Diversity-Based Inference of Finite Automata

(Extended Abstract)

Ronald L. Rivest Robert E. Schapire

MIT Laboratory for Computer Science
Cambridge, MA 02139

Abstract

We present a new procedure for inferring the structure of a finite-state automaton (FSA) from its input/output behavior, using access to the automaton to perform experiments.

Our procedure uses a new representation for FSA's, based on the notion of equivalence between *tests*. We call the number of such equivalence classes the *diversity* of the automaton; the diversity may be as small as the logarithm of the number of states of the automaton. The size of our representation of the FSA, and the running time of our procedure (in some case provably, in others conjecturally) is polynomial in the diversity and $\ln(\frac{1}{\epsilon})$, where ϵ is a given upper bound on the probability that our procedure returns an incorrect result. (Since our procedure uses randomization to perform experiments, there is a certain controllable chance that it will return an erroneous result.)

We also present some evidence for the practical efficiency of our approach. For example, our procedure is able to infer the structure of an automaton based on Rubik's Cube (which has approximately 10^{19} states) in about 2 minutes on a DEC MicroVax. This automaton is many orders of magnitude larger than possible with previous techniques, which would require time proportional at least to the number of global states. (Note that in this example, only a small fraction (10^{-14}) of the global states were even visited.)

1 Introduction

We address the problem of inferring a description of a deterministic finite-state automaton from its input/output behavior. (Kohavi [10] gives an excellent introduction to the theory of finite-state automata, as do Hartmanis and Stearns [9].)

This problem has a long history; the following works were particularly inspirational. Angluin and Smith [5] give an excellent overview of the entire field of inductive inference. Gold [8] gives an algorithm for inferring a machine from a sample of its input/ouput behavior. Angluin [3] elaborates this algorithm to show how to infer an automaton using active experimentation with the automaton, and a "teacher" who can provide counterexamples to incorrect conjectures. Angluin [4] and Gold [7] prove that finding an automaton of n states or less agreeing with a given sample of input/output pairs is NP-complete. Here the concern is with the number of states in the inferred automaton, and that the data is *given* – the inference algorithm does not have access to the automaton. Angluin [2] shows how to infer a special-class of finite-state automata, called "k-reversible" automata, from a sample of input/output behavior.

Our motivation is the "artificial intelligence" problem of identifying an environment by experimentation. We imagine a robot wandering around in an unknown environment, whose characteristics must be discovered. Such an environment need not be deterministic, or even finite-state, so the approach suggested here is only a beginning on the more general problem. For a fascinating discussion of the problem of inferring an environment from experience, the reader is encouraged to read Drescher [6], whose approach is based on the principles of Piaget.

In line with our motivation, our inference procedure experiments with the automaton to gather information.

A unique and valuable feature of our procedure is that it does *not* need to have the automaton "reset" to some start state or "backed-up" to a previous state; the data-gathering is one continuous experiment (as in real life).

Our procedure is practical; its time and memory requirements are quite reasonable. For example, our procedure does not need to store the entire observed input/output history.

We give some preliminary experimental results at the end of this paper.

*This paper prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and a grant from the Siemens Corporation. Authors' net addresses: rivest@theory.lcs.mit.edu, rs@theory.lcs.mit.edu.

0272-5428/87/0000/0078$01.00 © 1987 IEEE

2 Definitions

2.1 Automata and Environments

Our definition of a finite-state automaton is a generalization of the usual Moore automaton. [10]. (Our approach generalizes to handle Mealy automata; however, we find Moore automata more natural.)

Definition 1 *A* finite-state automaton $\mathcal{E}$ *is a* 5*-tuple* $(Q, B, P, \delta, \gamma)$ *where*

- Q is a finite nonempty set of *states.*
- B is a finite nonempty set of *input symbols*, also called *basic actions.*
- P is a finite nonempty set of *predicate symbols*, also called *sensations.*
- δ is a function from $Q \times B$ into Q; δ is called the *next-state function.*
- γ is a function from $Q \times P$ into {**true**, **false**}.

When P only contains a single predicate (e.g. **accept**), we have the standard definition of a Moore automaton. We allow multiple predicates to correspond to the notion of a robot having multiple sensations in a given state of the environment.

We assume henceforth that we are dealing with a particular finite-state automaton $\mathcal{E} = (Q, B, P, \delta, \gamma)$, which we call the *environment* of the learning procedure.

We say that $\mathcal{E}$ is a *permutation environment* if for each $b \in B$, the function $\delta(\cdot, b)$ is a permutation of Q.

We let $A = B^*$ denote the set of all sequences of zero or more basic actions in the environment $\mathcal{E}$; A is the set of *actions* possible in the environment $\mathcal{E}$, including the *null action* λ.

If q is a state in Q, and $a = b_1 b_2 \ldots b_n$ is an action in A, we let $qa = qb_1 b_2 \ldots b_n$ denote the state resulting from applying action a to state q:

$$qa = \delta(\ldots \delta(\delta(q, b_1), b_2) \ldots, b_n). \tag{1}$$

(The basic actions are performed in the order $b_1, b_2, \ldots, b_n$.) Similarly, if q is a state and p is a predicate, we let $qp = \gamma(q, p)$ denote the result of applying predicate p to state q.

We say that $\mathcal{E}$ is *strongly connected* if

$$(\forall q \in Q)(\forall r \in Q)(\exists a \in A) qa = r. \tag{2}$$

We do not assume that $\mathcal{E}$ is strongly-connected unless explicitly stated.

2.2 Tests

A *test* is an element of AP, that is, an action followed by a predicate. We let T denote the set of tests AP. We say that a test $t = ap$ *succeeds at state* q if $qt = q(ap) = qap = (qa)p$ is **true**. Otherwise we say that t *fails at* q. The *length* $|a|$ of a test a is the number of basic actions and predicates it contains.

We say that $\mathcal{E}$ is *reduced* if

$$(\forall q \in Q)(\forall r \in Q)(q \neq r \Rightarrow (\exists t \in T) qt \neq rt) \tag{3}$$

We assume henceforth that $\mathcal{E}$ is reduced.

It will be convenient to define a total order on the set of tests. We say that test t_1 *is smaller than* test t_2, written $t_1 \ll t_2$, if $|t_1| < |t_2|$, or, in the case that $|t_1| = |t_2|$, if t_1 (considered as a string in $AP = B^*P$) lexicographically precedes t_2 using an assumed fixed ordering of the symbols in B and P.

2.3 Equivalence of Tests and Diversity

A central notion in our development is that of *test equivalence.*

We say that tests t_1 and t_2 are *equivalent*, written $t_1 \equiv t_2$, if

$$(\forall q \in Q)(qt_1 = qt_2); \tag{4}$$

that is, from any state the two tests yield the same result.

The equivalence relation on tests partitions the set T of tests into equivalence classes. The equivalence class containing a test t will be denoted $[t]$.

The *diversity* of the environment $\mathcal{E}$, denoted $D(\mathcal{E})$, is the number of equivalence classes of tests of $\mathcal{E}$:

$$D(\mathcal{E}) = |\{[t] \mid t \in T\}|. \tag{5}$$

The following theorem demonstrates that the diversity of a finite-state automaton is always finite, but is only loosely related to the *size* (i.e. number of states) of the automaton.

Theorem 1 *For any reduced finite-state automaton* $\mathcal{E} = (Q, B, P, \delta, \gamma)$,

$$\lg(|Q|) \leq D(\mathcal{E}) \leq 2^{|Q|}.$$

Proof: The first inequality $|Q| \leq 2^{D(\mathcal{E})}$ follows since a state is uniquely identified by the set of (equivalence classes of) tests which are true at that state, since $\mathcal{E}$ is reduced. The second inequality follows since the equivalence class that a test belongs to is uniquely defined by the set of states at which that test succeeds. ■

Theorem 2 *The lower and upper bounds on $D(\mathcal{E})$ given in theorem 1 are best possible.*

Proof: For the lower bound, consider an environment where the states are n-bit words, and there is a predicate p_i which tests whether the i-th bit is one, for $1 \leq i \leq n$. The set B consists of a single action, which is the identity operation (no state change). Then $D(\mathcal{E}) = n$ but $|Q| = 2^n$.

For the upper bound, consider an environment consisting of n states numbered 0 through $n-1$, which has a single predicate p which succeeds only at state 0. For each subset X of the states, there is an action b_X which moves state x to state 0 if $x \in X$, or to state 1 otherwise. Thus, the test $b_X p$ is **true** iff we are in one of the states in X. Hence, $D(\mathcal{E}) = 2^{|Q|}$.

The upper bound can be nearly achieved with an environment which has only a constant number of actions. We omit its description here. ■

We propose that the notion of diversity is more suitable than that of size for many natural applications. To support this viewpoint, we will demonstrate that *there exists a natural encoding of a finite-state automaton, whose size is polynomial in the diversity of the automaton.* Furthermore, it is straightforward to use this representation to simulate the behavior of the automaton.

2.4 Canonical and Square Tests

We say that test t is *canonical* if it is the smallest test in its equivalence class $[t]$. The canonical test in $[t]$ is denoted $\langle t \rangle$. We let C denote the set of canonical tests. (By definition, $|C| = D(\mathcal{E})$.)

Theorem 3 *Let a be any action in A and let t be any test in T. If t is a non-canonical test, then so is at. Equivalently, if at is canonical, then so is t.*

Proof: Note that if $t \equiv \langle t \rangle$, then $at \equiv a\langle t \rangle$. ■

Let b be any basic action in B, and let t be any test in T. We say that the test bt is *square* if bt is non-canonical but t is canonical. We let S denote the set of square tests. Note that if a test t is canonical, then for any $b \in B$, the test bt is either canonical or square, and that $|S| \leq |B|D(\mathcal{E})$.

We can consider arranging the elements of T into a forest consisting of $|P|$ infinite trees, where

1. The roots of the trees are the predicates in P.
2. The node corresponding to a test t has one child corresponding to bt for each b in B.

In a drawing of such a forest, there will be a line one can draw such that all tests above the line are canonical, and those tests which are below the line but whose parents are above the line are square.

2.5 The Simulation Theorem

Theorem 4 *To simulate $\mathcal{E}$ it suffices to know:*

1. *The set C of canonical tests.*
2. *The value qt of each canonical test t at the current state q.*
3. *For each square test s in S, the corresponding canonical test $\langle s \rangle$.*

Proof: Suppose the automaton moves from state q to state qb, for some $b \in B$. We need to compute $(qb)t = q(bt)$ for each canonical test t. However, the test bt is either canonical or square, so that in either case the canonical test $u = \langle bt \rangle$ is known. By assumption, we know qu; this is the desired value of $(qb)t$. ■

2.6 The Update Graph

As another means of representing the test classes, we may build a graph in which each vertex is an equivalence class, and an edge labeled $b \in B$ is directed from test class $[t]$ to $[t']$ iff $t \equiv bt'$. We call this the *update graph* of the environment. The update graph may alternatively be derived from the forest of tests described in Section 2.4 by directing each tree edge up, and merging each square test with its equivalent canonical test.

We associate with each vertex $[t]$ the value of t at the current state q. When action b is executed, the value of each vertex is passed to each of its b-children yielding the new value of each test in state qb.

Neal Young and Dana Angluin have pointed out the following relationship between the update graph of an environment with a single predicate, and the original automaton: Let q_0 be the starting state, and p the sole predicate. Let L be the language accepted by our automaton, i.e., the set of action sequences a such that $q_0 ap$ is **true**. We define U to be the environment's update graph with all of the edges reversed in direction, and each vertex $[t]$ assigned the value $q_0 t$. That is, U is viewed as a DFA in which the accepting states are just those states $[t]$ for which t succeeds at q_0. Finally, let vertex $[p]$ be this DFA's starting state. Young and Angluin observed that the language accepted by U is exactly the reverse of L. A proof follows from the definition of U and the fact $(q_0 a)p = q_0(ap)$.

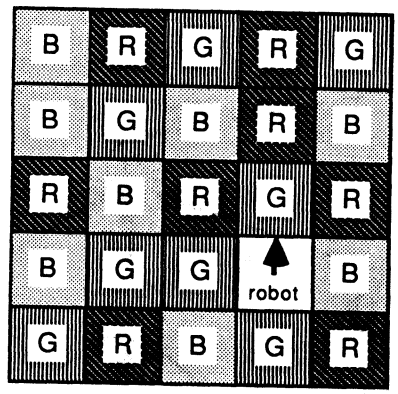

Figure 1: The 5 x 5 Grid World

3 Two Example Environments

The motivation for the introduction of the notion of diversity was the realization that many interesting "robot environments" can be modelled as finite automata which, although they have a large number of states, have low diversity. In this section, we make this point explicit by describing two particular small "robot environments".

3.1 $n \times n$ Grid World

Consider a robot on an $n \times n$ square grid (with "wraparound", so that it is topologically a torus). See Figure 1. The robot is on one of the squares and is facing in one of the four possible directions. Each square is either red, green, or blue. The robot can sense the color of the square it is facing. (This corresponds to the predicates of our previous development.)

The following actions are available to the robot: It can paint the square it faces red, green, or blue. The robot can turn left or right by 90 degrees, or step forward one square in the direction it is facing. Stepping ahead has the curious side effect of causing the square it previously occupied to be painted the color of the square it has just moved to, so moving around causes the coloring to get scrambled up.

This environment is a finite-state automaton which, even after reducing by factoring out some obvious symmetries, has an exponentially large (3^{n^2-1}) number of states.

However, the *diversity* of this environment is only $O(n^2)$. The state of this environment is completely characterized by knowing the color of each square (using a robot-relative coordinate system). It is not hard to devise a set of $O(n^2)$ tests whose results give all the desired information. (For example, the square behind the robot is red if and only if the test "`turn-left turn-left see-red`" is true.)

Given this information, it is easy to see how to predict the state of the environment after a given sequence of actions. In fact, it becomes clear that this is the "natural" representation of this environment, and that the intuitive representation and simulation procedure one would use for this environment are captured almost exactly by the diversity-based representation and simulation procedure given in the previous section.

We note that because of the "paint" operations, this environment is not a permutation environment.

3.2 n-bit Register World

In this environment, the robot is able to read the leftmost bit of an n-bit register. Its actions allow it to rotate the register left or right (with wraparound) or to flip the bit it sees.

Clearly, this automaton consists of 2^n global states, but its diversity is only $O(n)$ since there is one test for each bit, and one for the complement of each bit. We note that the register world is a permutation automaton.

Figure 2 shows the canonical and square tests of the three-bit register world arranged in the sort of forest described in Section 2.4. The update graph of this environment is depicted in Figure 3. The name "1" in the figures refers to the predicate which returns **true** if the leftmost bit is a 1, and "L", "R" and "F" refer to the actions which rotate left and right, and which flip the leftmost bit. In the current state, the register contains the values 101. The borders of the tests which are **true** in the current state have been darkened.

4 Our Inference Procedure

The inference procedure tries to identify the sets C and S of canonical and square tests, and for each square test $s \in S$ it tries to identify the corresponding canonical test $\langle s \rangle$.

4.1 An Inference Procedure Using an Oracle for Inequivalence

We begin by supposing that we have an oracle available that can tell us whether two tests s and t are *inequivalent*. This is the framework for our final algorithm,

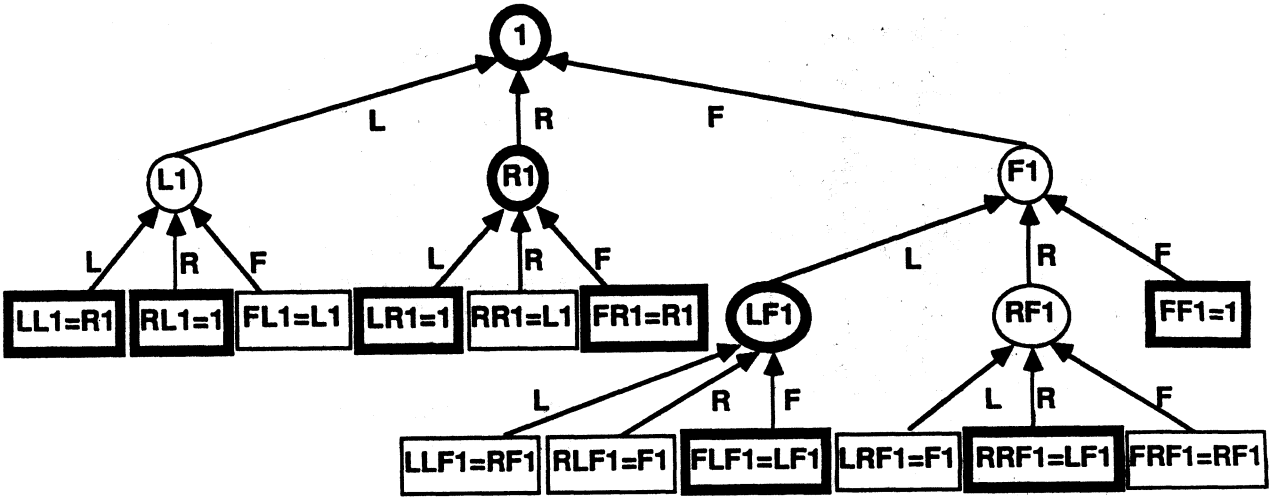

Figure 2: Forest of Tests for 3-bit Register World

where we can experiment with the automaton to disprove conjectured equivalences. For each square test s, the program computes $\chi(s) = \langle s \rangle$.

Then our algorithm would be as given in Figure 4. Here the program variables S and C represent the current set of candidates for square and canonical tests. The program variable R is the set of tests which require being tested for being canonical; we assume that all the predicates in P are canonical.

We see that the running time of this algorithm requires a number of inequivalence tests not greater than

$$|C| \cdot (|C| + |S|) \leq 2|B|D(\mathcal{E})^2 \qquad (6)$$

4.2 Determining if Two Tests are Equivalent

We now turn our attention to the problem of determining whether or not two tests are inequivalent. The inference procedure can *prove* that tests s and c are inequivalent if we can find a state q such that $qs \neq qc$; a single counterexample to the conjecture $s \equiv c$ suffices.

We wish to experiment with the available automaton $\mathcal{E}$ in order to prove $s \not\equiv c$. There are two problems we face:

- *(Accessibility of Counterexamples)* It may be difficult or impossible to get the automaton into a state q where $qs \neq qc$, even if such states exist. (The automaton may not even be connected.)
- *(Irreversibility of Actions)* Even if we can get the automaton into such a state q, once we run test s we are in general unable to "back up" so as to be able to run test c.

Let us define two tests to be *compatible* if the action sequence of one is a prefix of the action sequence of the other. We note that irreversibility of actions is not a problem when testing the equivalence of two compatible tests since they can be executed simultaneously. In particular, a predicate is compatible with all other tests.

We present solutions to these difficulties for the special class of permutation environments, and then discuss progress toward a solution in the general case.

4.3 Determining Test Equivalence in Permutation Environments

Assume then that $\mathcal{E}$ is a permutation environment. In this case the irreversibility of actions is not a problem. It is easy to show that each action permutes not only the global states, but the set of test equivalence classes as well. Thus, the set of permutations on T generated by the basic actions forms a group, and so each action sequence has an inverse. Therefore, for any test $t = ap$, we have $p = a^{-1}t$, and so there is always at least one predicate/action sequence pair (p_t, a_t) for which $p_t = a_t t$. We call such a pair a *forced compatibility pair (FC-pair)* for t.

The FC-pair allows us to effectively force a test to be compatible with any other test. Suppose that one FC-pair (p_c, a_c) is known for each canonical $c \in C$, and that t is some other test. Then $c \equiv t$ if and only if

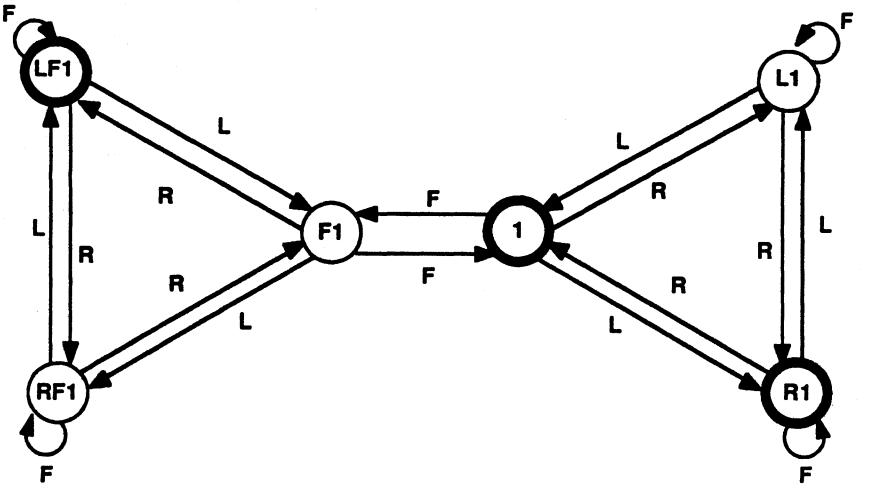

Figure 3: Update Graph of 3-bit Register World

$a_c t \equiv a_c c \equiv p_c$. Thus we can check c's equivalence to t by comparing the tests $a_c t$ and p_c. The values of these two tests can be determined simultaneously simply by executing them since they are compatible.

We can guarantee that we always have an FC-pair for each known canonical by modifying slightly the control mechanism of the algorithm. Here is the strategy: Each time a canonical bc, $b \in B$, $c \in \mathtt{C}$, is added to the set of canonicals, rather than extracting the smallest test in $\mathtt{R}$, immediately work to determine if this canonical's b-child, bbc, is square or canonical. To see that this works, assume inductively that an FC-pair is known for each canonical in $\mathtt{C}$, and that bc is added to $\mathtt{C}$. Because an FC-pair is known for all the members of the original set of canonicals, we can compare bbc with each for equivalence. Moreover, we need not compare bbc with bc since $c \not\equiv bc$ (because both are canonical) implies $bc \not\equiv bbc$. Thus, we can determine whether bbc is square or canonical. If it is canonical, we continue in this fashion, generating a series of canonicals of the form $b^i c$ until eventually one of these, $b^n c$, turns out to be equivalent to one of the starting canonicals c'. By assumption, an FC-pair, $(p_{c'}, a_{c'})$, is known for c', and thus an FC-pair is now known for all of the new canonicals since $a_{c'} b^{n-i}(b^i c) \equiv a_{c'} c' \equiv p_{c'}$.

(This strategy will cause some of the members of $\mathtt{C}$ not to be the smallest in their equivalence classes during the execution of the algorithm. This and other details can be overcome without considerable difficulty.)

Turning now to the problem of the accessibility of counterexamples, we find that for permutation environments this is again not a problem. We show that a random walk of sufficient length gives us an adequate probability of ending in a state where two inequivalent tests have different values.

Theorem 5 *Let t and t' be two inequivalent tests of a permutation environment $\mathcal{E}$ of diversity D. We take a random walk of length uniformly and randomly chosen between 1 and $2|B|D^2(D-1)^2$. Then the probability that the values of t and t' differ at the state where we complete this walk is at least $\frac{1}{4|B|D^2(D-1)^2}$.*

Proof: Let G be the group generated by the set of permutations of the test classes effected by the basic actions. Let H be the set of $h \in G$ such that $ht \equiv t$ and $ht' \equiv t'$. Then H is a subgroup of G.

For any $g \in G$, the left coset gH is exactly the set of permutations g' for which $g't \equiv gt$ and $g't' \equiv gt'$. Thus $[G : H]$ is limited by the number of distinct pairs $([gt], [gt'])$, and so $[G : H] \leq D(D-1)$.

Let $\mathcal{H}$ be the graph of left cosets of H, in which an edge labeled $b \in B$ is directed from vertex gH to $g'H$ if $bgH = g'H$. Then indegree equals outdegree at each vertex, and $\mathcal{H}$ is an Eulerian digraph.

Since $t \not\equiv t'$, there must be a permutation g which takes the robot from its current state to a state in which the values of t and t' differ. By the preceding argument, any permutation in gH will have this effect.

The robot takes a random walk $r = b_1 b_2 \ldots b_n$, each b_i a basic action, and succeeds in distinguishing t and t' if $r \in gH$, or equivalently, if $rH = gH$. We note that $rH = b_1 b_2 \ldots b_n H = b_1(b_2 \ldots (b_{n-1}(b_n H)) \ldots)$,

Input:
  P - set of predicates
  B - set of basic actions
  Oracle for testing if $s \not\equiv c$ for any tests s and c
Output:
  C - set of canonical tests
  S - set of square tests
  $\chi : \mathtt{S} \rightarrow \mathtt{C}$ such that $\chi(s) = \langle s \rangle$
Procedure:
  $\mathtt{C} = P$;
  $\mathtt{R} = \mathtt{S} = BP$;
  while R is non-empty **do**
    Extract the smallest test s from R;
    Let $\mathtt{V} = \mathtt{C}$;
    for each test c in V,
      if $s \not\equiv c$, **then** remove c from V;
    if $|\mathtt{V}| = 0$ **then** (s *is canonical*)
      Remove s from S, and add it to C;
      for each $b \in B$, add bs to S and R;
    else ($|\mathtt{V}| = 1$.)
      Set $\chi(s) = c$, where $\mathtt{V} = \{c\}$;
    endif
  end

Figure 4: An Inference Algorithm Using an Oracle for Inequivalence of Tests.

and therefore, because we are only concerned with t and t', we may view the robot's random walk as a (backwards) random walk on $\mathcal{H}$: The robot begins at H and follows the edges $b_n b_{n-1} \ldots b_1$; if it arrives at gH, then it succeeds in proving that $t \not\equiv t'$.

The expected number of steps to reach vertex gH starting from H is bounded by the product of the number of edges and the maximum distance between any two vertices (see [1]). In our case this product is bounded by $E = |B|D^2(D-1)^2$. So the probability of having visited gH in $2E$ steps is at least one-half. If we stop after a random number of steps we therefore have a probability of at least $\frac{1}{4E}$ of being at gH when we stop. ■

Using this result, we can show the following theorem:

Theorem 6 *Let $\mathcal{E}$ be a permutation environment with diversity D. Given $\epsilon > 0$, our algorithm will infer the structure of $\mathcal{E}$ in time $O(|B|^3 D^{10} \cdot \log(\frac{|B|D^2}{\epsilon}))$ with probability of error less than ϵ.*

Proof: The preceding theorem states that the probability of distinguishing two inequivalent tests, having taken an appropriate random walk, is at least $\frac{1}{4E}$. Thus, the probability of failing to do so after n trials is no greater than $(1 - \frac{1}{4E})^n$. This error is bounded by a parameter δ when

$$n \geq \frac{\log \delta}{\log(1 - \frac{1}{4E})}.$$

As many as $I = O(|B|D^2)$ inequivalence tests may be made in the course of inferring the automaton. The probability, then, of successfully distinguishing all of the inequivalent pairs of tests is at least $(1-\delta)^I$. Our goal is to make this probability more than $1 - \epsilon$. We have been given ϵ and choose $\delta \leq \frac{\epsilon}{I}$. Then

$$1 - \epsilon \leq 1 - I\delta \leq (1-\delta)^I$$

as desired.

Finally, if we choose $n \geq 4E \log \frac{I}{\epsilon}$, then our probability of error on an individual experiment is sufficiently small since

$$4E \log \frac{I}{\epsilon} \geq \frac{\log \frac{I}{\epsilon}}{\log \frac{4E}{4E-1}} = \frac{\log \frac{\epsilon}{I}}{\log(1 - \frac{1}{4E})} \geq \frac{\log \delta}{\log(1 - \frac{1}{4E})}.$$

Here, we have used the fact that $\log x \leq x - 1$ for all x. (In particular, if $x < 1$ then $\log x \leq x - 1 \Rightarrow \frac{1}{\log \frac{1}{x}} \leq \frac{1}{1-x}$. Above, we have applied this formula with $x = 1 - \frac{1}{4E}$.)

Hence, our procedure requires I inequivalence tests. Each of these requires up to $4E \log \frac{I}{\epsilon}$ experiments, each of which can involve a random walk of length $2E$. (The time to run the actual experiment, or to determine which experiment is to be performed next is negligible.) We thus arrive at the running time stated in the theorem. ■

We strongly believe that these bounds can be tightened significantly. Empirically, we have found that much shorter random walks and far fewer experiments are sufficient. In addition, we believe a recent unpublished result due to Gilbert Strang would allow us to improve our running time bound to $O(|B|^2 D^8 \cdot \log(\frac{|B|D^2}{\epsilon}))$.

As a final remark, we note that Angluin, addressing the problem of inferring regular languages in [2], also identified a subclass of permutation automata, which she called "zero-reversible," as worthy of special attention.

4.4 Determining Test Equivalence in General

We discuss now the general case in which $\mathcal{E}$ is not necessarily a permutation environment. We don't at the

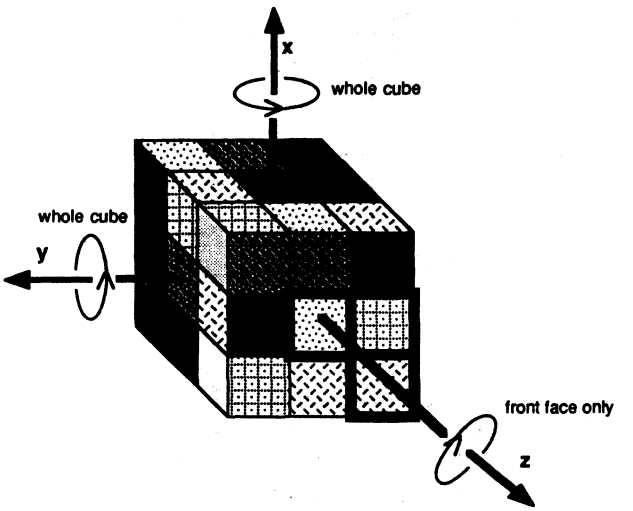

Figure 5: The Rubik's Cube World

moment know how to handle in a rigorous manner the first difficulty of finding a state in which two inequivalent tests can be distinguished, even if we assume that $\mathcal{E}$ is strongly connected. Nonetheless, in practice this may often not be a concern; if two tests s and c are inequivalent then there are usually many easily reached states q such that $qs \neq qc$.

We now propose a cute technique for handling the irreversibility of actions in general environments.

We need to figure out how to get $\mathcal{E}$ into a state q where we *know* the value of the test qc, even though we haven't run test c yet, so that we can run test s instead.

Let $c = ap$; here a is the action part of test c and p is the predicate.

Suppose we run action a repeatedly. Eventually the predicate p will exhibit periodic behavior. Once we know that this periodic behavior has been established, and once we know the period m of this behavior, then we can figure out the value of qc for the current state q without having to run the test c.

We have to address the problem that for general finite-state automata, it is well known that the eventual period can be as large as the size of the automaton. This would be a serious problem for our proposed approach, since the size can be an exponential function of the diversity. However, the following theorem shows that the period is no larger than the diversity.

Theorem 7 *If we run action a repeatedly, then the behavior of predicate p will exhibit transient behavior for no more than $D(\mathcal{E})$ steps, and then will settle down into periodic behavior with period at most $D(\mathcal{E})$.*

Proof: This follows easily from our simulation theorem (theorem 4), since any action has the effect of giving a new value to each canonical test which is an old value of one of the canonical tests. ■

To complete the description of our inference procedure, we suppose that an upper bound D_{max} is available on the diversity $D(\mathcal{E})$ of the automaton being inferred. (If no such bound is available, the algorithm can be executed repeatedly with $D_{max} = 1, 2, 4, 8, \ldots$.)

To run the algorithm of Figure 4, we need a way to test s and $c = ap$ for inequivalence. The following procedure is suggested by the previous theorem:

- Run action a for D_{max} steps. (This is to eliminate transient behavior of p.)
- Run action a for $2D_{max}$ steps, keeping track of qp for each state q reached.
- Use the information gathered in the previous step to determine the period of predicate p under action a. Use this information to determine whether qc is **true** or **false** in the current state q (without running test c).
- Run test s to determine qs.
- If $qc \neq qs$, return "`INEQUIVALENT`". Otherwise, return "`POSSIBLY EQUIVALENT`".

As it stands, this is a one-sided test: a report of `INEQUIVALENT` is certainly correct, but a report of `POSSIBLY EQUIVALENT` is not sufficient to conclude that $s \equiv c$ with any high degree of certainty.

The test should be re-run a number of times before concluding that $s \equiv c$. To make the trials as independent as possible, we may:

- Take a "random walk in $\mathcal{E}$" between each trial, by executing some randomly chosen sequence of actions.
- Repeatedly execute an action ab instead of just a in each trial, where b is an arbitrarily chosen action in A.

These heuristics may not help to find a counterexample in all cases; but are reasonably effective in practice. (We hope to prove the effectiveness of these techniques as we did in the permutation environment case.)

For efficiency, we are in many instances able to force compatibilities as in the permutation environment case, and can often compare many tests against

Environment	Diversity	Global States	$\|B\|$	$\|P\|$	Version	Time	Moves	Senses	Experiments
Little Prince	4	4	3	2	P	0.1	303	102	51
					M	0.2	900	622	50
Car Radio	9	27	6	3	M	3.7	27,695	9,557	1,146
Grid World	27	$\approx 10^{11}$	6	3	M	90.4	583,195	123,371	9,403
Rubik's Cube	54	$\approx 10^{19}$	3	18	P	126.3	58,311	4,592	2,296
					M	401.3	188,405	79,008	2,874
32-bit Register	64	$\approx 10^{9}$	3	1	P	29.8	270,771	10,914	5,457
					M	18.3	52,436	29,884	300

Table 1: Experimental Results

many other tests in single experiments. These heuristics lead to many-fold improvements of our running times.

5 Experimental Results

Consider the following permutation environment based on "Rubik's Cube" (Figure 5). The robot is allowed to see only three of the fifty-four tiles: a corner tile, an edge tile and a center tile, all on the front face. Each of these three senses can indicate any one of six colors. The robot may rotate the front face, and may turn the whole cube about the x and y axes. (By reorienting the cube he can thus turn and view any of the six faces.)

Table 1 summarizes how our procedures handled this environment, the 5×5 grid world environment, the 32-bit register environment, and two other comparatively simple environments not described here. (Our last paper [11] contains a more detailed description of our experimental results, although none of the proofs of theoretical results contained in this paper.)

The most complicated environment (Rubik's Cube) took less than two minutes of CPU time to master — we consider this very encouraging.

Rubik's Cube, the Little Prince and the 32-bit Register worlds were explored with an implementation (version "P") which exploits the special properties of permutation environments, but which only compares one pair of tests at a time. All worlds were explored as well by version "M", which tries to compare many tests against many other tests in a single experiment. The run times given are in seconds. The last three columns give the number of basic actions taken by the robot, the number of sense values asked for, and the number of experiments performed. (An experiment is defined loosely as a sequence of actions and senses from which the robot deduces a conclusion about equivalence between tests. Information about several tests may be obtained in a single experiment, and the same sequence of actions and senses may be repeated several times, each repetition counting as one experiment. Also, we have generalized the notion of a test here to allow the function γ to map $Q \times P$ into an arbitrary set of sensations, not necessarily the set {true, false}.) These implementations were done in C on a DEC MicroVax II workstation.

6 Conclusions

We have presented a new representation for finite-state systems (environments), and proposed a new procedure for inferring a finite state environment from its input/output behavior.

In the case of permutation environments, our procedure can infer the structure of the environment in expected time polynomial in the diversity of the environment, and $\ln(\frac{1}{\epsilon})$, where ϵ is an arbitrary positive upper bound given on the probability that our procedure will return an incorrect result.

For general environments, our procedure appears to work well in practice, although we don't have a proof to this effect.

When the environment has lots of "structure", the diversity will typically be many orders of magnitude smaller than the number of global states of the environment; in these cases our procedure can offer many orders of magnitude improvement in running time over previous methods.

Acknowledgements

We gratefully acknowledge helpful comments and suggestions from Dana Angluin, Ravi Boppana, Gilbert Strang, and Neil Young.

References

[1] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovász, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundation of Computer Science*, pages 218–223, San Juan, Puerto Rico, October 1979.

[2] Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, July 1982.

[3] Dana Angluin. *Learning Regular Sets from queries and counter-examples.* Technical Report YALEU/DCS/TR-464, Yale University Department of Computer Science, March 1986.

[4] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[5] Dana Angluin and Carl H. Smith. Inductive inference: theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[6] Gary L. Drescher. *Genetic AI – Translating Piaget into Lisp.* Technical Report 890, MIT Artificial Intelligence Laboratory, February 1986.

[7] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[8] E. Mark Gold. System identification via state characterization. *Automatica*, 8:621–636, 1972.

[9] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines.* Prentice-Hall, 1966.

[10] Zvi Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, second edition, 1978.

[11] Ronald L. Rivest and Robert E. Schapire. A new approach to unsupervised learning in deterministic environments. In Pat Langley, editor, *Proceeding of the Fourth International Workshop on Machine Learning*, pages 364–375, Irvine, California, June 1987.