurs positively in clause i , then $F(i, j) = 1110$, so we must have $E(j) = 111(0)^*$, and therefore $A(j) = 0$. Similarly, if variable j occurs negatively in clause i , we conclude that $A(j) = 1$. Hence the assignment A falsifies clause i and therefore cannot satisfy

ϕ , contradicting our choice of A .

(\Leftarrow) Conversely, suppose that there is some expression of length $\leq 5n$ which is compatible with $S(\phi)$. We will show that it must have essentially the form of the expression E constructed above, derive from this an assignment of truth-values, and show that this assignment must satisfy ϕ .

Let E be a shortest regular expression which is compatible with $S(\phi)$. Then by hypothesis, $|E| \leq 5n$. We use the associativity of concatenation to rewrite E as $E \equiv F_1 F_2 \cdots F_k$ for some $k \geq 1$, where each F_i is not itself a concatenation. Since $q \in L(E)$, we may write $q = q_1 q_2 \cdots q_k$ where $q_i \in L(F_i)$ for $i = 1, 2, \dots, k$. What possibilities do we have for F_i ?

1. $F_i = \emptyset$. This is impossible, because otherwise $L(E) = \emptyset$, and $q \notin \emptyset$.

2. $F_i = \underline{\Lambda}$. This is also impossible, because either $k > 1$, in which case dropping F_i from E produces a shorter expression which is still compatible with $S(\phi)$, contradicting our choice of E as shortest such, or $k = 1$, in which case $E = \underline{\Lambda}$, contradicting the requirement that $q \in L(E)$.

3. $F_i = (G_i \vee H_i)$ for some expressions G_i and H_i . This is likewise impossible, for $q_i \in L(F_i) \Rightarrow q_i \in L(G_i)$ or $q_i \in L(H_i)$, and if $q_i \in L(G_i)$, then replacing F_i by G_i in E will produce a shorter expression which is still compatible with $S(\phi)$, contradicting our choice of E ; similarly if $q_i \in L(H_i)$.

The only remaining possibilities for each F_i are

(i) $F_i = 0$ or $F_i = 1$

or (ii) $F_i = (\hat{F}_i)^*$ for some expression \hat{F}_i .

In either case, we know that q_i cannot contain both 0's and 1's; this is clear for case (i), and in case (ii) we will have $q_1 q_2 \cdots q_i^2 \cdots q_k \in L(E)$, so q_i cannot contain both 0's and 1's by condition #2 of $S(\phi)$.

Thus we may re-associate the concatenations in E and claim that $E \equiv G_1 H_1 G_2 H_2 \cdots G_n H_n$ where $111 \in L(G_j)$ and $000 \in L(H_j)$ for $j = 1, 2, \dots, n$. Since E is a shortest expression compatible with $S(\phi)$, we know that $|G_j|, |H_j| \leq 3$ for $j = 1, 2, \dots, n$, for otherwise we could produce a shorter compatible expression by replacing G_j by 111 in E (or H_j by 000). An enumeration of the expressions of length ≤ 2 shows that the only expression of length ≤ 2 which generates the string 111 is $(1)^*$, and similarly for 000 .

But we cannot have both $G_j = (1)^*$ and $H_j = (0)^*$ for any j , by condition #3 of $S(\phi)$, so $|G_j H_j| \geq 5$ for $j = 1, 2, \dots, n$. The only possible way to attain $|E| = 5n$ is to have for each j between 1 and n either $G_j = (1)^*$ or $H_j = (0)^*$ but not both. We now define an assignment A :

$$A(j) = \begin{cases} 1 & \text{if } G_j = (1)^* \\ 0 & \text{if } H_j = (0)^* \end{cases} .$$

To see that A satisfies ϕ , assume to the contrary that A falsifies clause i for some i between 1 and m . Then for each $j = 1, 2, \dots, n$, if variable j occurs positively in clause i , we must have $A(j) = 0$. Thus $H_j = (0)^*$, so $1110 \in L(G_j H_j)$. Similarly, if variable j occurs negatively in clause i , then $A(j) = 1$, so $1000 \in L(G_j H_j)$. In any case, $111000 \in L(G_j H_j)$, so we have $F(i, j) \in L(G_j H_j)$ for each j , and $F(i, 1)F(i, 2) \cdots F(i, n) \in L(E)$, contradicting condition #4 of $S(\phi)$. Hence A must satisfy ϕ , and we

conclude that ϕ is satisfiable.

Clearly, $S(\phi)$ may be constructed from ϕ in polynomial time, so Theorem 2 is proved. \square

In the proof of Theorem 2 above, if ϕ is unsatisfiable, then there will be no expression of length $\leq 5n$ which is compatible with $S(\phi)$, but if ϕ is satisfiable, then some expression from the set

$$E_n = \{E \mid E = E_1 E_2 \cdots E_n \text{ where each } E_i \text{ is either } (1)^*000 \text{ or } 111(0)^*\}$$

will be of length $= 5n$ and will be compatible with $S(\phi)$. Thus we get

Corollary 2.1: If E is any set of regular expression such that $E_n \subseteq E$ for all $n \geq 1$, then finding a good-guess from E for an arbitrary sample is an NP-hard problem.

Corollary 2.2: The set \mathcal{D} constructed in Section 3.3 is hard-to-infer.

Corollary 2.3: The set $\langle \{0,1\}, \cdot, * \rangle$ of regular expressions over $\{0,1\}$ involving only the operators \cdot and $*$ (optionally: of star-height ≤ 1) is hard-to-infer.

Corollary 2.3 suggests that we consider the set $\langle \{0,1\}, \cdot, \vee \rangle$ of expressions over $\{0,1\}$ involving only the operators \cdot and \vee , all of whose members denote finite sets of strings. We have

Theorem 3: The set $\langle \{0,1\}, \cdot, \vee \rangle$ is hard-to-infer.

Proof: The proof of this theorem is rather lengthy and will be given in outline only. The proof is by a polynomial-time reduction of

the SATISFIABILITY of propositional formulas in conjunctive normal form to the problem of finding good-guesses from $\langle\{0,1\}, \cdot, \vee\rangle$ for arbitrary samples.

Lemma 3.1: For the sample $T = \langle T_1, T_0 \rangle$ given by

<u>T₁</u>	<u>T₀</u>
1000	0001
1010	0011
1100	0101
1110	0111
1101	
0100	

the only expressions E such that $|E| \leq 15$ and $L(E) \subseteq L((0 \vee 1)^4)$ are (up to associativity of \vee and \cdot and commutativity of \vee):

- a) $E_1 = 1(1 \vee 0)(1 \vee 0)(1 \vee 0) \vee 0100$,
 and b) $E_0 = (1 \vee 0)(1 \vee 0)(1 \vee 0)0 \vee 1101$.

The proof of this lemma is a long and painstaking case-analysis, see Appendix. Note that $L(E_1) \neq L(E_0)$, in particular, $1111 \in L(E_1) - L(E_0)$ and $0000 \in L(E_0) - L(E_1)$. This provides the basic binary choice for the reduction.

Lemma 3.2: For any $k \geq 58$, if we specify the sample

$U = \langle U_1, U_0 \rangle$:

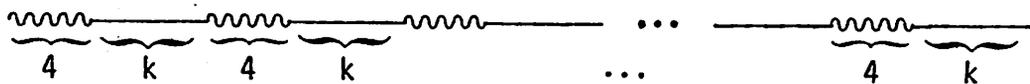
1. $(1000 \cdot 1^k)^j \begin{pmatrix} 1000 \\ 1010 \\ 1100 \\ 1110 \\ 1101 \\ 0100 \end{pmatrix} 1^k \cdot (1000 \cdot 1^k)^{n-j-1} \in U_1$ for $j = 0, 1, \dots, n-1$.
2. $(1000 \cdot 1^k)^j \begin{pmatrix} 0001 \\ 0011 \\ 0101 \\ 0111 \end{pmatrix} 1^k \cdot (1000 \cdot 1^k)^{n-j-1} \in U_0$ for $j = 0, 1, \dots, n-1$.

3. Whenever $x = rst$ and $y = r's't'$ are (possibly equal) strings from #1 above and $|s| \neq |s'|$, then we specify $rs't \in U_0$.

Then the only expressions of length $\leq (15+k)n$ which are compatible with U are (up to associativity of \cdot and \vee , and commutativity of \vee):

$$V = \{E \mid E = F_1 \cdot 1^k \cdot F_2 \cdot 1^k \cdots F_n \cdot 1^k \text{ where each } F_i \text{ is either } E_1 \text{ or } E_0 \text{ from Lemma 3.1}\}$$

Proof: Let E be an expression of minimum length which is compatible with U . We will show that $|E| = (15+k)n$ and E is equivalent to some element of V . It is possible to argue that all of the sub-expressions of E are "essential", that is, if F is a sub-expression of E , then there exists a string $s \in U_1$ such that $s = tuv$ and $u \in L(F)$. Then it can be argued that condition #3 of U will force $L(E) \subseteq L((0 \vee 1)^{(4+k)n})$. Now we rewrite E as an unassociated concatenation $E \equiv G_1 G_2 \cdots G_r$ where $r \geq 1$ and each G_i is not itself a concatenation. Clearly, for each i , $L(G_i) = \{\Lambda\}$ is impossible, and $L(G_i)$ must contain strings of only one length. The only possibilities for G_i are $G_i = 1$ and $G_i = (H_i \vee K_i)$ for some expressions H_i and K_i . Since for each i , $L(G_i)$ must contain strings of only one length, G_i "covers" the same positions in each $s \in U_1$. We represent all $(4+k)n$ positions in the strings of U_1 pictorially:

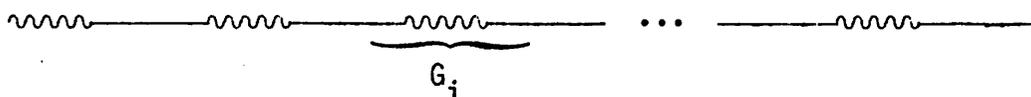


where the wavy lines represent positions which may be either 0 or 1 depending on which $s \in U_1$ we examine, and the straight lines represent

positions which contain 1's in every $s \in U_1$. The positions indicated by straight lines will be called "invariant positions".

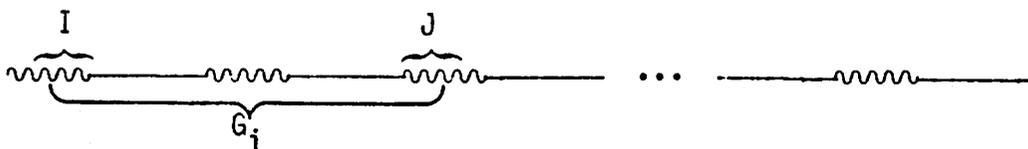
We will argue that if $G_i = (H_i \vee K_i)$, then G_i does not cover any of the invariant positions. We have essentially two possibilities:

1. One end of the positions covered by G_i is an "invariant position", e.g.,



In this case we argue that we may rewrite $H_i \equiv H'_i \cdot 1$ and $K_i \equiv K'_i \cdot 1$, where $|H'_i| < |H_i|$ and $|K'_i| < |K_i|$, and thus replace G_i in E by $(H'_i \vee K'_i) \cdot 1$ to get a shorter expression which is compatible with U , contradicting our choice of E as of minimum length. The argument in case the leftmost position covered by G_i is invariant is similar.

2. G_i includes one or more whole groups of invariant positions, but neither end-position in G_i is invariant:



So suppose that G_i includes m such groups of invariant positions, where $m \geq 1$. Then we argue that each of H_i and K_i must be of length $\geq km$, so that $|G_i| > 2mk$. But in this case, we may replace G_i in E by the expression $G'_i = L_i \cdot 1^k \cdot (M \cdot 1^k)^{m-1} \cdot N_i$, where L_i is the disjunction of all those strings appearing in the interval I (see picture above) in all of the strings $s \in U_1$, N_i is the disjunction of all those strings appearing in interval J in all of the strings $s \in U_1$, and $M = (1000 \vee 1010 \vee 1100 \vee 1110 \vee 1101 \vee 0100)$. But then

$|G_i| \leq mk + (m+1)29$ (since $|L_i|, |M|, |N_i| \leq 29$), so $|G_i| > |G_i'|$ because $m \geq 1, k \geq 58$. Thus, replacing G_i in E by G_i' produces a shorter expression compatible with U , a contradiction.

So we conclude that if $G_i = (H_i \vee K_k)$, then no invariant positions are covered by G_i . Thus we may reassociate the concatenations and claim that

$$E \equiv F_1 \cdot 1^k \cdot F_2 \cdot 1^k \cdots F_n \cdot 1^k$$

where each F_j is an expression of minimum length such that

- i) $L(F_j) \subseteq L((0 \vee 1)^4)$
- ii) F_j is compatible with the sample T of Lemma 3.1.

Now we apply Lemma 3.1 to conclude that each F_j must be (up to associativity and commutativity) either $E_1 = 1(1 \vee 0)(1 \vee 0)(1 \vee 0) \vee 0100$ or $E_0 = (1 \vee 0)(1 \vee 0)(1 \vee 0)0 \vee 1101$, as claimed. \square

The sample U of Lemma 3.2 gives us n independent binary choices; adding more conditions to encode a SATISFIABILITY problem is relatively easy. Let ϕ be a propositional formula in conjunctive normal form with clauses numbered 1 through m and variables numbered 1 through n . Let

$$F(i,j) = \begin{cases} 1000 & \text{if variable } j \text{ does not occur in clause } i \\ 0000 & \text{if variable } j \text{ occurs positively in clause } i \\ 1111 & \text{if variable } j \text{ occurs negatively in clause } i \end{cases}$$

To the conditions #1-3 of the sample U of Lemma 3.2 we add the condition

- 4. For each $i = 1, 2, \dots, m$, specify

$$F(i,1) \cdot 1^k \cdot F(i,2) \cdot 1^k \cdots F(i,n) \cdot 1^k \in U_0$$

Call the augmented sample $S(\phi) = \langle S_1, S_0 \rangle$.

Claim: ϕ is satisfiable if and only if there exists an expression of length $(15+k)n$ which is consistent with $S(\phi)$.

Proof: (\Rightarrow) If ϕ is satisfiable, choose $A: \{1, 2, \dots, n\} \rightarrow \{0, 1\}$ to be an assignment of truth-values which satisfies ϕ . Then for each $j = 1, 2, \dots, n$, let

$$G_j = \begin{cases} (1(1 \vee 0)(1 \vee 0)(1 \vee 0) \vee 0100) & \text{if } A(j) = 1 \\ ((1 \vee 0)(1 \vee 0)(1 \vee 0)0 \vee 1101) & \text{if } A(j) = 0 \end{cases}$$

and let

$$E = G_1 \cdot 1^k \cdot G_2 \cdot 1^k \dots G_n \cdot 1^k .$$

Clearly, E is consistent with conditions #1-3 of Lemma 3.2 and $|E| = (15+k)n$. To see that E is consistent with condition #4 (above), assume to the contrary that for some i between 1 and m we have:

$$F(i, 1) \cdot 1^k \cdot F(i, 2) \cdot 1^k \dots F(i, n) \cdot 1^k \in L(E) .$$

If variable j occurs positively in clause i , then $F(i, j) = 0000$, so we must have $G_j = (1 \vee 0)(1 \vee 0)(1 \vee 0)0 \vee 1101$ and thus $A(j) = 0$. Similarly, if variable j occurs negatively in clause i , then $A(j) = 1$. Hence the assignment A falsifies clause i , contradicting our choice of A .

(\Leftarrow) Assume that there is an expression of length $\leq (15+k)n$ which is compatible with $S(\phi)$. Then in particular, such an expression must be consistent with the conditions of Lemma 3.2, so it must be (up to associativity and commutativity) be of the form

$$G_1 \cdot 1^k \cdot G_2 \cdot 1^k \dots G_n \cdot 1^k$$

where each G_j is either $E_1 = 1(1 \vee 0)(1 \vee 0)(1 \vee 0) \vee 0100$ or $E_0 = (1 \vee 0)(1 \vee 0)(1 \vee 0)0 \vee 1101$. Define

$$A(j) = \begin{cases} 1 & \text{if } G_j = E_1 \\ 0 & \text{if } G_j = E_0 \end{cases}$$

for $j = 1, 2, \dots, n$. To see that this assignment satisfies ϕ , assume to the contrary that there is some clause i of ϕ which is falsified by A . Then for each variable j , $1000 \in L(G_j)$. Also, if j occurs positively in clause i , $A(j) = 0$, so $0000 \in L(G_j)$. Similarly, if variable j occurs negatively in clause i , then $1111 \in L(G_j)$.

Thus, $F(i, j) \in L(G_j)$ for $j = 1, 2, \dots, n$, so

$$F(i, 1) \cdot 1^k \cdot F(i, 2) \cdot 1^k \cdots F(i, n) \cdot 1^k \in L(E),$$

contradicting condition #4.

The sample $S(\phi)$ can be constructed from ϕ in polynomial time, so we conclude that the set $\langle \{0, 1\}, \cdot, \vee \rangle$ is hard-to-infer. \square

Let $n \geq 1$ and $k \geq 58$, and

$$E_n = \{E \mid E = G_1 \cdot 1^k \cdot G_2 \cdot 1^k \cdots G_n \cdot 1^k \text{ where each } G_j \text{ is either} \\ (1(1 \vee 0)(1 \vee 0)(1 \vee 0) \vee 0100) \text{ or} \\ ((1 \vee 0)(1 \vee 0)(1 \vee 0)0 \vee 1101)\}$$

Then an examination of the proof of Theorem 3 yields:

Corollary 3.1: If E is any set of regular expressions such that $E_n \subseteq E$ for all $n \geq 1$, then E is hard-to-infer.

The last result of this section is related to the class $\langle \{1\}, \vee, * \rangle$, but it is a little different in character from the preceding two

theorems; the reduction given is not difficult.

Theorem 4: Let

$$E = \{E \mid E = (1^{x_1})^* \vee (1^{x_2})^* \vee \dots \vee (1^{x_k})^* \text{ where } k \text{ and } x_1, x_2, \dots, x_k \text{ are positive integers}\} .$$

Then it is NP-hard to find an expression from E with a minimum number of disjuncts (not minimum total length) compatible with an arbitrary sample.

Proof: The proof is by a reduction of the SET COVERING problem to the problem in question.

SET COVERING: Given an $m \times n$ matrix M of 0's and 1's, and an integer $k \geq 1$, to decide whether there is a collection C of k or fewer of the columns of M such that every row of M has a 1 in common with some column from C . (C is called a cover of M .)

In [Karp (1972)], it is shown that SET COVERING is NP-complete; examination of the reduction given therein shows that the problem remains NP-complete even if we restrict the input M to contain exactly two 1's per row.

Let M be an $m \times n$ matrix of 0's and 1's with exactly two 1's per row. We enumerate the first n prime numbers: p_1, p_2, \dots, p_n , and define

$$r_i = p_{a(i)} \times p_{b(i)} ,$$

where $a(i)$ and $b(i)$ are the column-numbers of the two 1's in row i , for $i = 1, 2, \dots, m$.

We specify a sample, $S(M) = \langle S_1, S_0 \rangle$ by:

$$1. \quad 1^{r_i} \in S_1 \text{ for } i = 1, 2, \dots, m$$

2. $1 \in S_0$

Claim: There is a cover of M by k or fewer of its columns if and only if there is an expression $E \in E$ with k or fewer disjuncts which is compatible with $S(M)$.

Proof: (\Rightarrow) Suppose $C = \{j_1, j_2, \dots, j_k\}$ is a cover of M . We let

$$E = (1^{x_1})^* \vee (1^{x_2})^* \vee \dots \vee (1^{x_k})^*$$

where $x_s = p_{j_s}$ for $s = 1, 2, \dots, k$. Then $E \in E$ and has k disjuncts. Clearly, $1 \notin L(E)$. Also, for any $i = 1, 2, \dots, m$, row i has a 1 in common with some column $j_s \in C$. Thus, $p_{j_s} | r_i$, so $x_s | r_i$ and $1^{r_i} \in L(E)$. Thus E is compatible with $S(M)$.

(\Leftarrow) Suppose that there is some expression from E with k or fewer disjuncts which is consistent with $S(M)$. Let E be such an expression, where

$$E = (1^{x_1})^* \vee (1^{x_2})^* \vee \dots \vee (1^{x_k})^*$$

Since $1 \notin L(E)$, $x_s > 1$ for $s = 1, 2, \dots, k$. Let

$$C = \{j \mid 1 \leq j \leq n \text{ and } p_j \text{ is the least prime divisor of } x_s \text{ for some } s, 1 \leq s \leq k\}$$

Then C contains k or fewer elements. To see that C is a cover of M , let i be given, $1 \leq i \leq m$. Since $1^{r_i} \in L(E)$, we must have $x_s | r_i$ for some s , $1 \leq s \leq k$. But $r_i = p_{a(i)} \times p_{b(i)}$, and $x_s > 1$, so the least prime dividing x_s must be either $p_{a(i)}$ or $p_{b(i)}$, so either $a(i)$ or $b(i)$ is in C . Hence row i has a 1 in common with some column from C , so C is a cover of M .

The construction of $S(M)$ from M may be done in polynomial time, so Theorem 4 is proved. \square

3.5 Inference of "Most Likely" Expressions

In this section we briefly consider the computational tractability of an alternative definition of a good guess for a given sample. This new definition follows the approach of Horning in his study of the inference of stochastic grammars [Horning (1969)]. We choose a function P which assigns a probability, $P(E)$, to each regular expression E , and a probability, $P(T|E)$, of obtaining a particular set of strings T from a particular expression E . Then, for a given sample $S = \langle S_1, S_0 \rangle$, we seek an expression which is compatible with S and is "most likely" to have generated S_1 , i.e., which maximizes $P(E|S_1)$. Since

$$P(E|S_1) = \frac{P(E)P(S_1|E)}{P(S_1)} \quad (\text{Bayes' rule})$$

and $P(S_1)$ is fixed by S , it is sufficient to find an expression which maximizes $P(E)P(S_1|E)$.

The model that we choose to establish the distributions is fairly simple, for the convenience of the proofs to follow, but again the general approach will probably work under some variation in the definitions. We assume that symbols are drawn with equal probability from the set $\{0, 1, v, (,), *, _, \$\}$ until a "\$" is encountered. If the string drawn, up to the "\$", is a legal, fully-parenthesized regular expression, say E (concatenation being indicated by juxtaposition), then we begin randomly generating strings in the language of E , as follows: we trace through E constructing a string, and whenever we have a

choice (FVG) as a subexpression of E, we choose to trace either F or G, each with probability 1/2. Whenever we encounter (F)* as a subexpression of E, we choose either to skip F or to trace it (each with probability 1/2), and if we choose to trace F, then we return to the beginning of (F)* (if and) when we finish tracing F. If and when we finish generating one string in L(E) in this way, we begin again generating another, and so on, ad infinitum. (Rules for calculating $P(S_1|E)$ will be given below.)

We define random variables $\bar{E}, \bar{s}_1, \bar{s}_2, \bar{s}_3, \dots$ as follows:

$$\bar{E} = \begin{cases} \text{the expression } E \text{ drawn in the event described above, if any} \\ \# \text{ otherwise,} \end{cases}$$

$$\bar{s}_i = \begin{cases} \text{the } i^{\text{th}} \text{ string generated from } E \text{ in the event described above,} \\ \text{if any} \\ \# \text{ otherwise.} \end{cases}$$

Then $P(E)$ will stand for $P(\bar{E}=E)$, and will be $(1/8)^{n+1}$ if E is a string from $\{0,1,v,(,),*,\underline{\Lambda}\}^*$ of length n which is a legal, fully-parenthesized regular expression. $P(S_1|E)$ will stand for

$$P(\bar{s}_1 = s_1 \& \bar{s}_2 = s_2 \& \dots \& \bar{s}_t = s_t | \bar{E} = E), \text{ where } S_1 = \langle s_1, s_2, \dots, s_t \rangle.$$

(In this section we consider samples consisting of pairs of ordered finite sequences of strings, allowing multiple occurrences of strings.)

$P(s|E)$ will note $P(\bar{s}_1 = s | \bar{E} = E)$, for any string $s \in \{0,1\}^*$.

Clearly, $P(S_1|E) = \prod_{i=1}^t P(s_i|E)$, and we may calculate $P(s|E)$ inductively as follows:

1. if $a = 0, 1, \underline{\Lambda}, (E)^0$:

$$P(s|a) = \begin{cases} 1 & \text{if } L(a) = \{s\} \\ 0 & \text{otherwise.} \end{cases}$$

2. concatenation:

$$P(s|EF) = \sum_{s=s_1s_2} P(s_1|E)P(s_2|F)$$

3. union:

$$P(s|(E \vee F)) = \frac{1}{2} P(s|E) + \frac{1}{2} P(s|F)$$

4. star:

$$P(s|(E)^*) = \sum_{i=0}^{\infty} (1/2)^{i+1} P(s|(E)^i)$$

Example:

$$P((0 \vee 1)(01)^*1) = \left(\frac{1}{8}\right)^{12}$$

$$\begin{aligned} P(001011|(0 \vee 1)(01)^*1) &= P(0|(0 \vee 1))P(0101|(01)^*)P(1|1) \\ &= \left(\frac{1}{2}\right)\left(\frac{1}{2}\right)^3 1 \\ &= \left(\frac{1}{2}\right)^4 \end{aligned}$$

Definition: If S_1 is a set of strings, s is a string, and E is a regular expression we define

$$L(E, S_1) = P(E)P(S_1|E)$$

and

$$L(E, s) = P(E)P(s|E)$$

If $L(E, s) > L(F, s)$ then we say that E is a more likely explanation of s than F ; similarly for sets of strings.

Example: Let $m \geq 1$ be an integer. Then

$$\begin{aligned} L(0^{2m}, 0^{2m}) &= \left(\frac{1}{8}\right)^{2m+1} (1) = \left(\frac{1}{2}\right)^{6m+3} \\ L((0^m)^*, 0^{2m}) &= \left(\frac{1}{8}\right)^{m+4} \left(\frac{1}{2}\right)^3 = \left(\frac{1}{2}\right)^{3m+15} \end{aligned}$$

Thus, for $m \geq 5$, $(0^m)^*$ is a more likely explanation of the string 0^{2m} than the expression 0^{2m} .

This choice, between $(0^m)^*$ and 0^{2m} , will be used instead of that between $(0)^*$ and 000 in the proof of Theorem 2 to yield:

Theorem 5: The problem of finding a most likely explanation of an arbitrary sample S , that is, of finding an expression E which is compatible with S and maximizes $L(E, S_1)$, is NP-hard.

Proof: First we need:

Definition: If E is a regular expression, let

$$f(E) = \begin{cases} \text{the least positive integer } k \text{ such that } 1^k \in L(E), & \text{if any} \\ 0 & \text{otherwise} \end{cases}$$

(so, for example, $f(1(1 \vee 0)^*1) = 2$.)

Lemma 5.1: For every regular expression E , $P(E) \leq (\frac{1}{8})^{f(E)+1}$, that is, E must contain at least $f(E)$ symbols, and if $P(E) = (\frac{1}{8})^{f(E)+1}$, then we must have $E = (1)^{f(E)}$.

This lemma is not difficult to prove by induction on regular expressions as they are defined for this section.

Now we fix $n \geq 1$ and $k \geq 5$, and specify a sample $S = \langle S_1, S_0 \rangle$ as follows:

1. $q = (1^{2k}0^{2k})^n \in S_1$
2. If $q = rst$ and s contains both 0's and 1's, then $rsst \in S_0$.

3. $(1^{2k}0^{2k})^j r_0^s (1^{2k}0^{2k})^{n-j-1} \in S_0$ for all $j = 0, 1, \dots, n-1$
and $1 \leq r, s \leq 2k$ except $\langle r, s \rangle = \langle 2k, 2k \rangle, \langle 2k, k \rangle, \langle k, 2k \rangle$.

Then:

Lemma 5.2: The only expressions E which are compatible with S and which have $L(E, S_1) \geq \left(\frac{1}{8}\right)^{(3k+4)n+1}$ are from the set

$$V = \{E \mid E = E_1 E_2 \cdots E_n \text{ where each } E_j \text{ is either } (1^k)^* 0^{2k} \text{ or } 1^{2k} (0^k)^*\}$$

(up to associativity of concatenation).

Proof: Let E be an expression which maximizes $L(E, S_1)$ and which is compatible with S . We will show that $L(E, S_1) = \left(\frac{1}{8}\right)^{(3k+4)n+1}$ and E can be put in the form of some expression from V by reassociating concatenations. Let $E \equiv F_1 F_2 \cdots F_r$ for some $r \geq 1$, where each F_i is not itself a concatenation, and $q = q_1 q_2 \cdots q_r$ where each $q_i \in L(F_i)$. Then for each F_i we have:

1. $F_i = \underline{\Lambda}$ is impossible, for $E \neq \underline{\Lambda}$ because $q \in L(E)$, and then dropping F_i from E would produce an expression strictly more likely for S_1 than E , contradicting our choice of E .

2. $F_i = (G_i \vee H_i)$. To see that this is impossible, let E' be E with F_i replaced by G_i and E'' be E with F_i replaced by H_i .

Then $P(q|E) = \left(\frac{1}{2}\right)(P(q|E') + P(q|E''))$. Thus either $P(q|E') \geq P(q|E)$ or $P(q|E'') \geq P(q|E)$. But if $P(q|E') \geq P(q|E)$, then since E' is a strictly shorter expression than E , $P(E') > P(E)$, so $L(E', S_1) = L(E', q) > L(E, q) = L(E, S_1)$, and E' will be compatible with S , which contradicts our choice of E . The case of $P(q|E'') \geq P(q|E)$ is similar.

Thus we conclude that $F_i = 0, 1, \text{ or } (G_i)^*$ for some expression G_i , and in any case, q_i must contain only 0's or only 1's for each $i = 1, 2, \dots, r$, by condition 2 of the sample S and compatibility of E with S . Further, if $q = rst$ where s contains both 0's and 1's, we have

$$P(r|F_1 F_2 \cdots F_{i-1})P(s|F_i)P(t|F_{i+1} \cdots F_r) = 0$$

for all $i = 1, 2, \dots, r$. If not, then in particular, $r \in L(F_1 F_2 \cdots F_{i-1})$, $s \in L(F_i)$, and $t \in L(F_{i+1} \cdots F_r)$ and F_i must be $(G_i)^*$, so $rsst \in L(E)$, contradicting condition 2 of the sample S .

Thus we may reassociate the concatenations in E :

$$E \equiv K_1 L_1 K_2 L_2 \cdots K_n L_n$$

where $1^{2k} \in L(K_j)$ and $0^{2k} \in L(L_j)$ for $j = 1, 2, \dots, n$, and also

$$P(q|E) = \prod_{j=1}^n P(1^{2k}|K_j)P(0^{2k}|L_j)$$

so

$$(*) \quad \left(\frac{1}{8}\right)^{2n-1} L(E, q) = \prod_{j=1}^n L(K_j, 1^{2k}) L(L_j, 0^{2k}) .$$

Then from condition 3 of the sample S , we argue that for each $j = 1, 2, \dots, n$, either

1. 1^{2k} is the smallest nonnull string of 1's in $L(K_j)$, so that $P(K_j) \leq \left(\frac{1}{8}\right)^{2k+1}$ by Lemma 5.1,
- or 2. 0^{2k} is the smallest nonnull string of 0's in $L(L_j)$, so that $P(L_j) \leq \left(\frac{1}{8}\right)^{2k+1}$, similarly.

Further, if 1^{2k} is not the smallest nonnull string of 1's in $L(K_j)$, then by condition 3 of S , 1^k must be, and we will argue

that K_j must be $(1^k)^*$.

We know that $K_j = F_s F_{s+1} \cdots F_t$ for some positive integers $s \leq t$, where each F_u is either 1 or $(G_u)^*$. If $F_u = 1$ for all u between s and t , then K_j cannot generate both 1^{2k} and 1^k , so we know that $F_u = (G_u)^*$ for at least one u between s and t . Then we may argue that the smallest nonnull string of 1's generated by G_u must be 1^k , so either $G_u = 1^k$, or $P(G_u) \leq (\frac{1}{8})^{k+2}$, by Lemma 5.1. In the latter case, $P(K_j) \leq (\frac{1}{8})^{k+5}$, because G_u is of length $\geq k+1$, and K_j is at least 3 symbols longer than G_u . But $P(1^{2k} | K_j) < 1$, because both 1^k and 1^{2k} are in $L(K_j)$, by hypothesis, so $L(K_j, 1^{2k}) < (\frac{1}{8})^{k+5} = L((1^k)^*, 1^{2k})$, which in combination with equality (*) contradicts our choice of E as maximizing $L(E, q)$. Thus the only way to attain $L(K_j, 1^{2k}) = (\frac{1}{8})^{k+5}$ is by choosing $K_j = (1^k)^*$; similarly for L_j . Hence the only maximum likelihood possibilities are:

$$K_j L_j = (1^k)^* 0^{2k} \text{ or } 1^{2k} (0^k)^*$$

for $j = 1, 2, \dots, n$, which proves Lemma 5.2.

Lemma 5.2 gives us the requisite n independent binary choices; encoding a SATISFIABILITY problem is then possible by a straightforward modification of the proof of Theorem 2. This concludes the proof of Theorem 5. □

Discussion

One of the primary motivations for Horning's study of "most likely" inference is that it allows correct inference in the limit with probability 1 from positive information only, in contrast to the non-probabilistic case, in which negative information is required for correct

inference in the limit in general. The construction given above makes very heavy use of the negative part of the sample S to constrain the form of expressions compatible with it. The tractability of finding a most likely expression from positive information only therefore remains an open problem. Perhaps some efficient constructive techniques can be found in this case; the problem seems worthy of further study.

CHAPTER 4

Summary, AcknowledgmentsSummary

We have presented an investigation of the computational tractability of finding a "smallest" description compatible with a given sample of a finite-state language.

In Chapter 1 we explain the relation of this problem to the study of inductive inference. We also briefly consider the effects of the choice of descriptive systems upon the inference made from a given sample; an example is presented in which the language inferred from a sample depends on whether the description is to be given as a deterministic finite-state acceptor or a nondeterministic finite-state acceptor.

In Chapter 2 we consider the problem of finding a deterministic finite-state machine with a minimum number of states which is compatible with a given sample of behaviour. Mark Gold has shown this problem to be NP-hard for arbitrary samples. We restrict the samples to be "uniform-complete" (that is, to contain every input string of length \leq some k , and no others), and demonstrate a simple algorithm to find a smallest compatible machine in polynomial time. We show that relaxing this restriction to allow $O(n^\epsilon)$ unspecified strings in a sample of size n makes the problem NP-hard. We also show that the problem remains NP-hard when restricted to "finite-language" or "definite" automata.

In Chapter 3 we consider the problem of finding a shortest regular expression (using the operators $\{V, \cdot, *\}$) compatible with arbitrary (positive and negative) samples of finite-state languages. The

general problem is shown to be NP-hard, as are the problems restricted to just the operators $\{.,*\}$ and just the operators $\{.,\vee\}$. Some restricted, syntactically-given classes of regular expressions are exhibited in which shortest compatible expressions may be found for arbitrary samples in polynomial time. Finally, we consider a definition of a "most likely" expression compatible with an arbitrary sample, and show that the problem of finding such a "most likely" compatible expression from given data is NP-hard in general. We also suggest reasons why further work on this probabilistic kind of inference seems justified.

Acknowledgments

This work was supported by Grants GJ-35604-X1 and DCR72-03725-A02 of the National Science Foundation. My thanks to Ruth Suzuki for the typing of this dissertation. I would like to thank past and present members of and visitors to the University of California at Berkeley Electrical Engineering and Computer Sciences Department for their generous opposition and encouragement, particularly Professor R.M. Karp, who has been an invaluable source of insight, and Professor M. Blum, whose boundless patience and idealism can not be repaid.

APPENDIX

We present the substance of the case-argument for Lemma 3.1 of Section 3.4.

Definition: A regular expression $E \in \langle \{0,1\}, \cdot, \vee, * \rangle$ will be called k-minimum-uniform-language, hereafter written k-mul, if and only if $k \geq 1$ is an integer and

$$1. \quad \emptyset \neq L(E) \subseteq \{0,1\}^k$$

and 2. E is a shortest regular expression denoting $L(E)$.

Lemma A1: If E is k-mul then either

$$1. \quad E = 0$$

or 2. $E = 1$

or 3. $E = (F \vee G)$ where F and G are k-mul

or 4. $E = (F \cdot G)$ where F is i-mul and G is j-mul, and $i+j = k$.

Consequently, E cannot contain \emptyset , $\underline{\Lambda}$, or $*$.

Lemma A2: If E is k-mul, then to each occurrence of 0 (or 1) in E we may assign a unique position i , $1 \leq i \leq k$, such that this occurrence of 0 (or 1) is used only to generate a 0 (or 1) at position i in the strings of $L(E)$.

Lemma A3: If E is k-mul, then the number of "v"s in E must be at least $\lceil \log_2 |L(E)| \rceil$

Definition: If $S = \{s_1, s_2, \dots, s_m\} \subseteq \{0,1\}^k$ for some $k, m \geq 1$, then define $V(S)$ to be the number of positions in which both 0's and 1's appear in strings of S , that is:

$$V(S) = |\{i \mid \exists x, y, 1 \leq x, y \leq m \text{ such that position } i \text{ of } s_x \text{ is a 0 and position } i \text{ of } s_y \text{ is a 1}\}|$$

Lemma A4: If E is k -mul and $S = \{s_1, s_2, \dots, s_m\} \subseteq L(E)$, then we have the following bound on the length of E :

$$|E| \geq \lceil \log_2 m \rceil + k + V(S) .$$

To see this, note that $\lceil \log_2 m \rceil$ is the minimum number of "V"s required in E (by Lemma A3), k is the minimum number of 0's and 1's required in E to cover all the positions in strings of $L(E)$, and $V(S)$ is the minimum number of additional 0's and 1's required to cover the "variant" positions of $L(E)$.

Lemma A5: If E is 4-mul and $1101, 0100 \in L(E)$ and $0101 \notin L(E)$, then $E \neq F \cdot G$ for all expressions F and G .

Proof: Suppose to the contrary that $E = F \cdot G$ for some expressions F and G . The by Lemma A1, F is i -mul and G is j -mul where $i, j \geq 1$ and $i + j = 4$. We have three cases:

1. F is 1-mul and G is 3-mul. Since $0100 \in L(E) = L(F \cdot G)$, we must have $0 \in L(F)$. Since $1101 \in L(E) = L(F \cdot G)$, $101 \in L(G)$. Thus $0101 \in L(F \cdot G) = L(E)$, contradicting the hypothesis that $0101 \notin L(E)$.

2. F is 2-mul and G is 2-mul. Similarly.

3. F is 3-mul and G is 1-mul. Similarly. \square

Lemma 3.1 (from Section 3.4). If E is a regular expression of minimum length such that

1. $L(E) \subseteq \{0,1\}^4$

and 2. E is compatible with the sample $T = \langle T_1, T_0 \rangle$ where

$\underline{T_1}$	$\underline{T_0}$
1000	0001
1010	0011
1100	0101
1110	0111
1101	
0100	

(that is, $T_1 \subseteq L(E)$ and $T_0 \cap L(E) = \emptyset$) then E must be (up to associativity of \vee and \cdot and commutativity of \vee) one of the two expressions:

$$a) \quad E_1 = 1(1 \vee 0)(1 \vee 0)(1 \vee 0) \vee 0100$$

$$b) \quad E_0 = (1 \vee 0)(1 \vee 0)(1 \vee 0)0 \vee 1101$$

Proof: Observe that E is 4-mul. Also, since E_1 and E_0 satisfy conditions 1 and 2 above, E must be at least as short as they are, i.e., $|E| \leq 15$. Clearly $E \neq 0,1$ and, by Lemma A5, $E \neq F \cdot G$ for any expressions F and G . By Lemma A1 the remaining possibility is $E = (F \vee G)$. By associativity, we may write $E = F_1 \vee F_2 \vee \cdots \vee F_m$, for some $m \geq 2$, where each F_i is 4-mul and is not itself a union of two expressions.

By Lemma A4, $|F_i| \geq 4$ for $i = 1, 2, \dots, m$. Thus if $m \geq 4$, then $|E| \geq 19$, contradicting our choice of E . In the case of $m = 3$, if we are to have $|E| \leq 15$, we must have $4 \leq |F_i| \leq 5$ for $i = 1, 2, 3$. By Lemma A4, this implies that $|L(F_i)| = 1$ for $i = 1, 2, 3$ which in turn implies that $|L(E)| = 3$, contradicting the requirement that $T_1 \subseteq L(E)$. Hence we conclude that we must have $m = 2$, so

$$E = F_1 \vee F_2,$$

where F_1 and F_2 are 4-mul and not themselves unions.

Clearly, F_1 and F_2 must both be concatenations (by Lemma A1), and $0101 \notin L(F_1)$, $0101 \notin L(F_2)$, so we cannot have both 1101 and 0100 in either $L(F_1)$ or $L(F_2)$ (by Lemma A5), but they must both be in $L(E)$, so we assume without loss of generality that

$$1101 \in L(F_1) \quad \text{and} \quad 0100 \in L(F_2) .$$

Now we consider the possibilities for how the strings of T_1 can be generated by the two expressions F_1 and F_2 . For each possibility, the minimum lengths of F_1 and F_2 are calculated according to Lemma A4; the possibilities are tabulated below in Table A. This shows that the only way we can have $|E| \leq 15$ is in cases #1 and #16. We must still argue that this is attainable only by expressions equivalent to E_1 and E_0 . Take case #1: we must have $|F_1| = 10$ and $1101, 1000, 1010, 1100, 1110 \in L(F_1)$. We write $F_1 \equiv G_1 \cdot G_2 \cdots G_m$, $2 \leq m \leq 4$, where each G_i is not itself a concatenation. If G_1 is 2-mul or 3-mul, then we can argue that there will be at least two occurrences of 0 or 1 to cover position 1 in $L(F_1)$, which, with arguments from Lemma A4 will imply that $|F_1| \geq 11$, a contradiction. Thus, G_1 must be 1-mul, and in particular must be = 1. So,

$F_1 \equiv 1 \cdot G_2 \cdots G_m$. We consider cases:

1. $m = 2$, so G_2 is 3-mul, and is not a concatenation, by hypothesis. So $G_2 = (H_1 \vee H_2)$, where each of H_1 and H_2 are 3-mul. We have $101, 000, 010, 100, 110 \in L(G_2)$. Cases (without loss of generality):

a. at least one of these is in $L(H_1)$ and at least the other four are in $L(H_2)$. Thus $|H_1| \geq 3$ and since any four of the five strings 101, 000, 010, 100, 110 differ in at least two positions, by Lemma A4 we have $|H_1| \geq 7$, so $|G_2| \geq 11$, and $|F_1| \geq 12$,

TABLE A

	<u>$L(F_1)$ contains at least</u>	<u>$F_1 \geq$</u>	<u>$L(F_2)$ contains at least</u>	<u>$F_2 \geq$</u>	<u>$E \geq$</u>
1.	1101,1000,1010,1100,1110	10	0100	4	15
2.	1101,1010,1100,1110	9	0100,1000	7	17
3.	1101,1000,1100,1110	9	0100,1010	8	18
4.	1101,1000,1010,1110	9	0100,1100	6	16
5.	1101,1000,1010,1100	9	0100,1110	7	17
6.	1101,1100,1110	8	0100,1000,1010	9	18
7.	1101,1010,1110	9	0100,1000,1100	8	18
8.	1101,1010,1100	9	0100,1000,1110	9	19
9.	1101,1000,1110	9	0100,1010,1100	9	19
10.	1101,1000,1100	8	0100,1010,1110	9	18
11.	1101,1000,1010	9	0100,1100,1110	8	18
12.	1101,1000	7	0100,1010,1100,1110	9	17
13.	1101,1010	8	0100,1000,1100,1110	9	18
14.	1101,1100	6	0100,1000,1010,1110	9	16
15.	1101,1110	7	0100,1000,1010,1100	9	17
16.	1101	4	0100,1000,1010,1100,1110	10	15

a contradiction.

b. at least two of the five strings is in $L(H_1)$ and at least the other three are in $L(H_2)$. Since any two of the five strings differ in at least one place, $|H_1| \geq 5$. And similarly $|H_2| \geq 5$, so $|G_2| \geq 11$, and $|F_1| \geq 12$, a contradiction.

2. $m = 3$, so we have the two cases:

a. G_2 is 2-mul and G_3 is 1-mul. Thus, $0, 1 \in L(G_3)$, so $|G_3| \geq 3$. And $00, 01, 10, 11 \in L(G_2)$, where $G_2 = (H_1 \vee H_2)$.

Still further cases:

i. $L(H_1)$ contains one of $00, 01, 10, 11$ and $L(H_2)$ contains the other three. Thus $|H_1| \geq 2$, and $|H_2| \geq 6$, so $|G_2| \geq 9$, and $|F_1| \geq 13$, a contradiction.

ii. $L(H_1)$ contains two of $00, 01, 10, 11$ and $L(H_2)$ contains the other two. Thus $|H_1| \geq 4$, $|H_2| \geq 4$, $|G_2| \geq 9$, and $|F_1| \geq 13$, a contradiction.

b. G_2 is 1-mul and G_3 is 2-mul. Thus, $0, 1 \in L(G_2)$, $00, 01, 10 \in L(G_3)$, and $G_3 = (H_1 \vee H_2)$. So $|G_2| \geq 3$. Without loss of generality, $L(H_1)$ must contain one of $00, 01, 10$ and $L(H_2)$ the other two. Thus $|H_1| \geq 2$ and $|H_2| \geq 4$, so $|G_3| \geq 7$, and $|F_1| \geq 11$, a contradiction.

Hence, we may conclude that $m = 4$, so $F_1 \equiv 1 \cdot G_2 \cdot G_3 \cdot G_4$, where $0, 1 \in L(G_i)$ and G_i is 1-mul, for $i = 2, 3, 4$. Hence, we must have $G_i \equiv (0 \vee 1)$ for $i = 2, 3, 4$. Thus we have

$$E \equiv 1 \cdot (1 \vee 0)(1 \vee 0)(1 \vee 0) \vee 0100$$

as claimed. The case of alternative #16 in the table implying that $E \equiv 1101 \vee (1 \vee 0)(1 \vee 0)(1 \vee 0)0$ is argued in a parallel fashion, to conclude the proof of Lemma 3.1 (Section 3.4). \square

13

REFERENCES

- Aho, A.V., Hopcroft, J.E., and Ullman, J.D. (1974). The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts.
- Angluin, D. (1976). "Remarks on the difficulty of finding a minimal disjunctive normal form for a boolean function," to appear in IEEE Transactions on Computers.
- Blum, L. and Blum, M. (1975). "Toward a mathematical theory of inductive inference," Information and Control 28, 125-155.
- Cook S.A. (1971). "The complexity of theorem-proving procedures," Proceedings of the Third Annual ACM Symposium on Theory of Computing, 151-158.
- Feldman, J.A. (1972). "Some decidability results on grammatical inference and complexity," Information and Control 20, 244-263.
- Gold, E.M. (1967). "Language identification in the limit," Information and Control 10, 447-474.
- Gold, E.M. (1974). "Complexity of automaton identification from given data," Département d'Informatique, Université de Montréal, to be published.
- Horning, J.J. (1969). "A study of grammatical inference," Ph.D. thesis, Computer Science Department, Stanford University, CS 139.
- Karp, R.M. (1972). "Reducibilities among combinatorial problems," in Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, eds., Plenum Press, New York, 85-104.
- Meyer, A.R. and Fischer, M.J. (1971). "Economy of description by automata, grammars, and formal systems," Proceedings of the Twelfth Annual IEEE Symposium on Switching and Automata Theory, 188-191.
- Trakhtenbrot, B.A. and Barzdin, Ya. M. (1973). Finite Automata, North Holland Publishing Company, Amsterdam.