A new Method for Incremental Testing of Finite State Machines

Lehilton Lelis Chaves Pedrosa *
University of Campinas, Brazil
lehilton.pedrosa@students.ic.unicamp.br

Arnaldo Vieira Moura †
University of Campinas, Brazil
arnaldo@ic.unicamp.br

Abstract

The automatic generation of test cases is an important issue for conformance testing of several critical systems. We present a new method for the derivation of test suites when the specification is modeled as a combined Finite State Machine (FSM). A combined FSM is obtained conjoining previously tested submachines with newly added states. This new concept is used to describe a fault model suitable for incremental testing of new systems, or for retesting modified implementations. For this fault model, only the newly added or modified states need to be tested, thereby considerably reducing the size of the test suites. The new method is a generalization of the well-known W-method [4] and the G-method [2], but is scalable, and so it can be used to test FSMs with an arbitrarily large number of states.

1 Introduction

Test case generation for reactive and critical systems using formal methods has been widely studied [1, 2, 4, 6, 8, 11, 12, 14]. In such methods, system requirements are described by means of mathematical models and formally specified functionalities. When using formal specification models, the automatic generation of adequate test cases rises as an important problem. Methods to automate the generation of test suites must be *efficient*, in terms of test suites size, and *accurate*, in terms of fault detection [3, 11]. When test suites are applied, the notion of conformance [5] can be used, so that if an implementation passes a test suite, its behavior is said to conform to the behavior extracted from the specification.

Finite State Machines (FSMs) are the basic formalism in many methods that automate the generation of conformance test case suites. For surveys, see [1, 11, 14]. Among such methods, the W-method [4] is based on the notion of characterization sets, and provides full fault coverage for minimal, completely specified and deterministic FSMs. Several derivations have been proposed around it. In particular, the G-method [2] is a generalization of the W-method that does not depend on characterization sets.

These methods assume that the system specification is treated in a monolithic way. However, in many situations, systems are modular, with their specifications being formed by several subsystems. If one such subsystem is also modeled by a FSM, we call it a submachine. Then, the full FSM model is a combination of several submachines, with the aid of a few new states and transitions. In this article, we propose a new approach to test combined FSMs when submachine implementations are assumed correct.

Testing using the combined FSM abstraction is useful in at least two situations. If a new system is modeled as a combination of several submachines, then we can implement and test each submachine independently. Later, we can then test the combined machine using a smaller test suite. In this incremental testing approach, if an implementation does not pass a test suite, only a few states need to be retested, avoiding reapplying large test suites, as in the W-method. On the other hand, suppose that a given specification is changed, then only the corresponding part of a former implementation gets modified. If we use methods like the W-method or the G-method, we would have to test the entire system again. However, if the specification is a combined machine, only the affected submachines need to be retested.

There are related works on retesting modified implementations. But they are restricted to certain types of errors and modifications, and require that implementations maintain the same number of states

*Work supported by FAPESP grant 08/07969-9.
†Work supported by FAPESP grant 02/07473-7.

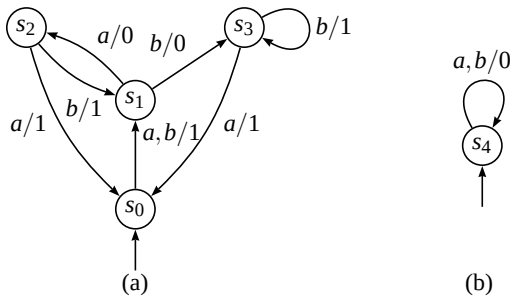

Figure 1: Finite State Machines.

as in the specification [7, 10]. In this paper, we do not restrict the types of errors in an implementation, neither how it is modified, and we allow implementations with more states than in the specification.

In Section 2, we review the FSM model and related conventions. In Section 3, we describe equivalence relations of FSMs and introduce the concept of separators, a powerful tool to test FSMs. In Section 4, we formalize the notion of combined FSMs. In Section 5, we present the new test case generation method, here named the C-method. In Section 6, we compare our method with the W-method, and discuss that the C-method is scalable, that is, it can be used to test FSMs with a large number of states.

2 Basic definitions

Let A be an alphabet. Then A^* is the set of all finite sequences of symbols, or words, over A. The length of a word $\rho \in A^*$ will be denoted by $|\rho|$, and ε will denote the empty word. So, $|\varepsilon| = 0$. The concatenation, or juxtaposition, of two words $\alpha, \beta \in A^*$ will be indicated by $\alpha\beta$.

2.1 Finite State Machines

A FSM is a tuple $M = (X, Y, S, s_0, \delta, \lambda)$, where: (i) X is a finite input alphabet, (ii) Y is a finite output alphabet, (iii) S is the set of states, (iv) $s_0 \in S$ is the initial state, (v) $\delta : X \times S \to S$ is the transition function, and (vi) $\lambda : X \times S \to Y$ is the output function.

From now on we fix the notation $M = (X, Y, S, s_0, \delta, \lambda)$ and $M' = (X, Y', S', s'_0, \delta', \lambda')$. Sequences of input symbols will be represented by words $\rho \in X^*$, and sequences of output symbols will be represented by words $\sigma \in Y^*$. The end state after the successive application of each input is given by the extended function $\widehat{\delta} : X^* \times S \to S$, and the output extended function is $\widehat{\lambda} : X^* \times S \to Y^*$, defined by

$$\widehat{\delta}(\varepsilon, s) = s, \qquad \widehat{\delta}(a\rho, s) = \widehat{\delta}(\rho, \delta(a, s)),$$
$$\widehat{\lambda}(\varepsilon, s) = \varepsilon, \qquad \widehat{\lambda}(a\rho, s) = \lambda(a, s)\widehat{\lambda}(\rho, \delta(a, s)).$$

for all $a \in X$, $\rho \in X^*$ and $s \in S$.

Usually, a FSM is represented by a state diagram. Figure 1 illustrates two FSMs with initial states s_0 and s_4, respectively. We will refer to this figure through the paper.

2.2 Concatenation of words and relative concatenation

We adopt the usual notation of concatenation of two sets of words and denote by X_n the set of all input words with length at most n. For the sake of completeness, we give a definition below.

Definition 1. *Let $A, B \subseteq X^*$ and let n be a non-negative integer. Then, (i) $AB = \{\alpha\beta | \alpha \in A, \beta \in B\}$, (ii) $X^n = \{\rho \in X^* | \, |\rho| = n\}$, and (iii) $X_n = \bigcup_{k=0}^{n} X^k$.* ∎

Suppose that a set of input words, Z, must be applied to a set of states, S. To accomplish this, we generate test cases, by selecting a set of words, Q, to reach the states in S, and concatenating Q with Z. For example, if $Z = \{a\}$, $S = \{s_1, s_2\}$ and we may reach s_1 and s_2 applying a and ab to s_0, respectively, then we may select $Q = \{a, ab\}$, and generate test cases $QZ = \{aa, aba\}$. Now, suppose that specific

sets are applied to distinct states, that is, $Z_1 = \{a\}$ is applied to s_1, and $Z_2 = \{b\}$ is applied to s_2. In this case, the conventional concatenation is not useful. To address this problem, the relative concatenation was introduced [8]. First, we need the following, where $\mathscr{P}(A)$ stands for the power set of a set A.

Definition 2. *Let M be a FSM and let Π a partition of S. A state attribution is a function $\mathcal{B} : S \to \mathscr{P}(X^*)$. A class attribution is a function $\mathcal{B} : \Pi \to \mathscr{P}(X^*)$.* ∎

A class attribution induces a state attribution a in a natural way. If $\mathcal{B}$ is a class attribution over a partition Π, then the induced state attribution, $\overline{\mathcal{B}}$, is defined by $\overline{\mathcal{B}}(s) = \mathcal{B}(C)$, for all $s \in C$ and all $C \in \Pi$.

Definition 3. *Let M be a FSM, $A \subseteq X^*$, and $\mathcal{B}$ be a state attribution of M. Given a state s, we define the s-relative concatenation of A and $\mathcal{B}$ as $A \underset{s}{\otimes} \mathcal{B} = \{\alpha\beta | \alpha \in A, \beta \in \mathcal{B}(\widehat{\delta}(\alpha, s))\}$.* ∎

Whenever $s = s_0$, we may drop the state index and simply write $A \otimes \mathcal{B}$. If $\mathcal{B}$ is a class attribution, then we may also write $A \underset{s}{\otimes} \mathcal{B}$ to mean $A \underset{s}{\otimes} \overline{\mathcal{B}}$. The usual concatenation may be thought of as a particular case of the relative concatenation, as observed below.

Observation 4. *Let M be a FSM and A, B be sets of input word. Let also $\mathcal{B}$ be a state attribution such that $\mathcal{B}(s) = B$ for all $s \in S$. Then $A \underset{s}{\otimes} \mathcal{B} = AB$.* ∎

2.3 State Reachability

Definition 5. *Let M be a FSM. A state s is reachable if and only if there exists $\rho \in X^*$ such that $s = \widehat{\delta}(\rho, s_0)$. M is connected if and only if every state is reachable.* ∎

When applying an input word ρ to a start state s, if all we know is that the length of ρ is at most n, then the output fetched when applying ρ starting at s will depend only on states that are at a distance of at most n from s. Such states around s form a *neighborhood*, defined as follows.

Definition 6. *Let M be a FSM.*

1. *The k-*radius *of a state s, denoted by* $\mathrm{rad}(s,k)$*, is the set of states that can be reached starting at s and using input words of length at most k. That is, $r \in \mathrm{rad}(s,k)$ if and only if there exist an input word $\rho \in X^*$ such that $r = \widehat{\delta}(\rho, s)$ and $|\rho| \leq k$.*
2. *The k-*neighborhood *of a set of states C, denoted by* $\mathrm{nbh}(C,k)$*, is formed by the k-radiuses of states in C. That is,* $\mathrm{nbh}(C,k) = \bigcup_{s \in C} \mathrm{rad}(s,k)$.

∎

2.4 Cover sets

Cover sets are used in many FSM test methods in order to guarantee that every state is reached from the initial state and that every transition in the model is exercised at least once. But, if we know that some states have already been tested, then we do not need to reach them or exercise their corresponding transitions. In this situation, only untested states must be covered, and partial cover sets are sufficient.

Definition 7. *Let M be a FSM and C be a set of states. A set $P \subseteq X^*$ is a partial transition cover set for C if, for every state $s \in C$ and every symbol $a \in X$, there exist $\rho, \rho a \in P$ such that $s = \widehat{\delta}(\rho, s_0)$.* ∎

Whenever C is the set of all states, P is, in fact, a transition cover set as defined in [2, 4, 8]. A transition cover set may be obtained from a labeled tree for M [4]. A procedure to construct the labeled tree is given in [2]. Although that is intended to cover the entire set of states, one can modify this procedure in a straightforward way in order to obtain a partial cover set.

3 State equivalence and state separators

In this section, we define state equivalences and introduce the essential notion of a separator.

3.1 State equivalence

Definition 8. *Let M and M' be two FSMs over the same input alphabet, X, and let s and s' be states of M and M', respectively.*

1. *Let $\rho \in X^*$. We say that s is ρ-equivalent to s' if $\widehat{\lambda}(\rho,s) = \widehat{\lambda}'(\rho,s')$. In this case, we write $s \approx_\rho s'$. Otherwise, s is ρ-distinguishable from s', and we write $s \not\approx_\rho s'$.*
2. *Let $K \subseteq X^*$. We say that s is K-equivalent to s' if s is ρ-equivalent to s', for every $\rho \in K$. In this case, we write $s \approx_K s'$. Otherwise, s is K-distinguishable from s', and we write $s \not\approx_K s'$.*
3. *State s is equivalent to s' if s is ρ-equivalent to s' for every $\rho \in X^*$. In this case, we write $s \approx s'$. Otherwise, s is distinguishable from s', and we write $s \not\approx s'$.* ∎

In Figure 1, state s_1 in (a) is bb-distinguishable from state s_4 in (b), so we write $s_1 \not\approx_{bb} s_4$.

We say that two FSMs, M and M', are equivalent, if $s_0 \approx s'_0$. So, we say that a FSM correctly implements another FSM if the initial states of the corresponding machines are equivalent.

If M and M' are the same machine, the definition above can be taken as specifying equivalence relations over sets of states $C \subseteq S$. In this case, for a set of input words $R \subseteq X^*$, the relation $\approx_R$ induces a partition of the states in C. We denote such partition by $[C/R]$. For example, in Figure 1(a), with $C = \{s_0, s_1, s_2, s_3\}$, $R = \{aaaa\}$ induces the partition $[C/R] = \{\{s_0\}, \{s_1\}, \{s_2, s_3\}\}$.

The number of pairwise distinguishable states of a FSM is called its index, as defined below.

Definition 9. *Let M be a FSM and C be a set of states. The number of equivalence classes induced by the $\approx$ relation over C is denoted by $\iota(C)$. The index of M is $\iota(S)$. If $\iota(S) = |S|$, then the machine is said to be minimal.* ∎

3.2 State separators

From Definition 8, two states s and r are distinguishable if and only if there exists a word γ with $s \not\approx_\gamma r$. Whenever this happens, we say that γ separates s and r. We extend this notion, so that we can *separate* any two sets of states. In this case, we use a collection of input sequences instead of just one sequence.

Definition 10. *Let M be a FSM, let A, B be two subsets of states, not necessarily disjoint, and let $R \subseteq X^*$ be a set of input words. R is a (A,B)-separator if and only if for each pair of distinguishable states s and r, such that $s \in A$ and $r \in B$, we have $s \not\approx_R r$.* ∎

To exemplify this new concept, consider machine (a) in Figure 1, and let $A = \{s_0, s_1\}$, $B = \{s_0, s_2\}$ and $C = \{s_0, s_3\}$. The set of input sequences $R = \{ab\}$ is a (A,B)-separator, but, since $s_2 \approx_R s_3$, and $s_2 \in B, s_3 \in C$, R is not a (B,C)-separator. Note that state s_0 is a common element of A and B.

Notice that, in this paper, we adopt a more flexible definition of characterization sets than that found in [9]. In the latter, the FSM being minimal is a necessary condition for the existence of a characterization set, while in our definition, any FSM has a characterization set. The same happens with respect to identification sets as defined in [8]. We don't even require a characterization set or an identification set to be minimal. Also, note that, in Definition 10, the sets A and B may have a nonempty intersection. This often happens in the case of characterization sets, which are used to separate any pair of distinguishable states of the machine. Actually, we impose no restriction on what sets of states we may select.

A number of special cases are worth noticing: (i) A (S,S)-separator is a *characterization set* for M. (ii) An *identification set* for a state s is any $(\{s\},S)$-separator. (iii) For a given set $C \subseteq S$, a (C,C)-separator is also called a *partial characterization set for C*. (iv) If $R \subseteq X^*$ is a (A,B)-separator such that $r \not\approx_R s$, for every pair $r \in A$, $s \in B$, then R is also called a *strict* (A,B)-separator.

In Section 5, R is a separator that exemplifies a number of situations: it will be an identification set for a state s, a partial characterization set for a set of states C, and a strict separator for sets of states A, B.

Next, we point out some useful separator properties.

Observation 11. *Consider a FSM, M. Let A,B,C and D be subsets of states, not necessarily disjoint, and let T and U be sets of input sequences. Let also r and s be states of M. Then,*

1. *T is a (A,B)-separator if and only if T is a (B,A)-separator;*
2. *If T is a (A,B)-separator and U is a (C,D)-separator, then $T \cup U$ is a $(A \cup C, B \cap D)$-separator;*
3. *If T is a strict (A,B)-separator, $r \in A$ and $r \approx_T s$, then $s \notin B$;*
4. *If T is a (A,B)-separator, $r \in A$, $s \in B$ and $r \approx_T s$, then $r \approx s$;*
5. *If T is a (A,B)-separator, $C \subseteq A$ and $D \subseteq B$, then T is a (C,D)-separator.* ∎

We can use a separator to construct another one. With the next lemmas and corollary, we obtain a partial characterization set from a weaker separator. The proofs are available in [13].

Lemma 12. *Let M be a FSM. Let $C \subseteq S$ be a set of states, let $B = \mathrm{nbh}(C,1)$ be its close neighborhood and let R be a $(B, B \backslash C)$-separator such that R partitions C into at least n classes, that is, $|[C/R]| \geq n$. If there exist two distinguishable states $r,s \in C$ such that $r \approx_R s$, then $XR \cup R$ separates C in at least $n+1$ classes, that is, $|[C/(XR \cup R)]| \geq n+1$.* ∎

Suppose that we applied the last lemma and obtained a new separator X_1R. If there exist two distinguishable states in C that are X_1R-equivalent, then we may use the lemma again to obtain a stronger separator, X_2R. In fact, the lemma may be used several times successively. We do this in the following.

Lemma 13. *Let M be a FSM. Let $C \subseteq S$ be a set of states, let $B = \mathrm{nbh}(C,1)$ be its close neighborhood and let R be a $(B, B \backslash C)$-separator such that R partitions C into at least n classes, that is, $|[C/R]| \geq n$. If m is an upper bound on the number of $\approx$-equivalence classes in C, and l is an integer such that $n \leq l \leq m$, then $X_{l-n}R$ separates C in at least l classes, that is, $|[C/X_{l-n}R]| \geq l$.* ∎

Corollary 14. *$X_{m-n}R$ is a (C,C)-separator.* ∎

This corollary can be used to obtain a partial characterization set for C. It generalizes a known result from Chow [4], demonstrated in [2], that gives us the ability to generate characterization sets. The latter result is, in fact, a particular case of Corollary 14, when $C = S$.

A separator for two sets of states A and B can be obtained by selecting a minimal subset of a characterization set that is also a (A,B)-separator. Standard methods to reduce a FSM and to obtain a characterization set for it are known [9]. Although this can be used for any FSM, we may obtain shorter separators if we take into consideration the specificities of the FSM being tested.

4 Combined Finite State Machines

Many systems are actually aggregations of other, smaller, subsystems. When modeling such systems, it is usual to adopt the *building block strategy* for the development cycle, in which each subsystem is designed, implemented and tested separately. Though each individual part of the system is tested and deemed correct, we have no guarantee that the integrated final implementation is also correct. In order to test such systems efficiently, we formalize below the concepts of combined FSMs.

Definition 15. *Let $M = (X,Y,S,s_0,\delta,\lambda)$ be a FSM. A FSM $\dot{N} = (\dot{X},\dot{Y},\dot{S},\dot{s_0},\dot{\delta},\dot{\lambda})$ is called a submachine of M if and only if $\dot{X} = X$, $\dot{Y} \subseteq Y$, $\dot{S} \subseteq S$ and, for every $a \in X$ and $s \in \dot{S}$, we have $\dot{\delta}(a,s) = \delta(a,s)$ and $\dot{\lambda}(a,s) = \lambda(a,s)$.* ∎

The definition ensures that a state of a subsystem behaves exactly in the same way, regardless of whether it is considered a state of a submachine or as new state of the combined machine. A combined FSM is formed by conjoining one or more submachines. That is, a FSM may be constructed by adding new states and new transitions to connect a set of submachines. Since each subsystem has only one entry point, every transition that enters a submachine should end in that submachine initial state. If, for specific needs, a submachine has more then one entry point, then we may consider several submachines, with the same set of states and the same transitions, but with different initial states.

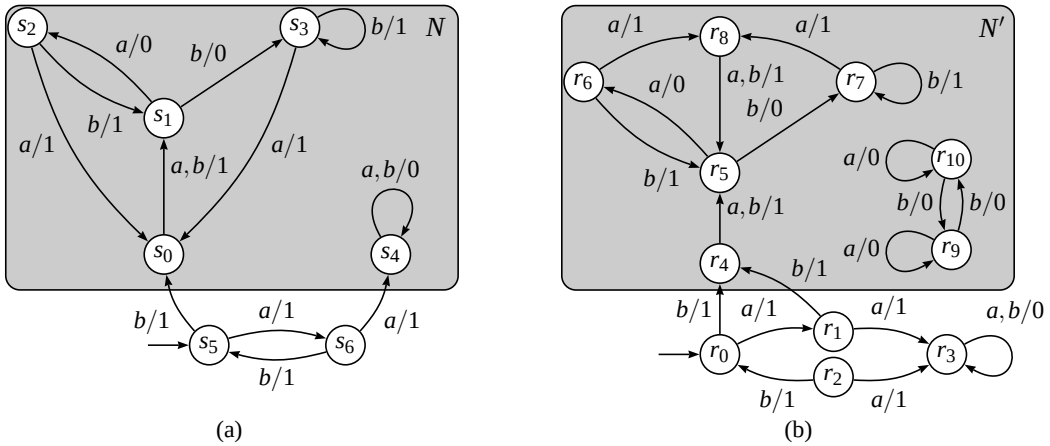

Figure 2: A Combined Finite State Machine and a candidate implementation.

Definition 16. *Let M be a FSM and N be a set of submachines of M. Define $S_N = \{s \in \dot{S} | \dot{N} \in N\}$ as the set of all submachine states, and $S_M = S \setminus S_N$ as the set of additional states. Also, define $I_N = \{\dot{s}_0 | \dot{N} \in N\}$ as the set of all submachines initial states. Then, M is N-combined if and only if $s_0 \in S_M$ and, for every pair of states s and r such that $s \in S_M$ and $r \in S_N$, if there exists $a \in X$ such that $r = \delta(a,s)$, then $r \in I_N$.* ∎

In Figure 2(a), we illustrate a combined FSM. The set of submachines, N, is formed by the machines defined in Figure 1. For this machine, we have $S_N = \{s_0, s_1, s_2, s_3, s_4\}$, $I_N = \{s_0, s_4\}$ and $S_M = \{s_5, s_6\}$. The initial state is $s_5 \in S_M$. We notice that, in fact, this machine satisfies the properties of Definition 16. For example, for states $s_5 \in S_M$ and $s_0 \in S_N$, since $s_0 = \delta(b, s_5)$, $s_0 \in I_N$.

We shall use the notation introduced in Definitions 15 and 16. So, given a machine M and a set of submachines N, we have the sets S_N, S_M, I_N and submachines $\dot{N}$ in N. Moreover, decorations carry over uniformly, *e.g.*, from a FSM M' and a set of submachines N', we have the sets S'_M, S'_N, and so forth.

5 The C-method

We present a new method, named the C-method, to test combined FSM specifications. We assume that an implementation is also a combined FSM in which each submachine is already tested and deemed correct. Also, the number of additional states is limited to a fixed upper bound. If these conditions are satisfied, then the C-method automatically yields a test case suite with full fault coverage.

A submachine can be itself a combined FSM. It can, of course, also be tested using the C-method, giving rise to a recursive testing approach. Notice that the set of submachines may be empty, so one can always use the C-method to directly test FSM specifications. In this particular case, using the C-method is equivalent to using the G-method [2]. Also, notice that it is necessary to test a submachine only once, and then implementations that use it can be tested several times at a reduced cost. Further, retesting is possible, so that, if the specification is changed, only the affected submachines need to be retested. Next, we formalize our fault model. Then, we describe the construction of the test suite.

5.1 The fault model

The system specification M is a combined FSM, obtained from a set N of submachines. We assume that M is connected, and that for every pair of states, $s \in S_M$ and $r \in S_N$, we have $s \not\approx r$. Such assumptions are reasonable, because there is no meaning in having unreachable states in the specification, or in reimplementing the behavior of an already available submachine state. We also assume that each submachine $\dot{N} \in N$ has a correct implementation $\dot{N}'$, and denote the set of submachine implementations by N'. A system implementation M' is a combination of submachines from N' with up to m new states. The goal is to test M' against M. But, first, we need to describe the fault model.

Definition 17. *Let M be a FSM specification and let N be a set of submachines of M such that M is N-combined. Let N' be a set of FSMs and m be a positive integer. A FSM candidate implementation M' is (N',m)-combined if: (i) M' is N'-combined; (ii) $\iota(S_M) \leq |S'_M| \leq m$; (iii) for every $\dot{N} \in N$, there exists $\dot{N}' \in N'$ such that $\dot{s_0} \approx \dot{s_0}'$; and (iv) for every $\dot{N}' \in N'$, there exists $\dot{N} \in N$ such that $\dot{s_0}' \approx \dot{s_0}$.* ∎

Figure 2(b) illustrates a candidate implementation for the combined machine depicted in Figure 2(a). We claim that the former obeys Definition 17 with $m = 4$. Clearly, $2 = \iota(S_M) \leq |S'_M| \leq m = 4$. Also, each state in S_N has a corresponding state in S'_N. For instance, we have $s_0 \in S_N$ and $r_4 \in S'_N$ such that $s_0 \approx r_4$. Notice that each submachine implementation need not be minimal. For example, we have $r_9 \approx r_{10}$.

5.2 Test suite generation

The C-method is now presented. We first obtain some intermediate sets of input words, namely, a partial transition cover set P, and two separators R and T. We use R to determine a parameter n, while R and T are used to define a state attribution $\mathcal{Z}$. Then, we use the relative concatenation operator to connect P and $\mathcal{Z}$, thus obtaining the final test suite. Procedure 1 summarizes all steps. We expand on each step.

THE COVER SET P: It is a partial transition cover set for S_M with $\varepsilon \in P$. This set is used to reach every additional state in the specification, so that one can exercise the corresponding transitions. Since states in S_N are known to be already correctly implemented, there is no need to cover them.

THE SEPARATOR R: We select R as any $(I_N \cup S_M, S_N)$-separator. This set assumes several different roles, depending on the states we are testing. For example, as a strict (S_M, S_N)-separator, R is used to distinguish submachine states from additional states. As a (I_N, S_N)-separator, R may be used to identify initial states of submachines, and so on.

THE PARAMETER n: The relation $\approx_R$ induces partitions on M. Based on this, we define a parameter l by letting $l = |[S/R]| - |[S_N/R]|$. Similarly, $\approx_R$ induces a partition on the states of M'. In this case, we have to *choose* a parameter l' with the proviso that $l' \leq |[S'/R]| - |[S'_N/R]|$. If no information about M' is available, we can always choose $l' = 0$. Then, we set $n = \max\{l, l'\}$. This parameter influences the size of the test suite, that is, the larger n is, the smaller the test suite will be. As suggested in the G-method [3], if knowledge is available about the implementation, then l' may be set to larger values, thus giving rise to more succinct test suites. We notice that we always have $m \geq n$, otherwise no correct candidate implementation would be possible.

THE SEPARATOR T: It is used to complement R, whenever there is a need to identify states in neighborhoods of I_N. We define $A = \text{nbh}(I_N, m-n-1)$ and select T to be any (A, S_N)-separator. Notice that in the case $m = n$, A contains no element, so we may define T as the empty set.

THE STATE ATTRIBUTION $\mathcal{Z}$: We use T only for input words that reach states in S_N. Then, to avoid generating unnecessary test sequences, we use a class attribution $\mathcal{R}$, given by $\mathcal{R}(S_N) = T \cup R$ and $\mathcal{R}(S_M) = R$. We then define a state attribution $\mathcal{Z}$ by letting $\mathcal{Z}(s) = X_{m-n} \underset{s}{\otimes} \mathcal{R}$, for all $s \in S$.

THE TEST SUITE π: The test suite generated by C-method is computed as $\pi = P \otimes \mathcal{Z}$.

The correctness of C-method is guaranteed by the following theorem.

Theorem 18. *Let M be a FSM specification and M' be a FSM candidate implementation, as described in Subsection 5.1. Obtain a test suite π using Procedure 1. Then, $s_0 \approx s'_0$ if and only if $s_0 \approx_\pi s'_0$.*

Proof sketch. The proof relies on Corollary 14. We use a weaker separator R to obtain a partial characterization set, $Z = X_{m-n}R$, for the set of additional states S'_M. We then use T to separate implementation states that are distinguishable only by sequences that reach the initial states of submachines. Once we have a partial characterization set for S'_M, we use arguments similar to those appearing in proofs involving the the G-method. We give a complete and detailed proof of the C-method correctness in [13]. ∎

Procedure 1: Test suite construction for C-method

Input: M, m **Output:** π

begin

Obtain a partial transition cover set P for S_M such that $\varepsilon \in P$;
Obtain a $(S_M \cup I_N, S_N)$-separator R ;
Define $l \longleftarrow |[S/R]| - |[S_N/R]|$;
Choose $l' \leq |[S'/R]| - |[S'_N/R]|$;
Define $n \longleftarrow \max\{l', l\}$;
Define $A \leftarrow \text{nbh}(I_N, m-n-1)$;
Obtain a (A, S_N)-separator T ;
Define $\mathcal{R}(S_M) \longleftarrow R$ and $\mathcal{R}(S_N) \longleftarrow R \cup T$;
foreach $s \in S$ **do**
  Define $\mathcal{Z}(s) \longleftarrow X_{m-n} \underset{s}{\otimes} \mathcal{R}$;
return $\pi \longleftarrow P \otimes \mathcal{Z}$;

6 Comparison and Discussion

In this section, we briefly review the W-method and generate test suites for an example specification using W-method and C-method. For this example, we limit the number of sequences generated by each method and give the number of unique and prefix-free test cases [3]. Then, we discuss the general case.

6.1 The W-method

The W-method applies to minimal, completely specified and deterministic FSMs. The set of implementation candidates comprehends all faulty machines with up to m_W states. Test suites for this fault model are called m_W-complete. The method depends on two sets, P_W and W. P_W is a transition cover set for S, and W is a characterization set for S. Then, an intermediate set Z_W is defined as $X_{m_W - n_W} W$, where n_W is the number of specification states. The final test suite is given by $\pi_W = P_W Z_W$.

6.2 Example

We will generate test suites for the specification depicted in Figure 2(a). The test suites are intended for (N', m)-combined candidate implementations, with $m = 4$ and where N' is illustrated in Figure 2(b).

USING THE W-METHOD**:** The specification in Figure 2(a) has $n_W = 7$ states and the candidate implementation has $|S'_N| = 7$ submachines states and up to $m = 4$ additional states. So the minimum value for m_W we may choose is $m_W = 7 + 4 = 11$. Next, we select a transition cover set, P_W, and then we choose a minimal characterization set, W. Finally the Z_W set is computed.

- $P_W = \{\varepsilon, a, b, aa, ab, aaa, aab, ba, bb, baa, bab, baaa, baab, baba, babb\}$;
- $W = \{aaaa, bb\}$;
- $Z_W = X_{m_W - n_W} W = X_4 W$.

So, $\pi_W = P_W Z_W$ and $|\pi_W| \leq |P_W||X_4||W| = 930$. In fact, π_W has 256 prefix-free words.

USING THE C-METHOD**:** We select P as a minimal subset of P_W that is a partial transition cover set for S_M. Then a $(S_M \cup I_N, S_N)$-separator R is extracted from W. Notice that R is a weaker separator than W, since, for example, $s_2 \approx_R s_3$, but $s_2 \not\approx_W s_3$. Next, we first partition the states of M and obtain the value l. Since no specific information is available about M' we choose $l' = 0$. From those two values we obtain the parameter n. Proceeding, we define A as the $(m-n-1)$-neighborhood of I_N, and then we select a (A, S_N)-separator T from W. Finally, we calculate the state attribution $\mathcal{Z}$:

- $P = \{\varepsilon, a, b, aa, ab, aaa, aab, ba, bb\}$;
- $R = \{aaaa\}$;
- $[S/R] = \{\{s_0\}, \{s_1\}, \{s_2, s_3\}, \{s_4\}, \{s_5\}, \{s_6\}\}$;

- $[S_N/R] = \{\{s_0\},\{s_1\},\{s_2,s_3\},\{s_4\}\}$;
- $l' = 0$, $l = |[S/R]| - |[S_N/R]| = 2$ and so $n = \max\{l', l\} = 2$;
- $A = \text{nbh}(I_N, m-n-1) = \text{nbh}(\{s_0,s_4\},1) = \{s_0,s_1,s_4\}$;
- $T = \{aaaa\}$;
- $\mathcal{R}(S_N) = T \cup R = R$ and $\mathcal{R}(S_M) = R$;
- $\mathcal{Z}(s) = X_{m-n} \underset{s}{\otimes} \mathcal{R} = X_{m-n}R = X_2R$ for every $s \in S$.

So, $\pi = P \otimes \mathcal{Z} = PX_2R$ and $|\pi| \leq |P||X_2||R| = 63$. In fact, π has 20 prefix-free test cases. Also, the submachines of Figure 2(a) may have been tested previously using 24 test cases. So, one can use the C-method to test the entire specification using only 44 words.

Comparing the results, the gains afforded by the C-method are evident.

6.3 Discussion

The difference between the two test suites obtained in the previous example is mainly due to two factors. First, in the C-method, we use a partial cover set, and so we can use a subset of the cover set used by W-method. Second, since $m - n \leq m_W - n_W$, the set X_{m-n} used by C-method may have exponentially less sequences than the set $X_{m_W-n_W}$ used by W-method. This can be seen by the following theorem. The proof is available in [13].

Theorem 19. *Let M be a minimal connected N-combined FSM. Assume that $|X| \geq 2$. Consider the fault model defined by all implementations M' such that M' is (N',m)-combined, $|S'_M| = m$ and $|S'_N| = k$. Let π_W be the test suite generated by the W-method, with P_W the set used as a transition cover set. Then, we can use the C-method and obtain a test suite π, associated with a partial transition cover set P obtained from P_W, in such a way that $\pi \subseteq \pi_W$, and satisfying*

$$\textit{(i)}\ \frac{|P_W|}{|P|} \geq 1 + \frac{|X|}{|X|+1}\frac{j}{l}, \qquad \textit{(ii)}\ |\pi| \in O(l(j+l)^2|X|^{m-l+1}), \quad \textit{and} \quad \textit{(iii)}\ |\pi_W| \in O((j+l)^3|X|^{m-l+k-j+1}),$$

where $l = |S_M|$ and $j = |S_N|$. ∎

This result allows us to compare the test suites generated by both the W-method and the C-method. Clearly, both test suites depend on the cover sets that are used. The first claim in Theorem 19, estimates the ratio between the sizes of the cover sets. It indicates that the larger is the number of submachines states, j, compared to the number of additional states, l, the greater is the advantage of using C-method. This is expected, since, when using C-method, we do not need to test submachine states. In Theorem 19, the second claim gives a bound to the size of the test suites generated by the C-method. The factor $l|X|$ corresponds to the cover set P, the factor $(j+l)^2$ is a rough limit on the size of separator R, and the factor $|X|^{m-l}$ comes from the set X_{m-l}. This set is used to allow the test of implementations with more states than the specification. Claim (iii) at Theorem 19 concerns the W-method. We have a similar bound, except that there is an extra factor of $|X|^{k-j}$, which corresponds to the difference between the number of submachine states in the implementation, k, and in specification, j. Since submachines are known to be correctly implemented, we do not need to deal with these states when using the C-method. This indicates that the C-method may generate test suites with exponentially less test cases than W-method.

We also argue that the C-method is scalable. That is, unlike the W-method, which requires that specifications have a small number of states, with the C-method we can test systems with a high number of states, provided that the number of additional states is kept low. This is due the fact that, in spite of the specification being arbitrarily large, the size of the generated test suite is only polynomial on the number of submachines states. Compare this to the bound obtained for W-method. We conclude that scalability is a major advantage of the C-method when testing systems designed with a building block strategy.

7 Conclusions

The W-method [4] is widely used to test critical systems modeled as FSMs. However, if the number of states is large, this method, and derivations from it, become impractical. Moreover, in several common situations, using the W-method is inefficient. Such cases include testing modified implementations with minor specifications changes and testing new systems modeled by the building block strategy.

To address these issues, we introduced the concept of combined FSMs, thus capturing the situation when the specification is given by a composition of other, previously tested, submachines. With this new concept, we were able to represent the above situations in a specific fault model. This resulted in the C-method, which can generate smaller test suites for combined FSMs.

We also introduced separators, generalizing the notion of characterization sets. Separators showed to be useful tools to prescribe the generation of test suites. By using separators, instead of characterization set, as in the W-method, we can distinguish only as few states as we need and, therefore, we may use smaller sets of distinguishing sequences, thereby reducing the size of the test suites beg generated.

Further, although the C-method can always obtain test suites that are subsets of those generated using W-method, our method may, in fact, generate exponentially less sequences than W-method.

Finally, we showed that C-method is scalable, provided that the number of additional states is kept small. This means that we can test FSMs with an arbitrarily large number of states if we apply a building block strategy during system development and maintenance.

References

[1] G. V. Bochmann and A. Petrenko. Protocol testing: review of methods and relevance for software testing. In *ISSTA '94: Proc. of the 1994 ACM SIGSOFT Inter. Sym. on Soft. Testing and Analysis*, pages 109–124, 1994.

[2] A. L. Bonifácio, A. V. Moura, and A. S. Simão. A generalized model-based test generation method. In *6th IEEE Inter. Conferences on Software Engineering and Formal Methods*, pages 139–148, 2008.

[3] A. L. Bonifácio, A. V. Moura, and A. S. Simão. Exponentially more succinct test suites. Technical Report IC-09-07, Institute of Computing, University of Campinas, 2009.

[4] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, 1978.

[5] E. M. Clarke and J. M. Wing. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[6] R. Dorofeeva, K. El-Fakih, and N. Yevtushenko. An improved conformance testing method. In *IFIP 25th Inter. Conference on Formal Techniques for Networked and Distributed Systems*, pages 204–218, 2005.

[7] K. El-Fakih, N. Yevtushenko, and G. V. Bochmann. Fsm-based incremental conformance testing methods. *IEEE Transactions on Software Engineering*, 30(7):425–436, 2004.

[8] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991.

[9] A. Gill. *Introduction to the theory of finite state machines.* McGraw-Hill, 1962.

[10] I. Koufareva, A. Petrenko, and N. Yevtushenko. Test generation driven by user-defined fault models. In *Proc. of the IFIP TC6 12th Inter. Workshop on Testing Communicating Systems*, pages 215–236, 1999.

[11] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. In *Proc. of the IEEE*, pages 1090–1123, 1996.

[12] G. Luo, A. Petrenko, and G. V. Bochmann. Selecting test sequences for partially-specified nondeterministic finite state machines. In *IFIP 7th Inter. Workshop on Protocol Test Systems*, pages 91–106, 1994.

[13] L. L. C. Pedrosa and A. V. Moura. Testing combined finite state machines. Technical Report IC-10-01, Institute of Computing, University of Campinas, 2010. Available in `http://www.ic.unicamp.br/~reltech/2010/abstracts.html`.

[14] D.P. Sidhu and T.-K. Leung. Formal methods for protocol testing: a detailed study. *IEEE Transactions on Software Engineering*, 15(4):413–426, 1989.