Geração Automática de Casos de Testes para Máquinas de Estados Finitos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Lehilton Lelis Chaves Pedrosa e aprovada pela Banca Examinadora.

Campinas, 13 de setembro de 2010.

Prof. Dr. Arnaldo Vieira Moura (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Pedrosa, Lehilton Lelis Chaves
P343g Geração automática de casos de testes para máquinas de estados finitos/Lehilton Lelis Chaves Pedrosa-- Campinas, [S.P. : s.n.], 2010.

Orientador : Arnaldo Vieira Moura.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Métodos formais (Ciência da computação). 2.Máquinas de estados finitos. 3.Teste baseado em modelos. 4.Software - Testes. I. Moura, Arnaldo Vieira. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Automatic test case generation for finite state machines

Palavras-chave em inglês (Keywords): 1. Formal methods (Computer science). 2. Finite state machines. 3. Model-based testing. 4. Software – Testing.

Área de concentração: Teoria da Computação

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Arnaldo Vieira Moura (IC – UNICAMP)
Prof. Dr. Guilherme Albuquerque Pinto (DCC – UFJF)
Profa. Dra. Eliane Martins (IC - UNICAMP)

Data da defesa: 01/09/2010

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 01 de setembro de 2010, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Guilherme Albuquerque Pinto
Departamento de Ciência da Computação / UFJF

Profª. Drª. Eliane Martins
IC / UNICAMP

Prof. Dr. Arnaldo Vieira Moura
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Geração Automática de Casos de Testes para Máquinas de Estados Finitos

Lehilton Lelis Chaves Pedrosa[1]

Setembro de 2010

Banca Examinadora:

- Prof. Dr. Arnaldo Vieira Moura (Orientador)
- Prof. Dr. Guilherme Albuquerque Pinto
  Departamento de Ciência da Computação, UFJF
- Profa. Dra. Eliane Martins
  Instituto de Computação, Unicamp
- Prof. Dr. Adilson Luiz Bonifácio (Suplente)
  Departamento de Computação, UEL
- Prof. Dr. Jacques Wainer (Suplente)
  Instituto de Computação, Unicamp

[1]Suporte financeiro: CNPq (processo 135014/2008-5), de agosto de 2008 a fevereiro de 2009, e FAPESP (processo 2008/07969-9), de março de 2009 a julho de 2010.

Resumo

Métodos formais são amplamente utilizados para modelar especificações e gerar casos de testes, imprescindíveis para validação de sistemas críticos. As Máquinas de Estados Finitos (MEFs) compõem um dos formalismos adotados, com várias aplicações em testes de sistemas aéreos e espaciais, além de sistemas médicos, entre vários outros. O objetivo de um método de geração automática de casos de testes é obter um conjunto de casos de testes, com o qual é possível verificar se uma dada implementação contém falhas. Um problema importante em métodos de geração de casos de teste com cobertura completa de falhas é o tamanho dos conjuntos de testes, que normalmente é exponencial no número de estados da MEF que está sendo testada.

Para minimizar esse problema, diversas abordagens são adotadas, envolvendo melhorias nos métodos existentes, restrições do modelo de falhas e o uso de novas estratégias de teste. Esta dissertação estuda métodos automáticos para geração de casos de testes com cobertura completa de falhas e propõe dois novos métodos, que permitem reduzir o tamanho dos conjuntos de testes gerados. Primeiro, combinamos ideias do método Wp e do método G, visando usufruir as vantagens de ambos e obtendo um novo método, denominado Gp. Em seguida, descrevemos um novo modelo de falhas para sistemas compostos de vários subsistemas, possivelmente com um número alto de estados. Formalizamos tais sistemas, introduzindo o conceito de MEFs combinadas, e apresentamos um novo método de testes, denominado método C. Além disso, propomos uma abordagem de testes incremental, baseada no método C, que torna possível o teste de MEFs com um número arbitrário de estados. Estabelecemos comparações com abordagens tradicionais e mostramos que o uso da estratégia incremental pode gerar conjuntos de testes exponencialmente mais eficientes.

Abstract

Formal methods are widely used to model specifications and to select test cases to validate critical systems. Finite State Machines (FSMs) are used as the basic formalism in several applications, such as those used in aerial, spacial, and medical systems. The goal of a test case generation method is to obtain a set of test cases, which can be used to check whether a given implementation is valid. For test methods with complete fault coverage, the size of the generated test suites is an important matter. This is a problematic issue, since the size of test suites is usually exponential on the number of states of the FSM under test.

To minimize this problem, several different approaches were devised, such as the improvement of existing test methods, the adoption of restricted fault models, and the use of new testing strategies. Here, we study existing test case generation methods with complete fault coverage, and we introduce two new methods, that can reduce the size of the generated test suites. First, we combine ideas of the Wp-method and of the G-method. We combine advantages of both, and obtain a new method, named the Gp-method. Then, we describe a new fault model for systems composed of several subsystems, which may possibly contain a large number of states. We formalize such systems using the new concept of combined FSMs, and proceed to present a new test method, called the C-method. Starting with the C-method, we propose a new approach for incremental testing, that allows the test of FSMs with an arbitrary number of states. We compare this new strategy with traditional approaches, and we show that the use of the incremental testing strategy may generate test suites exponentially more efficient than those resulting from traditional approaches.

Agradecimentos

Agradeço a meus pais e a minha família, que, mesmo na distância, demonstraram seu apoio.

Ao meu orientador, Prof. Arnaldo, pela oportunidade e pela orientação de confiança, com que me guiou neste trabalho.

Aos meus amigos, Aline, André, Anna, Robson, Sheila, Thiago e Thiago.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Com o avanço da tecnologia, os sistemas tornaram-se cada vez maiores e mais complexos, tais como programas dos mais diversos, como analisadores sintáticos, reconhecedores de padrão, circuitos e protocolos de comunicação. Consequentemente, eles ficaram mais suscetíveis a erros. O quadro se agrava quando é considerada também a necessidade de entrega de produtos competitivos em prazos curtos e com alta qualidade. Tudo isso, aliado ao fato de que, mesmo pequenas, falhas podem levar a prejuízos incalculáveis, torna indispensável incluir a verificação como parte do projeto de qualquer sistema crítico, a fim de se garantir sua confiabilidade [17].

Diversos cuidados podem ser tomados para assegurar que uma implementação satisfaça os requisitos do sistema, ou seja, esteja de acordo com a sua especificação. No entanto, durante o desenvolvimento, é na fase de teste que se valida o correto funcionamento de implementações. Uma forma de se realizar testes é modelar o sistema, geralmente descrito de maneira informal, na forma de uma especificação rigorosa. Assumindo-se certas hipóteses, pode-se, então, certificar que uma dada implementação corresponda exatamente à sua especificação. De fato, métodos formais são utilizados para testar os mais diversos sistemas, que têm aplicações em banco de dados, sistemas aéreo-espaciais e sistemas médicos, entre vários outros [6].

Dentre os vários formalismos possíveis, as *Máquinas de Estados Finitos* (MEFs) podem modelar uma enorme gama de sistemas. MEFs são especialmente adequadas aos sistemas que têm natureza reativa, isto é, que apresentam uma saída para cada ação de entrada realizada no sistema. Um exemplo clássico da aplicação de MEFs são os protocolos de comunicação, largamente modelados e testados com métodos baseados em MEFs [1]. Nesse caso, as saídas são as mensagens de resposta que dependem do estado atual da conexão. Outros formalismos, como as Máquinas de Estados Finitos Estendidas (MEFEs), também podem modelar sistemas reativos. No entanto, uma vez que a geração de casos de testes para MEFEs é um problema em aberto [8], este trabalho se restringe a MEFs.

O objetivo de um método de teste baseado em MEFs consiste em obter um conjunto de casos de testes para o sistema especificado. Um caso de teste é composto por uma entrada de teste e pela correspondente saída esperada, segundo a especificação [20]. No contexto das MEFs, a estratégia de teste utilizada é a de caixa-preta (*black-box testing*), isto é, dada uma sequência de ações de entrada, assume-se que é possível observar a saída da implementação, denominada caixa-preta, e compará-la com a saída esperada pela especificação para aquela sequência de teste [26]. Essa estratégia é particularmente útil em casos em que há necessidade de testar uma implementação para a qual não se tem acesso a seu funcionamento interno, ou é impraticável modelar seu comportamento. De forma geral, os passos de um teste caixa-preta são:

1. construção de modelos formais adequados para a especificação;
2. derivação de entradas de teste;
3. cálculo das saídas de teste a partir da especificação;
4. execução das entradas de teste sobre a implementação;
5. comparação entre os resultados das saídas calculadas e das execuções da implementação;
6. decisão se a implementação está correta.

Para uma MEF, um caso de teste contém uma sequência de ações de entrada, normalmente representada por uma sequência de símbolos de um determinado alfabeto. Questões relevantes aos métodos de geração de casos de teste dizem respeito a quais e quantos erros tais sequências podem identificar, qual seu tamanho, quais algoritmos são capazes de gerá-las e como gerar sequências de testes menores [17]. Tais questões estão inter-relacionadas, pois, de fato, quanto maior o número de erros que um conjunto de casos de testes pode detectar, maior o seu tamanho e, portanto, o custo de testar uma implementação.

Deve-se ressaltar que um problema recorrente entre os métodos de geração de casos de teste é o tamanho das sequências de teste que, como se pode observar, cresce rapidamente à medida em que as especificações ficam maiores, possivelmente inviabilizando o procedimento de teste. Além disso, a busca por entradas que permitam a identificação de todos os erros de um sistema certamente aumenta a complexidade dos algoritmos de geração de testes. Note, por exemplo, que a busca por sequências de entrada e saída únicas (UIO), que permitem identificar todos os erros de uma implementação, é um problema PSPACE-completo [16].

Vários métodos de geração de testes já foram propostos. Entre eles, estão métodos que detectam apenas parte dos possíveis erros de uma implementação, como o método

T, o método U e o método D [26], e os métodos que possuem cobertura completa de falhas, conjunto no qual se inclui o método W [5] e derivações, como o método Wp [12], o método HSI [19], o método G [3], entre outros. Embora os chamados métodos completos possuam a vantagem de detectar qualquer falha possível em uma determinada implementação, eles possuem a desvantagem de gerarem conjuntos de testes muito maiores que os outro métodos menos ambiciosos. Portanto, é fundamental encontrar métodos que gerem conjuntos de testes completos e compactos.

Essa dissertação propõe novos métodos para a geração de conjuntos completos de testes para MEFs. O objetivo é fornecer métodos que, para cada especificação, gerem sequências de testes reduzidas. Para alcançar tal objetivo, nós combinamos ideias de métodos existentes, consideramos conjuntos de especificações mais restritas e propomos uma nova abordagem para o teste de MEFs.

Organização do texto No Capítulo 2, revisamos o formalismo de MEFs e apresentamos alguns conceitos relacionados a testes de MEFs. No Capítulo 3, descrevemos brevemente alguns trabalhos relacionados que contém métodos de geração de casos de testes conhecidos. No Capítulo 4, é exposto um artigo apresentado na *4th IEEE International Conference on Secure Software Integration and Reliability Improvement*. O artigo apresenta um novo método para geração de casos de testes, que combina ideias de dois métodos existentes. No Capítulo 5, é exposto um artigo apresentado no *2nd NASA Formal Methods Symposium*. O artigo apresenta um novo método para geração de casos de testes, que torna possível testar MEFs de forma modular, mesmo que estas contenham um grande número de estados. No Capítulo 6, é exposto um artigo submetido ao periódico *IEEE Transactions on Software Engineering*. Nesse caso, usamos o método criado anteriormente e propomos uma nova abordagem para teste de MEFs, tornando o procedimento de teste incremental e escalável no número de estados. No Capítulo 7, concluímos a dissertação e apresentamos algumas sugestões para trabalhos futuros.

Capítulo 2

Máquinas de Estados Finitos

Diversos sistemas podem ser descritos em termos da ação de saída que é observada para cada ação de entrada executada. Essa característica justifica que os modelos formais adotados para a especificação sejam normalmente baseados em MEFs, também chamadas de máquinas de Mealy. Esses modelos formais são utilizados na prática, sendo especialmente relevantes em protocolos de comunicação, amplamente modelados por MEFs [1]. De fato, máquinas de estados são utilizadas na formalização de modelos usados por diversos métodos geradores de sequências de teste. Nesses métodos, um caso de teste consiste em uma palavra do alfabeto de entrada da máquina, que representa uma sequência de ações a serem executadas, enquanto o comportamento observado será uma sequência do alfabeto de saída. A confiabilidade do teste pode ser confirmada se forem providas algumas hipóteses razoáveis, como a de que o alfabeto da máquina é o mesmo que aquele da implementação, entre outras [1, 5, 17]. Nas seções seguintes, MEFs são apresentadas formalmente.

2.1 Máquina de Estados Finitos

Formalmente, uma Máquina de Estados Finitos (MEF) M é uma 6-tupla

$$M = (X, Y, S, s_0, \delta, \lambda),$$

onde:

- X é um alfabeto finito de entrada,
- Y é um alfabeto finito de saída,
- S é um conjunto finito de estados,

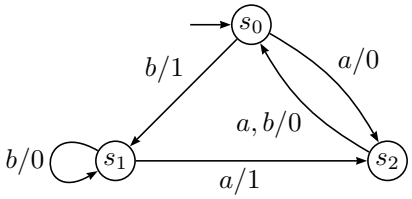

Figura 2.1: Exemplo de máquina de estados finitos.

- $s_0 \in S$ é o estado inicial,
- $\delta : X \times S \to S$ é a função de transição de estados e
- $\lambda : X \times S \to Y$ é a função de saída.

Doravante, uma MEF será sempre denotada por essa tupla. Eventuais decorações, tais como índices ou super-índices são aplicados uniformemente aos componentes da tupla. Assim, por exemplo, M' descreverá a MEF $(X', Y', S', s'_0, \delta', \lambda')$.

Sejam $s, r \in S$, $a \in X$ e $b \in Y$. Se $\delta(a, s) = r$ e $\lambda(a, s) = b$, então dizemos que, no estado s, ao receber a entrada a, a máquina vai para o estado r, com saída b. Denotamos esse fato por $s \xrightarrow{a/b} r$.

Dado um alfabeto A, denotamos por A^* o conjunto de todas as palavras finitas formadas por concatenação de símbolos de A. Representamos várias ações de entrada sucessivas como uma palavra $\rho \in X^*$ e estendemos a função de transição para $\hat{\delta} : X^* \times S \to S$, definida recursivamente por:

$\hat{\delta}(\varepsilon, s) = s$,
$\hat{\delta}(a\rho, s) = \hat{\delta}(\rho, \delta(a, s))$

onde ε representa a palavra vazia, $a \in X$, $\rho \in X^*$ e $s \in S$. Analogamente, estendemos a função de saída para $\hat{\lambda} : X^* \times S \to Y^*$, na forma:

$\hat{\lambda}(\varepsilon, s) = \varepsilon$,
$\hat{\lambda}(a\rho, s) = \lambda(a, s)\hat{\lambda}(\rho, \delta(a, s))$, onde $a \in X$, $\rho \in X^*$ e $s \in S$.

Assim, se uma máquina está no estado s e recebe uma entrada $\rho \in X^*$, ela irá parar no estado $s' = \hat{\delta}(\rho, s)$, também denotado por $s \xrightarrow{\rho} s'$, e fornecerá como saída $\hat{\lambda}(\rho, s) \in Y^*$. Para evitar sobrecarga de notação, podemos escrever δ no lugar de $\hat{\delta}$ e λ no lugar de $\hat{\lambda}$, sempre que não houver margem para confusão.

A MEF definida acima é *determinística* no sentido de que, para cada estado $s \in S$, a saída $b \in Y$ e o próximo estado $r \in S$ estão bem definidos para cada símbolo de entrada $a \in X$. Além disso, ela é *completa*, o que significa que sempre existem o próximo estado $r \in S$ e a saída $b \in Y$ para cada estado $s \in S$ e cada ação $a \in X$.

Normalmente, uma MEF M é representada por um diagrama de transição de estados, onde cada nó representa um estado e, para cada transição $s \xrightarrow{a/c} r$, com $s, r \in S$, $a \in X$ e $c \in Y$, existe um arco direcionado do nó correspondente a s ao nó correspondente a r rotulado com a/c. A Figura 2.1 ilustra uma MEF com alfabetos de entrada $X = \{a, b\}$, alfabeto de saída $Y = \{0, 1\}$ e estado inicial s_0.

Dizemos que um estado s é *atingível* se, existe $\rho \in X^*$ tal que $\delta(\rho, s_0) = s$. Ainda, dizemos que uma MEF é conexa, ou inicialmente conexa, quando cada $s \in S$ for atingível.

Equivalência de estados Em testes usando MEFs, é necessário adotar alguma noção de conformidade para determinar se uma certa implementação corresponde à sua especificação. Para isso, é fundamental adotar uma relação de equivalência de estados. Definimos dois estados como equivalentes se eles responderem com a mesma saída ao se aplicar uma sequência de entrada a cada um deles. Formalizamos esse conceito abaixo.

Sejam M e M' duas MEFs, com $X = X'$, $\rho \in X^*$, $K \subseteq X^*$, $s \in S$ e $s' \in S'$, então:

- Se M e M' respondem com a mesma saída para uma sequência de entrada ρ, isto é, se $\lambda(\rho, s) = \lambda'(\rho, s')$, então dizemos que os estados s e s' são ρ-*equivalentes* e escrevemos $s \approx_\rho s'$. Do contrário, são ρ-*distinguíveis* e escrevemos $s \not\approx_\rho s'$.

- Se para toda entrada $\rho \in K$, s e s' são ρ-equivalentes, ou seja, se para todo $\rho \in K$, tivermos $s \approx_\rho s'$, então diz-se que s e s' são K-*equivalentes*, com notação $s \approx_K s'$. Senão, s e s' são K-*distinguíveis*, denotado por $s \not\approx_K s'$.

- Se $s \approx_{X^*} s'$, dizemos simplesmente que s e s' são *equivalentes*, denotado por $s \approx s'$. Do contrário, s e s' são *distinguíveis* e escrevemos $s \not\approx s'$.

Intuitivamente, duas MEFs, M e M', são *equivalentes* se responderem com a mesma saída para cada sequência de entrada a elas fornecida. Assim, M é equivalente a M', se ambas possuírem os mesmos alfabetos de entrada e seus estados iniciais forem equivalentes, ou seja, $X = X'$ e $s_0 \approx s'_0$ e escrevemos $M \equiv M'$.

Se dois estados da mesma MEF M são equivalentes, então existe uma outra MEF, M', equivalente a M, mas com menos estados. Dizemos que M é *mínima* se, para todo par de estados distintos s e $r \in S$, tivermos $s \not\approx r$. De forma geral, uma relação de K-equivalência define classes de equivalências em um certo conjunto de estados C. Denotaremos a partição de C, composta por todas a classes de K-equivalência, por $[C/K]$.

Conjuntos de cobertura Definimos *um conjunto de cobertura de estados*, $Q \subseteq X^*$, como um conjunto de sequências de entrada que levam a máquina do estado inicial a qualquer dos outros estados. Ou seja, se para todo $s \in S$, existe $\rho \in Q$, tal que $\delta(\rho, s_0) = s$. Ainda, $P \subset X^*$ é um *conjunto de cobertura de transições* se para cada par $s \in S$ e $a \in X$, existe $\rho \in P$, tal que $\delta(\rho, s_0) = s$ e $\rho a \in P$.

Conjuntos característicos Um *conjunto característico*, $W \subset X^*$, de uma MEF M consiste de sequências de entrada que distinguem cada par de estados não equivalentes de M, isto é, para todos $s, r \in S$, com $s \not\approx r$, existe $\rho \in W$, tal que $s \not\approx_\rho r$.

2.2 Máquina de Estados Finitos Não-determinística Parcial

Alguns sistemas não podem ser descritos por MEFs tradicionais, pois suas especificações não são determinísticas e completas [19]. Para tais sistemas, o formalismo das máquinas de estados é estendido para permitir transições não-determinísticas e restrições sobre as ações de entrada permitidas em cada estado.

Uma máquina de estados finitos não-determinística parcial (MEFNP) M é uma tupla

$$M = (S, X, Y, T, s_0),$$

onde:

- S é um conjunto finito de estados,
- X é um conjunto finito de ações de entrada,
- Y é um conjunto finito de ações de saída,
- $T \subseteq X \times S \times Y \times S$ é o conjunto de transições e
- $s_0 \in S$ é o estado inicial da máquina.

Denotamos uma transição $(a, s, b, r) \in T$ por $s \stackrel{a/b}{\rightarrow} r$. Sejam $x_1, ..., x_k \in X$, $k \geq 0$, $\rho = x_1...x_k \in X^*$ e $s \in S$. Se existem $r_1, ..., r_{k+1} \in S$, $y_1, ..., y_k \in Y$, tais que $r_1 = s$ e $r_j \stackrel{x_j/y_j}{\rightarrow} r_{j+1}$ para cada $j = 1, ..., k$, então dizemos que ρ é uma *sequência de entrada definida* do estado s e $\gamma = x_1 y_1 ... x_k y_k$ é um rastro de s. O conjunto de todos os rastros de s é denotado por $Tr(s)$ e $Tr_X(s)$ é o conjunto de todas sequências de entrada definidas do estado s.

Seja Σ um alfabeto e sejam $z_1, ..., z_k \in \Sigma$, $\gamma = z_1...z_k \in \Sigma^*$, $k \geq 0$. Definimos a projeção de γ sobre $\Delta \subset \Sigma$ como $\gamma_\Delta = z'_1...z'_k$, onde $z'_i = z_i$ se $z_i \in \Delta$ e, caso contrário, $z'_i = \varepsilon$. A projeção de um conjunto A sobre Δ é $A_\Delta = \{\gamma_\Delta : \gamma \in A\}$.

Uma MEFNP *Observável* (MEFNPO) é uma MEFNP tal que, para cada estado s e par de ações de entrada e de saída a e b, há somente uma transição, isto é, se $s \xrightarrow{a/b} s'$ e $s \xrightarrow{a/b} s''$, então $s' = s''$. Uma é MEFNP é *reduzida* se cada par de estados possui rastros diferentes, isto é, para $s_i, s_j \in S$, se $s_i \neq s_j$, então $Tr(s_i) \neq Tr(s_j)$.

Relações de conformidade Dadas duas MEFNPs, M e M', com $X = X'$, dizemos que dois estados $s \in S$ e $s' \in S'$ são *distinguíveis*, escrito como $s \not\approx s'$, se existe $\gamma \in Tr(s) \oplus Tr(s')$, tal que $\gamma_X \in Tr_X(s) \cap Tr_X(s')$, onde $A \oplus B = (A \cup B) \setminus (A \cap B)$ é a união disjunta. Dito de outra maneira, dois estados são distinguíveis se existirem sequências de entradas definidas para ambos, mas que produzam rastros diferentes. Do contrário, dizemos que os estados são *indistinguíveis*, $s \approx s'$.

Dois estados, s e s', são *equivalentes*, escrito como $s \equiv s'$, se possuem os mesmo rastros, isto é, $Tr(s) = Tr(s')$. Duas MEFNPs M e M' são equivalentes, $M \equiv M'$, se seus estados iniciais o forem, *i.e*, $s_0 \equiv s'_0$. De forma menos restritiva, dizemos que estados s e s' são *quase-equivalentes*, denotado como $s \leq s'$, se o conjunto de rastros de s' inclui o de s e, para todo rastro γ de s' que contém uma sequência de entrada definida de s, γ também é rastro de s, isto é, $Tr(s) \subset Tr(s')$ e, para todo $\gamma \in Tr(s')$ com $\gamma_X \in Tr_X(s)$, temos $\gamma \in Tr(s)$.

2.3 Teste de Máquinas de Estados Finitos

Dada a especificação de uma máquina de estados, M, o problema do *teste de conformidade*, ou *detecção de falhas*, pode ser entendido como determinar se uma implementação dessa máquina, M', é equivalente a M, observando-se somente o comportamento de entrada e saída da implementação [17]. O problema consiste em obter sequências adequadas de ações de entrada e comparar as saídas da implementação com as da especificação quando tais sequências são dadas como entradas. Por esse motivo, o problema também é chamado de *geração de casos de teste*.

Conformidade Dizemos que uma implementação está *conforme* à sua especificação se não houver erros na implementação, isto é, se o conjunto de testes não detectou falhas na implementação. Se, para todas as sequências de entrada possíveis, as sequências de saída da implementação e especificação são as mesmas, então a relação de conformidade é dita *forte*. Se as sequências de saída da implementação e especificação forem as mesmas

somente para sequências de entrada definidas na especificação, então a relação é dita *fraca* [26].

Dependendo do modelo de especificação adotado, pode-se escolher métodos que geram conjuntos de testes com relações de conformidade distintas. Para máquinas completamente especificadas, podemos escolher relações de conformidade fortes, diga-se, a relação de equivalência para MEFs. Se a especificação for parcial, só podemos tratar relações fracas, como a quase-equivalência, definida para MEFNPs.

Devemos ressaltar que, em geral, o objetivo não é garantir a completa equivalência de uma implementação com sua correspondente especificação. Com efeito, na maioria das aplicações reais, é inviável estabelecer a equivalência usando métodos de geração de teste [2]. O objetivo de um teste de conformidade é verificar se uma dada implementação está conforme um modelo de especificação, o que significa que o comportamento desejado do sistema é observado quando exposto a estímulos externos [5].

Modelo de falhas O *modelo de falhas* é o conjunto de todas as implementações falhas cujos erros pretende-se identificar, mas que satisfaçam a determinadas *hipóteses* [1].

Em máquinas de estados finitos, os erros das implementações podem ser de várias naturezas [5]:

- *Erros de operação*, quando uma transição da implementação fornece saída diferente da transição correspondente na especificação.
- *Erros de transferência*, quando o estado final de uma transição da implementação não é equivalente ao estado final da transição correspondente na especificação.
- *Estados faltando ou sobrando*, quando não há um estado na implementação equivalente a algum estado da especificação e vice-versa.

Entre as hipóteses assumidas sobre as implementações estão:

- *Presença de* reset*, $\boldsymbol{r}$, confiável*: para cada estado da implementação, há uma transição para o estado inicial que aceita $\boldsymbol{r}$ como ação de entrada, não produz nenhuma saída e, além disso, $\boldsymbol{r}$ não está presente em qualquer outra transição.
- *Número limitado de estados*: assume-se que a implementação não possui mais estados do que certo número, m, estimado.
- *Rastros harmonizados*: em máquinas não-determinísticas, como as sequências de entrada são calculadas a priori, exige-se que, se dois estados distintos são alcançáveis por uma mesma sequência de ações de entrada, então eles possuem o mesmo conjunto de sequências de entradas definidas [7].

- *Teste completo*: em máquinas não-determinísticas, assume-se que todas as saídas possíveis de uma implementação para uma sequência de teste podem ser observadas aplicando-se essa sequência repetidas vezes.

Quando um conjunto de casos de teste puder identificar qualquer tipo de erro em uma implementação, dizemos que ele possui cobertura de falhas completa. Por exemplo, conjuntos de testes aplicáveis a qualquer implementação com um número de estados limitado por m são chamados conjuntos de testes m-completos.

Sequências de teste Uma *sequência de teste* ou *sequência de entrada* é uma sequência de símbolos do alfabeto de entrada que pode ser utilizada para testar uma implementação M' contra sua especificação M. Quando as ações de saída observadas pela execução de uma sequência de teste, ρ, na implementação e na especificação divergirem, então dizemos que ρ é uma *testemunha* e que *detectou* uma implementação falha. Quando sequências de teste estão acompanhadas pelas saídas esperadas, nos referimos a *casos de teste*.

Um *conjunto de testes* (*test suite*), π, é um conjunto de sequências de teste com o qual se procura verificar se uma especificação e uma implementação são equivalentes. Se, para cada implementação, M', do modelo de falha, existe uma sequência de teste $\rho \in \pi$ que detecta a falha em M', então π é chamado *completo*. Um conjunto de teste π utilizado para um modelo de falhas que contém todas as implementações incorretas com até m estados é denominado *m-completo*.

Eficiência de conjuntos de testes Como o custo de testar um sistema está diretamente relacionado com os conjuntos de testes utilizados, é importante obter um meio de medir a eficiência de um conjuntos de testes. Fatores que influenciam a eficiência de um conjunto de teste são o número de sequências de teste, da qual depende o número de *resets* necessários para testar um sistema, e o tamanho das sequências de teste, do qual depende o tempo de se testar cada sistema.

Muitas vezes, a eficiência de um conjunto de testes π depende principalmente do tamanho de seus casos de teste [26], e, portanto a sua eficiência é calculada como a soma dos tamanhos dos elementos em π [2]. Ainda, não é necessário aplicar nenhum prefixo próprio de alguma sequência de π, já que o teste sobre uma sequência maior revelará o comportamento sobre seus prefixos menores [2].

Denotamos por $\mathrm{pff}(\pi)$ o conjunto de todos os elementos de π que não são prefixos de algum outro elemento de π e definimos a eficiência de um conjunto de testes π como

$$||\pi|| = \sum_{\sigma \in \mathrm{pff}(\pi)} |\sigma|.$$

Capítulo 3

Trabalhos Correlacionados

Diversos métodos para geração de casos de teste foram propostos para máquinas de estados, com diferentes abordagens. Entre esses, está o percorrimento aleatório do diagrama de transição até passar por todas as transições, a busca por sequências únicas de entrada e saída para cada estado e a procura por sequências de entrada de uma máquina que produzam resultados diferentes para cada estado [26].

Neste trabalho, estudamos métodos completos. A seguir, descrevemos sucintamente o Método W e algumas de suas derivações.

3.1 Método W

Proposto por Chow em 1978 [5], esse método é restrito a MEFs (completamente especificadas e determinísticas) mínimas e conexas. Dadas duas MEFs, M e M', com alfabeto de entrada X, sendo M mínima e conexa, o método executa três passos: estimativa do número máximo, m, de estados em uma implementação correta, M', geração de casos testes baseada na especificação M e comparação das saídas fornecidas por M e M'. O conjunto de testes é dado pela concatenação de conjuntos de sequências PZ, em que P é um conjunto de cobertura de transições e Z constrói-se como

$$Z = X_{m-n}W \ , \quad \text{com } X_{m-n} = \bigcup_{k=0}^{m-n} X^k \ ,$$

onde n é o número de estados de M e W é um conjunto característico de M. $\pi = PZ$ é um conjunto de testes m-completo.

Chow propõe, para a construção do conjunto de cobertura P, a utilização de caminhos parciais da *árvore de testes*, que consiste, simplificadamente, em uma árvore cujos nós são estados e cada ramo é uma transição definida. A construção do conjunto W pode ser encontrada em [13].

3.2 Método Wp

O método Wp (*partial W*) [12] é baseado no método W. O conjunto de testes do método W tende a crescer à medida em que o número de estados aumenta. Em especial, um conjunto característico, W, é utilizado para testar a equivalência de cada estado. Para minimizar esse problema, o método Wp substitui o conjunto W do método anterior por subconjuntos característicos, W_i, no teste de equivalência para cada estado.

Seja M uma especificação, mínima e conexa, com n estados e M', conexa, uma implementação com no máximo m estados. Dado um estado s_i, define-se W_i um *conjunto de identificação* para s_i como um conjunto minimal tal que, para cada estado s_j, $s_j \neq s_i$, existe $\rho \in W_i$ com $\lambda(\rho, s_i) \neq \lambda(\rho, s_j)$. Neste método, um conjunto característico W é construído pela união de conjuntos de identificação, isto é, $W = \cup_{s_i \in S} W_i$.

O método consiste em duas fases: (i) verificar se podemos associar a cada estado em M um estado correspondente em M' e (ii) verificar a equivalência de cada par de estados correspondentes. Na primeira fase, a implementação é testada com um conjunto de testes definido por

$$\pi_1 = QZ,$$

onde $Z = X_{m-n}W$ e Q é um conjunto de cobertura de estados. Como a implementação M' tem no máximo m estados, temos que QX_{m-n} alcança qualquer estado e identificamos cada estado pelo conjunto característico correspondente, nos fornecendo $s_0 \approx_{QZ} s'_0$.

Na segunda fase, definimos $R = P \backslash Q$, que é o conjunto das sequências restantes de um conjunto de cobertura de transições, P. Nessa fase, o método Wp difere do W, pois cada $\rho \in R$ é concatenado apenas com o conjunto de identificação, W_i, do estado alcançável, $s_i = \delta(\rho, s_0)$. Isso é possível porque, na fase anterior, verificou-se que cada estado da implementação é W-equivalente ao estado correspondente da especificação. Assim, o conjunto de testes da segunda fase é

$$\pi_2 = \bigcup_{\rho \in R} \left(\rho \bigcup_{\sigma \in X_{m-n}} \sigma W_j \right)$$

onde W_j é o conjunto de identificação do estado $s_j = \delta(\sigma, s_i)$ e $s_i = \delta(\rho, s_0)$. A união $\pi = \pi_1 \cup \pi_2$ forma um conjunto de testes m-completo.

3.3 Método HSI

O método HSI [19], ao contrário dos métodos W e Wp, não é restrito a máquinas completamente definidas ou determinísticas, aplicando-se, portanto, a MEFNPs. Restringe-se a

máquinas observáveis, mas, também, fornece um método para converter uma MEFNP em uma MEFNPO, com comportamento equivalente ao da primeira. Baseia-se no conceito de quase-equivalência e produz como resultado um conjunto de testes m-completo. É capaz de lidar com máquinas não-mínimas, multiplicando o número de estados máximo, m, de uma implementação, M', por um fator proporcional ao número de estados redundantes da especificação, M.

Assim como o método Wp, constrói conjuntos de identificação para cada estado de M. Dado um conjunto característico, W, de M, denomina-se *conjunto de identificação de estados harmonizados* (*Harmonized state identification set - HSI*), D_i, do estado s_i, o conjunto minimal tal que

(i) $D_i \subset Tr_X(s_i) \cap \text{pref}(W)$ e,

(ii) para cada $s_i \not\approx s_j$, existe $\gamma \in Tr(s_i) \oplus Tr(s_j)$ tal que $\gamma_X \in \text{pref}(D_i) \cap \text{pref}(D_j)$,

onde W é um conjunto característico de M e $\text{pref}(W) = \{\alpha | \alpha\beta \in W \text{ e } \alpha \neq \varepsilon\}$ é o conjunto de prefixos de W. O item (i) mostra que D_i não é necessariamente todo o conjunto característico, enquanto o item (ii) garante que existe um rastro que distingua os estados em cada par de conjuntos de identificação.

Dados Q, um conjunto de cobertura de estados e d, o grau de redundância de estados, o método encontra o conjunto de identificação D_i, para cada estado $s_i \in S$, e fornece como resultado o conjunto de testes

$$\pi = \bigcup_{\substack{s_0 \xrightarrow{\rho} s_i \\ \rho \in QX_{dm-n+1}}} \rho D_i.$$

3.4 Método H

O método H [7], que é similar ao método HSI, é baseado na verificação de transições e também gera testes para MEFNPs. Diferentemente do método HSI, nesse método, o conjunto de testes é construído sequência por sequência, evitando-se incluir sequências desnecessariamente. Nesse sentido, o método H é uma melhoria do método HSI e a diminuição do tamanho das sequências pode chegar até a metade, para implementações com $m > n$.

O método define condições suficientes para que um conjunto de testes seja m-completo. Iniciando-se com um conjunto de cobertura de estados, outras sequências de entrada são incluídas, uma a uma, até formar um conjunto de testes que satisfaça tais condições. Por esse motivo, não há fórmula fechada para o conjunto de testes gerado.

Dado um conjunto de cobertura de estados, Q, um conjunto de testes é inicializado como $\pi = QX_{m-n+1} \cap Tr_X(s_0)$ e para cada par de sequências $\alpha_i\beta_i$ e $\alpha_j\beta_j \in QX_{m-n+1}$,

com $\alpha_i, \alpha_j \in Q$, e cada par de estados s_i e s_j, com $s_0 \overset{\alpha_i\beta_i}{\rightarrow} s_i$, $s_0 \overset{\alpha_j\beta_j}{\rightarrow} s_j$, então, se $s_i \not\approx s_j$, o método verifica se há sequências $\alpha_i\beta_i\sigma$ e $\alpha_j\beta_j\sigma \in \pi$ tais que $s_i \not\approx_\sigma s_j$ para algum $\sigma \in X^*$ e, se não houver, gera-as, agregando-as ao conjunto de testes.

3.5 Método G

Os métodos W, Wp, HSI e H têm em comum a necessidade de encontrar conjuntos característicos para a geração de seus testes, ou, pelo menos, conjuntos de identificação de estados, que são subconjuntos característicos. O método G estende o método W gerando sequências de teste para máquinas de estados finitos determinísticas sem a necessidade do cálculo de um conjunto característico W [3].

Ao invés de utilizar um conjunto característico, o método G requer um conjunto de sequências de entrada, R, qualquer. Esse conjunto pode ser escolhido de forma a maximizar o número de classes de equivalência entre estados de uma implementação, M', induzida por R. Similarmente ao método W, deve-se estimar um limite superior para o número de estados, m, de uma implementação M', e seu número mínimo de classes de equivalência induzidas por R, denotado por n.

Uma vez escolhido um conjunto R de sequências de entrada e estimados os valores m e n, podemos obter as sequências de teste com a concatenação de dois conjuntos, $\pi = PZ$, em que P é um conjunto de cobertura de transições da especificação M, e Z é dado por

$$Z = X_{m-n}R\ .$$

Note que R não é calculado em função da especificação M, o que nos permite, em um caso extremo, utilizar $R = \{\varepsilon\}$, e fornecer $n = 1$, eliminando completamente o cálculo de outro conjunto, senão o de cobertura, P. No entanto, o tamanho das sequências de teste depende de R e, ainda, quanto maior o número de classes de equivalência induzidas, n, menor será o tamanho das sequências de Z e, consequentemente, menor será o conjunto de testes.

Ressaltamos também que n é um limite estimado sobre a implementação e quanto maior for o valor de n, menor será o tamanho do conjunto de testes. Em [2], demonstrou-se que para uma família infinita de MEFs, o método G gera conjuntos de testes exponencialmente mais eficientes que o método W. Isso permite que sejam criados conjuntos de testes menores, ajustando o nível de cobertura de falha desejado. Assim, podemos utilizar conjuntos de teste muito compactos em uma fase inicial da verificação, e, se necessário, complementar com conjuntos de testes mais rigorosos.

3.6 Teste incremental de MEFs

Normalmente, os métodos para testes de MEFs são utilizados para testes de novas implementações. No entanto, é comum que os sistemas evoluam, e, portanto tenham sua especificação alterada. Nesses casos, é esperado que o novo sistema seja baseado numa implementação anterior, obtida com algumas poucas modificações. Para tratar essa situação, El Fakih *et al.* [11] propuseram métodos para *teste incremental de MEFs*.

Dada uma especificação M e uma implementação M', suponha que M' já tenha sido testada, utilizando algum método existente. Suponha que um certo número de modificações sejam aplicadas a especificação M, obtendo uma nova especificação M_1. Similarmente, a nova implementação do sistema, M'_1, é obtida a partir da implementação anterior M'. O objetivo do teste incremental é comparar a M'_1 com M_1, utilizando um conjunto de testes significativamente menor do que se fosse utilizado um outro método para testar a implementação completa.

Assume-se que as novas especificação e implementação são obtidas de forma restrita, como modificações da saída de uma transição, a inclusão de um novo estado, etc. Também, o número de estados da implementação deve ser o mesmo da especificação. Para cada tipo de modificação, sequências de testes distintas são geradas. Nos testes realizados, mostrou-se que os ganhos foram significativos, especialmente quando as partes modificadas correspondem a menos de 20 porcento da especificação.

Capítulo 4

Generalização do Método Wp

Prólogo

Vários métodos de geração de testes utilizam conjuntos característicos [5, 12, 17] em algum passo de seus algoritmos. De forma geral, dada uma especificação, um conjunto característico é formado por sequências de entrada que distinguem cada par de estados não-equivalentes de um modelo [13]. O método G [3] é uma generalização do método W [5], e foi proposto para prescindir do cálculo dos conjuntos característicos.

Uma vantagem do método G sobre o método W é o fato de ser possível gerar casos de testes com diferentes níveis de cobertura de falhas, balanceando-se o tamanho desejado do conjunto de teste [2]. Nessa abordagem, é possível testar MEFs com conjuntos de testes menores inicialmente, aumentando o nível de cobertura de falhas com testes complementares, se necessário.

O método Wp é uma derivação do método W que reduz o tamanho do conjunto de testes utilizando o conceito de identificadores de estados. Dado um conjunto característico W, em uma fase inicial, verifica-se se existe uma correspondência entre as classes de equivalência de estados com relação a W, entre a implementação e a especificação. Caso o sistema em teste passe por essa primeira fase, então é possível complementar os testes na segunda fase usando apenas os identificadores de estados, ao invés dos conjuntos característicos. Como consequência, o conjuntos de testes gerado é menor do que aquele gerado pelo método W.

O artigo a seguir foi apresentado no *2nd Model-Based Verification & Validation Workshop*, como parte da *4th IEEE International Conference on Secure Software Integration and Reliability Improvement*, que ocorreu em Cingapura, em junho de 2010. O artigo apresenta um novo método de geração de casos de testes, denominado Gp, que combina as ideias do métodos G e Wp. Além disso, investiga quais os ganhos desse novo método comparado com os métodos nos quais foi baseado.

Generalized Partial Test Case Generation Method

Lehilton Lelis Chaves Pedrosa, Arnaldo Vieira Moura
Computing Institute, University of Campinas
P.O. 6176 – Campinas – Brazil – 13083-970
lehilton.pedrosa@students.ic.unicamp.br, arnaldo@ic.unicamp.br

Abstract

Finite State Machines (FSMs) are widely used to generate test case suites for critical systems. Several test case generation methods are derived from the well-known W-method. The Wp-method is a variation of the W-method, which produces smaller test suites. The G-method is a generalization of the W-method, which dispenses the use of characterization sets. In this paper, we combine ideas of both, and introduce a new generalized method, that does not depend on characterization sets, as the Wp-method does, and may produce smaller test suites than those of the G-method.

Keywords–testing; finite-state machines; test case selection;

I Introduction

Formal methods are extensively used to automatically generate conformance test cases for several critical hardware and software systems [1]. In several such methods, a Finite State Machine (FSM) is the basic model used to describe the system specification [2, 3, 4, 5, 6, 7, 8, 9, 10]. Usually, the black-box testing strategy is adopted so that, if an implementation passes the test suite, it is said to conform to the specification. The objective is to verify whether a given implementation conforms to the specification model, meaning that the desired behavior of the system is observed when the implementation is exposed to external stimuli [2]. For surveys on this topic, see [11, 12, 13].

Methods that automate the generation of test suites are, in general, required to have contrasting features [12, 14]:

1. *Efficiency*: the generated test suites must be small, so the implementation can be tested at low cost.

2. *Accuracy*: the generated test suites must detect as many implementation faults as possible.

The W-method [2] is a widespread technique that can be used to automatically construct test suites for FSMs. It is based on the notion of characterization sets [15] and provides full fault coverage for minimal, completely specified and deterministic FSMs. Several other methods are inspired in the W-method [3, 4, 5, 6, 7]. In particular, there are the Wp-method [3] and the more recent G-method [4]. The Wp-method, or partial W-method, uses the notion of identification sets to produce test suites which are subsets of those generated with the original W-method. The G-method, or Generalized W-method, adapts the W-method in such a way that the need of characterization sets is suppressed.

In this paper, we describe a new test case generation method, named *Generalized partial W-method*, that combines the ideas of the Wp-method and the G-method. For the sake of brevity, we may also write Gp-method. We introduce the new concept of class separators, so as to replace the identification sets of Wp-method, and, similarly to the G-method, we use arbitrary sets of sequences as a base to construct test suites. Thus, comparing with the W-method, the Gp-method does not depend on characterization sets, and produces smaller test suites. Also, our method applies to non-minimal FSMs.

We also discuss some differences between the G-method and the Gp-method. When using the G-method for certain FSMs, we may employ specific knowledge about the implementations, and generate very compact test suites with less stringent fault coverage criteria [14]. When using a partial method, we cannot use this information, in order to reduce the size of test suites. However, it turns out that when complete fault coverage is needed, or when it is not possible to relax the black-box assumption, the Gp-method can always yield smaller test suites.

In Section II, we describe the FSM model and fix some notations. In Section III, we review some related techniques. In Section IV, we introduce the concept of class separators, and then describe the Gp-method. In Section V, we prove that test suites generated using the Gp-method are m-complete. In Section VI, we discuss the differences between the Gp-method and the G-method.

II The FSM Model

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 α, $\beta \in A^*$ will be indicated by $\alpha\beta$.

A Finite State Machines

A FSM is a tuple $M = (X, Y, S, s_0, \delta, \lambda)$, where

- X is a finite input alphabet,
- Y is a finite output alphabet,
- S is a finite set of states,
- $s_0 \in S$ is the initial state,
- $\delta : X \times S \to S$ is the transition function, and
- $\lambda : X \times S \to Y$ is the output function.

Hereafter, we consider FSMs M and M' in the form $M = (X, Y, S, s_0, \delta, \lambda)$ and $M' = (X, Y', S', s'_0, \delta', \lambda')$. Note that they have the same input alphabet, X.

Let $s, r \in S$, $a \in X$ and $b \in Y$ be such that $\delta(a, s) = r$ and $\lambda(a, s) = b$. We say that, in state s, and given input a, the machine M changes to state r, yielding b as output. We denote this by $s \xrightarrow{a/b} r$.

The end state after the successive application of each input symbol of a word $\rho \in X^\star$ is given by the extended function $\widehat{\delta} : X^* \times S \to S$, defined by

$$\begin{aligned} \widehat{\delta}(\varepsilon, s) &= s, \\ \widehat{\delta}(a\rho, s) &= \widehat{\delta}(\rho, \delta(a, s)), \end{aligned}$$

for all $a \in X$, $\rho \in X^*$ and $s \in S$. If we are not interested in the output of ρ and $\widehat{\delta}(\rho, s) = r$, we may write $s \xrightarrow{\rho} r$.

Similarly, the output extended function is $\widehat{\lambda} : X^* \times S \to Y^*$, defined by

$$\begin{aligned} \widehat{\lambda}(\varepsilon, s) &= \varepsilon, \\ \widehat{\lambda}(a\rho, s) &= \lambda(a, s)\widehat{\lambda}(\rho, \delta(a, s)). \end{aligned}$$

We say that a FSM is connected if every state is reachable, as defined below.

Definition 1. *A state s is reachable if and only if there exists a word $\rho \in X^*$ such that $s_0 \xrightarrow{\rho} s$. A FSM M is connected if and only if every state $s \in S$ is reachable.*

Usually, a FSM is represented by a state diagram. Figure 1 illustrates a FSM.

B Cover sets

The notion of cover sets is used to describe test case generation of several methods. The idea is to a obtain sets of sequences such that every state and transition in the specification can be reached and exercised by one such sequence. We define state and transition cover sets.

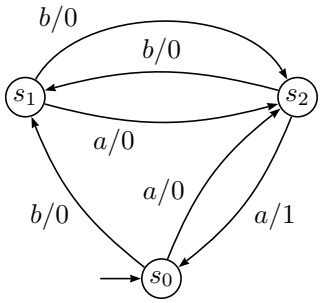

Figure 1: A simple FSM.

Definition 2. *A set of sequences $Q \subseteq X^*$ is a state cover set for a FSM M if, for every state $s \in S$, there exists $\rho \in Q$ such that $s_0 \xrightarrow{\rho} s$.*

Definition 3. *A set of sequences $P \subseteq X^*$ is a transition cover set for a FSM M if, for every state $s \in S$ and every symbol $a \in X$, there exist $\rho, \rho a \in P$ such that $s_0 \xrightarrow{\rho} s$.*

These definitions imply that every transition cover set P includes a corresponding state cover set Q. More precisely, there exists a state cover set Q such that $Q \subseteq P$ and, for every $s \in S$ and $a \in X$, there exists $\rho \in Q$ such that $s_0 \xrightarrow{\rho} s$ and $\rho a \in P$.

C State equivalence

In order to compare two FSMs, we will use state equivalences as conformance relations. Intuitively, two FSMs behave equivalently if, at the initial state, they produce the same output sequence as response to every input sequence. Next, we define this conformance relation formally.

Definition 4. *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. *Let $k \geq 0$. We say that s is k-equivalent to s' if s is X^k-equivalent to s'. In this case, we write $s \approx_k s'$. Otherwise, s is k-distinguishable from s', and we write $s \not\approx_k s'$.*

4. *State s is equivalent to s' if s is X^*-equivalent to s'. In this case, we write $s \approx s'$. Otherwise, s is distinguishable from s', and we write $s \not\approx s'$.*

We say that two FSMs are equivalent if their initial states are equivalent.

If M and M' are the same machines, the above relations are, in fact, equivalence relation over the set of states, and then $\approx_R$ induces a partition $[R]$ of the set of states S. Let M be a FSM, $R \subseteq X^*$ a set of input words, and $s \in S$ a state of M. We denote by $\overline{s}_R$ the class of the partition $[R]$ such that $s \in \overline{s}_R$. To avoid overloading the notation, if the partition is clear from the context, then the set R may be omitted, and we may simply write $\overline{s}$.

Definition 5. *Let M be a FSM. The index of M, ι_M, is the number of equivalence classes induced by the $\approx$ relation over the states of M. If $\iota_M = |S|$, then the machine is said to be minimal.*

D Relative concatenation

We use the conventional notation to represent the concatenation of two sets of input sequences, and we also define the set X_n as the set of all input words with length up to n, that is, $X_n = \bigcup_{k=0}^{n} X^k$. However, the usual concatenation operation is not particularly adequate to describe some FSM test methods, as we now explain. In general, when using a FSM test method, a set of input words must be applied to a set of states. This is handled by concatenating a state cover set with the set of input words. If to each state it must be applied a specific set of sequences, then the conventional concatenation operation is not appropriate. In order to alleviate this situation, the notion of relative concatenation was introduced in [3]. But, before precisely describing this new operation, we need the following definitions, where $\mathcal{P}(A)$ stands for the power set of a set A.

Definition 6. *Let M be a FSM. A state attribution is a function $\mathcal{B} : S \to \mathcal{P}(X^*)$.*

Definition 7. *Let M be a FSM and Π be a partition of S. A class attribution is a function $\mathcal{B} : \Pi \to \mathcal{P}(X^*)$.*

A state attribution can be induced by a class attribution in a very intuitive way. Let M be a FSM and $\mathcal{B}$ be 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 every $s \in C$ and every $C \in \Pi$.

Now, we can define the relative concatenation.

Definition 8. *Let M be a FSM, $A \subseteq X^*$, and $\mathcal{B}$ 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 leave the state index out of the operator symbol, and the relative concatenation is written as $A \otimes \mathcal{B}$. If $\mathcal{B}$ is a class attribution, then we may write $A \underset{s}{\otimes} \mathcal{B}$ to mean $A \underset{s}{\otimes} \overline{\mathcal{B}}$.

III Related works

In this section we briefly review the W-method. We also describe the generation of test suites using the Wp-method and the G-method. We can use each generated test suite, π, to check a FSM implementation, M', against its FSM specification, M, applying it to the algorithm CheckTestSuite in Figure 2. Clearly, CheckTestSuite returns `true` if and only if s_0 and s'_0 are π-equivalent.

```
function CheckTestSuite(M, M', π)
    for each σ ∈ π do
        if λ̂(σ, s_0) ≠ λ̂'(σ, s'_0) then
            return false
        end if
    return true
end function
```

Figure 2: General conformance test algorithm.

A The W-method

The W-method assumes that the specification is a minimal, completely specified and deterministic FSM M with n states. An implementation is assumed to be a completely specified FSM M', such that the number of states is not greater than a user provided constant m. Therefore, the generated test suite is m-complete [13]. We notice that m-complete test suites may have exponential size, so it is important that the number of states is small [2].

The method is based on two sets, P and Z. Set P is a transition cover set, and Z is a set used to distinguish pairs of states in implementations. Set Z is obtained concatenating a set of input words, X_{m-n}, with a characterization set for the specification, W, that is,

$$Z = X_{m-n}W.$$

A characterization set is basically a set of input sequences that distinguishes between the behaviors of every pair of states in the specification, as defined below.

Definition 9. *Let M be a minimal FSM. A set of input sequences W is a* characterization set *for M if and only if for every pair of states $s, r \in S$, with $s \neq r$, there exists an input sequence $\rho \in W$ such that $s \not\approx_\rho r$, and no proper subset of W has this property.*

Once we have P and Z, the desired test suite is given by

$$\pi = PZ.$$

B The Wp-method

The Wp-method [3] is a variation of the W-method that may potentially reduce the size of test suites. We present a version of this method that generates complete test suites for implementation with the same number of states as in the specification. However the technique can also be used with implementations with more states [3].

The Wp-method is also based on a transition cover set P and a characterization set W. However, the latter is used only in a first phase of the test. The main idea is to define a set of distinguishing sequences, called *identification sets*, that can be used to identify each state in the specification.

Definition 10. *Let M be a minimal FSM specification. A set of input sequences W_s is an* identification set *for state s if and only if for each state $r \in S$, with $s \neq r$, there exists an input sequence $\rho \in W_s$ such that $s \not\approx_\rho r$, and no proper subset of W_s has this property.*

Let P be a transition cover set for M and let Q be the associated state cover set. For each state $s \in S$ we also select an identification set W_s, and let

$$W = \bigcup_{s \in S} W_s.$$

The method is two-phased. In the first phase, an implementation is tested using a set of test cases π_1, the state cover set and the complete characterization set. If the implementation passes the first phase, then we apply a second set of test cases π_2, using only the remaining elements of the transition cover set and state identification sets. We define a state attribution $\mathcal{W}$ such that $\mathcal{W}(s) = W_s$ for each $s \in S$. The complete test suite is the union of

$$\pi_1 = QW \qquad \text{and} \qquad \pi_2 = (P \setminus Q) \otimes \mathcal{W}.$$

C The G-method

The G-method [4] is a generalization of W-method that dispenses the generation of the characterization set W. The main idea is to replace the set W by an arbitrary set of input sequences R. Since we do not need a characterization set for the specification, M is not required to be minimal.

Also, differently from W-method, n is defined as a lower bound on the number of equivalence classes induced by the relation $\approx_R$ on the states of implementation M'. This information is provided by the user. Let P be a transition cover set for M, and define $Z = X_{m-n}R$. Then, the generated test suite is

$$\pi = PZ.$$

IV The Generalized Partial W-method

In this section, we present the Gp-method. We use the same two-phase testing approach used by the Wp-method. That is, first we apply a set of test cases to the implementation. If the implementation passes this test, we then apply a set of complementary tests. So, the complete test suite is obtained by the union of the two partial test suites. The key difference between the Gp-method and the Wp-method is that, instead of using a characterization set W for the specification M, we use an arbitrary set of input sequences R and, instead of using identification sets, we use the more general concept of class separators.

We assume that the specification M is a connected FSM. This is reasonable since there is no meaning in having a specification with unreachable states. We also assume that m is an upper bound on the number of states of the implementation M'. A m-complete test suite will be obtained.

A Class separators

Now, we introduce the notion of class separators. The idea of class separators extends the notion of identification sets, in such a way that we can uniquely identify a whole class of states, instead of just one specific state.

Definition 11. *Let M be a FSM, $C \subseteq S$ a class of states, $T \subseteq X^*$ a set of input words. We say that T is a C-separator if and only if, for each pair of distinguishable states s and r, and such that $s \in C$ and $r \in S \setminus C$, we have $s \not\approx_T r$.*

Notice that this definition is a generalization of Definition 10. In fact, let M be a minimal FSM and W_s be an identification set for a state s, then W_s is a $\{s\}$-separator.

We have a correspondence between a state attribution and a set of class separators. Let M be a FSM, Π be a partition of S, and let $\mathcal{B}$ be a state attribution over S. We say that $\mathcal{B}$ is Π-*separating* if, for each state $s \in S$, $\mathcal{B}(s)$ is a $\bar{s}$-separator, where $\bar{s}$ is the class of Π such that $s \in \bar{s}$. Similarly, we can also relate a class attribution and a set of class separators. Let M be a FSM, Π be a partition of S, and $\mathcal{B}$ be a class attribution over Π. We say that $\mathcal{B}$ is Π-separating if $\overline{\mathcal{B}}$ is Π-separating.

Consider the state partition $[R]$ induced by $\approx_R$. The next lemma shows that in order to have a $\bar{s}$-separator it is enough to distinguish s from states of any other class.

Lemma 1. *Let M be a FSM, $R \subseteq X^*$ a set of input sequences, and $s, r \in S$ be states of M. Consider the partition $[R]$ and let $R_{\bar{s}}$ be a $\overline{s}$-separator, where $\bar{s} \in [R]$. If $s \approx_{R_{\bar{s}}} r$, then $s \approx_R r$.*

```
function TestSuite1(Q, R, m, n)
    Z ⟵ ⋃_{i=0}^{m−n} X^i R
    return QZ
end function

function TestSuite2(H, R, [R], m, n)
    for each C ∈ [R] do
        Select a minimal C-separator R_C
            with R_C ⊆ R
        R(C) ⟵ R_C
    for each s ∈ S do
        Z(s) ⟵ ⋃_{i=0}^{m−n} X^i ⊗_s R
    return H ⊗ Z
end function

procedure Gp-Method(M, M′, R, m)
    Obtain the partition [R] of S induced by ≈_R
    Obtain a transition cover set P for S with ε ∈ P
    Obtain a state cover set Q such that ε ∈ Q ⊆ P
    n ⟵ |[R]|
    if n ≤ m then
        π1 ⟵ TestSuite1(Q, R, m, n)
        if CheckTestSuite(M, M′, π1) then
            π2 ⟵ TestSuite2(P \ Q, R, [R], m, n)
            if CheckTestSuite(M, M′, π2) then
                mesg "M and M′ are equivalent",
                    and halt
            end if
        end if
    end if
    mesg "M and M′ are not equivalent"
end procedure
```

Figure 3: The Gp-method.

Proof. Assume that $s \approx_{R_{\bar{s}}} r$. Suppose, for the sake of contradiction, that $s \not\approx_R r$. Then $\overline{s} \neq \overline{r}$, so $r \notin \overline{s}$, that is, $r \in S \setminus \overline{s}$. Since $R_{\bar{s}}$ is a $\overline{s}$-separator, we have, by Definition 11, $s \not\approx_{R_{\bar{s}}} r$, a contradiction. □

B Test suite construction

We describe the construction of the two partial test suites, and then we obtain a complete test suite. The Gp-method is illustrated in Figure 3.

1 The first phase

In the first phase, we check for Z-equivalence between the states of the specification and the states of the implementation. Once the implementation passes the first phase, we use such Z-equivalence to ensure that the implementation satisfies certain conditions needed for the second phase.

We select a transition cover set, P, for the specification, with $\varepsilon \in P$. A transition cover set may be obtained from a labeled tree [14] in polynomial time. Then, we select the state cover set Q corresponding to P. That is, we select the minimal set Q, such that $\varepsilon \in Q \subseteq P$ and, for every $s \in S$ and $a \in X$, there exists $\rho \in Q$ such that $s_0 \xrightarrow{\rho} s$ and $\rho a \in P$.

Instead of a characterization set W, as is used by the Wp-method, we rely on any user provided set $R \subseteq X^*$ to generate the test sequences for this phase. We define n as the number of classes induced by the relation $\approx_R$ over the set of state S, *i.e.*, we set $n = |[R]|$. Then, we calculate $Z = X_{m-n}R$. The partial test suite for the first phase of the method is

$$\pi_1 = QZ.$$

2 The second phase

The second phase of the method complements the first partial test suite. It is used to ensure that, for each transition in the specification, we can associate one transition in the implementation. That is, if there exists a transition $r \to s$ that connects the specification states r and s, then there must exist a transition $r' \to s'$ for corresponding states of the implementation r' and s', where r is Z-equivalent to r', and s is Z-equivalent to s'.

Once the implementation passes the first test suite, we know that there exists a correspondence between the classes induced by $\approx_R$ in the specification and in the implementation. Then, we only need to use class separators to test for R-equivalence. Also, since Z is obtained from R, we need to use only subsets of Z to test for Z-equivalence. To test states with only the corresponding class separators, we define a class attribution $\mathcal{R}$, and, since to each state we apply a different subset of Z, we use a state attribution $\mathcal{Z}$.

First, we calculate a separator R_C for each class $C \in [R]$. That is, we define R_C to be the minimal subset of R such that R_C is a C-separator[1]. We can decide whether a set of input words is a class separator using the Definition 11 in polynomial time. Then, we define $\mathcal{R}$ to be a class attribution such that $\mathcal{R}(C) = R_C$, for each $C \in [R]$. Now, we use relative concatenation to obtain the state attribution $\mathcal{Z}$. For each $s \in S$, we compute $\mathcal{Z}(s) = X_{m-n} \underset{s}{\otimes} \mathcal{R}$.

The remaining sequences of the transition cover set are in $H = P \setminus Q$. Then, the partial test suite for the second phase is given by

$$\pi_2 = H \otimes \mathcal{Z}.$$

3 The test suite

The m-complete test suite is the union of the two partial test suites

$$\pi = \pi_1 \cup \pi_2.$$

[1] In fact, it is not necessary that $R_C \subseteq R$. Rather, we only need that $R_C \subseteq \text{pref}(R)$, where $\text{pref}(R)$ is the set of all prefixes of R.

C An example

As an illustration, we generate a m-complete test suite with $m = 3$, using the Gp-method for the FSM M illustrated in Figure 1. Note that M is connected.

For the first phase, we obtain a partial transition cover set P that includes ε. Then, we select the corresponding state cover set Q, and in such a way that $Q \subseteq P$ and $\varepsilon \in Q$. We then choose an arbitrary set of input sequences R, and partition the states of M in order to calculate the parameter n. Finally, we compute Z and the first partial test suite π_1. Note that, in this particular case, R is not a characterization set, since states s_0 and s_1 are distinguishable, but $s_0 \approx_R s_1$.

- $P = \{\varepsilon, a, b, aa, ab, ba, bb\}$
- $Q = \{\varepsilon, a, b\}$
- $R = \{a\}$
- $n = |[R]| = |\{\{s_0, s_1\}, \{s_2\}\}| = 2$
- $Z = X_{m-n}R = X_1R = \{\varepsilon, a, b\}\{a\} = \{a, aa, ba\}$
- $\pi_1 = QZ = \{a, aa, ba, aa, aaa, aba, ba, baa, bba\}$

Now, we choose a separator for each class induced by R, and construct the class attribution $\mathcal{R}$. We highlight that the procedure used by the Gp-method is slightly different from the procedure used by the Wp-method, since we choose each separator as a subset of the arbitrarily chosen set R whilst, in the Wp-method, a characterization set is obtained as a union of all identification sets. To obtain the second part of the test suite, we calculate the state attribution $\mathcal{Z}$, and concatenate the remaining sequences of the transition cover set H with $\mathcal{Z}$.

- $H = P \setminus Q = \{aa, ab, ba, bb\}$
- $\mathcal{R}(\{s_0, s_1\}) = \mathcal{R}(\{s_2\}) = \{a\}$
- $Z(s) = X_{m-n} \underset{s}{\otimes} \mathcal{R} = \{\varepsilon, a, b\}\{a\} = \{a, aa, ba\}$ for all $s \in \{s_0, s_1, s_2\}$
- $\pi_2 = H \otimes \mathcal{Z} = \{aaa, aaaa, aaba, aba, abaa, abba, baa, baaa, baba, bba, bbaa, bbba\}$

V Correctness of the method

In this section, we will show that a candidate implementation M' is equivalent to the specification M if and only if the implementation passes the Gp-method. First, we give necessary and sufficient conditions for checking whether two FSMs are equivalent. Since this theorem is a particular case of results in [4], we omit the proof.

Theorem 1. *Let M and M' be two FSMs with input alphabet X, and let $R \subseteq X^*$ be a set of input sequences. Assume that M' has at most m states, and that R partitions M in at least n classes, with $n \leq m$. Let also P be a transition cover set for M such that $\varepsilon \in P$. Then, $s_0 \approx s'_0$ if and only if $s_0 \approx_{PZ} s'_0$, where $Z = X_{m-n}R$.*

The next auxiliary result is immediate.

Lemma 2. *Let M and M' be two FSMs with input alphabet X, and let $Z \subseteq X^*$ be a set of input sequences. Let also Q be a state cover set for M such that $\varepsilon \in Q$. If $s_0 \approx_{QZ} s'_0$, then, for each state $s \in S$, there exists state $s' \in S'$ such that $s \approx_Z s'$, and $s_0 \approx_Z s'_0$.*

Now, we show that there exists an onto correspondence between R-equivalence classes of the specification M and R-equivalence classes of a candidate implementation M' which has already passed the first phase of the test.

Lemma 3. *Let M and M' be two FSMs with input alphabet X, and let $R \subseteq X^*$ be a set of input sequences. Assume that M' has at most m states, and that R partitions M in at least n classes, with $n \leq m$, and define $Z = X_{m-n}R$. Let also Q be a state cover set for M such that $\varepsilon \in Q$. If $s_0 \approx_{QZ} s'_0$, then, for each reachable state $r' \in S'$, there exists $r \in S$, such that $r \approx_R r'$.*

Proof. Let $I \subseteq S'$ be the set of states in M' such that $s' \in I$ if and only if there exists a state $s \in S$ with $s \approx_Z s'$. Now, let C be a class of $[R]$, and s a state of C. Since $s \in S$, from Lemma 2, there exists a corresponding state s' in S', such that $s \approx_Z s'$. So $s' \in I$. Since $\varepsilon \in X_{m-n}$ we get $R \subseteq Z$, and so $s \approx_R s'$. We have showed that, given a class $C \in [R]$, we obtain a state $s' \in I$ such that $s \approx_R s'$, for all $s \in C$.

Claim $|[R]| \leq |I|$: Suppose, for the sake of contradiction, that $|[R]| > |I|$. Then, by the pinhole principle, there exists two distinct classes $C_1, C_2 \in [R]$ corresponding to the same state s' in S'. Let $s_1 \in C_1$ and $s_2 \in C_2$ be such that $s_1 \approx_R s'$ and $s_2 \approx_R s'$. We get $s_1 \approx_R s_2$, and then we must have $s_2 \in C_1$. It follows that $C_1 \cap C_2 \neq \emptyset$, so $C_1 = C_2$, contradicting the fact that C_1 and C_2 are distinct. This establishes the Claim.

Now, let r' be a reachable state of S'. If $r' \in I$, then there exists $r \in s$, such that $r \approx_Z r'$, so $r \approx_R r'$, and we are done. Assume that $r' \notin I$. Since r' is reachable, there must exist an input sequence $\rho \in X^*$ such that $s'_0 \xrightarrow{\rho} r'$. Let ρ be such a sequence with the smallest length. Then, every state reached when ρ is applied starting at s'_0 must be distinct, and so $|\rho| \leq m$. Since $\varepsilon \in Q$, we get $s_0 \approx_Z s'_0$, and then, trivially, ρ reaches at least one state of I. Let s' be the last state of I reached when ρ is applied starting at s'_0. From $s' \in I$, there exists $s \in S$ such that $s \approx_Z s'$. Now take $\rho_1, \rho_2 \in X^*$ such that $\rho = \rho_1\rho_2$ and $s'_0 \xrightarrow{\rho_1} s'$. As ρ_2 does not reach any state of I other than s', and all states reached when ρ_2 is applied at s' are distinct, using the Claim, we get $|\rho_2| \leq m - |I| \leq m - |[R]| = m - n$. Let $r = \widehat{\delta}(\rho_2, s)$. Since $\rho_2 \in X_{m-n}$, $s \approx_Z s'$, and $Z = X_{m-n}R$, it follows that $s \approx_{\{\rho_2\}R} s'$, and then $r \approx_R r'$. This completes the proof. □

In order to check whether a state in the specification and a state in the implementation are R-equivalent, we can use only a class separator.

Lemma 4. *Let M and M' be two FSMs with input alphabet X, and let $R \subseteq X^*$ be a set of input sequences. Take $s \in S$ and $s' \in S'$. Consider the partition $[R]$, and let $R_{\bar{s}} \subseteq R$ be a $\overline{s}$-separator, where $\bar{s} \in [R]$. If there exists $t \in S$ such that $t \approx_R s'$, then $s \approx_R s'$ if and only if $s \approx_{R_{\bar{s}}} s'$.*

Proof. Assume that there exists $t \in S$ such that $t \approx_R s'$. Let $s \approx_R s'$. Since $R_{\bar{s}} \subseteq R$, it is clear that $s \approx_{R_{\bar{s}}} s'$. For the other direction, let $s \approx_{R_{\bar{s}}} s'$. Since $t \approx_R s'$, we get $t \approx_{R_{\bar{s}}} s'$. Then $s \approx_{R_{\bar{s}}} t$, and, from Lemma 1, $s \approx_R t$. It follows that $s \approx_R s'$. □

The next result states that we can use subsets of Z to check for Z-equivalence if the implementation has already passed the first phase of the method.

Lemma 5. *Let M and M' be two FSMs with input alphabet X. Let $R, U \subseteq X^*$ be two sets of input sequences and define $Z = UR$. Consider a $[R]$-separating class attribution $\mathcal{R}$ such that $\mathcal{R}(C) \subseteq R$, for every class $C \in [R]$. Take $s \in S$, $s' \in S'$, and define $Z_s = U \underset{s}{\otimes} \mathcal{R}$. If, for each $\rho \in U$, there exist $t \in S$ and $r' \in S'$ such that $s' \xrightarrow{\rho} r'$ and $t \approx_R r'$, then $s \approx_Z s'$ if and only if $s \approx_{Z_s} s'$.*

Proof. Assume that, for each $\rho \in U$, there exist $t \in S$ and $r' \in S'$ such that $s' \xrightarrow{\rho} r'$ and $t \approx_R r'$.

Let $\rho \in U$ be an input sequence, r and r' be states of M and M', respectively, with $s \xrightarrow{\rho} r$ and $s' \xrightarrow{\rho} r'$. Let $R_{\bar{r}} = \mathcal{R}(\overline{r})$. We know that $R_{\bar{r}} \subseteq R$, $R_{\bar{r}}$ is a $\overline{r}$-separator, and there exists $t \in S$ such that $t \approx_R r'$, then, from Lemma 4, $r \approx_R r'$ if and only if $r \approx_{R_{\bar{r}}} r'$. It follows that $s \approx_{\{\rho\}R} s'$ if and only if $s \approx_{\{\rho\}R_{\bar{r}}} s'$. As we have made no assumptions about ρ, the lemma holds. □

The first phase of the method uses only the partial test suite $\pi_1 = QZ$. From Theorem 1, we know that PZ is a complete test suite, so we need to complement π_1 with the remaining sequences in PZ, *i.e.*, $HZ = PZ \setminus QZ$. The next lemma states that it is not necessary to use all the sequences in HZ.

Lemma 6. *Let M and M' be two FSMs with input alphabet X, and let $R \subseteq X^*$ be a set of input sequences. Assume that M' has at most m states, and that R partitions M in at least n classes, with $n \leq m$. Let also Q, P be a pair of a state and a transition cover set for M such that $\varepsilon \in Q \subseteq P$, and define $H = P \setminus Q$ and $Z = X_{m-n}R$. Consider a $[R]$-separating class attribution $\mathcal{R}$ such that $\mathcal{R}(C) \subseteq R$, for every class $C \in [R]$, and let $\mathcal{Z}$ be a state attribution such that $\mathcal{Z}(s) = X_{m-n} \underset{s}{\otimes} \mathcal{R}$, for each state $s \in S$. If $s_0 \approx_{QZ} s'_0$, then $s_0 \approx_{HZ} s'_0$ if and only if $s_0 \approx_{H \otimes \mathcal{Z}} s'_0$.*

Proof. Assume $s_0 \approx_{QZ} s'_0$.

Let $\alpha \in H$ be an input sequence, s and s' be states of M and M', respectively, such that $s_0 \xrightarrow{\alpha} s$ and $s'_0 \xrightarrow{\alpha} s'$. Let $Z_s = \mathcal{Z}(s)$. For each $\rho \in X_{m-n}$, we obtain $r' \in S'$, such that $s' \xrightarrow{\rho} r'$. Since r' is reachable, from Lemma 3 there exists $t \in S$ such that $t \approx_R r'$. Now, using Lemma 5, we know that $s \approx_Z s'$ if and only if $s \approx_{Z_s} s'$. Then $s_0 \approx_{\{\alpha\}Z} s'_0$ if and only if $s_0 \approx_{\{\alpha\}Z_s} s'_0$. As we have made no assumptions about α, the lemma holds. □

Corollary 1. *Consider the hypotheses as in Lemma 6. Then, $s_0 \approx_{PZ} s'_0$ if and only if $s_0 \approx_{QZ} s'_0$ and $s_0 \approx_{H \otimes \mathcal{Z}} s'_0$.*

Proof. If $s_0 \approx_{PZ} s'_0$ then $s_0 \approx_{QZ} s'_0$ and Lemma 6 gives $s_0 \approx_{H \otimes \mathcal{Z}} s'_0$. Now, if $s_0 \approx_{QZ} s'_0$ and $s_0 \approx_{H \otimes \mathcal{Z}} s'_0$ then Lemma 6 again gives $s_0 \approx_{HZ} s'_0$. Since $H = P \setminus Q$, we get $s_0 \approx_{PZ} s'_0$. □

Now we combine the partial test suites of the two phases of the method to obtain a m-complete test suite. The following result ensures that the Gp-method is correct.

Theorem 2. *Let M and M' be two FSMs with input alphabet X, and let $R \subseteq X^*$ be a set of input sequences. Assume that M is connected, and that M' has at most m states. Then,* Gp-method(M, M', R, m)*, in Figure 3 correctly yields whether M and M' are equivalent.*

Proof. Notice that, since M is connected, we can obtain a state cover set Q, and a transition cover set P. Also, from construction, $\mathcal{R}$ is a $[R]$-separating class attribution.

Case $n > m$: Trivially, Gp-method states that M and M' are not equivalent. Suppose, for contradiction, that M and M' are equivalent. Then, clearly, for each $s \in S$, there exists a $s' \in S'$, such that $s \approx s'$. Since $|S'| \leq m < n = |[R]|$, we obtain two distinct classes $C_1, C_2 \in [R]$, states $s_1 \in C_1$, $s_2 \in C_2$, and a state $s' \in S'$, such that $s' \approx s_1$ and $s' \approx s_2$. Hence, $s_1 \approx s_2$. But $s_1 \not\approx_R s_2$, and so $s_1 \not\approx s_2$, a contradiction. Then, in fact, M and M' are not equivalent.

CASE $n \leq m$: 1) Suppose that GP-METHOD states that M and M' are equivalent. It follows that the function calls of CHECKTESTSUITE(M, M', π_1) and of CHECKTESTSUITE(M, M', π_1) returned `true`. Then, clearly, $s_0 \approx_{\pi_1} s'_0$ and $s_0 \approx_{\pi_2} s'_0$, and thus $s_0 \approx_{QZ} s'_0$ and $s_0 \approx_{H \otimes Z} s'_0$. Using Corollary 1, we obtain $s_0 \approx_{PZ} s'_0$, and using Theorem 1, $s_0 \approx s'_0$, so M and M' are equivalent in fact.

2) Now, suppose that GP-METHOD states that M and M' are not equivalent. It follows that either the call CHECKTESTSUITE(M, M', π_1), or the call CHECKTESTSUITE(M, M', π_2) returned `false`. Then, clearly, $s_0 \not\approx_{\pi_1} s'_0$, or $s_0 \not\approx_{\pi_2} s'_0$, so $s_0 \not\approx s'_0$, and, in fact, M and M' are not equivalent.

We conclude that GP-METHOD is correct. □

VI Comparison

There are two differences between the G-method and the Gp-method. As we pointed above, one difference is that, in Gp-method, we use only part of the set R in the second phase of the test, thus possibly reducing the test case suite that is generated. The other main difference is that, with the Gp-method we need only information from the specification whereas, using the G-method, we must rely on user provided information about the implementation. More specifically, in the Gp-method we set n as a lower bound on the number of classes induced by R in the *specification*, and in the G-method n is set as a lower bound on the number of classes induced by R in the *implementation*.

Although providing such information about the implementation may lead to more succinct test suites in some specific situations, it somewhat relaxes the black-box assumption about the implementation. In fact, for certain infinite families of FSM implementations, the G-method generates very compact test suites [14]. In the discussion that follows, we will expose the difficulty of using similar ideas in partial test methods. On the other hand, when the black-box assumption cannot be relaxed, or when m-complete test suites are needed, we can always obtain test suites generated with the Gp-method that are subsets of the test suites generated using the G-method.

Limitation of partial test methods Once the implementation has passed the first phase of the Wp-method, we know that, for each R-equivalence class in the specification, there exists a corresponding R-equivalence class in the implementation. The converse is not always true if the implementation may have more states than the specification. In this case, there may exist states in the implementation that are R-distinguishable with respect to any state in the specification. Therefore, class separators for the specification are not class separators for the corresponding implementation classes.

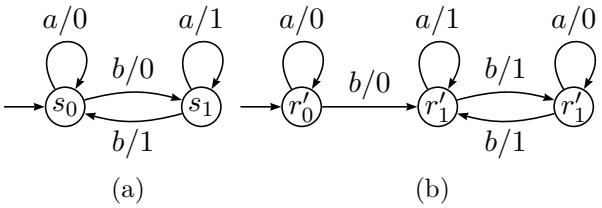

Figure 4: Specification and candidate implementation.

As a consequence, we may not generate a m-complete partial test suite if n is defined as in the G-method. As an example, we will obtain a test suite for the specification depicted in Figure 4(a) using the same procedure as in the Gp-method with $m = 3$. Now, we will also set n to a lower bound on the number of classes induced by R in *implementation*. With a candidate implementation as depicted in Figure 4(b) we can produce a counterexample to show that the obtained test suite is not m-complete.

We use $R = \{a, b\}$ and set $n = 3$. The first partial test suite is computed below.

- $P = \{\varepsilon, a, b, ba, bb\}$
- $Q = \{\varepsilon, b\}$
- $Z = X_{m-n}R = X_{3-3}R = X_0R = \{\varepsilon\}\{a, b\} = \{a, b\}$
- $\pi_1 = QZ = \{a, b, ba, bb\}$

Now, we compute the second partial test suite:

- $H = P \setminus Q = \{a, ba, bb\}$
- $\mathcal{R}(\{s_0\}) = \{a\}$ and $\mathcal{R}(\{s_1\}) = \{b\}$
- $Z(s_0) = X_{m-n} \underset{s_0}{\otimes} \mathcal{R} = X_0 \underset{s_0}{\otimes} \mathcal{R} = \{\varepsilon\} \underset{s_0}{\otimes} \mathcal{R} = \{\varepsilon\}\mathcal{R}(s_0) = \{\varepsilon\}\{a\} = \{a\}$
- $Z(s_1) = X_{m-n} \underset{s_1}{\otimes} \mathcal{R} = X_0 \underset{s_1}{\otimes} \mathcal{R} = \{\varepsilon\} \underset{s_1}{\otimes} \mathcal{R} = \{\varepsilon\}\mathcal{R}(s_1) = \{\varepsilon\}\{b\} = \{b\}$
- $\pi_2 = H \otimes \mathcal{Z} = \{a, ba, bb\} \otimes \mathcal{Z} = \{a, bb\}\mathcal{Z}(s_0) \cup \{ba\}\mathcal{Z}(s_1)$
  $= \{a, bb\}\{a\} \cup \{ba\}\{b\} = \{aa, bba\} \cup \{bab\} = \{aa, bba, bab\}$

The generated test suite is the union $\pi = \pi_1 \cup \pi_2 = \{a, b, ba, bb, aa, bba, bab\}$. Note that $s_0 \approx_\pi r'_0$, but for $\gamma = bbb$ we have $\widehat{\lambda}(\gamma, s_0) = 010$ and $\widehat{\lambda}'(\gamma, r'_0) = 011$, then $s_0 \not\approx_\gamma r'_0$, and so $s_0 \not\approx r'_0$. Therefore, this specification and implementation pair shows that a m-complete test suite cannot be obtained using Gp-method if we set n to be a lower bound to the number of classes induced by R in the *implementation*.

VII Conclusions

The G-method is a generalization of the W-method that avoids the necessity of using characterization sets. In this paper, we extended the main ideas of the G-method to the Wp-method, introducing a new test case generation method, named the Gp-method. Using the Gp-method, we can use only part of the sequences generated using the G-method in a second phase of the test, thus reducing the test case suite.

In order to avoid characterization sets, we generalized the notion of identification sets used by Wp-method. We introduced the concept of class separators. A separator is used to uniquely identify one class of states as opposed to identification sets which were used to uniquely identify single states. We also used the concept of relative concatenation, introduced in [3], and gave a precise definition of this operation, which was lacking in the original paper.

We compared the new method with the G-method. The latter relaxes the black-box assumption and may generate much more succinct test suites with less stringent fault coverage. We showed, using a counterexample, that we cannot use the same information when using partial test suites. However, when m-complete test suites are required, the Gp-method generates smaller test suites than the G-method.

Acknowledgment

The authors would like to thank FAPESP for its financial support (grant # 2008/07969-9).

References

[1] E. M. Clarke and J. M. Wing, "Formal methods: state of the art and future directions," *ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, 1996.

[2] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on Software Engineering*, vol. 4, no. 3, pp. 178–187, 1978.

[3] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 591–603, 1991.

[4] A. L. Bonifácio, A. V. Moura, and A. S. S. ao, "A generalized model-based test generation method," in *6th IEEE International Conferences on Software Engineering and Formal Methods*, 2008, pp. 139–148.

[5] R. Dorofeeva, K. El-Fakih, and N. Yevtushenko, "An improved conformance testing method," in *IFIP 25th International Conference on Formal Techniques for Networked and Distributed Systems*, 2005, pp. 204–218.

[6] G. Luo, A. Petrenko, and G. V. Bochmann, "Selecting test sequences for partially-specified nondeterministic finite state machines," in *IFIP 7th International Workshop on Protocol Test Systems*, 1994, pp. 91–106.

[7] L. L. C. Pedrosa and A. V. Moura, "Testing combined finite state machines," Institute of Computing, University of Campinas, Tech. Rep. IC-10-01, 2010.

[8] A. Petrenko, "Fault model-driven test derivation from finite state models: annotated bibliography," in *Modeling and verification of parallel processes*, vol. 2067, 2001, pp. 196–205.

[9] G. Luo, G. V. Bochmann, and A. Petrenko, "Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method," *IEEE Transactions on Software Engineering*, vol. 20, no. 2, pp. 149–162, 1994.

[10] C. Robinson-Mallett, P. Liggesmeyer, T. Mücke, and U. Goltz, "Extended state identification and verification using a model checker," *Information & Software Technology*, vol. 48, no. 10, pp. 981–992, 2006.

[11] G. V. Bochmann and A. Petrenko, "Protocol testing: review of methods and relevance for software testing," in *ISSTA '94: Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, 1994, pp. 109–124.

[12] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines - a survey," in *Proceedings of the IEEE*, 1996, pp. 1090–1123.

[13] D. Sidhu and T.-K. Leung, "Formal methods for protocol testing: a detailed study," *IEEE Transactions on Software Engineering*, vol. 15, no. 4, pp. 413–426, 1989.

[14] A. L. Bonifácio, A. V. Moura, and A. S. S. ao, "Exponentially more succinct test suites," Institute of Computing, University of Campinas, Tech. Rep. IC-09-07, 2009.

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

Epílogo

Nesse artigo, combinamos dois métodos, apresentando um novo, o método Gp. Mais especificamente, generalizamos o método Wp, usando as mesmas ideias vindas do método G. Desse modo, o método Gp prescinde do cálculo de conjuntos característicos, assim como o método G, e utiliza, em uma segunda fase de teste, conjuntos de testes parciais, assim como o método Wp, reduzindo o tamanho do conjunto de testes. Além disso, o novo método se aplica MEFs não minimais, ao contrário do método Wp.

Assim como feito no método G, nós substituímos o conjunto caraterístico, usado pelos métodos W e Wp, por conjuntos arbitrários fornecidos pelo usuário. Uma implicação disso, após a primeira fase do teste, é que não obtemos uma correspondência entre estados da especificação e implementação, como ocorria no método Wp. Por esse motivo, a utilização de identificadores de estados não seria suficiente na segunda fase do teste. Para resolver esse problema, introduzimos e utilizamos o conceito de separadores de classes, que é uma generalização do conceito de identificador de estado.

Na segunda fase do método, concatenamos sequências remanescentes de um conjunto de cobertura de transições com sequências de separadores de classes. Aplicamos cada sequência, ρ, do conjunto de cobertura na especificação, e identificamos qual estado, s, foi atingido por ρ. Depois disso, concatenamos ρ com cada sequência do separador da classe de s. Esse tipo de concatenação foi introduzida informalmente em [12]. Aqui, formalizamos esse conceito, introduzindo atribuições de classe e estados, que associam um conjunto de palavras a cada classe ou estado, e definindo o operador de concatenação relativa.

Comparamos o novo método com o método G. A diferença fundamental entre os dois métodos é a definição do parâmetro n. Enquanto n para o método G é um limite inferior estimado do número de classes de R-equivalência na implementação, para o método Gp, n é o número calculado de classes de R-equivalência na especificação. Uma consequência importante dessa diferença é que para o novo método não é possível obter conjuntos de testes compactos balanceando-se os níveis de cobertura de falhas desejados. Por outro lado, quando a cobertura de falhas completa é desejada, isto é, é necessário obter conjunto de testes m-completos, o método Gp sempre produz conjuntos de testes menores que o método G.

No artigo, mostramos, por meio de um contraexemplo, que não é possível utilizar conjuntos parciais, como os do método Wp, e obter as mesmas vantagens do método G descritas acima. Questões que surgem dizem respeito à possibilidade de realizar esse mesmo tipo de generalização a outro métodos que utilizam identificadores de estados harmonizados, como o método HSI.

Capítulo 5

Um novo método para teste de MEFs

Prólogo

Os conjuntos de testes que proveem cobertura de falhas completa são exponenciais, o que torna impraticável o teste de MEFs com número elevado de estados. Isso significa que, com os métodos de geração de casos de teste existentes, só é possível testar sistemas cujas especificações sejam modeladas por MEFs com conjuntos pequenos de estados. No entanto, em aplicações reais, os sistemas são modulares e podem ser compostos de vários subsistemas. Obviamente, o teste de cada módulo separadamente não garante o correto funcionamento do sistema como um todo.

Para tais sistemas modulares, que são criados a partir de módulos previamente testados, é ineficiente utilizar um dos métodos de geração de caso de testes existentes. Isso porque tais métodos assumem que a implementação é uma completa caixa preta e ignoram a informação acerca de seus subsistemas. Decorrência disso é que, se for utilizado um método existente, todo o sistema seria novamente testado, incluindo todos os seus submódulos.

Situação semelhante também ocorre no *reteste* de sistemas modificados. Os sistemas evoluem com o passar do tempo e, normalmente, as novas especificações são obtidas aplicando pequenas alterações de uma especificação anterior. Da mesma forma, é razoável assumir que a nova implementação será obtida a partir de uma implementação anterior, com poucas modificações. Novamente, a utilização da maioria dos métodos de testes existentes, como o método W, seria ineficiente, já que implicaria no retestes da implementação, inclusive das partes inalteradas.

Para resolver problemas como os descritos acima, é necessário a utilização de métodos de teste incrementais. Por exemplo, em [11], métodos de geração de conjuntos de testes

eficientes são propostos para casos em que especificação e implementação têm tipos controlados de alterações. Em outro artigo [15], métodos são propostos para geração de casos de testes para tipos restritos de falhas, determinados pelo usuário.

O artigo a seguir foi apresentado no *2nd NASA Formal Methods Symposium*, que ocorreu em Washington, D.C., em abril de 2010. Ele apresenta uma formalização de subsistemas no contexto de MEFs, definindo o conceito de submáquinas e MEFs combinadas. Depois, apresenta um novo método, denominado método C, para o teste de MEFs combinadas em situações em que as submáquinas já foram previamente testadas. Com o novo método, é possível testar MEFs, mesmo que tenham um número elevados de estados. Além disso, desenvolveu-se uma estratégia incremental de teste de MEFs.

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.

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

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 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 α, $\beta \in A^*$ will be indicated by $\alpha\beta$.

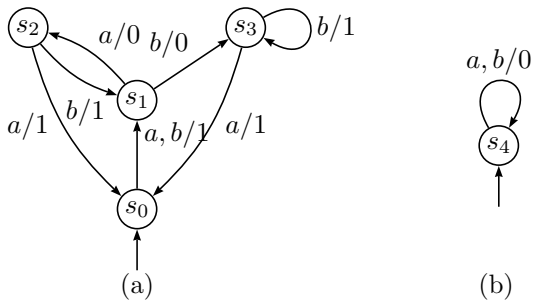

Figure 1: Finite State Machines.

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 $\mathcal{P}(A)$ 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 $\mathcal{B} : S \to \mathcal{P}(X^*)$. A class attribution is a function $\mathcal{B} : \Pi \to \mathcal{P}(X^*)$.* □

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.

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 $S_N = \{s_0, s_1, s_2, s_3, s_4\}$,

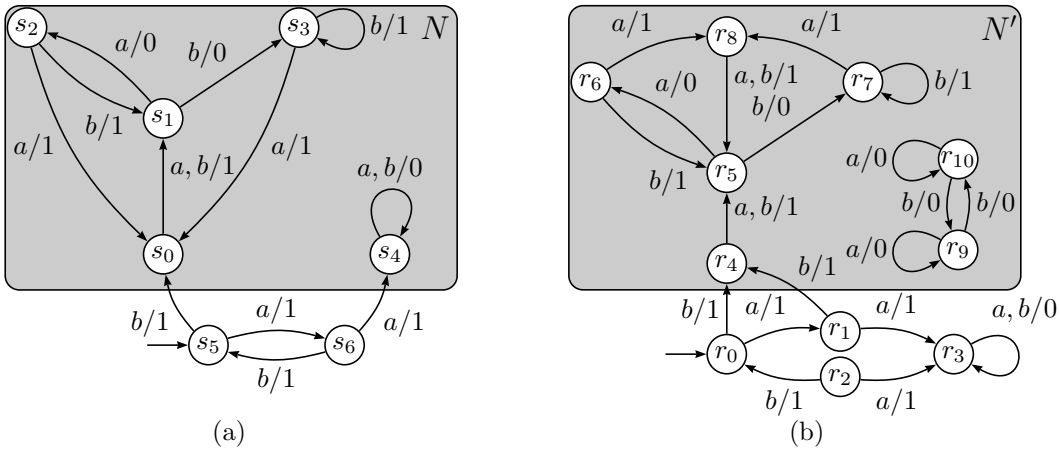

Figure 2: A Combined Finite State Machine and a candidate implementation.

$I_N = \{s_0, s_4\}$ 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.

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 $\pi \longleftarrow P \otimes \mathcal{Z}$;

end

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]. □

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 $n = \max\{l', l\} = 2$;
- $A = \mathrm{nbh}(I_N, m{-}n{-}1) = \mathrm{nbh}(\{s_0, s_4\}, 1) = \{s_0, s_1, s_4\}$;
- $T = \{aaaa\}$;
- $\mathcal{R}(S_N) = T \cup R = R$ and $\mathcal{R}(S_M) = R$;
- $\mathcal{Z}(s) = X_{m-n} \underset{s}{\otimes} \mathcal{R} = X_{m-n} R = X_2 R$ for every $s \in S$.

So, $\pi = P \otimes \mathcal{Z} = P X_2 R$ 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

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

$$\textit{and (iii)} \quad |\pi_W| \in O((j+l)^3|X|^{m-l+k-j+1}),$$

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.

Epílogo

Apresentamos um novo método, denominado método C, para casos em que uma especificação é uma composição de outros subsistemas. O novo método é útil para o teste eficiente de sistemas modulares, quando subsistemas são testados previamente. Além disso, possibilita utilizar um conjunto de testes compacto para o reteste de implementações que sofreram alterações em apenas um de seus subsistemas.

Para descrever formalmente um sistema modular especificado como uma MEF, definimos o conceito de submáquina e introduzimos a noção de MEF combinada. Uma submáquina é uma MEF cujos estados também fazem parte da MEF que a contém. Uma MEF combinada é obtida pela união de submáquinas que são interligadas utilizando estados e transições adicionais.

Para testar sistemas nas situações descritas acima, definimos um modelo de falhas para MEFs combinadas. Assumimos que cada submáquina foi implementada separadamente. Além disso, assumimos que a implementação de uma MEF combinada também pode ser modelada como uma MEF combinada e que foi obtida a partir das implementações das submáquinas, testadas previamente, juntamente com um número limitado de estados adicionais. Utilizando esse modelo de falhas, foi possível garantir o correto funcionamento de uma implementação, testando-se apenas os estados adicionais.

Como no teste de MEFs combinadas, não precisamos testar as submáquinas, descobrimos que não precisamos utilizar conjuntos característicos completos, utilizados para separar cada par de estados. Portanto, para diminuir o tamanho dos conjuntos de teste gerados, introduzimos o conceito de separadores de estados. Um separador é um conjunto de sequências de entradas utilizado para separar estados de dois conjuntos dados. A utilização dos separadores mostrou-se uma ferramenta valiosa para testes de MEFs, uniformizando conceitos relacionados oriundos de vários métodos. De fato, a noção de separadores é uma generalização de conjuntos característicos, de identificadores de estados e de separadores de classe.

Um dos principais motivos para o método C gerar conjuntos mais compactos que outros métodos é o fato de ser necessário testar apenas os estados adicionais. Por esse motivo, utilizamos somente parte de um conjunto de cobertura. Nós definimos conjuntos parciais de cobertura de estados e de cobertura de transições, de forma que os estados adicionais sejam os únicos a serem testados.

Comparamos o método C com o método W. Mostramos um exemplo em que o conjunto de testes gerado pelo novo método é sensivelmente menor. De maneira mais geral, mostramos que é sempre possível obter conjuntos de testes utilizando o método C a partir de um subconjunto pequeno dos testes gerados pelo método W.

Investigamos o número de sequências geradas pelos dois métodos. Uma análise as-

sintótica mostrou que, assim como o método W, o número de sequências geradas pelo método C pode ser exponencial no número de estados. No entanto, os expoentes que limitam o conjunto de teste do método C podem assumir valores significativamente menores que os correspondentes no método W, caso o número de estados adicionais seja mantido baixo. Concluímos que, satisfeita essa condição, podemos testar MEFs com números arbitrários de estados.

Capítulo 6

Teste incremental de MEFs

Prólogo

Como visto anteriormente, devido aos conjuntos de testes gerados com cobertura completa de falhas serem exponenciais no número de estados, para testar MEFs completamente, os métodos requerem que o número de estados da especificação seja pequeno. Portanto, estratégias de testes alternativas devem ser adotadas, já que não se espera obter conjuntos de testes com cobertura completa sensivelmente menores. Tais estratégias incluem restringir o modelo de falhas, ou utilizar teste incremental de sistemas.

Com o método C aqui apresentado, mostramos que é possível testar especificações de MEFs combinadas, com um número elevado de estados, em situações em que submáquinas já foram previamente testadas. Suponha que essas submáquinas também sejam combinadas. Então podemos utilizar o próprio método C para testá-las, dando origem a um procedimento recursivo. Inspirados nisso, propusemos uma nova estratégia de testes de MEFs em geral, visando tornar o procedimento de teste mais escalável.

O artigo a seguir foi submetido ao periódico *IEEE Transactions on Software Engineering*. Ele é uma extensão do artigo anterior. Descreve o método C e apresenta uma nova estratégia incremental de teste de MEFs. Além disso, é incluída uma demostração da corretude do método, ausente no outro artigo.

A estratégia apresentada corresponde a abstrair cada especificação como uma MEF combinada e testar cada submáquina utilizando o método C, recursivamente. Nós comparamos essa estratégia com a abordagem tradicional e mostramos que, para uma família infinita de MEFs, que ocorrem naturalmente na prática, a nova abordagem gera conjuntos de testes exponencialmente mais eficientes.

Incremental Testing of Finite State Machines

Lehilton Lelis Chaves Pedrosa, Arnaldo Vieira Moura
Institute of Computing, University of Campinas
lehilton.pedrosa@students.ic.unicamp.br, arnaldo@ic.unicamp.br

Abstract

The automatic generation of test case suites for systems modeled as Finite State Machine (FSMs) is an important testing problem in several critical applications. Known methods that automatically generate test cases for FSMs, specially the W-method and derivations, strongly assume that the number of system states is small. If the overall number of states in the FSM specification is relatively large, such methods become very difficult to use. However, often in practice a system is defined as a combination of several subsystems, with the latter already independently designed, developed and tested. In this paper, we define the concept of combined FSMs and introduce a new method to test modular compositions of FSMs. The new method is scalable on the number of states. It can so be used for incremental testing of new systems, or for retesting modified implementations taking advantage of the fact that several subcomponents have already been tested and need not be tested again. Further, we present an infinite family of naturally occurring FSM models for which our method produces exponentially more compact test suites than the W-method.

Keywords–finite state machine, testing, verification, test generation

I Introduction

Automatic test case generation for reactive and critical systems using formal methods has been widely studied [1, 2, 3, 4, 5, 6, 7, 8]. With an automated generated test case suite, one tries to ensure that a candidate implementation meets the system's requirements, the latter being described by means of mathematical models and formally specified functionalities. When applying these testing strategies, the notion of conformance testing is often used, in the sense that if an implementation passes a given test suite, then its behavior is said to conform to the behavior extracted from the specification [9]. In order to ensure that test suites are useful, we require that they are: *efficient*, so that the overall sum of

test case length is small [10]; and *accurate*, so that any fault in the implementation is detected by one of the test cases [6].

Finite State Machines (FSMs) are usually the basic formalism in many methods that automate the generation of conformance test case suites. For surveys on this topic, see [1, 6, 8]. Among such methods, the so called W-method [3] provides full fault coverage for minimal, completely specified and deterministic FSMs. The size of the generated test suite might be exponential, in the worst case, and so the W-method assumes that, in practice, the number of states in the specification model is kept small. Otherwise, if the number of states is large, then the size of the test suite could explode, and the method becomes impractical to use.

Several derivations have been proposed around the W-method. For example, the Wp-method [5] reduces the size of the test suite using the notion of identification sets, and the G-method [2] generalizes the W-method, using any set of input words as a basis to construct the test suite, instead of forcing the use of characterization sets. Nevertheless, in order to be useful, these derivations still require that the number of states be small.

Current FSM test methods treat a specification model as a single block, and generate test suites to verify whether each and any state in the specification model has a corresponding correctly implemented state in the implementation under test. This often results in very large test suites, making these methods inadequate when the specification FSM has a large number of states. However, in many practical situations, we find that systems are modular, with their specifications being formed by several subsystems. In this case, it is reasonable to test each subsystem separately, and only later attempting to test the composite system without retesting the component subsystems again. This strategy might give rise to much more succinct test suites.

If one such subsystem is also modeled by a FSM, we call it a submachine. The overall FSM model can then be obtained by conjoining several submachines, with the aid of a few new states and transitions. In this article, we propose a new approach to test combined FSMs, which could be specially useful in cases when submachine implementations are known to be correct in advance. In this strategy, the specification model is first abstracted as a combined FSM, then each submachine is implemented and tested independently. Finally, the integrated system is verified by a very succinct test suite.

This new strategy makes testing FSMs an incremental process. For example, suppose that a FSM is abstracted as a set of submachines, and that one such submachine is also abstracted as a combined submachine, and so on. First, we would have to test the atomic subsystems, then the subsystems composed by simpler subsystems, and the more complex subsystems, incrementally, until we have tested the entire specification. One could also test the implementation by applying some traditional test case generation methods, but, since such methods ignore the information about the submachines, they

would be redundant, and thus inefficient.

Abstracting the specification as a combined FSM is also useful in the case of retesting modified systems. In fact, systems evolve with time, so the specification is very likely to be changed along time. If the new system is obtained from the older one, then, only the corresponding part of a former implementation is modified. Again, retesting the entire system using methods like the W-method or the G-method could be inefficient. Alternatively, one can employ our strategy, and test only the submachines that were altered.

Some related work on retesting modified implementations has been done by Koufareva *et al.* [11], who presented methods for restricted types of errors, and El-Fakih [12, 13], who introduced methods for retesting an implementation with controlled types of modifications. In this paper, we do not restrict the types of errors the implementation can harbor, neither how it is modified. Additionally, we allow implementations with more states than the specification.

To illustrate the new approach, we show how it can be used to test a model specification with a large number of states. To make explicit the scalability of the new method, we show that our method can be more efficient then the W-method in an infinite class of naturally occurring FSM models. In fact, we show that the new method generates test suites exponentially more succinct than those generated by the W-method, when applied over such infinite families.

In Section II, we review the FSM model and present some conventions used in this paper. In Section III, we describe equivalence relations between states of two FSMs. In Section IV, we present the new test case generation method, here named the C-method. In Section V, we show that the C-method is exponentially more efficient than the W-method over some infinite classes of FSMs. We concentrate the technical arguments showing the correctness of the C-method in Section VI. Finally, in Section VII we present our conclusions.

II FSM Basics

In this section, the FSM model is reviewed and we also introduce some conventions used in this work. Concepts related to the concatenation of words, reachability of states, and partial cover sets are presented.

A Finite State Machines

A FSM is a tuple $M = (X, Y, S, s_0, \delta, \lambda)$, where X is a finite input alphabet, Y is a finite output alphabet, S is the set of states, $s_0 \in S$ is the initial state, $\delta : X \times S \to S$ is the

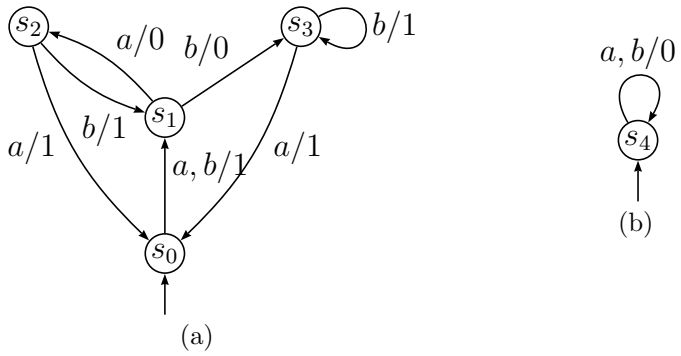

Figure 1: Finite State Machines.

transition function, and $\lambda : X \times S \to Y$ is the output function. Hereafter, we will consider FSMs $M = (X, Y, S, s_0, \delta, \lambda)$ and $M' = (X, Y', S', s'_0, \delta', \lambda')$ with the same input alphabet, X.

Let $r, s \in S$, $a \in X$ and $b \in Y$ be such that $\delta(a, r) = s$ and $\lambda(a, r) = b$. We say that, in state r, and given input a, M changes to state s and yields b as output. We also denote this by $r \xrightarrow{a/b} s$. The empty word will be denoted by ε and the set of all finite words over an alphabet X will be denoted by X^*. Input and output sequences of symbols will, usually, be represented by words $\rho \in X^*$ and $\sigma \in Y^*$, respectively. The end state reached and the sequence of output symbols generated after the successive application of an input word will be given, respectively, by the extended functions $\widehat{\delta} : X^* \times S \to S$ and $\widehat{\lambda} : X^* \times S \to Y^*$, defined inductively as

$$\begin{array}{rclcrcll} \widehat{\delta}(\varepsilon, s) & = & s, & & \widehat{\delta}(a\rho, s) & = & \widehat{\delta}(\rho, \delta(a, s)), & \text{and} \\ \widehat{\lambda}(\varepsilon, s) & = & \varepsilon, & & \widehat{\lambda}(a\rho, s) & = & \lambda(a, s)\widehat{\lambda}(\rho, \delta(a, s)), & \end{array}$$

where $a \in X$, $\rho \in X^*$ and $s \in S$. If $\widehat{\delta}(\rho, s) = r$ and $\widehat{\lambda}(\rho, s) = \sigma$, we may also write $s \xrightarrow{\rho/\sigma} r$, and, if we are not interested in the output, we may simply write $s \xrightarrow{\rho} r$.

Usually, a FSM is represented by a state diagram, as in Figure 1. We will be referring to this figure throughout. Clearly, the arrows indicate state transitions, and a label $a/0$ indicates that the machine on input a outputs the symbol 0. To simplify figure labellings, more than one transitions may be represented together, *e.g.*, in $a, b/1$.

B Concatenation of words and relative concatenation

We adopt the usual notation for the concatenation of two sets of words, and we let X^n denote the set of all word of length n over X. Also, we indicate by X_n the set of all words with length at most n over X. More specifically, let $A, B \subseteq X^*$, $\rho \in X^*$, and let

n be a non-negative integer. Then $AB = \{\alpha\beta | \alpha \in A, \beta \in B\}$, $\rho A = \{\rho\}A$, $A\rho = A\{\rho\}$, $X^n = \{\rho \in X^* |\ |\rho| = n\}$, and $X_n = \bigcup_{k=0}^{n} X^k$.

In several situations, a set of input words has to be applied to a set of initial states. If to each initial state, we will want to apply a specific set of words, then the conventional concatenation operation is not enough. To address this problem, the notion of relative concatenation was introduced in [5]. To formally describe this operation, we first define state and class attributions. Let $\mathcal{P}(A)$ denote the power set of a set A.

Definition 1. *Let M be a FSM, and Π be a partition of S. A state attribution is a function $\mathcal{B} : S \to \mathcal{P}(X^*)$. A class attribution is a function $\mathcal{B} : \Pi \to \mathcal{P}(X^*)$.*

A state attribution can be induced by a class attribution in a very intuitive fashion. Let M be a FSM and $\mathcal{B}$ be a class attribution over a partition Π, then the induced state attribution is $\overline{\mathcal{B}}$, where $\overline{\mathcal{B}}(s) = \mathcal{B}(C)$, for every $s \in C$ and every $C \in \Pi$.

Definition 2. *Let M be a FSM, $A \subseteq X^*$, and $\mathcal{B}$ 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 leave the state index out the operator symbol, and the relative concatenation is written as $A \otimes \mathcal{B}$. If $\mathcal{B}$ is a class attribution, then we may simply 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 3. *Let M be a FSM and $A, B \subseteq X^*$. Let $\mathcal{B}$ be a state attribution such that $\mathcal{B}(s) = B$ for all $s \in S$. Then $A \underset{s}{\otimes} \mathcal{B} = AB$.*

C State Reachability

Let M be a FSM and let $s, r \in S$ and $\rho \in X^*$. If $s \xrightarrow{\rho} r$, we say that r is reachable from s using ρ. If the word ρ is not important, we say that r is simply reachable from s. A state s is reachable if it is reachable from the initial state, that is, $s_0 \xrightarrow{\rho} s$ for some word $\rho \in X^*$. We say that machine M is connected if every state $s \in S$ is reachable. If $C \subseteq S$, we will denote by C^r the set of all states reachable from states in C.

A state r is reached from a state s using a word $\rho \in X^*$ if r is reachable from s using a prefix of ρ, that is, if we have $\rho = \rho_1\rho_2$ and $s \xrightarrow{\rho_1} r$, for some ρ_1, $\rho_2 \in X^*$. If a state r is reachable from a state s, then we can ask for a minimum length word ρ such that $s \xrightarrow{\rho} r$. Informally, the length of ρ may be thought of as the distance between s and r. The next observation, although immediate, is useful. We will use it to bound the distance between s and r by the number of states that can be reached from s using prefixes of ρ.

Observation 4. *Let M be a FSM and let r, s be states of M. Let ρ be a sequence with minimum length such that $r \xrightarrow{\rho} s$. If C is the set of states reached from r using ρ, then $|C| = |\rho| + 1$.*

The output word generated when an input word ρ is applied to a state s depends on the states reached from s using ρ, with exception of the last one. If we consider a set of words of length bounded by a value n, the machine's behavior starting at s will depend on the states within distance of at most n from s. Such states around s form a *neighborhood*, defined next.

Definition 5. *Let M be a FSM and k an integer.*

1. *The k-*radius *of a state s, denoted by $\mathrm{rad}(s, k)$, is the set of states r for which there exists a word $\rho \in X^*$ with $s \xrightarrow{\rho} r$ and $|\rho| \leq k$.*

2. *The k-*neighborhood *of a set of states C, denoted by $\mathrm{nbh}(C, k)$, is the union of the k-radiuses of states in C. That is, $\mathrm{nbh}(C, k) = \bigcup_{s \in C} \mathrm{rad}(s, k)$.*

Notice that, if k is negative, the radius and the neighborhood sets are empty.

D Cover sets

Cover sets are used in many FSM test methods in order to guarantee that every state is reached, and that every transition 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 its corresponding transitions. In this situation, only untested states must be covered, and so partial cover sets are used. Partial state cover sets and partial transition cover sets are defined next.

Definition 6. *Let M be a FSM and C be a set of states. A set $Q \subseteq X^*$ is a partial state cover set for C if, for every state $s \in C$, there exists $\rho \in Q$ such that $s_0 \xrightarrow{\rho} s$.*

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 $s_0 \xrightarrow{\rho} s$.*

Every partial transition cover set P includes a corresponding partial state cover set Q. More precisely, for each partial transition cover set P, one corresponding partial state cover set $Q \subset P$ can be obtained by choosing, for every state $s \in C$ and every symbol $a \in X$, a $\rho \in P$ such that $s_0 \xrightarrow{\rho} s$. Whenever C is the set of all states, Q and P are, in fact, a state cover set and a transition cover set, respectively, as defined in [5].

Transition cover sets may be obtained from labeled trees for M [3], and a procedure to construct labeled trees is given in [2]. Although the covers so constructed would suffice for the entire set of states, one can readily modify that procedure to obtain partial cover sets.

III State equivalence and state separators

In this section, we define state equivalence. We also introduce the essential notion of a separator. The latter will be useful when comparing state sets.

A 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. Let also $R \subseteq X^*$. We say that s is R-equivalent to s' if $\widehat{\lambda}(\rho, s) = \widehat{\lambda}'(\rho, s')$, for all $\rho \in R$. In this case, we write $s \approx_R s'$. Otherwise, s is R-distinguishable from s', and we write $s \not\approx_R s'$.*

When $R = \{\rho\}$ is a singleton, we also say that s ρ-equivalent to s', and we write $s \approx_\rho s'$. Likewise for a pair of ρ-distinguishable states and the notation $s \not\approx_\rho s'$. Further, when $R = X^*$ we may simply write $s \approx s'$ for $s \approx_{X^*} s'$, and we say that s and s' are equivalent, otherwise we say tat they are distinguishable.

If M and M' are the same machine, the definition just given can be taken as specifying equivalence relations over sets of states $C \subseteq S$. In this case, $\approx_R$ partitions 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 $[C/R] = \{\{s_0\}, \{s_1\}, \{s_2, s_3\}\}$.

Observation 9. *Let M and M' be FSMs and $A, B \subseteq S$, $C \subseteq S'$ and $R \subseteq X^*$. Then*

1. *if for every pair $r \in A, s \in B$ we have $r \not\approx_R s$, then $[(A \cup B)/R] = [A/R] \cup [B/R]$ and $|[(A \cup B)/R]| = |[A/R]| + |[B/R]|$;*
2. *if for every $r \in A$ there exists $s \in B$ such that $r \approx_R s$, then $|[(A \cup B)/R]| = |[B/R]|$;*
3. *if for every $r \in C$ there exists $s \in B$ such that $r \approx_R s$, then $|[C/R]| \leq |[B/R]|$;*
4. *if $A \subseteq B$ then $|[(B \setminus A)/R]| + |[A/R]| \geq |[B/R]|$.*

The number of pairwise distinguishable states of a FSM is called its index. Let M be a FSM and C be a set of states. The number of equivalence classes induced by $\approx$ over C is denoted by $\iota(C)$. Therefore, the index of M is $\iota(S)$. If $\iota(S) = |S|$, then machine M is said to be minimal.

B State separators

From Definition 8, we know that two states s and r are distinguishable if there exists a word γ such that $s \not\approx_\gamma r$. Whenever this happens, we say that γ separates s and r. We want to extend this notion, so that we can also separate state sets. In this case, we use a collection of input sequences instead of just one sequence. This concept is formalized next.

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, 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 $s_2 \in B, s_3 \in C$ are two distinguishable states, R is not a (B, C)-separator. Note that state s_0 is a common element of A and B.

If A and B are such that, for every $r \in A$ and $s \in B$, not necessarily distinguishable, it is the case that $r \not\approx_R s$, then R is called a *strict* (A, B)-separator. A (S, S)-separator is called a *characterization set* and, if $C \subseteq S$, then a (C, C)-separator is also called a *partial characterization set*. Additionally, any $(\{s\}, S)$-separator is called an *identification set* for s. Notice that, in this paper, we adopting a more flexible definition of characterization sets than that found in [14], where FSMs are required to be minimal for characterization sets to exist. In contrast, by our definition, any FSM has a characterization set. The same happens with respect to identification sets, as defined in [5]. We do not even require separators to be minimal. Also, it is worth noticing that, in Definition 10, sets A and B may have common elements as, clearly, is the case with characterization sets. Actually, there is no restriction to what sets of states we may select.

We also remark that a set of input sequences can assume, simultaneously, different roles for different states: it can be an identification set for s, a partial characterization set for a set of states C, and a strict separator for sets of states A, B. In Section IV-C, R is a separator that exemplifies this situation.

The concept of separators is a powerful tool that will prove instrumental when we develop our test case generation method. The next lemma points out some simple observations.

Lemma 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 take one separator and use it as a base to construct another one. With the next lemmas and corollary, we obtain a partial characterization set from a weaker special kind of separator.

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

Proof. For each pair $\hat{r}, \hat{s} \in C$ of states such that $\hat{r} \not\approx \hat{s}$ and $\hat{r} \approx_T \hat{s}$, define $\gamma_{\hat{r}\hat{s}}$ as a sequence with minimum length such that $\hat{r} \not\approx_{\gamma_{\hat{r}\hat{s}}} \hat{s}$. Choose states r and s in C such that γ_{rs} has minimum length. Note that $|\gamma_{rs}| \geq 1$ since $r \not\approx_{\gamma_{rs}} s$. Let $\gamma \in X^*$, $a \in X$ be such that $\gamma_{rs} = a\gamma$. Let r', s' be states such that $r \xrightarrow{a} r'$ and $s \xrightarrow{a} s'$. To prove the lemma, we only need to establish that $r \not\approx_{aT} s$.

First consider the case $|\gamma| = 0$. We then have $\gamma_{rs} = a$, and so $r \not\approx_a s$. Clearly, $r \not\approx_{aT} s$. Now consider the case $|\gamma| \geq 1$. Since γ_{rs} is minimal, we have $\lambda(a, r) = \lambda(a, s)$. It follows that $r' \not\approx_\gamma s'$, otherwise we would obtain the contradiction $\widehat{\lambda}(\gamma_{rs}, r) = \lambda(a, r)\widehat{\lambda}(\gamma, r') = \lambda(a, s)\widehat{\lambda}(\gamma, s') = \widehat{\lambda}(\gamma_{rs}, s)$. If either s' or r' is in $B \setminus C$, then $r' \not\approx_T s'$, because T is a $(B, B\backslash C)$-separator. Again, we get $r \not\approx_{aT} s$. Now, assume that $r', s' \in C$. If $r' \approx_T s'$, then, since $r' \not\approx_\gamma s'$, the minimal sequence that distinguishes r' and s', $\gamma_{r's'}$, has length $|\gamma_{r's'}| \leq |\gamma| < |\gamma_{rs}|$. This is a contradiction to the choice of r and s. So $r' \not\approx_T s'$, and again $r \not\approx_{aT} s$.

In any case, from $r \not\approx_{aT} s$, we have $r \not\approx_{XT} s$. This means that XT separates two states in the same class of $|[C/T]|$. Hence $|[C/(XT \cup T)]| \geq |[C/T]| + 1$. □

Suppose that using the last lemma we obtain a new separator X_1T. If there exist two distinguishable states in C that are X_1T-equivalent, then we may use the lemma again to obtain a stronger separator, X_2T. In fact, the lemma may be used several times successively, until all states in C are distinguishable. In this case, the final separator is a partial characterization set for C. The following corollary can be easily proved by induction.

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

One way of obtaining a separator for two sets of states, A and B, is by selecting the minimal subset, R, of a characterization set for an equivalent reduced FSM, such a way that R is also a (A, B)-separator. Standard methods to reduce FSMs and to obtain characterization sets are known [14]. Although those are simple procedures that can be applied to any FSM, shorter separators may result if we take into consideration the particularities of the FSM being tested.

IV Incremental testing of FSMs

Many systems are actually aggregations of other, smaller, subsystems. When modeling such systems, its usual to adopt a *building block strategy* for the development cycle, in which each subsystem is separately designed, implemented and tested. Though each individual part of the system is tested and deemed correct, we have no guarantee that the integrated final implementation is correct.

When using traditional test generation methods based on FSM models, such as the W-method [3] and the G-method [2], one is lead to ignore subsystem abstractions and, thus, proceeds to generate test suites for the entire system, as a single block. Since these methods are intended for use against untested implementations, they are inefficient in this situation. In this section, we will extend these methods to deal with combined FSMs. First, we define combined FSMs, then we present some hypotheses about the fault model, and, finally, we describe the test suite construction.

A Combined Finite State Machines

The next definition ensures that a state of a subsystem behaviors exactly in the same way, regardless whether it is considered as a state of an isolated submachine, or an incorporated state of the combined machine.

Definition 14. *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 $\dot{\lambda}(a, s) = \lambda(a, s)$.*

An easy induction immediately implies the following fact.

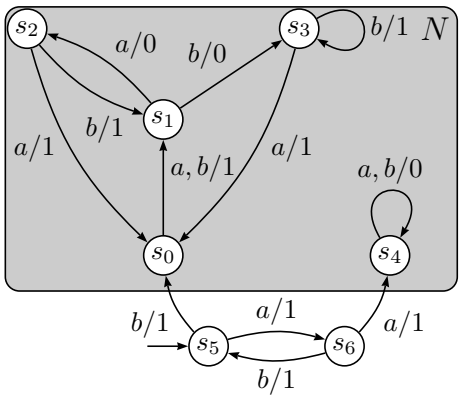

Figure 2: A Combined Finite State Machine.

Observation 15. *Let $M = (X, Y, S, s_0, \delta, \lambda)$ be a FSM and let $\dot{N} = (\dot{X}, \dot{Y}, \dot{S}, \dot{s_0}, \dot{\delta}, \dot{\lambda})$ be a submachine of M. Let also $s \in \dot{S}$ be an state of M and $\rho \in X^*$, with $s \xrightarrow{\rho} r$ and $r \in S$. Then, in fact, $r \in \dot{S}$.*

A combined FSM is formed by adding new states and transitions, and using those to conjoin one or more submachines. But, since each subsystem may have only one entry point, every transition that enters a submachine should have as its target that submachine's initial state.

Definition 16. *Let M be a FSM and N be a set of submachines of M. Define $S_N = \{s \in \dot{S} | \dot{N} \in N\}$ to be the set of all submachine states, and let $S_M = S \setminus S_N$ be the set of additional states. Also, define $I_N = \{\dot{s_0} | \dot{N} \in N\}$ as the set of all submachine initial states. Then, we say that M is N-combined if and only if $s_0 \in S_M$, and for every pair of states $s \in S_M$ and $r \in S_N$, such that $s \xrightarrow{a} r$ for some $a \in X$, we must have $r \in I_N$.*

In the sequel, we shall be using the notation introduced in Definitions 14 and 16, that is, given a machine M and a set of submachines N, we have already defined submachines $\dot{N}$ in N, and the sets S_M, S_N and I_N.

Figure 2 illustrates a combined FSM. The set of submachines, N, is formed by the machines depicted in Figure 1. We also 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 remark that this machine satisfies the properties of Definition 16. For example, for states $s_5 \in S_M$ and $s_0 \in S_N$, since $s_5 \xrightarrow{b} s_0$, we have $s_0 \in I_N$.

B The fault model

We assume that the system specification M is a combined FSM, obtained from a set N of submachines. We require that M is connected, and that for every pair of states, $s \in S_M$

and $r \in S_N$, we have $s \not\approx r$. These assumptions are reasonable, since there is no meaning in having unreachable states in the specification, or in reimplementing the behavior of an already available submachine state.

We presume that each submachine $\dot{N} \in N$ has a correct implementation $\dot{N}'$, and denote the set of submachine implementations by N'. One system's implementation M' is obtained by combining the set of submachines N' using up to m additional states. The fault model is formed by (N', m)-combined FSM candidate implementations, as defined below.

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*

1. *M' is N'-combined;*
2. *$\iota(S_M) \leq |S'_M| \leq m$;*
3. *For every $\dot{N} \in N$, there exists $\dot{N}' \in N'$ such that $\dot{s_0} \approx \dot{s_0}'$;*
4. *For every $\dot{N}' \in N'$, there exists $\dot{N} \in N$ such that $\dot{s_0} \approx \dot{s_0}'$.*

For the ease of notation, we shall write S'_M, S'_N and I'_N, instead of $S_{M'}$, $S_{N'}$ and $I_{N'}$, respectively, for the occasions where M' and N' appear as indexes.

In the following observation, the first two claims may be obtained by an easy induction using hypotheses (3) and (4) of Definition 17, respectively. The last two claims are special cases.

Observation 18. *Let M be N-combined specification and M' be a (N', m)-combined candidate implementation. Then,*

1. *if $t \in I_N$, $s \in S_N$ and $\rho \in X^*$ are such that $t \xrightarrow{\rho} s$, then there exist $t' \in I'_N$ and $s' \in S'_N$ such that $t' \xrightarrow{\rho} s'$, $t \approx t'$ and $s \approx s'$.*
2. *if $t' \in I'_N$, $s' \in S'_N$ and $\rho \in X^*$ are such that $t' \xrightarrow{\rho} s'$, then there exist $t \in I_N$ and $s \in S_N$ such that $t \xrightarrow{\rho} s$, $t' \approx t$ and $s' \approx s$.*
3. *if $s \in S^r_N$ ($s \in I_N$), then there exists $s' \in S'_N$ ($s' \in I'_N$) such that $s \approx s'$.*
4. *if $s' \in S'^r_N$ ($s' \in I'_N$), then there exists $s \in S_N$ ($s' \in I_N$) with $s' \approx s$.*

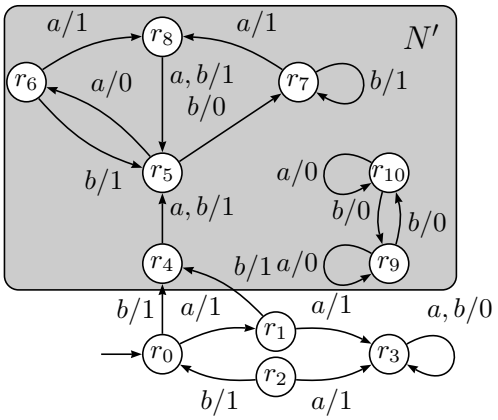

Figure 3: A candidate implementation.

Figure 3 illustrates a candidate implementation for the combined machine depicted in Figure 2. We claim that this combined machine obeys Definition 17 for $m = 4$. Observe that $2 = \iota(S_M) \leq |S'_M| \leq m = 4$. We can easily check that for each state in S_N, there exists a corresponding state in S'_N. For instance, for $s_0 \in S_N$ there is $r_4 \in S'_N$ such that $s_0 \approx r_4$.

Though we assume that the candidate implementation is (N', m)-combined, generally no other information about it is available, and we must, therefore, treat M' as a *black-box*. Observe that M' may have unreachable states, such as r_{10} in Figure 3. Also, in implementations, some of the additional states may be equivalent to submachine states, as it is the case for r_3 and r_9, respectively, in Figure 3. Further, a submachine implementation needs not be minimal. In fact, the number of states used for submachine implementation is usually greater than the number of states of the corresponding submachine specification. For example, r_9 and r_{10} are equivalent states that correspond to state s_4.

C The C-method

Now, we present a new method, named the C-method, to generate test suites for combined FSM specifications. The C-method is depicted as Algorithm 1. We expand on each step.

The Input: We start with a specification M that is a N-combined FSM, where N is a set of submachines of M. We adopt the notations exposed in Definition 16. The input parameter m is the maximum number of additional states and is defined by the user. Clearly, m must be at least as large as the number of equivalence classes of S_M, *i.e.*, $m \geq \iota(S_M)$, otherwise no correct candidate implementation would be possible.

Algorithm 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 $n \longleftarrow \max\{l', l\}$;
  if $m < n$ **then**
    mesg "No correct candidate implementation is possible" ;
  else
    Define $A \leftarrow \text{nbh}(I_N, m-n-1)$;
    Obtain a (A, S_N)-separator T ;
    Let $\mathcal{R}(S_M) \longleftarrow R$ and $\mathcal{R}(S_N) \longleftarrow R \cup T$;
    forall $s \in S$ **do**
      Define $\mathcal{Z}(s) \longleftarrow X_{m-n} \underset{s}{\otimes} \mathcal{R}$;
    Define $\pi \longleftarrow P \otimes \mathcal{Z}$;
    return π
end

The Cover Set P**:** P is a partial transition cover set for S_M such that $\varepsilon \in P$. 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 takes on several roles, depending on the states we are considering. For example, if we consider R as a strict (S_M, S_N)-separator, we can use R to distinguish additional states from submachine states. As a (I_N, S_N)-separator, R can be used to uniquely identify initial states of submachines, and so on.

The Parameter n**:** The relation $\approx_R$ partitions the states of M. Based on this, we let $l = |[S/R]| - |[S_N/R]|$. Similarly, the relation $\approx_R$ also partitions the states of M'. We now *choose* a value l' such that $l' \leq |[S'/R]| - |[S'_N/R]|$. If no information about M' is available, we can always choose $l' = 0$. We then set $n = \max\{l, l'\}$. The larger n is, the less we need to strengthen R in order to obtain a partial characterization set for S'_M. Here, we can use G-method [10] ideas, so that, if knowledge is available about the implementation, l' may be set larger than l, thus giving rise to more succinct test suites. We notice that we will always have $m \geq n$, otherwise the input

m is not valid. If this is the case, the algorithm halts and issues a message to the user.

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

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

THE TEST SUITE π**:** Finally, the test suite generated by the C-method is given by $\pi = P \otimes \mathcal{Z}$.

V Comparison with more traditional approaches

We compare our incremental testing approach with a traditional one. More specifically, we compare the C-method against the W-method. First, we define the efficiency of a test suite, and use this concept to compare test suites generated for the specification depicted in Figure 2, using both the W-method and the C-method. We conclude this section showing that the C-method outperforms the W-method in an infinite family of naturally occurring FSMs.

A Test suite efficiency

Consider the input alphabet X, an let $\pi \subseteq X^*$ be a test suite over X. In order to test an implementation against a specification, one needs to apply each sequence of π, and compare the outputs of both FSMs. Of course, it is not necessary to apply any proper prefix of a word in π, since testing over a longer sequence will already reveal behaviors over any shorter prefix [10] of that sequence.

Definition 19. *Let X be an alphabet, and let $V \subseteq X^*$. Define* $\text{pref}(V)$ *as the set of all prefixes of V,* i.e., $\text{pref}(V) = \{\varphi \in X^* \mid \varphi\vartheta \in V \text{ for some } \vartheta \in X^*\}$. *Also, define* $\text{pff}(V)$ *as the set of prefix-free elements of V,* i.e., $\text{pff}(V) = \{\varphi \in V \mid \varphi\vartheta \notin V \text{ for all } \vartheta \in X^* \text{ with } \vartheta \neq \varepsilon\}$.

The following observations are immediate.

Observation 20. *Let X be an alphabet, and let $U, V \subseteq X^*$. Then,*

1. $\text{pff}(U \cup V) = \text{pff}(\text{pff}(U) \cup \text{pff}(V))$.
2. *if* $\text{pff}(U) \cap \text{pref}(V) = \emptyset$, *then* $\text{pff}(U) \subseteq \text{pff}(U \cup V)$.
3. *if* $U \subseteq \text{pref}(V)$, *then* $\text{pff}(U \cup V) = \text{pff}(V)$.
4. $\text{pff}(U \cup V) \subseteq \text{pff}(U) \cup \text{pff}(V)$.

The efficiency of a test suite π depends on the length of its test cases [8], and it is usually measured as the sum of the lengths of all elements in $\text{pff}(\pi)$ [10].

Definition 21. *Let X be an alphabet, and $V \subseteq X^*$. The efficiency of V is $||V|| = \sum_{\sigma \in \text{pff}(V)} |\sigma|$.*

B The W-method: a brief overview

The W-method [3] applies to minimal, completely specified and deterministic FSMs. So, to apply the W-method to a specification that is not minimal, one must first reduce it. Let n_W be the number of states in the specification. The set of implementation candidates comprehends all faulty machines with up to m_W states, with m_W chosen to satisfy $n_W \leq m_W$. Test suites for this fault model are shown to be m_W-complete.

The method depends on two sets, P_W and W. The first is a transition cover set, and the second is a characterization set for the specification machine. The test suite generated by the W-method is then is $\pi_W = P_W Z_W$, where $Z_W = X_{m_W - n_W} W$.

C An Example

We will generate test suites for the specification depicted in Figure 2 and the candidate implementation shown in Figure 3.

Using the W-method: The specification 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 construct a transition cover set P_W, for example using a labeled tree [10], then we choose a minimal characterization set W, and, finally, the set Z_W is computed:

- $P_W = \{\varepsilon, a, b, aa, ab, aaa, aab, ba, bb, baa, bab, baaa, baab, baba, babb\}$;
- $W = \{aaaa, bb\}$;
- $Z_W = X_{m_W - n_W} W = X_4 W = (X^0 \cup X^1 \cup X^2 \cup X^3 \cup X^4)W$.

The test suite generated by the W-method is then $\pi_W = P_W Z_W$, with efficiency $||\pi_W|| = 2624$.

Using the C-method: The test suites will be generated for (N', m)-combined candidate implementations, with $m = 4$ and where N' is illustrated in Figure 3. We first select P as the 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 obtained 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 partition the states of the specification and obtain the value l, and then, since no specific information is available about the implementation, we assume the worst case and set $l' = 0$. From those two values we obtain a value for n. Proceeding, we define A as a $(m{-}n{-}1)$-neighborhood of I_N, and then we select a (A, S_N)-separator T from W. Finally, we calculate the state attribution $\mathcal{Z}$. Following these steps, we have:

- $P = \{\varepsilon, a, b, aa, ab, aaa, aab, ba, bb\}$;
- $R = \{aaaa\}$;
- $[S/R] = \{\{s_0\}, \{s_1\}, \{s_2, s_3\}, \{s_4\}, \{s_5\}, \{s_6\}\}$;
- $[S_N/R] = \{\{s_0\}, \{s_1\}, \{s_2, s_3\}, \{s_4\}\}$;
- $l' = 0$, $l = |[S/R]| - |[S_N/R]| = 2$, 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 $\mathcal{R}(S_M) = R$;
- $\mathcal{Z}(s) = X_{m-n} \underset{s}{\otimes} \mathcal{R} = X_{m-n}R = X_2 R$, for every $s \in S$.

The test suite generated by the C-method is then $\pi = P \otimes \mathcal{Z} = PX_2R$, with an efficiency $||\pi|| = 168$.

Additionally, using the same C-method strategy, we can obtain a test suite π' for the submachines in Figure 2, with $||\pi'|| = 120$. We thus conclude that we can use the C-method to test the entire specification using the combined suite $\pi \cup \pi'$, with an efficiency of $||\pi \cup \pi'|| \leq 288$, a considerable improvement over the result generated by the W-method.

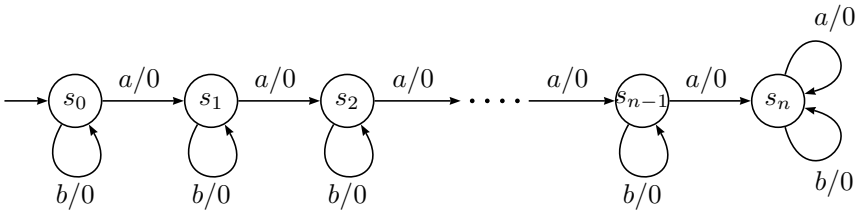

Figure 4: Specification Model M_n.

D Special families of FSMs

Now we want to compare the C-method strategy against the W-method strategy over infinite families of FSMs. We will show that, if the models are abstracted as combined FSMs, then the C-method can generate test suites significantly more efficient than those generated by the W-method.

Consider a FSM, M_n, depicted in Figure 4, where $n \geq 0$ is an integer, and the input alphabet is $X = \{a, b\}$. It is easy to see that M_n is minimal, $|S| = n + 1$, and that $W = \{a^{n+1}\}$ is a characterization set with $||W||$ minimum. Also, we can construct the transition cover set $P_W = \{\varepsilon\} \cup \{a^i a, a^i b | 0 \leq i \leq n\}$.

Now, consider a candidate implementation, M'_n. We assume that the number of states of the implementation is bounded by a factor, α, of the number of states of the specification, that is, we take $m+1 = \lceil \alpha \cdot (n+1) \rceil$, and assume that the number of implementation states is $|S'| \leq m + 1$.

1 Using the W-method

Using W-method, we obtain a test suite π_W by letting $Z_W = X_{m-n}W$ and setting $\pi_W = P_W Z_W$. To obtain $\text{pff}(\pi_W)$, we first expand the concatenation

$$\begin{aligned} \pi_W = P_W Z_W &= \Big(\{\varepsilon\} \cup \bigcup_{i=0}^{n} a^i a \cup \bigcup_{i=0}^{n} a^i b\Big) X_{m-n} a^{n+1} \\ &= X_{m-n}a^{n+1} \cup \bigcup_{i=0}^{n} a^i a X_{m-n} a^{n+1} \cup \bigcup_{i=0}^{n} a^i b X_{m-n} a^{n+1}. \end{aligned} \tag{1}$$

Now we show that $X_{m-n}a^{n+1}$ is contained in the prefixes of the unions in (1).

Case 1: If $m - n = 0$, then $X_0 = \{\varepsilon\}$, and so

$$X_{m-n}a^{n+1} = a^{n+1} \subseteq \text{pref}(aa^{n+1}) = \text{pref}(a^0 a X_{m-n} a^{n+1}).$$

Case 2: If $m - n > 0$, then

$$\begin{aligned} X_{m-n}a^{n+1} &= XX_{m-n-1}a^{n+1} \\ &= aX_{m-n-1}a^{n+1} \ \cup \ bX_{m-n-1}a^{n+1} \\ &\subseteq \text{pref}(a^0aX_{m-n}a^{n+1} \ \cup \ a^0bX_{m-n}a^{n+1}). \end{aligned}$$

We conclude, applying Observation 20(3) to (1), that

$$\text{pff}(\pi_W) = \text{pff}(\bigcup_{i=0}^{n} a^iaX_{m-n}a^{n+1} \ \cup \ \bigcup_{i=0}^{n} a^ibX_{m-n}a^{n+1}). \tag{2}$$

Consider the first union of in (2) and fix a value for i. We will show that $\text{pff}(a^iaX_{m-n}a^{n+1}) = \text{pff}(a^iaX^{m-n}a^{n+1})$. Recall that $X_{m-n} = \cup_{j=0}^{m-n} X^j$. Then, using Observation 20(3), it suffices to show that $a^iaX^ja^{n+1} \subseteq \text{pref}(a^iaX^{m-n}a^{n+1})$, for all $0 \leq j \leq m-n$. Fix some j and take $\beta \in X^j$. We show that $a^ia\beta a^{n+1} \in \text{pref}(a^iaX^{m-n}a^{n+1})$. Define $\gamma = a^{(m-n)-j}$. Since $X = \{a, b\}$, we get $\beta\gamma \in X^{m-n}$ and so $a^ia\beta\gamma a^{n+1} \in \text{pref}(a^iaX^{m-n}a^{n+1})$. But $a^ia\beta\gamma a^{n+1} = a^ia\beta a^{n+1}\gamma$ and so $a^ia\beta a^{n+1} \in \text{pref}(a^iaX^{m-n}a^{n+1})$, as desired.

Similarly, we also get $\text{pff}(a^ibX_{m-n}a^{n+1}) = \text{pff}(a^ibX^{m-n}a^{n+1})$. Using these facts and Observation 20(1), from (2) we may write

$$\text{pff}(\pi_W) = \text{pff}\left(\text{pff}(\bigcup_{i=0}^{n} a^iaX^{m-n}a^{n+1}) \ \cup \ \text{pff}(\bigcup_{i=0}^{n} a^ibX^{m-n}a^{n+1})\right). \tag{3}$$

We expand the first term in (3). If $m - n = 0$ then, clearly,

$$\text{pff}(\bigcup_{i=0}^{n} a^iaX^{m-n}a^{n+1}) = a^naX^{m-n}a^{n+1}. \tag{4-a}$$

Now, let $m - n > 0$. We get

$$\begin{aligned} \bigcup_{i=0}^{n} a^iaX^{m-n}a^{n+1} &= a^0aX^{m-n}a^{n+1} \ \cup \ \bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1} \\ &= a^0aXX^{m-n-1}a^{n+1} \ \cup \ \bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1} \\ &= a^0aaX^{m-n-1}a^{n+1} \ \cup \ a^0abX^{m-n-1}a^{n+1} \ \cup \ \bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1}. \end{aligned}$$

Since $a^0aaX^{m-n-1}a^{n+1} \subseteq \text{pref}(\bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1})$, we obtain, using Observation 20(3),

$$\text{pff}(\bigcup_{i=0}^{n} a^iaX^{m-n}a^{n+1}) = \text{pff}(a^0abX^{m-n-1}a^{n+1} \ \cup \ \bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1}).$$

But it is easy to see that $\text{pff}(a^0abX^{m-n-1}a^{n+1}) \cap \text{pref}(\bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1}) = \emptyset$, and also that $\text{pff}(\bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1}) \cap \text{pref}(a^0abX^{m-n-1}a^{n+1}) = \emptyset$. Using Observation 20(2) twice, together with Observation 20(4), we get

$$\begin{aligned}\text{pff}(\bigcup_{i=0}^{n} a^iaX^{m-n}a^{n+1}) &= \text{pff}(a^0abX^{m-n-1}a^{n+1}) \ \cup \ \text{pff}(\bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1}) \\ &= a^0abX^{m-n-1}a^{n+1} \ \cup \ \text{pff}(\bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1}).\end{aligned}$$

Repeating the previous argument $n-1$ more times, we obtain

$$\begin{aligned}\text{pff}(\bigcup_{i=0}^{n} a^iaX^{m-n}a^{n+1}) &= \bigcup_{i=0}^{n-1} a^iabX^{m-n-1}a^{n+1} \ \cup \ \text{pff}(a^naX^{m-n}a^{n+1}) \\ &= \bigcup_{i=1}^{n} a^ibX^{m-n-1}a^{n+1} \ \cup \ a^naX^{m-n}a^{n+1}. \qquad \text{(4-b)}\end{aligned}$$

We can also see that $\cup_{i=0}^{n} a^ibX^{m-n}a^{n+1}$ is prefix-free. For, if not, we would get $\alpha, \alpha\beta \in \cup_{i=0}^{n} a^ibX^{m-n}a^{n+1}$ with $\beta \neq \varepsilon$. This gives $\alpha = a^jb\gamma_1a^{n+1}$ and $\alpha\beta = a^kb\gamma_2a^{n+1}$, with $|\gamma_1| = |\gamma_2| = m-n$ and $0 \leq j, k \leq n$. Then $a^jb\gamma_1a^{n+1}\beta = a^kb\gamma_2a^{n+1}$ and so, clearly, we must have $k = j$ and $\gamma_1 = \gamma_2$. This gives $a^{n+1}\beta = a^{n+1}$, and so $\beta = \varepsilon$, a contradiction. We can now write

$$\text{pff}\Big(\bigcup_{i=0}^{n} a^ibX^{m-n}a^{n+1}\Big) = \bigcup_{i=1}^{n} a^iaX^{m-n}a^{n+1}. \qquad \text{(4-c)}$$

Replacing (4-b) and (4-c) in (3), we get

$$\text{pff}(\pi_W) = \text{pff}\left(\bigcup_{i=1}^{n} a^ibX^{m-n-1}a^{n+1} \ \cup \ a^naX^{m-n}a^{n+1} \ \cup \ \Big(\bigcup_{i=0}^{n} a^ibX^{m-n}a^{n+1}\Big)\right).$$

Now, let $\beta \in X^{m-n-1}$. Then, $a^ib\beta aa^{n+1} \in a^ibX^{m-n}a^{n+1}$ and, since $a^ib\beta aa^{n+1} = a^ib\beta a^{n+1}a$, we conclude that $a^ib\beta a^{n+1} \in \text{pref}(a^ibX^{m-n}a^{n+1})$. Thus,

$$\bigcup_{i=1}^{n} a^ibX^{m-n-1}a^{n+1} \subseteq \text{pref}(\bigcup_{i=0}^{n} a^ibX^{m-n}a^{n+1}).$$

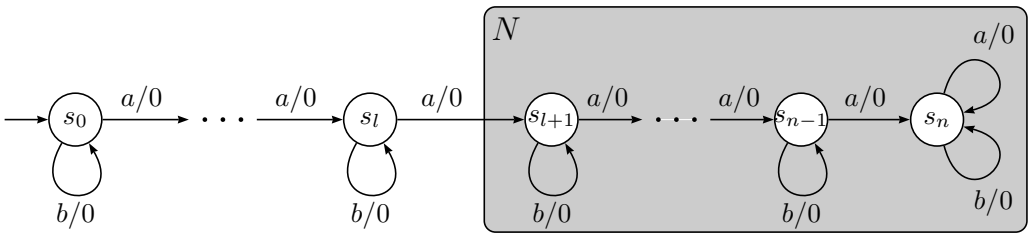

Figure 5: Combined Abstraction of specification Model M_n.

Using Observation 20(3) we then get

$$\text{pff}(\pi_W) = \text{pff}\left(a^n a X^{m-n} a^{n+1} \cup \Big(\bigcup_{i=0}^{n} a^i b X^{m-n} a^{n+1}\Big)\right). \tag{4-d}$$

It is easy to see that we can not have $\alpha, \alpha\beta \in \Big(a^n a X^{m-n} a^{n+1} \cup \ (\cup_{i=0}^{n} a^i b X^{m-n} a^{n+1})\Big)$, with $\beta \neq \varepsilon$. That is, $a^n a X^{m-n} a^{n+1} \cup \ (\cup_{i=0}^{n} a^i b X^{m-n} a^{n+1})$ is prefix-free. Thus, from (4-d) we can write

$$\text{pff}(\pi_W) = a^n a X^{m-n} a^{n+1} \ \cup \ \bigcup_{i=0}^{n} a^i b X^{m-n} a^{n+1}. \tag{5}$$

Now, we may calculate $||\pi_W|| = (m+n+2)\cdot 2^{m-n} + \sum_{i=0}^{n}(i+m+2)\cdot 2^{m-n}$, and this is easily seen to be[1] $\Theta(n(n+m)\cdot 2^{m-n})$.

Since $m+1 = \lceil \alpha\cdot(n+1)\rceil$, we conclude that the cost of testing M_n using the W-method is

$$C_w(n) = ||\pi_W|| = \Omega(n^2 2^{n(\alpha-1)}), \tag{6}$$

that is, $n^2 2^{n(\alpha-1)}$ is a lower bound on the size of the test when the W-method is applied to machine M_n.

2 Using the C-method

Now, we will adopt an incremental testing strategy for independently implementing and testing the submachines of M_n. Consider the FSM depicted in Figure 5, where l is an integer and $0 \leq l \leq n$. Notice that it is exactly the same FSM illustrated in Figure 4, except that now it is abstracted as a combined FSM. For the machine in Figure 5, we have $S_M = \{s_0, ..., s_l\}$, $S_N = \{s_{l+1}, ..., s_n\}$. We extract the R and T separators from the

[1] In the sequel, we will be using the asymptotic notations Θ, Ω and O; for details, see any good text on algorithm complexity.

same characterization set W and obtain $R = T = W = \{a^{n+1}\}$. A partial transition cover set for S_M is $P_C = \{\varepsilon\} \cup \{a^i a, a^i b | 0 \leq i \leq l\}$. The implementation M'_n is also a combined FSM. Since its number of states, $|S'|$, is bounded by the factor α, we assume the same holds for the set of additional states, $|S'_M|$, and the set of submachine states, $|S'_N|$. In face of that, we choose k such that $k + 1 = \lfloor \alpha(l + 1) \rfloor$, and we let $|S'_M| \leq k + 1$.

Finally, we obtain a test suite π_C by letting $Z_C = X_{k-l}W$ and setting $\pi_C = P_C Z_C$. By a completely analogous reasoning that gave us $\text{pff}(\pi_W)$ in (5), we can obtain

$$\text{pff}(\pi_C) = a^l a X^{k-l} a^{n+1} \ \cup \ \bigcup_{i=0}^{l} a^i b X^{k-l} a^{n+1}. \tag{8}$$

Therefore $||\pi_C|| = (k + n + 2) \cdot 2^{k-l} + \sum_{i=0}^{l} (i + k - l + n + 2) \cdot 2^{k-l}$. We now show that we can make $||\pi_C||$ grow asymptotically as $O(n^2 2^{n(\alpha-1)/2})$. It is easy to see that $||\pi_C|| \leq (k + n + 2)(l + 2)2^{k-l}$. Choose $l = \lceil n/2 \rceil$, so that $n/2 \leq l \leq n/2 + 1$. Let $n \geq 6$. Then $l + 2 \leq n/2 + 3 \leq n$. Recalling that $k + 1 \leq \alpha(l + 1)$, we get $k + 1 \leq \alpha(n/2 + 2) \leq \alpha n$ and so $k + n + 2 \leq \alpha n + n + 2 \leq n(\alpha + 2)$. Also, $k - l \leq \alpha(l + 1) - l = l(\alpha - 1) + \alpha \leq n/2(\alpha - 1) + (2\alpha - 1)$, and we get $2^{k-l} \leq 2^{2\alpha-1} 2^{n(\alpha-1)/2}$. Putting it together, we get $||\pi_C|| \leq K n^2 2^{n(\alpha-1)/2}$, where $K = (\alpha + 2)2^{2\alpha-1}$ is a constant, concluding the argument.

We assumed that the submachine was already tested. In order to obtain $C_c(n)$, the cost of testing M_n in its entirety using the C-method, we must also consider the cost of testing the submachine. A submachine can be itself a combined FSM, and so it can, of course, also be tested using the C-method, giving rise to a recursive testing approach. Notice that, if $n \geq 1$, then the submachine corresponds to the model M_{n-l-1}. We then get

$$C_c(n) = \begin{cases} ||\pi_C|| + C_c(n - l - 1) = O(n^2 2^{n(\alpha-1)/2}) + C_c(\lfloor \frac{n}{2} \rfloor) & \text{if } n \geq 6, \\ ||\pi_C|| = O(1) & \text{otherwise.} \end{cases}$$

Solving the recurrence, we get $C_c(n) = O(n^2 2^{n(\alpha-1)/2} \log n)$.

Now, consider the ratio $Q_\alpha(n) = \frac{C_w(n)}{C_c(n)}$. From (6) and (8), we readily obtain $Q_\alpha(n) = \Omega(2^{cn})$, where $c = \frac{\alpha' - 1}{2}$ and provided that $\alpha' < \alpha$. We conclude that, for the family of specifications M_n and corresponding implementations M'_n, $n \geq 0$, we may use the C-method to generate test suites exponentially more succinct than the W-method, as n grows.

We note that the gains of the C-method expressed for this family of FSMs may, potentially, occur in other models, that is, a combined FSM may contain the specification model M_n as a submachine. Also, notice that the model specification M_n may be much more flexible. For example, the results still hold if we change the target state of any

additional state to another additional state. In either case, the advantage of incremental testing using the C-method is preserved, and so the test suites generated would still be exponentially more succinct than those obtained with the W-method, asymptotically.

VI Correctness of the C-method

We prove that the C-method generates a complete test suite. Throughout this section, let the specification M be a connected N-combined FSM, where N is a set of submachines of M. We assume that for every pair of states $s \in S_M$ and $r \in S_N$ we have $s \not\approx r$. Also, let M' be a (N', m)-combined candidate implementation, where N' is a set of submachines of M'.

We generate a test suite π using Algorithm 1, at page 76, giving M and m as input. In the following, we refer to P, R, l', l, n, A, T, $\mathcal{R}$ and $\mathcal{Z}$ as defined at that algorithm. We denote by Q the minimal partial state cover set associated to P with $\varepsilon \in Q$. Also, we let $Z = X_{m-n}R$, and $A' = \mathrm{nbh}(I'_N, m{-}n{-}1)$.

The proof is divided in three parts. First, we obtain some auxiliary results. Then, we use the concept of separators to show that Z is a partial characterization set of S'^r_M, and, finally, we show that π is a complete test suite. Although we may omit proofs for the simpler results, detailed arguments for all claims can be found in [15].

A Preliminary results

We need to ensure that the Algorithm 1 always returns a test suite π. It is enough to verify the assertion $m \geq n$. The next lemma states that $m \geq n$ holds, unless the user has provided a value of m too small. In this case, no correct candidate implementation is possible, and the algorithm halts yielding an appropriate message.

Lemma 22. *If $m < n$, then $m < \iota(S_M)$, and so $s_0 \not\approx s'_0$.*

A hierarchy appearing in the computation of test suite is captured the following observation. For example, if we have $s_0 \approx_\pi s'_0$, then we also get $s_0 \approx_{PZ} s'_0$. This is observation is easily derived from the Algorithm 1, and we may used it without further explicit mention.

Observation 23. *We always have:*

1. *$QZ \subseteq PZ \subseteq P \otimes \mathcal{Z} = \pi$, and*
2. *for all $s \in S$, $R \subseteq Z \subseteq \mathcal{Z}(s)$.*

The following auxiliary lemma is used to decide whether a reachable implementation state is an additional state, or is a submachine state.

Lemma 24. *Let $s \in S_M$, $s' \in S'^r$ be such that $s \approx_R s'$. Then $s' \in S'_M$.*

Proof. Suppose, for the sake of contradiction, that $s' \in S'_N$. Then $s' \in S'^r_N$, and, by Observation 18(4), there exists $r \in S_N$ such that $r \approx s'$. In particular, $r \approx_R s'$, and then $s \approx_R r$. Since R is a strict (S_M, S_N)-separator and $r \in S_N$, from Lemma 11(3) we get $s \notin S_M$. This contradicts the hypothesis, so $s' \in S'_M$. □

We know that T is as a (S_N, A)-separator and R is a (S_N, I_N)-separator. The next lemma states that they are also separators for the reachable states of implementation analogous sets S'_N, A' and I'_N.

Lemma 25. *Let S'^{ri}_N be the set of submachine states reachable from I'_N, that is, $s' \in S'^{ri}_N$ if and only if there exist $\rho \in X^*$, $t' \in I'_N$ such that $t' \xrightarrow{\rho} s'$. Then,*

1. *T is a (S'^{ri}_N, A')-separator,*
2. *R is a (S'^{ri}_N, I'_N)-separator,*
3. *T is a (S'^{r}_N, A')-separator, and*
4. *R is a (S'^{r}_N, I'_N)-separator.*

Proof. For the first claim, if R were not a (S'^{ri}_N, I'_N)-separator, then we would get distinguishable states $s' \in S'^{ri}_N$ and $r' \in I'_N$ with $s' \approx_R r'$. But, by Observation 18(2), we then get $s \in S_N$ with $s \approx s'$, and, by Observation 18(4), we also get $r \in I_N$ with $r \approx r'$. This gives $s \approx_R r$, and since R is a (I_N, S_N)-separator, $s \approx r$, but then, $s' \approx r'$, that is a contradiction.

Proofs for the other claims are similar. □

Next, we specify four subsubsets of S'_M.

Definition 26. *Let $I, J, K, L \subseteq S'_M$ be as follows. For any $s' \in S'_M$:*

1. *If there is $s \in S_M$ such that $s' \approx_R s$. Then:*
   (a) *If there is $\rho \in Q$ with $s_0 \xrightarrow{\rho} s$, $s'_0 \xrightarrow{\rho} s'$ and $s' \approx_Z s$. Then, put s' in I.*
   (b) *Else, put s' in J.*
2. *If there is $s \in S_N$ such that $s' \approx_R s$. Then, put s' in K.*
3. *Else,* i.e., *$s' \not\approx_R s$ for all $s \in S$. Then, put s' in L.*

Throughout this section, I, J, K and L will always refer to the four sets in Definition 26. The following observations are immediate.

Observation 27. *We have:*

1) They are two by two disjoint, and 2) $I \cup J \cup K \cup L = S'_M$.

Observation 28. *If* $s_0 \approx_\pi s'_0$, *then* $s'_0 \in I$.

In the sequel, sets I, J, K and L will be used to estimate the lengths of certain distinguishing sequences. In order to do that, we need bounds on their sizes.

Lemma 29. *If* $s_0 \approx_\pi s'_0$, *then the following inequalities hold:*

1) $|I| \geq l$; *2)* $|L| \geq n - l$; *3)* $|J| + |K| \leq m - n$; *4)*$|[S'_M / R]| \geq n$.

Proof. We will prove Claims 1 and 2, the others are obtained from the first two Claims.

Claim 1: $|I| \geq l$.
Let $s \in S_M$. Since Q covers all states of S_M, there is $\rho \in Q$ such that $s_0 \xrightarrow{\rho} s$. Take $s' \in S'$ such that $s_0 \xrightarrow{\rho} s'$. From $s_0 \approx_\pi s'_0$, we obtain $s_0 \approx_{QZ} s'_0$, and then, from $\rho \in Q$, we get $s_0 \approx_{\rho Z} s'_0$. It follows that $s \approx_Z s'$, and, in particular, $s \approx_R s'$. Using Lemma 24, we get $s' \in S'_M$, and then, clearly, $s' \in I$.

We have shown that for each $s \in S_M$ there is a corresponding state $s' \in I$ such that $s \approx_R s'$. Clearly, $|[S/R]| - |[S_N/R]|$ counts only the classes that are entirely contained in S_M, hence $l \leq |[S_M/R]|$. Using 9(3), we have $l \leq |[S_M/R]| \leq |[I/R]| \leq |I|$.

Claim 2: $|L| \geq n - l$.
If $n = l$, then clearly the claim holds. Let now $n = l'$. We have the following.

$$
\begin{aligned}
l &= |[S/R]| - |[S_N/R]| = |[S_N \cup S_M/R]| - |[S_N/R]| && (1)\\
&= |[S_N/R]| + |[S_M/R]| - |[S_N/R]| = |[S_M/R]|, && (2)\\
|L| &\geq |[L/R]| = |[L/R]| + |[S_M/R]| - l && (3)\\
&\geq |[L/R]| + |[I \cup J/R]| - l && (4)\\
&= |[I \cup J \cup L/R]| - l && (5)\\
&= |[S'_M \setminus K/R]| - l && (6)\\
&\geq |[S'/R]| - |[S'_N \cup K/R]| - l && (7)\\
&= |[S'/R]| - |[S'_N/R]| - l && (8)\\
&= l' - l = n - l. && (9)
\end{aligned}
$$

Detailed justifications for each step above are as follows:

1. From l definition and, from Definition 16, $S = S_M \cup S_N$.

2. Because R is a strict (S_M, S_N)-separator, and using Observation 9(1).

3. The inequality is obvious. For the equality, note that item (2) implies $l = |[S_M/R]|$.

4. From Observation 9(3) and Definition 26(1), we immediately get $|[I \cup J/R]| \leq |[S_M/R]|$.

5. Let $r' \in I \cup J$ and $s' \in L$. From Definition 26(1), there exists $t \in S_M$ such that $r' \approx_R t$. Suppose that $s' \approx_R r'$, then $s' \approx_R t$. But, since $t \in S$, from Definition 26(3), $s' \notin L$. This is a contradiction, so $r' \not\approx_R s'$. Now, we may use Observation 9(3).

6. Immediate from Observation 27(1,2).

7. Immediate from Observation 9(4).

8. Let $r' \in K$. Then, from Definition 26(3), there exist $s \in S_N$ such that $r' \approx_R s$. From Observation 18(3) and since $S_N = S_N^r$, there exists $s' \in S'_N$, such that $s \approx_R s'$, and so $r' \approx_R s'$. Now, we may use Observation 9(2).

9. This is from l' definition. □

The next lemma says that, in the implementation, if a given additional state and a submachine initial state are R-equivalent, then the concatenation of a certain sequence with $R \cup T$ can distinguish them.

Lemma 30. *Let $s \in K^r$ and $r \in I'_N$ be two distinguishable states of M'. If $s_0 \approx_\pi s'_0$, then there exists an input sequence β that satisfies:*

1. *Either $s \not\approx_\beta r$, or $\widehat{\delta}'(\beta, s) \not\approx_{T \cup R} \widehat{\delta}'(\beta, r)$, and*

2. *With the exception of $\widehat{\delta}'(\beta, s)$, the states reached using β starting at s are all distinct and belong to K.*

Proof. Since s and r are distinguishable, there exists a sequence that conforms to the first requirement. So, let α be a minimal sequence satisfying the first property. If $\alpha = \varepsilon$, then α reaches only $s \in K$. In this case, α also satisfies the second property and we are done.

Now, assume that $\alpha \neq \varepsilon$, and let $\alpha = \alpha' a$, where $\alpha' \in X^*$ and $a \in X$. Let $s_1, s_2 \in S'$ be such that $s \xrightarrow{\alpha'} s_1 \xrightarrow{a} s_2$. From Observation 15, we also have $r_1, r_2 \in S'_N$ with $r \xrightarrow{\alpha'} r_1 \xrightarrow{a} r_2$. See Figure 6.

Claim 1: $s_1 \in S'_M$.
Suppose, for the sake of contradiction, that $s_1 \in S'_N$. Let $s_N \in I'_N$ be the first state in S'_N reached using α' starting at s, and let $\alpha = \alpha_M \alpha_N$ with $s \xrightarrow{\alpha_M} s_N$. From Observation 15,

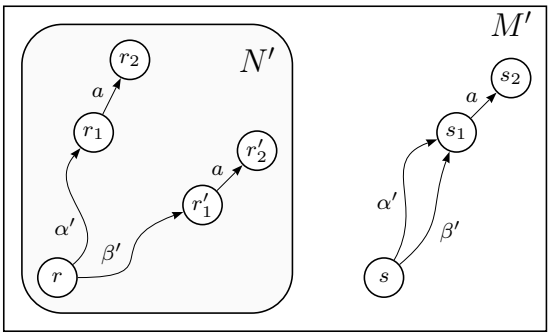

Figure 6: Lemma 30 illustration

applying α_M starting at r, we reach some $r_N \in S'_N$ such that $r \xrightarrow{\alpha_M} r_N$. Let S'^{ri}_N as in Lemma 25. Since $r \in I_N$, we get $r_N \in S'^{ri}_N$. Clearly, $|\alpha_M| \leq |\alpha'| < |\alpha|$ and so, from the minimality of α, we conclude that $s_N \approx_{T \cup R} r_N$ and $s \approx_{\alpha_M} r$. Therefore, $s_N \approx_R r_N$. Recalling, we have $s_N \in I'_N$, $r_N \in S'^{ri}_N$. Since, from Lemma 25(2), R is a (I'_N, S'^{ri}_N)-separator, we get $s_N \approx r_N$. So, $s_2 \approx_{T \cup R} r_2$. Since $s_2 = \widehat{\delta}'(\alpha, s)$ and $r_2 = \widehat{\delta}'(\alpha, r)$, we may write $\widehat{\delta}'(\alpha, s) \approx_{T \cup R} \widehat{\delta}'(\alpha, r)$. Because α satisfies the first property, we conclude that $s \not\approx_\alpha r$, and so we must have $\widehat{\lambda}'(\alpha, s) \neq \widehat{\lambda}'(\alpha, r)$. But

$$\widehat{\lambda}'(\alpha, s) = \widehat{\lambda}'(\alpha_M, s)\widehat{\lambda}'(\alpha_N, s_N) = \widehat{\lambda}'(\alpha_M, r)\widehat{\lambda}'(\alpha_N, r_N) = \widehat{\lambda}'(\alpha, r).$$

This contradiction establishes the Claim 1.

Let now β' be a sequence with minimum length such that $s \xrightarrow{\beta'} s_1$. We will show that $\beta = \beta' a$ satisfies both desired properties.

Claim 2: β satisfies property (2).
Let β'' be a prefix of β', then $|\beta''| \leq |\beta'| \leq |\alpha'| < |\alpha|$. Let states r' and s' be such that $s \xrightarrow{\beta''} s'$ and $r \xrightarrow{\beta''} r'$. From Observation 15, we get $r' \in S'_N$. Also, we have $s' \in S_M$, otherwise we would get $s_1 \in S'_N$, using Observation 15 again. By the minimality of α, we get $s' \approx_{T \cup R} r'$, and so $s' \approx_R r'$. From $r \xrightarrow{\beta''} r'$ and $r \in I'_N$, using Observation 18(2), we get $u' \in S_N$ such that $u' \approx r'$, so $s' \approx_R u'$. Since $s' \in S'_M$, $u' \in S_N$ and $s' \approx_R u'$, from Definition 26(2), we have $s' \in K$. We have thus shown that all states reached from s using β' are in K. Since β' is minimal, they are also all distinct. So, given that $\beta = \beta' a$, we conclude that β satisfies the second property.

Claim 3: β satisfies property (1).
Let r'_1 and r'_2 be states such that $r \xrightarrow{\beta'} r'_1 \xrightarrow{a} r'_2$. Recall Figure 6. From Claim 2, we know that, starting in s, β' reaches only distinct states in K. Using Lemma 29(3), we may write $|\beta'| \leq |K| - 1 \leq |K| + |J| - 1 \leq m - n - 1$. Since $r \in I'_N$, it follows that r'_1 is in the

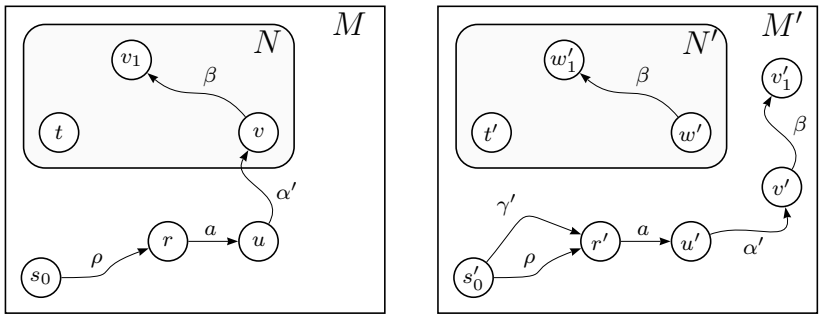

Figure 7: Lemma 31 illustration

$(m-n-1)$-neighborhood of I'_N, that is, $r'_1 \in A'$. Also, since $r \in I'_N$, and $r \xrightarrow{\alpha'} r_1$, we have $r_1 \in S'^{ri}_N$. Since $|\beta'| < |\alpha|$ and $|\alpha'| < |\alpha|$, by the minimality of α again, $r'_1 \approx_{T \cup R} s_1$ and $r_1 \approx_{T \cup R} s_1$. Hence, $r'_1 \approx_{T \cup R} r_1$, with $r'_1 \in A'$ and $r_1 \in S'^{ri}_N$. Because, from Lemma 25(1), T is an (A', S'^{ri}_N)-separator, we get $r'_1 \approx r_1$. Then $r_2 \approx r'_2$.

Since α satisfies the first property, $s_2 \not\approx_{T \cup R} r_2$ or $s \not\approx_\alpha r$.

1. In the first case, take $s_2 \not\approx_{T \cup R} r_2$. We get $s_2 \not\approx_{T \cup R} r'_2$, since $r_2 \approx r'_2$. But $s_2 = \widehat{\delta}'(\beta, s)$ and $r'_2 = \widehat{\delta}'(\beta, r)$. So, $\widehat{\delta}'(\beta, s) \not\approx_{T \cup R} \widehat{\delta}'(\beta, r)$, and so β also satisfies the first property.

2. In the second case, take $s \not\approx_\alpha r$. Since $|\alpha'| < |\alpha|$, the minimality of α gives $s_1 \approx_{\alpha'} r_1$. Together with $\alpha = \alpha' a$, we conclude that $\lambda'(a, r_1) \neq \lambda'(a, s_1)$. So $\lambda'(a, r'_1) \neq \lambda'(a, s_1)$, since $r_1 \approx r'_1$. But then we obtain $s \not\approx_\beta r$, so β also satisfies the first property in this case.

The proof is now complete. □

B Obtaining a partial characterization set

The set R is a separator for submachine states and additional states in the specification. We show that R is also such a separator in an implementation that passes the test suite.

Lemma 31. *If $s_0 \approx_\pi s'_0$, then R is a (I'_N, S'^{r}_M)-separator.*

Proof. Let $s' \in S'^{r}_M$ and $t' \in I'_N$ be two distinguishable states. It suffices to show that $s' \not\approx_R t'$.

We will depict the situation. Since $t' \in I'_N$, using Observation 18(4), we get $t \in I_N$ such that $t \approx t'$. Suppose that $s' \notin K$, then by Definition 26(2), we get $s' \not\approx_R t$, and so $s' \not\approx_R t'$, as desired. We will show that, in fact, $s' \notin K$. So, assume, for the sake of contradiction, that $s' \in K$.

From Observation 28, we have $s'_0 \in I$, and since $s' \in K$, we have $s'_0 \neq s'$, otherwise I and K would intersect, contradicting Observation 27(1). We have $s' \in S'^r_M$, then there exists a sequence γ, with minimum length, such that $s'_0 \xrightarrow{\gamma} s'$. Since $s'_0 \neq s'$, we have $|\gamma| \geq 1$.

Let r' be last state in I reached using γ from s'_0, and let $\gamma', \alpha \in X^*$, $a \in X$, and $u' \in S'^r$, such that $\gamma = \gamma' a \alpha$, and $s'_0 \xrightarrow{\gamma'} r' \xrightarrow{a} u' \xrightarrow{\alpha} s'$. Since $s' \in S'_M$, using Observation 15, we get $u' \in S'_M$. Because $r' \in I$, by Definition 26(1), there exist $r \in S_M$, $\rho \in Q$, such that $s'_0 \xrightarrow{\rho} r' \xrightarrow{a} u' \xrightarrow{\alpha} s'$, and $s_0 \xrightarrow{\rho} r \xrightarrow{a} u \xrightarrow{\alpha} s$, for some states $u, s \in S$. See Figure 7.

We have $s_0 \approx_\pi s'_0$, that is, $s_0 \approx_{P \otimes \mathcal{Z}} s'_0$. Since $\rho a \in P$ and $\widehat{\delta}(\rho a, s_0) = u$, we get $s_0 \approx_{\rho a \mathcal{Z}(u)} s'_0$. It follows that $u \approx_{\mathcal{Z}(u)} u'$, and so $u \approx_Z u'$.

Clearly, starting at u', the word α does not reach any state in I, because r' is the last state in I reached using γ. With the next claims, we show that α does not reach any state in L or K.

Claim 1: α does not reach any state in L.
For contradiction, let α' be the minimal prefix of α, such that $u' \xrightarrow{\alpha'} v'$ and $v' \in L$. Therefore, starting at u', α' reaches only distinct states in $J \cup K$ and state v'. From Lemma 29(3), $|\alpha'| \leq |J| + |K| \leq m - n$, so $\alpha' \in X_{m-n}$. Now, let $v \in S$, such that $u \xrightarrow{\alpha'} v$. From $u \approx_Z u'$, and since $Z = X_{m-n}R$, we get $u \approx_{\alpha' R} u'$, and so $v \approx_R v'$. Since $v \in S$, by Definition 26(3), we get $v' \notin L$. This contradiction estabilishes Claim 1.

Claim 2: Let $\sigma \in X^*$. Suppose that, starting at u', σ reaches only distinct states of $J \cup K$ and another state of S'. If $u \xrightarrow{\sigma} v$ and $u' \xrightarrow{\sigma} v'$, then $v \approx_{\overline{\mathcal{R}}(v)} v'$ and $v \approx_R v'$.

Claim 3: Let α' be the minimal prefix of α, such that $u' \xrightarrow{\alpha'} v'$ and $v' \in K$. If $v \in S$ such that $u \xrightarrow{\alpha'} v$, then $v \in I_N$.

Claim 4: α does not reach any state in K.
For contradiction, let α' be the minimal prefix of α, such that $u' \xrightarrow{\alpha'} v'$ and $v' \in K$, and let $v \in S$ such that $u \xrightarrow{\alpha'} v$. From Claim 3, we have $v \in I_N$, then, from Observation 18(3), we have $w' \in I'_N$, such that $v \approx w'$. Let β be a sequence that satisfies the properties of Lemma 30 for states $v' \in K^r$ and $w' \in I'_N$. Let v_1, w'_1 and v'_1 be states such that $v \xrightarrow{\beta} v_1$, $w' \xrightarrow{\beta} w'_1$ and $v' \xrightarrow{\beta} v'_1$. Recall Figure 7.

Consider the sequence $\alpha'\beta$. From Claim 1, combined with the minimality of α', starting at u', only v' and distinct states in set J are reached using α'. Also, since β satisfies property (2) of Lemma 30, starting at v', only v'_1 and distinct states in set K are reached using β. Therefore, from Lemma 29(3), we get $|\alpha'\beta| \leq |J| + |K| \leq m - n$, so $\alpha'\beta \in X_{m-n}$.

AFFIRMATION: $v \approx_\beta v'$ and $v_1 \approx_{R \cup T} v'_1$.
We have $u \approx_Z u'$, so $u \approx_{X_{m-n}} u'$. It follows that $u \approx_{\alpha'\beta} u'$, and then $v \approx_\beta v'$. Also, from CLAIM 2, we get $v_1 \approx_{\overline{\mathcal{R}}(v_1)} v'_1$. Since $v \in S_N$, from Observation 15, $v_1 \in S_N$, so $\overline{\mathcal{R}}(v_1) = \mathcal{R}(S_N) = R \cup T$. Therefore, $v_1 \approx_{R \cup T} v'_1$.

Recall that β satisfies property (1) of Lemma 30. Then $w'_1 \not\approx_{R \cup T} v'_1$ or $w' \not\approx_\beta v'$. Since $v \approx w'$, and then $v_1 \approx w'_1$, we have either $v_1 \not\approx_{R \cup T} v'_1$ or $v \not\approx_\beta v'$. However, this contradicts the AFFIRMATION, establishing the CLAIM 4.

We supposed $s' \in K$, so the CLAIM 4 is a contradiction, and the proof is complete. □

The next corollaries show that R takes on the role of special kinds of separator.

Corollary 32. *If $s_0 \approx_\pi s'_0$, then R is a $(I'_N \cup S'^r_M, I'_N)$-separator.*

Corollary 33. *If $s_0 \approx_\pi s'_0$, then R is a $(\mathrm{nbh}(S'^r_M, 1), \mathrm{nbh}(S'^r_M, 1) \setminus S'^r_M)$-separator.*

Now we show that, for an implementation that passes the test suite, Z distinguishes every pair of additional reachable and distinguishable states. This is in contrast with the analogous sets defined by W-method [3] and by G-method [2]. Note that, here, Z is meant to separate only additional states, thus being a potentially weaker separator than the analogous sets used by the W-method and the G-method, which must separate any pair of states in the entire implementation.

Corollary 34. *If $s_0 \approx_\pi s'_0$, then Z is a partial characterization set for S'^r_M.*

Proof. Define $d = |[(S'_M \setminus S'^r_M)/R]|$, $m' = m - d$ and $n' = n - d$. We have $m' = m - d \geq |S'_M| - |[(S'_M \setminus S'^r_M)/R]| \geq |S'_M| - |S'_M \setminus S'^r_M| = |S'^r_M|$. Also, from Lemma 29(4) and Observation 9(4), we have $n' = n - d \leq |[S'_M/R]| - |[(S'_M \setminus S'^r_M)/R]| \leq |[S'^r_M/R]|$.

Next, we want to use Corollary 13. Let $C = S'^r_M$, $B = \mathrm{nbh}(S'^r_M, 1) = \mathrm{nbh}(C, 1)$. Then, R is a $(B, B \setminus C)$-separator. Since $n' \leq |[S'^r_M/R]|$, R partitions C in at least n' classes. Also, $m' \geq |S'^r_M|$, and so m' is an upper bound on the number of classes of equivalence in C. Now, using Corollary 13, we know that $X_{m'-n'}R$ is a partial characterization set for S'^r_M. Note that $m' - n' = m - n$, then $Z = X_{m-n}R = X_{m'-n'}R$ is a partial characterization set for S'^r_M. □

C Generating a complete test suite

The next lemma states that, for each reachable state in the specification, there exists a corresponding Z-equivalent state in an implementation that passes the test suite.

Lemma 35. *Suppose $s_0 \approx_\pi s'_0$. Let $\gamma \in X^*$ and let $s \in S$, $s' \in S'$ be such that $s_0 \xrightarrow{\gamma} s$ and $s'_0 \xrightarrow{\gamma} s'$. Then $s \approx_Z s'$.*

Proof. For the sake of contradiction, let $\gamma \in X^*$ be minimal such that $s_0 \xrightarrow{\gamma} s$, $s'_0 \xrightarrow{\gamma} s'$ and $s \not\approx_Z s'$. From Observation 28, we know that $s_0 \approx_Z s'_0$, then $|\gamma| > 0$. Let $\gamma' \in X^*, b \in X$ be such that $\gamma = \gamma' b$. Let r be the last state in S_M that can be reached using γ' and starting at s_0. Take $\gamma_1, \gamma_2 \in X^*$ such that $\gamma = \gamma_1\gamma_2$ and $s_0 \xrightarrow{\gamma_1} r$. Note that γ_1 is the maximal prefix of γ' such that $r \in S_M$. Let also $r' \in S'$ be such that $s'_0 \xrightarrow{\gamma_1} r'$. Since $|\gamma_2| > 0$, there exist $\alpha \in X^*, a \in X$ such that $\gamma_2 = a\alpha$. Let $u \in S, u' \in S'$ be such that $r \xrightarrow{a} u$ and $r' \xrightarrow{a} u'$.

We now have

$$s_0 \xrightarrow{\gamma_1} r \xrightarrow{a} u \xrightarrow{\alpha} s \qquad \text{and} \qquad s'_0 \xrightarrow{\gamma_1} r' \xrightarrow{a} u' \xrightarrow{\alpha} s'.$$

Claim 1: $u \approx_Z u'$.
Since $r \in S_M$, we know that there exist $\rho, \rho a \in P$ such that $s_0 \xrightarrow{\rho} r$, and there exist $r'', u'' \in S'$ such that $s'_0 \xrightarrow{\rho} r'' \xrightarrow{a} u''$. Since $s_0 \approx_{PZ} s'_0$, we have $r \approx_Z r''$ and $u \approx_Z u''$. From the minimality of γ, we also have $r \approx_Z r'$. Then $r' \approx_Z r''$. Since $R \subseteq Z$, we get $r \approx_R r'$ and $r \approx_R r''$. From Lemma 24, $r', r'' \in S'_M$, and then $r', r'' \in S'^r_M$. From Corollary 34, Z is a partial characterization set of S'^r_M. It follows that $r' \approx r''$, and so $u' \approx u''$. Since we already have $u \approx_Z u''$, we get $u \approx_Z u'$.

Claim 2: $s \approx_Z s'$.
Let $|\alpha| = 0$, then $u = s$ and $u' = s'$. From Claim 1, we obtain $s \approx_Z s'$, and we are done. Now, assume $|\alpha| > 0$, then we conclude that γ' reaches u. We must have $u \notin S_M$, otherwise γ_1 would not be maximal. Since $r \in S_M$, we also obtain $u \in I_N$, using Definition 16. From Observation 18(3), there exists $t' \in I'_N$ such that $u \approx t'$. Since, from Claim 1, we already have $u \approx_Z u'$, we get $t' \approx_Z u'$, and so $t' \approx_R u'$.

Since $r' \in S'^r_M$, we get $u' \in I'_N \cup S'^r_M$. We now have $t' \approx_R u'$, with $t' \in I'_N$ and $u' \in I'_N \cup S'^r_M$. From Corollary 32, R is a $(I'_N, I'_N \cup S'^r_M)$-separator, then it follows that $t' \approx u'$. Since we already have $u \approx t'$, we get $u \approx u'$, and then $s \approx_Z s'$.

The Claim 2 is a contradiction, so the proof is complete. □

The next result states that in an incorrect implementation we cannot have Z-equivalence between corresponding states in the specification and in the implementation. This result can be established by an argument that closely imitates the proof of the last lemma.

Lemma 36. *Suppose $s_0 \approx_\pi s'_0$. If $s_0 \not\approx s'_0$, then there exist $\gamma \in X^*$, $s \in S$ and $s' \in S'$ such that $s_0 \xrightarrow{\gamma} s$, $s'_0 \xrightarrow{\gamma} s'$, and $s \not\approx_Z s'$.*

The last two lemmas immediately lead to our main result: a combined candidate implementation is equivalent to the combined specification if and only if the implementation passes the given test suite.

Theorem 37. *$s_0 \approx s'_0$ if and only if $s_0 \approx_\pi s'_0$.*

VII Concluding Remarks

Model-based testing methods are widely used to test critical systems. The well-known W-method [3] is one among such methods. It can be used to test completely specified and deterministic FSMs. However, if the number of states is large, then the W-method, and variations of it, become impractical. Moreover, in several common situations, using the W-method can be very inefficient. Such scenarios may arise when testing large implementations that were modified after minor specifications changes have been applied. Another example may happen when testing systems are developed using a building block strategy.

In order to address these issues, we abstracted a FSM as a composition of other previously tested submachines. This gave rise to the concept of combined FSMs, and lead a new test method for combined FSMs, called the C-method, which can generate much smaller test suites. We also introduced the new concept of separators, that generalize the notions of characterization sets and identification sets. This allowed us to replace characterizations sets by the much smaller separators and, by so doing, we could distinguish only as few states as were needed.

Testing separately each submachine using the C-method itself gives rise to a recursive testing strategy, allowing for incremental system testing with a much lower total cost than when using some traditional approaches. We also proved that for some naturally occurring infinite families of FSMs, the recursive strategy can lead to exponentially more efficient test suites, when compared to those obtained using the W-method. This shows that such an incremental strategy, in combination with the C-method, may turn the test of FSMs a much more scalable process.

Acknowledgment

This work was supported by FAPESP (grant # 2008/07969-9).

References

[1] G. V. Bochmann and A. Petrenko, "Protocol testing: review of methods and relevance for software testing," in *ISSTA '94: Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, 1994, pp. 109–124.

[2] A. L. Bonifácio, A. V. Moura, and A. da Silva Simão, "A generalized model-based test generation method," in *6th IEEE International Conferences on Software Engineering and Formal Methods*, 2008, pp. 139–148.

[3] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Transactions on Software Engineering*, vol. 4, no. 3, pp. 178–187, 1978.

[4] R. Dorofeeva, K. El-Fakih, and N. Yevtushenko, "An improved conformance testing method," in *IFIP 25th International Conference on Formal Techniques for Networked and Distributed Systems*, 2005, pp. 204–218.

[5] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Transactions on Software Engineering*, vol. 17, no. 6, pp. 591–603, 1991.

[6] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines - a survey," in *Proceedings of the IEEE*, 1996, pp. 1090–1123.

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

[8] D. Sidhu and T.-K. Leung, "Formal methods for protocol testing: a detailed study," *IEEE Transactions on Software Engineering*, vol. 15, no. 4, pp. 413–426, 1989.

[9] E. M. Clarke and J. M. Wing, "Formal methods: state of the art and future directions," *ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, 1996.

[10] A. L. Bonifácio, A. V. Moura, and A. da Silva Simão, "Exponentially more succinct test suites," Institute of Computing, University of Campinas, Tech. Rep. IC-09-07, 2009.

[11] Koufareva, A. Petrenko, and N. Yevtushenko, "Test generation driven by user-defined fault models," in *Proceedings of the IFIP TC6 12th International Workshop on Testing Communicating Systems*, 1999, pp. 215–236.

[12] K. El-Fakih, N. Yevtushenko, and G. von Bochmann, "Fsm-based re-testing methods," in *TestCom '02: Proceedings of the IFIP 14th International Conference on Testing Communicating Systems XIV*, 2002, pp. 373–390.

[13] ——, "Fsm-based incremental conformance testing methods," *IEEE Transactions on Software Engineering*, vol. 30, no. 7, pp. 425–436, 2004.

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

[15] L. L. C. Pedrosa and A. V. Moura, "Testing combined finite state machines," Institute of Computing, University of Campinas, Tech. Rep. IC-10-01, 2010, available in http://www.ic.unicamp.br/~reltech/2010/abstracts.html.

Epílogo

No artigo apresentado, estendemos o trabalho anterior, apresentando uma nova estratégia de teste para MEFs. Sugerimos uma nova abordagem incremental, que visa viabilizar o teste de novas MEFs, mesmo que essas tenham um número elevado de estados. Mostramos que o método C pode ser uma sub-rotina de um procedimento recursivo para teste de MEFs.

A eficiência de um conjunto de testes depende do tamanho de suas sequências livres de prefixos. Ilustramos a nova abordagem utilizando-a em um exemplo, já apresentado no artigo anterior. Dessa vez, comparamos os método C e W utilizando diretamente a eficiência dos conjuntos de teste gerados. Novamente, obtivemos um ganho bastante razoável. De fato, para o exemplo apresentado, o ganho é de um fator de quase dez vezes.

Embora o exemplo apresentado seja meramente ilustrativo, argumentamos que o ganho do teste incremental de MEFs pode ocorrer com bastante frequência na prática. Ressaltamos a dificuldade de realizar uma análise mais geral da eficiência de conjuntos de teste, já que esta exigiria o cálculo de conjuntos livres de prefixos, que dependem fortemente da MEF sendo testada. Ainda assim, ilustramos com um exemplo mais genérico da utilização dessa nova estratégia, envolvendo uma família infinita de MEFs, que ocorrem naturalmente na prática. Para essa família, geramos os conjuntos de testes utilizando o método C e o método W e calculamos os correspondentes conjuntos livres de prefixos. Em seguida, nós realizamos uma análise assintótica da eficiência dos métodos para a família de máquinas apresentada. Os resultados obtidos apontam para um ganho exponencial quando utilizamos a estratégia incremental. Dito de outra maneira, os conjuntos de testes gerados pelo método C são exponencialmente mais sucintos do que os conjuntos de teses gerados pelo método W sobre essa família infinita de MEFs. Note que os resultados são mantidos em classes de MEFs semelhantes, obtidas com pequenas modificações a partir das MEFs originais. Além disso, tais MEFs podem ocorrer como submodelos de outros sistemas.

Concluímos enfatizando que o método C é o único método a permitir teste de MEFs de maneira escalável, isto é, os outros métodos existentes assumem que o número de estados seja pequeno para que o conjunto de testes gerado possa ser utilizado na prática. Obviamente, o método assume que a implementação satisfaz ao modelo de falhas, isto é, que os sistemas são modulares, com subsistemas bem definidos e desenvolvidos separadamente. No entanto, uma vez que sistemas modulares bem definidos decorrem da boa prática no desenvolvimento de qualquer sistema, argumentamos que as hipóteses do modelo de falhas são razoáveis e ocorrem em aplicações reais.

Capítulo 7

Conclusões

Métodos formais são utilizados para o teste de diversos sistemas críticos. A estratégia consiste em modelar a especificação do sistema como um objeto de algum formalismo e produzir casos de teste a fim de se obter uma verificação metematicamente confiável do sistema. As MEFs são especialmente úteis para modelar uma grande gama de sistemas, dada a sua natureza reativa. Nesse contexto, um caso de teste para uma MEF é uma sequência de símbolos de um determinado alfabeto de entrada acompanhada da saída esperada pela especificação.

Essa dissertação tratou de métodos automáticos para geração de casos de teste. Particularmente, abordamos métodos com cobertura total de falhas. Um problema desses métodos é o tamanho das sequências de testes geradas, que cresce exponencialmente com o número de estados da especificação. É desejável, portanto, encontrar métodos de geração de casos de testes que amenizem esse problema.

Dada a complexidade dos casos de testes gerados, os métodos hoje existentes requerem que as especificações tenham um número de estados reduzido. Em casos de especificações com um número de estados relativamente alto, o custo de utilizar as sequências de teste geradas pode ser excessivamente alto, tornando impraticável o uso dos métodos.

Diversos métodos utilizaram diferentes estratégias para minimizar esse problema, como a utilização identificadores de estados [12], a substituição de conjuntos característicos por conjuntos escolhidos pelo usuário [3] e o uso de testes incrementais para MEFs [11]. Neste trabalho, combinamos tais estratégias e propusemos outras novas, desenvolvendo, assim, novos métodos de geração de casos de testes.

O método G é uma generalização do método W que dispensa a necessidade do cálculo de conjuntos característicos e o método Wp é uma derivação que utiliza parte das sequencias de testes geradas pelo método W. Apresentamos um novo método, denominado método Gp, que combina essas ideias, resultando em um método que não calcula o conjunto característico, como o método Wp, e que produz conjuntos de testes menores que

o método G. No entanto, não foi possível construir conjuntos de testes sucintos com diferente níveis de cobertura de falhas, uma vantagem do método G [2]. De fato, mostramos, por meio de um contraexemplo, que isso não é possível quando são utilizados os conjuntos de testes parciais do método Wp. Além disso, formalizamos a operação de concatenação relativa, introduzida informalmente em [12].

Diversos sistemas complexos são formados por vários subsistemas, que são especificados, desenvolvidos e testados separadamente. Embora as várias partes possam já estar testadas e consideradas corretas, é importante verificar que o sistema integrado apresente um funcionamento correto. Os métodos de geração de casos de teste hoje existentes não são escaláveis no número de estados, o que pode tornar inviável o teste de MEFs complexas. Para resolver esse problema, introduzimos o conceito de MEFs combinadas e apresentamos um novo método de testes para tais MEFs combinadas, o método C, para situações em que as submáquinas já estão previamente testadas. O novo método fornece conjuntos de testes muito mais compactos do que os métodos existentes. Também, introduzimos a noção de separadores de estados, que é uma generalização de conjuntos característicos, e de identificadores de estados, que se mostrou uma ferramenta valiosa na geração de sequências de testes para MEFs.

Utilizando o novo método apresentado para MEFs combinadas, propusemos uma nova estratégia de teste, tornando o teste de MEFs em geral um processo escalável e incremental. Dada uma especificação, possivelmente com um número elevado de estados, esta é abstraída como uma MEF combinada. Cada submáquina da MEF combinada, pode, por sua vez, ser também abstraída como uma MEF combinada. Utilizando esse procedimento recursivo, obtivemos uma abordagem que primeiro testa os subsistemas atômicos e depois os mais complexos, de maneira incremental. Mostramos que, para uma família infinita de MEFs que ocorrem naturalmente na prática, a nova estratégia gera conjuntos de testes exponencialmente mais eficientes do que os conjuntos de testes gerados por métodos tradicionais.

Trabalhos futuros Para trabalhos futuros, sugerimos a extensão dos métodos desenvolvido nessa dissertação. A combinação dos métodos Wp e G, aqui realizada, pode também ser estendida a outros métodos, como o método HSI, que utiliza identificadores de estados harmonizados. É importante investigar se os ganhos reportados para o método G seriam conservados em uma proposta generalização do método HSI. Também, conjecturamos que é possível estender o método C para lidar com máquinas de estados não-determinísticas e parciais – MEFNPs. Assim, seria possível também adotar a mesma estratégia de teste incremental para MEFNPs. Além disso, propomos a realização de experimentos que utilizem os métodos aqui desenvolvidos em aplicações reais.

Referências Bibliográficas

[1] G. V. Bochmann and A. Petrenko. Protocol testing: review of methods and relevance for software testing. In *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, pages 109–124, 1994.

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

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

[4] J. Chen and L. Duan. Optimal synchronizable test sequence from test segments. In *Proceedings of the Sixth International Conference on Quality Software*, pages 266–273, 2006.

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

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

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

[8] A. Y. Duale and M. Uyar. A method enabling feasible conformance test sequence generation for efsm models. *IEEE Transactions on Computers*, 53(5):614–627, 2004.

[9] K. El-Fakih, V. Trenkaev, N. Spitsyna, and N. Yevtushenko. Fsm based interoperability testing methods for multi stimuli model. In *Testing of Communicating Systems*, pages 60–75, 2004.

[10] K. El-Fakih, N. Yevtushenko, and G. V. Bochmann. Fsm-based re-testing methods. In *Proceedings of the IFIP 14th International Conference on Testing Communicating Systems XIV*, pages 373–390, 2002.

[11] 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.

[12] 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.

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

[14] R. M. Hierons and H. Ural. Optimizing the length of checking sequences. *IEEE Transactions on Computers*, 55(5):618–629, 2006.

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

[16] D. Lee and M. Yannakakis. Testing finite-state machines: state identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.

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

[18] G. Luo, G. V. Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–162, 1994.

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

[20] G. J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., 2004.

[21] 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.

[22] A. Petrenko. Fault model-driven test derivation from finite state models: annotated bibliography. In *Modeling and verification of parallel processes*, pages 196–205, 2001.

[23] A. Petrenko and N. Yevtushenko. On test derivation from partial specifications. In *Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification*, pages 85–102, 2000.

[24] A. Rezaki and H. Ural. Construction of checking sequences based on characterization sets. *Computer Communications*, 18(12):911–920, 1995.

[25] P. J. Schroeder and B. Korel. Black-box test reduction using input-output analysis. *SIGSOFT Software Engineering Notes*, 25(5):173–177, 2000.

[26] 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.

[27] N. Spitsyna, K. El-Fakih, and N. Yevtushenko. Studying the separability relation between finite state machines. *Software Testing, Verification and Reliability*, 17(4):227–241, 2007.

[28] M. Yannakakis and D. Lee. Testing finite state machines: Fault detection. *Journal of Computer and System Sciences*, 50(2):209–227, 1995.