A Reduced Test Suite for Protocol Conformance Testing

Philip J. BERNHARD
Harris Space Systems Corporation

Let M be a finite-state machine, and let S be an implementation of M. The *protocol-testing problem* is the problem of determining if S is a correct implementation of M. One known method for solving this problem, called the *W-method*, has the disadvantage that it generates a relatively large test set. In this paper, we describe three new versions of this method. We prove that these versions all have the same fault detection capability as the W-method. In addition, we show that in most cases all three generate a smaller number of tests than the W-method. Specifically, suppose M_1 and M_2 are finite-state machines having n and m states, respectively, where M_1 is a *specification* (M), M_2 is an *implementation* (S), and $m \geq n$. In addition, suppose they have input alphabet Σ, where $|\Sigma| = k$; let α be the total number of strings in a *characterization set* for M_1, and let β be the total number of strings in a *transition cover set* for M_1. The W-method will generate a test set consisting of $\alpha\beta(k^{m-n+1} - 1)/(k - 1)$ strings. In contrast, our first algorithm will generate a test set containing at most $\beta(\alpha + k^{m-n})$ strings. For our second algorithm, the number of strings will be $\beta k^{\max(n-1, m-n)}$, and for the third, $\beta(k^{n-1} + k^{m-n})$. When $m \gg n$, all three of our algorithms will produce fewer strings than the W-method. Finally, two of our algorithms make use of a heuristic for minimizing the number of strings in a characterization set. We show that the *performance ratio* for this heuristic has an upper bound of $O(\log n)$.

Categories and Subject Descriptors: C.2.2. [**Computer-Communication Networks**]: Network Protocols—*protocol verification*; 0.2.4 [**Software Engineering**]: Testing and Debugging—*test data generators*

General Terms: Algorithms, Reliability, Theory, Verification

Additional Key Words and Phrases: Heuristics

1. INTRODUCTION

The problem of testing software has been approached in a variety of ways. Particular testing methods are typically categorized as either *specification based* or *program based* [Howden 1987]. Alternatively, these are often referred to as *black-box* and *white-box*, respectively. Most are general-purpose methods, in the sense that they are designed to work for just about any type of software. On the other hand, some special-purpose testing techniques have also been developed. These techniques try to exploit constraints on software structure in an attempt to improve the effectiveness of the testing process.

Author's address: Harris Space Systems Corporation, 295 Barnes Blvd., Rockledge, FL 32955.

1.1 Protocol Testing

In this paper we focus on the problem of testing a specific class of software system. Specifically, we focus on the problem of testing the control portion of *protocols*. Generally speaking, we can say that a protocol specifies the rules used by a collection of computer systems that are communicating on some network. It has been shown by Chow [1978] that by restricting consideration to this special case, we can generate test data that is guaranteed to detect specific types of errors. Though Chow's method will not detect all errors, it is guaranteed to detect certain types of errors in all cases, regardless of how many occur or in what combination.

The specification of the control portion of a protocol is often given in the form of a finite-state machine that produces output. Such machines are often called *Mealy* machines, and they differ from deterministic finite-state automata in that each transition is associated with an output symbol. The idea is that when a transition occurs from one state to another, a symbol associated with the transition is generated as output. In a protocol specified by such a machine, the output symbol typically corresponds to some command or message. An example of such a machine is shown in Figure 1. For this particular machine, an input of *01100* will produce *abbba* as output.

The protocol-testing problem can be stated as follows. Suppose that the specification of a protocol is given in the form of a Mealy machine M. In addition, suppose that S is an implementation of M.[1] The question we wish to ask is whether or not S is a correct implementation of M. Protocol-testing methods answer this question by prescribing a way to generate a set of tests from the specification M, which is then used to test the implementation S. Ideally, we would like to construct a test set D such that the output produced by M on D is the same as the output produced by S on D, if and only if S correctly implements M.

1.2 Previous Work

Problems such as the protocol-testing problem have been considered for some time, and much of the current work has its roots in problems concerned with state identification and fault detection for sequential circuits. Gill [1962] and Hennie [1968] summarize much of the work in these areas, respectively. Many of the ideas in these areas can be traced back to the seminal work of Moore [1956]. The protocol-testing methods most often referred to are the T-method [Naito and Tsunoyama 1981], the D-method [Gonenc 1970], the U-method [Sabnani and Dhabura 1988], and the W-method [Chow 1978]. All of these are similar in that they provide a method for generating a set of one or more strings $\{s_1, s_2, \ldots, s_m\}$, which is often referred to as a *test suite*. The implementation is then tested using these strings to verify that it conforms to the specification M. Among these methods, the T-method is usually consid-

[1] Note that the "implementation" could be in hardware, software, or some combination of the two. For the purposes of this paper it is convenient to assume that the implementation is in software; however, the results presented hold regardless of the exact implementation medium

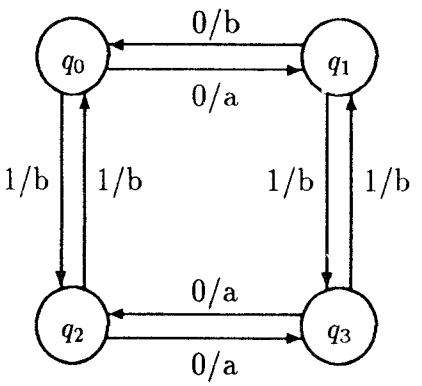

Fig. 1. A minimal Mealy machine.

ered to be unique, while the other three are considered to be somewhat similar. Specifically, the T-method is primarily concerned with checking the transitions between states in a protocol, whereas the others will check both states and transitions.

The different protocol-testing methods can be compared on a number of different dimensions. The two most obvious are the size and fault detection capability of the resulting test set. With regard to the size of the test set produced, Sidhu and Leung [1989] have observed empirically that the T-method will usually produce the smallest, while the W-method will usually produce the largest. On the other hand, it is generally accepted that the T-method has less fault detection capability than the other methods. Similarly, the methods could be compared based on their underlying assumptions. For example, the D, U, and W-methods all assume a *reset* capability whereas the T-method does not. The methods developed in this paper also assume a reset capability as well.

In addition, though it has not been proven *formally*, it is also generally accepted that the D, U, and W-methods have equivalent fault detection capabilities [Sabnani and Dhabura 1988]. The only exception to this, according to the literature, is that the W-method can detect "additional states" in the implementation, whereas the D and U-methods cannot. Furthermore, the fault detection capability of the W-method, which is explained below, has been formally proven by Chow [1978], whereas the same cannot be said for the D or U-methods.

From a theoretical, worst-case perspective, another disadvantage of the D and U-methods is that they require the construction of a *distinguishing entity* of some kind, and constructing such an entity is often intractable. For example, the D-method makes use of a string called a *distinguishing string*.[2] Yannakakis and Lee [1988] have shown that determining if a given finite-state machine has a distinguishing string is PSPACE-complete. Similarly, the U-method makes use of a string called a *unique input-output* (UIO) string.

[2]The reader should note that the *distinguishing string* referred to here is different from that defined in Section 2.

Yannakakis and Lee have also shown that determining if a given state in a finite-state machine has a UIO string is PSPACE-complete. What these results mean is that no polynomial-time algorithm exists for computing such strings, unless PSPACE = P. Furthermore, such strings will not exist for certain protocols, and even when they do exist, they can have exponential length.

It must of course be recognized that the above results are worst case in nature. In fact, in most cases UIO sequences are typically very short [Dhabura et al. 1990]. It must also be recognized that the T and W-methods do not require the computation of any such distinguishing entity. Though the W-method does make use of a set of strings called a *characterization set*, such a set will always exist, at least for a minimal finite-state automaton; the set will have polynomial size; and it can always be computed in polynomial time [Gill 1962].

The issue we wish to address in this paper is concerned with the primary disadvantage of the W-method. Specifically, we would like to know if there is a method that has all of the advantages of the W-method, yet generates a smaller resulting test set. In this paper we present three algorithms that do just this.

It should also be noted that a variety of improvements and modifications to the above methods have been suggested in other papers. For examples of such improvements, the interested reader is referred to papers by Fujiwara et al. [1991], Chen et al. [1990], and Miller and Paul [1991]. Also, the basic idea of the W-method is from Vasilevskii [1973].

1.3 Our Results

In this paper we present three modified versions of Chow's [1978] W-method. As stated above, the W-method has a number of advantages and disadvantages compared with other methods. In particular, let M_1 be a minimal Mealy machine with n states, and let M_2 be a minimal Mealy machine with at most m states. In addition, suppose that M_1 represents a specification and M_2 an implementation. Then the set S of inputs generated by the W-method can be used to determine if M_2 correctly implements M_1. In other words, the output produced by M_1 on the inputs in S will be identical to those produced by M_2 if and only if M_1 and M_2 is identical.

It should be noted that in no way are we claiming that the W-method is the "best" method. In particular, relative to most other methods, the W-method has the disadvantage that it generates extremely large test sets. On the other hand, it has the advantage that it's fault-tolerant capability can be precisely defined and proven. The primary value of the modified versions of the W-method presented here is that they have the advantages of the W-method (their fault-tolerant capability can be precisely defined and proven), yet at the same time, these versions of the W-method improve upon the main disadvantage of the W-method. Specifically, the algorithms presented here have the advantage that they will generate a much smaller number of tests in many cases.

Suppose M_1 and M_2 are minimal finite-state machines, as above. In addition, suppose that M_1 and M_2 both have input alphabet Σ, where $|\Sigma| = k$; let α be the total number of strings in a *characterization set* for M_1, and let β be the total number of strings in a *transition cover set* for M_1. The W-method will generate a test set consisting of $\alpha\beta(k^{m-n+1} - 1)/(k - 1)$ strings. In contrast, our first algorithm will generate a test set containing at most $\beta(\alpha + k^{m-n})$ strings; the second will contain $\beta k^{\max(n-1, m-n)}$, and the third $\beta(k^{n-1} + k^{m-n})$. When $m \gg n$, all three of our algorithms produce fewer strings than the W-method. Two of these algorithms make use of a heuristic for minimizing the number of strings in a characterization set. We show that the *performance ratio* for this heuristic has a logarithmic upper bound.

1.4 Applications to Software Testing

Though the immediate topic of this paper is protocol testing, it is important to note that the results described may be applicable to software testing as well. Finite-state machines have been used in the analysis and design of many different kinds of software systems, including compilers, real-time and object-oriented systems. Exactly how they are used in the development of different kinds of software tends to vary greatly. However, the possibility does exist that finite-state models, developed during either analysis or design, could be used in the generation of test data. Of course, before test data could be generated using protocol test generation methods one would first have to define the types of finite-state machines that were being used, and they would have to be developed and specified in a very precise manner. This is typically not done when finite-state models are used during development, but the approach is worthy of additional consideration.

2. DEFINITIONS

Throughout this paper we will make use of the following definitions. Let Σ be a finite alphabet, and let X and Y be sets of strings over Σ. The *concatenation* of X and Y, denoted $X \cdot Y$, is the set $\{w \mid x \in X, y \in Y \text{ and } w = xy\}$. The *union* of X and Y, denoted $X \cup Y$, is the set $\{w \mid w \in X \text{ or } w \in Y\}$. The *cardinality* of X, denoted $|X|$, is the total number of strings in X. Let ϵ denote the empty string, and let $X^0 = \{\epsilon\}$. Then X^i is the set $X \cdot X^{i-1}$, and X^* is the set $\bigcup_{i=0}^{\infty} X^i$. Note that since any alphabet Σ is a set of strings, each having length 1, the set Σ^i is the set of all strings over Σ having length i, and Σ^* is the set of all strings over Σ.

As mentioned in Section 1, the control portion of a protocol can be described by a Mealy machine $M = (Q, \Sigma, \Gamma, \delta, \gamma, q)$, where Q is a finite set of states; Σ is a finite input alphabet; Γ is a finite output alphabet; δ: $Q \times \Sigma \to Q$ is a transition function; γ: $Q \times \Sigma \to \Gamma$ is an output function, and $q \in Q$ is a starting state. For example, the Mealy machine in Figure 1 has $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{a, b\}$, and $q = q_0$. Note that if $p, q \in Q$, $s \in \Sigma$,

$t \in \Gamma$, $\delta(p, s) = q$, and $\gamma(p, s) = t$, then in Figure 1 we have shown an edge from p to q labeled s/t. Such a transition will be denoted by $p \xrightarrow{s/t} q$.

The function δ can be extended to a function on strings δ': $Q \times \Sigma^* \to Q$. Specifically, if $q \in Q$, $a \in \Sigma$, $y \in \Sigma^*$, and $w = ya$, then define $\delta'(q, \epsilon) = q$ and $\delta'(q, w) = \delta(\delta'(q, y), a)$. Similarly, γ can also be extended to γ': $Q \times \Sigma^* \to \Gamma^*$. If $q \in Q$, $a \in \Sigma$, $y \in \Sigma^*$, and $w = ya$, then define $\gamma'(q, \epsilon) = \epsilon$ and $\gamma'(q, w) = \gamma'(q, y)\gamma(\delta'(q, y), a)$. Throughout this paper we will use δ for δ' and γ for γ'. In other words, if $q \in Q$ and $w \in \Sigma^*$, then $\delta(q, w)$ is the state that M will be in after processing the string w when starting in state q. Similarly, $\gamma(q, w)$ will be the string produced as output when M processes w when starting in state q. For example, for the machine in Figure 1, $\delta(q_0, 00110) = q_1$ and $\gamma(q_0, 00110) = abbba$. In addition, let $w = w_0 w_1 \cdots w_{n-1}$ be a string, and let q be a state. Then the k *successor* of q (with respect to w) is the state $p = \delta(q, w_0 w_1 \cdots w_{k-1})$, where $k \geq 1$.

Let $p, q \in Q$ and $w \in \Sigma^*$. Then w is said to *distinguish* between p and q if $\gamma(p, w) \neq \gamma(q, w)$. In other words, if we consider two different computations of M on input w, one starting in state p and one starting in state q, and if these computations produce different outputs, then w distinguishes p and q. For example, for the machine shown in Figure 1, the string 110 will distinguish q_1 from q_2. A *characterization set* for M is a set $S \subseteq \Sigma^*$ such that for all $q_i, q_j \in Q$, where $i \neq j$, there exists a string $w \in S$ such that $\gamma(q_i, w) \neq \gamma(q_j, w)$. For example, for the machine shown in Figure 1, $\{110, 010, 10\}$ is a characterization set. If a machine M has a characterization set W, then M is said to be *minimal*.

Let $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, \gamma_1, q_1)$ and $M_2 = (Q_2, \Sigma, \Gamma, \delta_2, \gamma_2, p_1)$ be Mealy machines; let $q_i \in Q_1$, $p_j \in Q_2$, and let $S \subseteq \Sigma^*$. If $\gamma_1(q_i, w) = \gamma_2(p_j, w)$, for all $w \in S$, then q_i and p_j are *S-equivalent*. If q_i and p_j are S-equivalent for all $S \subseteq \Sigma^*$, then q_i and p_j are *equivalent*. If q_0 and p_0 are S-equivalent then M_1 and M_2 are *S-equivalent*. If q_0 and p_0 are equivalent then M_1 and M_2 are *equivalent*. Let f: $Q_1 \to Q_2$ be a one-to-one and onto function such that

(1) $f(q_1) = p_1$ and

(2) $q_i \xrightarrow{x/y} q_j$ if and only if $f(q_i) \xrightarrow{x/y} f(q_j)$, for all $q_i, q_j \in Q_1$, $x \in \Sigma$, and $y \in \Gamma$.

Then f is said to be an *isomorphism* from M_1 to M_2. Finally, if there is an isomorphism from M_1 to M_2, then M_1 and M_2 are *isomorphic*.

The reader should note that the above definition of a characterization set differs from that sometimes encountered in the literature. For example, Sidhu and Leung [1989] define a characterization set W for a Mealy machine M to be a set of strings such that the last output symbols observed from the application of these strings are different at each state of M. On the other hand, Chow [1978] and Fujiwara et al. [1991] define a characterization set to be a set of input sequences that can distinguish between the behaviors of every pair of states in S (not necessarily on the last output symbols). For this paper, we use the definition given above. Though it is stated slightly differently, the above definition is equivalent to the one given by Chow and

Fujiwara. We require this definition in order to apply the minimization heuristic in Section 5.

3. THE IMPROVED W-METHOD

The W-method solves a formal version of the protocol-testing problem that we call the *Mealy machine equivalence problem*. In this problem we are given two Mealy machines $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, \gamma_1, q_1)$ and $M_2 = (Q_2, \Sigma, \Gamma, \delta_2, \gamma_2, p_1)$, where M_1 is referred to as the *specification*, and M_2 is referred to as the *implementation*. It is assumed that M_1 is minimal and strongly connected, i.e., that given any two states p and q in M_1, there is always a path from p to q. It is also assumed that $|Q_1| = n$ and $|Q_2| = m$, where $m \geq n$.

The test set generated by the W-method will result from the concatenation of two sets of strings P and Z. The set P, called a *transition cover set*, is any set of input sequences such that $\epsilon \in P$ and for all $p, q \in Q$ and $x \in \Sigma$, where $\delta(p, x) = q$, there exist $w, wx \in P$ such that $\delta(q_1, w) = p$. In other words, the set P is any set of strings such that for any transition from a state p to another state q on input symbol x, there will exist input strings w and wx in P such that w causes M_1 to enter state p from the starting state q_1. Thus, the string wx will result in a computation that starts at q_1, proceeds to p, and then traverses the transition to q. An algorithm for constructing a transition cover set is easy to design. Such an algorithm is presented by Chow [1978].

The set Z will be constructed as follows. Given that M_1 has a minimal number of states, a string w can always be constructed that distinguishes between any two given states in M_1. An algorithm for constructing such a string is given by Algorithm 4.1 by Gill [1962]. It is also shown that such a string will consist of at most $n - 1$ symbols. Thus, a characterization set W for M_1 can be formed by constructing a distinguishing string for every pair of states in M_1. Thus, the resulting set will contain $n(n-1)/2$ strings.[3] Finally, Z is defined to be the set $\bigcup_{i=0}^{m-n} \Sigma^i \cdot W$:

$$Z = W \cup \Sigma^1 \cdot W \cup \Sigma^2 \cdot W \cup \cdots \cup \Sigma^{m-n} \cdot W \tag{1}$$

The fault detection capability of the W-method is summarized by the following theorem due to Chow [1978].

THEOREM 3.1. *Let M_1 be a minimal Mealy machine with n states, and let M_2 be a minimal Mealy machine with at most m states. In addition, suppose that they both have input alphabet Σ. Let P be a transition cover set for M_1. Let W be a characterization set for M_1, and let $Z = \bigcup_{i=0}^{m-n} \Sigma^i \cdot W$. Then M_1 and M_2 are equivalent if and only if M_1 and M_2 are $P \cdot Z$-equivalent.*

Recall that in the original protocol-testing problem, as stated in Section 1.1, we are given a Mealy machine M_1 and an implementation S of M_1. Assuming that S implements some Mealy machine M_2 with at most $m \geq n$ states, Theorem 3.1 provides a method to determine if S correctly implements M_1. Specifically, the set $P \cdot Z$ is constructed, and then the outputs produced by

[3] In Section 5 we will describe a heuristic for minimizing the number of strings in the set W.

M_1 and S are compared on every string in $P \cdot Z$. Assuming that S implements some Mealy machine with $m \geq n$ states, then from Theorem 3.1, S correctly implements M_1 if and only if they produce the same output on every string in $P \cdot Z$.

In practice, it is unlikely that we will have an upper bound value for m. Consequently, one way to apply Theorem 3.1 in the testing process is to rule out larger and larger Mealy machines successively that S may actually implement. In other words, first test S under the assumption that $m = n$, in which case we use the set $P \cdot Z$, where $Z = \bigcup_{i=0}^{m-n} \Sigma^i \cdot W$ and $m - n = 0$. If M_1 and S produce equivalent outputs on all of these strings, then this rules out the possibility that S implements some Mealy machine M_2 that is not equivalent to M_1, but has the same number of states. Next, we test S using the set $P \cdot Z$, where $Z = \bigcup_{i=0}^{m-n} \Sigma^i \cdot W$ and $m - n = 1$. If M_1 and S produce equivalent outputs on all of these strings, then this rules out the possibility that S implements some Mealy machine M_2 that is not equivalent to M_1, but has one more state than M_1. Next, we test S using the set $P \cdot Z$, where $Z = \bigcup_{i=0}^{m-n} \Sigma^i \cdot W$ and $m - n = 2$, etc. Thus, we rule out larger and larger possible machines for S, until we are satisfied.

One of our improvements to the W-method results from an alternative set Z. Specifically, in our method we construct a test set by concatenating two sets P and Z, where P is defined just as in the W-method, but where

$$Z = W \cup \Sigma^{m-n}. \tag{2}$$

Using the above definitions of P and Z, we can prove the following theorem.

THEOREM 3.2. *Let M_1 be a minimal Mealy machine with n states, and let M_2 be a minimal Mealy machine with at most m states. In addition, suppose that they both have input alphabet Σ. Let P be a transition cover set for M_1. Let W be a characterization set for M_1, and let $Z = W \cup \Sigma^{m-n}$. Then M_1 and M_2 are equivalent if and only if M_1 and M_2 are $P \cdot Z$-equivalent.*

PROOF. We refer the interested reader to Corollary A.9 in the Appendix. □

In fact, we can prove the following theorem.

THEOREM 3.3. *Let M_1 be a minimal Mealy machine with n states, and let M_2 be a minimal Mealy machine with at most m states. In addition, suppose that they both have input alphabet Σ. Let P be a transition cover set for M_1. Let W be a characterization set for M_1, and let $Z = W \cup \Sigma^{m-n}$. Then M_1 and M_2 are isomorphic if and only if M_1 and M_2 are $P \cdot Z$-equivalent.*

PROOF. We refer the interested reader to Theorem A.8 in the Appendix. □

The above improvement results from the fact that the set Z given in (1) will in most cases be larger than the set Z given in (2). This is particularly true when m is larger than n. Specifically, suppose that M_1 and M_2 have n and m states, respectively, where $m \geq n$. In addition, let the input alphabet for M_1 and M_2 be Σ, where $|\Sigma| = k$. Then the W-method will generate a test

set consisting of $|P||W|(k^{m-n+1}-1)/(k-1)$ strings. In contrast, using the definition of Z as in (2) will result in a test set with $|P|(|W|+k^{m-n})$ strings. Thus, given a Mealy machine M and an implementation S, we can repeatedly test S using successively larger values of m. In other words, we can test S using $P \cdot Z$, where $Z = W \cup \Sigma^{m-n=0}$, $Z = W \cup \Sigma^{m-n=1}$, $Z = W \cup \Sigma^{m-n=2}$, etc., until we are satisfied. Practically speaking, by using the set Z as defined in (2), we will be able to continue this process much longer than if we used the set Z as defined in (1).

One thing that follows from Theorem 3.3 is that if $m = n$ then $Z = W \cup \Sigma^{m-n} = W \cup \{\epsilon\}$. In addition, it is also worth noting that if a single distinguishing string s does actually exist, i.e., a string that distinguishes all pairs of states, then $\{s\} \cup \Sigma^{m-n}$ is sufficient for the set Z. Finally, if a single distinguishing string s does exist and $m = n$ then $\{s\} \cup \{\epsilon\}$ is sufficient for Z.

4. AN ALTERNATIVE CHARACTERIZATION SET

As described in Section 3, our first version of the W-method makes use of a characterization set W, which is computed from a given Mealy machine M. By definition, such a set will contain a string that distinguishes any two states in M. In addition, based on the assumption that the specification machine M has a minimal number of states, say n, it can be shown that any two states have a distinguishing string of length at most $n - 1$ [Gill 1962]. Thus, we can prove the following lemma.

LEMMA 4.1. *Let M be a Mealy machine with a minimal number of states n. Then $\bigcup_{i=1}^{n-1} \Sigma^i$ is a characterization set for M.*

PROOF. This result follows from the fact that $\bigcup_{i=1}^{n-1} \Sigma^i$ contains all strings of length at least 1 and at most $n - 1$, and since any two states in a minimal state machine can be distinguished by a string of length at most $n - 1$. □

LEMMA 4.2. *Let M be a Mealy machine with n states and input alphabet Σ. In addition, suppose that M contains a minimal number of states. Then Σ^{n-1} is a characterization set for M.*

PROOF. Lemma 4.1 states that $\bigcup_{i=1}^{n-1} \Sigma^i$ is a characterization set for M. Suppose that some string $w \in \Sigma^k$ distinguishes between states p and q in M, where $1 \leq k < n - 1$. Then since w is the prefix of some string $x \in \Sigma^{n-1}$ it follows that x distinguishes p and q as well. Thus, only strings in Σ^{n-1} are needed to distinguish all pairs of states in M. □

THEOREM 4.3. *Let M_1 be a minimal Mealy machine with n states, and let M_2 be a minimal Mealy machine with at most m states. In addition, suppose that they both have input alphabet Σ. Let P be a transition cover set for M_1, and let $Z = \Sigma^{max(n-1, m-n)}$. Then M_1 and M_2 are equivalent if and only if M_1 and M_2 are $P \cdot Z$-equivalent.*

PROOF. From Lemma 4.2, Σ^{n-1} is a characterization set for M_1. It follows that Theorem 3.2 holds when $Z = \Sigma^{n-1} \cup \Sigma^{m-n}$. Now consider $P \cdot Z = P \cdot \Sigma^{n-1} \cup P \cdot \Sigma^{m-n}$, and first suppose that $\max(n - 1, m - n) = n - 1$. It fol-

lows that every string $w \in P \cdot \Sigma^{m-n}$ is the prefix of some string in $x \in P \cdot \Sigma^{n-1}$. Thus, M_1 and M_2 are equivalent with respect to $P \cdot \Sigma^{n-1} \cup P \cdot \Sigma^{m-n}$ if and only if they are equivalent with respect to $P \cdot \Sigma^{n-1}$. On the other hand, if $\max(n-1, m-n) = m-n$ then every string $w \in P \cdot \Sigma^{n-1}$ is the prefix of some string in $x \in P \cdot \Sigma^{m-n}$. Thus, M_1 and M_2 are equivalent with respect to $P \cdot \Sigma^{n-1} \cup P \cdot \Sigma^{m-n}$ if and only if they are equivalent with respect to $P \cdot \Sigma^{m-n}$. □

Note that Theorem 4.3 suggests another method for generating a test set for solving the Mealy machine equivalence problem. Specifically, we can construct the set $Z = \Sigma^{\max(n-1, m-n)}$, which is then used in the set $P \cdot Z$.

5. MINIMIZING CHARACTERIZATION SETS

In many of the known protocol-testing procedures, an additional minimization algorithm is applied to a given test set. Often times, these minimization procedures are based on the observation that if a test set contains two strings, one of which is a prefix of the other, then only the longer string is required, and the shorter string can be removed from the set. Examples of such minimization procedures are given by Sabnani and Dahbura [1988], and by Sidhu and Leung [1989]. Similarly, we made use of this idea in Theorem 4.3. In this section, we present a minimization heuristic that is not based on eliminating redundant prefixes. This procedure provides another method for minimizing the size of test sets.

Let $M = (Q, \Sigma, \Gamma, \delta, \gamma, q_1)$ be a Mealy machine containing a minimal number of states $n = |Q|$. As stated in Section 3, given two states $q_i, q_j \in Q$, a string $w \in \Sigma^*$ can be computed that distinguishes q_i and q_j. In addition, such a string can be computed in polynomial time and will have length at most $n-1$. Thus, a characterization set W for M can be constructed in polynomial time by constructing such a string for every pair of states.

Let W be a characterization set for M. Though it is not necessary, for the sake of discussion we will assume that W was constructed as described above. Another method to minimize the set $P \cdot Z$ is based on the observation that any particular string $w \in W$ may distinguish between more than one pair of states in Q. For example, consider the Mealy machine shown in Figure 1. Note that the string 110 distinguishes not only q_0 from q_1, but it also distinguishes q_1 from q_2. Thus, if 110 is placed in a characterization set for the purpose of distinguishing q_0 from $q_1$in M, then no additional string is required to distinguish q_1 from q_2. Stated another way, given any characterization set (for example, as constructed by the above procedure), that set may contain redundant strings. The question we wish to address in this section is whether or not there are efficient algorithms for removing such strings.

More formally, we can define the following problem called the *characterization set minimization problem*, abbreviated *CSM*, as follows.

INSTANCE: Mealy machine $M = (Q, \Sigma, \Gamma, \delta, \gamma, q_1)$, characterization set $W \subseteq \Sigma^*$.

PROBLEM: Construct a set $W' \subseteq W$ such that both W' is also a characterization set for M and $|W'|$ is minimized.

In the following we will present a polynomial-time heuristic for this problem. Though the heuristic is not guaranteed to solve the problem optimally, we can provide an upper bound on the *performance ratio* for the heuristic. In doing so we shall make use of the following definitions.[4]

Let M and W be an instance of CSM. Then a feasible solution for M and W is a subset $W' \subseteq W$ such that W' is a characterization set for M. An *optimal solution* for M and W is a feasible solution W' such that for any other feasible solution W'', $|W'| \leq W''$. Now let P be a heuristic for CSM. The *performance ratio* for P is defined as $R_P(M, W) = \mathrm{P}(M, W)/\mathrm{OPT}(M, W)$, where $\mathrm{P}(M, W)$ is the total number of strings resulting when heuristic P is applied to M and W,in the worst case, and where $\mathrm{OPT}(M, W)$ is the total number of strings in an optimal solution [Garey and Johnson 1979]. Generally, we would like to identify a heuristic whose performance ratio can be bounded from above by a slowly growing function, and ideally, by a small constant factor. Such an upper bound would indicate how close the heuristics solution would be to an optimal solution.

In this section we will present a heuristic P for CSM such that $R_P(M, W) \leq O(\log k)$. In doing so we will make use of the following problem called *set cover* (SC) [Johnson 1974].

INSTANCE: Collection C of subsets of a finite set A.
PROBLEM: Construct a subcollection $C' \subseteq C$ such that every element of A belongs to at least one member of C' and such that $|C'|$ is minimized.

In other words, we would like to obtain a subcollection C' of the sets in C, such that C' contains as few sets as possible and such that C' covers A. Such a subcollection is referred to as a *minimal cover* for C. For example, let $A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$ and $C = \{\{a_1, a_2\}, \{a_3, a_4\}, \{a_5, a_6\}, \{a_1, a_3, a_5\}, \{a_2, a_4, a_6\}\}$. Then $C' = \{\{a_1, a_3, a_5\}, \{a_2, a_4, a_6\}\}$ would be minimum cover for C.

Fact. SC is NP-hard [Garey and Johnson 1979].

Now consider the heuristic for SC, called *h1*, shown in Figure 2.

Fact. $R_{h1}(S, C)$ is $O(\log k)$ [Johnson 1974].

Using this result we can now show how to construct a heuristic *h2* for the CSM problem that has a similar upper bound.

THEOREM 5.1. *$R_{h2}(M, W)$ is $O(\log k)$.*

PROOF. First we will show how to transform, in polynomial time, any given instance of CSM to an instance of SC. The transformation and the heuristic *h1* together form the heuristic *h2*. We will then show that this

[4]Note that we leave open the problem of whether or not this problem is NP-Hard. As we shall show, this problem can be transformed to the set-cover problem, which is known to be NP-hard. However, this does not show that CSM is NP-hard. To do so would require that the reduction be performed in the other direction.

```
procedure H1(C = {S_1, S_2, ..., S_N});
input  : collection C of N sets
output : a subcollection C' ⊆ C such that
         ⋃_{S∈C'} S = ⋃_{S∈C} S and |C'| is minimized.
begin
    Sub:=∅;
    UnCov:=⋃_{S∈C} S;
    N:=|C|;
    Set[i]:=S_i, 1 ≤ i ≤ N;
    while (UnCov≠ ∅) do
        j = max{|Set[j]| such that 1 ≤ j ≤ N};
        Sub:=Sub⋃{S_j};
        UnCov:=UnCov-Set[j];
        for i := 1 to N do
            Set[i]:=Set[i]-Set[j];
        end;
    end;
    return Sub;
end;
```

Fig 2. An algorithm for constructing an approximation of a minimum cover.

transformation is *approximation preserving*. In other words, we will show that since $R_{h1}(S) \leq O(\log k)$, it follows that $R_{h2}(M, W) \leq O(\log k)$.

Let $M = (Q, \Sigma, \Gamma, \delta, \gamma, q_1)$ be a Mealy machine, and let $W \subset \Sigma^*$ be a finite characterization set for M, where $W = \{w_1, w_2, \ldots, w_p\}$. The idea of the conversion is to construct a set A and a collection C of subsets of the elements in A. First, for each pair of states $q_i, q_j \in Q$, an element $a_{i,j}$ is placed in A. The idea is that each element in A will correspond to one pair of states in Q. In addition, for each string $w_i \in W$ we will construct a corresponding set of elements $\{a_{i,1}, a_{i,2}, \ldots, a_{i,t}\}$ to be placed in C. The idea is that each element in the set $\{a_{i,1}, a_{i,2}, \ldots, a_{i,t}\}$ represents a pair of states that are distinguished by the string w_i. Thus, the set corresponding to w_i will contain all of those elements that correspond to pairs of states distinguished by w_i. Note that the set of elements corresponding to each string w_i can be computed easily in polynomial time from W and M, simply by determining which pairs of states are distinguished by w_i. In addition, it can easily be verified that a minimum cover for C and A corresponds to a minimum cardinality subset of W that is a characterization set for M. Consequently, given an approximate minimum cover $C' \subseteq C$ constructed by the heuristic *h1*, the corresponding characterization set $W' \subseteq W$ can be reconstructed directly.

We must now show that this transformation is approximation preserving. As stated above, it is easy to verify that a minimum solution to the CSM problem corresponds to a minimum solution to the SC problem. However, we must also show that if *h1* comes to within a logarithmic factor of the

optimum, then so does $h2$. To do this, let I be an instance of CSM consisting of Mealy machine M and characterization set W, where k_1 is the total number of states in M and k_2 is the total number of strings in W. In addition, let I' be the corresponding instance resulting from the above transformation. As stated above, it is easy to verify that $\mathrm{OPT}(M, W) = \mathrm{OPT}(A, C)$. In addition, by construction $h2(M, W) = h1(A, C)$. It follows that

$$R_{h2}(M, W) = R_{h1}(A, C). \tag{3}$$

As stated above, $R_{h1}(A, C)$ is $O(\log k)$ which, by definition, means that

$$R_{h1}(A, C) \leq c \log k \tag{4}$$

for some positive constant c, where k represents the input length of the SC instance consisting of A and C. Now let the total length of the instance I be l. Since an element is placed in A for each pair of states in M, it follows that A will contain at most $k_1 \times k_1$ elements. In addition, since each set in C will contain at most $k_1 \times k_1$ elements from A, it follows that the size of all the sets in C combined will be at most $|C| \times k_1 \times k_1 = k_2 \times k_1 \times k_1$. It follows that the length k of the instance I' is at most $(k_1 \times k_1) + (k_2 \times k_1 \times k_1)$. Since $k_1, k_2 \leq l$, it follows that $k \leq 2l^3$. Hence,

$$c \log k \leq c \log(2l^3) \tag{5}$$

$$= 3c \log(2l) \tag{6}$$

$$\leq c' \log l \tag{7}$$

for some positive constant c'. From (3), (4), and (7) it follows that $R_{h2}(M, W)$ is $O(\log l)$. □

The idea used in Theorem 5.1 has actually a more general application in the context of software testing. Often times one has a set of requirements $R = \{r_1, r_2, \ldots, r_k\}$ that some software S must satisfy. To verify that S satisfies the requirements, a set of tests $T = \{t_1, t_2, \ldots, t_l\}$ is devised such that each requirement in R is tested, or rather "covered," by at least one of the tests in T. As with a characterization set, it may be the case that any particular test covers more than one requirement. Consequently, the set T could be redundant, in the sense that there may exist a subset $T' \subset T$ such that the tests in T' cover all the requirements in R. As in Theorem 5.1, the sets R and T can be converted to the set cover problem, and the heuristic $h1$ can be applied to minimize the total number of tests.

The minimization heuristic $h2$, combined with the results from Sections 3 and 4, suggest three different ways to generate a test set, as shown in Figure 3. Let M_1 and M_2 be Mealy machines that have n and m states, respectively, where $m \geq n$. In addition, let M_1 and M_2 have input alphabet Σ, where $|\Sigma| = k$. Let α be the total number of strings in a *characterization set* for M_1, and let β be the total number of strings in a *transition cover set* for M_1. Then the W-method will generate a test set consisting of $\alpha\beta(k^{m-n+1} - 1)/(k - 1)$ strings. In contrast, our first algo-

```
procedure generate1(M,m);
input  : minimal Mealy machine M with n states
         and input alphabet ∑, upper bound m ≥ n;
output : a set of strings over ∑;
begin
   W:=∅;
   for each pair q_i, q_j ∈ ∑ do
      construct a distinguishing string x for q_i and q_j;
      W:=W ⋃{x};
   end;
   minimize W using h2 to get W';
   construct a transition cover set P for M;
   return P · (W' ⋃ ∑^(m-n));
end;

procedure generate2(M,m);
input  : minimal Mealy machine M with n states
         and input alphabet ∑, upper bound m ≥ n;
output : a set of strings over ∑;
begin
   construct a transition cover set P for M;
   return P · ∑^max(n-1,m-n);
end;

procedure generate3(M,m);
input  : minimal Mealy machine M with n states
         and input alphabet ∑, upper bound m ≥ n;
output : a set of strings over ∑;
begin
   W:=∑^(n-1);
   minimize W using h2 to get W';
   construct a transition cover set P for M;
   return P · (W' ⋃ ∑^(m-n));
end;
```

Fig. 3. Algorithms for constructing a test set for Mealy machine M.

rithm will generate a test set consisting of at most $\beta(\alpha + k^{m-n})$ strings. Our second algorithm will generate a test set consisting of $\beta k^{\max(n-1, m-n)}$ strings, and our third, at most $\beta(k^{n-1} + k^{m-n})$. When $m \gg n$, all three of our algorithms will produce fewer strings than the W-method.

6. CONCLUSIONS

In this paper we have considered the protocol-testing problem. Several methods for solving this problem have been presented in the literature. In this

paper we have presented three new versions of the W-method due to Chow [1978]. We have shown that these algorithms all have the same fault detection capability as the W-method. However, in most cases our algorithms will generate a far shorter test set. Two of these algorithms make use of a heuristic for minimizing the number of strings in a characterization set. We have shown that the performance ratio for this heuristic has an upper bound of $O(\log k)$.

Though our improvements are encouraging, much work still remains to be done. Specifically, we have left open the issue of which of the three methods generates a smaller set. This would of course depend on several factors, such as the size of m and n. Experimentation would probably be helpful in this regard. In addition, though the set of strings constructed by our methods is smaller than that constructed by the W-method, our methods still construct test sets that contain an exponential number of symbols. This leaves open the possibility that there may be a method for constructing a test set that consists of a polynomial number of symbols, while at the same time having the same fault detection capability. Finally, insight would also be gained by an empirical comparison of our algorithms to each other, as well as to Chow's.

APPENDIX

OBSERVATION A.1. *Let $M = (Q, \Sigma, \Gamma, \delta, \gamma, q_1)$ be a Mealy machine. Let $p, q \in Q$, $w, z \in \Sigma^*$, where w is a prefix of z, and suppose that w distinguishes p and q. Then z will distinguish p and q.*

LEMMA A.2. *Let $M = (Q, \Sigma, \Gamma, \delta, \gamma, q_1)$ be a minimal Mealy machine, and let $S \subset \Sigma^*$ be a set of strings that partitions the states of M into at least α equivalence classes, where $0 \leq \alpha \leq m = |Q|$. Then $S \cup \Sigma^i$ partitions Q into at least $\alpha + i$ equivalence classes, for all $0 \leq i \leq m - \alpha$.*

PROOF. By induction on i.

Basis. $i = 0$. Since $S \subset \Sigma^*$ partitions the states of M into at least α equivalence classes, it follows directly that $S \cup \Sigma^i = S \cup \{\epsilon\}$ partitions the states of M into at least $\alpha + i = \alpha$ equivalence classes.

Inductive Hypothesis. Suppose there exists a k, where $0 \leq k \leq m - \alpha - 1$, such that $S \cup \Sigma^k$ partitions Q into at least $\alpha + k$ equivalence classes.

Inductive Step. We will show that $S \cup \Sigma^{k+1}$ partitions Q into at least $\alpha + k + 1$ equivalence classes. First, by the inductive hypothesis $S \cup \Sigma^k$ partitions Q into at least $\alpha + k$ equivalence classes. In fact, it may be the case that $S \cup \Sigma^k$ partitions Q into at least $\alpha + k + 1$ equivalence classes in which case, by Observation A.1, it follows that $S \cup \Sigma^{k+1}$ also partitions Q into at least $\alpha + k + 1$ equivalence classes. So suppose that $S \cup \Sigma^k$ does not partition Q into at least $\alpha + k + 1$ equivalence classes. It follows, by the

inductive hypothesis, that $S \cup \Sigma^k$ partitions Q into *exactly* $\alpha + k$ equivalence classes. In addition, since $0 \leq k \leq m - \alpha - 1$, it follows that the total number of equivalence classes is $\alpha + k \leq m - 1$, which is less than the number of states m in Q. It then follows that there must be two states $q_i, q_j \in Q$ that are not distinguished by any string in $S \cup \Sigma^k$. However, since M is minimal, it follows that q_i and q_j are distinguished by some string. Suppose that the shortest such string is of length $t > k$. Thus q_i and q_j are distinguished by a string of length t but not by any string of length $t - 1$ or less. In other words q_i and q_j are distinguished by some string in Σ^t but by no string in Σ^{t-1}. Let w be a string in Σ^t that distinguishes q_i and q_j, and consider the states $q_i', q_j' \in Q$ that are the $(t - k - 1)$ successors of q_i and q_j, respectively, when processing w. Then q_i' and q_j' must be distinguished by a string of length $k + 1$ but by no string of length k. Otherwise, q_i and q_j would be distinguished by a string of length $t - 1$. It follows that q_i' and q_j' are distinguished by Σ^{k+1} but not by Σ^k. Hence, $S \cup \Sigma^{k+1}$ partitions the states in Q into at least $\alpha + k + 1$ equivalence classes. □

LEMMA A.3. *Let* $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, \gamma_1, q_1)$ *and* $M_2 = (Q_2, \Sigma, \Gamma, \delta_2, \gamma_2, p_1)$ *be Mealy machines, where* $n = |Q_1|$, $m = |Q_2|$, *and* $m \geq n$. *Let* P *be a transition cover set for* M_1, *and let* W *be a characterization set for* M_1. *If* M_1 *and* M_2 *are* $P \cdot W$*-equivalent then for every* $q_i \in Q_1$ *there exists at least one* $p_j \in Q_2$ *such that* q_i *and* p_j *are* W*-equivalent. In addition, for no other* $q_k \in Q_1$, *where* $k \neq i$, *is it the case that* q_k *and* p_j *are also* W*-equivalent.*

PROOF. Suppose that M_1 and M_2 are $P \cdot W$-equivalent. By definition of P, for every $q_i \in Q_1$ there exists a $w \in P$ such that $\delta_1(q_1, w) = q_i$. Let $\delta_2(p_1, w) = p_j$. Since M_1 and M_2 cannot be distinguished by any string in $P \cdot W$, it follows that q_i and p_j cannot be distinguished by a string in W. Thus q_i and p_j are W-equivalent.

Now suppose there exists another state $q_k \in Q_1$ such that $k \neq i$, and such that q_k and p_j are also W-equivalent. It would then follow that q_i and q_k would be W-equivalent, a contradiction to the fact that W is a characterization set for M_1. □

LEMMA A.4. *Let* $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, \gamma_1, q_1)$ *and* $M_2 = (Q_2, \Sigma, \Gamma, \delta_2, \gamma_2, p_1)$ *be Mealy machines, where* $n = |Q_1|$, $m = |Q_2|$, *and* $m \geq n$. *Let* P *be a transition cover set for* M_1. *Let* W *be a characterization set for* M_1, *and let* $Z = W \cup \Sigma^{m-n}$. *If* M_1 *and* M_2 *are* $P \cdot W$*-equivalent then* Z *will partition the states* Q_2 *into* $m = |Q_2|$ *classes.*

PROOF. From Lemma A.3 it follows that each state in Q_1 is W-equivalent to a unique state in Q_2. Thus W-equivalence partitions the states in Q_2 into at least $n = |Q_1|$ classes. The result then follows from Lemma A.2. □

LEMMA A.5. *Let* $M_1 = (Q_1, \Sigma, \delta_1, \gamma_1, q_1)$ *and* $M_2 = (Q_2, \Sigma, \delta_2, \gamma_2, p_1)$ *be Mealy machines, where* $n = |Q_1|$, $m = |Q_2|$, *and* $m \geq n$. *Let* W *be a characterization set for* M_1. *Let* W *be a characterization set for* M_1. *Let* $q_i, q_j \in Q_1$,

$x \in \Sigma$ and $y \in \Gamma$, where $q_i \xrightarrow{x/y} q_j$, and let $Z = W \cup \Sigma^{m-n}$. If M_1 and M_2 are $P \cdot Z$-equivalent then there exist states $p_k, p_l \in Q_2$ such that p_k and p_l are Z-equivalent to q_i and q_j, respectively, and $p_k \xrightarrow{x/y} p_l$.

PROOF. Suppose that $q_i, q_j \in Q_1$, $x \in \Sigma$, and $y \in \Gamma$, where $q_i \xrightarrow{x/y} q_j$. By definition of P there exists a $w \in \Sigma^*$ and $z \in \Gamma^*$ such that w, $wx \in P$ and $q_1 \xrightarrow{w/z} q_i \xrightarrow{x/y} q_j$. Let $p_k, p_l \in Q_2$, $z' \in \Gamma^*$, and $y' \in \Gamma$ be such that $p_1 \xrightarrow{w/z'} p_k \xrightarrow{x/y'} p_l$. Since M_1 and M_2 are $P \cdot Z$-equivalent, and in particular $\{w\} \cdot Z$-equivalent and $\{wx\} \cdot Z$-equivalent, it follows that q_i and q_j are Z-equivalent to p_k and p_l, respectively, and $y = y'$. □

LEMMA A.6. *Let $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, \gamma_1, q_1)$ and $M_2 = (Q_2, \Sigma, \Gamma, \delta_2, \gamma_2, p_1)$ be Mealy machines, where $n = |Q_1|$, $m = |Q_2|$, and $m \geq n$. Let P be a transition cover set for M_1. Let W be a characterization set for M_1, and let $Z = W \cup \Sigma^{m-n}$. If M_1 and M_2 are $P \cdot Z$-equivalent, then Z-equivalence is an isomorphism from M_1 to M_2.*

PROOF. We will prove this in several steps. First we will show that Z-equivalence defines a function from the states in M_1 to those in M_2. We will then show that this function is one-to-one and onto. Finally, we will show that both conditions in the definition of isomorphism are satisfied.

(a) *Z-equivalence defines a function from Q_1 to Q_2.*

Since M_1 and M_2 are $P \cdot Z$-equivalent, it follows from Lemma A.3 that Z-equivalence maps each state in Q_1 to at *least* one state in Q_2. In addition, it must also be the case that each state in Q_1 is mapped to at *most* one state in Q_2. Otherwise, if Z-equivalence mapped some state in Q_1 to two states in Q_2, say p_i and p_j, then those two states would be Z-equivalent, which would contradict Lemma A.4. Thus, Z-equivalence defines a function from Q_1 to Q_2.

(b) *Z-equivalence defines a one-to-one function from Q_1 to Q_2.*

Suppose that the function defined by Z-equivalence from Q_1 to Q_2 was not one-to-one, and suppose that states $q_i, q_j \in Q_1$ mapped to the same state in Q_2. It would then follow that q_i and q_j would be Z-equivalent. Since $W \subseteq Z$, it would follow that q_i and q_j were also W-equivalent, which would contradict the fact that W is a characterization set for M_1.

(c) *Z-equivalence defines an onto function from Q_1 to Q_2.*

Suppose that that Z-equivalence did not map any state in Q_1 to some state $p_j \in Q_2$. Since p_j is reachable in Q_2, there is a path from p_1 to p_j in M_2. Assume without loss of generality that p_j is the only state along this path that is not mapped to from any state in Q_1 by Z-equivalence. (Otherwise, use the first such state along the path in the following.) Now consider the state $p_i \in Q_2$ preceding p_j on this path, and suppose that $p_i \xrightarrow{x/y} p_j$, where $x \in \Sigma$

and $y \in \Gamma$. In addition, let q_k be the state in Q_1 that Z-equivalence maps to p_i, and suppose that $q_k \xrightarrow{x/y'} q_l$, for some $y' \in \Gamma$ and $q_l \in Q_1$. Such a state q_l must exist since the transition function δ is a function. Now suppose that Z-equivalence maps q_l to some state p_s where $s \neq j$. Such a state p_s must exist since, from part (b), Z-equivalence defines a one-to-one function. Since M_1 and M_2 are $P \cdot Z$-equivalent, it follows from Lemma A.5 that $p_i \xrightarrow{x/y'} p_s$, for some $y' \in \Gamma$, in which case there are two transitions from p_i on input symbol x, one to p_j and one to p_s. But then M_2 is not a deterministic Mealy machine, a contradiction.

Note that from (a) to (c) it follows that Z-equivalence defines a one-to-one and onto function from Q_1 to Q_2. In the following, we will refer to this function as f. We will now show that this function is an isomorphism.

(d) *Z-equivalence maps q_1 to p_1, i.e., $f(q_1) = p_1$.*

Since M_1 and M_2 are $P \cdot Z$-equivalent, and since $\epsilon \in P$, it follows that q_1 and p_1 are Z-equivalent. Thus Z-equivalence maps q_1 to p_1.

(e) *$q_i \xrightarrow{x/y} q_j$ if and only if $f(q_i) \xrightarrow{x/y} f(q_j)$, for all $q_i, q_j \in Q_1$, $x \in \Sigma$, and $y \in \Gamma$.*

(if) Suppose that $f(q_i) \xrightarrow{x/y} f(q_j)$, for some $q_i, q_j \in Q_1$, $x \in \Sigma$, and $y \in \Gamma$. We will show that $q_i \xrightarrow{x/y} q_j$. Suppose to the contrary that $q_i \xrightarrow{x/y'} q_k$, for some $y' \in \Gamma$, but that $q_j \neq q_k$. Since M_1 and M_2 are $P \cdot Z$-equivalent, by Lemma A.5 it follows that $f(q_i) \xrightarrow{x/y'} f(q_k)$. Furthermore, it must be the case that $f(q_k) \neq f(q_j)$; otherwise Z-equivalence would not define a one-to-one function from Q_1 to Q_2. But then we have two transitions from $f(q_i)$ on input symbol x, one to $f(q_j)$ and one to $f(q_k)$. It follows that M_2 is not deterministic, a contradiction.

Now suppose that $q_i \xrightarrow{x/y'} q_j$, but that $y \neq y'$. By definition there exists a string $w \in P$ such that $\delta_1(q_1, w) = q_i$ and $\delta_2(p_1, w) = f(q_i)$. It follows that the string $wx \in P$ distinguishes q_i and $f(q_i)$. Since $wx \in P \cdot Z$, this contradicts the fact that M_1 and M_2 are $P \cdot Z$-equivalent.

(only if) Now suppose that $q_i \xrightarrow{x/y} q_j$ for some $q_i, q_j \in Q_1$, $x \in \Sigma$, and $y \in \Gamma$. Since M_1 and M_2 are $P \cdot Z$-equivalent, by Lemma A.5 it follows that $f(q_i) \xrightarrow{x/y} f(q_j)$.

From (d) and (e) it follows that Z-equivalence defines an isomorphism from M_1 to M_2. □

LEMMA A.7. *Let M_1 and M_2 be Mealy machines. If M_1 and M_2 are isomorphic then M_1 and M_2 are equivalent.*

PROOF. This proof is omitted since it is a well-known result. See for example Hopcroft and Ullman [1979]. □

THEOREM A.8. *Let M_1 and M_2 be Mealy machines. Let P be a transition cover set for M_1. Let W be a characterization set for M_1, and let $Z = W \cup \Sigma^{m-n}$. Then M_1 and M_2 are isomorphic if and only if M_1 and M_2 are $P \cdot Z$-equivalent.*

PROOF.

(if) From Lemma A.6 it follows that Z-equivalence is an isomorphism from M_1 to M_2. By definition it follows that M_1 and M_2 are isomorphic.

(only if) Suppose that M_1 and M_2 are isomorphic. By Lemma A.7 it follows that M_1 and M_2 are also equivalent. Hence, M_1 and M_2 are $P \cdot Z$-equivalent. □

COROLLARY A.9. *Let M_1 and M_2 be Mealy machines. Let P be a transition cover set for M_1. Let W be a characterization set for M_1, and let $Z = W \cup \sum^{m-n}$. Then M_1 and M_2 are equivalent if and only if M_1 and M_2 are $P \cdot Z$-equivalent.*

PROOF.

(only if) Suppose that M_1 and M_2 are equivalent. Then by definition M_1 and M_2 are S-equivalent for any set S. Hence, M_1 and M_2 are $P \cdot Z$-equivalent.

(if) Suppose that M_1 and M_2 are $P \cdot Z$-equivalent. From Theorem A.8 it follows that M_1 and M_2 are isomorphic, which by Lemma A.7 implies that M_1 and M_2 are equivalent. □

REFERENCES

BOOCH, G. 1991. *Object Oriented Design With Applications*. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, Calif.

CHEN, M. S., CHOI, Y., AND KERSHENBAUM, A. 1990. Approaches utilizing segment overlap to minimize test sequences. In *Protocol Specification, Testing and Verification*. Elsevier Science Publishers B.V., North-Holland, Amsterdam, 85–98.

CHOW, T. S. 1978. Testing software design modeled by finite-state machines. *IEEE Trans. Softw. Eng. SE-4*, 3 (May), 178–187.

DAHBURA, A. T., SABNANI, K., AND UYAR, M. U. 1990. Formal methods for generating protocol conformance test sequences. *Proc. IEEE* (Aug.), 1317–1326.

FUJIWARA, S., BOCHMANN, G. V., KHENDEK, F., AMALOU, M., AND GHEDAMSI, A. 1991. Test selection based on finite state models. *IEEE Trans. Softw. Eng. 17*, 6 (June), 591–603.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability*. W. H. Freeman and Company, New York.

GILL, A. 1962. *Introduction to the Theory of Finite-State Machines*. McGraw-Hill, New York.

GONENC, G. 1970. A method for the design of fault detection experiments. *IEEE Trans. Comput. C-19*, 6 (June), 551–558.

HENNIE, F. C. 1968. *Finite-State Models for Logical Machines*. John Wiley & Sons, New York.

HOPCROFT, J. E. AND ULLMAN, J. D. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass.

HOWDEN, W. 1987. *Software Engineering and Technology: Functional Program Testing and Analysis*. McGraw-Hill, New York.

JOHNSON, D. S. 1974. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci. 9*, 256–278.

MOORE, E. F. 1956. Gedanken-experiments on sequential machines. In *Automata Studies*. Princeton Univ. Press, Princeton, N.J. Ann. Math. Stud. 34, 1, 129–153.

MILLER, R. E. AND PAUL, S. 1991. Generating minimal length test sequences for conformance testing of communication protocols. In *INFOCOM*. 970–979.

NAITO, S AND TSUNOYAMA, M 1981. Fault detection for sequential machines by transition tours. In *IEEE Fault Tolerant Computing Conference*. IEEE, New York.

SABNANI, K. AND DAHBURA, A. 1988. A protocol test generation procedure. *Comput. Netw. ISDN Syst. 15*, 4, 285–297

SIDHU, D. P. AND LEUNG, T K. 1989. Formal methods for protocol testing: A detailed study. *IEEE Trans. Softw. Eng. 15*, 4 (Apr.), 413–426.

VASILKEVSKII, M. P. 1973. Failure diagnosis of automata. *Kibernetika* 4, (July–Aug.), 98–108.

YANNAKAKIS, M AND LEE, D. 1988. Testing finite state machines. In *23rd STOC* (New Orleans, La., May). 476–485.

Received September 1992; revised June 1993; accepted August 1994