Uma Estratégia para Redução de Conjuntos de Seqüências de Teste para Máquinas de Estados Finitos

Jorge Francisco Cutigi

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 07 de abril de 2010

Assinatura: _____

Uma Estratégia para Redução de Conjuntos de Seqüências de Teste para Máquinas de Estados Finitos

Jorge Francisco Cutigi

Orientador: ***Prof. Dr. Adenilso da Silva Simão***

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC/USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Abril/2010

Aos meus pais, João e Fátima.

Agradecimentos

A Deus, por ter colocado pessoas especiais em meu caminho, que tanto contribuíram durante toda a minha vida.

Especialmente aos meus pais, "Jão Cutigi" e "Dona Fátima", por TUDO. Tudo que conquistei devo a eles. Agradeço pela paciência, carinho, compreensão e apoio em minhas decisões. Outro agradecimento mais que especial ao meu inesquecível vô Nicola, por tudo o que vivemos juntos e por todas as boas lembranças que carrego comigo até hoje e que jamais serão esquecidas.

Ao meu professor, orientador e amigo Adenilso, pela paciência, ensinamentos, conselhos e por todas as conversas durante os mais de 5 anos em que trabalhamos juntos. À Simone, sempre muito prestativa.

A toda minha família. Avós, tios, primos, que são tão importantes em minha vida. Em especial aos meus primos Danilo, Patrícia, Fabiano e Cristiano, pela importância que tiveram durante o desenvolvimento deste trabalho. À Helena, pelo carinho.

Aos amigos da minha querida cidade Ibaté, pelas risadas aos fins de semana: Dan, Pati, Mariana, Dú, Kalel, Manu, Brunão, Rayana, Lucas, Loko, Neto, Henrique, Valéria, Thati, Leandro, Pâmela, Thales, Virgílio, Marco, Guilherme, Marina, Zizi e Brenda.

Aos amigos do LabES e agregados por esses quase dois anos de convívio: Fabiano, Nerso, Endo, Gondim, Rodolfo, Rodrigo, Vinícius, Frotinha, Cabeção, Marcão, Marcelo, Draylson, David, Vânia, Rafael, Abe, Bruno, Otávio, KLB, Leandro, Delinha, Gambi, Dusse, Maria, Nardi, Kátia, Erika, Marllos, Vanessa, Lúcio e Alex. Também aos professores do LabES que tiveram participação direta neste trabalho e nas outras atividades da pós-graduação: Adenilso, Simone, Ellen, Rosana e Masiero.

Aos grandes e bons amigos da Info04. Deixo meus agradecimentos especiais aos que foram importantes para mim e que mantenho contato até hoje: Nerso, Ki-Suco, WC, Fabi, Andrezão, Kell, Ribeirão, Carrara, Tio, Thiago, Rubinho, Zanilo, Astro, Xandão, Nagao, Ramon, Nany, Teteco, Bruno, Caneca, Daniel, Alê, Tati, Marco, Josi e ao inesquecível Mineiro.

A outros grandes amigos que sempre estiveram presentes: Drayton, Wesley, Cristiano, Grá, Fran, Piauí, Vinícius, Thais, Fabu, Diogo, Vivian, Bia, Dodô, Marina, Marília, Amandita, Flavinha, Luiz, Taco, Miguel, Damaris, Dai, Ju, Rê, Sérgio, Rita, Guilherme e Rafa.

Aos amigos que ajudaram na revisão desta dissertação: Adenilso, Fabiano, Vinícius, Nerso, Endo, Rodrigo, Gondim e Marcão.

Aos funcionários do Instituto, que tanto contribuem para o bom ambiente de trabalho e estudo. Em especial aos "guardinhas", por todos os momentos de descontração, e às funcionárias da secretaria de pós-graduação, sempre muito atenciosas.

À Capes e à Fapesp pelo apoio financeiro.

Obrigado a todos.

Resumo

O teste baseado em modelos visa à derivação de casos de teste a partir de modelos produzidos ao longo do desenvolvimento de software. Nesse contexto, as Máquinas de Estados Finitos têm sido amplamente pesquisadas e utilizadas para derivação de seqüências de teste. Para isso, vários métodos de geração de seqüências de teste têm sido desenvolvidos há várias décadas. O objetivo desses métodos é a obtenção de um conjunto de teste que seja capaz de revelar os defeitos de uma implementação. Entretanto, muitas vezes os conjuntos gerados são muito grandes, o que torna sua aplicação inviável. Trabalhos recentes definiram condições que podem ser utilizadas para investigar mecanismos de redução de casos de teste. Este trabalho apresenta uma estratégia para a redução de conjuntos de seqüências de teste a partir de Máquinas de Estados Finitos com base em condições de suficiência. A estratégia baseia-se na combinação de seqüências de um conjunto de teste, de forma a reduzir o número de seqüências e o tamanho delas, mantendo a completude do conjunto. São apresentadas seis abordagens de redução baseadas na estratégia proposta, as quais foram implementadas em uma ferramenta. Para avaliar as abordagens foram conduzidos estudos experimentais, os quais também serviram para inferir sobre as características e propriedades de cada abordagem. Além disso, um estudo de caso com MEFs reais também foi realizado.

Abstract

Model-based testing aims at generating test cases from models produced along the software development process. In this context, Finite State Machines (FSM) have been largely investigated and used for generating test sequences. In the past decades, several test generation methods have been proposed to obtain test suites that are able to reveal implementation faults. Nevertheless, most of the generated test suites are huge, thus hindering their application in practice. Recent research has defined new sufficient conditions that can be employed in mechanisms for reducing the length of test sequences. This work presents a strategy based on sufficient conditions for reducing the length of test cases derived from FSMs. Our strategy is based on sequence combination of a test suite, aiming to reduce the number of sequences and their length, however still keeping full fault coverage. Six reduction approaches are presented based on the proposed strategy and implemented in a tool. In order to evaluate the strategy, we conducted experimental studies that identified characteristics and properties for each of the six proposed approaches. Moreover, a case study with real-world FSMs was performed.

Sumário

1 Introdução **1**
1.1 Contextualização 1
1.2 Motivação 3
1.3 Objetivos 4
1.4 Organização 4

2 Teste Baseado em Máquinas de Estados Finitos **5**
2.1 Considerações Iniciais 5
2.2 Fundamentos do Teste de Software 6
2.3 Teste Baseado em Modelos 8
2.4 Processo do Teste Baseado em Modelos 9
2.5 Máquinas de Estados Finitos 11
2.6 Seqüências Básicas 15
2.7 Teste Baseado em Máquinas de Estados Finitos 17
2.7.1 Métodos de Geração de Seqüências 17
2.7.1.1 Método W 18
2.7.1.2 Método Wp 19
2.7.1.3 Método HSI 20
2.7.1.4 Método H 21
2.7.1.5 Método SPY 22
2.7.2 Condições de Suficiência 23
2.7.2.1 Exemplo 25
2.8 Considerações Finais 31

3 Redução de Conjuntos de Seqüências de Teste **33**
3.1 Considerações Iniciais 33
3.2 Exemplo de Redução 34
3.3 Estratégia e Abordagens de Redução 35
3.3.1 Passo 1 35
3.3.2 Passo 2 40
3.4 Exemplos 40
3.4.1 Exemplo da Abordagem Red1.1 40
3.4.2 Exemplo da Abordagem Red1.2 41

3.4.3 Exemplo da Abordagem Red2.1 42
3.4.4 Exemplo da Abordagem Red2.2 43
3.4.5 Exemplo da Abordagem Red3.1 44
3.4.6 Exemplo da Abordagem Red3.2 45
3.4.7 Sumarização dos Exemplos 45
3.5 Aspectos de Implementação 46
3.5.1 Verificador das Condições de Suficiência: CheSCon 46
3.5.2 Abordagens de Redução: RedTest 48
3.6 Considerações Finais 49

4 Avaliação Experimental 51
4.1 Considerações Iniciais 51
4.2 Redução de Conjuntos Gerados por Métodos Clássicos de Geração 52
4.3 Redução de Conjuntos Considerando Variações nos Parâmetros das MEFs . 57
4.3.1 Variação no Número de Estados 57
4.3.2 Variação no Número de Entradas 59
4.3.3 Variação no Número de Saídas 59
4.4 Comparação com Estratégia de Minimização 60
4.5 Análise do Estudo Experimental com MEFs Aleatórias 62
4.6 Estudos de Caso 63
4.6.1 Utilização de *Benchmark* 64
4.6.2 Utilização de Especificações de Protocolos 65
4.7 Considerações Finais 67

5 Conclusões 69
5.1 Contribuições 70
5.2 Dificuldades, Limitações e Trabalhos Futuros 70

Referências 72

Lista de Figuras

2.1 Processo do teste baseado em modelos (Utting e Legeard, 2006). 10
2.2 Representação da MEF da Tabela 2.1 como diagrama de transição de estados. 12
2.3 Exemplo de árvore de teste. 14
2.4 Exemplo de grafo de distinção. 15
2.5 Árvore de teste. 26
2.6 Grafo obtido a partir da árvore de teste. 27

3.1 Sobreposição e Seqüência de Transferência. 37
3.2 Formatos de entrada. 47
3.3 Interfaces da ferramenta *CheSConViz*. 48

4.1 Tamanho do conjunto × Número de *resets*. 54
4.2 Tamanho do conjunto × Tempo. 55
4.3 Número de *resets* × Tempo. 55
4.4 Boxplots da taxa de redução do tamanho do conjunto. 56
4.5 Boxplot da taxa de redução do número de *resets*. 57
4.6 Variação do número de estados. 58
4.7 Variação do número de entradas. 60
4.8 Variação do número de saídas. 62
4.9 Protocolos especificados em MEFs. 66

Lista de Tabelas

2.1 Representação da MEF como uma tabela de transição. 12
2.2 Resultado da confirmação de seqüências por meio do Lema 3. 29
2.3 Verificação da cobertura de estados e transições do conjunto de seqüências confirmadas. 30

3.1 Passo 1 da abordagem **Red1.1**. 41
3.2 Passo 1 da abordagem **Red2.1**. 42
3.3 Passo 2 da abordagem **Red2.1**. 43
3.4 Passo 2 da abordagem **Red2.2**. 44
3.5 Passo 2 da abordagem **Red3.1**. 45
3.6 Sumarização dos resultados dos exemplos. 46

4.1 Média do tamanho dos conjuntos. 53
4.2 Média do número de *resets*. 53
4.3 Taxa de redução do tamanho do conjunto. 53
4.4 Taxa de redução do número de *resets*. 53
4.5 Taxas de redução para cada abordagem considerando conjuntos gerados pelo método HSI. 59
4.6 Taxas de redução do tamanho dos conjuntos, considerando variação nas entradas. 59
4.7 Taxas de redução do número de *resets*, considerando variação nas entradas. 60
4.8 Taxas de redução do tamanho dos conjuntos, considerando variação nas saídas. 61
4.9 Taxas de redução do número de *resets*, considerando variação nas saídas. . 61
4.10 Comparação com a estratégia de minimização proposta por Neto (2008). . 61
4.11 Comparativo entre as abordagens. 63
4.12 MEFs extraídas do *benchmark*. 64
4.13 Taxas de redução do tamanho dos conjuntos para as MEFs do *benchmark*. 65
4.14 Taxas de redução do número de *resets* para as MEFs do *benchmark*. 65
4.15 Taxas de redução do tamanho dos conjuntos para as especificações de protocolo. 67
4.16 Taxas de redução do número de *resets* para as especificações de protocolo. 67

Capítulo

1

Introdução

1.1 Contextualização

O uso de software nas mais diversas áreas de aplicação impõe a necessidade de técnicas e ferramentas que auxiliem em seu desenvolvimento. Com a alta complexidade dos sistemas de software produzidos atualmente, a atividade de teste torna-se uma das etapas mais importantes no ciclo de desenvolvimento. Devido à sua importância, o teste deve ser conduzido com apoio de técnicas, métodos e ferramentas precisas e eficazes, uma vez que essa fase é crucial para se obter software confiável e de boa qualidade.

A atividade de teste consiste em encontrar defeitos inseridos no software durante o seu desenvolvimento, visando garantir a qualidade tanto do processo quanto do produto final. É desejável que os defeitos sejam identificados o mais cedo possível, uma vez que podem ser inseridos em qualquer etapa do desenvolvimento.

Devido ao grande esforço consumido pela atividade de teste durante o desenvolvimento de software, diversas pesquisas têm sido realizadas com o objetivo de identificar técnicas e métodos eficazes para tornar essa atividade mais viável e com menor custo. Nesse contexto, tem-se o teste de conformidade (Bochmann e Petrenko, 1994), que consiste em confrontar o comportamento do software com o comportamento especificado. Como apoio ao teste de conformidade, tem-se o teste baseado em modelos, que possibilita a redução

do custo de geração das seqüências de teste, devido ao fato de ser possível gerar conjunto de teste de maneira automatizada.

No teste baseado em modelos, a representação por meio de Máquinas de Estados Finitos (MEF) (Gill, 1962) vem sendo freqüentemente utilizada devido à sua simplicidade e capacidade de modelar sistemas, principalmente na modelagem de protocolos de comunicação e sistemas reativos. Além disso, o teste baseado em MEFs pode ser aplicado em outros tipos de sistemas, como sistemas orientados a objetos (Binder, 1999; Hong et al., 1995) e sistemas Web (Andrews et al., 2005). Outra vantagem do uso de MEFs segue do fato de existirem vários métodos de geração de seqüências de teste, oferecendo um suporte e direcionamento nos testes gerados e executados.

A utilização de MEFs no contexto do teste de software vem sendo investigado há várias décadas, sendo que os primeiros trabalhos datam das décadas de 50 (Moore, 1956) e 60 (Hennie, 1964). Dentre os métodos mais conhecidos, pode-se destacar os métodos W (Chow, 1978), DS (Gonenc, 1970), UIO (Anido e Cavalli, 1995; Sabnani e Dahbura, 1988), Wp (Fujiwara et al., 1991), HSI (Luo et al., 1994; Petrenko et al., 1993), H (Dorofeeva et al., 2005a) e *State Counting* (Petrenko e Yevtushenko, 2005).

Em geral, a aplicação dos métodos de geração requer que as MEFs possuam certas propriedades, sendo que diferentes métodos podem requerer diferentes conjuntos de propriedades. Dessa forma, os diversos métodos de geração de seqüências de teste a partir de MEFs podem ser classificados com base em três características:

- Aplicabilidade, que se refere às propriedades necessárias para aplicação do método.
- Completude, que se refere à classe de defeitos que o método garante revelar.
- Tamanho dos conjuntos e número de seqüências de teste geradas.

O custo de aplicação dos métodos pode ser considerado sobre o custo de geração das seqüências de teste e sobre o custo da execução. Por exemplo, um método pode ser eficiente para gerar as seqüências de teste, porém se as seqüências geradas forem muito grandes, seu custo de execução é alto, o que pode torná-lo ineficiente. O custo de execução das seqüências de teste é normalmente o fator dominante quando se avalia o custo da aplicação de um método. Sendo assim, o tamanho do conjunto de seqüências de teste é geralmente utilizado para comparar o custo de aplicação do método. Além disso, o número de seqüências geradas pelos métodos também é um fator de influência no custo do teste. Em geral, assume-se a existência de uma operação *reset*, que leva tanto a MEF quanto sua implementação ao seu estado inicial. A operação *reset* deve ser inserida no início de cada seqüência do conjunto de teste; portanto, o número de operações *resets* é igual ao número de seqüências de um conjunto de teste.

Um tópico que possui estreita relação com os métodos de geração e que está presente explícita ou implicitamente nos diversos métodos encontrados na literatura são as condições de suficiência para completude. Um conjunto de condições de suficiência determina quais são as condições que tornam um conjunto de seqüências de teste capaz de revelar todos os defeitos de um dado domínio. As condições de suficiência podem ser utilizadas para abordar vários aspectos como, por exemplo, para demonstrar que um determinado método é aplicável a uma classe maior de MEFs, provando-se que os conjuntos gerados satisfazem as condições de suficiência nessa classe. Além disso, pode-se reduzir o custo de aplicação definindo-se quais seqüências de teste não são necessárias.

1.2 Motivação

Petrenko et al. (1996) definiram conjuntos de condições de suficiência para que um conjunto de seqüências de teste seja completo. Algumas dessas condições são baseadas em métodos de geração já estabelecidos, como por exemplo, os métodos W, Wp, HSI e H. No entanto, a investigação dessas condições possibilitou um melhor entendimento dos métodos e permitiu que melhorias fossem propostas em conjuntos gerados pelos métodos existentes. Outro conjunto de condições de suficiência foi apresentado por Ural et al. (1997), porém são condições para o caso particular de conjuntos que contém apenas uma seqüência de teste. Dorofeeva et al. (2005a) melhoraram os resultados de Petrenko et al. (1996), definindo condições de suficiência que podem ser satisfeitas por mais conjuntos de seqüências de teste.

Simao e Petrenko (2010) definiram condicões de suficiência que generalizam tanto as condicões propostas por Dorofeeva et al. (2005a) quanto as propostas por Ural et al. (1997). Como se tratam de generalizações das condicões anteriores, elas são capazes de reconhecer a completude de conjuntos menores. Sendo assim, é possível a utilização dessas condições de suficiência na redução, minimização e geração de conjuntos de seqüências de teste.

Grande parte dos métodos existentes geram testes compostos de várias seqüências distintas que devem ser aplicadas no estado inicial do sistema. Essa operação pode aumentar o custo do teste, reduzindo sua efetividade (Fujiwara et al., 1991; Hierons, 2004; Yao et al., 1993). Sendo assim, é desejável que um conjunto de teste tenha, além do menor tamanho possível, o mínimo de operações *reset* possível (Hierons, 2004; Hierons e Ural, 2006). Alguns métodos de geração de conjuntos de teste geram seqüências de verificação (Chen et al., 2005; Gonenc, 1970; Hennie, 1964; Hierons e Ural, 2002, 2006; Simao e Petrenko, 2008; Ural et al., 1997; Ural e Zhang, 2006), que se trata da geração de um conjunto de teste unitário, ou seja, com apenas uma seqüência de teste. Nesses casos, considerados

como ideais, o número de seqüência corresponde ao mínimo possível. Porém, esses métodos requerem que a MEF possua uma seqüência de distinção e nem todas as MEFs minimais possuem essa característica (Lee e Yannakakis, 1994).

Considerando que o tamanho do conjunto e o número de seqüências são considerados como os custos na aplicação dos conjuntos de teste, torna-se necessário o estudo e investigação de mecanismos para redução dos conjuntos obtidos por meio do métodos de geração existentes. Os conjuntos de teste gerados pelos métodos existentes podem conter redundâncias e as condições de suficiência podem ajudar a identificar tais redundâncias, produzindo um conjunto menor, com menos seqüências e com mesma efetividade em revelar defeitos.

1.3 Objetivos

O objetivo deste trabalho é a investigação de mecanismos e estratégias para redução de conjuntos de seqüências de teste a partir de MEFs gerados. Busca-se a investigação de mecanismos de redução de casos de teste, os quais foram baseados nas condições de suficiência propostas por Simao e Petrenko (2010). Teve-se como objetivos principais: (1) a redução do tamanho do conjunto de teste; e (2) redução do número de seqüências do conjunto.

Estudos experimentais foram conduzidos para tornar claras as vantagens e desvantagens da estratégia de redução proposta. Dessa forma, o estudo experimental deve evidenciar as características e propriedades da redução, levando-se em conta a redução do tamanho do conjunto, a redução do número de seqüências e o tempo de execução. Além disso, buscou-se também a comparação com uma abordagem de minimização de conjunto de teste, além de estudos em especificações em MEFs de problemas reais.

1.4 Organização

O restante deste documento está organizado da seguinte forma. No Capítulo 2 são apresentados conceitos básicos do teste de software e do teste baseado em modelos, além dos principais conceitos sobre MEFs, sobre o teste baseado em MEFs e sobre métodos de geração de casos de teste. Nesse capítulo também são apresentados os conceitos e propriedades das condições de suficiência. No Capítulo 3 são apresentadas a estratégia de redução e as abordagens propostas, assim como exemplos de cada uma. Resultados de estudos experimentais das abordagens de redução são apresentados no Capítulo 4. Por fim, no Capítulo 5 são apresentadas as conclusões, contribuições e trabalhos futuros provenientes deste trabalho de mestrado.

Capítulo

2

Teste Baseado em Máquinas de Estados Finitos

2.1 Considerações Iniciais

O teste baseado em modelos busca a geração de casos de teste a partir de modelos produzidos durante o ciclo de desenvolvimento de software. Dentre os diversos modelos existentes, as Máquinas de Estados Finitos (MEFs) têm sido amplamente utilizadas para a modelagem de diversas classes de sistema, principalmente a classe de sistemas reativos.

Há na literatura vários métodos de geração de seqüências de teste, além de um conjunto de ferramentas de automação de alguns métodos. Cada método possui certas características, como a classe de MEFs que o método pode ser aplicado, as seqüências básicas utilizadas para geração do conjunto de teste e as propriedades do conjunto gerado.

Uma vez que o teste baseado em MEFs é um assunto que vem sendo discutido há muito tempo pela comunidade científica, neste capítulo é apresentada uma visão mais detalhada sobre esse tipo de teste. Na Seção 2.2 são apresentados os fundamentos do teste de software. Em seguida, na Seção 2.3 é apresentado o teste baseado em modelos. Na Seção 2.5 é apresentado o conceito de Máquinas de Estados Finitos, com sua definição e formas de representação. Além disso, nesse capítulo também é apresentado um conjunto de

características e propriedades de MEFs. Seqüências básicas normalmente caracterizadas em MEFs são apresentadas na Seção 2.6. Em seguida, na Seção 2.7, são apresentadas as características do teste baseado em MEFs, assim como alguns métodos relevantes de geração de seqüências de teste. Ainda na Seção 2.7 é apresentado o conceito de condições de suficiência, além de um exemplo de sua aplicação.

2.2 Fundamentos do Teste de Software

Embora a Engenharia de Software proporcione métodos, técnicas e ferramentas para auxiliar na garantia da qualidade do produto de software desenvolvido, defeitos podem ser inseridos, o que traz a necessidade de uma etapa no desenvolvimento de software que tenha como objetivo minimizar a ocorrência de erros e riscos associados (Maldonado et al., 2004). Uma das atividades dessa etapa é a V&V – Verificação e Validação.

O Teste de Software é uma das atividades de V&V, a qual consiste na análise dinâmica do software com o objetivo de encontrar defeitos no produto e aumentar a confiança nesse. Um teste bem sucedido é aquele que revela a presença de um ou mais defeitos até então não encontrados (Myers, 1979). Quando executado de forma sistemática e criteriosa, o teste contribui para aumentar a confiança de que o software apresenta as funções especificadas, uma vez que, em geral, não é possível provar que um programa está correto (Harrold, 2000; Weyuker, 1996).

Para fins de entendimento deste trabalho, as definições utilizadas para os termos *defeito*, *erro* e *falha* seguem o padrão IEEE 610.12-1990 (IEEE, 1990), tais que:

Defeito (*Fault*): passo ou processo que leva a uma definição de dados incorreta.

Erro (*Error*): manifestação de um defeito no software, caracterizando um estado interno do software que diverge do estado supostamente correto.

Falha (*Failure*): manifestação externa de um erro, ou seja quando ocorre uma divergência na saída do software em relação à especificação.

De acordo com Pressman (2005), uma estratégia de teste de software integra módulos de projeto de casos de teste em uma série planejada de etapas, fornecendo um roteiro que descreve os passos a serem conduzidos. Essa estratégia deve ser flexível e ao mesmo tempo controlada, de forma a promover um planejamento razoável e acompanhamento gerencial à medida que o projeto avança. Uma estratégia de teste deve incorporar atividades, tais como: planejamento de teste, que é responsável por formular a maneira em que a atividade de teste será conduzida, como por exemplo, a escolha das técnicas e critérios a serem utilizados; projeto de casos de teste, o qual consiste na elaboração dos casos de

teste a partir dos critérios estabelecidos; execução do teste, que conduz a aplicação dos casos de teste criados anteriormente; coleta e avaliação dos resultados do teste, a qual se tem um levantamento de como a atividade foi conduzida e os resultados obtidos (Beizer, 1990; Maldonado, 1991; Pressman, 2005).

Um dos pontos mais importantes e cruciais da atividade de teste é o projeto de casos de teste. Um caso de teste é um par ordenado composto pela entrada e pela saída esperada. De acordo com Myers (1979), um bom caso de teste é aquele que tem alta probabilidade de encontrar um defeito ainda não descoberto. Porém, a construção do conjunto de teste não é trivial, umas vez que deve-se selecionar um conjunto específico e finito, já que se torna impraticável utilizar todo o domínio de entrada do software em teste na maioria dos casos. Para isso, tem-se o conceito de critério de teste, que tem como objetivo a seleção e/ou avaliação dos casos de teste, de forma a aumentar as possibilidades em revelar a presença de defeitos e estabelecer um nível elevado de confiança na correção do produto (Fabbri e Maldonado, 2001).

Técnicas de teste foram estabelecidas com o objetivo de encontrar o máximo de defeitos possíveis de um software. Essas técnicas são classificadas de acordo com a origem da informação que é utilizada para estabelecer os requisitos de testes (Maldonado, 1991). As principais técnicas de teste são:

Funcional: conhecido também como caixa-preta, considera o sistema como uma caixa fechada da qual não se tem conhecimento sobre sua implementação ou seu comportamento interno. No teste funcional os testes são gerados somente considerando os valores de entrada e saída do sistema utilizando como base a sua especificação

Estrutural: conhecido também como caixa branca, estabelece os requisitos do software baseados na estrutura interna do produto em teste. A geração dos testes considera as estruturas lógicas e funcionais implementadas, verificando se as funcionalidades e resultados gerados estão de acordo com a especificação. Por ser baseado no conhecimento da estrutura interna da implementação, o testador deve ter acesso ao código fonte do programa, que é utilizado para gerar os casos de teste.

Baseada em Defeitos: estabelece os requisitos de teste explorando os defeitos típicos e comuns cometidos durante o desenvolvimento de software (DeMillo, 1980). Várias características do desenvolvimento de software devem ser consideradas quando se trata do teste baseado em defeitos, como a linguagem utilizada, ferramentas, tipo de software, entre outros.

2.3 Teste Baseado em Modelos

O uso de modelos no contexto de teste de software vem sendo discutido há muito tempo pela comunidade científica. A modelagem é a atividade de construção de modelos que descrevem o comportamento de um sistema. Nesse contexto, o teste baseado em modelos utiliza informações sobre o comportamento do modelo para derivar os casos de teste. Esse modelo deve descrever quais são as ações possíveis e quais são as saídas esperadas. A partir dessa descrição, podem ser aplicados critérios de seleção de casos de teste.

Diversos trabalhos tratam do desenvolvimento e aplicação da geração automática de casos de teste a partir de modelos de software, conhecido como Teste Baseado em Modelos. Essa abordagem consiste em desenvolver e usar um modelo de software, o qual é construído por meio dos requisitos funcionais do software, para derivação dos casos de teste. O modelo é basicamente a especificação das entradas do software e das saídas esperadas, e pode ser criado no início do ciclo de desenvolvimento do software (Dalal et al., 1999). Entretanto, o modelo deve ser construído de modo a ser suficientemente preciso, para que se possa gerar casos de teste significativos a partir dele.

Com essa abordagem, as atividades de teste podem ser iniciadas antes da implementação do software, uma vez que modelos comportamentais são desenvolvidos antes do início da fase de codificação. Dessa forma, é possível revelar defeitos na especificação de requisitos ou na fase de projeto de software (Sinha e Smidts, 2006).

A abordagem do teste baseado em modelos, quando automatizada, pode reduzir o custo da geração dos casos de teste, aumentando a qualidade deles e reduzindo o esforço manual dessa atividade. A automatização é benéfica visto que o software e o modelo são alterados com muita freqüência. Dessa forma os testadores podem atualizar o modelo e rapidamente gerar novos casos de teste, evitando o trabalho manual nessa tarefa (Dalal et al., 1999). Portanto, é de extrema importância que essa fase seja automatizada por meio de ferramentas. Uma ferramenta de apoio ao teste baseado em modelos deve ser capaz de, dado um modelo de software, gerar automaticamente os casos de teste de acordo com algum critério escolhido.

Dalal et al. (1999) realizaram quatro estudos de caso, nos quais foram gerados os casos de teste e executados. Os resultados mostraram que foram revelados muitos defeitos que não haviam sido revelados por abordagens tradicionais de teste. Os resultados de Boberg (2008) indicam que o teste baseado em modelos, desde que praticado a partir do início do processo de desenvolvimento de software, aumenta significativamente o número de defeitos detectados durante o teste.

Existe uma grande variedade de técnicas de modelagem distintas que podem ser usadas no desenvolvimento de modelos de software. Essas técnicas possuem diferenças entre si, tais como o grau de formalidade e mecanismos de modelagem oferecidos por elas. Assim, cada uma dessas técnicas pode oferecer certas vantagens e desvantagens, dependendo do enfoque que se deseja e de qual fase do processo de engenharia de software o modelo será desenvolvido e em quais fases ele será utilizado. Dentre os modelos mais conhecidos e difundidos pode-se citar: Máquinas de Estados Finitos (Gill, 1962), Máquinas de Estados Finitos Estendidas(Petrenko et al., 2004), MEFs comunicantes (Bourhfir et al., 1996), Statecharts (Harel, 1987), Redes de Petri (Peterson, 1977), SDL (ITU, 1993, 2002), Estelle (Turner, 1993), diagramas de estado da UML (Pilone, 2006) e Lotos (Turner, 1993).

Embora o teste baseado em modelos não seja a solução para todos os problemas do teste de software, esse tipo de teste oferece uma possibilidade considerável na redução do custo da atividade de teste. Bertolino (2007) afirma que um dos desafios do teste baseado em modelos é tornar freqüente a utilização de modelos durante o ciclo de desenvolvimento de software, uma vez que a modelagem de sistemas de software complexos é uma tarefa árdua. Por outro lado, após construído o modelo, torna-se possível a geração automática de conjuntos de teste, além de possibilitar um aumento na eficiência dos testes gerados.

2.4 Processo do Teste Baseado em Modelos

Para que o teste baseado em modelos possa ser aplicado de maneira precisa e eficaz, é necessário rigor no processo de teste, o qual envolve atividades e diretrizes para que se tenha sucesso nessa atividade. Para isso, Utting e Legeard (2006) definem um processo, ilustrado na Figura 2.1, para o teste baseado em modelos. Utting e Legeard (2006) dividem o processo em 5 etapas:

1. **Modelar o sistema:** fase em que se cria o modelo do sistema em teste. O modelo criado deve ter um nível de abstração maior que o sistema em si, ou seja, deve ser menor e mais simples. Esse modelo é normalmente criado antes da codificação do sistema, o que possibilita o início da fase de teste antes do fim da construção do software. Terminada a construção do modelo, uma boa prática é utilizar técnicas e ferramentas para verificar se o modelo está consistente e realmente apresenta o comportamento esperado. Para isso, pode-se utilizar técnicas de simulação e *model checking*. O resultado dessa fase é um modelo suficientemente correto, que viabiliza as fases posteriores.

2. **Gerar casos de teste abstratos:** geralmente apoiada por ferramentas, essa fase tem como objetivo a geração dos casos de teste a partir do modelo criado e de

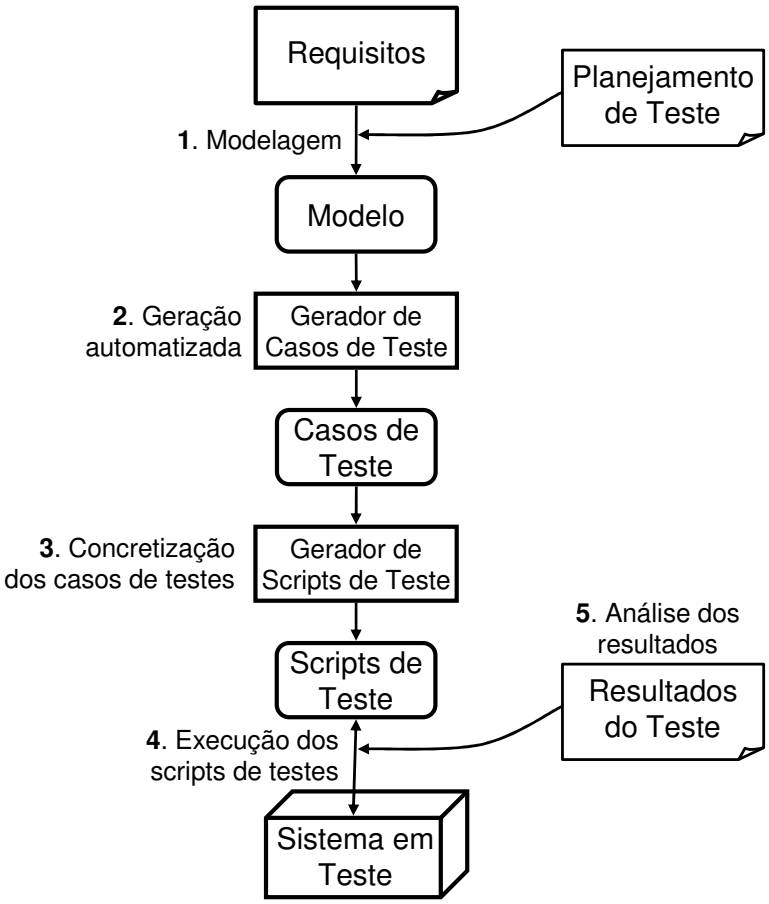


Figura 2.1: Processo do teste baseado em modelos (Utting e Legeard, 2006).

acordo com os critérios de teste estabelecidos. O resultado dessa fase é um conjunto de seqüência de operações executáveis no modelo.

3. **Concretizar os casos de teste abstratos:** fase em que os casos de testes abstratos, gerados na fase 2, são transformados em casos teste concretos e executáveis no sistema em teste. O resultado dessa fase é um conjunto de *scripts* de teste.

4. **Executar os casos de teste:** fase em que se executam, no sistema em teste, os *scripts* de teste gerados na fase anterior. As saídas obtidas na execução do teste no modelo são confrontadas com as saídas obtidas na execução no sistema. O resultado dessa fase é um relatório de execução dos testes.

5. **Analisar o resultado do teste:** fase de análise dos resultados obtidos no relatório de execução do teste. O resultado dessa fase é a definição de ações corretivas no sistema ou no modelo.

2.5 Máquinas de Estados Finitos

As Máquinas de Estados Finitos (MEFs) são uma técnica formal que se tem mostrado bastante útil para tratar o comportamento de sistemas e para ser utilizada no teste de software. Essa técnica é muito utilizada para modelar o comportamento de sistemas reativos, pois esses são essencialmente dirigidos a eventos e dominados por controle. Além disso, as MEFs possuem uma gama de aplicações bastante grande e genérica, podendo ser utilizadas na modelagem de vários tipos de sistemas. Sendo assim, seus modelos são aplicáveis praticamente em todo campo de investigação, como por exemplo, em protocolos de comunicação, sistemas reativos, circuitos elétricos, entre outros.

Segundo Gill (1962), uma MEF é uma máquina hipotética composta por estados e transições. Cada transição liga um estado a a um estado b (a e b podem ser o mesmo estado). A cada instante, uma máquina pode estar em apenas um de seus estados, o que caracteriza uma máquina determinística, caso contrário é uma máquina não-determinística. Em resposta a um evento de entrada, a máquina gera um evento de saída e executa uma transição. Tanto o evento de saída gerado quanto o novo estado são definidos unicamente em função do estado atual e do evento de entrada (Davis, 1988).

De acordo com Petrenko e Yevtushenko (2005), uma MEF M (determinística e de *Mealy*) pode ser representada formalmente por uma tupla $(S, s_0, X, Y, D_M, \delta, \lambda)$, onde:

- S é um conjunto finito de estados, incluindo o estado inicial s_0.
- X é um conjunto finito de entradas.
- Y é o conjunto finito de saídas.
- D é o domínio da especificação, $D \subseteq S \times X$.
- δ é uma função de transição, $\delta : D \rightarrow S$.
- λ é uma função de saída, $\lambda : D \rightarrow Y$.

Uma MEF pode ser representada por um diagrama de transição de estados, conforme a Figura 2.2, que se trata de um grafo direcionado, sendo que cada estado é representado por um vértice e cada transição representada por uma aresta direcionada. Cada aresta possui um rótulo que indica o par *entrada/saída* e uma direção que leva ao próximo estado. Além dessa representação, uma MEF pode ser representada por uma tabela de transição de estados, conforme a Tabela 2.1, por meio das funções λ e δ, que representam os símbolos de saída e o próximo estado da MEF, respectivamente, em relação aos símbolos de entradas.

Tabela 2.1: Representação da MEF como uma tabela de transição.

	δ		λ	
Estado	**a**	**b**	**a**	**b**
s_0	s_3	s_2	1	1
s_1	s_0	s_1	1	1
s_2	s_0	s_0	0	0
s_3	s_1	s_0	1	0

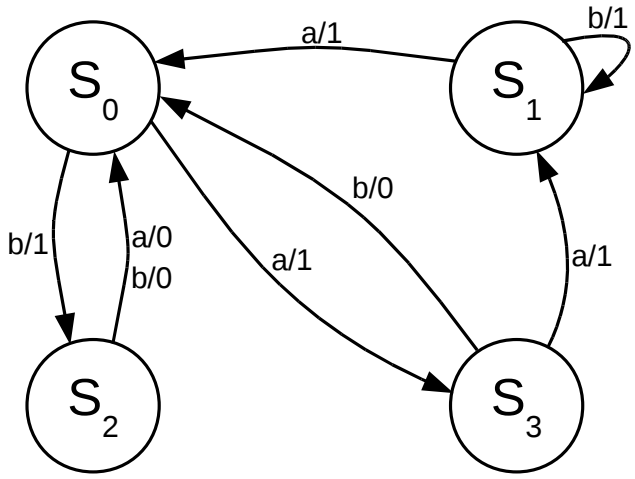


Figura 2.2: Representação da MEF da Tabela 2.1 como diagrama de transição de estados.

O conjunto S de estados representa o conjunto de todas as configurações possíveis do sistema em relação aos símbolos de entrada e saída. A MEF da Figura 2.2 possui o conjunto de estados $S = \{s_0, s_1, s_2, s_3\}$, sendo s_0 o estado inicial. O conjunto finito de entradas X descreve as variáveis de entrada do sistema. A MEF da Figura 2.2 possui o alfabeto de entrada $X = \{a, b\}$. O conjunto finito de saída Y descreve as variáveis de saída do sistema. A MEF da Figura 2.2 possui o alfabeto de saída $Y = \{0, 1\}$. Um exemplo de transição da MEF da Figura 2.2 e da Tabela 2.1, seria que o estado s_3 com entrada a, realiza a transição para o estado s_1, produzindo saída 1.

Uma tupla $(s, x) \in D$ é uma transição *definida* de M. Um seqüência $\alpha = x_1x_2...x_k$ é dito ser uma seqüência de entrada definida no estado $s \in S$ se existe $s_1, s_2, ..., s_{k+1}$, onde $s_1 = s$, tal que $(s_i, x_i) \in D$ e $\delta(s_i, x_i) = s_{i+1}$ para todo $1 < i < k$. Denota-se por $\Omega(s)$ o conjunto de todas as seqüências de entradas definidas no estado s.

A função de transição δ e a função de saída λ são estendidas para seqüência de entradas definida, incluindo a seqüência vazia, que é denotada por ϵ. Tem-se que $\delta(s, \epsilon) = s$ e $\lambda(s, \epsilon) = \epsilon$ para todo $s \in S$. Seja β uma seqüência de entradas e $\delta(s, \beta) = s'$, então, para todo $x \in X$ define-se $\delta(s, \beta x) = \delta(s', x)$ e $\lambda(s, \beta x) = \lambda(s, \beta)\lambda(s', x)$.

Uma MEF é *completa* ou *completamente especificada* se para todo estado $s \in S$ existe uma transição para cada símbolo de entrada $x \in X$, ou seja, se para todo estado da MEF existe uma transição definida para cada símbolo do alfabeto de entrada. Caso contrário, a MEF é parcial ou parcialmente especificada. Formalmente, uma MEF é completa se $D = S \times X$.

Uma MEF é *minimal* se nenhum par de estados da máquina é equivalente, ou seja, para todo par de estados (s_i, s_j), existe ao menos uma seqüência de entradas, definidas em s_i e s_j, cujas saídas produzidas sejam distintas.

Uma MEF é *determinística* se para cada entrada, existe no máximo uma transição definida em cada estado. Caso contrário, a MEF é não-determinística.

Uma MEF é de *Mealy* se para toda transição existe uma saída associada.

Uma MEF é *fortemente conexa* se para todo par de estados (s_i, s_j), existe uma seqüência de entradas, definidas em s_i, que leve de s_i até s_j. De forma análoga, uma MEF é *inicialmente conexa* se qualquer estado pode ser alcançado a partir do estado inicial.

Dois estados $s_i, s_j \in S$ são *equivalentes* se $\lambda(s_i, \gamma) = \lambda(s_j, \gamma)$ para todo $\gamma \in \Omega(s_i) \cap \Omega(s_j)$. Esse conceito pode ser aplicado em estados de MEFs diferentes. MEFs são equivalentes se seus estados iniciais são equivalentes. De forma análoga, dois estados, $s_i, s_j \in S$ são distinguíveis se $\lambda(s_i, \gamma) \neq \lambda(s_j, \gamma)$ para todo $\gamma \in \Omega(s_i) \cap \Omega(s_j)$. MEFs são distinguíveis se seus estados iniciais são distinguíveis.

Duas seqüências α, β são *distinguíveis* se existe $\alpha\gamma, \beta\gamma$, tal que $\lambda(\delta(s_0, \alpha), \gamma) \neq \lambda(\delta(s_0, \beta), \gamma)$. De forma análoga, duas seqüências α, β são *T-distinguíveis* se existe $\alpha\gamma, \beta\gamma \in T$, tal que $\lambda(\delta(s_0, \alpha), \gamma) \neq \lambda(\delta(s_0, \beta), \gamma)$. Duas seqüências α, β são *T-equivalentes* se existe $\alpha\gamma, \beta\gamma \in T$, tal que $\lambda(\delta(s_0, \alpha), \gamma) = \lambda(\delta(s_0, \beta), \gamma)$.

Uma *seqüência de teste* (ou *caso de teste*) é uma seqüência de entrada definida na MEF. Um *conjunto de seqüências de teste* é um conjunto finito de seqüências de teste, tal que não existam dois casos de teste α e β em que α é prefixo de β. O tamanho de um conjunto de seqüências de teste é obtido pelo número de símbolos de entrada contido no conjunto adicionado com o número de operações *resets*.

A operação *reset* é uma operação especial que leva a MEF de qualquer estado para o estado inicial e com saída nula. É denotada pela letra r e aparece como primeiro símbolo em uma seqüência de teste. Portanto, o número de *resets* de um conjunto de teste é o número de seqüências que ele possui.

Na geração de testes a partir de MEFs, assume-se que a implementação pode ser modelada como uma MEF contida em um domínio de defeitos. Essa hipótese, conhecida como hipótese de teste, é necessária para que um conjunto finito de testes possa ser gerado (Chow, 1978; Hennie, 1964; Hierons e Ural, 2006; Ural et al., 1997). $\Im(M)$ denota o domínio de defeitos definido pelo conjunto de todas as MEFs com o mesmo alfabeto de

entrada e no máximo o mesmo número de estados de M, utilizado por grande parte dos métodos de geração, como por exemplo, os métodos W (Chow, 1978), Wp (Fujiwara et al., 1991), HSI (Luo et al., 1994; Petrenko et al., 1993), H (Dorofeeva et al., 2005a), entre outros. $\Im_T(M)$ denota o domínio de defeitos definido pelo conjunto N de todas as MEFs pertencentes a $Im(M)$, tal que N e M são T-equivalentes.

Um conjunto de seqüências de teste T é *n-completo*, ou simplesmente *completo*, se para cada MEF $N \in \Im(M)$ tal que N e M são distinguíveis, existe uma seqüência pertencente a T que distingue N de M. Ou seja, se o conjunto é completo, ele é capaz de revelar todos os defeitos de uma implementação de M que possa ser modelada por uma MEF de $\Im(M)$.

Uma *árvore de teste* é uma representação do conjunto de seqüências de teste, em que cada nó da árvore representa uma seqüência composta pelos símbolos de entrada do caminho que vai da raíz da árvore até ele. Cada aresta da árvore representa a transição da MEF, rotulada com a entrada e saída produzida. A Figura 2.3 representa a árvore de teste considerando a MEF da Figura 2.2 e o conjunto de teste $TS = \{aaab, aab, abab, baab\}$.

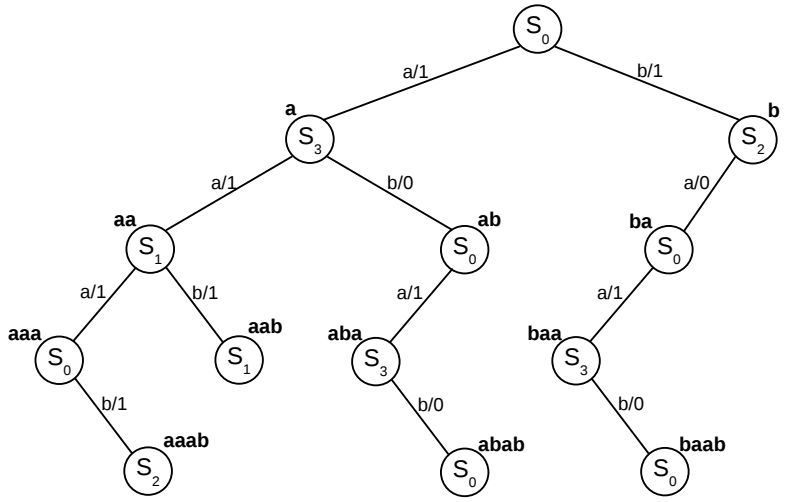


Figura 2.3: Exemplo de árvore de teste.

Um *grafo de distinção* também se trata de uma representação do conjunto de seqüências de teste, em que seus nós, assim como a *árvore de teste*, são representados pela seqüência e pelo estado atingido com essa seqüência. Os nós do grafo são ligados apenas se as seqüências do nó do grafo são T-distinguíveis. A Figura 2.4 apresenta o grafo de distinção considerando a MEF da Figura 2.2 e a árvore de teste da Figura 2.3.

Um *clique* de um grafo é um conjunto de vértices, tal que cada vértice possui ligação com os demais vértices desse mesmo conjunto. As arestas destacadas no grafo da Figura 2.4 apresentam um clique de tamanho 4.

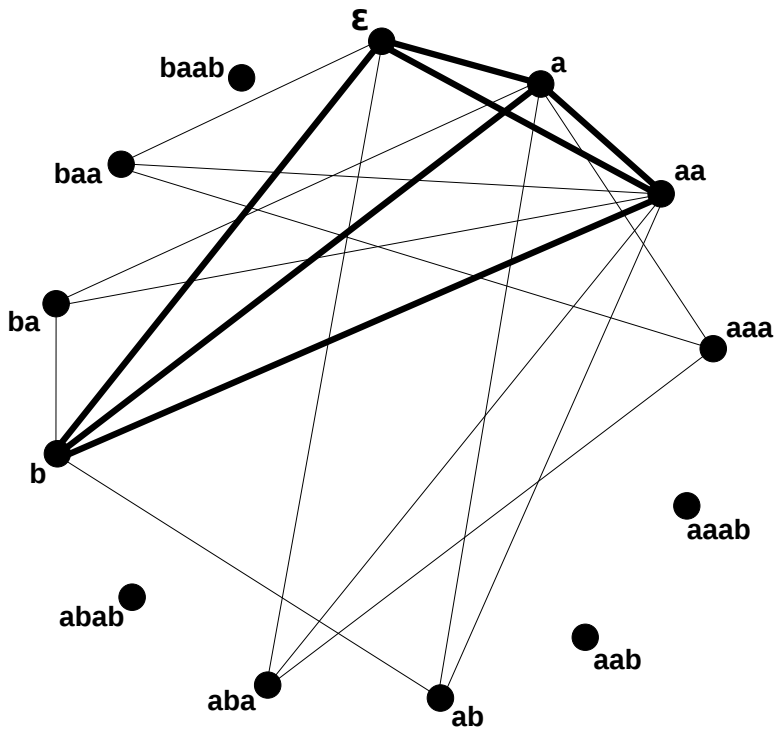


Figura 2.4: Exemplo de grafo de distinção.

Uma *seqüência de verificação* (*checking sequence*) da MEF M é uma seqüência de entradas que diferencia a classe de MEFs equivalentes à M de todas as outras MEFs com no máximo n estados, onde n é o número de estados de M. Ou seja, uma seqüência de verificação é uma seqüência de entradas n-completa que pode ser usada para verificar se uma implementação está correta.

2.6 Seqüências Básicas

Métodos de geração de seqüências de teste são baseados em um conjunto de seqüências básicas, que são muitas vezes utilizadas para obtenção de um resultado parcial importante na geração das seqüências finais de teste (Simao, 2007).

O conjunto *state cover* Q de uma MEF se trata do conjunto de seqüências que leva a MEF do seu estado inicial para cada um dos seus estados, incluindo a seqüência vazia ϵ. Ou seja, para todo estado $s \in S$, existe uma seqüência de entrada $\alpha \in Q$, tal que $\delta(s_0, \alpha) = s$. Dessa forma, uma MEF com n estados, possui um conjunto *state cover* com n seqüências. A MEF da Figura 2.2 possui o conjunto *state cover* $Q = \{\epsilon, a, aa, b\}$.

O conjunto *transition cover* P de uma MEF se trata do conjunto de seqüências que faz com que a MEF exercite todas as suas transições ao menos uma vez, incluindo a seqüência vazia ϵ. O *transition cover* pode ser obtido pelas seqüências do *state cover* concatenadas com as entradas definidas no estado final da aplicação da seqüência do *state cover*. Ou

seja, para cada estado $s \in S$ e para cada entrada $x \in X$, existe uma seqüência de entrada $\alpha \in P$ tal que $\delta(s_0, \alpha) = s$ e $\alpha.x \in P$. A MEF da Figura 2.2 possui o conjunto *state cover* $P = \{a, b, aa, ab, aaa, aab, ba, bb\}$.

Uma *seqüência de transferência* χ de s_i para s_j, é uma seqüência que conduz M do estado s_i para o estado s_j, ou seja, $\delta(s_i, \chi) = s_j$.

Uma *seqüência de distinção* DS de uma MEF é uma seqüência de identificação que produz saídas distintas em cada estado da MEF. Ou seja, para $s_i, s_j \in S$ tal que $i \neq j$, $\lambda(s_i, DS) \neq \lambda(s_j, DS)$. Com isso, pode-se determinar em que estado a MEF estava originalmente. MEFs podem não conter uma DS. A MEF da Figura 2.2 possui o conjunto $DS = \{aba\}$.

Um *conjunto de caracterização*, também chamado de conjunto W, é um conjunto de seqüências tal que a saída observada após a aplicação dessas seqüências sobre cada um dos estados é diferente. Com isso, pode-se identificar todos os estados da MEF. Ou seja, para $s_i, s_j \in S$ tal que $i \neq j$, $\lambda(s_i, W) \neq \lambda(s_j, W)$. Uma seqüência DS pode ser considerada um conjunto W, portanto a MEF da Figura 2.2 possui o conjunto $W = \{aba\}$. Entretanto, essa MEF também possui o conjunto $W = \{ba, a\}$, que não é considerada uma DS, já que é composto por duas seqüências.

Um subconjunto de W também pode ser utilizado para identificação de cada estado da MEF. Este subconjunto W_i é definido para cada estado S_i e é chamado de *conjunto de identificação* para S_i. Ou seja, para $s_i, s_j \in S$ tal que $i \neq j$, $\lambda(s_i, W_i) \neq \lambda(s_j, W_j)$. A união de todos os conjuntos de identificação W_i é o conjunto W. A MEF da Figura 2.2 possui os conjuntos $W_0 = \{ab\}$, $W_1 = \{aba\}$, $W_2 = \{a\}$ e $W_3 = \{aba\}$.

As *seqüências de separação* são determinadas para um par de estados s_i e s_j, e se trata de uma seqüência de entradas que, quando aplicada a s_i e s_j, produz saídas distintas. Dessa forma, a seqüência de separação de s_i e s_j faz a distinção entre esses dois estados. A união das seqüências de separação de s_i com cada estado da MEF é chamado de *conjunto de identificação* H_i do estado i. Ou seja, para $s_i, s_j \in S$ tal que $i \neq j$, $\lambda(s_i, H_i) \neq \lambda(s_j, H_j)$ Para a MEF da Figura 2.2, o conjunto de identificação H para cada estado pode ser $H_0 = \{ab, b\}$, $H_1 = \{ab, b\}$, $H_2 = \{a\}$ e $H_3 = \{a, b\}$.

As seqüências DS, UIO e o conjunto W, pertencem às famílias das seqüências de identificação, que possuem o objetivo básico de identificar o estado em quem a MEF se encontrava antes da execução destas seqüências. Dadas suas características, toda DS é uma seqüência UIO e um conjunto W, e toda seqüência UIO é um conjunto W. Nem toda MEF pode conter seqüências DS ou UIO, porém todas contêm o conjunto W, desde que a MEF seja minimal.

2.7 Teste Baseado em Máquinas de Estados Finitos

O teste baseado em MEFs consiste na geração de um conjunto de seqüências de teste cujo objetivo é encontrar o máximo de defeitos em uma implementação. Dessa forma, é possível verificar se a implementação da MEF está de acordo com sua especificação.

De acordo com Chow (1978), os defeitos são classificadas em:

Defeito de transferência: transição atinge estado incorreto.

Defeito de saída: transição gera uma saída incorreta.

Estados faltantes: os estados da implementação devem ser aumentados para torná-la equivalente à especificação.

Estados extras: os estados da implementação devem ser reduzidos para torná-la equivalente à especificação.

Para que o teste em MEFs possa ser realizado, deve-se estimar o número m de estados da implementação, sendo que quanto melhor for essa estimativa, melhor será o conjunto de teste formado. Os métodos de geração de casos de teste consideram que a MEF possui no máximo m estados, tal que m seja maior ou igual a n (número de estados da especificação). A partir dessa informação, a implementação estará de acordo com sua especificação se não possuir defeitos de transferências nem defeitos de saída. Para fins de entendimento deste trabalho, será considerado o teste de MEFs em que o número de estados é igual o da implementação, ou seja, $n = m$ e a falha de estados extras não é considerada. Dessa forma, $\Im(M)$ contém todas as MEFs que modelam os defeitos que se encaixam no contexto deste trabalho.

Na seção seguinte são apresentados os métodos mais relevantes de geração de seqüências de teste como o W (Chow, 1978), Wp (Fujiwara et al., 1991), HSI (Luo et al., 1994; Petrenko et al., 1993) e H (Dorofeeva et al., 2005a), além do método SPY (Simao et al., 2009), que é o mais recente entre os métodos de geração. Esses métodos são analisados quanto aos aspectos de detecção de defeitos, generalidade, características, além da descrição da geração dos casos de teste para a MEF da Figura 2.2.

2.7.1 Métodos de Geração de Seqüências

Segundo Fujiwara et al. (1991), um método de geração de seqüências de teste tem como objetivo principal oferecer a possibilidade de se executar atividades de teste e validação em sistemas modelados de acordo com alguma técnica de modelagem, por meio de

procedimentos bem definidos para geração dessas seqüências, fazendo com que os sistemas desenvolvidos apresentem maior confiabilidade e qualidade.

Embora os métodos possuam o objetivo comum de verificar se uma implementação está correta com sua especificação, eles diferem com relação ao custo da geração das seqüências de teste, tamanho do conjunto de teste e capacidade de detecção de defeitos (efetividade). Da mesma forma que as seqüências geradas precisam detectar o máximo de defeitos existentes em uma implementação, elas devem ser relativamente pequenas para que seja possível sua aplicação na prática.

De acordo com Fujiwara et al. (1991), um conjunto de seqüências de teste deve apresentar as seguintes características:

1. Ser relativamente pequeno, para que a aplicação do teste seja feita de forma rápida e facilmente executável.

2. Ser capaz de identificar todos os defeitos que a especificação possa conter.

O custo do conjunto de seqüências de teste é obtido pelo tamanho do conjunto gerado. Para cada seqüência do conjunto, deve se considerar a operação *reset*, denotado por um r no início de cada seqüência, que indica o reinício da MEF, ou seja, que a implementação volte ao seu estado inicial. Essa operação de *reset* também deve ser considerada no custo.

2.7.1.1 Método W

Proposto por Chow (1978), o método W pode ser considerado o método mais clássico e famoso no contexto de geração de seqüências de teste para MEFs. O método consiste, basicamente, na geração do conjunto *transition cover* P e do conjunto de caracterização W. Considerando a MEF da Figura 2.2, a aplicação do método W seria:

- Conjunto de caracterização $W = \{aba\}$

- Conjunto *transition cover* $P = \{\epsilon, a, b, aa, ab, aaa, aab, ba, bb\}$

Como esperado, o conjunto de caracterização $W = \{aba\}$ identifica todos os estados da MEF. Para o estado s_0, s_1, s_2, s_3, tem-se as saídas 101, 110, 010 e 111 respectivamente.

Depois de gerados esses dois conjuntos, concatena-se as seqüências do conjunto *transition cover* com as seqüências do conjunto W, ou seja, o conjunto de seqüências de teste final $TS_W = P \bullet W$. Portanto, tem-se o conjunto de teste

$$TS_W = \{aba, aaba, baba, aaaba, ababa, aaaaba, aababa, baaba, bbaba\}$$

Pode-se ainda eliminar os prefixos no conjunto final de teste, por exemplo, eliminando a seqüência *aba*, por ser prefixo da seqüência *ababa*, uma vez que a seqüência *aba* será exercitada pela seqüência *ababa*. Dessa forma tem-se o conjunto final

$$TS_W = \{rbaba, raaaba, rababa, raaaaba, raababa, rbaaba, rbbaba\}$$

de tamanho 43 e com 7 operações *resets*.

Para ser submetida à aplicação do método W, a MEF deve ser completa, minimal, determinística, de Mealy e inicialmente conexa.

2.7.1.2 Método Wp

Fujiwara et al. (1991) introduzem o método Wp (*partial W*), que se trata de uma melhoria no método W, possuindo a vantagem de que os conjuntos gerados são menores que os gerados pelo método W e com a mesma garantia de cobertura dos defeitos. Essa melhoria é obtida pelo fato de que o método Wp utiliza um subconjunto do conjunto W para identificação dos estados. Nesse contexto, cada estado possui um conjunto de identificação de estado, sendo que, em grande parte dos casos, apenas uma parte do conjunto W é utilizada em cada estado.

Como já exemplicado, a MEF da Figura 2.2, possui o conjunto de caracterização $W = \{aba\}$. Com o método Wp, consegue-se reduzir esse conjunto, deixando um conjunto de identificação para cada estado, sendo esse um possível subconjunto do conjunto W. Por exemplo, a seqüência *ab* já é suficiente para distinguir o estado s_0 de todos os outros, uma vez que nenhum outro estado gera a mesma saída para essa seqüência. Dessa forma tem-se o conjunto W_i definido para cada estado s_i, que contém a seqüência de menor tamanho possível suficiente para dintingui-lo dos outros estados. Portanto, conforme descrito na Seção 2.6, tem-se os conjunto $W_0 = \{ab\}$, $W_1 = \{aba\}$, $W_2 = \{a\}$ e $W_3 = \{aba\}$, gerando as saídas 10, 110, 0 e 111 respectivamente.

O método Wp consiste basicamente em duas fases:

Fase 1: Para todas as seqüências do conjunto *state cover* Q é aplicado o conjunto de caracterização W. Ou seja, a primeira fase gera um conjunto $C_1 = Q \bullet W$.

Fase 2: Para todas as seqüências do conjunto *transition cover* P que não estão em Q, ou seja, $R = P - Q$. No conjunto R, aplica-se o W_i correspondente ao estado atingido após a execução de R. Ou seja, tem-se o resultado da segunda fase o conjunto $C_2 = R \otimes W = \bigcup_{p \in R} \{p\} \bullet W_i$, tal que i, tal que $\delta(S_0, p) = S_i$.

Como conjunto final, tem-se a união dos conjuntos gerados pelas duas fases do método, ou seja, $TS_{Wp} = C_1 \cup C_2$. Dessa forma, obtém-se um conjunto de casos de teste menor

em relação ao conjunto gerado pelo método W, pois a concatenação ocorre com um subconjunto W_i ao invés de ocorrer com o conjunto W.

Considerando a MEF da Figura 2.2, a aplicação do método Wp seria:

Conjuntos $W_0 = \{ab\}$, $W_1 = \{aba\}$, $W_2 = \{a\}$ e $W_3 = \{aba\}$

Conjunto *state cover* $Q = \{\epsilon, aa, b, a\}$

Conjunto gerado na Fase 1 $C_1 = Q \bullet W = \{aba, aaba, aaaba, baba\}$

Conjunto *transition cover* $P = \{\epsilon, a, b, aa, ab, aaa, aab, ba, bb\}$

Conjunto $R = P - Q = \{ab, aaa, aab, ba, bb\}$

Conjunto gerado na Fase 2 $C_2 = R \otimes W = \{abab, aaaab, aababa, baab, bbab\}$

Conjunto final $TS_{Wp} = C_1 \cup C_2 = \{aba, aaba, aaaba, baba, abab, aaaab, aababa, baab, bbab\}$

Eliminando os prefixos no conjunto final de teste tem-se o conjunto final

$TS_{Wp} = \{raaaba, rbaba, rabab, raaaab, raababa, rbaab, rbbab\}$

de tamanho 39 e com 7 operações *resets*.

Para ser submetida à aplicação do método Wp, a MEF deve possuir as mesmas características exigidas pelo método W, ou seja, deve ser completa, minimal, determinística, de Mealy e inicialmente conexa.

2.7.1.3 Método HSI

O método HSI (Luo et al., 1994; Petrenko et al., 1993) é o primeiro a gerar seqüências tanto para MEFs parciais quanto para MEFs completas. O método garante cobertura completa dos defeitos existentes, o que o torna um método interessante, uma vez que pode ser aplicado a um conjunto maior de MEFs. O método HSI utiliza o conceito de seqüências de separação para fazer a identificação do estado.

O método HSI consiste basicamente em concatenar as seqüências de separação no fim do *state cover* e do *transition cover*. Para cada estado da MEF, concatena-se o *state cover* do estado com o conjunto de identificação desse mesmo estado. Considerando o *state cover* ($Q = \{\alpha_0, \alpha_1, \ldots, \alpha_n\}$) e o conjunto de identificação ($H = \{H_0, \ldots, H_n\}$), em que n é o número de estados da MEF. As seqüências são geradas de acordo com $r.\alpha_i.H_i$ para cada estado S_j, resultando num conjunto $C_1 = \alpha \otimes H$. Para cada seqüência do conjunto *transition cover* P, é concatenado o conjunto de identificação do estado final da execução da seqüência de P. Com isso, tem-se que para cada seqüência $\alpha \in P$ que leva a

MEF para um estado s_i, a concatenação de α com o conjunto de identificação referente a s_i, que seria H_i, ou seja, o conjunto $C_2 = P \otimes H$.

Considerando a MEF da Figura 2.2, a aplicação do método HSI seria:

Conjuntos $H_0 = \{ab, b\}$, $H_1 = \{ab, b\}$, $H_2 = \{a\}$ e $H_3 = \{a, b\}$

Conjunto $Q \cup P = \{\epsilon, aa, b, a, ab, aaa, aab, ba, bb\}$

Conjunto final $TS_{HSI} = \{ab, b, aaab, aab, ba, aa, ab, abab, abb, aaaab,$
$aaab, aabab, aabb, baab, bab, bbab, bbb\}$

Eliminando os prefixos no conjunto final de teste tem-se o conjunto final

$$TS_{HSI} = \{raaaab, raaab, raabab, raabb, rabab, rabb, rbaab, rbab, rbbab, rbbb\}$$

de tamanho 49 e com 10 operações *resets*.

Para ser submetida à aplicação do método HSI, a MEF deve ser minimal, determinística, de Mealy e inicialmente conexa.

2.7.1.4 Método H

O método H, proposto por Dorofeeva et al. (2005a), é um dos mais recentes métodos de geração de seqüência de teste, além de ser um dos mais poderosos, pois pode ser aplicado em MEFs cuja características e propriedades não são admitidas por outros métodos, além de gerar conjuntos, na maioria das vezes, menores que os outros métodos e garantir cobertura completa dos defeitos existentes. No trabalho de Dorofeeva et al. (2005a) o método H, proposto originalmente para MEFs completas e determinísticas, é estendido para MEFs determinísticas parciais.

Esse método também faz uso de seqüências de separação, tratando-se de uma melhoria do método HSI. Esses dois métodos se diferenciam no fato de que, no método H, as seqüências de separação são geradas durante a construção do conjunto de seqüências de teste. Dessa forma, pode-se obter conjuntos menores de identificação, reaproveitando seqüências que já estão no conjunto de teste.

O método H também pode ser dividido em duas fases:

Fase 1: todas as seqüencias do *state cover* são confrontadas entre si.

Fase 2: determinação das seqüências de separação.

Na primeira fase do método, tomam-se todas as seqüências do conjunto *state cover* Q duas-a-duas. Executa-se o par de seqüências na MEF, determinando os estados de destino. Com isso, identifica-se uma seqüência que faz a separação entre os estados de

destino, definindo assim uma seqüência de separação. Por fim, concatena-se o par extraído do *state cover* com a seqüência de separação e insere a seqüência gerada em uma árvore, chamada de árvore de teste. Nessa primeira fase, para a MEF da Figura 2.2, a árvore de teste compreende o conjunto $A_1 = \{aaab, aab, ab, ba\}$.

Na segunda fase do método toma-se cada seqüência do conjunto *transition cover* e as confronta com o *state cover*, utilizando as seqüências de separação para fazer a distinção entre eles. Por exemplo, toma-se a seqüência $\alpha = aaa$ do *transition cover*, que representa a transição (S_1, a) e que leva ao estado S_0. Deve-se concatenar ao fim de α uma seqüência de separação para distinguir das seqüências do *state cover*, com exceção da seqüência que leva ao estado S_0. Nesse exemplo a seqüência bb faz a distinção entre as seqüências do *state cover*. Nem sempre a inclusão de seqüência é necessária, por exemplo, a seqüência $\alpha = a$ do *transition cover*, que representa a transição (S_0, a) necessita da seqüência de separação abb para fazer a distinção, porém essa seqüência já está no conjunto de teste.

Como resultado dessa fase, tem-se, para a MEF da Figura 2.2, a inclusão do conjunto $A_2 = \{aa, ab, ba, aaabb, aabb, bb, aabbb, babb, bbbb, abbb\}$ na árvore de teste. Com isso, chega-se na árvore de teste final, em que as seqüências de teste são formadas percorrendo a árvore a partir de seu nó inicial até seus nós folhas.

Por fim, considerando a MEF da Figura 2.2, a aplicação do método H resulta no conjunto final

$$TS_H = \{raaabb, raabbb, rabbb, rbabb, rbbbb\}$$

de tamanho 27 e com 5 operações *resets*.

Os autores realizaram um experimento com o método H e mostraram que os conjuntos gerados por esse método é, em média, 66% do tamanho dos conjuntos gerados pelo método HSI (Luo et al., 1994; Petrenko et al., 1993). Nesse mesmo trabalho, os autores afirmam que o tamanho do conjunto gerado depende da ordem na qual as transições são verificadas.

2.7.1.5 Método SPY

O método SPY, proposto por Simao et al. (2009), é o mais recente método de geração de seqüências de teste. Esse método contempla a idéia da redução de ramificações da árvore de teste. Sendo assim, busca-se a geração de um conjunto com um número menor de seqüências, o que, conseqüentemente, diminui a quantidade de operações *resets*. O método SPY, assim como os métodos HSI e H, pode ser aplicado em MEFs determinísticas parciais.

Assim como os outros métodos, o método SPY também possui as fases de verificação de estados e verificação das transições. Considerando a MEF da Figura 2.2, tem-se como resultado da primeira fase a árvore de teste como o conjunto $A_1 = \{aaab, aab, ab, ba\}$.

Na segunda fase, o método SPY se difere dos outros métodos ao não apenas considerar o conjunto *state cover* para verificar as transições. Conforme apresentado em (Simao et al., 2009), novas seqüências podem ser utilizadas, em que as ramificações podem ser evitadas. Como resultado da segunda fase, para a MEF da Figura 2.2 tem-se a árvore de teste e conseqüentemente o conjunto final

$$TS_{SPY} = \{raaabb, raabba, rabbb, rbaab, rbbbb\}$$

de tamanho 27 e com 5 operações *resets*.

Simao et al. (2009) mostram por meio de resultados experimentais que o método SPY gera, em média, conjuntos 40% menores que os conjuntos gerados pelo método HSI. Considerando apenas $m = n$ a redução é de aproximadamente 30%.

2.7.2 Condições de Suficiência

Ao se gerar um conjunto de seqüências de teste, um ponto importante a ser abordado é a maneira de se avaliar a qualidade do conjunto gerado, ou seja, avaliar a efetividade desse conjunto em identificar defeitos. Uma das maneiras de se decidir sobre esse problema é apresentado por Yao et al. (1994), o qual faz uso de uma *tree machine* (outra forma de representação de MEFs) para avaliar a completude. É aplicado um procedimento de redução da *tree machine* e, a partir dessa redução, é possível saber se o conjunto é completo ou não. Outro procedimento possível seria a técnica exaustiva, a qual todas as implementações possíveis da MEF são confrontadas com o conjunto de teste, verificando se existe alguma outra MEF que satisfaz o mesmo conjunto de teste com as mesmas saídas. Simao e Petrenko (2010) afirmam que tanto a técnica exaustiva quanto a de redução de *tree machine* não são escaláveis, ou seja, suas aplicações se tornam inviáveis à medida que a MEF aumenta de tamanho.

Como alternativa a esse problema, tem-se as condições de suficiência para completude de um conjunto de seqüências de teste. As condições de suficiência determinam quais são as condições que tornam um conjunto de seqüências de teste capaz de revelar todos os defeitos de um dado domínio. Ou seja, se determinado conjunto de seqüências de teste satisfaz as condições de suficiência, garante-se que esse conjunto é completo, caso contrário não se pode fazer nenhuma afirmação a respeito da completude do conjunto. Os métodos de geração apresentados neste capítulo geram conjunto de seqüências que, implícita ou explicitamente, fazem uso de condições de suficiência para garantir a completude do conjunto gerado.

Petrenko et al. (1996) definiram um conjunto de condições de suficiência, as quais permitiam provar a completude de diversos métodos de geração já existentes, como o W, Wp, HSI e UIOv. Com isso, pode-se notar que esses métodos já satisfaziam tais condições

de suficiência, porém de maneira implícita. O método H utiliza o potencial das condições de suficiência definidas por Dorofeeva et al. (2005a), satisfazendo as condições explicitamente. As condições definidas por Dorofeeva et al. (2005a) generalizam as condições de Petrenko et al. (1996), ou seja, elas estão contidas nas definidas por Dorofeeva et al. (2005a).

As condições de suficiência de Dorofeeva et al. (2005a) são aplicáveis apenas em conjuntos não unitários, ou seja, quando a MEF possui a operação *reset*. Entretanto, vários métodos de geração tratam da geração de seqüência de verificação, que é um conjunto unitário de teste. Nesse contexto, têm-se os métodos definidos por Chen et al. (2005); Gonenc (1970); Hierons e Ural (2002, 2006); Simao e Petrenko (2008); Ural et al. (1997); Ural e Zhang (2006); Yalcin e Yenigun (2006) cuja completude pode ser provada por meio de condições de suficiência definidas por Ural et al. (1997).

As condições definidas por Dorofeeva et al. (2005a) e Ural et al. (1997) são ortogonais, ou seja, não é possível reduzir umas às outras. Entretanto, Simao e Petrenko (2010) definiram um conjunto de condições de suficiência que generalizam ambos os trabalhos, ou seja, são definidas condições de suficiência para se provar a completude tanto de conjuntos unitários quanto de conjuntos com a operação *reset*.

Simao e Petrenko (2010) definem o conceito de conjunto de seqüências confirmadas.

Definição 1 *(Simao e Petrenko, 2010) Dado um caso de teste T de uma MEF $M = (S, s_0, X, Y, \delta, \lambda)$ e $K \in \Omega_M(s_0)$. O conjunto K é $\Im_t(M)$-confirmado (ou simplesmente confirmado) se $\delta(s_0, K) = S$ e, para cada $N = (Q, q_o, X, Y', \Delta, \Lambda) \in \Im_t(M)$, para todos $\alpha, \beta \in K, \Delta(q_0, \alpha) = \Delta(q_0, \beta)$ se e somente se $\delta(s_0, \alpha) = \delta(s_0, \beta)$. Uma seqüência de entradas é confirmada se existe um conjunto confirmado que a contém.*

Ou seja, em um conjunto confirmado, seqüências que convergem (divergem) na especificação também convergem (divergem) na implementação. Por exemplo, o conjunto *state cover* pode ser considerado um conjunto de seqüências confirmadas, uma vez que se tem a certeza que cada seqüência leva a um estado diferente. Os lemas a seguir indicam algumas maneiras de se construir um conjunto de seqüências confirmadas.

Lema 1 *Seja T um conjunto de seqüências de teste de uma MEF M e K um conjunto* state cover*. Se as seqüências de K são T-distinguíveis entre si, então o conjunto K é confirmado.*

Por exemplo, pode ser confirmado um conjunto de seqüências que fazem parte de um mesmo clique no grafo de distinção, uma vez que todas as seqüências de um clique são T-distinguíveis. Considerando o grafo da Figura 2.4, tem-se as seqüências ϵ, a, aa e b confirmadas.

Lema 2 *Seja K um conjunto de seqüências confirmadas e α uma seqüência que leva a MEF a um estado s. Se para cada $s' \in S$, tal que $s' \neq s$, existe $\beta \in K$ que leva a MEF ao estado s', tal que α e β são T-distinguíveis, então $K \cup \{\alpha\}$ também é um conjunto confirmado.*

Por exemplo, considerando um clique de tamanho N, se uma seqüência α ainda não confirmada está ligada, no grafo de distinção, a um conjunto de $N - 1$ seqüências que fazem parte de um mesmo clique, então α é confirmado. Considerando o grafo da Figura 2.4, a seqüência ab pode ser confirmada, uma vez que ela está ligada no grafo com as seqüências confirmadas a, b e aa, as quais fazem parte de um mesmo clique.

Esses dois lemas consistem a *primeira fase* de confirmação de seqüências. A seguir, no lema , é apresentado a *segunda fase* de confirmação de seqüências.

Lema 3 *Seja K um conjunto de seqüências confirmadas e α algum prefixo do conjunto de teste T. Se existem seqüências β e $\chi \in K$, tal que $\delta(s_0, \beta) = \delta(s_0, \chi)$ e uma seqüência ω, tal que $\beta\omega \in K$ e $\chi\omega = \alpha$, então o conjunto $K \cup \{\alpha\}$ também é um conjunto confirmado.*

Por exemplo, ainda considerando a árvore de teste da Figura 2.3, tem-se a confirmação de $\alpha = baab$. Pois toma-se $\beta = \epsilon$, $\chi = ba$ e $\omega = ab$, tendo $\delta(s_0, baab) = \delta(s_0, ba)$ e $\epsilon.ab \in K$ e $baab = \alpha$. Portanto, $\alpha = baab$ é confirmado

Por fim, após a construção do conjunto K de seqüências confirmadas, aplica-se o Teorema 1 apresentado por Simao e Petrenko (2010). Esse teorema é anunciado abaixo.

Teorema 1 *Dado um conjunto de teste T e uma MEF M com n estados. T é n-completo para M se existe um conjunto K de seqüências confirmadas com as seguintes propriedades:*

1. *$\epsilon \in K$; e*
2. *Para todo $(s, x) \in D$, existe α e $\alpha x \in K$, tal que $\delta(s_o, \alpha) = s$*

Dessa forma tem-se a confirmação da completude de um conjunto de seqüências de teste em relação a uma determinada MEF. Na seção seguinte é apresentado um exemplo prático da aplicação das condições de suficiência.

2.7.2.1 Exemplo

Para melhor entendimento das condições de suficiência, um exemplo é apresentado a seguir. O objetivo desta seção é utilizar as condições de suficiência de Simao e Petrenko (2010) para provar a completude de determinado conjunto de seqüências de teste. No exemplo, define-se na prática os conceitos das condições de suficiência. Para isso, será utilizada a MEF da Figura 2.2 e o conjunto de seqüências de teste

$TS = \{raababa, rabab, rbaabbbab\}$ de tamanho 21 e com 3 operações *resets*. Nota-se que esse conjunto é menor que todos os gerados pelos métodos apresentados na Seção 2.7.1.

Primeiramente cria-se a árvore de teste. A Figura 2.5 ilustra a árvore de teste para o exemplo proposto.

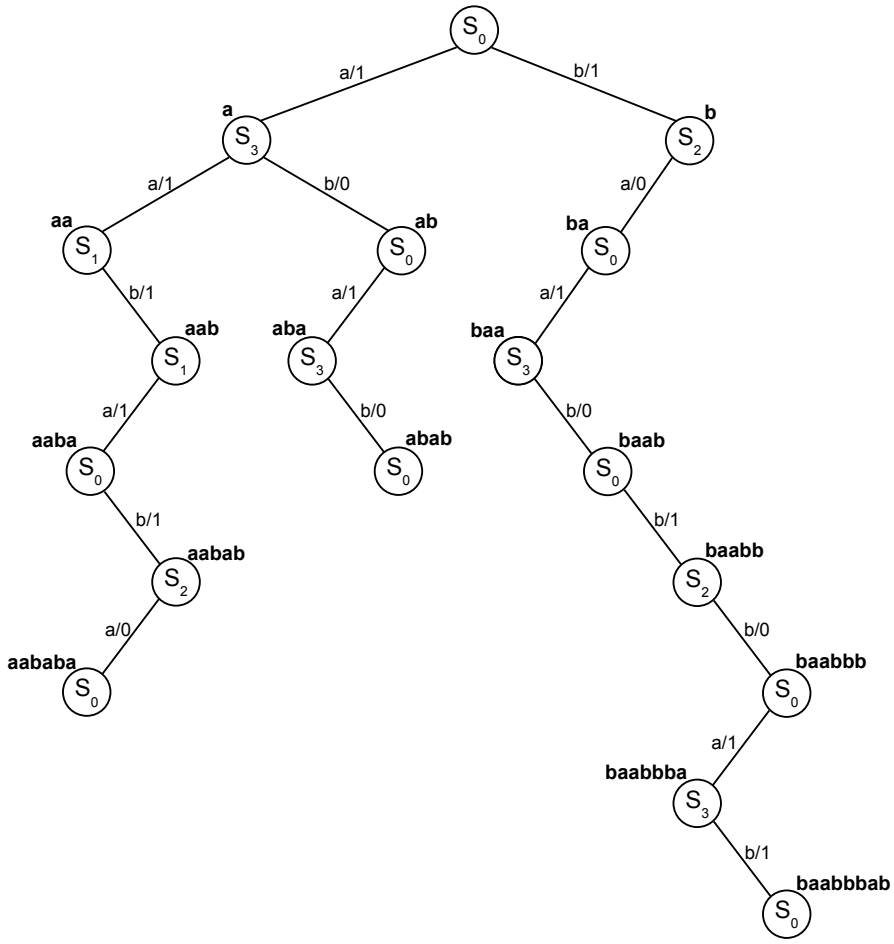


Figura 2.5: Árvore de teste.

Após construída a árvore de teste, o próximo passo é construir o grafo de distinção. Nesse caso, o grafo possui o número de nós igual ao número de nós da árvore de teste. Análogo à arvore de teste, cada nó do grafo também é representado por uma seqüência e por um estado. Para a criação das arestas do grafo, faz-se uso do seguinte procedimento:

1. Tomam-se dois nós da árvore de teste e verifica-se, por meio das saídas geradas, se esses geram saídas distintas com a mesma entrada (definida na árvore).

   Por exemplo, os nós a e ab geram, para a entrada ab, as saídas 11 e 10 respectivamente, que são distintas. Os nós ab e ba não geram saídas distintas para nenhuma entrada definida na árvore. Os nós aa e b também não geram saídas distintas, uma vez que eles não possuem entradas em comum na árvore.

2. Se as saídas geradas pelos nós da árvore são distintas, é criada uma aresta no grafo que liga esses dois nós.

Esse procedimento é realizado para todos os pares de nós da árvore. Com isso, pode-se afirmar que todos os nós ligados no grafo representam estados diferentes na MEF, uma vez que produziram saídas diferentes para determinadas entradas. Porém, a recíproca não é verdadeira, ou seja, não se pode afirmar que os nós que não estão ligados representam o mesmo estado.

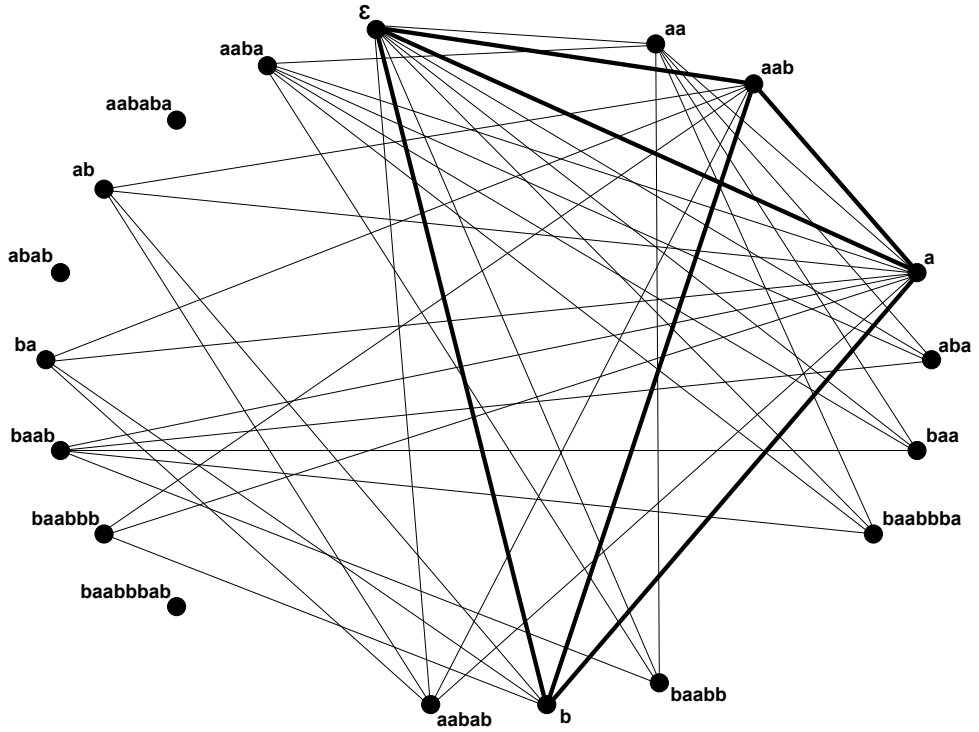


Figura 2.6: Grafo obtido a partir da árvore de teste.

Após esse procedimento, é obtido um grafo k-partido, o qual é ilustrado na Figura 2.6, que se trata de uma categoria de grafos tal que os nós podem ser divididos em k conjuntos distintos, sendo que os vértices de um mesmo conjunto não possuem ligações entre si. Isto acontece devido ao fato de que cada nó do grafo representa uma seqüência que leva a um estado da MEF; logo, o conjunto dos nós que representam seqüências que levam a um mesmo estado não será distinto, ou seja, os nós desse conjunto não estarão ligados no grafo.

Com o grafo construído, o próximo passo é confirmar os nós. Quando um nó é confirmado, tem-se a identificação do estado. Para a primeira etapa de confirmação, tem-se a aplicação do Lema 1, que se reduz ao problema de encontrar *cliques* em um grafo. As seqüências que aparecem em um clique de tamanho n (onde n é o número de estados da

MEF) é um conjunto confirmado, uma vez que, como todos os nós estão ligados, tem-se a certeza que são estados distintos entre si, satisfazendo as condições do Lema 1. O conjunto de seqüências extraído do clique do grafo é o *state cover* da MEF.

Observa-se na Figura 2.6 os quatro conjuntos de nós que levam a cada estado da MEF, caracterizando um grafo 4-partido. As arestas destacadas indicam um clique, o qual gera o conjunto K inicial de estados confirmados, que, nesse caso, é o conjunto $K = \{\epsilon, a, aab, b\}$.

Por ser um problema NP-Completo (Garey e Johnson, 1979), a tarefa de encontrar cliques em um grafo é, em geral, custosa. Porém Grunert et al. (2002) propuseram uma heurística que auxilia essa tarefa. Simao e Petrenko (2010) afirmam que essas heurísticas são satisfatórias para que se tenha uma boa eficiência na solução desse problema.

Depois de identificado um clique no grafo, pode-se confirmar outros nós tomando por base os nós já confirmados. Nessa etapa, aplica-se o Lema 2. Por exemplo, é possível confirmar o nó ab, uma vez que ele está ligado com os nós a, aab e b que já estão confirmados e representam estados diferentes da MEF.

Para o exemplo proposto, após a aplicação dos Lemas 1 e 2 com a construção do grafo, determinação do clique e confirmação dos nós, tem-se o conjunto $K = \{\epsilon, a, aab, b, ab, ba, baabbb, aabab\}$ de seqüências confirmadas.

Quando não se torna mais possível a confirmação de nós no grafo por meio dos Lemas 1 e 2, o Lema 3 pode ser aplicado, consistindo de um novo procedimento para confirmação. Esse novo procedimento tem por objetivo confirmar uma seqüência χ, por meio de seqüências α e β já confirmadas. Se β é prefixo de χ e leva ao mesmo estado de α, e a substituição de β por α gera um seqüência confirmada, então se confirma χ. Por exemplo, tem-se a seqüência $\chi = aaaa$ ainda não confirmada. Porém, as seqüências $\beta = aaa$, que é prefixo de χ, e $\alpha = ab$ já estão confirmadas e levam ao mesmo estado; logo, substituindo β por α tem-se a seqüência aba, que já está contida no conjunto K de seqüências confirmadas. Sendo assim, a seqüência $\chi = aaaa$ também pode ser confirmada, sendo inserida no conjunto K.

Aplicando esse procedimento para as outras seqüências, tem-se como resultado a confirmação de mais oito nós, conforme Tabela 2.2, a qual β e ϕ representam o prefixo e o sufixo de χ, respectivamente.

Com isso, após a aplicação do Lema 3 tem-se o conjunto de seqüências $K = \{\epsilon, a, aab, b, ab, ba, baabbb, aabab, baa, baab, baabb, baabbba, baabbbab, aababa, aba, abab\}$. Porém, nota-se que os nós aa e $aaba$ ainda não foram confirmados. Com isso, o Lema 2 pode ser utilizado novamente, uma vez que as seqüências confirmadas por meio do Lema 3 pode abrir mais possibilidades de confirmação por meio do Lema 2. Nesse caso, aa e $aaba$ podem ser confirmados pelo Lema 2, resultando o conjunto final confirmado

Tabela 2.2: Resultado da confirmação de seqüências por meio do Lema 3.

Seq. χ			Conjunto K resultante
baa	$\beta = ba$ $\phi = a$	$\alpha = \epsilon$ $\alpha.\phi = a$	$\{\epsilon, a, aab, b, ab, ba, baabbb, aabab,$ **baa**$\}$
$baab$	$\beta = baa$ $\phi = b$	$\alpha = a$ $\alpha.\phi = ab$	$\{\epsilon, a, aab, b, ab, ba, baabbb, aabab, baa,$ **baab**$\}$
$baabb$	$\beta = baab$ $\phi = b$	$\alpha = a$ $\alpha.\phi = ab$	$\{\epsilon, a, aab, b, ab, ba, baabbb, aabab, baa, baab,$ **baabb**$\}$
$baabbba$	$\beta = baabbb$ $\phi = a$	$\alpha = \epsilon$ $\alpha.\phi = a$	$\{\epsilon, a, aab, b, ab, ba, baabbb, aabab, baa, baab,$ $baabb,$ **baabbba**$\}$
$baabbbab$	$\beta = baabbba$ $\phi = b$	$\alpha = a$ $\alpha.\phi = ab$	$\{\epsilon, a, aab, b, ab, ba, baabbb, aabab, baa, baab,$ $baabb, baabbba,$ **baabbbab**$\}$
$aababa$	$\beta = aabab$ $\phi = a$	$\alpha = b$ $\alpha.\phi = ba$	$\{\epsilon, a, aab, b, ab, ba, baabbb, aabab, baa, baab,$ $baabb, baabbba, baabbbab,$ **aababa**$\}$
aba	$\beta = ab$ $\phi = a$	$\alpha = \epsilon$ $\alpha.\phi = a$	$\{\epsilon, a, aab, b, ab, ba, baabbb, aabab, baa, baab,$ $baabb, baabbba, baabbbab, aababa,$ **aba**$\}$
$abab$	$\beta = aba$ $\phi = b$	$\alpha = a$ $\alpha.\phi = ab$	$\{\epsilon, a, aab, b, ab, ba, baabbb, aabab, baa, baab,$ $baabb, baabbba, baabbbab, aababa, aba,$ **abab**$\}$

$K = \{\epsilon, a, aab, b, ab, ba, baabbb, aabab, baa, baab, baabb, baabbba, baabbbab, aababa$
$aba, abab, aa, aaba\}$.

Para o exemplo apresentado, as condições do Teorema 1 são verdadeiras, uma vez que ϵ está no conjunto de seqüência confirmadas e existem seqüências α e $\alpha.x$ para todo o par (S, X) da MEF. Por exemplo, as seqüência $\alpha = aa$, que leva ao estado S_2, e a seqüência $\alpha.x = aab$ estão no conjunto K de seqüências confirmadas. Dessa forma, a transição (S_2, b) foi verificada. Na Tabela 2.3 são apresentadas todas as seqüências α e $\alpha.x$ que verificam todas as transições da MEF.

Portanto, prova-se, de acordo com as condições do Teorema 1, que o conjunto $TS = \{raababa, rabab, rbaabbbab\}$ é completo.

É importante ressaltar que esse conjunto é menor que todos os gerados pelos métodos apresentados na Seção 2.7.1, tanto no tamanho quanto no número de operações *resets*. Quanto ao tamanho os métodos W, Wp, HSI, H e SPY geraram conjuntos de tamanhos 43, 39, 49, 27 e 27, respectivamente. Quanto ao número de *resets*, os métodos W, Wp,

Tabela 2.3: Verificação da cobertura de estados e transições do conjunto de seqüências confirmadas.

Par estado/entrada (S, x)	Seqüência α	Seqüência $\alpha.x$
(S_0, a)	ϵ	a
(S_0, b)	ϵ	b
(S_1, a)	aab	$aaba$
(S_1, b)	aa	aab
(S_2, a)	b	ba
(S_2, b)	$baabb$	$baabbb$
(S_3, a)	a	aa
(S_3, b)	a	ab

HSI, H e SPY geraram conjunto com número de *resets* 7, 7, 10, 5 e 5, respectivamente. Já, o conjunto ilustrado nesse exemplo possui tamanho 21 e 3 operações *resets* e possui o mesmo poder de detecção de defeitos, no domínio $\Im(M)$, que os conjuntos gerados pelos métodos apresentados na Seção 2.7.1.

Os passos para verificar as condições de suficiência são apresentados no Algoritmo 1. Esse algoritmo é a base da abordagem de redução apresentada nesta dissertação. Além disso, foi desenvolvida uma ferramenta que implementa as abordagens de redução: uma versão em linha de comando, denominada *CheSCon*, e outra em modo gráfico, denominada *CreSConViz*, com o objetivo de auxiliar a pesquisa. Os detalhes dessas implementações são discutidos na Seção 3.5.1.

Algoritmo 1: Algoritmo de verificação das condições de suficiência.

Data: MEF M e Conjunto TS de seqüências de teste
Result: **Verdadeiro** se o conjunto TS satisfaz as condições de suficiência

```
1  Construir grafo de distinção G de TS;
2  Determinar um clique de tamanho n em G e inserir os elementos do clique no conjunto K;
3  if encontrou clique then
4      while Encontrar uma seqüência α ∈ TS\K, em que os Lemas 2 ou 3 podem ser aplicados do
5          Inserir α em K;
6      end
7      if K satisfaz o Teorema 1 then
8          return verdadeiro;
9      end
10 else
11     return false;
12 end
```

2.8 Considerações Finais

Neste capítulo foram apresentados os principais conceitos e definições a respeito de máquinas de estados finitos e sobre o teste baseado nesse tipo de modelo. Os métodos W, Wp, HSI, H e SPY foram apresentados pelo fato de gerarem conjuntos completos, por construção.

A questão da completude de um conjunto de seqüências de teste foi discutida com a apresentação de um exemplo da aplicação das condições de suficiência. O exemplo mostrou que é possível satisfazer a completude com um conjunto de teste menor que os gerados pelos métodos apresentados na Seção 2.7.1.

Com a fundamentação teórica apresentada, tem-se a base para a realização deste trabalho de mestrado. O capítulo a seguir apresenta as abordagens de redução investigadas e propostas durante a execução deste projeto.

Capítulo

3

Redução de Conjuntos de Seqüências de Teste

3.1 Considerações Iniciais

Os conjuntos de teste gerados pelos métodos clássicos geralmente apresentam redundâncias, ou seja, há seqüências ou subseqüências que poderiam ser descartadas ou substituídas por outra de tamanho menor, obtendo um conjunto menor e mantendo a propriedade de completude do conjunto. A redução do número de seqüências é relevante no sentido de que o número de operações *reset* também diminui.

Neste capítulo são propostas abordagens de redução de conjuntos completos. A estratégia consiste basicamente em dois passos: (1) Combinar as seqüências de um conjunto de seqüências de teste de forma a obter um conjunto ainda completo e com menos seqüências que o conjunto original; (2) A partir do conjunto gerado no **Passo 1**, identificar em cada seqüência do conjunto combinado o menor prefixo dela que ainda mantém a completude do conjunto, para que assim sejam eliminados símbolos de entrada desnecessários.

Evidências encontradas para elaboração da estratégia e das abordagens de redução são descritas na Seção 3.2. Na Seção 3.3 são apresentadas a estratégia utilizada e as abordagens de redução baseadas nela. Na Seção 3.4 é apresentado um exemplo prático para cada

abordagem. Por fim, na Seção 3.5 são apresentados os aspectos de implementação das ferramentas desenvolvidas neste trabalho de mestrado, que inclui a implementação de um verificador das condições do Teorema 1 e a implementação das abordagens de redução.

3.2 Exemplo de Redução

Em conjuntos de seqüências de teste gerados pelos métodos existentes, o número de seqüências é, em geral, alto. Uma maneira de diminuir o número de seqüências do conjunto é a combinação dessas seqüências, mantendo a completude do conjunto. Notou-se, durante o desenvolvimento deste trabalho, que essa combinação, além de diminuir o número de seqüências, pode ainda reduzir o conjunto final, uma vez que a seqüência resultante da combinação pode conter redundâncias.

Por exemplo, toma-se o conjunto

$$TS = \{raaaab, raaab, raabab, raabb, rabab, rabb, rbaab, rbab, rbbab, rbbb\}$$

gerado pelo método HSI para a MEF da Figura 2.2. Tomando-se duas seqüências α e β quaisquer e as combinando, pode-se verificar, por meio das condições do Teorema 1, se o conjunto resultante se mantém completo. Considerando $\alpha = bbab$ e $\beta = abab$, e tendo como resultado de suas combinações a seqüência $\omega = bbababab$, tem-se o conjunto

$$TS = \{rbbababab, raaaab, raaab, raabab, raabb, rabb, rbaab, rbab, rbbb\}$$

que é completo, pois satisfaz as condições do Teorema 1 e pode ser verificado pelo Algoritmo 1. Nota-se que com esse passo já se obteve uma diminuição no número de seqüências do conjunto. A seqüência $\omega = bbababab$ gerada pode conter redundâncias no sentido de que ela não é necessária por inteiro para manter o conjunto de teste completo. Por exemplo, tem-se que apenas o prefixo $bbabab$ de ω é necessário para manter a completude do conjunto. Com isso, tem-se o conjunto

$$TS = \{rbbabab, raaaab, raaab, raabab, raabb, rabb, rbaab, rbab, rbbb\}$$

ainda completo. Com isso, além da redução do número de seqüências, tem-se também a redução de símbolos de entradas desnecessários, o que resulta na redução do tamanho do conjunto final.

A partir dessas evidências, tem-se o caminho para a investigação de abordagens de redução. Algumas formas de combinação podem ser utilizadas, bem como várias maneiras de escolher as seqüências para combinação. A junção dessas possibilidades resulta em abordagens com propriedades distintas, conforme apresentadas nas seções seguintes.

3.3 Estratégia e Abordagens de Redução

As abordagens de redução admitem duas entradas: (1) uma MEF M completa ou parcial, fortemente conexa, determinística e reduzida, e (2) um conjunto de seqüências de teste T que satisfaz as condições do Teorema 1. Note que todos os conjuntos gerados pelos métodos W, Wp, HSI e H satisfazem a condições do Teorema 1, por construção.

A estratégia de redução é dividida em dois passos, descritos a seguir:

Passo 1: Dado um conjunto completo de seqüências de teste, enquanto existem seqüências que se combinadas formam um conjunto completo, tomam-se duas seqüências que se combinadas ainda mantém a completude do conjunto. O conjunto resultante é então submetido ao mesmo processo de combinação de seqüências, até que não seja mais possível manter a completude do conjunto com as combinações de seqüências.

Passo 2: Após a obtenção do conjunto combinado, para cada seqüência deve-se identificar o prefixo necessário para manutenção da completude do conjunto, descartando os símbolos de entrada desnecessários.

Nas seções seguintes são apresentados, em detalhes, os Passos 1 e 2 da estratégia de redução.

3.3.1 Passo 1

No **Passo 1**, alguns mecanismos podem ser utilizados. Os mecanismos se diferem na forma em que as seqüências são escolhidas para combinação e na maneira em que são combinadas. Na exemplificação do **Passo 1**, será considerado o conjunto $TS = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$, e considera-se que qualquer combinação de seqüências gera um conjunto completo. A seguir, são apresentados os mecanismos em que as seqüências são escolhidas para combinação:

Esc1: Qualquer par de seqüências pode ser escolhido para que seja feita a combinação, inclusive seqüências que já foram combinadas. Dessa forma, a cada combinação de seqüências o conjunto todo pode ser combinado entre si novamente.

Por exemplo, tomam-se as seqüências α_1 e α_2, gerando a seqüência $\alpha_1\alpha_2$ e o conjunto $TS = \{\alpha_1\alpha_2, \alpha_3, \alpha_4\}$. Pode-se, então, utilizar a seqüência $\alpha_1\alpha_2$ em outra combinação, por exemplo, combinando-a com α_4, e gerando o conjunto $TS = \{\alpha_1\alpha_2\alpha_4, \alpha_3\}$. Esse conjunto ainda pode ser combinado, até não haver mais possibilidades.

Esc2: Os pares de seqüências podem ser escolhidos para combinação desde que não tenham sido combinados anteriormente. Dessa forma, a cada combinação, apenas as

seqüências ainda não combinadas, ou seja, que fazem parte do conjunto original, podem ser combinadas entre si. Portanto, o conjunto resultante é formado apenas por pares de seqüências combinadas uma só vez.

Por exemplo, a combinação de α_1 e α_2 gera a seqüência $\alpha_1\alpha_2$ e o conjunto $TS = \{\alpha_1\alpha_2, \alpha_3, \alpha_4\}$. Nesse conjunto, pode-se apenas escolher as seqüências que ainda não foram combinadas, ou seja, α_3 e α_4, que combinadas geram a seqüência $\alpha_3\alpha_4$ e o conjunto $TS = \{\alpha_1\alpha_2, \alpha_3\alpha_4\}$. Como as seqüências só podem ser combinadas uma única vez, esse é o conjunto final resultante nesse tipo de escolha.

Esc3: Similiar a **Esc2**, porém o conjunto combinado em pares é submetido ao mesmo procedimento **Esc2** até que se obtenha um conjunto em que nenhuma combinação de seqüências resulta em um conjunto completo.

Por exemplo, tomando-se o conjunto resultante em **Esc2**, pode-se submetê-lo novamente ao processo de combinação aos pares. Portanto as seqüências $\alpha_1\alpha_2$ e $\alpha_3\alpha_4$ podem ser novamente combinadas, resultando na seqüência $\alpha_1\alpha_2\alpha_3\alpha_4$ e no conjunto unitário $TS = \{\alpha_1\alpha_2\alpha_3\alpha_4\}$.

A seguir, as maneiras em que as seqüências são combinadas são apresentadas:

- **Sobreposição:** Essa forma de combinação consiste na sobreposição de entradas em duas seqüências de entrada. A combinação por sobreposição se dá em uma seqüência $\alpha = \chi\phi$ concatenada com uma seqüência $\beta = \phi\gamma$, gerando uma seqüência $\omega = \alpha\gamma$, onde $\delta(s_1, \chi) = s_1$. Nesta dissertação, a operação de combinação por sobreposição é denotada por $concatSP(\alpha, \beta)$.

- **Seqüência de transferência:** Ao fim da seqüência α deve ser adicionada uma seqüência de transferência χ que leva a MEF do estado $s = \delta(s_1, \alpha)$ ao estado inicial s_1. Antes de concatenar a seqüência β escolhe-se a menor seqüência de transferência. Dessa forma, a combinação final resulta em $\omega = \alpha\chi\beta$. Nesta dissertação, a operação de combinação por seqüência de transferência é denotada por $concatST(\alpha, \beta)$.

A combinação por sobreposição é preferível em relação à que utiliza seqüências de transferência, uma vez que, com a sobreposição, entradas são reaproveitadas, o que leva a uma diminuição no tamanho da seqüência ω. O contrário acontece com o uso de seqüência de transferência, em que o tamanho da seqüência ω obtida é maior que a soma dos tamanhos de α e β, o que aumenta o tamanho da seqüência. Nota-se que essas duas formas de combinação mantém em ω as transições originais de α e β. A Figura 3.1 ilustra esses dois processos de combinação, em que a seqüência $\alpha = aaabb$ é combinada por sobreposição com a seqüência $\beta = bbaa$, que resulta na seqüência $aaabbaa$. A seqüência

$\beta = bbaa$ também é combinada com a seqüência $\gamma = baab$ por meio da seqüência de transferência $\chi = a$, resultando na seqüência $bbaaabaab$.

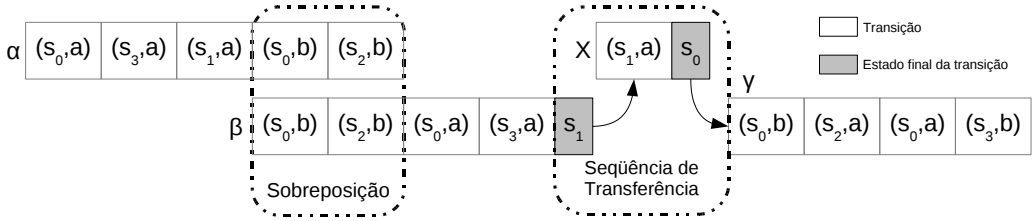


Figura 3.1: Sobreposição e Seqüência de Transferência.

Com isso, têm-se dois mecanismos para a combinação das seqüências, os quais são descritos a seguir:

Comb1: A primeira forma combina as seqüências apenas se a sobreposição for possível. Caso contrário, não há combinação.

Comb2: A primeira forma verifica todas as possibilidades em que a sobreposição de seqüências pode ser realizada. Após combinar as seqüências utilizando sobreposição, seqüências de transferência devem ser utilizadas para combinar as seqüências restantes.

Dessa forma, **Comb2** sempre combina as seqüências escolhidas, enquanto **Comb1** apenas em alguns casos.

Com as três possibilidades de escolhas de seqüências para combinação e com as duas maneiras de se combinar as seqüências, têm-se um total de seis abordagens de redução, provenientes da combinação de **Esc1**, **Esc2** e **Esc3** com **Comb1** e **Comb2**:

Abordagem Red1.1 As seqüências são escolhidas para combinação de acordo com **Esc1** e as combina de acordo com **Comb1**. O Algoritmo 2 apresenta essa abordagem.

Algoritmo 2: Algoritmo da abordagem **Red1.1**.

1 **while** *existe* $\alpha, \beta \in TS$, *tal que* $TS - \{\alpha, \beta\} \cup \{concatSP(\alpha, \beta)\}$ *é completo* **do**
2 $\quad TS \leftarrow TS - \{\alpha, \beta\} \cup \{concatSP(\alpha, \beta)\}$;
3 **end**
4 **return** TS

Abordagem Red1.2 As seqüências são escolhidas para combinação de acordo com **Esc1** e as combina de acordo com **Comb2**. O Algoritmo 3 apresenta essa abordagem.

Algoritmo 3: Algoritmo da abordagem **Red1.2**.

1 **while** *existe* $\alpha, \beta \in TS$, *tal que* $TS - \{\alpha, \beta\} \cup \{concatSP(\alpha, \beta)\}$ *é completo* **do**
2 $\quad TS \leftarrow TS - \{\alpha, \beta\} \cup \{concatSP(\alpha, \beta)\}$;
3 **end**
4 **while** *existe* $\alpha, \beta \in TS$, *tal que* $TS - \{\alpha, \beta\} \cup \{concatST(\alpha, \beta)\}$ *é completo* **do**
5 $\quad TS \leftarrow TS - \{\alpha, \beta\} \cup \{concatST(\alpha, \beta)\}$;
6 **end**
7 **return** TS

Abordagem Red2.1 As seqüências são escolhidas para combinação de acordo com **Esc2** e as combina de acordo com **Comb1**. O Algoritmo 4 apresenta essa abordagem.

Algoritmo 4: Algoritmo da abordagem **Red2.1**.

1 $TS_{naoEsc} \leftarrow TS$;
2 $TS_{concat} \leftarrow \{\emptyset\}$;
3 **while** *existe* $\alpha, \beta \in TS_{naoEsc}$, *tal que* $(TS_{naoEsc} - \{\alpha, \beta\}) \cup (TS_{concat} \cup \{concatSP(\alpha, \beta)\})$ *é completo* **do**
4 $\quad TS_{concat} \leftarrow TS_{concat} \cup \{concatSP(\alpha, \beta)\}$;
5 $\quad TS_{naoEsc} \leftarrow TS_{naoEsc} - \{\alpha, \beta\}$;
6 **end**
7 **return** TS_{concat}

Abordagem Red2.2 As seqüências são escolhidas para combinação de acordo com **Esc2** e as combina de acordo com **Comb2**. O Algoritmo 5 apresenta essa abordagem.

Algoritmo 5: Algoritmo da abordagem **Red2.2**.

1 $TS_{naoEsc} \leftarrow TS$;
2 $TS_{concat} \leftarrow \{\emptyset\}$;
3 **while** *existe* $\alpha, \beta \in TS_{naoEsc}$, *tal que* $(TS_{naoEsc} - \{\alpha, \beta\}) \cup (TS_{concat} \cup \{concatSP(\alpha, \beta)\})$ *é completo* **do**
4 $\quad TS_{concat} \leftarrow TS_{concat} \cup \{concatSP(\alpha, \beta)\}$;
5 $\quad TS_{naoEsc} \leftarrow TS_{naoEsc} - \{\alpha, \beta\}$;
6 **end**
7 **while** *existe* $\alpha, \beta \in TS_{naoEsc}$, *tal que* $(TS_{naoEsc} - \{\alpha, \beta\}) \cup (TS_{concat} \cup \{concatST(\alpha, \beta)\})$ *é completo* **do**
8 $\quad TS_{concat} \leftarrow TS_{concat} \cup \{concatST(\alpha, \beta)\}$;
9 $\quad TS_{naoEsc} \leftarrow TS_{naoEsc} - \{\alpha, \beta\}$;
10 **end**
11 **return** TS_{concat}

Abordagem Red3.1 As seqüências são escolhidas para combinação de acordo com **Esc3** e as combina de acordo com **Comb1**. O Algoritmo 6 apresenta essa abordagem.

Algoritmo 6: Algoritmo da abordagem **Red3.1**.

1 completo $\leftarrow$ *verdadeiro*;
2 $TS_{naoEsc} \leftarrow TS$;
3 **while** *completo* **do**
4 completo $\leftarrow$ *falso*;
5 $TS_{concat} \leftarrow \{\emptyset\}$;
6 **while** *existe* $\alpha, \beta \in TS_{naoEsc}$, *tal que* $(TS_{naoEsc} - \{\alpha, \beta\}) \cup (TS_{concat} \cup \{concatSP(\alpha, \beta)\})$ *é completo* **do**
7 $TS_{concat} \leftarrow TS_{concat} \cup \{concatSP(\alpha, \beta)\}$;
8 $TS_{naoEsc} \leftarrow TS_{naoEsc} - \{\alpha, \beta\}$;
9 completo $\leftarrow$ *verdadeiro*;
10 **end**
11 **end**
12 **return** TS_{concat}

Abordagem Red3.2 As seqüências são escolhidas para combinação de acordo com **Esc3** e as combina de acordo com **Comb2**. O Algoritmo 7 apresenta essa abordagem.

Algoritmo 7: Algoritmo da abordagem **Red3.2**.

1 completo $\leftarrow$ *verdadeiro*;
2 $TS_{naoEsc} \leftarrow TS$;
3 **while** *completo* **do**
4 completo $\leftarrow$ *falso*;
5 $TS_{concat} \leftarrow \{\emptyset\}$;
6 **while** *existe* $\alpha, \beta \in TS_{naoEsc}$, *tal que* $(TS_{naoEsc} - \{\alpha, \beta\}) \cup (TS_{concat} \cup \{concatSP(\alpha, \beta)\})$ *é completo* **do**
7 $TS_{concat} \leftarrow TS_{concat} \cup \{concatSP(\alpha, \beta)\}$;
8 $TS_{naoEsc} \leftarrow TS_{naoEsc} - \{\alpha, \beta\}$;
9 completo $\leftarrow$ *verdadeiro*;
10 **end**
11 **while** *existe* $\alpha, \beta \in TS_{naoEsc}$, *tal que* $(TS_{naoEsc} - \{\alpha, \beta\}) \cup (TS_{concat} \cup \{concatST(\alpha, \beta)\})$ *é completo* **do**
12 $TS_{concat} \leftarrow TS_{concat} \cup \{concatST(\alpha, \beta)\}$;
13 $TS_{naoEsc} \leftarrow TS_{naoEsc} - \{\alpha, \beta\}$;
14 completo $\leftarrow$ *verdadeiro*;
15 **end**
16 $TS_{naoEsc} \leftarrow TS_{concat} \cup TS_{naoEsc}$;
17 **end**
18 **return** TS_{concat}

3.3.2 Passo 2

A partir do conjunto gerado no **passo 1**, o **passo 2** é responsável por identificar em cada seqüência do conjunto combinado o menor prefixo dela que ainda mantém a completude do conjunto, para que assim sejam eliminados símbolos de entrada desnecessários. Esse passo é o principal responsável pela redução do tamanho final do conjunto de seqüências de teste. O passo 2 é apresentado no algoritmo 8.

Algoritmo 8: Algoritmo do passo 2.

```
1 for cada seqüência α ∈ TS do
2     seja β o menor prefixo de α tal que TS − {α} ∪ {β} é completo;
3     TS ← TS − {α} ∪ {β};
4 end
5 return TS;
```

3.4 Exemplos

Nesta seção são apresentados os exemplos de cada abordagem de redução, considerando a MEF da Figura 2.2 e o conjunto completo

$$TS = \{raaaab, raaab, raabab, raabb, rabab, rabb, rbaab, rbab, rbbab, rbbb\}$$

gerado pelo método HSI, de tamanho 49 e 10 operações *resets*.

3.4.1 Exemplo da Abordagem Red1.1

Tomam-se $\alpha = aaaab$ e $\beta = abab$, ambas sem as operações *resets*. Nesse caso, a combinação por sobreposição é possível, pois a seqüência α pode ser decomposta em $\chi = aaa$ e $\phi = ab$, com $\delta(s_0, \chi) = s_0$, e a seqüência β pode ser decomposta em $\phi = ab$ e $\gamma = ab$. Com isso, tem-se $\omega = \alpha\gamma = aaaabab$, que resulta no conjunto $TS = \{raaaabab, raaab, raabab, raabb, rabb, rbaab, rbab, rbbab, rbbb\}$. Verifica-se que esse conjunto satisfaz as condições do Teorema 1. A partir desse conjunto resultante, repete-se o mesmo procedimento. Neste exemplo, mais 5 sobreposições são possíveis. A Tabela 3.1 apresenta as sobreposições realizadas, inclusive a que já foi apresentada nesse parágrafo.

Dessa forma, após a impossibilidade de combinação por sobreposição, tem-se o conjunto resultante $TS_{Esc_1Comb_1} = \{raaabaab, raababbbbab, raabb, raaaababbab\}$ como resultado do **passo 1**.

Após a criação do conjunto combinado obtido no **passo 1**, tem-se o **passo 2** do algoritmo, o qual consiste em remover os símbolos de entradas desnecessários. Por exemplo, tomando-se o conjunto $TS_{Esc_1Comb_1} = \{raaabaab, raababbbbab, raabb, raaaababbab\}$, e a

Tabela 3.1: Passo 1 da abordagem **Red1.1**.

α (χ - ϕ)	β (ϕ - γ)	ω	Conjunto resultante
$aaaab$ (aaa - ab)	$abab$ (ab - ab)	$aaaabab$	$\{aaaabab, aaab, aabab, aabb, abb, baab, bab, bbab, bbb\}$
$aaaabab$ ($aaaab$ - ab)	abb (ab - b)	$aaaababb$	$\{aaaababb, aaab, aabab, aabb, baab, bab, bbab, bbb\}$
$aaab$ (aaa - b)	$baab$ (b - aab)	$aaabaab$	$\{aaaababb, aaabaab, aabab, aabb, bab, bbab, bbb\}$
bbb (bb - b)	$bbab$ (b - bab)	$bbbbab$	$\{aaaababb, aaabaab, bbbbab, aabab, aabb, bab\}$
$aabab$ ($aaba$ - b)	$bbbbab$ (b - $bbbab$)	$aababbbbab$	$\{aaaababb, aaabaab, aababbbbab, aabb, bab\}$
$aaaababb$ ($aaaabab$ - b)	bab (b - ab)	$aaaababbab$	$\{aaabaab, aababbbbab, aabb, aaaababbab\}$

seqüência $\alpha = aaaababbab$, tem-se que apenas o prefixo $\beta = aaaabab$ é necessário para manter a completude do conjunto. Analisando as seqüências restantes, tem-se que apenas mais uma seqüência pode ser reduzida. Toma-se a seqüência $\alpha = aababbbbab$, tem-se que apenas o prefixo $\beta = aababbbb$ é necessário para manter a completude do conjunto.

Ao fim da execução dos passos 1 e 2, o conjunto final e reduzido da abordagem é

$$TS_{Red1.1} = \{raaabaab, raababbbb, raabb, raaaabab\}$$

de tamanho 30 e com 4 operações *reset*.

3.4.2 Exemplo da Abordagem Red1.2

A abordagem **Red1.2** considera, além da combinação por sobreposição, a combinação utilizando seqüências de transferência. Primeiramente obtém-se um conjunto em que todas as combinações possíveis por sobreposição são realizadas. Sendo assim, para o exemplo, tem-se como resultado dessa etapa o conjunto $TS_{Esc_1Comb_1} = \{raaabaab, raababbbbab, raabb, raaaababbab\}$ obtido na Seção 3.4.1. A partir desse conjunto, seqüências de transferência são usadas para combinação.

Toma-se $\alpha = aaaababbab$ e $\beta = aababbbbab$. Nesse caso, uma seqüência de transferência χ deve ser inserida entre α e β. Tem-se então $\delta(s_0, \alpha) = s_2$, o que indica que χ deve ser uma seqüência que leva a MEF do estado s_2 ao estado inicial s_0, resultando na seqüência $\chi = a$. Com isso, tem-se a seqüência $\omega = \alpha\chi\beta$, que resulta na seqüência $\omega = aaaababbabaaababbbbab$. Tem-se então o conjunto $TS = \{raaaababbabaaababbbbab, raaabaab, raabb\}$. Verifica-se que o conjunto satisfaz as condições do Teorema 1. Repete-se o procedimento no conjunto, que nesse caso possui mais três seqüências para serem combinadas. Nesse exemplo, mas nenhuma combinação mantém

a completude do conjunto, sendo assim, tem-se, como resultado do **passo 1**, o conjunto $TS_{Esc_1Comb_2} = \{raaaababbabaaababbbbab, raaabaab, raabb\}$.

A partir do conjunto $TS_{Esc_1Comb_2}$, executa-se o **passo 2** do algoritmo. Por exemplo, tomando-se o conjunto $TS_{Esc_1Comb_2} = \{raaaababbabaaababbbbab, raaabaab, raabb\}$ e a seqüência $\alpha = aaaababbabaaababbbbab$, tem-se que apenas o prefixo $\beta = aaaababbabaaababbbb$ é necessário para manter a completude do conjunto. Para a seqüência $\alpha = aaabaab$ apenas o prefixo $\beta = aaab$ é necessário para a manutenção da completude. Já a seqüência $\alpha = aabb$ não pode ser reduzida. Dessa forma, o resultado final da redução dessa abordagem é o conjunto

$$TS_{Red1.2} = \{raaaababbabaaababbbb, raaab, raabb\}$$

de tamanho 30 e com 3 operações *reset*.

3.4.3 Exemplo da Abordagem Red2.1

Essa abordagem permite apenas a combinação em pares e com uso apenas de sobreposição na combinação das seqüências. No **passo 1**, deve-se combinar todas as seqüências em que é possível a sobreposição e que ainda não tenham sido combinadas. Por exemplo, toma-se $\alpha = abab$ e $\beta = abb$, que podem ser combinadas por sobreposição, resultando na seqüência $\omega = ababb$, resultando no conjunto $TS = \{rababb, raaaab, raaab, raabab, raabb, rbaab, rbab, rbbab, rbbb\}$. Verifica-se que esse conjunto satisfaz as condições do Teorema 1. Sendo assim, o conjunto é submetido a uma nova escolha para combinação, porém a seqüência ω não pode ser escolhida. Dessa forma, a escolha só pode ser feita no conjunto $TS_{naoComb} = \{raaaab, raaab, raabab, raabb, rbaab, rbab, rbbab, rbbb\}$. A Tabela 3.2 apresenta as sobreposições possíveis nesse exemplo.

Tabela 3.2: Passo 1 da abordagem **Red2.1**.

α (χ - ϕ)	β (ϕ - γ)	ω	Conjunto de seqüências não combinadas
$abab$ (ab - ab)	abb (ab - b)	$ababb$	$\{aaaab, aaab, aabab, aabb, baab, bab, bbab, bbb\}$
$aaab$ (aaa - b)	$baab$ (b - aab)	$aaabaab$	$\{aaaab, aabab, aabb, bab, bbab, bbb\}$
$aabab$ ($aaba$ - b)	bbb (b - bb)	$aababbb$	$\{aaaab, aabb, bab, bbab\}$
bab (ba - b)	$bbab$ (b - bab)	$babbab$	$\{aaaab, aabb\}$

Nota-se, a partir da Tabela 3.2 que ainda sobram duas seqüências para serem combinadas. Porém, não é possível combinar essas seqüências por sobreposição. Como

essa abordagem não permite que nenhuma outra seqüência já combinada seja novamente combinada, tem-se como resultado do **passo 1** o conjunto $TS_{Esc2Comb1} = \{rababb, raaabaab, raababbb, rbabbab, raaaab, raabb\}$.

Com a aplicação do **passo 2**, toma-se $\alpha = aaabaab$ e verifica-se que o prefixo $\beta = aaaba$ mantém a completude do conjunto. Há casos em que a seqüência toda pode ser removida, como por exemplo, tomando-se a seqüência $\alpha = aaaab$, em que o prefixo $\beta = \epsilon$ mantém a completude, ou seja, pode-se descartar a seqüência α. Na Tabela 3.3 é apresentada a ordem das escolhas das seqüências para o passo 2.

Tabela 3.3: Passo 2 da abordagem **Red2.1**.

α	β	Conjunto resultante
$aaabaab$	$aaaba$	$\{ababb, aababbb, babbab, aaaab, aabb, aaaba\}$
$aaaab$	ϵ	$\{ababb, aababbb, babbab, aabb, aaaba\}$
$ababb$	$abab$	$\{abab, aababbb, babbab, aabb, aaaba\}$
$aababbb$	$aababb$	$\{abab, aababb, babbab, aabb, aaaba\}$
$babbab$	$babbab$	$\{abab, aababb, babbab, aabb, aaaba\}$
$aabb$	$aabb$	$\{aaaba, abab, aababb, babbab, aabb\}$

Com isso, obtém-se o conjunto final reduzido

$$TS_{Red2.1} = \{raaaba, rabab, raababb, rbabbab, raabb\}$$

de tamanho 30 e com 5 operações *reset*.

3.4.4 Exemplo da Abordagem Red2.2

Essa abordagem permite a combinação em pares com uso de sobreposição e seqüências de transferência. No **passo 1**, ao se combinar todas as seqüências em que é possível a sobreposição, tem-se o conjunto $TS_{Esc2Comb1} = \{rababb, raaabaab, raababbb, rbabbab, raaaab, raabb\}$, obtido na Seção 3.4.3 como resultado da combinação por sobreposição. Dessa forma, deve-se escolher apenas seqüências que ainda não foram combinadas para realização da combinação por meio de seqüências de transferência. Sendo assim, a escolha de seqüências deve ser feita dentre as seqüências do conjunto $T_{naoComb} = \{raaaab, raabb\}$, conforme apresentado na última linha da Tabela 3.2. Toma-se então a seqüência $\alpha = aaaab$ e a seqüência $\beta = aabb$. Como $\delta(s_0, \alpha) = s_0$, deve-se usar uma seqüência de transferência χ que leve a MEF do estado s_0 ao próprio estado s_0, que no caso é a seqüência vazia. Dessa forma, usa-se $\chi = \epsilon$, resultando em $\gamma = \alpha\chi\beta = aaaabaabb$. Com isso, verifica-se que o conjunto $TS = \{rababb, raaabaab, raababbb, rbabbab, raaaabaabb\}$ satisfaz as condições do Teorema

1. Como todas as seqüências já foram combinadas, não há mais possibilidade de combinação. Portanto, o **passo 1** da abordagem **Red2.2** tem como resultado o conjunto $TS_{Esc2Comb2} = \{rababb, raaabaab, raababbb, rbabbab, raaaabaabb\}$.

Na Tabela 3.4 é apresentada a ordem das escolhas da seqüências para o passo 2.

Tabela 3.4: Passo 2 da abordagem **Red2.2**.

α	β	Conjunto resultante
$ababb$	$abab$	$\{aaabaab, aababbb, babbab, aaaabaabb, ababb\}$
$aaabaab$	ϵ	$\{aababbb, babbab, aaaabaabb, abab\}$
$aababbb$	$aababb$	$\{babbab, aaaabaabb, abab, aababb\}$
$babbab$	$babbab$	$\{aaaabaabb, abab, aababb, babbab\}$
$aaaabaabb$	$aaaabaabb$	$\{abab, aababb, babbab, aaaabaabb\}$

Com isso, após a aplicação do **passo 2**, tem-se o conjunto final reduzido

$$TS_{Red2.2} = \{rabab, raababb, rbabbab, raaaabaabb\}$$

de tamanho 33 e com 4 operações *reset*.

3.4.5 Exemplo da Abordagem Red3.1

Por possibilitar mais iterações no processo de escolha das combinações por pares, essa abordagem toma como resultado da primeira iteração o conjunto $TS_{Esc2Comb1} = \{rababb, raaabaab, raababbb, rbabbab, raaaab, raabb\}$ obtido na Seção 3.4.3. Esse conjunto é submetido ao mesmo processo **Esc2** de escolhas de seqüências, porém com o uso apenas de sobreposição na combinação. Por exemplo, toma-se a combinação de $\alpha = aaaab$ e $\beta = ababb$, que é passível de sobreposição e resulta em $\omega = aaaababb$, resultando no conjunto $T = \{raaaababb, raaabaab, raababbb, rbabbab, raabb\}$, que satisfaz as condições do Teorema 1. Sendo assim, de acordo com **Esc2** tem-se mais quatro seqüências que podem ser combinadas. Neste caso, apenas mais uma combinação por sobreposição é possível, com $\alpha = aababbb$ e $\beta = babbab$, resultando em $\omega = aababbbabbab$. Dessa forma, tem-se como resultado da segunda iteração o conjunto $T = \{raababbbabbab, raaaababb, raaabaab, raabb\}$. Depois de terminada cada iteração, uma nova etapa pode ser iniciada de acordo com **Esc2**. Nesse exemplo, não é mais possível a combinação a partir do conjunto resultante da segunda iteração, portanto, tem-se como resultado do **passo 1** o conjunto $TS_{Esc3Comb1} = \{raababbbabbab, raaaababb, raaabaab, raabb\}$.

Na Tabela 3.5 é apresentada a ordem das escolhas da seqüências para o passo 2.

Com isso, após a aplicação do **passo 2**, tem-se o conjunto final reduzido

$$TS_{Red3.1} = \{raaaba, raababbbabb, raaaabab, raabb\}$$

Tabela 3.5: Passo 2 da abordagem **Red3.1**.

α	β	Conjunto resultante
$aaabaab$	$aaaba$	$\{aababbbabbab, aaaababb, aabb, aaaba\}$
$aababbbabbab$	$aababbbabb$	$\{aaaababb, aabb, aaaba, aababbbabb\}$
$aaaababb$	$aaaabab$	$\{aabb, aaaba, aababbbabb, aaaabab\}$
$aabb$	$aabb$	$\{aaaba, aababbbabb, aaaabab, aabb\}$

de tamanho 30 e com 4 operações *reset*.

3.4.6 Exemplo da Abordagem Red3.2

Por possibilitar mais iterações no processo de escolha das combinações por pares, essa abordagem toma como resultado da primeira iteração o conjunto $TS_{Esc2Comb2} = \{rababb, raaabaab, raababbb, rbabbab, raaaabaabb\}$ obtido na Seção 3.4.4. Tem-se como resultado da segunda iteração o conjunto $T = \{raaabaababb, raababbbabbab, raaaabaabb\}$. A terceira iteração também é possível, devido ao uso de seqüências de transferência. Como resultado da terceira iteração tem-se o conjunto $T = \{raababbbabbab, raaaabaabbaaaabaababb\}$. A quarta iteração não é possível, pois nenhuma combinação resulta em um conjunto completo, sendo assim tem-se como resultado do **passo 1** o conjunto $T_{Esc3Comb2} = \{raababbbabbab, raaaabaabbaaaabaababb\}$.

Com a aplicação do **passo 2**, tem-se que apenas a seqüência $\alpha = aaaabaabbaaaabaababb$ pode ser reduzida, em que o prefixo $\beta = aaaabaabbaaaabaabab$ é necessário para manter a completude do conjunto. Com isso, tem-se o conjunto final reduzido

$$T_{Red3.2} = \{raababbbabbab, raaaabaabbaaaabaabab\}$$

de tamanho 33 e com 2 operações *reset*.

3.4.7 Sumarização dos Exemplos

Diferentes resultados podem ser obtidos com a execução de uma mesma MEF e um mesmo conjunto de seqüências, dependendo da ordem em que as seqüências são consideradas. Para evitar que o algoritmo tenha o desempenho influenciado pela ordem das seqüências, elas são selecionadas de forma aleatória. Executando-se 10 vezes as abordagens de redução com o exemplo apresentado, obtém-se como resultado os valores médios apresentados na Tabela 3.6, em que são considerados a média do número de *resets* e o tamanho dos conjuntos obtidos na redução. Deve-se ressaltar que os conjuntos de seqüências de testes obtidos possuem o mesmo poder de revelar defeitos modelados pelo domínio de defeitos.

Tabela 3.6: Sumarização dos resultados dos exemplos.

Abordagem	Tamanho do conjunto	Número de *resets*
Red1.1	30,3	3,9
Red1.2	30,1	2,6
Red2.1	32,0	5,5
Red2.2	28,4	4,0
Red3.1	30,3	4,0
Red3.2	33,9	1,6

Os resultados obtidos nos exemplos referem-se às reduções obtidas no conjunto gerado pelo método HSI. Nota-se pelos dados da Tabela 3.6 e pelos dados da geração pelos métodos W, Wp, HSI e H, apresentados na Seção 2.7.1, que os conjuntos reduzidos também são menores que os conjuntos gerados pelos métodos W, Wp, que são conjuntos de tamanho 43 e 39, respectivamente, e com 7 operações *resets* em ambos. Em relação aos métodos H e SPY, as reduções do tamanho do conjunto se aproximam do conjunto obtido por esses métodos, já o número de operações *resets* só é maior considerando a abordagem **Red2.1**.

3.5 Aspectos de Implementação

Em termos de aplicações computacionais, neste trabalho de mestrado foram desenvolvidas duas ferramentas: (1) Verificador das condições de suficiência propostas em Simao e Petrenko (2010) e o (2) Implementação das abordagens de redução de conjuntos de seqüências de teste. Ambas foram implementadas na linguagem Java, o que torna as ferramentas portáveis, podendo ser executadas independente do sistema operacional.

A seguir, são apresentados os aspectos de implementação dessas duas ferramentas, bem como a maneira de se executar e o formato dos dados de entrada.

3.5.1 Verificador das Condições de Suficiência: CheSCon

Essa ferramenta foi desenvolvida em conjunto com outro aluno do grupo de pesquisa do ICMC/USP e foi denominada **CheSCon** (*Check Sufficient Conditions*). Ela tem como objetivo verificar se determinado conjunto de seqüências de teste satisfaz as condições de suficiência propostas por Simao e Petrenko (2010) para determinada MEF, ou seja, se satisfaz as condições do Teorema 1. A ferramenta foi implementada de acordo com o Algoritmo 1 e recebe como entrada uma MEF e um conjunto de seqüências de teste. Os formatos de entrada da MEF e do conjunto de teste são apresentados na Figura 3.2, os quais exemplificam a MEF da Figura 2.2 e um conjunto de seqüências qualquer. Cada linha do formato de entrada da MEF representa uma transição no seguinte formato:

estadoOrigem – entrada / saída -> estadoDestino. Por exemplo, a transição (s_1, a) é representada pela *string* `s1 - a / 1 -> s0`, cujos estados de origem e destino são s_1 e s_0, respectivamente, e a entrada e a saída os símbolos a e 1, respectivamente.

`s0 - a / 1 -> s3`	`a a a b b`
`s0 - b / 1 -> s2`	`a a b a b`
`s1 - a / 1 -> s0`	`a a b b a b`
`s1 - b / 1 -> s1`	`a b a b`
`s2 - a / 0 -> s0`	`a b b b a a`
`s2 - b / 0 -> s0`	`b a a b b a`
`s3 - a / 1 -> s1`	`b a b a`
`s3 - b / 0 -> s0`	`b b a b a b`

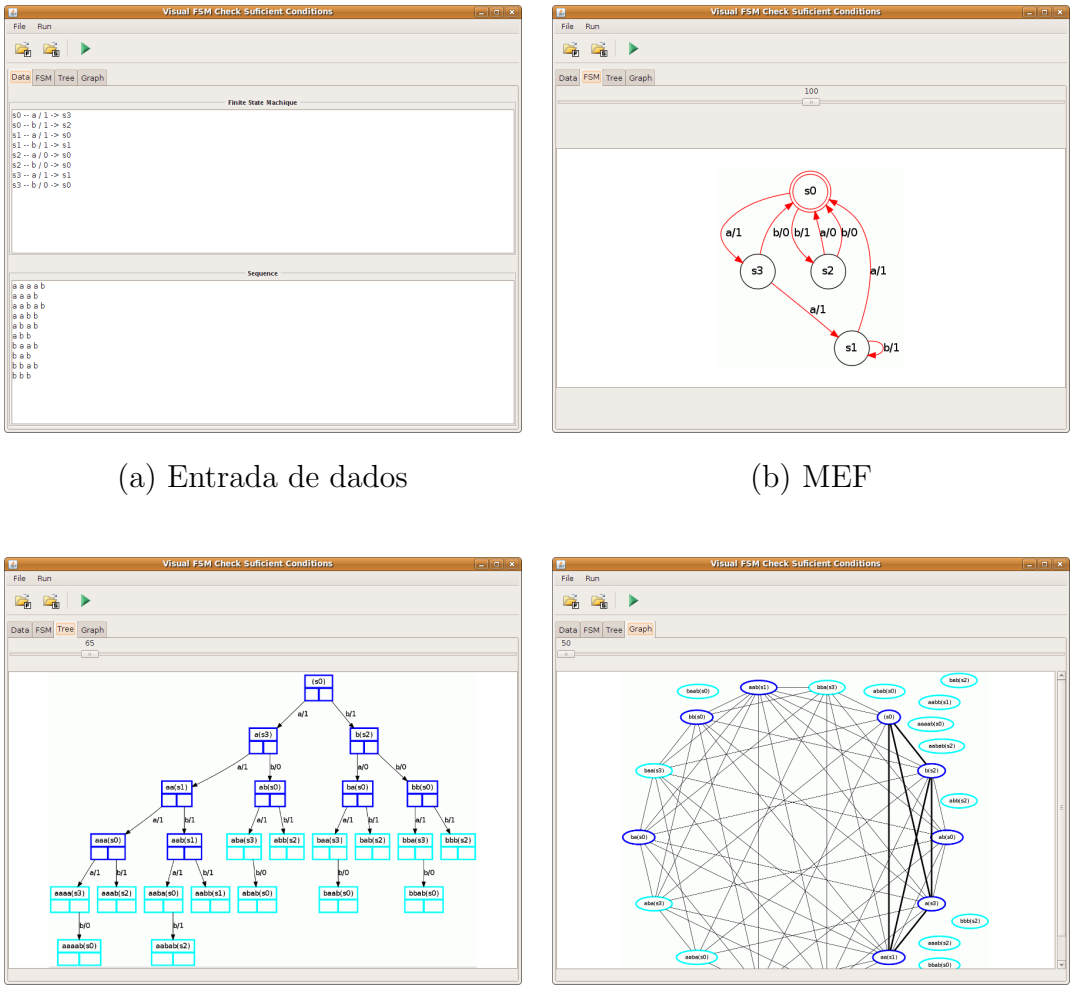
Figura 3.2: Formatos de entrada.

A MEF é armazenada em uma matriz estática de acordo com sua representação em tabela de transição de estados. O conjunto de seqüências de teste é armazenado em uma lista seqüencial. Os símbolos de entrada e saída, e o nome dos estados são do tipo *String*. Conforme apresentado na Seção 2.7.2, o algoritmo do processo de verificação das condições do Teorema 1 faz uso de uma árvore e de um grafo. A árvore é armazenada por meio de listas que são criadas dinamicamente após se ter conhecimento do número de entradas na MEF e do conjunto de teste. Já o grafo é armazenado por meio da representação de listas dinâmicas de adjacência.

Para se executar a **CheSCon**, informa-se a MEF e o conjunto de seqüências de teste, como primeiro e segundo parâmetros. Saída `0` indica que o conjunto satisfaz as condições do Teorema 1 e saída `255` indica o contrário. Como exemplo de execução, tem-se: `java -jar chescon mef conj`. A ferramenta **CheSCon** foi utilizada na elaboração de um método de geração de seqüência de verificação baseado em algoritmo genético (Ribeiro et al., 2009).

Com o objetivo de auxiliar as pesquisas neste trabalho de mestrado, uma versão gráfica, denominada **CheSConViz**, também foi implementada. Nessa ferramenta é utilizado o aplicativo *Graphviz* para a geração das imagens da MEF, árvore e grafo. A Figura 3.3 apresenta a interface da **CheSConViz**.

Por meio da interface gráfica pode-se alterar o conjunto ou a MEF de maneira dinâmica, verificando o que acontece nas estruturas de dados envolvidas no algoritmo. Também é possível verificar as transições cobertas na MEF e saber quais seqüências estão confirmadas e quais os lemas, apresentados na Seção 2.7.2, responsáveis pelas confirmações.



(a) Entrada de dados

(b) MEF

(c) Árvore de teste

(d) Grafo de distinção

Figura 3.3: Interfaces da ferramenta *CheSConViz*.

3.5.2 Abordagens de Redução: RedTest

Essa ferramenta, denominada **RedTest**, tem como objetivo reduzir um conjunto de seqüências teste de acordo com as abordagens descritas na Seção 3.3. Como o algoritmo das abordagens de redução necessita verificar as condições do Teorema 1, a implementação da **CheSCon** foi utilizada como base, porém com algumas modificações para aumentar o desempenho, conforme discutidas a seguir.

As mesmas estruturas de dados utilizadas na **CheSCon** são utilizadas nessa implementação. Para se executar a **RedTest** passa-se como parâmetro uma MEF e um conjunto de teste de acordo com o formato exibido na Figura 3.2. Outros parâmetros também devem ser passados, como a abordagem de redução a ser utilizada e a semente que controla a aleatoriedade da escolha das seqüências. A **RedTest** retorna como saída o conjunto de

teste reduzido. Um exemplo de execução da abordagem **Red3.1** e com semente 5 é: `java -jar redtest mef conjunto 3.1 5`.

Como o processo de redução consiste em combinar duas seqüências do conjunto de teste, não é necessário refazer toda a árvore e o grafo com o conjunto gerado após a combinação. Basta apenas modificá-los dinamicamente. Por exemplo, escolhendo as seqüências α e β para combinação, deve-se remover da árvore e do grafo a seqüência β, uma vez que não é necessária a remoção de α, pois ela vai se tornar prefixo da seqüência ω, resultado da combinação. No caso do grafo, além da remoção da seqüência β, deve-se remover também as ligações em que β implicava. Feito isso, insere-se na árvore e no grafo a seqüência ω. Com isso, atualiza-se a árvore e o grafo, e o processo de verificação das condições do Teorema 1 pode ser executado.

Outro ponto em que a mudança dinâmica do grafo e da árvore se torna vantajosa é o fato da busca do clique. Em muitos casos a substituição de α e β por ω não modifica o grafo a ponto de também modificar o clique. Com isso, após a modificação do grafo, verifica-se se o clique encontrado antes da modificação ainda é válido. Caso seja válido, ele é mantido; caso contrário, o processo normal de busca de clique é executado. Isso gera um ganho de desempenho, pois muitas vezes não é necessária a execução do algoritmo de busca de cliques.

3.6 Considerações Finais

Ao longo deste capítulo foi apresentada a estratégia de redução, bem como todas as abordagens nela baseadas. Para cada abordagem proposta, um exemplo foi apresentado. Os aspectos de implementação das ferramentas desenvolvidas também foram apresentados, assim como os formatos de entradas e formas de execução.

No capítulo seguinte é apresentada uma avaliação experimental das abordagens de redução propostas. O objetivo da avaliação foi identificar as características de cada abordagem, destacando suas particularidades e cenários para aplicação de cada uma.

Capítulo

4

Avaliação Experimental

4.1 Considerações Iniciais

Com o objetivo de avaliar as abordagens propostas, estudos experimentais foram conduzidos de forma a verificar a redução do tamanho do conjunto e do número de operações *resets*. Para isso, quatro estudos foram realizados. O primeiro, descrito na Seção 4.2 trata de experimentos conduzidos com MEFs aleatórias e reduções aplicadas em conjuntos de teste gerados pelos métodos W, Wp, HSI e H. O segundo, descrito na Seção 4.3 também é conduzido com MEFs aleatórias e em conjuntos gerados pelo método HSI, com o intuito é observar como as abordagens se comportam com variações nos parâmetros das MEFs. Na Seção 4.4 é apresentado o terceiro estudo, o qual compara os resultados obtidos nas abordagens de redução com os resultados de uma estratégia de minimização proposta por Neto (2008). Por fim o quarto experimento, descrito na Seção 4.6, se trata de estudos de caso, em que as abordagens de redução são aplicadas em um *benchmark* e em especificações reais de protocolos.

A metodologia utilizada nos estudos experimentais pode ser descrita pelas seguintes etapas:

1. Seleção/Geração da MEF

2. Geração dos casos de teste por meio de métodos de geração
3. Redução dos casos de teste por meio das abordagens propostas
4. Comparação do tamanho e do número de *resets* de cada conjunto gerado na etapa 2 em relação ao conjunto reduzido na etapa 3
5. Análise dos resultados

Os experimentos consistem na comparação dos conjuntos de casos de teste gerados com os conjuntos obtidos na redução. Os principais objetos de comparação são o tamanho do conjunto reduzido e o número de operações *resets*. Além desses o tempo de execução de cada abordagem também é analisado.

4.2 Redução de Conjuntos Gerados por Métodos Clássicos de Geração

A utilização de MEFs aleatórias permite que se tenha uma caracterização do comportamento médio das gerações e reduções, contrapondo-se aos valores teóricos disponíveis, que normalmente representam apenas o comportamento no pior caso. A utilização de MEFs aleatórias é freqüentemente adotada para essa caracterização (Dorofeeva et al., 2005b; Hierons e Ural, 2006; Sidhu e kau Leung, 1989; Simao et al., 2007).

O processo de geração das MEFs, descrito por Simao e Petrenko (2010), é realizado em três etapas: Na primeira etapa, um estado é selecionado como estado inicial e marcado como alcançável. Então, para cada estado s não marcado como alcançável, o gerador aleatório seleciona um estado alcançável s', uma entrada x e uma saída y e adiciona uma transição de s' para s com entrada x e saída y, marcando s como alcançável. Feito isso, tem-se início a segunda etapa, em que o gerador adiciona, se necessário, mais transições aleatórias. Na terceira etapa, é verificado se a MEF é minimal. Caso não seja minimal, a MEF é descartada e outra MEF é gerada.

Esse experimento foi conduzido com o objetivo de analisar o comportamento das abordagens de redução com conjuntos gerados por outros métodos de geração em MEFs geradas aleatoriamente. Para isso, foram utilizados conjuntos gerados pelos métodos W, Wp, HSI e H. Esses conjuntos foram gerados a partir de 30 MEFs com 10 estados, 3 entradas e 3 saídas. Na Tabela 4.1 são apresentados os valores médio do tamanho do conjunto para cada método e os valores médio obtidos na redução para cada abordagem. A mesma informação, porém com respeito a redução de *resets* é apresentada na Tabela 4.2

Quanto às taxas de redução, são apresentados nas Tabelas 4.3 e 4.4 os valores para redução do tamanho do conjunto e do número de *resets*, respectivamente.

Tabela 4.1: Média do tamanho dos conjuntos.

Método	Tamanho do conjunto	Red1.1	Red1.2	Red2.1	Red2.2	Red3.1	Red3.2
W	521	267	477	267	239	265	512
Wp	406	284	313	281	236	281	373
HSI	264	210	248	210	177	211	242
H	187	174	181	173	174	173	186

Tabela 4.2: Média do número de *resets*.

Método	Número de *resets*	Red1.1	Red1.2	Red2.1	Red2.2	Red3.1	Red3.2
W	72	36	7	36	24	35	7
Wp	58	39	15	39	26	39	13
HSI	51	40	16	40	26	40	13
H	33	29	20	30	22	29	19

Tabela 4.3: Taxa de redução do tamanho do conjunto.

Método	Red1.1	Red1.2	Red2.1	Red2.2	Red3.1	Red3.2
W	48,7%	8,6%	48,8%	54,2%	49,2%	1,7%
Wp	30,0%	22,8%	30,8%	41,9%	30,8%	8,1%
HSI	20,5%	6,0%	20,3%	33,1%	20,2%	8,5%
H	7,2%	3,4%	7,4%	7,0%	7,7%	0,7%

Tabela 4.4: Taxa de redução do número de *resets*.

Método	Red1.1	Red1.2	Red2.1	Red2.2	Red3.1	Red3.2
W	50,1%	90,5%	50,3%	66,2%	50,8%	89,7%
Wp	32,1%	75,0%	32,6%	55,6%	32,8%	78,2%
HSI	22,2%	67,6%	21,7%	48,9%	22,0%	73,5%
H	10,7%	38,1%	7,0%	32,9%	11,2%	42,8%

Nota-se que o método H é o que menos reduz, já o método W apresenta as maiores taxas de redução. Isso acontece devido ao fato do método H gerar conjuntos menores, além de serem conjuntos com menos redundâncias que os outros métodos. Já, o método W, gera os maiores conjuntos entre os 4 métodos utilizados, o que conseqüentemente apresenta maior redundância e possibilidade de redução. Os métodos Wp e HSI apresentam valores médios e similares. Com isso, na análise dos dados desse experimento e nos experimentos das seções seguintes, são utilizados apenas conjuntos gerados pelo método HSI.

Os gráficos dos experimentos a seguir apresentam a relação entre os principais resultados coletados durante esse estudo experimental. Para facilitar a visualização, apenas 10 resultados de cada abordagem foram considerados na construção dos gráficos.

A Figura 4.1 apresenta a relação da redução do tamanho do conjunto com a redução do número de *resets*. Os dados são das taxas de redução de conjuntos gerados pelo método HSI. Nota-se que as abordagens **Red1.1**, **Red2.1** e **Red3.1** se agrupam de maneira muito semelhante. Com isso, pode-se dizer que essas três abordagens são equivalentes. A abordagem **Red2.2** apresenta bons resultados em ambas as reduções. Já as abordagens **Red1.2** e **Red3.2** apresentam resultados muito bons para redução de *resets*, porém a redução do tamanho do conjunto não é tão boa, sendo que em algumas situações o conjunto final é até maior que o original.

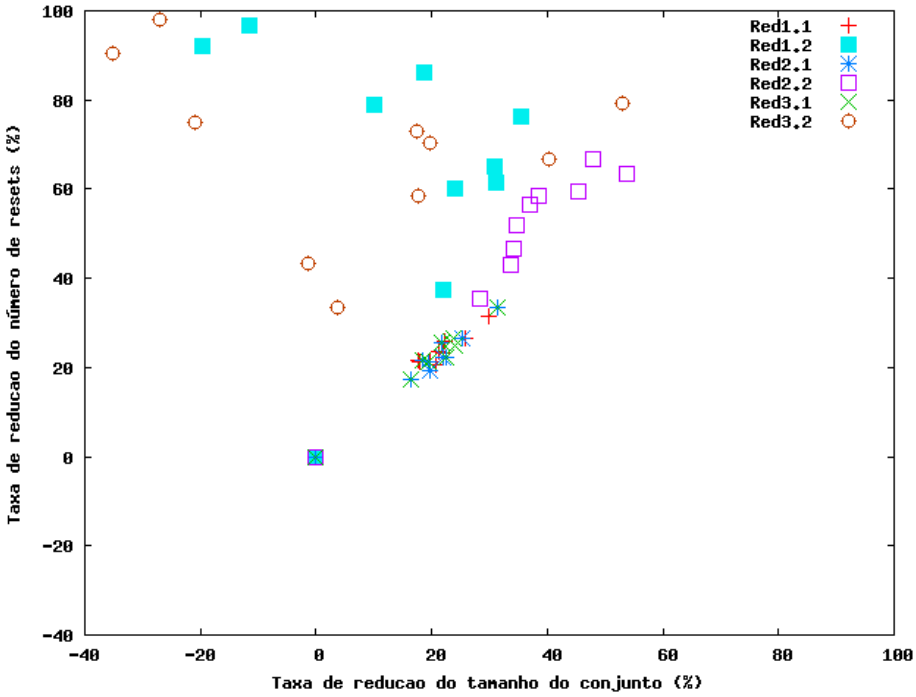


Figura 4.1: Tamanho do conjunto × Número de *resets*.

Quanto ao tempo de execução, a Figura 4.2 apresenta a relação do tamanho do conjunto em função do tempo de execução, medido em milissegundos. Nota-se que as abordagens **Red1.2** e **Red3.2** apresentam os maiores tempos e em alguns casos a redução não é significativa. Na Figura 4.3 é apresentado a relação da redução de *resets* em função do tempo, em que a redução de *resets* dessas mesmas duas abordagens é muito significativo. Com isso, tem-se que as abordagens que são melhores para reduzir *resets* tem, em geral, um maior tempo de execução. As abordagens **Red1.1**, **Red2.1** e **Red3.1** apresentam tempo de execução também parecidos, o que reforça a afirmação de que elas são equiva-

lentes. Os tempos de execução foram medidos em um computador *Intel(R) Core(TM)2 Duo, 3.16GHz* no sistema operacional GNU/Linux.

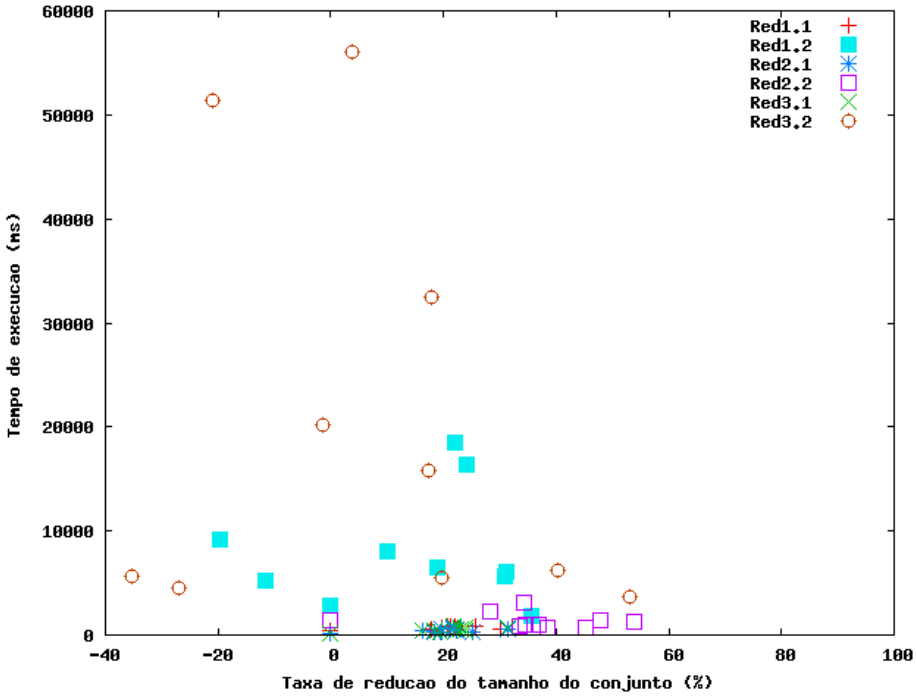


Figura 4.2: Tamanho do conjunto × Tempo.

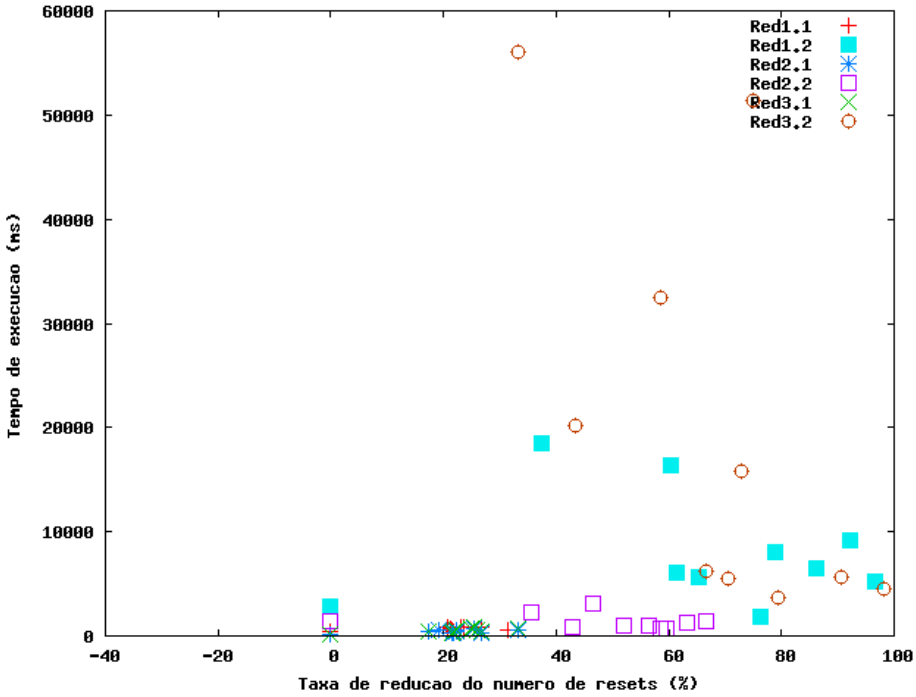


Figura 4.3: Número de *resets* × Tempo.

As Figuras 4.4 e 4.5 são dois gráficos boxplot (McGill et al., 1978), que representam a distribuição dos dados das taxas de redução de cada abordagem para o tamanho do conjunto e para o número de *resets*, respectivamente. Um gráfico boxplot é construído a partir dos valores máximo e mínimo da amostragem de dados, representados pelas hastes horizontais. A caixa representa maior agrupamento de dados e é delimitada por quartis, indicados pelas suas linhas horizontais. A linha horizontal superior indica o primeiro quartil, que é o valor que deixa um quarto dos valores abaixo e três quartos acima dele. A linha horizontal inferior indica o terceiro quartil, que é o valor que deixa três quartos dos dados abaixo e um quarto acima dele. O segundo quartil é a mediana dos dados. A linha vertical representa a dispersão dos dados que estão fora dos quartis. Os dados discrepantes, representados por pontos, são chamados de *outliers* e indicam os dados que saem completamente fora do padrão da amostragem. Nota-se que nas abordagens **Red1.2** e **Red3.2**, muitas vezes o tamanho do conjunto se torna maior que o conjunto original, porém o número de *resets* para essas duas abordagens tem redução muito significativa. Observa-se também que as abordagens **Red1.1**, **Red2.1** e **Red3.1** apresentam dispersão parecidas. Em poucos casos não houve redução de *resets*, qualquer que seja a abordagem, sendo verificados apenas alguns *outliers* inferiores, indicando taxa zero de redução.

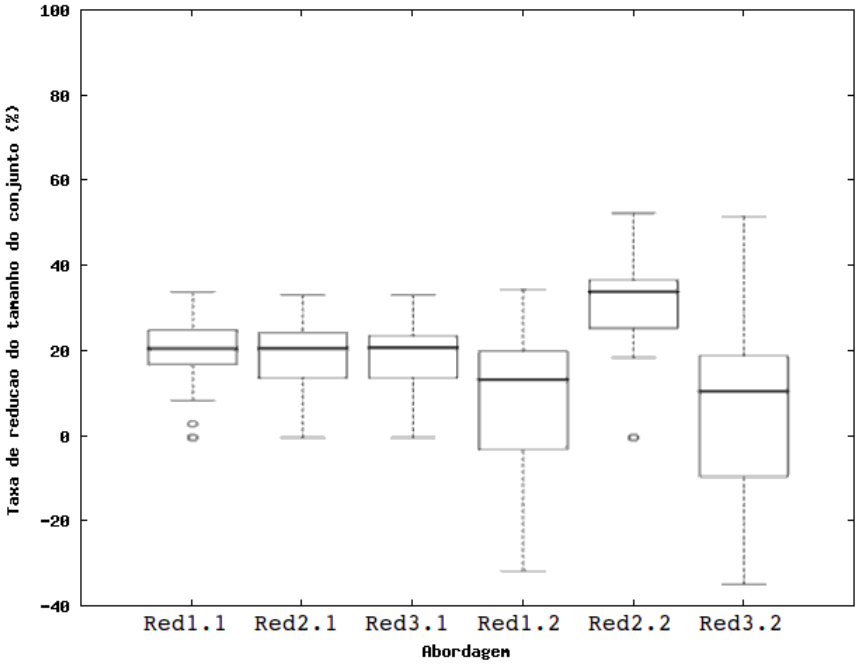


Figura 4.4: Boxplots da taxa de redução do tamanho do conjunto.

Como já discutido e observado por meio dos dados obtidos, tem-se que as abordagens **Red1.1**, **Red2.1** e **Red3.1** apresentam resultados muito parecidos. Dessa forma, pode-se dizer que se tratam de abordagens equivalentes. Nos próximos experimentos as abordagens

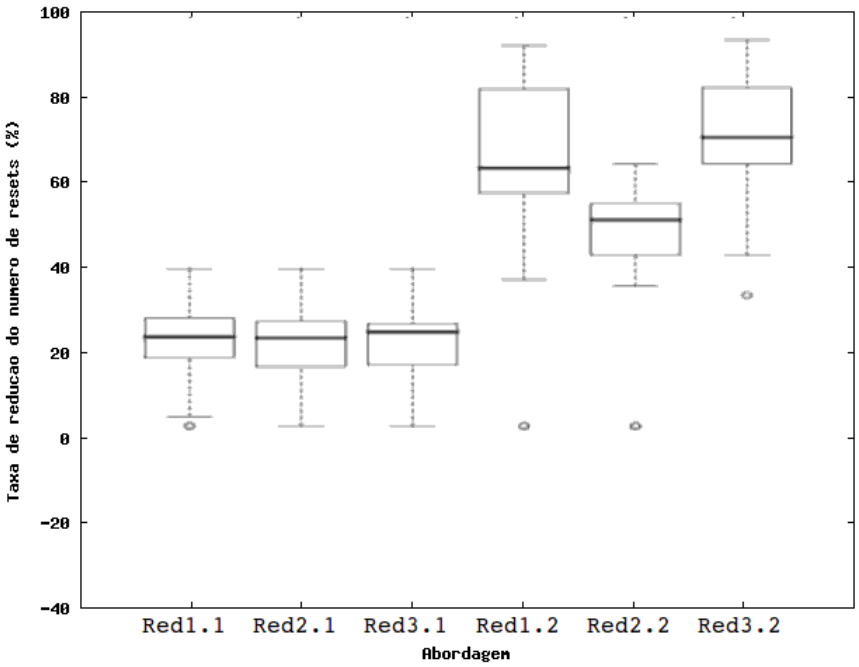


Figura 4.5: Boxplot da taxa de redução do número de *resets*.

Red2.1 e **Red3.1** não são utilizadas, pois seus resultados podem ser inferidos por meio da abordagem **Red1.1**.

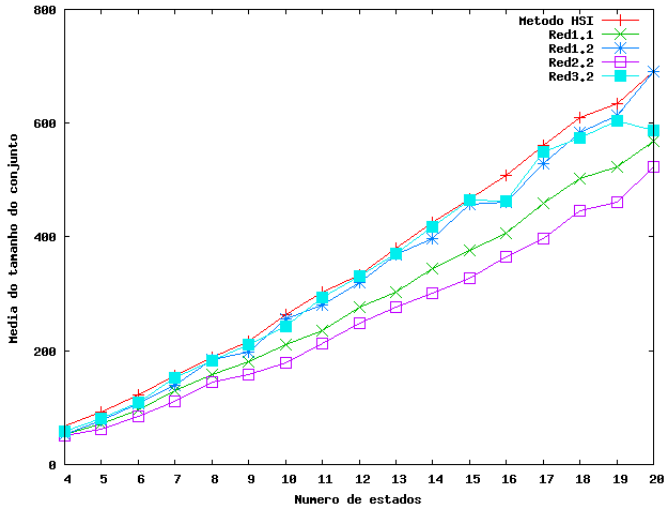
4.3 Redução de Conjuntos Considerando Variações nos Parâmetros das MEFs

As abordagens **Red1.1**, **Red1.2**, **Red2.2** e **Red3.2** foram submetidas a três tipos de experimentos, em que a quantidade de estados, entradas e saídas são variadas, com o objetivo de observar a influência na redução com a variação desses parâmetros. Os conjuntos gerados para realizar a redução foram obtidos a partir do método HSI em MEFs aleatórias. A seguir os resultados dos experimentos são apresentados e discutidos.

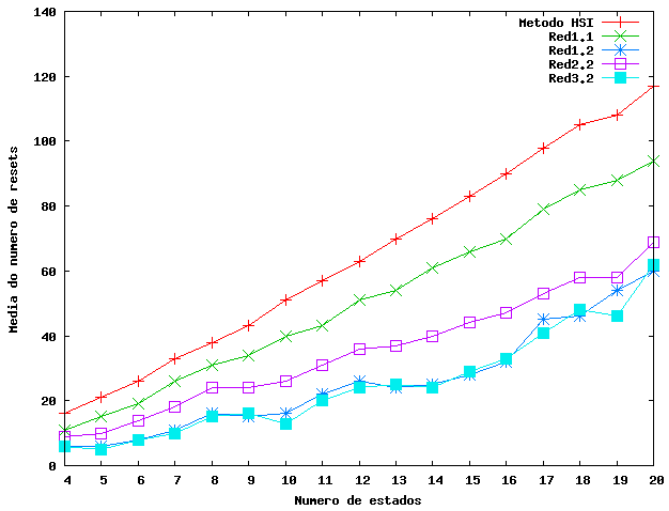
4.3.1 Variação no Número de Estados

Para esse experimento foram geradas MEFs completas com número de estados variando entre 4 e 20 e com número de entradas e saídas fixados em 3, o que resulta em 17 configurações diferentes de MEFs. Para cada configuração foram geradas 30 MEFs, totalizando 510 MEFs ao total para esse experimento. Portanto, para cada uma das 510 MEFs, um conjunto de seqüências de teste foi gerado pelo método HSI. Os 510 conjuntos foram submetidos às 4 abordagens de redução descritas na início desta seção. Dessa forma, 2040 reduções foram realizadas.

No gráfico da Figura 4.6.a são apresentadas as variações nas médias do tamanho dos conjuntos obtidos na redução dos conjuntos gerados pelo método HSI. Já no gráfico da Figura 4.6.b são apresentadas as médias do número de operações *resets*. As abordagens **Red1.2** e **Red3.2** apresentam as maiores taxas de redução de operações *resets*, embora possuam tempo de execução elevado. A abordagem **Red2.2** apresenta os melhores resultados na redução do tamanho do conjunto e também uma taxa de redução significativa na redução de *resets*. Na Tabela 4.5 são apresentadas as taxas de reduções para o tamanho do conjunto e para o número de operações *resets*, considerando todos os conjuntos reduzidos nesse experimento.



(a) Média do tamanho do conjunto na variação do número de estados.



(b) Média do número de *resets* na variação do número de estados.

Figura 4.6: Variação do número de estados.

Tabela 4.5: Taxas de redução para cada abordagem considerando conjuntos gerados pelo método HSI.

Abordagem	Taxa de Redução do Tamanho do Conjunto	Taxa de Redução do Número de *resets*
Red1.1	19.1%	21.6%
Red1.2	7,0%	61,8%
Red2.2	28.1%	45.4%
Red3.2	4,1%	63,2%

4.3.2 Variação no Número de Entradas

Para esse experimento foram geradas MEFs completas com número de entradas variando entre 2 e 6, com número de estados fixados em 10 e número de saídas fixados em 3, o que resulta em 5 configurações diferentes de MEFs. Para cada configuração foram geradas 30 MEFs, totalizando 150 MEFs ao total para esse experimento. Para cada uma das 150 MEFs, conjuntos HSI foram gerados, que foram submetidos às 4 abordagens de redução descritas no início dessa seção. Dessa forma, 600 reduções foram realizadas.

Na Tabela 4.6 são apresentadas, para cada tamanho de entrada, a média das taxas de redução do tamanho dos conjuntos gerados pelo método HSI. Percebe-se uma tendência de maior redução conforme a quantidade de entradas cresce. De maneira análoga, na Tabela 4.7 são apresentadas as taxas de redução de *resets*, as quais mostram a mesma tendência de redução em relação à redução do tamanho dos conjuntos. As tendências da média do tamanho do conjunto e da média do número de *resets* em relação à variação do número de entradas são apresentadas nos gráficos das Figuras 4.7.a e 4.7.b, respectivamente.

Tabela 4.6: Taxas de redução do tamanho dos conjuntos, considerando variação nas entradas.

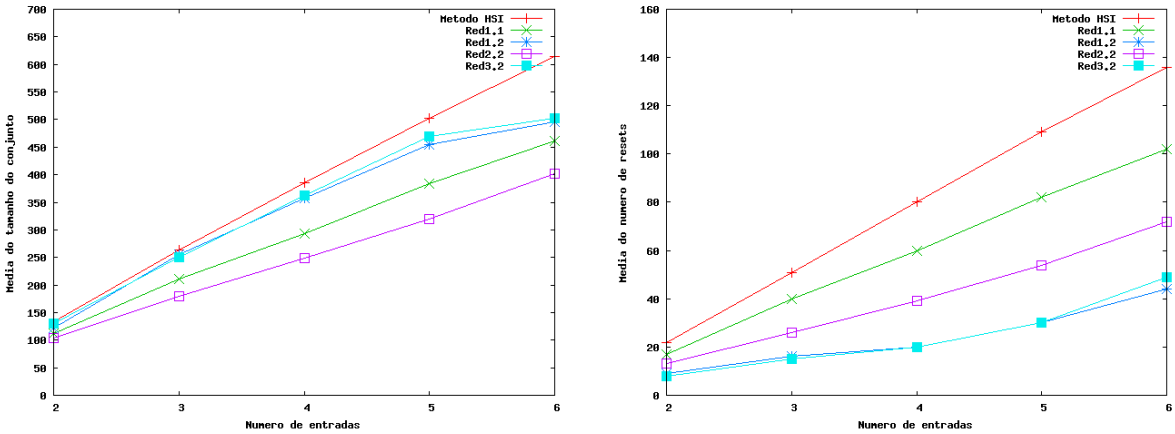
Entradas	Red1.1	Red1.2	Red2.2	Red3.2
2	16,4%	9,1%	22,4%	3,0%
3	20,5%	6,0%	33,1%	8,5%
4	23,9%	7,0%	35,6%	6,3%
5	23,4%	9,3%	36,2%	6,5%
6	24,9%	17,5%	34,6%	18,3%

4.3.3 Variação no Número de Saídas

Para esse experimento foram geradas MEFs completas com número de saídas variando entre 2 e 6, com número de estados fixados em 10 e número de entradas fixados em 3, o

Tabela 4.7: Taxas de redução do número de *resets*, considerando variação nas entradas.

Entradas	Red1.1	Red1.2	Red2.2	Red3.2
2	22,0%	60,2%	40,6%	62,1%
3	22,2%	67,6%	48,9%	73,5%
4	25,6%	75,4%	51,8%	75,1%
5	24,8%	72,2%	50,7%	72,1%
6	25,6%	67,0%	47,0%	63,9%



(a) Média do tamanho do conjunto na variação do número de entradas.

(b) Média do número de *resets* na variação do número de entradas.

Figura 4.7: Variação do número de entradas.

que resulta em 5 configurações diferentes de MEFs. Similar ao experimento descrito na Seção 4.3.2, 600 reduções foram realizadas.

Assim como apresentado na Seção 4.3.2, são apresentadas nas Tabelas 4.8 e 4.9 as taxas de reduções em relação ao tamanho do conjunto e em relação ao número de *resets*, respectivamente, para conjuntos gerados pelo método HSI. Com a variação de saídas, percebe-se que, assim como acontece com a variação do número de entradas, a taxa de redução também aumenta conforme o a quantidade de saídas cresce. As tendências da média do tamanho do conjunto e da média do número de *resets* em relação a variação do número de entradas são apresentadas nos gráficos das Figuras 4.8.a e 4.8.b, respectivamente.

4.4 Comparação com Estratégia de Minimização

O terceiro estudo experimental conduzido foi a comparação das abordagens de redução com a estratégia de minimização proposta em (Neto, 2008; Neto e Simao, 2007, 2008). O processo de redução difere do processo de minimização no fato de que as seqüências geradas ao fim do processo de redução não são subseqüências do conjunto original. Já na

Tabela 4.8: Taxas de redução do tamanho dos conjuntos, considerando variação nas saídas.

Saídas	Red1.1	Red1.2	Red2.2	Red3.2
2	18,2%	14,9%	25,8%	13,0%
3	20,5%	6,0%	33,1%	8,5%
4	18,5%	3,3%	30,8%	1,6%
5	17,1%	9,5%	31,3%	6,5%
6	21,3%	6,4%	34,0%	8,9%

Tabela 4.9: Taxas de redução do número de *resets*, considerando variação nas saídas.

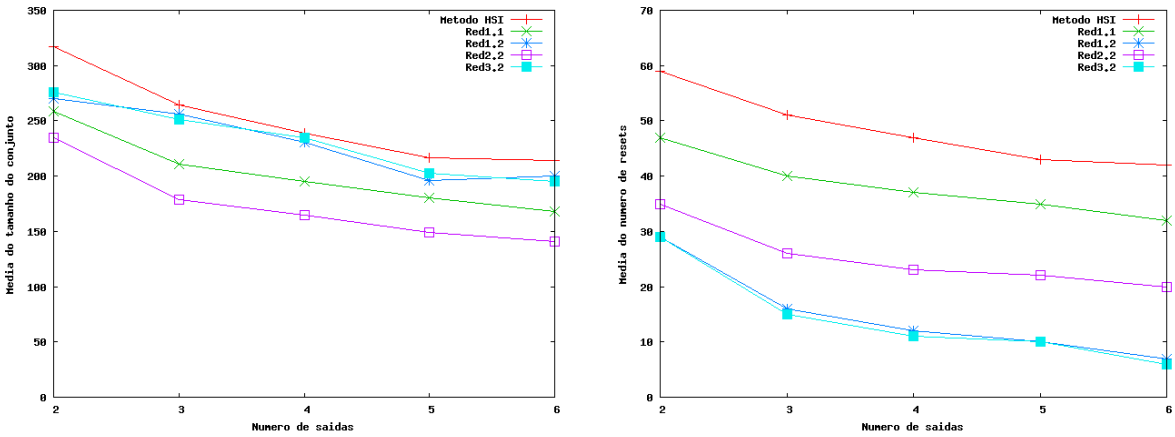
Saídas	Red1.1	Red1.2	Red2.2	Red3.2
2	20,2%	50,0%	39,8%	51,4%
3	22,2%	67,6%	48,9%	73,5%
4	21,3%	75,4%	50,4%	77,6%
5	19,2%	77,5%	49,5%	77,5%
6	24,5%	83,6%	53,1%	85,3%

minimização, o conjunto gerado deve ser composto apenas por subseqüências do conjunto original.

Conjuntos gerados pelo método HSI em MEFs de 10 estados, 3 saídas e 3 entradas foram submetidos ao processo de minimização proposto por Neto (2008). Tem-se, a partir dos resultados, que tanto o tamanho do conjunto quanto o número de *resets*, são menores quando submetidos aos processos de redução. Na Tabela 4.10 são apresentados os dados dos tamanhos do conjunto e o número de *resets* obtidos pelo método HSI, pela estratégia de minimização proposta em (Neto, 2008) e pelas abordagens de redução apresentadas nesta dissertação. Como já discutido na Seção 4.2, as abordagens **Red1.1**, **Red2.1** e **Red3.1** podem ser consideradas equivalentes, sendo assim, apenas a abordagem **Red1.1** foi utilizada nesse estudo experimental.

Tabela 4.10: Comparação com a estratégia de minimização proposta por Neto (2008).

	Tamanho do conjunto	Número de *resets*
Red2.2	177	26
Red1.1	210	40
Minimização (Neto, 2008)	240	47
Red3.2	242	13
Red1.2	248	16
HSI	264	51



(a) Média do tamanho do conjunto na variação do número de saídas.

(b) Média do número de *resets* na variação do número de saídas.

Figura 4.8: Variação do número de saídas.

Nota-se que em relação ao tamanho do conjunto, a minimização gera conjuntos menores que as abordagens **Red1.2** e **Red3.2**, ainda que a diferença seja pequena. Já, considerando o número de *resets*, os resultados das abordagens de redução apontam conjuntos com número de *resets* consideravelmente menores que os gerados pela minimização.

4.5 Análise do Estudo Experimental com MEFs Aleatórias

Observa-se, por meio dos dados obtidos nos estudos experimentais, que as abordagens **Red1.1**, **Red2.1** e **Red3.1** apresentam resultados bem similares tanto na redução do tamanho do conjunto quanto na redução de *resets*. Isso acontece devido ao fato de que essas três abordagens utilizam apenas combinações por sobreposição. Com isso, limita-se muito a quantidade de combinações que podem ser feitas em determinado conjunto de teste, e conseqüentemente os conjuntos gerados por essas três abordagens possuem tamanho e número de seqüências similares.

As abordagens **Red1.2**, **Red2.2** e **Red3.2** utilizam tanto sobreposição quanto seqüências de transferência para a combinação de seqüências. Com isso, a quantidade de combinações possíveis em um conjunto de teste aumenta muito. Como conseqüência, um número menor de seqüências pode ser observado no conjunto final, ou seja, menos operações *resets*. As abordagens **Red1.2** e **Red3.2** combinam as seqüências exaustivamente, enquanto a abordagem **Red2.2** combina aos pares em apenas uma iteração. Dessa forma, como evidenciado no estudo experimental, a redução de *resets* é maior nas duas primeiras. Já a abordagem **Red2.2** apresenta redução de *resets* a uma taxa menor.

Quanto ao tamanho dos conjuntos, nas abordagens **Red1.2** e **Red3.2** as taxas de reduções do tamanho do conjunto não são tão expressivas. Isso acontece devido ao fato que essas duas abordagens combinam as seqüências exaustivamente e, como conseqüência dessa operação, muitas seqüências de transferência são utilizadas, o que leva, em alguns casos, o **passo 2** da abordagem eliminar menos símbolos de entradas do que as inseridas durante a construção do conjunto pelo **passo 1**. A abordagem **Red2.2**, também utiliza seqüências de transferência, porém o número de combinações é limitado. Com isso, os resultados do estudo experimental se mostram bem interessantes para essa abordagem.

Em relação ao tempo de execução, as abordagens **Red1.1**, **Red2.1** e **Red3.1** apresentam um tempo de execução parecidos. Já, as abordagens **Red1.2** e **Red3.2** apresentam tempos mais elevados, devido ao fato do alto número de combinações realizadas. A abordagem **Red2.2** apresenta um bom tempo de redução em relação às reduções obtidas.

Na Tabela 4.11 é apresentado um comparativo entre as abordagens, em que são levados em consideração o tamanho do conjunto, número de *resets* e o tempo de execução. $X \prec Y$ representa que a abordagem X gera resultados menores que Y para o critério da primeira coluna da tabela e $X \simeq Y$ representa que a abordagem X gera resultados equivalentes a Y para o critério da primeira coluna da tabela. Por exemplo, quanto ao tamanho do conjunto, a abordagem **Red2.2** tem como resultado conjuntos com os menores tamanhos entre todas as abordagens. Salienta-se que esse comparativo leva em conta o comportamento médio de cada abordagem, inferidos por meio dos estudos experimentais.

Tabela 4.11: Comparativo entre as abordagens.

Critério	Comparação
Tamanho do conjunto	$Red2.2 \prec Red1.1 \simeq Red2.1 \simeq Red3.1 \prec Red1.2 \prec Red3.2$
Número de *resets*	$Red3.2 \prec Red1.2 \prec Red2.2 \prec Red1.1 \simeq Red2.1 \simeq Red3.1$
Tempo de execução	$Red1.1 \simeq Red2.1 \simeq Red3.1 \prec Red2.2 \prec Red1.2 \prec Red3.2$

Com isso, se o objetivo principal na redução for a redução de *resets* e o tempo de execução não for importante, as abordagens **Red1.2** e **Red3.2** são recomendadas. Se o tempo for importante, as abordagens **Red1.1**, **Red2.1** e **Red3.1** são boas alternativas. Já, a abordagem **Red2.2** executa com um tempo razoável e com boas taxas de redução tanto de *resets* quanto do tamanho do conjunto.

4.6 Estudos de Caso

Com o objetivo de exercitar as abordagens de redução em MEFs não aleatórias e especificações reais, nesta seção são apresentados dois estudos de caso. O primeiro trata-se da utilização de um *benchmark* de especificações baseadas em MEFs. O segundo se trata

de especificações de protocolos, em que uma MEF do projeto Plavis (PLAVIS, 2005) é utilizada, além de uma outra especificação real de protocolo.

4.6.1 Utilização de Benchmark

Nesse estudo de caso, um *benchmarck* de MEFs foi utilizado. Esse *benchmark* compreende um total de 53 MEFs obtidos do *International Workshop on Logic Synthesis (LGSynth) realizado em 1991.* Com um refinamento na seleção feita por (Neto, 2008), 45 MEFs do *benchmark* foram descartadas, de acordo com os critérios abaixo:

- 12 MEFs, pois continham mais de 15 mil transições ou mais de 100 estados, o que inviabiliza a execução das abordagens.
- 19 MEFs, pois não eram minimais.
- 2 MEFs, pois não eram determinísticas.
- 1 MEF, pois não era máquina de *Mealy.*
- 11 MEFs descartadas, pois não foi possível gerar casos de teste para elas.

Com isso, oito MEFs foram utilizadas. Na Tabela 4.12 são apresentadas essas MEFs, com informações do número de estados, entradas e saídas. Todas essas MEFs são minimais, determinísticas, de *Mealy* e completas.

Tabela 4.12: MEFs extraídas do *benchmark.*

MEF	Estados	Entradas	Saídas
bbtas	6	4	4
dk16	27	4	5
dk17	8	4	5
dk27	7	2	3
lion	4	4	2
mc	4	8	8
shiftreg	8	2	3
train4	4	4	2

Para as oito MEFs selecionadas foram gerados conjuntos de teste por meio do método HSI. Os dados das taxas de redução do tamanho do conjunto para as abordagens **Red1.1**, **Red1.2**, **Red2.2** e **Red3.2** são apresentados na Tabela 4.13, enquanto na Tabela 4.14 são apresentadas as taxas de redução de *resets.*

Tabela 4.13: Taxas de redução do tamanho dos conjuntos para as MEFs do *benchmark*.

MEF	Red1.1	Red1.2	Red2.2	Red3.2
bbtas	10,9%	1,9%	7,1%	2,8%
dk16	34,1%	27,9%	32,6%	32,6%
dk17	18,6%	22,1%	23,6%	4,7%
dk27	10,9%	18,8%	3,1%	3,1%
lion	27,9%	25,6%	30,2%	21,7%
mc	1,5%	14,4%	7,6%	11,4%
shiftreg	10,1%	13,0%	15,9%	14,5%
train4	18,5%	18,5%	19,4%	13,0%

Tabela 4.14: Taxas de redução do número de *resets* para as MEFs do *benchmark*.

MEF	Red1.1	Red1.2	Red2.2	Red3.2
bbtas	8,0%	48,0%	44,0%	52,0%
dk16	40,3%	41,5%	37,7%	37,7%
dk17	18,9%	54,7%	35,8%	56,6%
dk27	9,1%	54,5%	27,3%	45,5%
lion	33,3%	48,1%	44,4%	51,9%
mc	3,4%	93,1%	48,3%	96,6%
shiftreg	11,1%	77,8%	44,4%	88,9%
train4	27,3%	40,9%	36,4%	40,9%

Observa-se por meio dos dados que o comportamento das reduções se mostraram semelhantes às MEFs geradas aleatoriamente, com exceção na redução de *resets* para a MEF *dk16*, em que a abordagem **Red1.1** apresentou resultados parecidos em relação às outras três abordagens.

4.6.2 Utilização de Especificações de Protocolos

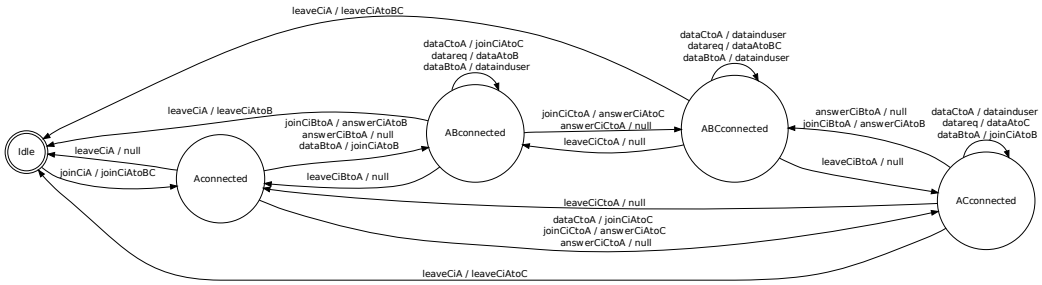
Nesse estudo experimental, as abordagens de redução foram aplicadas em especificações de protocolos reais. O experimento foi construído sobre dois protocolos:

ConfCase: se trata de uma especificação de um protocolo de conferência. Essa especificação foi uma das utilizadas no contexto do projeto Plavis, porém seu desenvolvimento se deu no projeto Côte Resyste[1]. Esse protocolo fornece um serviço no qual os usuários podem participar de uma conferência, trocar mensagens e sair dela. Na Figura 4.9.a é apresentada a especificação do protocolo ConfCase. A MEF é determinística e parcialmente especificada, porém não é minimal. Para permitir a geração dos ca-

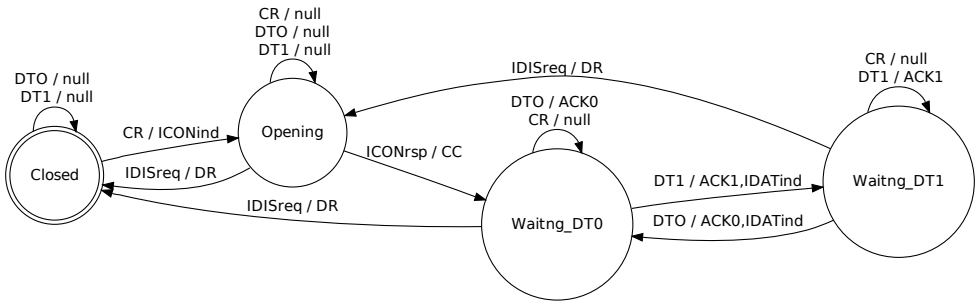
[1]http://fmt.cs.utwente.nl/ConfCase/

sos de teste e sua redução, transições foram adicionadas ao estado inicial da MEF, de forma a torná-la minimal. Originalmente a especificação contém 5 estados, 11 entradas, 13 saídas e 28 transições.

INRES: se trata de um protocolo de comunicação (INitiator-REsponder) (Hogrefe, 1991; Tan et al., 1996) que contém aspectos essenciais dos protocolos de comunicação e mantém a regra de comunicação entre as duas entidades do protocolo: *Initiator* e *Responder*. Na Figura 4.9.b é apresentada a MEF que especifica o comportamento do *Responder* do protocolo INRES. A MEF é determinística, minimal e parcialmente especificada, a qual contém 4 estados, 5 entradas, 7 saídas e 11 transições.



(a) ConfCase



(b) Inres

Figura 4.9: Protocolos especificados em MEFs.

Para as duas especificações foram gerados conjuntos por meio do método HSI e reduzidos pelas abordagens **Red1.1**, **Red1.2**, **Red2.2** e **Red3.2**. Na Tabela 4.15 são apresentadas as taxas de redução do tamanho do conjunto, enquanto na Tabela 4.16 são apresentadas as taxas de redução do número de *resets*.

Deve-se observar que, por se tratarem de duas MEFs parciais, o número de seqüências geradas é geralmente menor e, conseqüentemente, as possibilidades de redução são

Tabela 4.15: Taxas de redução do tamanho dos conjuntos para as especificações de protocolo.

MEF	Red1.1	Red1.2	Red2.2	Red3.2
ConfCase	22,7%	27,6%	28,0%	19,1%
INRES	20,2%	14,9%	21,3%	16,0%

Tabela 4.16: Taxas de redução do número de *resets* para as especificações de protocolo.

MEF	Red1.1	Red1.2	Red2.2	Red3.2
ConfCase	37,7%	96,2%	50,9%	96,2%
INRES	33,3%	61,9%	52,4%	61,9%

menores. Contudo, obteve-se uma redução significativa, tanto em relação ao tamanho do conjunto quanto em relação ao número de *resets*.

4.7 Considerações Finais

Com o objetivo de avaliar as abordagens de redução propostas e inferir sobre as características de cada uma, alguns estudos experimentais foram conduzidos, os quais foram apresentados ao longo deste capítulo.

Os experimentos foram planejados para serem os mais diversificados possíveis. O primeiro buscou identificar o comportamento da redução em conjuntos gerados pelos métodos clássicos. O segundo teve como principal objetivo analisar a influência da variação de parâmetros na MEF durante o processo de redução. Buscou-se no terceiro experimento a comparação da redução com uma estratégia de minimização. Por último, buscou-se exercitar as abordagens de redução em MEFs não aleatórias, sendo conduzido um estudo de caso com um *benchmark* de MEFs e duas especificações de protocolos. Os resultados obtidos no estudo de caso confirmaram os resultados da avaliação experimental.

Foi possível identificar as características de cada abordagem, o que resulta num conjunto de abordagens bem interessantes, e que podem ser utilizadas para diversos fins, dependendo do tipo de redução desejada. De uma maneira geral, a Tabela 4.11 apresenta um comparativo entre as abordagens que sintetiza suas características e diferenças.

No capítulo seguinte são apresentadas as conclusões deste trabalho de mestrado, com as contribuições, dificuldades, limitações e direções para trabalhos futuros.

Capítulo

5

Conclusões

A redução de um conjunto de seqüências de teste é relevante no sentido de que os métodos de geração produzem, em geral, conjuntos muito grandes e com um elevado número de seqüências. Dessa forma, é desejável que os conjuntos gerados por esses métodos sejam reduzidos, de forma a agilizar a etapa de teste e torná-la menos onerosa. No entanto, para que a qualidade do teste não seja afetada, é necessário que o conjunto gerado apresente a mesma efetividade em revelar defeitos em relação ao conjunto de teste original.

Neste trabalho de mestrado, foi investigado o problema da redução de conjunto de seqüências de teste. Além da redução do tamanho do conjunto, busca-se a redução do número de operações *resets*, uma vez que essa operação é, geralmente, custosa para a aplicação. Para isso, foi proposta uma estratégia de redução baseada nas condições de suficiência definidas por Simao e Petrenko (2010). Foram elaboradas seis abordagens de redução baseadas nessa estratégia, as quais mostraram ganhos significativos e propriedades interessantes que foram comprovadas pela condução de estudos experimentais. A seguir são apresentadas as principais contribuições deste trabalho de mestrado, assim como as dificuldades, limitações e perspectivas de trabalhos futuros.

5.1 Contribuições

A principal contribuição deste trabalho de mestrado foi a definição da estratégia de redução e a elaboração de seis abordagens baseadas nessa estratégia. Tem-se, com isso, um conjunto de abordagens para redução com diferentes características, podendo ser escolhidas de acordo com o objetivo principal a ser atingido na redução. Por exemplo, se o objetivo principal é a redução de *resets*, independente do tempo de execução, poderiam ser utilizadas as abordagens **Red1.2** ou **Red3.2**. Caso o tempo de execução seja importante, pode-se utilizar as abordagens **Red1.1**, **Red2.1**, **Red3.1** e **Red2.2**, sendo a última com tempo de execução um pouco maior, porém com melhores resultados tanto na redução do tamanho do conjunto quanto na redução de *resets*.

São apresentados estudos experimentais em que a redução é aplicada em conjuntos gerados pelos métodos W, Wp, HSI e H. Além disso, foi apresentado um estudo sobre o comportamento das abordagens de redução quando se varia os parâmetros das MEFs. Uma comparação com uma estratégia de minimização também foi conduzida. Por fim, estudos de caso com especificações em protocolos foram realizados, inclusive com uma MEF do projeto Plavis, de forma a verificar o comportamento das reduções com MEFs não aleatórias.

Além da ferramenta **RedTest**, que compreende as abordagens de redução, também foram implementadas as ferramentas **ChesCon** e **ChesConViz**, sendo que esta última pode ser um interessante instrumento para o estudo das condições de suficiência e para auxiliar trabalhos futuros de investigação nessa área.

5.2 Dificuldades, Limitações e Trabalhos Futuros

A redução de conjuntos de seqüências de teste gerados pelo métodos clássicos não é uma tarefa trivial. Por exemplo, os conjuntos gerados pelo método H satisfazem de maneira implícita as condições de suficiência utilizadas neste trabalho (Simao e Petrenko, 2010). Porém, a forma com que o método H gera o conjunto não permite melhorias que possam reduzir significativamente o conjunto gerado, mesmo considerando as condições de suficiência de Simao e Petrenko (2010) explicitamente. Com isso, tem-se a redução dos conjuntos como uma atividade custosa e não trivial. A utilização das condições de suficiência para identificação de um conjunto inicial (*state cover*) ótimo para o método H é um caminho em que pode haver evoluções, no sentido de reduzir o conjunto gerado.

Considerando as abordagens de redução propostas neste trabalho, o problema da combinação aleatória de seqüências, realizada no **Passo 1**, é um ponto em que pode haver evoluções. A identificação de propriedades nas seqüências pode direcionar uma melhor

escolha delas, otimizando o processo final de redução. Pode-se investigar mecanismos para identificar pares de seqüências que, se combinados, manteriam obrigatoriamente o conjunto completo. A ordem em que as seqüências são combinadas também é um fator que influencia o resultado final da redução. A investigação de um direcionamento na escolha das seqüências pode ser interessante, uma vez que se pode identificar as seqüências que, se escolhidas, otimizariam o processo de redução. Quanto ao **Passo 2**, a escolha de seqüências aleatórias para obtenção de prefixos menores também é outro ponto em que pode haver melhorias, como a identificação das seqüências que, se escolhidas, otimizariam o resultado final da redução.

Quanto à estratégia em geral, pode-se investigar o comportamento da redução no caso em que os **Passos 1** e **2** são executados juntamente, ou seja, faz-se apenas um número estipulado de combinações no **Passo 1** e se executa o **Passo 2**, intercalando-os, até não ser mais possível a redução do conjunto.

A partir dos resultados obtidos na redução, pode-se investigar mecanismos de geração de conjuntos já reduzidos. A geração de conjuntos com menos seqüências é um caminho em que pode haver evolução, uma vez que a maioria dos métodos existentes gera um número muito elevado de seqüências. Isso acontece devido ao fato que os métodos existentes utilizam, em geral, o mesmo conjunto inicial para gerar o conjunto de seqüências de teste. Um novo conjunto inicial poderia ser investigado, de forma a avaliar o quanto esse conjunto inicial influencia o conjunto final.

Referências

Andrews, A.; Offutt, J.; Alexander, R. Testing web applications by modeling with fsms. *Software Systems and Modeling*, v. 4, n. 3, p. 326–345, 2005.

Anido, R. O.; Cavalli, A. *Guaranteeing full fault coverage for UIO-based methods.* Relatório Técnico, Department of Computer Science, University of Campinas, 1995.

Beizer, B. *Software testing techniques (2nd ed.).* New York, NY, USA: Van Nostrand Reinhold Co., 1990.

Bertolino, A. Software testing research: Achievements, challenges, dreams. In: *FOSE '07: 2007 Future of Software Engineering*, Washington, DC, USA: IEEE Computer Society, 2007, p. 85–103.

Binder, R. V. *Testing object-oriented systems: models, patterns, and tools.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

Boberg, J. Early fault detection with model-based testing. In: *ERLANG '08: Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, New York, NY, USA: ACM, 2008, p. 9–20.

Bochmann, G. V.; Petrenko, A. 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*, New York, NY, USA: ACM, 1994, p. 109–124.

Bourhfir, C.; Dssouli, R.; Aboulhamid, E. M. *Automatic test generation for efsm-based systems.* Relatório Técnico, 1996.

Chen, J.; Hierons, R. M.; Ural, H.; Yenigun, H. Eliminating redundant tests in a checking sequence. In: *TestCom 2005*, n. 3502 in Lecture Notes in Computer Science, 2005, p. 146–158 (*Lecture Notes in Computer Science*,).

Chow, T. S. Testing software design modeled by finite-state-machines. *IEEE Transactions on Software Engineering*, v. 4, n. 3, p. 178–186, 1978.

Dalal, S. R.; Jain, A.; Karunanithi, N.; Leaton, J. M.; Lott, C. M.; Patton, G. C.; Horowitz, B. M. Model-based testing in practice. In: *ICSE '99: Proceedings of the 21st international conference on Software engineering*, New York, NY, USA: ACM, 1999, p. 285–294.

Davis, A. M. A comparison of techniques for the specification of external system behavior. *Communications of the ACM*, v. 31, n. 9, 1988.

DeMillo, R. A. Mutation analysis as a tool for software quality assurance. 1980.

Dorofeeva, R.; El-Fakih, K.; Yevtushenko, N. An improved conformance testing method. In: *FORTE*, 2005a, p. 204–218.

Dorofeeva, R.; Yevtushenko, N.; El-Fakih, K.; Cavalli, A. R. Experimental evaluation of fsm-based testing methods. In: *SEFM '05: Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods*, Washington, DC, USA: IEEE Computer Society, 2005b, p. 23–32.

Fabbri, S. C. P. F.; Maldonado, J. C. Teste de software. In: *Qualidade de Software: Teoria e Prática*, cáp. 4, São Paulo, Brasil: Prentice-Hall, p. 73–84, 2001.

Fujiwara, S.; Bochman, G. V.; Khendek, F.; Amalou, M.; Ghedamsi, A. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, v. 17, n. 6, p. 591–603, 1991.

Garey, M. R.; Johnson, D. S. *Computers and intractability: A guide to the theory of np-completeness.* W. H. Freeman, 1979.

Gill, A. *Introduction to the theory of finite-state machines.* New York: McGraw-Hill, 1962.

Gonenc, G. A method for design of fault detection experiments. *IEEE Transactions on Computers*, v. 19, n. 6, p. 551–558, 1970.

Grunert, T.; Irnich, S.; Zimmermann, H.-J.; Schneider, M.; Wulfhorst, B. Finding all k-cliques in k-partite graphs, an application in textile engineering. *Computers & Operations Research*, v. 29, p. 13–31, 2002.

Harel, D. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, v. 8, n. 3, p. 231–274, 1987.

Harrold, M. J. Testing: A roadmap. In: *In The Future of Software Engineering*, ACM Press, 2000, p. 61–72.

Hennie, F. C. Fault detecting experiments for sequential circuits, p. 95–110. 1964.

Hierons, R. M. Using a minimal number of resets when testing from a finite state machine. *Inf. Process. Lett.*, v. 90, n. 6, p. 287–292, 2004.

Hierons, R. M.; Ural, H. Reduced length checking sequences. *IEEE Transactions on Computers*, v. 51, n. 9, p. 1111–1117, 2002.

Hierons, R. M.; Ural, H. Optimizing the length of checking sequences. *IEEE Transactions on Computers*, v. 55, n. 5, p. 618–629, 2006.

Hogrefe, D. *Osi formal specification case study: the inres protocol and service.* Relatório Técnico IAM-91-012, University of Bern, 1991.

Hong, H. S.; Kwon, Y. R.; Cha, S. D. Testing of object-oriented programs based on finite state machines. In: *2nd Asia-Pacific Software Engineering Conference (APSEC'95)*, Brisbane, Queensland, Australia: IEEE Computer Society, 1995, p. 234–241.

IEEE IEEE standard glossary of software engineering terminology. Standard 610.12, Institute of Electric and Electronic Engineers, 1990.

ITU CCITT specification and description language. Recommendation Z.100 (03/03), International Telecommunication Union, 1993.

ITU Revised ITU-T specification and description language. Recommendation revision Z.100, International Telecommunication Union, 2002.

Lee, D.; Yannakakis, M. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, v. 43, n. 3, p. 306–320, 1994.

Luo, G.; Petrenko, R.; Bochmann, G. V. Selecting test sequences for partially-specified nondeterministic finite state machines. In: *In IFIP 7th International Workshop on Protocol Test Systems*, 1994, p. 91–106.

Maldonado, J. C. *Critério potenciais usos: Uma contribuição ao teste estrutural de software.* Tese de Doutoramento, DCA/FEE/UNICAMP, Campinas, 1991.

Maldonado, J. C.; Barbosa, E. F.; Vincenzi, A. M. R.; Delamaro, M. E.; Souza, S. R. S.; Jino, M. *Introdução ao teste de software.* Relatório Técnico 65, ICMC/USP, São Carlos, SP, notas Didáticas do ICMC, Série Computação, 2004.

McGill, R.; Tukey, J. W.; Larsen, W. A. Variations of box plots. *The American Statistician*, v. 32, n. 1, p. 12–16, 1978.

Moore, E. F. Gedanken-experiments on sequential machines. , n. 34, p. 129–153, 1956.

Myers, G. J. *The art of software testing.* New York: Wiley, 1979.

Neto, L. F. M. *Minimização de conjuntos de casos de teste para máquinas de estados finitos.* Dissertação de Mestrado, ICMC/USP, São Carlos/SP, 2008.

Neto, L. F. M.; Simao, A. S. Minimização de conjuntos de casos de teste por meio de condições de suficiência. In: *1st Brazilian Workshop on Systematic and Automated Software Testing*, João Pessoa, PB, 2007, p. 55–62.

Neto, L. F. M.; Simao, A. S. Test suite minimization based on fsm completeness sufficient conditions. In: *Latin-American Test Workshop*, Puebla, Mexico, 2008.

Peterson, J. L. Petri nets. *ACM Comput. Surv.*, v. 9, n. 3, p. 223–252, 1977.

Petrenko, A.; von Bochmann, G.; Yao, M. Y. On fault coverage of tests for finite state specifications. *Computer Networks and ISDN Systems*, v. 29, n. 1, p. 81–106, 1996.

Petrenko, A.; Boroday, S.; Groz, R. Confirming configurations in efsm testing. *IEEE Transactions on Software Engineering*, v. 30, n. 1, p. 29–42, 2004.

Petrenko, A.; Yevtushenko, N. Testing from partial deterministic fsm specifications. *IEEE Transactions on Computers*, v. 54, n. 9, p. 1154–1165, 2005.

Petrenko, A.; Yevtushenko, N.; Lebedev, A.; Das, A. Nondeterministic state machines in protocol conformance testing. In: *Protocol Test Systems*, 1993, p. 363–378.

Pilone, D. *Uml 2.0 pocket reference (pocket reference (o'reilly)).* O'Reilly Media, Inc., 2006.

PLAVIS Plavis - platform for software validation & integration on space systems. Online em: `http://www.labes.icmc.usp.br/plavis/index.html`, Último acesso: 17/12/2008, 2005.

Pressman, R. S. *Engenharia de software.* Makron Books do Brasil, 2005.

Ribeiro, P. H.; Cutigi, J. F.; Simao, A. S. Geração de seqüências de verificação baseado em algoritmos genéticos. In: *3st Brazilian Workshop on Systematic and Automated Software Testing*, Gramado, RS, 2009, p. 61–70.

Sabnani, K. K.; Dahbura, A. Protocol test generation procedure. *Computer Networks and ISDN Systems*, v. 15, n. 4, p. 285–297, 1988.

Sidhu, D. P.; kau Leung, T. Formal methods for protocol testing: A detailed study. *IEEE Transactions on Software Engineering*, v. 15, n. 4, p. 413–426, 1989.

Simao, A.; Petrenko, A. Checking completeness of tests for finite state machines. *IEEE Transactions on Computers*, v. 99, n. PrePrints, 2010.

Simao, A. S. Teste baseado em modelos. In: *Introdução ao Teste de Software*, cáp. 3, Rio de Janeiro, Brasil: Campus, p. 27–45, 2007.

Simao, A. S.; Petrenko, A. Generating checking sequences for partial reduced finite state machines. In: *TestCom/FATES: Proceedings of the 20th IFIP TC 6/WG 6.1 international conference on Testing of Software and Communicating Systems*, Berlin, Heidelberg: Springer-Verlag, 2008, p. 153–168.

Simao, A. S.; Petrenko, A.; Maldonado, J. C. Experimental evaluation of coverage criteria for fsm-based testing. In: *Simpósio Brasileiro de Engenharia de Software*, João Pessoa, PB, 2007.

Simao, A. S.; Petrenko, A.; Yevtushenko, N. Generating reduced tests for fsms with extra states. In: *TestCom/FATES*, 2009, p. 129–145.

Sinha, A.; Smidts, C. Hottest: A model-based test design technique for enhanced testing of domain-specific applications. *ACM Transactions on Software Engineering and Methodology*, v. 15, n. 3, p. 242–278, 2006.

Tan, Q. M.; Petrenko, A.; Bochmann, G. V. A test generation tool for specifications in the form of state machines. In: *in Proceedings of the International Communications Conference*, 1996, p. 225–229.

Turner, K. J. *Using formal description techniques: An introduction to estelle, lotos, and sdl.* New York, NY, USA: John Wiley & Sons, Inc., 1993.

Ural, H.; Wu, X.; Zhang, F. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, v. 46, n. 1, p. 93–99, 1997.

Ural, H.; Zhang, F. Reducing the lengths of checking sequences by overlapping. *Lecture Notes on Computer Science*, , n. 3964, p. 274–288, 2006.

Utting, M.; Legeard, B. *Practical model-based testing: A tools approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

Weyuker, E. J. Using failure cost information for testing and reliability assessment. *ACM Transactions on Software Engineering and Methodology*, v. 5, n. 2, p. 87–98, 1996.

Yalcin, M. C.; Yenigun, H. Using distinguishing and uio sequences together in a checking sequence. In: *TestCom 2006*, n. 3964 in Lecture Notes in Computer Science, 2006, p. 274–288 (*Lecture Notes in Computer Science*,).

Yao, M.; Petrenko, A.; Bochmann, G. v. Conformance testing of protocol machines without reset. In: *Proceedings of the IFIP TC6/WG6.1 Thirteenth International Symposium on Protocol Specification, Testing and Verification XIII*, Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1993, p. 241–256.

Yao, M. Y.; Petrenko, A.; von Bochmann, G. Fault coverage analysis in respect to an fsm specification. In: *IEEE INFOCOM94*, Toronto, Canadá, 1994, p. 768–775.